Medieval climate change and settlement in Iceland

Andrew Casely

Thesis for the degree of Doctor of Philosophy

September 2005

Institute of Geography, University of Edinburgh

Declaration

I declare that this thesis is entirely my own work, including text, maps, photos and figures, except where explicitly acknowledged in the text.

Signed _

Andrew Casely, 30th September 2005

Abstract

A key issue in studies of Norse settlement in Iceland is the role that climate has played in shaping the history of the island. The thesis has two main objectives: (1) to constrain the timing and nature of any significant variations in climate during the Medieval period, and around the time of Norse settlement in Iceland (*c.*500-1500AD). (2) to evaluate likely impacts post-settlement of such changes upon those living in Iceland. To do this, the project uses a multidisciplinary approach, combining new empirical data, existing empirical and documentary data, and a new climate model for Iceland.

Medieval glacier retreat was examined using lacustrine sediment sequences collected from two localities, that from Skeiðsvatn in Tröllaskagi being notable as of being at least 1350 years in length. At Skeiðsvatn, lacustrine evidence indicates the onset of Late Holocene glaciation around A.D. 650, and that glaciation of the catchment has continued uninterrupted to the present day. This constrains the warmth of the 'Medieval Warm Period', while terminal moraines constrain both 'Little Ice Age' and earlier Neoglacial advances. This is the first such lacustrine record of glacier retreat from Iceland.

Two models are presented in the thesis, the most important being an original mass-balance and vegetation cover model. The first modelling approach used is to apply the three-dimensional ice sheet model GLIMMER to test the sensitivity of selected small glaciers to growth and disappearance. The second modelling approach was the construction of a new model of climate and vegetation for Iceland.

The Iceland climate model is Java-based, and includes well over 12,000 lines of original code. It shows that it is possible to model monthly and annual vegetation limits, snowlines and snowcover based on inputs of primarily topography, temperature and precipitation, and constrain the modelling using glacier mass balance. This model has been used to evaluate spatial and temporal environmental responses to changes in temperature and precipitation of known magnitude, and to test the responses to long timeseries of temperature data.

Integrated mass balance and ecological modelling combined with geomorphic data can be used to assess settlement changes in southern Iceland. Soil erosion in Thórsmörk was not directly driven by climatic change, and the degradation has a land management origin. The solution, as shown in the preservation of woodlands in Thórsmörk, was a land management decision, and not deterministically driven by a drop in temperature. Birch woodland in Thórsmörk is at risk of removal during sustained colder spells, but the climatic impact on soil erosion is limited.

Modelling of the Mývatn region indicates increased vulnerability of the landscape to environmental changes on the order of ±1°C, which are likely to trigger large changes in tree birch cover or good quality grazing land area. Growing season length is shortened by *c.*25%, and late-lying snowcover is an additional challenge in colder years. Response to warming is one of substantially reduced snowcover at lower elevations, and an inland shift of vegetation boundaries.

Aeolian sediment sequences from Geithellnadalur, in combination with modelling data gives further insight into the relative importance of climate and human land management practices, and this can be used to identify threshold events. Geithellnadalur shows that environmental change in Iceland is not always directly related to climate, but that climate may have an indirect influence on landscape changes. Within the sediment accumulation record, evidence of both the impact of settlement and the impact of the Little Ice Age are visible.

The new modelling approach, combined with the gathered empirical data, has provided key insights into the many different ways in which climate and environment interact over a varied topography, with consequently diverse effects upon settlement.

Acknowledgements

I'd like to acknowledge some people and organisations without which I could not have completed this thesis.
ESRC-NERC Interdisciplinary Studentship Award no. R42200134010 funded this thesis. This was augmented by the Institute of Geography, University of Edinburgh for additional fieldwork and vehicle support.
The Leverhulme Trust funded three rangefinder radiocarbon dates.
NABO / NSF Polar programmes for vehicle support. NABO for visit to Archaeology Field School at Mývatn.
The NSF for conference support to the 2004 Arctic Workshop.
Andy Dugmore and Nick Hulton at the University of Edinburgh – PhD supervision, support, ideas and inspiration (and to Martin, Madeleine, David and Sarah for their timely arrivals into this world to keep their fathers occupied!)
Nick Hulton, Kate Smith, Jez Everest, Chris Fogwill, Stewart Jamieson, Ian Lawson, Iestyn Barr, Rannveig Ólafsdóttir, Lindsay Sugden and especially Teresa Howard for fieldwork support in sunny Iceland.
Rannveig Ólafsdóttir for accommodation support and discussions at the Háskoli Íslands University Centre in Höfn.
Sveinbjörn Steinþórsson, Ryan Metters, Skúli and Thórkel for their help (and inspired corer building by Sveinbjörn) on the winter coring trip to southeast Iceland.
Many thanks to Tom McGovern and David Sugden for taking the time to examine this thesis and provide helpful comments.
And to Mum and Dad who supported me during my final year and put up with my nine years in higher education!

Table of Contents

Chapter I: Aim & Research Questions 1

1 Project Aim 1

2 Settlement Preconceptions 3

3 Climate in the North Atlantic 6

4 Medieval Review – was there a 'Medieval Warm Period?' 10

5 Conclusions to Background sections 32

6 Principal Research Questions 34

Figures and Tables – Chapter I 36

Chapter II: Approach 55

1 Overall project approach 55

2 Datasets 57

3 Empirical data collection: Catchment geomorphology and environmental change 60

4 Modelling approach 66

5 Field site selection 71

6 Data collection methods 77

Figures and Tables – Chapter II 89

Chapter III: Datasets 103

1 Introduction 103

2 Northern Iceland 104

3 Southeastern Iceland 117

Figures and Tables – Chapter III 125

Chapter IV: Modelling 163

1 Introduction 163

2 Numerical modelling of mass balance and vegetation in Iceland 164

3 Mass balance modelling – results 175

4 Catchment-scale glacier modelling – application of GLIMMER 177

Figures and Tables – Chapter IV 179

Chapter V: Discussion 197

1 Introduction 197

2 Late Holocene climate 197

3 Southern Iceland – Climate and Settlement in Thórsmörk 199

4 Northern Iceland – Climate and Settlement around Mývatn 201

5 Eastern Iceland – Climate and Settlement in Geithellnadalur 203

Figures and Tables – Chapter V 205

Chapter VI: Conclusions 227

References 230

Appendices 244

Chapter I: Aim & Research Questions

1 Project Aim
2 Settlement Preconceptions
3 Climate in the North Atlantic
4 Medieval Review – was there a 'Medieval Warm Period?'
5 Conclusions to Background sections
6 Principal Research Questions

I.1: Project Aim

The research aim is to determine whether the Norse colonisation of Iceland took place under notably benign climates (warmer and drier than today) and whether early phases of farm abandonment took place when local conditions became colder or wetter.

Norse expansion and colonisation of the North Atlantic islands took place during a pivotal period of European history, during which many of the present nation states of Western Europe were becoming established. It is associated with the period of Viking raids and settlement throughout northwestern Europe and the North Atlantic as far west as Greenland and Newfoundland (Jones, 1984; Ólafsson, 2000, Figure 1.1). Prior to Norse settlement around 870AD, Iceland was most likely uninhabited by people, a pristine environment. The impact of settlement on this landscape provides a unique perspective on issues of anthropogenic land use and exploitation of an untouched natural environment at the margin of the temperate climatic zone.

In the present context of anthropogenic climate change, there are concerns regarding the scale and scope of anthropogenic vs natural climate change, and the resulting impacts through land degradation. The evaluation of the impacts of climate change during the most recent period when climate was comparable to that of today can provide valuable insight into both the impact of climate as well as the impact of people upon the environment.

The value of Iceland

Settlement in Iceland appears to have had a tremendous impact on the local ecosystem, with deforestation of large parts of the island and large-scale soil erosion

both significant problems in Iceland today (e.g. Arnalds *et al.*, 1987; Dugmore and Erskine, 1994; 1997; Ólafsdóttir and Júlíusson, 2000). The spatial extent of both of these problems before, during and following Norse settlement of the island, and the extent to which natural changes have played a role is still unclear. The environmental conditions within which settlement of Iceland took place around 870AD, and their consequences, are less well understood.

Iceland's location within the North Atlantic combined with its relatively high latitude allows the investigation of these conditions. Iceland lies close to the atmospheric and oceanic polar fronts – the boundaries between Arctic air masses and water sourced from the Arctic Ocean to the northeast, and temperate air and water sourced from the central Atlantic and Gulf Stream to the southwest. With a maritime climate, there is high precipitation all year round and significant snowfall during the winter months. Small shifts in the atmospheric and oceanic boundaries have a relatively large impact on the Icelandic climate, increasing the sensitivity of the Icelandic environment to change. The topography includes a number of mountainous regions that extend above 1000m which, combined with the precipitation, allows for the growth of glaciers in a variety of topographic settings (Björnsson, 1979), including several large ice caps. At the boundary of the Arctic, the environment ranges from a boreal classification in the south, to low Arctic in the north and the interior, and this is reflected in the spatial pattern of soil and vegetation and the duration of snow cover.

This physical setting provides the basis for this project, as changes in climatic conditions are reflected in variations in glacier extent as well as variations in soil and vegetation that can be successfully investigated. Climate and associated changes is the central axis around which several key themes can be discussed, relating to both the physical environment in Iceland, and to the consequences of that physical environment for Medieval Iceland and Icelanders:

Key Research Themes:

i) Medieval climate and settlement history of Iceland.

ii) Medieval climate and environmental degradation in Iceland.

iii) Medieval climate and farm abandonment in Iceland.

iv) North Atlantic climate changes during the 'Medieval Warm Period'

I.2: Settlement Patterns and Preconceptions

The coastal and lowlands of Iceland were quickly settled during the late 9th and early 10th Century during the *Landnám* period (Figure 1.2). As described by McGovern *et al.* (2005) there is a "traditional narrative" concerning the nature of Icelandic settlement, subsequent impact upon the landscape, and responses to those impacts. Part of this view is a particularly environmentally deterministic approach, where the influence of harsher climatic regimes has a direct effect on Norse settlement. This was envisaged both through their approach to the initial settlement, and the import of land and livestock management practices from Scandinavia that were not suited to Iceland, as well as the deterministic impact of climatic change as the principal key to early farm abandonment. Both of these points have also been highlighted in discussions regarding the demise of the Norse settlement of Greenland (Barlow *et al.*, 1997). Within this was the concept of a rapid deforestation of Iceland in the early Settlement Period (Figure 1.3) to levels comparable with those of today (Figure 1.4), and the extension of large areas of degraded land (Figure 1.5; 1.6) The 'traditional narrative' has been the accepted view of events in the Icelandic environment following settlement (for example McGovern *et al.*, 1988; Dugmore and Buckland, 1991; Dugmore and Erskine, 1994; Ólafsdóttir and Guðmundsson, 2002), and is summarised below from McGovern *et al.* (2005):

Settlement

1: Settlement of Iceland around A.D.870 (e.g. Grönvold *et al.*, 1995; Vésteinsson, 2000).

2: First settlers claim largest tracts of land and become the most powerful chieftains.

3: Expansion of settlement inland from initial coastal landtaking – leading to a risk of 'overoptimistic' settlement.

4: Later / less wealthy / less successful settlers take land further inland.

Post-Settlement

5: Human impacts on the 'natural capital' of the land is immediate and severe.

6: This includes rapid deforestation – trees almost completely cleared (Figures 1.3;1.4).

7: Dependency on imported goods from Europe, including iron.

8: Poor adaptation of a NW European economy and lifestyle to the Icelandic environment.

9: "Tragedy of the commons" (Feeny *et al.*, 1990), where overoptimistic settlement and use of highland common grazing land leads to erosion that subsequently impacts on infields and farms.

10: Climatic cooling following the Medieval Warm Period exacerbates these impacts and the low natural capital and consequently has a disproportionate impact.

11: Land degradation enhanced by competition among the elite, and tenants forced into unsustainable land use, with little in the way of integrated environmental management strategy.

This view of the patterns of settlement and the subsequent land degradation is now being challenged through an integrated research programme, with principal sites in the Mývanssveit and Eyjafjallsveit regions in northern and southern Iceland respectively. The patterns of settlement are now thought to be more complex, with a rapid colonisation of inland sites rather than a progression from the coast. In an alternative model of settlement, Vesteinsson (2000) suggests that upland pasture areas towards the treeline would have been favourable early settlement sites, as they did not require extensive clearance of the land. In the Mývatn area, the attraction of such a site is aided by the rich resource of birds and fish provided by the lake.

Deforestation is not now seen as being quite as universally immediate and rapid as was previously assumed. The assumption of early deforestation and subsequent dependency on imported iron has been challenged by the excavation of a number of charcoal pits in both northern and southern Iceland (Dugmore *et al.*, in press, Figure 1.7), which show that charcoal production (required for iron smelting) extended well beyond the initial Landnám period, suggesting both woodland resource management as well as local smelting of iron for tool manufacture. Palynological evidence in lacustrine sediments from Helluvaðstjörn near lake Mývatn, supported by charcoal finds in nearby middens, indicates that the most significant decline in birch occurred around A.D. 1300 (Lawson *et al.*, in press; Figure 1.8), which is in contrast to the rapid decline in southwest Iceland reported by Hallsdóttir (1987). Simpson *et al.* (2003) show that, at least at higher status farms, fuel use was succesfully managed to a significant extent in the post-Landnám period. Also at Lake Mývatn, there is clear

evidence that the large bird population has been successfully managed for over a thousand years without critically depleting it as a resource (McGovern *et al.*, in press). Such evidence points towards some successful land management practices rather than an immediate and universal drawdown of the natural capital.

If woodlands are managed, which areas of land are kept forested? If it is marginal land (close to the ecological limit for the trees) that remains covered by woodland, then this wood may be the most vulnerable to a detrimental change in climate. There is some evidence that this is the case in Thórsmörk.

In relation to upland grazing, the model of a landscape scarred by overgrazing and poor land management may be challenged by one in which the impact of climatic variability at ecological margins has the greater impact. The 'tragedy of the commons' may not explain erosion of the uplands. In such a model, it is the inability to adapt to a changing environment that has a greater effect, for example the impact of weather on the critical date when livestock is brought in from the upland grazing to be fed on fodder for the winter (McGovern *et al.*, 2005).

I.3: Climate in a cool and stormy environment:

What controls the climate of Iceland? As introduced above, one of the aspects that makes Iceland an inviting location for research is its physical setting in the North Atlantic, and the crucial role of atmospheric circulation in determining Iceland's environment. The climate is characterised by the regular passage of low pressure systems as they cross the North Atlantic between North America and northwestern Europe (Figure 1.9). At present, these storms frequently track across Iceland, giving a prevailing southerly wind direction, of ten strong winds, and high precipitation in the southern part of the country. Warm ocean currents originating to the southwest give the island a temperate climate despite being north of 60°N latitude. The waters surrounding Iceland are also among the richest fishing grounds in the world due to the high plankton productivity, giving the island a valuable natural resource.

Being at the boundary between Arctic and temperate air and water masses provides the key to Iceland's sensitivity to climate. Relatively minor shifts in these boundaries are significant at the scale of Iceland as they can result in a sharp contrast in local weather conditions. Even within the North Atlantic, which is recognised for its sensitivity to climate changes, there are few locations with a comparable situation in relation to the boundary between Arctic and temperate environments.

One of the significant factors resulting from the strong pressure and temperature gradients across the North Atlantic is the very strong spatial variation in patterns of climate. Two stations located on either side of a weather system will experience widely differing weather: if this is sustained over a season or the year, the climate in these regions will be different, despite their proximity to one another.

Atmospheric weather systems

Weather, as opposed to climate is crucial to the understanding many patterns of the Icelandic environment, and ultimately the response of elements of that environment (including glaciers) to change. The North Atlantic is one of the principal regions of cyclogenesis in the Northern Hemisphere, due to the interaction of polar air masses moving southwards and subtropical air masses moving northwards and meeting at the polar front. On this front, depressions form off the coast of eastern Canada all year round, and track eastwards and northeastwards towards Scandinavia, crossing Iceland on the way (Figure 1). This process is strongest during the winter,

resulting in persistently low pressure in the region of Iceland (the Icelandic Low pressure system) and more severe storms. The depressions are usually bounded on their southern side by the Azores High (pressure system).

It is the average track of these depressions throughout a season or the year that determined whether Iceland spends more time influenced by polar airmasses or by temperate airmasses. A slight northerly shift in the average position of the polar front takes the storm tracks north of the island, bathing Iceland in warmer air located to the south of depression centres. A southerly shift takes the depressions south of Iceland and results in Arctic air masses sourced to the north and northeast controlling the weather. The dramatic difference in the temperatures of the source regions for the air masses gives Iceland both great variability in weather, and sensitivity to climate change.

The persistence of an 'Icelandic Low' and an 'Azores High' in the Atlantic has been used to examine weather patterns, through the North Atlantic Oscillation index (NAO). This simple index is constructed by comparing the average air pressure at the centre of these two quasi-stable atmospheric systems. The winter NAO (usually December-March, but sometimes other months are used) is strongly correlated to winter weather records in northwestern Europe (Figure 1.10, Nesje *et al.*, 2000). 'Positive' NAO years, when air pressure in Iceland is relatively low, are characterised by mild, wet and windy winters in Iceland, the UK and southern Scandinavia. Southern Europe is drier, and western Greenland is frequently colder. 'Negative' NAO years are characterised by the reverse, and tend to occur when high pressure areas over Greenland or Scandinavia/Russia extend their influence into the North Atlantic and force the polar front further south. The southerly depression tracks tend to bring wetter and stormier winters to southern Europe and colder, drier conditions to northern countries, including Iceland.

Oceanic currents

Iceland is warmed by the Irminger Current, an offshoot of the North Atlantic Drift which encircles Iceland (Figure 1.11). In the Greenland Sea off northern Iceland, there is a battle between this warmer water and the cold East Greenland current flowing southwards along the coast of Greenland. In a mirror of events within the atmosphere, the boundary between these two divides water of sharply contrasting temperatures. If the East Greenland Current dominates, it brings sea ice closer to the

northern shore of Iceland which has a dramatic cooling effect on the local climate. This tends to occur in late winter and spring, and is associated with cold weather and hardship during the first part of the year (Ogilvie and McGovern, 2000). Such conditions were last prevalent during the 'Great Salinity Anomaly' years between ~1962 until the early 1970s (Bigg, 1996). They were characterised by colder surface water temperatures and reduced salinity in comparison to the preceding 15 years (Lamb, 1979). Sea ice was recorded frequently around Iceland, and on land there was a cool period from 1965-1971, as well as a persistently negative NAO index.

The combined pattern of atmospheric and oceanic circulation has led Dawson *et al.* (2002; 2003; 2004) to relate weather records around the North Atlantic and the GISP2 ice core records by examining the 'see-saws' caused by different quasi-stable weather patterns. Some of these patterns are shown in Figure 1.12.

The inherent variability in Icelandic weather is important in considering likely effects upon those living on the island. In a Norse-age agrarian system, animal husbandry was a key ingredient for subsistence, supplemented by fish, birds and (occasionally) sea mammals. Management of animals centred on the areas of infield, outfield and common grazing land. Within the infield, hay was grown for winter fodder, while animals graze on the outfield and common grazing. During the winter, the animals are brought in, and feed of the fodder collected in the previous summer. The consequence is the crucial relationship between the quantity of fodder harvested and the number of animals that are kept. Too many animals, a poor harvest or a long winter will result in the loss of livestock, and the key to this is the length, start and end of the growing season. Timber could be imported for roofing of buildings, but a supply of wood is required for fuel and to make charcoal for iron smelting (iron tools are crucial, particularly for the harvest), and therefore trees are a valuable economic resource if they are scarce.

Summary of sections I.2 & I.3

These two sections have introduced the setting of this project: both in terms of the regional climate and weather patterns, and the patterns of Norse settlement in Iceland. This setting is that which climatic change influences, both the physical setting and the

settlements and farming of Medieval Iceland. The issue of climatic *change* has not been discussed in full, though was mentioned in relation to some of the events following Settlement of Iceland. Climatic change forms the focus of Section I.4, with the emphasis being upon the evidence for climate change surrounding the Medieval period in the North Atlantic.

I.4. Climate change: Did climate help or hinder Norse exploration and settlement of the North Atlantic?

I.4.1: Introduction

The previous two sections discuss the environment within the North Atlantic and some of the issues regarding settlement of Iceland. A key questions is the current level of understanding about climatic conditions shortly prior to and during the period of settlement, between the eighth and eleventh centuries A.D. This period includes what has loosely been termed the 'Medieval Warm Period' (MWP), within which there has been some indication that environmental conditions were relatively favourable, allowing the overoptimistic settlement of marginal areas of Iceland and Greenland (Thórarinsson, 1956; Buckland *et al.*, 1994; Barlow *et al.*, 1997). The existence of a climatic 'window' during the Medieval would be of great relevance to those researching the history of North Atlantic settlement, and leads to some further questions:

1: Were the Norse reacting to favourable climatic conditions during this period, and did reduced frequency of sea ice and storms permit exploration, before inhibiting subsistence and trade in later years?

2: Is there any indication of how much warmer the climate was during the MWP?

3: What was the scale of Medieval climatic vaiability?

The first question cannot be answered from climatic records alone, but there is some evidence available to examine questions two and three. This section of the thesis explores present knowledge surrounding the patterns of climatic variation surrounding the Viking Age.

The 'Medieval Warm Period' and the 'Little Ice Age'

Simplistically, the climate of the last 1000 years has been divided into a warm early phase ('Medieval Warm Period' or 'Little Climatic Optimum') and a cooler later phase (the 'Little Ice Age'), which ended sometime between 80 and 150 years ago (Hughes and Diaz, 1994). Conceptually, the observed and suggested regional effects of these can be seen in Figure 1.13. Considerable doubt exists over the timing and extent of the MWP, with the term being used loosely to describe any sustained warmer spell between 900 and 1400AD (Hughes and Diaz, 1994). Much of this uncertainty lies in the difficulty of clearly resolving changes around the MWP, and in determining a real climatic signal from a 'noisy' raw record. In particular the inability of glacial records to indicate *warm* periods is a problem for glacially-derived information (Grove and Switsur, 1994; Kirkbride and Brazier, 1998). The documentary record from this time period is patchy, and good European records are limited to continental sources, and there are relatively few good records for Iceland (e.g. Ogilvie, 1991). This is all in contrast to the extensive documentary and proxy records available for the more recent LIA; despite this there are still some debates in Iceland regarding the timing of the LIA maximum, particularly in relation to glacier fluctuations (Evans *et al.*, 1999; Bradwell, 2001a, b; Casely, 2001; Kirkbride and Dugmore, 2001; Casely and Dugmore, 2004).

Some researchers have even questioned the existence of the MWP and LIA as distinct climatic episodes (Hughes and Diaz, 1994; Jones and Bradley, 1995), though most others have embraced the terms. There is a wide variety of proxy evidence that points towards climatic events during this period, and more recently there have been a number of (mainly northern) hemispheric summaries of climatic change during the last 1000 years (Jones *et al.*, 1998; Mann *et al.*, 1999; Crowley and Lowery, 2000). Glaciogeomorphic evidence during this period has been discussed by Grove (1988) and Grove and Switsur (1994). The diversity of the research, and the variety of climatic proxies used has the consequence that there is not a full consensus on the nature of climate during this critical period of Norse exploration of the North Atlantic.

Short-term climate variability

The pattern of climatic variations over the last 1000 years shows considerable interannual, interdecadal and multidecadal variability, as shown in instrumental records from the past 100 years (IPCC, 2001). The patterns of greatest LIA cooling show considerable variation in timing and extent across regions or interhemispherically (Grove, 1988; Bradley and Jones, 1995). Consequently the presence of a globally uniform, synchronous, and significant MWP or LIA is perhaps unlikely. However, within the North Atlantic and across northwestern Europe, there appears to be greater agreement in the presence and timing of colder episodes during the LIA, and so there may have been comparable regionally significant warm episodes during the Medieval.

The North Atlantic has one of the most dynamic climatic regimes in the world, and is recognised as one of the most sensitive to climatic change. It exhibits rapid interannual fluctuations that are characterised by the North Atlantic Oscillation, and it was around here that the strongest evidence for the LIA was found (Hurrell, 1995; Hurrell and van Loon, 1997; Pohjola and Rogers, 1997; Grove, 2001; Wanner *et al.*, 2001). There are also indications that this region had the strongest expression of the MWP (Lamb, 1977; Mann *et al.*, 1999). The interaction between this climatic sensitivity and Norse exploration of previously uninhabited islands makes the area particularly important for both climatic and human-environment research (Stötter and Wilhelm, 1994).

Relative sensitivity of weather, climate and proxy records

Although many areas have had changes in climate during the past millennium (Grove, 1988; Dahl and Nesje, 1994; Pfister *et al.*, 1998), these changes are small in comparison to those occurring at the end of the last major glacial episode, between c.22,000 and 10,000 ^{14}C years ago. Changes are typically on the order of ±1°C, in stark contrast to the increase of 7°C within a few decades at the end of the Younger Dryas (Dansgaard *et al.*, 1989). A 1°C change is considerably smaller in amplitude than interannual variability at any one site: for example even the maritime

Stykkishólmur record has a range of 4.3°C in annual averages, and monthly averages exhibit the variability even more starkly (Figure 1.14). This effect complicates the identification and interpretation of longer-term and regional patterns (e.g. Dowdeswell *et al.*, 1997; Dawson *et al.*, 2003).

Due to the small range of the century-scale variations, Broecker (2001) made the suggestion that the only records sufficiently sensitive to record Late Holocene climatic changes were glacier ELA fluctuations and borehole thermometry (Figure 1.15). Such a controversial statement implies the elimination of many valuable long-term records used to reconstruct Late Holocene climate, as well as greatly limiting the diversity of locations from which data can be gathered (Bradley *et al.*, 2001). Broecker's view highlights some limitations in searching for, and making a positive identification of, warmer conditions during the Medieval. In particular, if a record does not show the appropriate change at the relevant time the sensitivity of the record can be questioned, or conversely the significance of an observed change may be questioned. Below, a wide range of data sources are reviewed, with strengths and limitations evaluated, in order to gain the fullest picture of the evidence for Medieval climatic changes, as they affect the North Atlantic.

I.4.2: Climate and Viking exploration of the North Atlantic

During the 8th to 11th centuries, the Norse became the first European people to explore and permanently settle in the North Atlantic islands. The 'Vikings', as they became known, were excellent seafarers, capable of crossing stormy North Atlantic waters in relativey small boats, and had reached the Faeroe Islands and Iceland by *c.*800 and 870A.D. (Fitzhugh, 2000; Figure 1.1). During this dynamic period, settlements were established between about 870 and 930A.D. and the Commonwealth Parliament was set up in 930A.D. The Norse (through Erik the Red) settled western Greenland around 983A.D. with strong links to nearby Iceland, and the westward expansion culminated in exploration of some of the Canadian Arctic islands, and a brief settlement of North America at L'Anse Aux Meadows (Wallace, 1991; Schledermann, 2000). This rapid period of exploration and settlement took the Norse far from their 'home' climatic regimes in Scandinavia, and across invisible climatic

boundaries in the North Atlantic. While they would be accustomed to snowy and dark winters, large glaciers would be unfamiliar, and the lowland landscape was more marginal in their new lands than those that they departed from. They would be accustomed to the variability and storminess of the North Atlantic, but the effects both of variability and of change are more marked in Iceland and Greenland than they are in Scandinavia. Vésteinsson (2000) has produced a cartoon that neatly summarises some of the environmental conditions in Iceland, assuming a MWP to LIA climate progression over the last millennium (Figure 1.16).

Climate change, Norse settlement and land degradation

In Iceland, settlement had a great effect on the local biota, with the introduction of domesticated sheep, cattle, goats, horses and pigs as the Norse brough their farming methods with them to Iceland. It appears, based on palynological evidence from southern Iceland, that the consequence was rapid deforestation, and subsequent degradation of existing soil and vegetation cover (Hallsdóttir, 1987). A model of the progression of land degradation in southern Iceland has been prepared by Simpson *et al.* (2001; Figure 1.17), and shows a progression from near-complete cover to substantial degradation since before Landnám. However, there are indications from the Mývatn area that deforestation occurred more gradually over a longer time period (Lawson *et al.*, in press; Figure 1.8). Buckland and Dugmore (1991) described it as an 'ovigenic' landscape – modified by sheep, and there is no doubt that the Icelandic landscape has been greatly affected by the arrival of people (Buckland *et al.*, 1994; Figure 1.18). Pigs and goats in particular are capable of breaking the topsoil and roots, exposing the soil to erosion, and consequently pig and goat remains become rare after c.950-1000AD (Ogilvie and McGovern, 2000). People are not the only force that can cause changes in soil cover in Iceland. It is clear from the work of (Ólafsdóttir and Schlyter, 2001) and Ólafsdóttir and Guðmunsson (2002) that soil erosion in Iceland is not restricted to the time of Norse settlement, and that there is likely to be a climatic element, both during the LIA and earlier Holocene cool episodes. Any cooler episodes cause additional strain on Icelandic vegetation cover, whether already stressed by people or not.

Farm abandonment appears to have played a significant role during the 11th to 14th centuries. Sveinbjarnardóttir (1991; 1992) carried out a study into farm abandonment in three areas around Iceland. She found evidence of abandonment at Þórsmörk as early as the 11th or 12th Century, in Fossárdalur before 1362AD, and inland sites at Skagafjörður close to 1100AD. Later abandonments appear to have occurred around the 15th to 17th Centuries. Sveinbjarnardóttir identified that sites furthest inland and at higher altitudes were more likely to be abandoned, and less likely to be resettled. Potential causes for abandonment include disease epidemics, volcanic activity and climatic changes as well as soil erosion, and there are documented abandonments due to epidemics in 1402-4 and 1495, as well as due to the Hekla 1104, Laki 1783, Öraefi 1362 and Veidivötn 1477 eruptions. The significant shift in North Atlantic climate visible in the Greenland ice cores at around 1400AD, may be related to later phases of abandonment (eg Mayewski *et al.*, 1993; O'Brien *et al.*, 1995).

The early farm abandonments prior to 1300AD do not appear to be directly linked to single events such as a major epidemic or volcanic activity, although such events appear to have caused abandonments in some cases such as 1104AD in Þjorsárdalur, which lay close to and directly downwind of the Hekla eruption that year (Sveinbjarnardóttir, 1992). Therefore any widespread cause of farm abandonment seems to return to two interacting factors – that of human impact on the landscape, and episodes of severe climate. Thórarinsson (1956) indicated a pattern of overoptimistic pioneer settlement following *landnám*, leading to permanent farm abandonment. For example, the site at Sveigakót in northern Iceland, which is 'on the edge of the modern erosion desert', has archaeological evidence of sheep, goats, cattle and horses, which would clearly be unable to exist there at present (Vésteinsson, 2001; Simpson *et al.*, 2003). Within the midden at the site, there is "consistent evidence of aeolian deposition" (Simpson *et al.*, 2003), suggesting erosion and dust transport was occurring locally throughout the period of settlement from the 10th to 12th Century. Sveigakót was abandoned in the late 12th Century, and the extent to which climatic change might have exacerbated the drawdown of natural capital remains unclear. The site and its surrounding area is severely denuded at the present day (Figure 1.19).

The Norse settlements in Greenland were even more marginal than those in Iceland. Although this is the case, a successful colony lasted for more than 400 years. The cause for the final abandonment of these settlements is unknown, though the end

appears to have occurred in the Eastern Settlement around the 15th Century. It is coincidental with a colder period recorded in the ice core records (Barlow *et al.*, 1997). Given the settlement's marginality, particularly for a culture centred around sedentary pastoralism, fodder production and dependent on walrus hunting in dangerous northern hunting grounds, the suggestion is of a particularly environmentally deterministic end to the settlement. In contrast, the Inuit lifestyle based around hunting (mostly of marine mammals) was well adapted to, and less sensitive to, colder conditions. Once again however, there are other factors that may have come into play when investigating the demise of the Norse Greenlanders. Potential factors include economic isolation following political events in Norway and Iceland in the 14th century, a damaging plague in the small population, or violent interactions with the native Inuit population. Economic isolation may also be important if the Norse settlements were initially driven by trade: fur and ivory available in relative abundance in Greenland was scarce in Iceland, and so they may have been trading successfully with Iceland and Europe. A change in this trading relationship would spell the demise of the settlements (Dugmore *et al.*, manuscript).

These events in Iceland and the North Atlantic have occurred on a background of ongoing climatic changes that may have helped or hindered exploration and settlement of the North Atlantic islands. There is some evidence from Greenland ice cores, from early written sources, and from later documentary records, of relatively stable, benign climate in the North Atlantic during the settlement period. Unlike the Little Ice Age of the 16th to early 20th Centuries, there are no reliable documentary historical records of climatic change relating to Iceland and Greenland. There is also relatively little evidence of glacier fluctuations during this period, as the Little Ice Age glacier advances have tended to erase these records. This leaves several questions relating to climate and the Norse Settlement of Iceland:

1: What were the climatic conditions experienced by the Norse in the 9th-11th Centuries?

2: Did favourable climates significantly help the Norse in their expansion and colonisation across the North Atlantic, or did the Norse colonise the North Atlantic islands regardless of climatic events?

4: Did the human impact on Icelandic soil and biota happen regardless of climate?

To help answer some of these questions, we need to evaluate some of the existing physical and documentary evidence relating to Medieval climate in the North Atlantic.

I.4.3: Regional evidence: Oceanic sediments, dendrochronological records, documentary records

Evidence from oceanic sediments off northern Iceland

Oceanic cores have tended to be a relatively poor record for interpreting climatic changes on a sub-millennial timescale, with a combination of low sedimentation rates and bioturbation giving poor resolution. However, several sedimentary sequences from the Atlantic Ocean are both sufficiently relevant, and have sufficient resolution to be useful over the Medieval timescale. Those from close to Iceland and east Greenland (Eiríksson *et al.*, 2000a, b; Andrews *et al.*, 2001a, b; Andrews and Girardeau, 2003; Andrews *et al.*, 2003a, b) in particular have the potential to shed light on climatically relevant oceanic changes over the late Holocene close to Iceland.

A series of cores taken by Andrews *et al* (2001) from close to the northern shore of Iceland appeared to show a very similar pattern of percentage carbonate content to the temperature record derived from Summit, Greenland. This was interpreted to be due to variation in marine carbonate productivity rather than from varying terrestrial sediment supply. Consequently, the carbonate variations may be related to the presence of either temperate or polar water at the sample site. The LIA shows up as a distinct carbonate minimum, but there is no clear signal for Medieval warmth. There is a small peak in carbonate content around 1000 cal. BP lasting c. 200 years (event 'c' in Andrews *et al.*, 2001; Figure 1.20) following a trough (Event 'd'), and this may be the expression of a period with temperate oceanic water surrounding Iceland. The Holocene carbonate minima have not been clearly correlated to glacial advances onshore (e.g. Stotter *et al.*, 1999).

Eiríksson *et al* (2000a) carried out a similar study slightly farther north, with comparable results in relation to the LIA. In addition, the foraminiferal data from core HM107-03 shows evidence for warmer oceanic conditions around 500-1000B.P.

with the presence of *Globigerina bulloides*, and reduced percentages of *Neogloboquadrina pachyderma* (sinistral). Diatom evidence from the same core presented by Jiang *et al.*, (2002) concurs with this, and estimates summer sea surface temperatures around 7.7°C for a period centred around A.D.1100 (correlated with the Hekla 1104 tephra), and a reduction of 1.5°C by A.D.1500. Ice-rafted debris also reaches a minimum between c.500-1000B.P. (Eiríksson *et al.*, 2000a). These results are interesting in that they link the relative strength of the warm Irminger Current and the cold East Icelandic Current around the area of the coring site north of Iceland (Eiríksson *et al.*, 2000a,b).

A further connection between oceanic and terrestrial Medieval climatic change comes from the NEAP15K core south of Iceland (Bianchi and McCave, 1999). This core shows periodicity in the strength of the Iceland-Scotland Overflow Water, and an oscillation from faster to slower ISOW flow coincides with the transition between MWP and LIA. While it is possible that these oscillations are coincidental and unrelated, they may be linked to oscillations found in ice core evidence (O'Brien *et al.*, 1995) and ocean core evidence (Bond *et al.*, 1997), as well as to oceanic changes occurring around northern Iceland.

Evidence has also emerged from elsewhere in the Atlantic for the presence of warmer sea surface temperatures around the Medieval Warm Period. A core taken in the Sargasso Sea shows percentage carbonate and oxygen isotope evidence for a 1-2°C lowering of temperature during the LIA and a 1°C rise during the Medieval period (Keigwin, 1996). The location of the Sargasso Sea at the 'upstream' end of the Gulf Stream is suggestive of larger-scale oceanic temperature/circulation changes being influential in the climatic changes of the North Atlantic, and therefore of MWP having it's causes beyond the North Atlantic. Wetter conditions inferred from isotopic studies of a Caribbean lake core between ~500-1100AD may also indicate a different climatic state to drier conditions before and after this period (Hodell *et al.*, 1991).

Evidence from oceanic sediments around Greenland

Oceanic productivity would have been a valuable component to settlement survival in Greenland. Arneborg *et al.*(1999) show a shift from *c.*20% to *c.*80%

marine diet during the period of Norse settlement in Greenland. Throughout much of this period, Lassen *et al.*, (2004) suggest that the inner fjords around the settlement experienced greater wind stress, indicated by enhanced water column mixing (*c.*885 – 1240A.D.), although with warmer climate. Lassen *et al.* suggest that this might have limited the scale of intensive fishing. Cooling is recorded after *c.*1405A.D., and is correlated with increased water column stratification and the presence of East Greenland Current water. This connects well with the timing of settlement demise. A similar study by Jensen *et al.*, (2004), using marine diatoms in two cores, records Medieval warming from approximately 770-1325A.D., though with periodic declines in warm-water species, most notably around 1090-1140A.D. These records of wind stress and of cooler episodes within the MWP have been related to similar events in Canada, and the GISP2 ice core by Lassen *et al.* and Jensen *et al.*, though the strength of that relationship is unclear. In east Greenland, there is once again comparable evidence for warmer Medieval ocean temperatures from 730A.D. to 1110A.D., with a 'dramatic' cooling and influence of colder water culminating around 1150A.D. (Jennings and Weiner, 1996), and subsequently a gradual cooling from 1300-1630A.D. into the LIA.

In summary, there is relatively abundant evidence from Icelandic and Greenlandic ocean cores that suggests warmer ocean water conditions between about the 8th Century and the end of the 13th Century. A transitional phase then occurs towards the LIA by the end of the 15th Century.

Dendrochronology

Dendrochronological records have been produced for a number of locations around the world, producing summer temperature reconstructions. They are not so good at recording long-term trends (e.g. century-scale changes), as the individual records that make up the chronology are too short (Cook *et al.*, 1995), though there are some techniques that attempt to get round this issue. Consequently the records are useful for indicating particular episodes of temperature change, and often highlight the variability in climate. Evidence presented by Hughes and Diaz (1994) suggests a limited amount of correlation between records in relation to the climate of the 10th to 14th Centuries. They cite significant regional variations between NW Europe (Guiot

et al, 1988; Guiot, 1992), Fennoscandia (Briffa *et al.*, 1990) and the Polar Urals (Graybill and Shiyatov, 1989). This most likely is an indication of regional temperature variability, and shows the difficulty of searching for synchrony in short-term climatic variations over a very wide regional scale. Few records show a sustained Medieval warmth, though the Fennoscandian record is one.

Such a conclusion for dendrochronological records is contested by Esper *et al.* (2002), who argues for a distinctly visible MWP in a hemispheric tree-ring reconstruction, with the warmest period occurring from 950-1045A.D. The record of Briffa *et al.*, (1990) are inconclusinve in recording a strong MWP or LIA, and they highlight the variability, with warm years in the LIA and colder years during the MWP (most notably the first half of the 12th Century once again, in agreement with marine sediments). Particularly cold conditions, including frost-damaged tree rings, occurred around 540A.D., with gradual warming to around 1000A.D. is visible in pine records from northern Sweden (Grudd *et al.*, 2002), and includes cold episodes in the 12th Century and 17th/18th Century that are comparable to events mentioned earlier.

Evidence from ice cores

Ice cores have provided a wealth of proxy climatic data, and during the period of interest for this study, the resolution of the cores is as good as annual or sub-annual. Several proxies extracted from ice core material are of particular relevance for Icelandic climate – the $\delta^{18}O$ isotope record, snow accumulation, Cl^- excess and Na^+ excess in particular.

The $\delta^{18}O$ temperature record for the Holocene is relatively stable, with the exception of a large excursion around 8200 years BP (Alley *et al.*, 1997). The accumulation record follows the $\delta^{18}O$ closely, although there are variations in the details. There is an interesting possible indicator of the MWP in the accumulation record of Meese *et al.* (1994), which shows one of the largest Holocene increases in accumulation, associated with several melt layers between about 620 and 1150A.D (Figure 1.21). The LIA and Subatlantic cool periods bracket this episode, and show up as the lowest $\delta^{18}O$ values of the Late Holocene, suggesting lowest temperatures and lowest snow accumulation.

Data for sea salt chlorine has been taken as a proxy for sea ice cover, and is shown in Figure 1.22 (A Dawson, pers. comm.). Cl- excess values from Greenland ice cores have been used as sea ice proxies, allowing the extension of documentary sea ice records pre-Landnám. The record shows a comparable general pattern to that of the long timeseries data and other sea ice records, with reduced (though not continuously) values up until approximately 1400A.D., and a series of peaks during the LIA from *c.*1500A.D. – 1900A.D. This record does not differentiate between sea ice on eastern and western Greenland (Greenland Sea and Davis Strait), and so may be susceptible to see-saw weather effects if any are present.

Sodium ion concentrations, indicative of sea salt transport from the North Atlantic during storm events, show a distinct rise subsequent to 1400AD (Mayewski *et al.*, 1993; Figure 1.23), suggesting stormier conditions during the LIA. In relating storminess and temperature, Dawson *et al* (2003) showed that although there were still years of extreme temperature, there were no years of extreme storminess between 1000 and 1400AD. This is in contrast to the 20th century, which has had both the most extreme sodium ion concentrations, and the most extreme low temperatures, highlighting that the two periods are not exactly alike.

The GISP2 EOF1 record is interesting in relation to Medieval climatic change, as there is a sharp transition preceding the LIA (O'Brien *et al.*, 1995; Figure 1.24). It records changes in the concentrations of ion and particulate matter in the core, and is an indicator of atmospheric circulation changes, perhaps towards a more meridonal airflow during colder periods, and to cooler temperatures as a result.

There is good potential for an effective proxy global temperature record being generated from isotope evidence in ice cores. Cores from Greenland, Tibet, Peru and Antarctica have been used to search for a consistent global isotopic temperature signal indicating medieval warmth (higher δ18O and δD values; Thompson, 1991). Hughes and Diaz (1994) suggest caution when interpreting the isotopic signal, due to a poor relationship between isotopic records and (albeit limited) instrumental records. However, high altitude sites appeared to show more positive isotope concentrations during the period ~800 to ~1550AD. This included Quelccaya in Peru, the South Pole and Crête, Greenland (Thompson, 1991). This perhaps indicates a general trend less affected by local variations than lower altitude records.

An alternative approach to temperature reconstruction from ice cores is through direct temperature measurement down the borehole. The GRIP and Dye 3 ice cores have given a direct record of temperature through the change in temperature down the borehole being related to temperature changes at the surface (Dahl-Jensen *et al.*, 1998; Figure 1.15). The resolution of such borehole records dramatically decreases in relation to time, but are useful over the Late Holocene. The recent rise in temperature, as well as the LIA are clearly shown, with the LIA having two distinct phases. For the Medieval period the resolution is sufficient to show a general maximum in temperature around 900AD and a cooling trend between 1000 and 1400AD. Prior to this, cooler temperatures are inferred for c.2-3000 years BP, and the greatest warmth is indicated between 5-8000 years BP, correlating with the Climatic Optimum observed in palaeoclimatic records elsewhere (e.g. Nesje and Kvamme, 1991; Dahl and Nesje, 1994; Nesje *et al.*, 2001).

The record from the ice cores, and in particular those from Greenland is interesting, and often presents a fairly cohesive picture of a cool, stormy LIA subsequent to 1150-1400A.D., a preceding warm and settled phase (the MWP), and earlier cool and warm periods. There are problems in interpreting the record at a resolution much greater than a century scale, due to the question of connecting the Greenlandic record across atmospheric weather systems and potential see-saws to Iceland and the rest of Europe. This is the reason why Hughes and Diaz (1994) highlight the difficulty in matching meteorological records with ice core data. The problem is enhanced in that the ice cores come from a region with its own stable weather system, the Greenland High pressure, which isolates it still further from changes in nearby weather systems. I would be cautious about using the annual resolution of the ice core data at least for temperature changes, though there is great value in the storminess and sea ice proxies provided by the Na^+ and Cl^- ion data.

Documentary Evidence

Documentary evidence for climatic changes in Europe is relatively extensive by comparison to that from elsewhere in the world, and Lamb (1977) suggested that the warmest period appeared to be '...between 1150 and about 1300, though with notable warmth also in the late 900s.' More rigorous examination of documentary historical sources has led to a reassessment of the evidence, and to some disagreement among

more recent work by climate historians. In relation to Iceland, the most common documented event related to weather and climate was the occurrence of sea ice, which will be discussed later in this section. Ogilvie (1991) discusses the problem of a lack of secure documentary evidence of warmer conditions 'during the early years of Iceland's history', despite considerable circumstantial evidence supporting the idea of a more favourable climate during this time. The main problem has been the validation of documentary records of Medieval climate. Many reviews of the documentary records of Medieval climatic change rely on compilations made from the original records since 1858 (Bell and Ogilvie, 1978). The compilers achieved the admirable job of extracting and translating the scattered references to weather from the very many volumes of original records (written in medieval Latin or Old Icelandic for example). However, there were a number of problems with the methodology of these compilers (including Arago, Hennig, Lowe and Thoroddsen). There are mistranslations of the original dates or descriptions, duplication of events through repetition in different compilations, sources including non-contemporary information, and even sources with fictitious information (Bell and Ogilvie, 1978). Many of these problems were not assessed by the compilers, and sources were not cross-checked for validity.

There are some interesting observations from the documentary sources however. Distinct Medieval warmth is well documented for England, and according to Lamb (1977) vines were grown from at least 1000-1300AD, and perhaps earlier – in significant contrast with conditions since then. Such a distinct episode is not so clear elsewhere, suggesting substantial regional variation in the amount of warming experienced. The best recent evaluation of documentary sources is one for western Europe by Pfister *et al.*, (1998)., who suggests winters of reduced severity between 900and 1300A.D., but highlights great variability in conditions, including some LIA-magnitude severe winters around 1125A.D. The transition from MWP to LIA is thought to be sharp in continental Europe (14th Century; Pfister *et al.*, 1996), but more gradual in England and Iceland (Ogilvie and Farmer, 1997).

The consequence is that, while there is a relative abundance of documentary data from continemtal Europe, there does not appear to be clear evidence for the MWP in Iceland from documentary sources.

Evidence from glacier fluctuations

Glaciers record a smoothed climatic signal, and so advances are likely to occur when there is a trend towards cooler or wetter conditions over a decadal to multidecadal timescale, even if this includes short periods of warmer conditions. A number of Scandinavian studies suggest that some small glaciers completely disappeared during the mid-Holocene and reformed within the last 3,000 years (Nesje and Kvamme, 1991; Nesje *et al.*, 1991; 1994; 2001; Snowball, 1993; 1996; Figure 1.25). The mid-Holocene disappearances involved a 4-500m rise in the ELA, including isostatic effects. Many studies include evidence for glacier oscillations during the last 1500 years, most of which are related to evidence of glacier advance, although a few studies in favourable locations have documented evidence of glacier retreat (Holzhauser, 1984, 1997). Grove and Switsur (1994) review the evidence for glacier fluctuations around this time period, and suggest that there is a lack of montane glacier advances between 900AD and 1250AD, consistent with warmer climatic conditions during this period.

Hemispheric summaries of continuous records

Several authors have gathered continuous records from around the Northern Hemisphere in order to detect changes in the average hemispheric climate of the last 1000 years. One of the main problems of this is that many of these records have a cut-off at about 1000AD, well after the beginning of the Viking expansion. Jones *et al* (1998), Mann *et al* (1999), Crowley and Lowery (2000) and Crowley (2000) have all summarised records to produce a hemispherically averaged composite temperature change (Figure 1.26). These are broadly similar, with 20th Century warming, most severe LIA conditions from c.1550 – 1900AD and peak Medieval warmth from 1000AD (the start of their curves) to 1200AD. Crowley and Lowery indicate that the peak hemispheric Medieval warmth was concentrated to three short phases – 1010-1040AD, 1070-1105AD and 1155-1190AD. They suggest that the mean temperatures of the LIA were only 0.2°C cooler than the temperatures from 1000-1200AD and that LIA temperatures were c.0.5°C cooler than the peak warm periods of the Medieval and of the 20th Century.

Although these continuous records are successful at reconstructing hemispheric variations, there are limitations in relation to the relative regional strength of the

signal. North Atlantic changes tend to be more pronounced than those from many other Northern Hemisphere areas, and therefore details of a North Atlantic signal may be lost or subdued in the hemispheric summary. Also, as discussed by Crowley and Lowery, and previously mentioned by Dansgaard *et al.* (1975), there is evidence for discordance in the timing of peak Medieval warmth within the North Atlantic basin, with peak Medieval warmth occurring significantly earlier in Greenland than in the Central England Temperature series – this discordance will further diminish the signal in a combined record.

Summary of regional evidence

Of the six types of evidence presented, four provide some good evidence for a widespread and well-defined MWP. They are oceanic sediments, ice core data, glacier records and hemispheric summaries, although that from glacier records is very patchy. In each of these records there is evidence for distinctly different conditions in the North Atlantic region during the MWP and LIA, usually in the form of colder, stormier conditions during the latter. The timing of the MWP is ot always as well defined, with large discrepancies obvious between the different records. The other two types of evidence, documentary records and dendrochronology. Do not provide as clear a picture. The evidence from documentary sources is, like that of glacier fluctuations, patchy, and rapidly reducing in quality during the Medieval and before. It does suggest similarities with the better records in terms of warmer climate during the Medieval, but the evidence is not conclusive. The evidence from dendrochronological sources, while showing excellent resolution and small-scale change, does not pick up the large-scale changes sufficiently well to give a clear MWP or LIA signal, and is therefore not so useful in determining the overall nature of the North Atlantic region throughout the MWP and LIA.

I.4.4: Evidence for Medieval climatic changes in Iceland

Having examined records from beyond Iceland in an attempt to determine the nature of the climate during the MWP and LIA in the North Atlantic region, I will now look at evidence from various Icelandic sources for the same time periods. Icelandic climate records of the last 1200 years tend to come from three primary sources. The many glaciers and ice caps on the island have left a considerable

geomorphic record of glacier fluctuations, although this record tends to be, with a few exceptions, limited to the last 300 years of the LIA. Documentary sources provide other information on conditions since the Settlement of Iceland, and in particular the records of variation in sea ice are a valuable indicator of past conditions. The reliance of Icelanders since Viking times on the resources provided by the sea means that severe sea ice had a particularly strong direct effect in addition to cooling the air temperatures. Consequently, sea ice events have been well documented in Iceland. Other indicators that have been used to indicate climatic changes have been the amount of periglacial activity, and the rate of rockfall activity.

Icelandic glacier fluctuations

As mentioned above, the vast majority of Icelandic glaciers reached their maximum extent during the LIA, and there are only a few locations where there are known advances prior to this. Allied to this is the difficulty of demonstrating climatic amelioration from an *absence* of glacial geomorphic evidence: records are censored by the most recent and largest LIA advances (e.g. Kirkbride and Brazier, 1998). Consequently the geomorphic record of Icelandic glacial fluctuations is of limited value for climatic events prior to the LIA, though there are some significant exceptions.

The earliest onset of post-Climatic Optimum glacial expansion appears with the Drangagil Stage at Sólheimajökull, and with the Vatnsdalur I and Baegisárdalur I stages in northern Iceland between approximately 7,000 and 4,000 ^{14}C years BP (Dugmore, 1989; Häberle, 1991; Stötter, 1991; Guðmunsson, 1997; Figure 1.27). Several glacier advances between c.4700BP and 1000BP have been found in the Trollaskagi region of northern Iceland (Häberle, 1991; Stötter, 1991). These events following the Climatic Optimum may each reflect periods when the Icelandic climate was as cool as the LIA, with each cool period being preserved in a few select locations as the advances were only marginally larger than those of the LIA (Stotter *et al.*, 1999). Conditions in northern Iceland appear to have been slightly more favourable for the preservation of these pre-LIA advances, with the exception of Sólheimajökull.

There are a number of locations in southern and central Iceland where pre-LIA glacier advances have been recorded. Mid-Holocene advances have been found at

Hagafellsjökull Eystri, Skálafellsjökull and Sólheimajökull (Thórarinsson, 1966; Sharp and Dugmore, 1985; Dugmore, 1989 respectively), although the Hagafellsjökull Eystri and Skálafellsjökull advances were not as extensive as the LIA. At Sólheimajökull, perhaps due to the nature of the topography, a record of glacier fluctuations extends back to between 45000 and 7000BP, with a recent glacier advance between 600-800AD, and an extended position during the 10th Century AD, as well as during the late LIA (Dugmore, 1989; Figure 1.28). The 600-800AD till has been correlated with a climatic deterioration seen in pollen studies from southern Iceland (Hallsdóttir, 1987) as well as a cooling recorded in the Crête ice core (Dansgaard *et al.*, 1975). A pre-LIA advance was found by Thórarinsson (1964) at Hálsajökull on Snaefell by dating the advance as being prior to the Öraefi 1362 volcanic eruption, although the actual age of this moraine. Thórarinsson's suggestion was that this reflects a glacier advance during the Medieval. Similarly, there is a pre-Ö1362 moraine at Kvíarjökull, lying just beyond the main amphitheatre moraine at this glacier.

The Icelandic glacier evidence points to the presence of several periods prior to the LIA, when temperature or precipitation conditions were favourable enough to place glacier margins close to, and occasionally beyond, LIA maximum limits. It is not clear from the geomorphic evidence whether these advances were solely temperature-controlled, or the length of time that is reflected in a glacier advance.

Sea ice records from Iceland

Iceland was known of prior to Norse settlement, and the account of Diucil, an Irish monk, of a land farther north than the Faeroe Islands is perhaps the earliest existing account of Iceland (Diucilus, 1967). There's interesting evidence of Iceland's 'old enemy' the sea ice: Diucil states that his brethren found the south coast ice-free, but encountered it a days voyage from the north coast, a situation which would be familiar to today's climatic conditions, but perhaps rather more favourable than those of the LIA (e.g. Grove, 1988). This indication of relatively ice-free conditions prior to the Norse settlement appears to follow through to Norse colonisation and more favourable conditions following settlement, according to Ogilvie (1991; 2000; Ogilvie and Jonsson, 2001). Ogilvie has come to this conclusion despite the documentary evidence for this being very limited in

comparison to her documentary studies of more recent climatic changes. The earliest documentary sources from Norse times date from the writing down of the sagas during the 12th Century. As the writers would write in relation to the environmental conditions that they lived in, they may not have accurately reflected the conditions at the time of the stories.

The incidence of sea ice around the Icelandic coast is an important record of climatic change, due to its impact upon the terrestrial environment. Several authors have studied the historical records and produced palaeoclimatic reconstructions based on the presence of sea ice around the coasts of Iceland (including (Koch, 1945; Bergþórsson, 1969; Sigtryggsson, 1972; Ogilvie, 1984, 1992). Bergþórsson (1969; Figure 1.29) used statistical methods to estimate the average temperature in Iceland based on the recorded incidence of sea ice (1591-1846AD), and the relative number of severe years (930-1591AD). This was subsequently compared with the Camp Century ice core record from Greenland, suggesting a good correlation. For the Little Ice Age period, colder phases were noted early and late in the 17th Century, during the mid and late 18th Century, and in the late 19th Century (Bergþórsson, 1969). Ogilvie (1984) found periods of extensive sea ice at the end of the 17th Century and in the mid-18th Century, using more rigorous source analysis than Bergþórsson.

The sea ice indices of Koch, Bergþórsson and Ogilvie agree reasonably well from 1600AD onwards, but prior to this there are problems with the interpretation of few and scattered sources. The work of Thoroddsen (1916-17) in compiling climatic and sea ice information over the previous 1000 years, while valuable, was not as rigorous as modern standards for source analysis (e.g. Ingram *et al.*, 1978). As this provided the basis of Koch's and Bergthorsson's indices, the accuracy of the indices prior to 1600AD, and especially prior to 1200AD is questionable. Bergþórsson's (1969) sea ice index, which between 900 and c.1200AD appears to be based on only two data points, appeared to be suggesting fluctuating and cool conditions. This was taken by Gudmunsson (1997) as evidence of relatively cooler Medieval conditions, despite the sparseness and unreliability of the data (see Ogilvie, 1984; 1991). Astrid Ogilvie has evaluated in detail the climatic and sea ice data, and was the first to critically assess the sporadic evidence prior to the 17th Century. Ogilvie accepted evidence, although sparse, for milder conditions around the time of Settlement. She indicated that the first indications of cooler conditions came towards the close of the 12th Century, with more continuously cold conditions only first appearing in the late 13th Century and

followed by greater variability in the pattern of mild and severe winters (Ogilvie, 1984; 1991). Ogilvie's conclusion from documentary sources is supported by the Cl^- excess data (Figure 1.22), which indicate reduced sea ice cover for most of the 9th to 14th Centuries.

Palaeoecological evidence for Medieval climate in Iceland

Pollen records from Iceland are dominated by the reduction in birch forest from the time of settlement. A combination of a relatively poor range of species, dominated by Cyperaceae and birch pollen, as well as the enormous human impact after c.870AD (coincidental with the 'Landnám' tephra layer) limits useful pollen information to the period prior to human settlement (Caseldine and Hatton, 1994). Pahlsson (1981) suggested warmer conditions just prior to Settlement based on an increase in birch pollen. Conversely, there is some evidence emerging from several studies of a change in environmental conditions prior to Norse settlement. A reduction in birch pollen prior to the deposition of the Landnám tephra was noted by Hallsdóttir (1987). This may be related to the 'abrupt stratigraphic change 10-30mm below the Landnám Tephra' marking a change to widespread increased aeolian erosion and deposition (Dugmore and Erskine, 1994), which would reasonably be associated with a reduction in vegetation cover. There are three main likely causes of such an event: climatic cooling, clearing by pre-Norse Celtic settlers, or volcanic activity. The impact of tephra fall on vegetation appears to be limited, as seen in Caseldine and Hatton's (1994) study. The cause of this reduction in birch cover remains unclear, as do the consequences of there being cooler climate close to the time of Norse settlement.

I.4.5: Summary and conclusions: is there evidence for Medieval warmth in the North Atlantic?

Hughes and Diaz conclude that '...the time interval known as the Medieval Warm Period from the ninth to perhaps the mid-fifteenth century A.D. may have been associated with warmer conditions than those prevailing over the next five centuries (including the twentieth century), at least during some seasons of the year in some regions.' This is indicative of many of the problems facing those attempting to infer climatic records for this period from the variety of sources – namely the spatial and

temporal variability within the records leading to a blurring of larger-scale changes. Each method used over this time period has its own problems, leading to greater difficulty in interpreting a temperature signal. However, sufficient evidence exists from a variety of records in regions close to Iceland (including Scandinavia, Greeenland and England) to warrant further investigation into the Medieval climate experienced there. There appears to be agreement that on a hemispheric scale, Medieval warmth is present and distinct from the Little Ice Age, although there are discrepancies in the specific timing of the warmth in different regions of the North Atlantic. These may well be due to the influence of shifts in the Atlantic storm tracks, which strongly influence the climate of regions in different areas of the North Atlantic. Grove and Switzur (1994) concluded that there is good evidence for glacier advances prior to 900AD and following 1250AD, with relatively few advances during that interval, providing a clearer bracketing of the MWP. Allied to documentary evidence for periods of warmth during the Medieval time, then the case for a period of ameliorated climate, if not constantly warmer, in areas close to the North Atlantic is very strong.

Summary of evidence for Icelandic Medieval warmth

There is some documentary evidence for warmer conditions in Iceland during and following Settlement (Ogilvie, 1991); Ogilvie *et al*, 2001), but crucially the sparse sea ice information appears to support criteria for warmer conditions. There appears to be little evidence of a glacier advance in Iceland dated to between <1000AD and early LIA advances. However, the information is very sparse, and the fluctuations of Sólheimajökull are the only evidence of a 10^{th} Century advance or stillstand if the inferences of Dugmore, (1989) and the modelling work of Mackintosh *et al.* (2002) are correct in suggesting at least a partially climatic signal for this unusual glacier. The only case for an advance after the Ystagil Stage at Sólheimajökull (between 600 and 800AD) is the Baegisdalur II advance before *c.*1000AD which is based on a minimum ^{14}C date on the beginning of peat development and hence may be older (Häberle, 1991) This would suggest that indications of cooler conditions during the Medieval period in Iceland, as suggested by Gudmunsson (1997) are incorrect, and that the suggestions of warmer conditions may well be right. Icelandic records would then be in line with the European documentary evidence of Medieval warmth (Pfister

et al., 1998) and evidence of glacier retreat (Grove and Switsur, 1994). There are some complications due to conflicting evidence between inferred warmer conditions based on documentary evidence and evidence of glacier advances in the 10th Century AD (Stotter *et al.*, 1999). These advances may be due to one of three causes in a warmer 10th Century: markedly increased precipitation at high levels, increasing the mass balance (unlikely, as the glaciers on the south coast are likely to be temperature dependent; Mackintosh, 2000); unusual glaciological or hypsometric characteristics (Dugmore and Sugden, 1991); or cool periods interspersed in Medieval warmth that are of sufficient intensity to cause a glacier advance.

I.5: Conclusions to background material

Viking expansion and settlement during the 9th Century, during which they crossed the Atlantic and settled Iceland and Greenland, is coincidental with a relatively little understood time period for climatic change in Iceland – the 'Medieval Warm Period'. It is a time during which documentary sources are sparse, and the glaciogeomorphic record is poor. Spatially, although there are Medieval-age documentary sources, and proxy records scattered round the North Atlantic, there are few well-constrained or high resolution records from within the region explored by the Norse, with the exception of relatively low-resolution oceanic cores. The evidence that exists points towards there being relatively warmer air and sea temperatures, and if the dendrochronological values from Scandinavia area guide, perhaps these are on the scale of 1°C rise (summer temperature), with less stormy and less severe winters. In all records there is evidence for considerable interannual and interdecadal variability, with colder episodes reported as glacier advances, sea ice anomalies, winter severity, ocean temperature and summer temperature. There is some agreement that the latter part of the MWP included a particular cold episode in the early 12th Century, and a gradual transition to the LIA through the 14th and 15th Centuries.

One issue is the climatic significance of the 'Medieval Warm Period' and 'Little Ice Age', as discussed above. Given the interdecadal variability within the proposed warm and cold spells, some have questioned whether there are in fact distinctly warmer and cooler episodes, as it is clear that the year-to-year variability is much larger than the long-term changes. From the evidence presented above I would argue that, though the temperature variations are only on the order of 1°C, a Medieval 'warm' period is visible in the hemispheric timeseries as well as oceanic and ice core proxies, and that the LIA is even more clearly visible and highlighted by glacier advances. This suggestes that the MWP is significant enough to be a distinct climatic episode, though perhaps one which is most pronounced only in the North Atlantic.

There is considerable spatial variation in the timing of peak Medieval warmth, with the warmth in England occurring towards and beyond the end of the globally averaged warm peak indicated by Crowly and Lowery (2000). Discordance in timing also appeared in the records assessed by Dansgaard *et al* (1975) with the Greenland peak warming occurring much earlier (9th Century). This raises the question, for the

Iceland region, of what time period the 'Medieval Warm Period' represents, and whether it is coincidental with the MWP as seen in Europe. Given the distances involved, and as seen in the see-saw effects in weather records (Dawson *et al.*, 2004), the timing of peak warmth may not be uniform across the North Atlantic. The peak of the 'Medieval Warm Period' may depend on the most favourable regional (e.g. Greenland) weather conditions superimposed on higher average global temperatures.

The extent to which climate aided settlement, and the possible extent to which it subsequently put stress on the settlements remains unclear from the present research. Some of the settlement abandonments in Iceland and Greenland occur at climatically significant time periods: the evidence for the end of the MWP points towards cooling variously between the 13th and 15th centuries. The oceanic evidence for higher productivity levels in Greenland, reduced sea ice around Iceland, and reduced Atlantic storminess would all be conditions more conducive towards successful or uninhibited settlement.

There are limitations on all the present data used to explore the climate of the Medieval period in relation to the North Atlantic in terms of the location of the records, the length of the records, the resolution, reliability and dating accuracy, and the relevance to the climate of the North Atlantic. General hemispheric reconstructions agree on the presence of Medieval warmth, but these reconstructions include data from regions far outwith the North Atlantic, and only go back to 1000AD. There is a considerable need for a high resolution record from within the North Atlantic that extends throughout the age of Norse exploration and settlement. Additionally, there is little terrestrial climatic data concerning Medieval Icelandic climate: most existing data has the complication of the human impact of settlement. It is this challenge of distinguishing the terrestrial climatic conditions and relating them quantitatively to settlement patterns that will be examined in the subsequent chapters.

I.6: Primary Research Objectives and Questions

These objectives relate to points raised within the contextual discussions of the previous three sections. They will be tackled in the subsequent chapters of this thesis.

Norse Settlement and subsistence in Iceland:

1: 'Abandoned frontiers' – is there a case for the argument that early settlement took place in regions, particularly inland, that were subsequently unable to sustain the population because of climatic change?

2: 'Rangeland degradation' – to what extent did climatic change play a significant role in influencing degradation of the upland areas in Iceland?

3: Deforestation – has climate played a role in the reduction of Icelandic forest (and other vegetation) cover?

Late Holocene climate in Iceland:

4: To assess the potential impacts of warmer climate/weather, and the effects of different spatial and temporal temperature distributions during the Medieval period in Iceland.

5: To constrain the timing, extent and nature of any significant variations or variability in climate during this period.

6: To model spatial and temporal patterns of Medieval climate change in Iceland. This includes envelopes of equilibrium line altitude change, as well as consequent changes in other environmental boundaries during the Medieval period:

Further objectives, combining objectives 1-6:

7: Environmental modelling:

a) To model the present-day mass balance and resulting equilibrium line altitude for Iceland.

b) To model changes based on fixed points determined from lake coring and dating.

c) To model associated potential ecological changes across Iceland.

8: To determine the spatial extent and significance of climatically-induced environmental degradation following settlement in Iceland.

9: To integrate this information into the context of ongoing research into settlement and subsistence in Iceland, and to evaluate the likely climatically-derived impacts, if any, on those living in Iceland at that time.

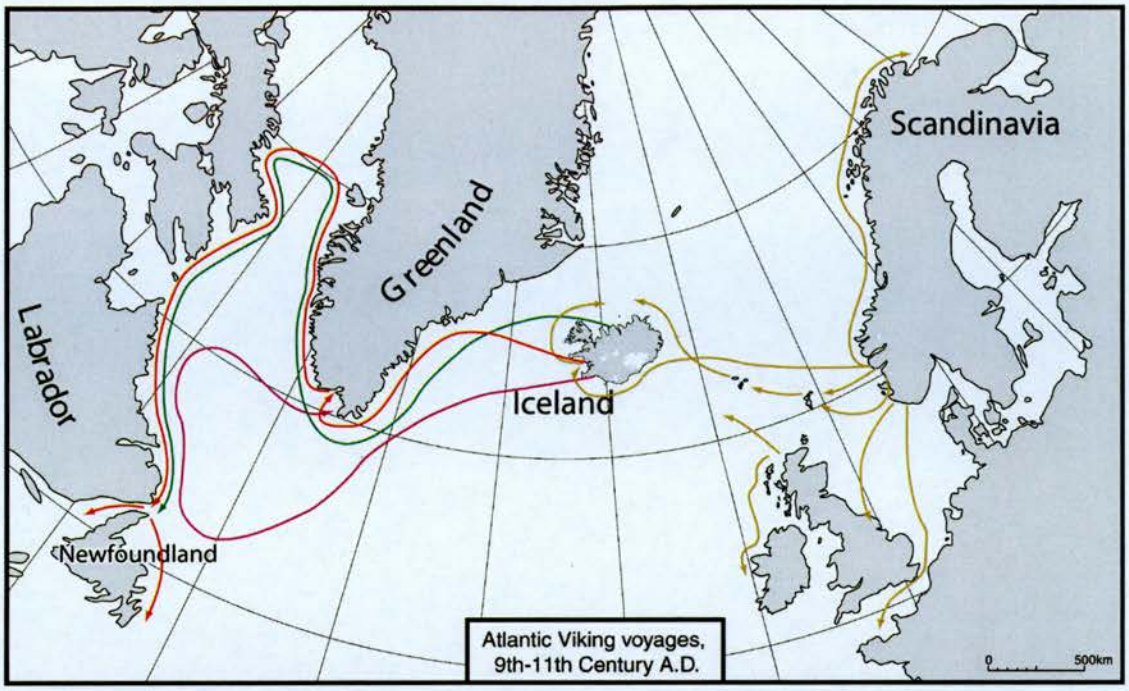

Figure 1.1: 8th to 11th century Norse exploration of the North Atlantic (Fitzhugh, 2000). Key individual voyages shown are Eiríkur Rauða and Leifur Eiríksson (red, A.D. 985 and 1000), Bjarni Herjolfsson (purple, A.D. 985-6) and Thorfinn Karlsefni (green, A.D. 1005).

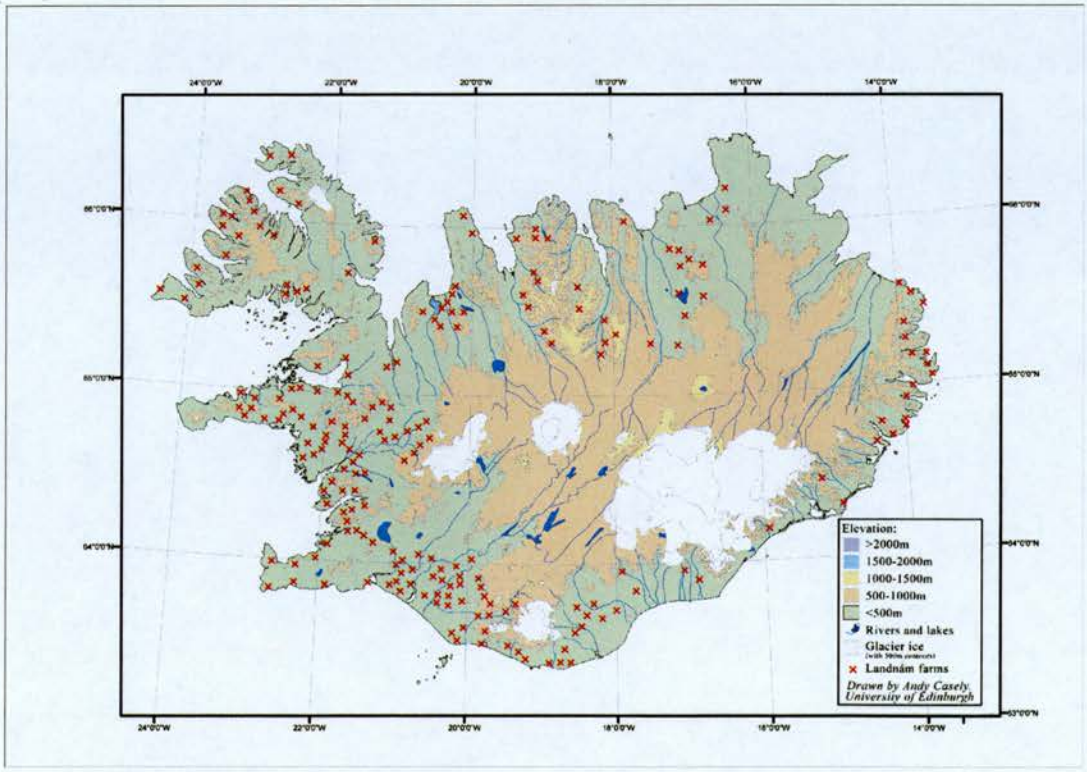

Figure 1.2: Settlements established during the early *Landnám* period, A.D.870-930 (Vésteinsson, 2000)

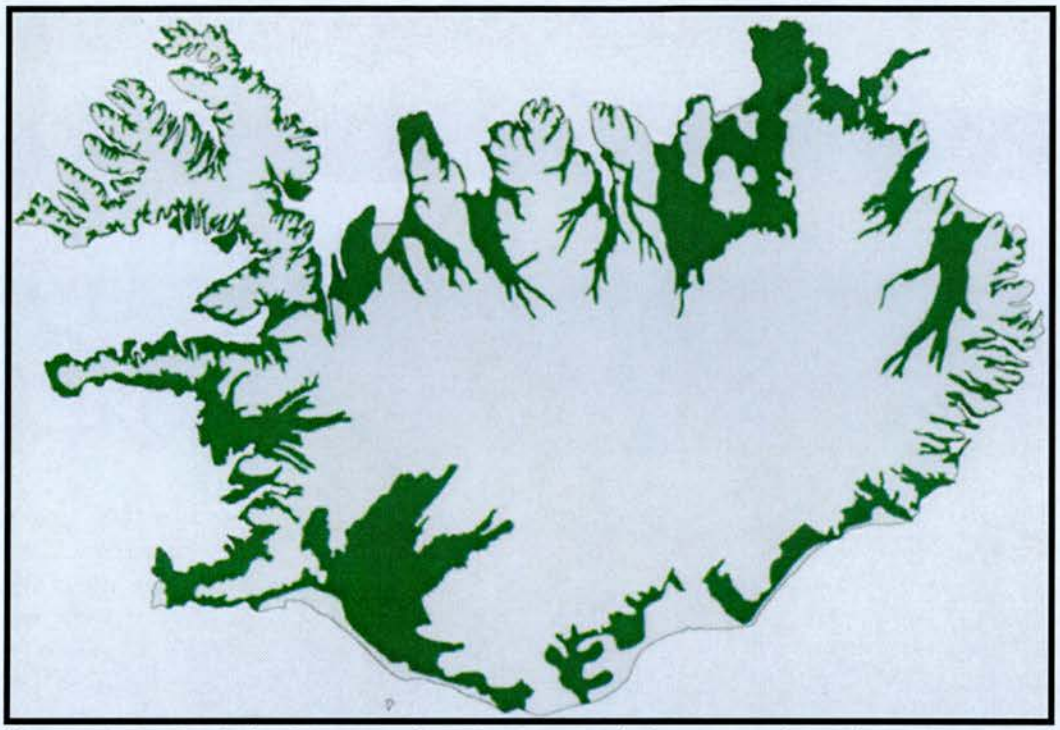

Figure 1.3: Pre-Landnám forest cover in Iceland (Icelandic forestry service, written comm.).

.

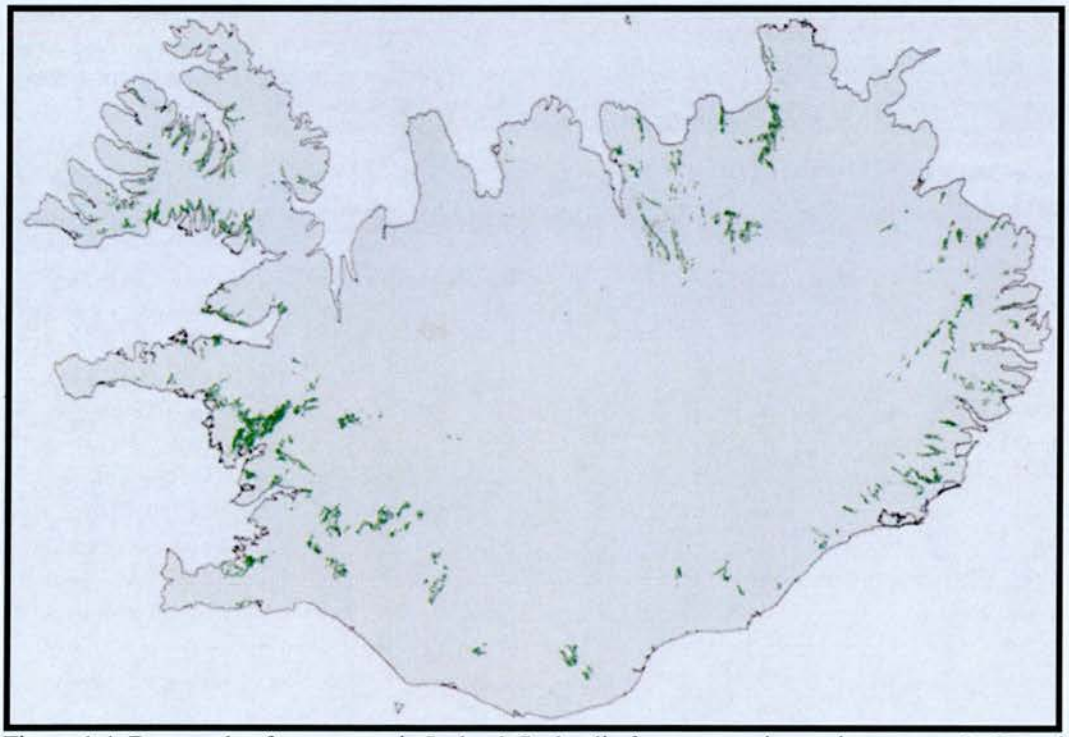

Figure 1.4: Present-day forest cover in Iceland (Icelandic forestry service, written comm.). Remaining woodland tends to be in inland and marginal areas.

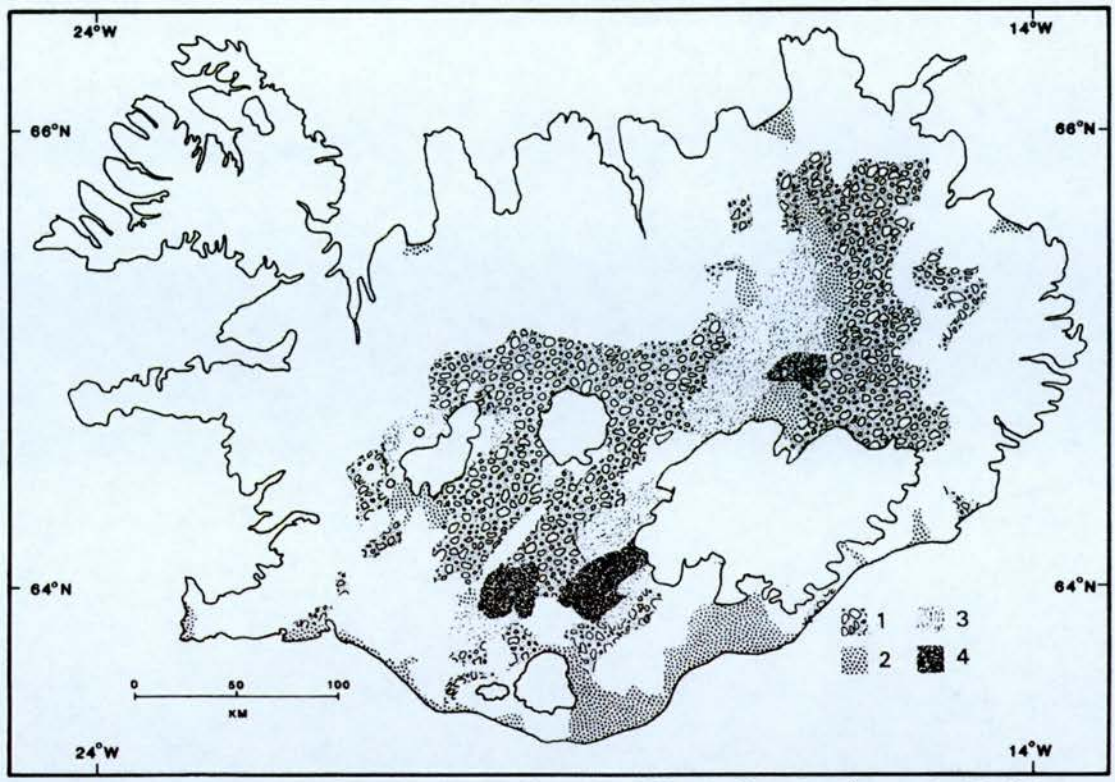

Figure 1.5: Denuded land area in Iceland (Arnalds *et al.*, 1987). Surface deposits are: 1: Glacial deposits; 2: Sandy areas; 3: Postglacial lavas; 4: Pumice.

Figure 1.6: A small rofabarð in Geithellnadalur, eastern Iceland. The birch trees are on the original land surface, indicating >1m of soil has been eroded, although birch trees are capable of survival at present.

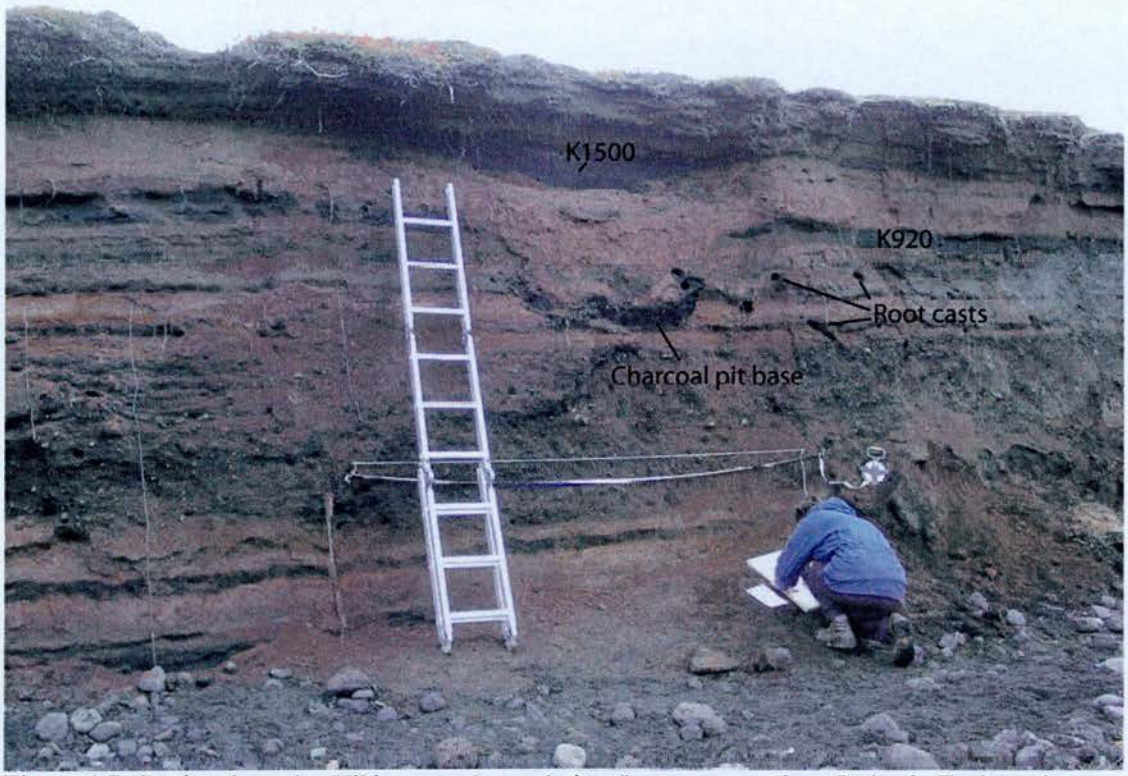

Figure 1.7: Section through a Viking-age charcoal pit at Langanes, southern Iceland. Tephras above and below the pit overspill constrain the age to between the Eldgjá 935 and Hekla 1341 layers, and wood macrofossils from the charcoal have been radiocarbon dated to the 11th Century (Dugmore *et al.*, in press). Tree root casts are also visible in the older layers.

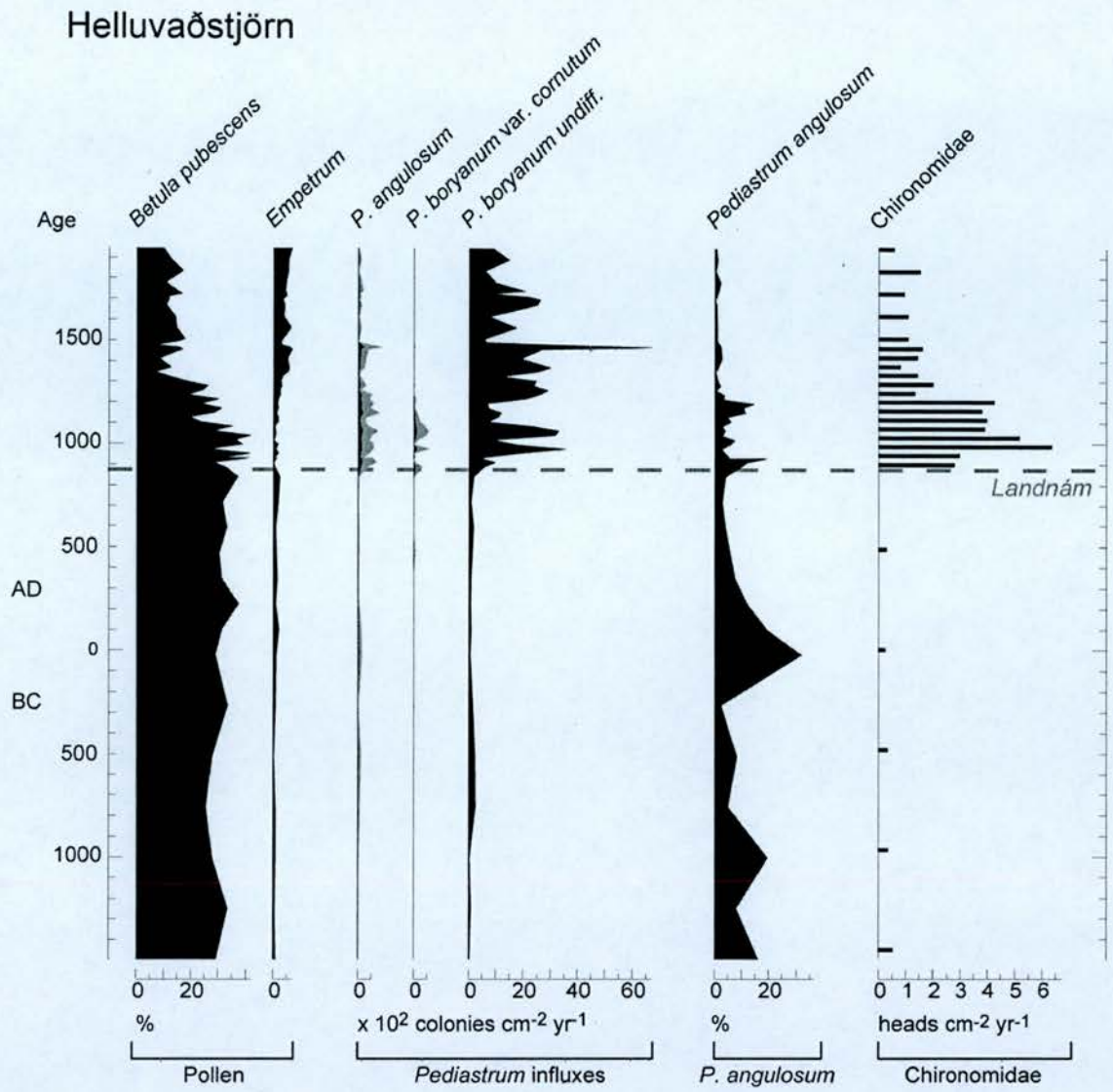

Figure 1.8: Pollen and chironomidae from Helluvaðstjörn, near Mývatn, north Iceland (Lawson *et al.*, in press). The pollen indicates a gradual decline of birch woodland during the post-*Landnám* period.

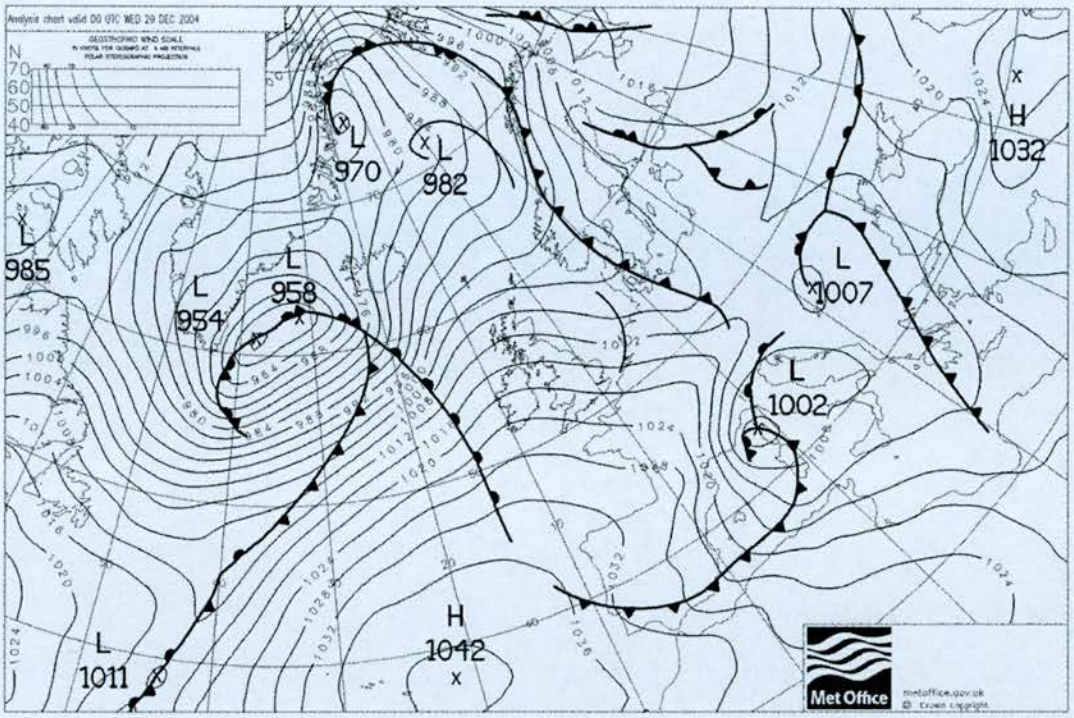

Figure 1.9: A sample Atlantic sea surface pressure chart from 29th December 2004. It demonstrates a common synoptic situation in the Atlantic winter: dynamic low pressure areas close to Iceland (bringing high winds and rain or snow to Iceland), and the quasi-stable Azores High located west of Spain (UKMO, 2004)

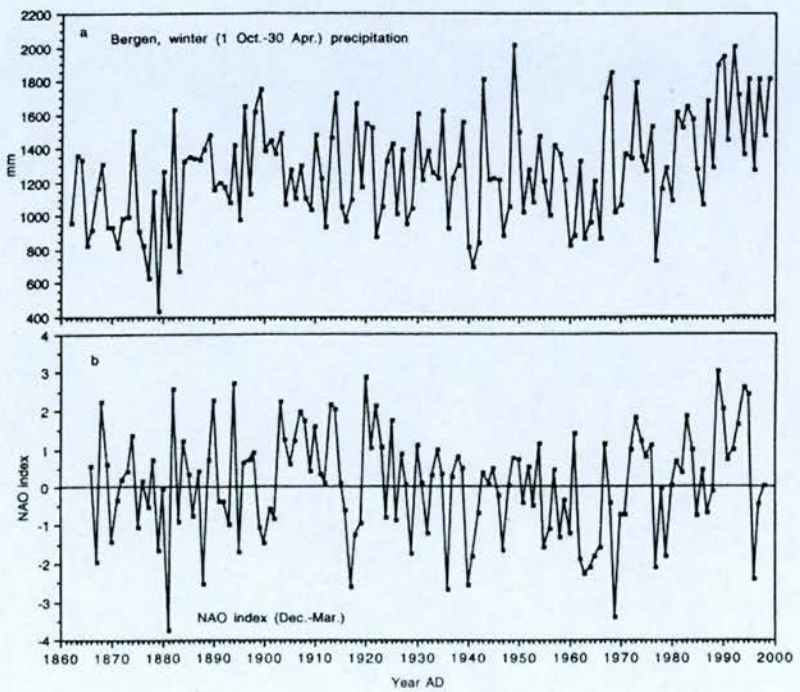

Figure 1.10: An example of the relationship between the NAO index and climate in the North Atlantic: Bergen in western Norway is strongly affected by the prevailing southwesterly winds (strongest during positive NAO years), and this is reflected in the levels of precipitation (Nesje *et al.*, 2000).

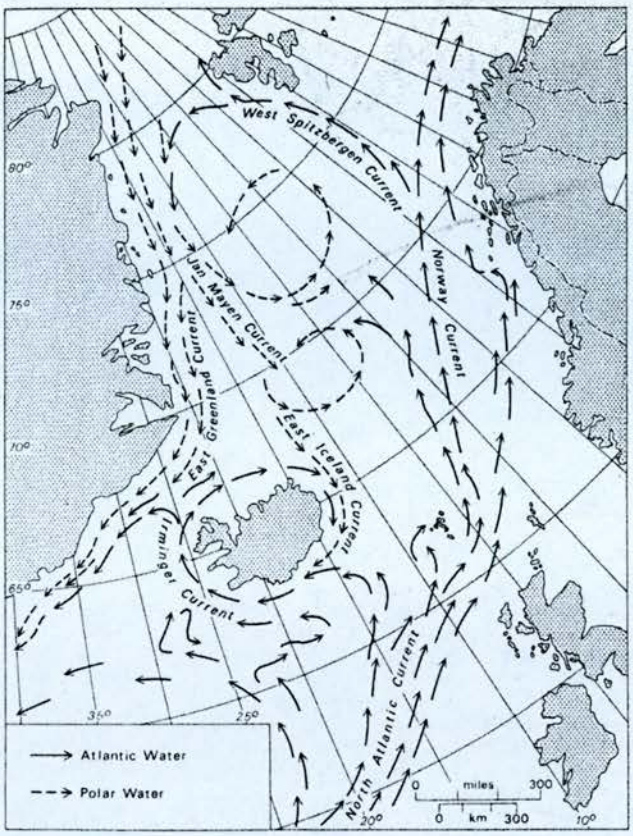

Figure 1.11: Surface oceanic circulation around Iceland. Key features are the warm Irminger current, an offshoot of the Gulf Stream encircling Iceland, and the two cold currents, the East Greenland and East Iceland currents (Grove, 1988).

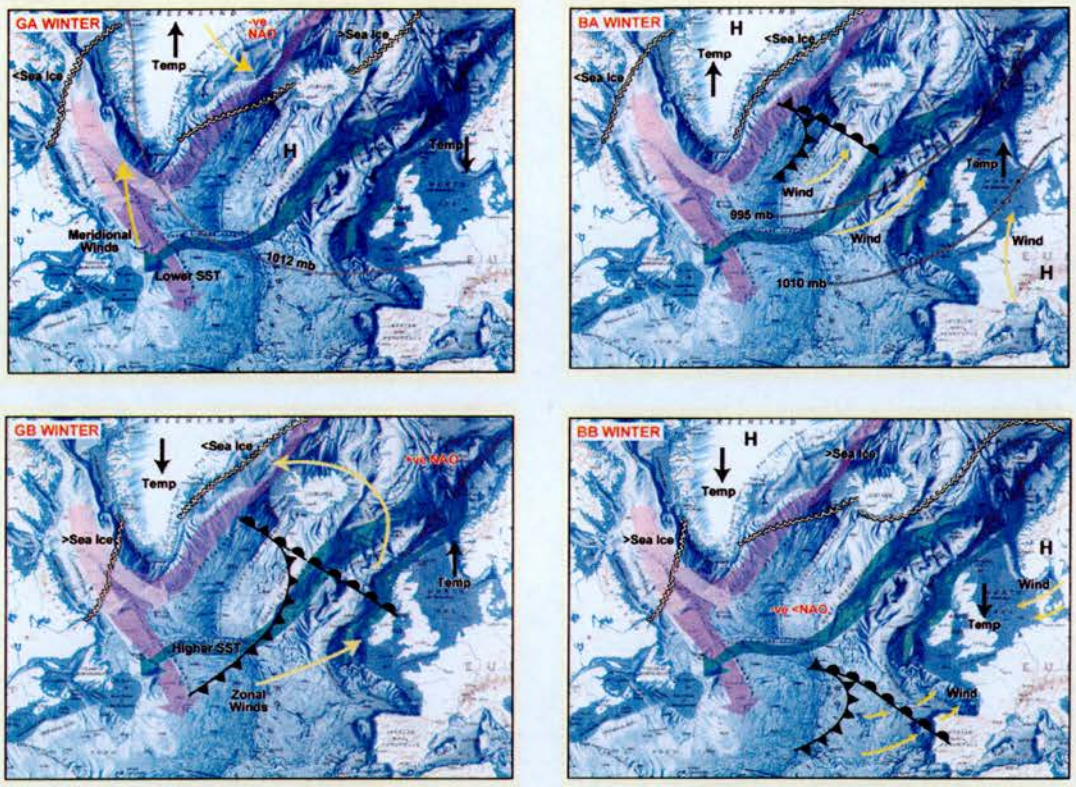

Figure 1.12: Some generalised patterns of North Atlantic climate variability, indicating prevalent pressure systems, wind directions, sea ice prevalence, and spatial variations in temperature under different scenarios. GA = Greenland Above; GB = Greenland Below; BA = Both Above; BB = Both Below (referring to temperature anomalies, as in Dawson *et al.*, 2003; 2004, diagram by Dawson).

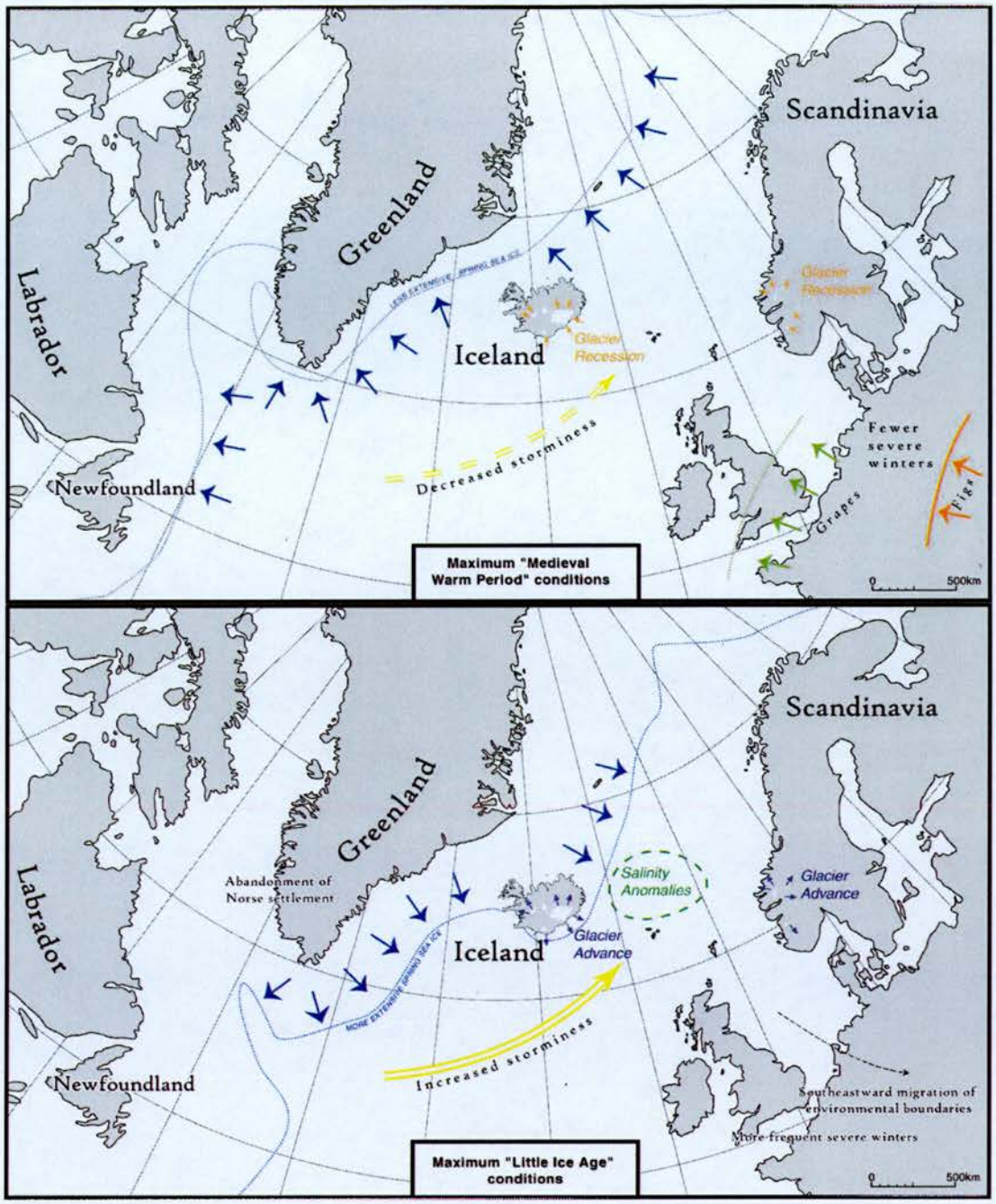

Figure 1.13: Conceptual diagram of favourable (top) and unfavourable (bottom) conditions during the Medieval Warm Period and Little Ice Age. For a seafaring people such as the Norse, reduced frequency and severity of storms and sea ice would be important factors in helping to establish and maintain travel and trade routes.

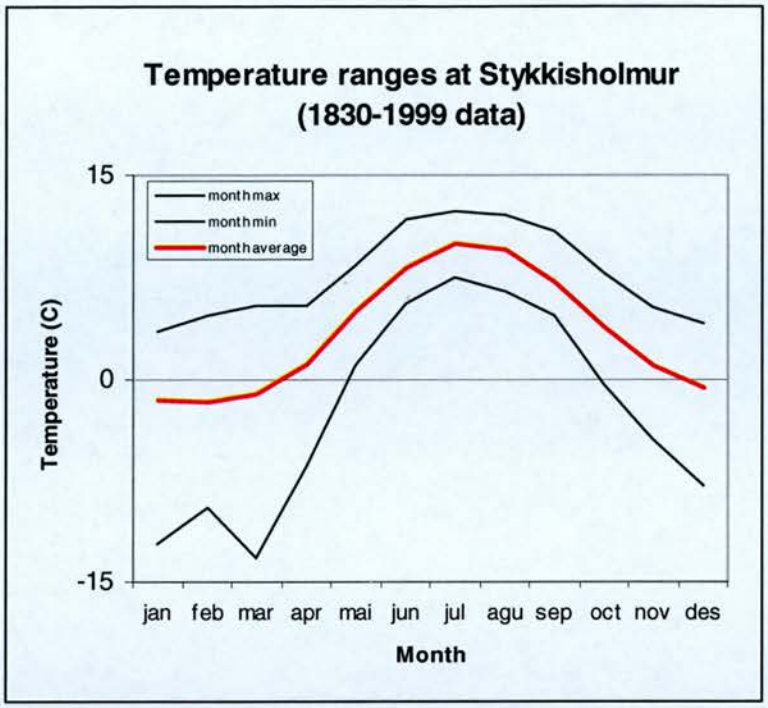

Figure 1.14: Mean monthly temperatures at Stykishólmur, Iceland, and extreme val ues, indicating the scale of interannual variability throughout the year. Variability is notably greater in winter months, with greater extremes of cold and warmth.

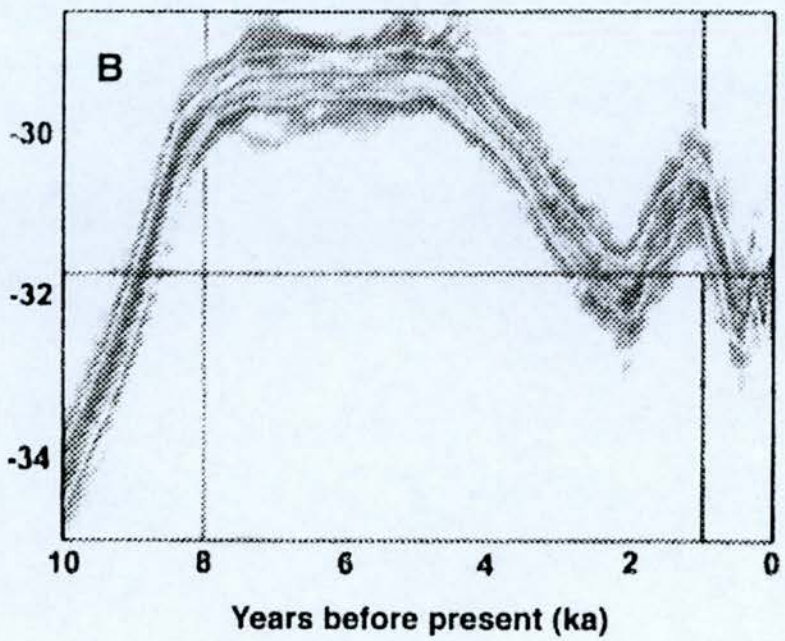

Figure 1.15: Temperature variation during the Holocene, as measured directly in a Greenland borehole (GRIP). Although temporal resolution rapidly decreases down the borehole (to the left), the plot indicates the mid-Holocene climatic optimum around 7000 years B.P., and warmer temperatures can be seen around 1000 years B.P., prior to the cooler conditions within the last 700 years (Dahl-Jensen *et al.*, 1998).

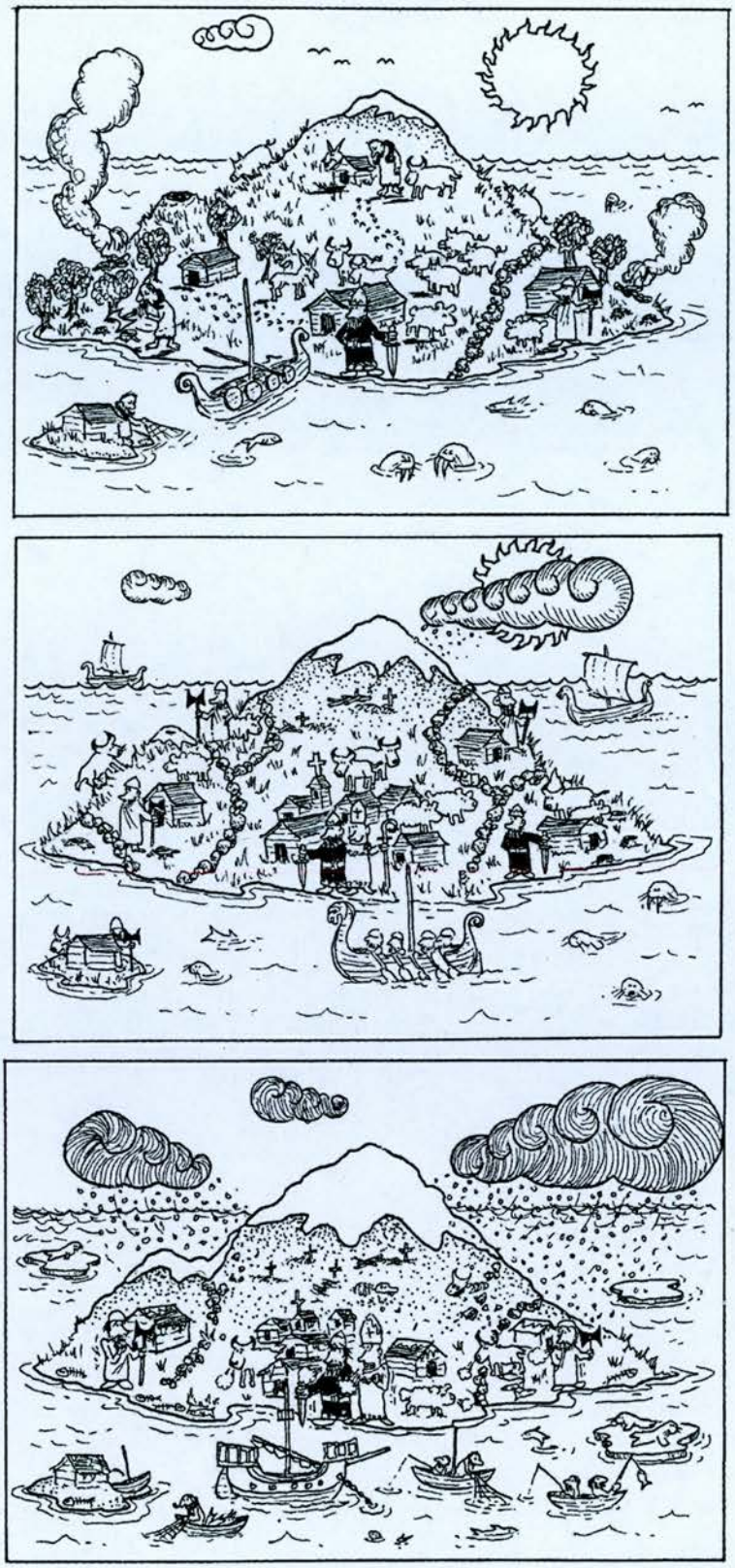

Figure 1.16: A cartoon of life in Iceland from the time of Landnám (*c.* 870-1000A.D.), through to the end of the Viking Age (*c.*1250A.D.), and into the start of the Little Ice Age, *c.*1350-1550A.D. (Vésteinsson, 2000). Key environmental points are the gradually lowering snowline and worsening weather, the reduction in grassland, and the arrival of sea ice. Additional points are the abandonment of upland shielings and farms, the hunting to extinction of walrus, intensification of the fishing economy the transition to tenant farms, the shift to Christianity and appointment of a bishop at the main farm.

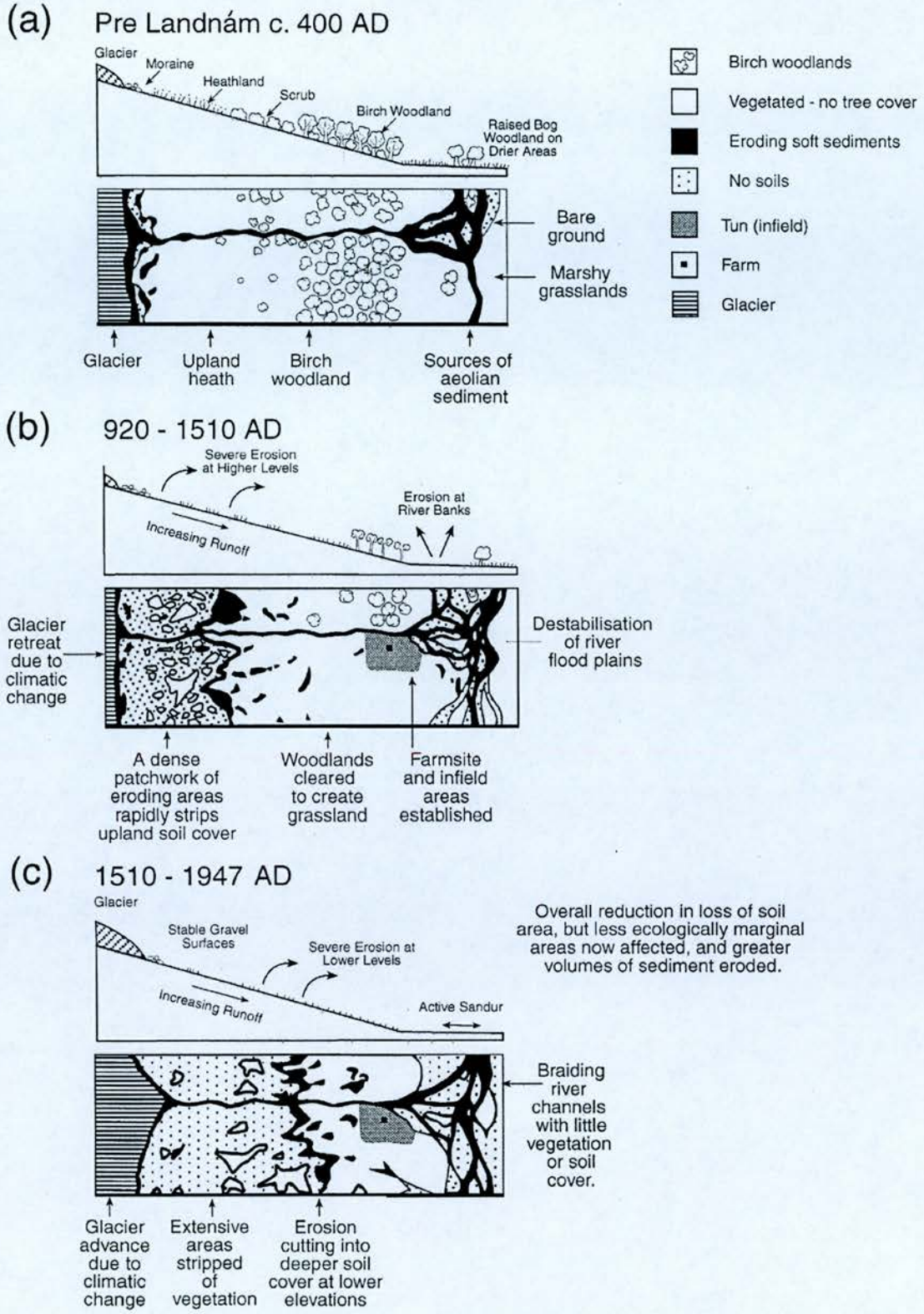

Figure 1.17: Model of soil erosion in southern Iceland (Simpson *et al.*, 2001) through the Late Holocene. Destabilisation of the uplands post-Landnám leads to severe erosion, and eventually to the present-day landscape in southern Iceland, which is one of erosional fronts cutting into and removing deep soils at low levels (Figure 1., while the uplands are degraded.

Figure 1.18: An 'ovigenic landscape'? Rofabarðs, sheep and land degradation at Hamragarðar, south Iceland.

Figure 1.19: Denuded landscape around Sveigakot, near Mývatn. Little of the pre-existing soil cover is left at a location that once supported a farm.

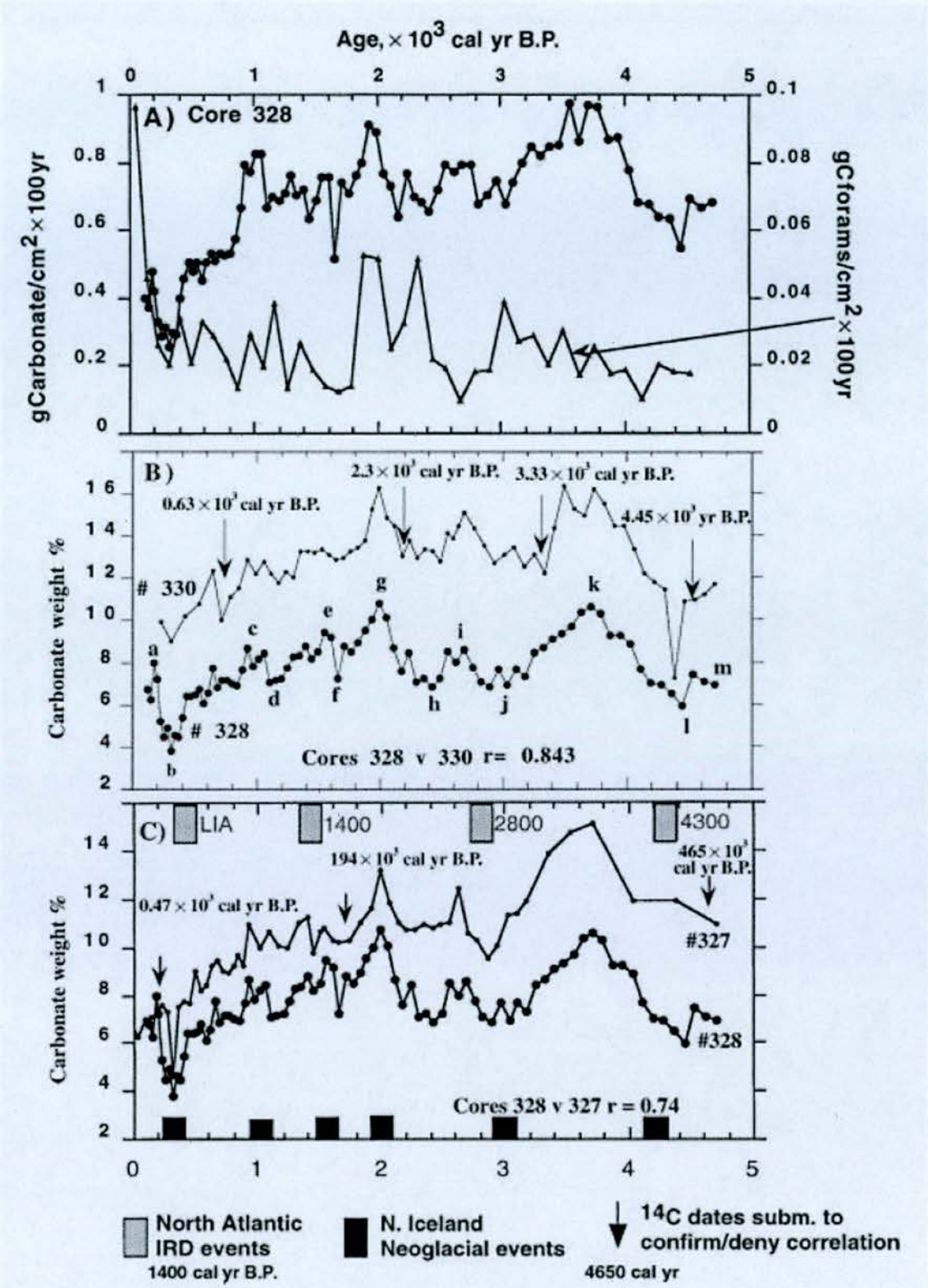

Figure 1.20: Records of carbonate flux from marine sediment cores B997-328 and B997-330 from the Húnaflói bay off the north coast of Iceland (Andrews *et al.*, 2001). Letters in the middle panel reflect successive events of higher and lower carbonate. Neoglaciation in the form of IRD and Icelandic glacial events is also shown. Carbonate flux is a proxy for productivity: higher productivity occurs when warmer ocean wasters are dominant in the region.

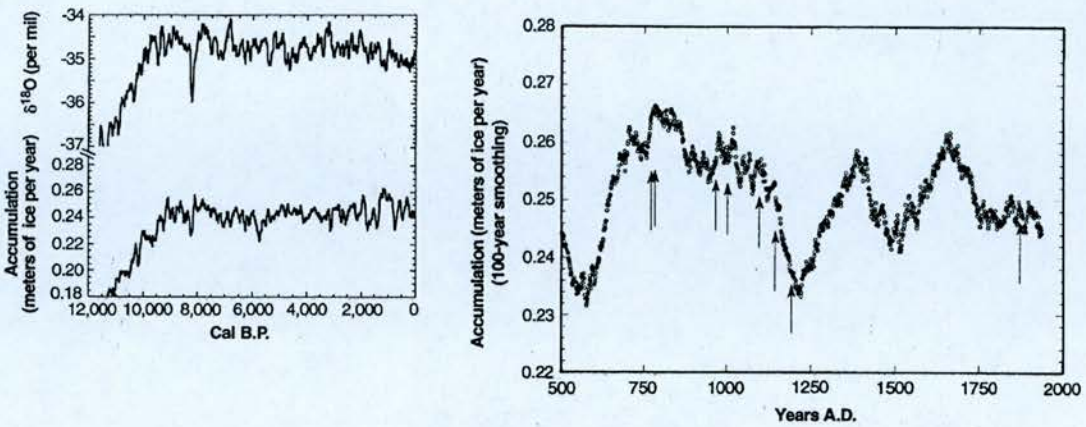

Figure 1.21: GISP2 snow accumulation and $\delta^{18}O$ for the Holocene (left) and 100 year smoothed snow accumulation from 500-1950A.D. (right), from Meese *et al.* (1994). The large peak in snow accumulation during the Medieval Warm Period is visible in both panels.

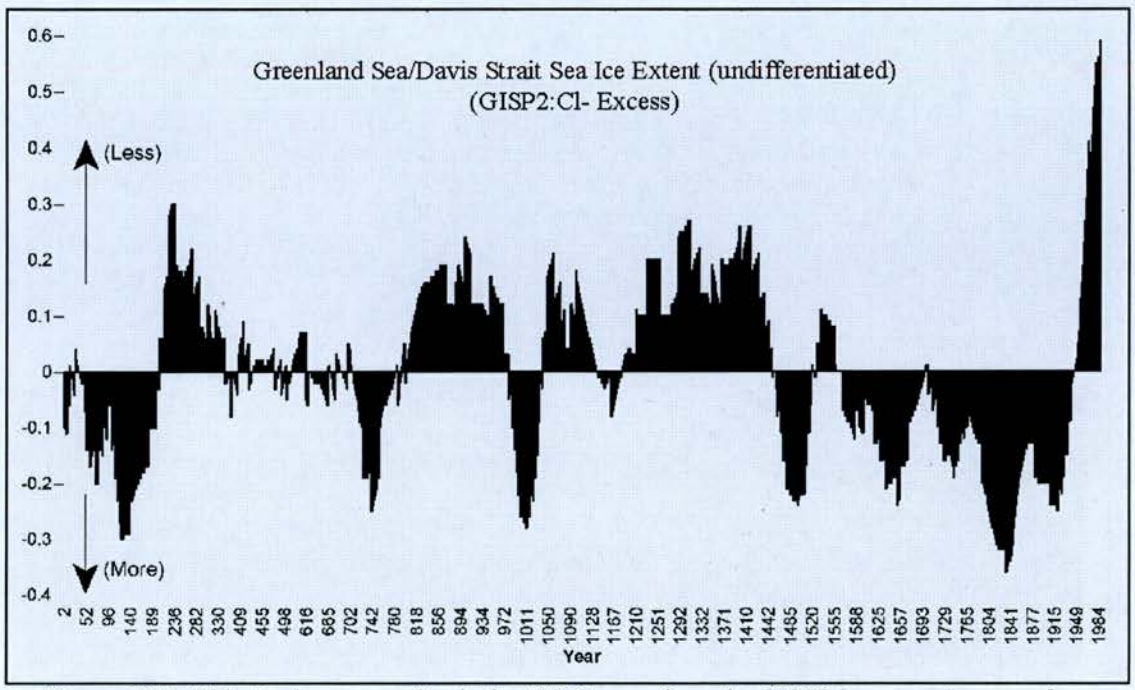

Figure 1.22: Chlorine ion excess for the last 2000 years from the GISP2 ice core. Lower values indicate more extensive sea ice (Dawson, pers. comm.). The LIA is a notable negative (more ice) excursion, while the MWP is largely positive.

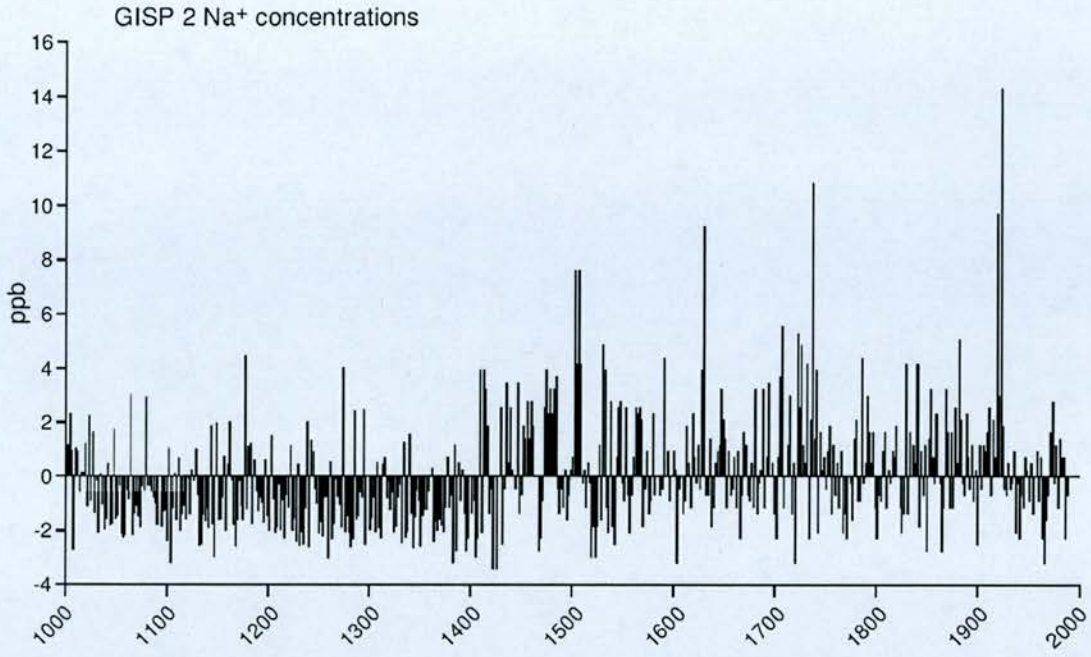

Figure 1.23: GISP2 sodium ion concentration, a proxy for North Atlantic storminess (Dawson *et al.*,2003). There is a marked increase in storminess in the early 15th century, with the individual extreme years increasing in severity towards the early 20th century.

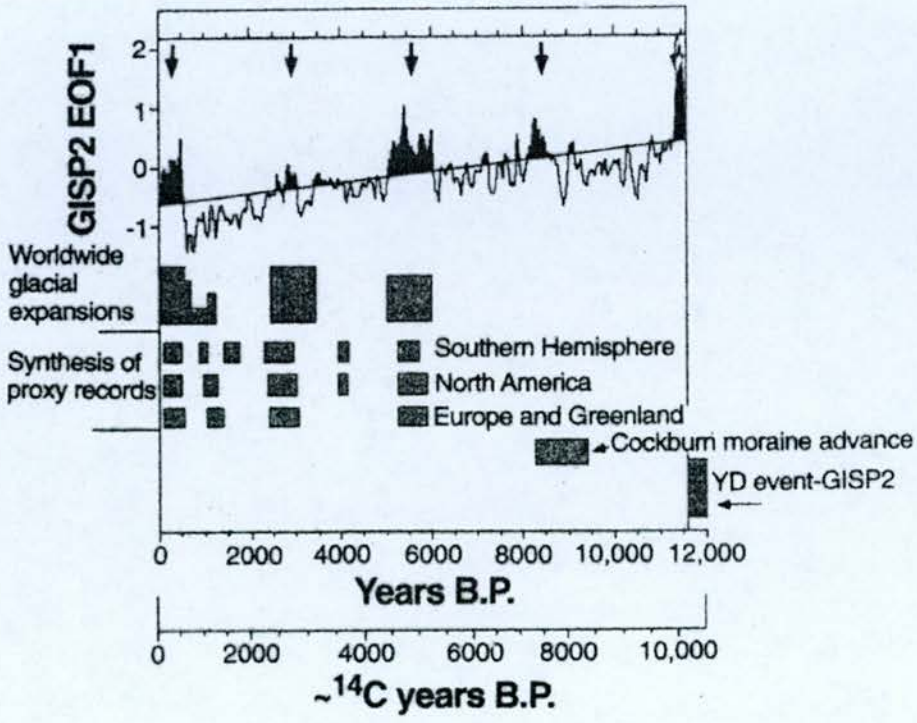

Figure 1.24: The GISP2 EOF1 record and correlations with climatic events (grey bars are proxy records of colder events). Of particular note is the sharp transition around the 14th century, at the MWP/LIA boundary. From O'Brien *et al.* (1995). During positive (black) excursions, more ions and dust were deposited on the ice sheet, suggesting changes in atmospheric circulation and colder temperatures.

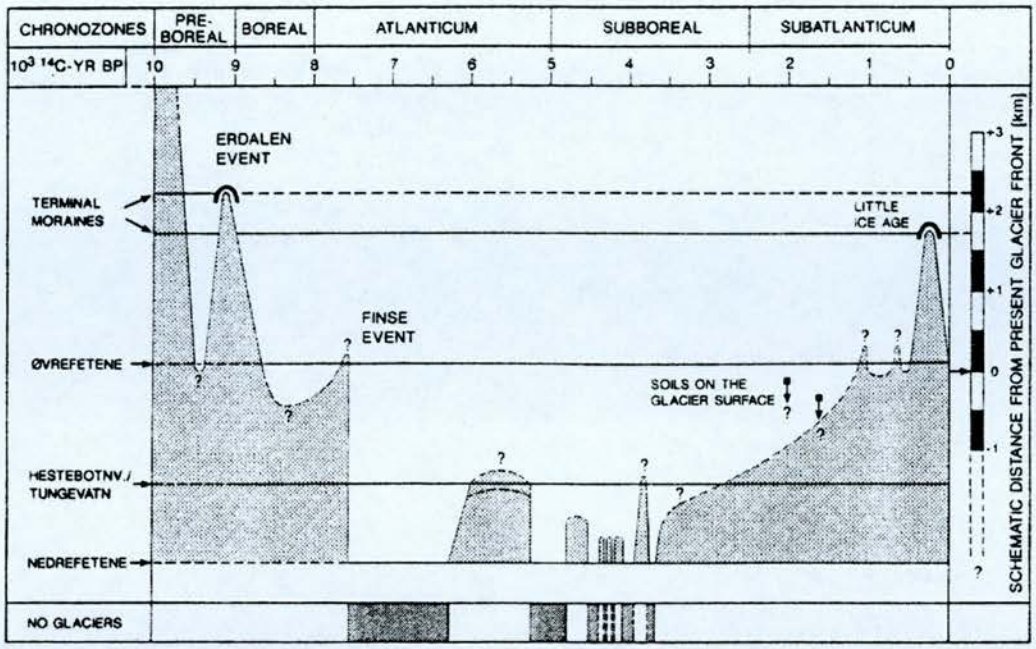

Figure 1.25: Hardangerjøkulen glacier fluctuations during the Holocene reconstructed from multi-proxy evidence (Dahl and Nesje, 1994).

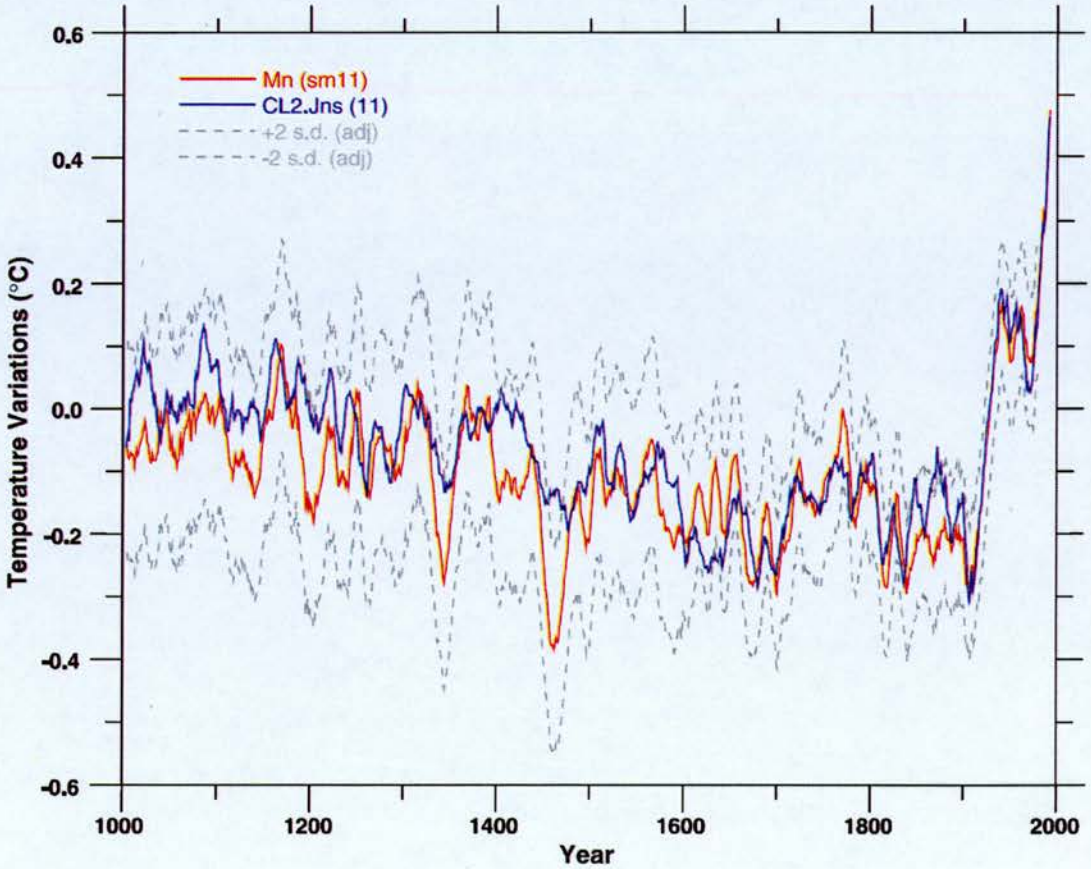

Figure 1.26: Crowley *et al.* (2000) summary of Northern Hemisphere temperature, based on Mann *et al.* (1999) and Crowley and Lowery (2000). The scale of Hemispheric change is relatively small, though the general pattern of MWP to LIA and 20th Century warming is visible.

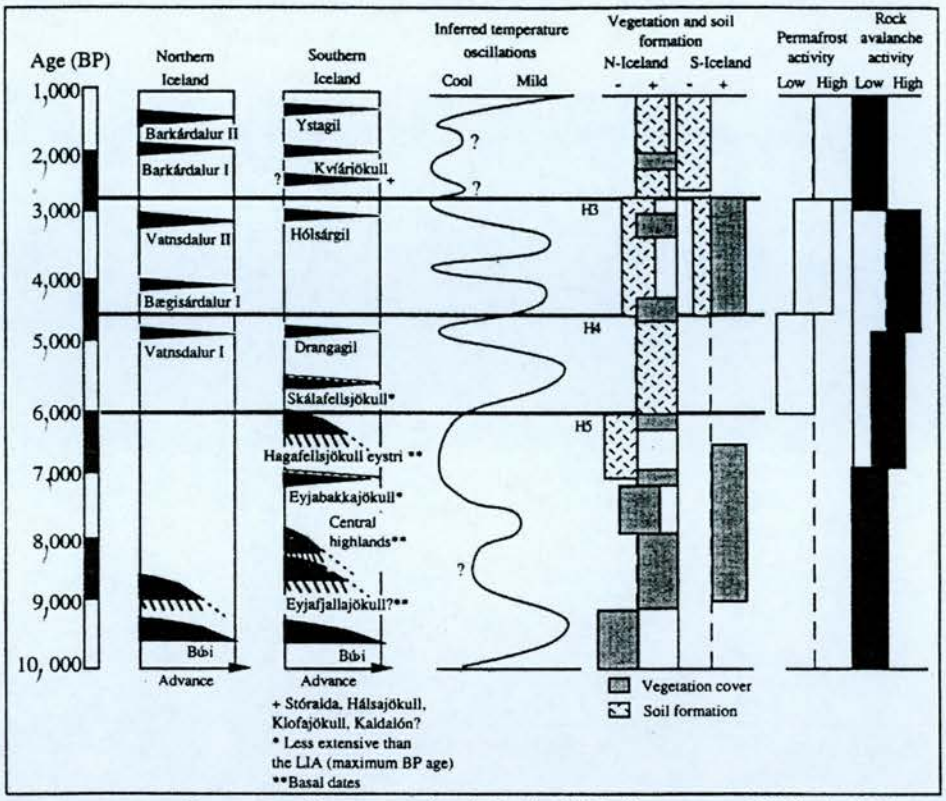

Figure 1.27: Summary of pre-LIA climate proxies in Iceland (Guðmunsson, 1997), including some of the notable glacier advances.

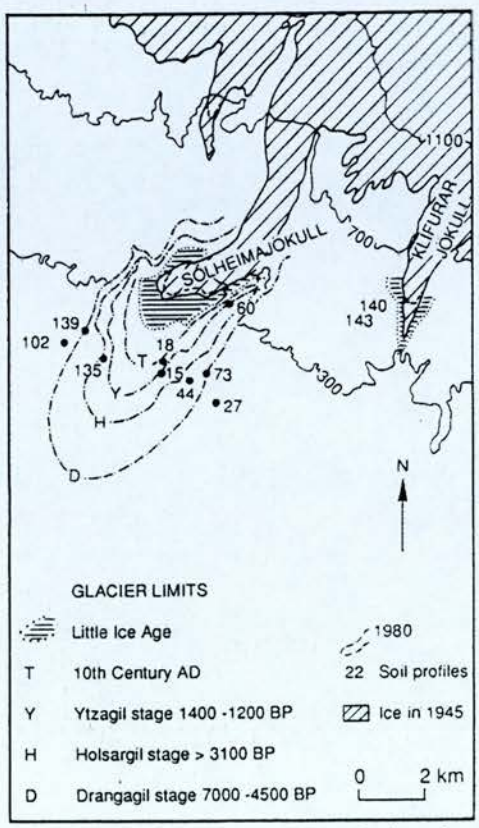

Figure 1.28: Sólheimajökull ice margins relating to pre-LIA glacier advances (Dugmore, 1989).

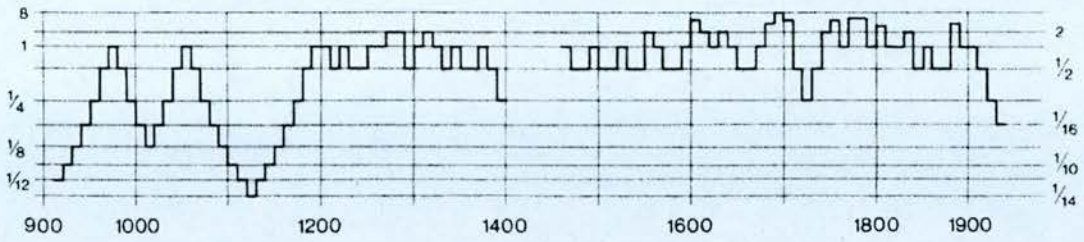

Figure 1.29: Sea ice index generated by Bergþórsson (1969). Earlier years are unreliable due to the lack of data points.

Chapter II: Approach and Methods

1 Overall project approach
2 Datasets
3 Empirical data collection: Catchment geomorphology and environmental change
4 Modelling approach
5 Field site selection
6 Data collection methods

II.1: Overall project approach

At the end of Chapter I, the central research questions arising within this project are outlined. At present, little has been done to evaluate the spatial and temporal patterns of both climatic change and the impacts of change on Icelandic society during the Medieval period. In the majority of cases, climate has either been simplistically seen as the backdrop to a historical and archaeological story, or been seen as the driver of glacier fluctuations, and the environmental side of the historical story has not been integrated with the glaciological story. These connections, and their consequences, are what the project aims to explore through a multidisciplinary approach.

The principal approach in this thesis is to combine both empirical and model data. It would have been feasible to tackle some of the main thesis aims through the application of either a purely modelling approach, such as that applied by Mackintosh (2000), or by a purely geomorphological and sedimentological approach (e.g. Dugmore, 1987). Clearly, empirical data has the benefit of a stronger connection to actual past events, but has the primary weakness of both spatial and temporal censorship of the record (e.g. Kirkbride and Brazier, 1998). There are some datasets that are complete temporally, but spatially limited, such as those from continuously accumulating sediments. By contrast, although data from environmental models may be spatially and temporally complete, it is frequently constrained by both the quality of the input data and by the assumptions and calibrations used within the model structure. Consequently there is great value in combining these two approaches to maximise the strengths of both approaches. Hence this thesis will use selected

empirical data to test models of environmental change in order to explore Medieval climate and environment in Iceland.

This chapter discusses the reasons behind the use of the different data sources (both empirical and modelling), leading to an examination of appropriate sites within Iceland that can be used for the project. It provides a background to Chapters III and IV, where the empirical and model data are presented.

II.2 Available datasets

There are three principal datasets available to explore the major questions stated in I.5. These are:

(i) new empirical data.

(ii) pre-existing empirical, documentary and model data.

(iii) new data from environmental modelling.

New empirical data

The first dataset available is through the collection of original empirical data, gathered from suitable locations in Iceland, which can provide crucial new information to apply to the problem. Despite the limitations (discussed in Chapter I.4) involved in the investigation of Medieval climate change, there are several potential sources of new data: data from glacier fluctuations, data from lacustrine sediments, and data from other terrestrial sedimentary sequences. Terrestrially-gathered empirical data relating to climatic change in Iceland has many of the same strengths and limitations as that gathered in other Arctic locations, but with one notable additional strength: the extra chronostratigraphic tool of tephrochronology (discussed in II.6.3).

In Iceland, nobody has attempted to use lacustrine sediments as a tool for investigating Holocene glacier retreat. This is in contrast to the extensive work that has been carried out in Scandinavia (Karlén, 1976, 1981; Snowball, 1993; Dahl and Nesje, 1994; Nesje *et al.*, 1994; Snowball and Sandgren, 1996; Nesje *et al.*, 2001; Snowball *et al.*, 2002), which has led to a number of useful Holocene environmental records (e.g. Figure 1.23). There is an opportunity to locate and core suitable lakes in Iceland.

The second potentially good terrestrial record of Holocene environmental change is the geomorphological record left by fluctuations of glaciers. This project aims to combine both lacustrine sediment records and glaciogeomorphic records to investigate the fluctuations of small glaciers. In particular, the question of glacier retreat is one that is not so easily answered through geomorphology alone (due to there frequently being an 'absence of evidence'), and so extra lines of enquiry

afforded by lacustrine sediment studies and glacier modelling allow reasons for any 'absence of evidence' to be tested.

In Iceland, a third terrestrial record is contained in the accumulation of aeolian sediment as Icelandic 'soil'. Where there are sufficient tephra layers contained within these aeolian sediment sequences, they provide an excellent stratigraphy for further examination of environmental changes (e.g. Thorarinsson, 1944; Dugmore, 1987; 1989; Dugmore and Erskine, 1994; Larsen *et al.*, 1999; 2001; Ólafsdóttir and Guðmundsson, 2002).

Pre-existing empirical data

Pre-existing empirical data is a valuable resource to add to new data, and allows a fuller exploration of the central research questions. This data is often incomplete and patchy, depending on data type and the focus of previous research, but when it is combined with other data sources it allows a broadening of the project scope in time and space. In certain regions of Iceland, there is a substantial archive of available relevant material; in others there has been little work done.

As shown in Chapter I, some regions of Iceland have been key to the present level of understanding about climate and Medieval environment. Two of these, the Eyjafjallsveit and Mývatnssveit regions in southern and northern Iceland respectively, have a wealth of existing environmental data and are the focus of ongoing archaeological research into their environmental and cultural contexts around the Landnám period (McGovern *et al.*, 2005). Existing data from these two regions is readily available for use.

A useful resource has been constructed by Veðurstófa Íslands: they have made monthly meteorological data available from a series of weather stations around Iceland, and have constructed maps of monthly mean temperature from this data which are also available to use.

Modelling data

The third dataset is that available from numerical modelling both of glacier mass balance and of glaciers, which can be used to explore the sensitivity of these glaciers and landscapes to climatic change. The aim is to understand potential impacts of climatic change on human settlement. This means understanding ecological change over the historical period, but empirical datasets over the last 1200 years are

dominantly affected by human impact – reducing their utility to constrain models. The solution is to use glacier models to explore climatic change and then transfer the glaciological data generated (mass balance, temperature and precipitation) into ecological data (growing seasons, vegetation viability). Crucially, glacier models can be used to rigorously explore spatial patterns and this can be used to effectively assess the impact of settlement. The key question is the parameterisation of the glacier mass balance, discussed below. The modelling will be done at two spatial scales. The selected glaciers will be modelled at the catchment scale in order to examine the changes in temperature and precipitation required to cause significant changes in the glaciers (for example their complete removal). Modelling will also be carried out at the regional scale to investigate wider questions of the sensitivity of glaciers and small ice caps to climatic change. The same regional model will also be used to investigate the impact of these climatic changes on settlement and subsistence in Iceland through parameters such as growing season length, snowcover on grazing lands, potential extent of various vegetation types, and the carrying capacity of individual farmholdings.

II.3 New empirical data collection: Catchment geomorphology and environmental change

As discussed previously, there are three sources of evidence available within an Icelandic glacierised catchment to investigate past environmental change. The glaciers themselves frequently leave clear (if incomplete) evidence of their former extent. Lacustrine sediments, if present, may contain a continuous record of environmental change throughout the history of that lake. In addition, sediment traps and other sedimentary accumulations elsewhere within the catchment provide further environmental information.

Glacial geomorphology

Glaciers have been used for many years to investigate past changes in climate. They are among the most clearly visible barometers of climatic changes in a landscape, and they are powerful agents of change in a landscape. Advance and retreat of glacier margins have been significant events for people living nearby to glaciers. For example in Norway, historical records show that glacial advances were affecting mountain pastureland in southern Norway by AD1684 (Eide, 1955). Changes in glacier extent can be traced in time with local observations, such as those in Iceland (Thorarinsson, 1943), or through the evidence left by previously more extensive ice. Erosional and depositional features that are left on a glacier foreland provide indications of the former greater extent of most glaciers – if the age of these features can be determined, then this provides an insight into the history of that glacier's fluctuations.

Glaciers are sensitive to climatic change, as manifest by temperature and/or precipitation change, on a decadal to century scale, with larger glaciers responding more slowly to change (e.g. Paterson, 1994). A glacier will change in relation to climate through an alteration to the mass balance (Figure 2.1). Mass balance on a glacier is the combination of the inputs and outputs at any point on the glacier surface, so, for example, positive mass balance areas experience a greater mass input (through snowfall and ice flow) than output (through ablation and ice flow). If ice flow is reasonably constant, accumulation through snowfall and ablation through melting are the dominant processes. The line on a glacier that experiences zero net gain or loss of

snow each year is the equilibrium line, which divides the accumulation area from the ablation area, and is the snowline at the end of the summer melt season on most glaciers. The concepts of mass balance and equilibrium line altitude (ELA) are crucial in connecting glaciers to climatic change. A map of Icelandic equilibrium line altitude was generated by Ahlmann and Thorarinsson (1937) using known glacier ELAs and estimating contours, and to date this is the only such map for the whole of Iceland (Figure 2.2).

The sensitivity of a glacier to climatic change will depend on a variety of factors, including the topographic setting, hypsometry, and pre-existing climate. The type of the glacier, determined by the topographic setting, goes a long way towards determining its relative sensitivity to mass balance change (Table 2.1). Large changes in glacier area with height (as seen in the glacier hypsometry) can also affect the sensitivity of the glacier (e.g. Furbish and Andrews, 1984). This is increasingly an issue in southern Iceland, where glacier equilibrium lines are retreating towards the plateaux of ice cap domes (Mackintosh, 2000; Casely, 2001), triggering rapid retreat as increasingly large ice areas enter the ablation area. The length of a valley glacier is more sensitive to climatic change than a glacier ending in a piedmont lobe, in that it is more likely that a mass balance change will be reflected as a progressive shift in the position of the glacier margin (Figure 2.3). The sensitivity level of corrie glaciers and small ice caps to the ELA rising above mountain summits means that they have the potential to provide useful information about ELA change (Figure 2.4).

It is relatively straightforward to investigate the magnitude and timing of glacier *advances* during this period (e.g. Matthews and Karlén, 1992; Dahl and Nesje, 1994; Nesje *et al.*, 1994; Guðmunsson, 1997; Stotter *et al.*, 1999), provided the chronology can be established. In the case of advances during the Little Ice Age occurring close to settlements, advances have been recorded as significant historical events, associated with severe climate and, rarely, direct destruction of property (Thorarinsson, 1943; Eide, 1955; Karlén, 1988; Nesje *et al.*, 1991). In front of many glaciers, advances are recorded by the deposition of terminal and lateral moraines, indicating past advance or stillstand. Retreats are rarely recorded, either in documentary sources or in the physical record, because subsequent glacier advances destroy the record of retreat. There are a few rare exceptions to this, for example the work of Holzhauser (1984; 1997) at Grosser Aletsch Glacier in the Alps, and

Thórarinsson (1966) and Sharp and Dugmore (1985) in Iceland. Such cases require fortuitous preservation of datable material beneath the subsequent glacier advance.

The effect of such censorship of the glaciogeomorphic record is a rapid reduction in the resolution of the record with increasing age (Kirkbride and Brazier, 1998). Only the largest older advances are visible, and smaller advances or stillstands as well as any evidence of retreat have been erased. Consequently a glacier will always provide an incomplete record of previous position, despite its sensitivity.

One of the most valuable attributes of a glacier's response is its independence from anthropogenic influencing factors, something that marks the glacial response out from many other palaeoenvironmental proxies. With the exception of anthropogenic impacts on recent climatic change (IPCC, 2001), temperature and precipitation have been largely independent of anthropogenic influence (though there is a case for a longer-term impact proposed by Ruddiman and Thomson, 2001; Ruddiman, 2003). A glacial record has a significant advantage over palaeoecological proxies (pollen, diatoms, coleoptera, chironomids) in this respect, which frequently have a significant, and sometimes dominant anthropogenically-derived signal within them (e.g. Lowe and Walker, 1997). In regions with a significant human presence, the arrival of people is often a large (if not the largest) event within the record. For example in Iceland, pollen diagrams from southwest Iceland show the deforestation of the region as a large drop in *Betula* pollen (Haraldsson, 1981; Hallsdóttir, 1987) along with the introduction of species not previously present, such as *Plantago lanceolata*, introduced at the arrival of people to area. Similar effects can be seen in the coleoptera record (Buckland *et al.*, 1994). In relation to this project, it is clear that glaciers are a particularly useful tool for determining environmental changes in Iceland that are purely climatic in nature.

It is worth noting that glaciers may not always be responding directly to changes in climate. A significant number of glaciers in Iceland show evidence of surging behaviour, which is unrelated to climatic change. Even if these glaciers show some evidence of a climatic response, it is not possible to disentangle these responses from surging behaviour. Surging has been observed at outlets of Vatnajökull, Mýrdalsjökull and Drangajökull. Surging rarely occurs on small corrie and valley glaciers, and is generally restricted to large glaciers with a high mass throughput. In certain parts of Iceland, there is increased basal ablation caused by geothermal heat flow near volcanic centres and the Mid-Atlantic Rift system. Provided this does not

change dramatically, the glacier response may still be largely a climatic one, but it is preferable to select glaciers farther from such centres of geothermal heat. In cases of volcanic eruptions, large quantities of ice can be melted, and this will have a significant short-term effect on the glacier's mass balance. Tephra deposition on a glacier surface has been observed to affect mass balance by inhibiting ablation (Kirkbride and Dugmore, 2003), but this is likely to be an infrequent and short-lived (~7 years) occurrence. A glacier's response may also be out of phase with climatic changes if the response is related to increased snow accumulation through drifting or through increased precipitation: therefore the glacier is responding to climate, but more extensive terminal and lateral moraines are not necessarily reflecting a 'colder' climate.

Proglacial and lacustrine sediments

The censorship of glacier retreats and smaller advances is one of the key limitations of the glaciogeomorphological record. However, the influence of glacier ice in a catchment extends far beyond the ice itself. One of the strongest impacts is on the sediment load carried by proglacial rivers. The erosive effect of glaciers results in significant changes to the fluvial sediment load, which is reflected in the material deposited in proglacial lakes. Fine material may remain in suspension until it settles in still water (a lake or the sea), while coarser material is transported shorter distances from the ice margin. Landforms created by this coarser material are manifest in geomorphic features such as alluvial fans and river terraces, and in sediment deposited in lake basins (see chapter II.6). Alluvial fans and river terraces may show evidence of aggradation and deposition of coarser material during glacier advances, and incision during periods of glacier retreat. In addition, lacustrine sediments may hold other information from the catchment, including diatoms from fluvial and lacustrine environments, pollen from terrestrial sources and chironomids. Many other inputs into lacustrine sediments are visible in Figure 2.5 (Birks *et al.*, 2000).

Sediment cores obtained from lakes located beyond the snout of the glacier provide the opportunity to assess the magnitude of glacier retreat. This has been demonstrated in Scandinavia by several research groups (Karlén, 1976, 1981; Nesje *et al.*, 1991; 1994; Snowball and Sandgren, 1996; Snowball, 1996; 2001; Snowball *et al.*, 2002; Dahl *et al.*, 2003; Figure 1.23). In favourable locations, a lake will collect meltwater from a glacier that is prone to disappearing during warmer episodes in the

Holocene, and in such circumstances, the sedimentological change will be distinct (e.g. Matthews and Karlén, 1992; Figure 2.6). Data from such lakes can provide considerable insight into not only the timing but also the magnitude of any milder episodes during the Holocene. This data has fewer temporal limitations, provided there is continuous sediment accumulation within the lake.

Geomorphic change elsewhere in a catchment

Glaciers are not the only components of a catchment that will respond to environmental (including climatic) changes. Several factors in Iceland are of particular relevance: vegetation stability, slope stability and aeolian sediment accumulation.

One of the greatest challenges for catchment geomorphology is to date events within a catchment. In Iceland, the frequent volcanic eruptions and associated tephra layers, combined with the rapid accumulation of aeolian sediment provide valuable dating tools for examining the timing of events within a catchment, particularly those that are either buried by or recorded within soils. This fortuitous combination of factors allows many features within a catchment to be assigned minimum ages, and occasionally have their age bracketed by the dates provided by two or more tephras.

The stability of the vegetation cover is an important topic in Iceland, as the effects of large-scale soil erosion both in the Icelandic highlands and in lowland areas have a big impact on the Icelandic farming community (Ólafsdóttir and Júlíusson, 2000; Ólafsdóttir, 2001). In a geomorphic context, breaking of the vegetation cover may lead to the partial or complete erosion of the entire aeolian sediment sequence. Such changes are often readily visible within suitable aeolian sediment sequences. In the Icelandic highlands, the margin of viable vegetation changes depending on the prevailing climate, consequently exposing land to erosion during cooler or drier periods, and encouraging stabilisation of the land.

The geology of large areas of Iceland comprises of interbedded basaltic lava flows and palaeosols. This combination of strong (though jointed) rock and weaker soil layers produces good conditions for large landslide events (Bentley and Dugmore, 1998). These can be significant in the context of a catchment – altering catchment drainage or sediment input.

A great deal of additional information is available through the study of Icelandic 'soils' (aeolian sediment sequences). With large amounts of fine material available

for transport, due to a combination of frequent volcanic eruptions, glacial erosion, and large areas of unvegetated land, sediment accumulation in most areas of Iceland is rapid (frequently >0.1mm/a: Casely and Dugmore, 2004). In association with deposition of tephra markers providing chronological control, these sediment sequences can record changes in sediment deposition (for example in the rate or type of material deposited).

The approach used in this thesis is to investigate all of these factors in order to assess the sensitivity of catchments to environmental change, as well as the likely impact of catchment geomorphological changes upon lacustrine sediment deposition. This will provide the best opportunity to evaluate the extent and sigificance of environmental changes during the Medieval Period.

II.4: Modelling approach: Spatial/temporal patterns of mass balance, snowcover, and growng season

A variety of different types of model have been used to investigate glacier fluctuations, environmental and climatic changes. It has been argued that the word 'model' itself has been used so often and so loosely as to lose meaning (Haines-Young and Petch, 1986). This accusation necessitates a clarification of the nature of the model and of the model's application to the problem. Models may have direct practical purpose, to provide reconstructions or predictions of events, or they may have a philosophical purpose, in which they are designed to improve our understanding of the operation of environmental systems through a simplified representation of that system (Beck *et al.*, 1993). The modelling in this thesis is numerical modelling of the second type, in that it is not designed as an exact reconstruction of all the physical interactions and feedback within a glacial or climatic system, but uses suitable analogues and empirical relationships where appropriate alongside physical relationships. The consequence of such modelling, whether heavily dependent upon physics or upon analogue relationships, is that predictions of specific outcomes are not necessarily possible, and not necessarily desired, provided the model improves our understanding of the operation of the environmental system, identifying thresholds, sensitivities and patterns of change.

Numerical glacier modelling may be of two major types – ice flow models attempt to replicate the characteristics of ice flow over a topography, and mass balance models that attempt to characterise the nature of accumulation and ablation of a glacier. Integrated mass balance/ice flow models combine the two. Numerical models attempt to explain different characteristics of a glacier through the solution of equations (physical or empirical). The model may be designed to simulate a glacier system (usually with a high level of complexity), or it may be designed for sensitivity tests upon the system.

There have been two critical developments within the last 40 years in relation to climate and glacier models. The ongoing development of computers and rapid increase in computing power has allowed a great increase in the complexity of glacier models. There has been a transition from one- and two-dimensional models to complex three-dimensional models incorporating many different physical characteristics of a glacier, such as ice flow, basal sliding and longitudinal stress

(Harbor, 1993), and a move towards modelling of more complex problems such as erosion, debris cover and calving. The second useful development is that of GIS (Geographic Information Systems). GIS are not used for the ice sheet modelling process itself, due to the present lack of sufficient modelling tools within the GIS structure. However, as the GIS raster architecture is comptible with cell-based numerical ice sheet models, and this allows for the interpretation of numerical model information in a GIS.

The approach here is to attempt to model glacier systems as they respond to climate. However, the primary challenge, as it will become apparent is to provide good models of the mass balance.

Mass Balance Models

Mass balance modelling involves the quantification of snow accumulation and ice melting (ablation) in the glacier system, usually over the period of a balance year, or over a run of many years. The difference between accumulation and ablation at any point defines the mass balance at that point (e.g. Figure 2.1; 2.4). Positive mass balance indicates more accumulation than ablation, and that therefore there will be a net gain in snow or ice at the location. If a site experiences positive mass balance for a single or small number of years, then the result is a semi-permanent patch of snow or firn. Negative mass balance in subsequent years may remove this firn. If a positive mass balance is sustained over many years, the firn patch may grow to the point at which it becomes large enough to deform or slide downhill as a glacier. Similarly sustained periods of negative mass balance can cause an active glacier to shrink to the point at which it ceases to flow and becomes a patch of firn, or disappears entirely.

There are a variety of levels of sophistication to models of accumulation and ablation. Accumulation is largely through snowfall, though in some locations, melting and refreezing of ice, or snow drifting or avalanching can significantly affect the rate of mass accumulation. Ablation of snow and ice can be modelled using an energy balance approach or by degree-day modelling. Energy balance modelling requires the collection of meteorological data for the specific site, as has been done in some mass balance studies in Iceland (Aðalgeirsdottir *et al.*, 2003; Guðmundsson *et al.*, 2003a, b). The meteorological information for a good energy balance model is site-specific, and will vary significantly on a regional basis. An effective alternative

approach to modelling ablation is to use degree-days (Braithwaite, 1995; Kayastha *et al.*, 2003; Aðalgeirsdottir *et al.*, 2003; 2004). The principle of degree-day modelling is that there is a direct relationship between the total number of positive degree-days and the total ablation (e.g. a temperature of +6°C = 6 positive degree-days; -3°C = 0 positive degree-days). The benefit of this approach is that far less data is required to construct the model, and therefore ablation can be modelled over wider areas than a single specific site. The ablation will depend upon the type of material being ablated (ice or snow), and the degree-day correction factor, k, is a parameterisation of this. An estimation of total ablation can therefore be made from temperature data alone.

Ice flow models

Glacier ice will move downhill through the processes of sliding, internal deformation and bed deformation (e.g. Paterson, 1994), depending on a combination of air temperature, ground temperature, and the mass of overlying ice. In temperate and maritime regions such as Iceland, ice at the glacier bed is always at the melting point (pressure melting point), and therefore sliding is by far the most important process. Once a glacier is sliding over its bed, the major processes of process of glacial erosion can operate, producing both fine and coarse-grained material, and this changes the sedimentary output of the proglacial rivers.

Using modern computing power, modelling ice flow can be done at a continental/ice sheet scale to tackle a variety of problems including ice sheet evolution through time, ice streaming and ice sheet erosion rates (Hulton *et al.*, 2002; Sugden *et al.*, 2002; Bingham *et al.*, 2003; Boulton *et al.*, 2003; Jamieson *et al.*, 2005). At a catchment scale, the development and fluctuations of individual glaciers may be examined (Oerlemans, 1992, 1996, 1997; Mackintosh and Dugmore, 2000; Mackintosh *et al.*, 2002) at a variety of levels of complexity depending on the physics incorporated into the model, and the resolution of the model (in 2 or 3 dimensions).

Glacier modelling in Iceland

In Iceland, ice extent around the Mýrdalsjökull ice cap since the LGM in southern Iceland has been modelled by Bingham (1999; Bingham *et al.*, 2003) using a combined mass balance and ice flow model. Using this, Bingham tested the sensitivity of this ice cap to ELA changes, and to evaluate the scale of ELA lowering

required to produce LGM and Younger Dryas ice sheets in the area. Fluctuations of Sólheimajökull during the late Holocene were modelled by Mackintosh (2000; Mackintosh *et al.*, 2002) using a flowline model – ice flow was modelled along a central flowline and integrated with mass balance (using an energy balance model). This model was then calibrated using geomorphic and chronological data of glacier extent from Dugmore (1987). At the scale of an individual glacier, this modelling approach was successful for Mackintosh's reconstruction of the fluctuations of Sólheimajökull. The same model was applied to Tungnakvislajökull, a neighbouring outlet glacier of Mýrdalsjökull, and used to investigate the significance of 'Little Ice Age' fluctuations of that glacier (Casely, 2001). The glaciers in both cases were large valley glaciers fed by accumulation zones of the Mýrdalsjökull ice cap, and flowline modelling was able to successfully capture the main advances of these glaciers in response to temperature changes during the 'Little Ice Age'.

There has been a recent attempt to reconstruct LIA ice volumes and to model the sensitivity of Vatnajökull to future climatic warming using a model coupling mass balance, ice dynamics and hydrology (Adalgeirsdottir *et al.*, 2003; Flowers *et al.*, 2005; Magnusson *et al.*, 2005). This model predicts the loss of up to 25% of ice volume in the next 100 years, and the hydrological component is used to predict changes in runoff from the glacier.

Vegetation modelling in Iceland

Degree-day models have been frequently used in studies on the relationship between vegetation and climate (for example Parry, 1978; 1990). In Iceland, the relationship between positive degree-days and vegetation has been demonstrated by Friðriksson and Sigurðsson (1983) and Bergþórsson (1985). There is a connection between the number of degrees above a given lower threshold and the success of a selected plant species. This approach has been used by Ólafsdóttir (2001; Ólafsdóttir *et al.*, 2001; Ólafsdóttir and Guðmundsson, 2002; Haraldsson and Ólafsdóttir, 2003) to investigate soil development the distribution of vegetation around Iceland during the Holocene. This work connected a simple positive degree-day sum for grass and birch to a temperature record derived from the GRIP $\delta^{18}O$ record to investigate the potential distribution of vaegetation in Iceland for reference to the issue of land degradation.

Modelling approach in this thesis

The questions in this thesis are related to changes in the Icelandic climate, and their relationship to environmental change and settlement changes. In order to explore this, the chosen route is to use glaciers as a starting point. The best measure of a glacier's 'health' in relation to climatic changes is mass balance, and so a model of glacier mass balance has been created for Iceland. This model uses meteorological data to estimate snow accumulation through snowfall and ablation though degree-days in order to estimate the mass balance. It was not possible to generate an energy balance model to cover the diversity of spatial areas covered in the thesis, and so the degree-day approach has been used. Ice flow is relevant at the scale of an individual catchment or glacier, in order to investigate growth and disappearance of the glaciers. For this, a high-resolution mass-balance/ice flow model (GLIMMER) will incorporate the mass balance data from the mass balance model to explore the growth and disappearance of these ice masses.

One of the key novel approaches in this thesis is to transform the outputs from mass balance modelling to explore vegetation changes (e.g. growing season). The same input meteorological data can be applied both to a model of glacier ablation (degree-days above 0°C) and to a model of potential vegetation cover (total number of degree-days). Consequently there is the scope to independently calibrate the vegetation model through the use of glacier mass balance and the detailed spatial data this can generate for the whole island.

II.5 Field Site Selection

The focus of the research is to untangle the twin forces of people and climate on the environmental record. This will help to determine the climatic setting for the period of settlement by the Norse, placing their undoubted anthropogenic impact into a climatic context, and allowing the archaeological record to be understood more clearly. Given the multidisciplinary approach used in this thesis, a regional approach has been used rather than single locations. This allows data from the variety of sources to be considered within each region, and for the modelling studies to cover those regions. New empirical data can be gathered from specific locations within these regions: from sites that have both suitable small glaciers and suitable lakes accumulating sediment on the glacier foreland, or from suitable sedimentary sequences within the area. Additionally, there is existing data for the Mývatnssveit and Eyjafjallsveit regions which can be included.

II.5.1: Selection of lacustrine sediment coring sites

As described above, an important data source is that available from lacustrine sediments that are in a position to receive glacigenic sediments. In particular, it is those sites where the sediment supply may change or be removed upon glacier retreat that are of interest. The lakes themselves must not have been affected by glacier advances during the Little Ice Age in order to have accumulated a continuous record of sedimentary variations during the Late Holocene. Two positions favour such a lacustrine record:

1: *Adjacent lakes*: In rare cases, thickening and advance of a large glacier will overtop a col allowing glacial meltwater to flow down an adjacent unglaciated catchment.

2: *Distal proglacial lakes*: More commonly, small glaciers that exist at the margins of present-day glaciation will be present during colder/wetter periods, and absent during warmer/drier periods. Lakes that exist on the foreland, but beyond the limit of LIA glaciation, provide favourable sites.

Iceland's climate and topography supports a large variety of glacier types, large and small, as well as piedmont lobe and valley types. Most of the ice mass is concentrated in the five major ice caps: Vatnajökull, Mýrdalsjökull, Hofsjökull, Langjökull and Drangajokull, and much of the ice from these drains via piedmont

lobe glaciers. A key problem with many proglacial areas, particularly around the large ice caps, is that periodic volcanic activity (and jokulhlaups) effectively destroys downstream records. There are many small glaciers in montane regions on the north and east of the island. It is in such settings that distal proglacial lakes can provide a valuable environmental record, as they are outwith areas of volcanic activity. No sites were found where there are lakes in valley adjacent to large valley glaciers. This is primarily due to there being relatively few large valley glaciers in Iceland. Although dendritic valley networks exist, they are generally separated from the larger ice masses, and contain small corrie glaciers. The most important criteria are: (1) any of: an ice free mountain catchment, corrie glacier or small, sensitive ice cap lying close to the present ELA; (2) a lake continuously receiving meltwater from this catchment in an undisturbed geomorphic setting. (3) in the Icelandic context having a good tephrochronology – for example parts of northwest Iceland have a relatively poor tephrochronology, making dating more difficult. (4) A location outwith areas of volcanic activity.

II.5.2: Searching for appropriate sites

Finding suitably marginal glaciers

Using these simple criteria, 1:100,000 scale topographic maps were examined in the search for glaciers that could meet these criteria. Many regions in Iceland could easily be eliminated due to unfavourable glacial settings. The icecaps of Vatnajökull, Mýrdalsjökull, Langjökull and Hofsjökull, as well as a number of the smaller icecaps could be eliminated due to their size and relatively poor sensitivity to short-term climate variations, as well as their position over volcanic centres. Potentially favourable regions initially were the Tröllaskagi and adjacent montane regions of northern Iceland which contain many small corrie glaciers. The icecap of Drangajökull and high plateau of Gláma on the northwest peninsula also lay close to the ELA. In central Iceland, the Kerlingarfjöll montane area and nearby Hrutfell has a number of small glaciers. The mountains of eastern Iceland also have potential with some small glaciers, two small icecaps, and mountains lying close to the present glaciation limit.

Finding appropriate lakes

Within these regions, a further search was carried out to isolate catchments where meltwater from a small glacier drains directly into a lake. Catchments were eliminated where there is additional meltwater from less sensitive glaciers or icecaps, or where the meltwater appears to be strongly seasonal or intermittent.

Such a search relies strongly on the accuracy of the maps used, and it is possible that potentially favourable lakes were missed due to inaccuracies on the maps. It is also possible that lakes drawn on the map are unsuitable in reality as they are misplaced on the map. However, from this search, a shortlist of 10 sites with potentially suitable lakes was drawn up based on the map search (Table 2.2; Figure 2.7).

II.5.3: Elimination of unsuitable sites

Immediate elimination of unsuitable sites from the shortlist

Lakes provisionally selected from map evidence were not always favourable, due to several factors. Some lakes were inaccessible within the means available for this project, including Drangajökull and Gláma. They are also far from the main volcanic centres, and so are subject to fewer and smaller tephra falls. The Kerlingarfjöll and Hrutfell sites were rejected as it is unclear if meltwater reaches the lake, due to the presence of lava fields close to the lakes. Marginal lakes at Þrandarjökull were rejected in favour of Hofsvötn at Hofsjökull, due to the greater amount of visible geomorphology on aerial photographs and better accessibility to these lakes from Geithellnadalur.

Lake northeast of Snæfell

An example of a lake that was unsuitable for other reasons is the lake immediately northeast of Snæfell (Figure 2.8). In this case, field inspection showed that the lake is unsuitable for preserving a complete record of the changes in the glacier present in the northeast corrie of Snæfell. The lake receives meltwater from a small glacier on the northern side of Snæfell; however, not all meltwater travels directly to the lake.

In this case, a large delta has prograded into the southwestern arm of a Y-shaped depression, with the catchment exit located at the end of the western arm of the 'Y'. A lake that presently receives glacially-derived sediment exists in the northeastern arm of the 'Y', but receives its sediment from several minor streams crossing the delta from southwest to northeast. The water from the lake presently drains from the

southern end of the 'Y' to drain out of the western arm. It is therefore possible that this lake has at times received no meltwater across the delta (for example if the main stream channel became incised). It is also possible that the delta is a relatively recent feature, filling the lake largely during the LIA. No lichens were visible on the active delta surface, and so the age of elements of the delta surface could not be determined. While it remains possible that the lake has always received glacial meltwater, the variability in sediment supply would render any core extracted to be potentially unreliable. Hence, despite a favourable glaciogeomorphic setting, this lake was not used for the project.

II.5.4 Suitable sites

Skeiðsvatn in Tröllaskagi

Skeiðsvatn proved to be a suitable site for coring, with the right elements present to give the potential of a complete preserved record of glacier changes. Five small glaciers, all with similar ELAs, are present, draining directly into the lake, and will drain into the lake continuously provided there is ice present (Figure 2.9). The geological setting is of Tertiary basalt lava flows with interbedded palaeosols, and there is relatively low geothermal heat flow as the site is far from the areas of present-day volcanism.

The lake Skeiðsvatn is situated in the north-south trending valley Vatnsdalur in the central part of the Tröllaskagi mountains. It is ringed by mountains that are up to 1350m high, and consequently at the southern end of the main valley and in the tributary valley there are five small glaciers presently feeding meltwater into the lake (3 of which are in Figure 2.10). The largest of these is Vatnsdalsjökull. The glaciers were the subject of a study by Stötter (1991): the valley has given its name to two mid-Holocene glacier advances, highlighting the climatic sensitivity of the valley. Due to the dendritic drainage and valley system, and the location of the lake within the main valley, the lake receives all meltwater from the glaciers, and has almost certainly done so throughout the Holocene.

The valley is surrounded by mountain arêtes, and small areas of plateau. Consequently, accumulation through drifting snow is a factor that may need to be considered when evaluating their mass balance. The glacier forelands are largely uncomplicated, although in front of the largest glacier, there are two small proglacial rock glaciers.

Despite more than 20 years of geomorphological and ecological research in this region (including the work of Caseldine, Kugelmann, Häberle, Stötter and Wastl), including the valley, the lake has not previously been cored!

Hraunsvatn in Tröllaskagi

Apart from Skeiðsvatn, the other examples of such localities in Tröllaskagi are the lake Stífluvatn located in Fljót, and Hraunsvatn, located between Hörgárdalur and Öxnadalur. Stífluvatn is the location of a modern reservoir, and has been cored by Boygle (1994). It was not chosen for study as its catchment is large and complex, including four tributary valleys Tungudalur, Móafellsdalur, Hoarfadalur and Klaufabrekknadalur. These tributaries contain a number of small corrie glaciers. Tungudalur itself contains a lake which may be worthy of further investigation: the corries at the head of that valley (below mountains *c.*1000m.a.s.l) may have been glaciated during colder periods.

Hraunsvatn is culturally significant in Iceland, and the setting is impressive, beneath the pinnacles of Hraundrangi. The simple catchment is a southwest-northeast trending valley, also named Vatnsdalur, draining from a very small glacier on the northeastern side of the mountain Bessahlaðaskarð at the southwestern end of the valley. The 'hraun' (Icelandic for 'lava') is in fact a very large landslide that has come from the side of Hraundrangi, left behind the great pinnacles, and dammed the end of Vatnsdalur on its way into Öxnadalur, creating Hraunsvatn (figure 2.11). There is a small rock glacier below the snout of the present-day glacier. Meltwater from this glacier drains directly into Hraunsvatn, and there are no other glaciated catchments draining into the lake. Hraunsvatn is a suitable site for examination of glacier reteat, but the combination of easier access and a proven sensitivity to climate meant the Skeiðsvatn was chosen over Hraunsvatn as the favoured site in Tröllaskagi. It is clearly a site worthy of future investigation.

Hofsvötn, below Hofsjökull í Lón

Hofsvötn is situated at an elevation of *c.*700m on the upland plateau above the valleys of Hofsdalur and Geithellnadalur. There are two lakes, both collecting meltwater from Hofsjökull, that are set within a bedrock hollow and drain into Hofsdalur (Figure 2.12). The meltwater source for this lake is a small ice cap, Hofsjökull, situated to the south of the lake. Meltwater from several streams

originating at the present-day ice cap enter the lake. Hofsjökull is currently small and retreating, and the altitudinal range from the edge of the ice to the summit of the ice cap is small (200m). Several bedrock protrusions are visible within the margins of the ice cap, suggesting that the ice is relatively thin. The geological setting is largely subhorizontal sequences of Tertiary basalts, but includes many dolerite dykes orientated SW-NE, perpendicular to the main valley systems.

II.5.5 Selected regional study areas

As mentioned earlier, a regional approach is used in order to test the sensitivity of different parts of Iceland to environmental change. The search for suitable locations with the potential to provide lacustrine sediments has provided two key locations: Tröllaskagi in northern Iceland and Hofsvötn/Hofsjökull in southeastern Iceland. In addition, there is further available material from Mývatnsveit and Eyjafjallsveit. The combination of these localities provides the three regions that will be used as a base for the modelling studies (Figure 2.13).

1: northern Iceland – including Tröllaskagi and Mývatn. This region is relatively colder and drier than other parts of Iceland, and the Mývatn area is relatively flat lying upland, with important settlement sites located both around the lake and farther inland. Glaciation is limited to the corries in the Tröllaskagi mountains, and includes Vatnsdalur (Figures 2.14 and 2.15).

2: southeastern Iceland – including Hofsjökull (Figures 2.16 and 2.17), Þrandarjökull, Geithellnadalur and Snæfell. This area is one of fjords and deep valleys cutting into a high upland plateau. Farms in the area are and were predominantly on the coast, and there are some sites present farther inland on the valley floors. Good pasture land is present on lower-lying wetlands near Snæfell. Glaciation of the area includes the eastern margin of Vatnajökull, the summit and corries of Snæfell, the plateau ice caps of Hofsjökull and Þrandarjökull, and Jökulgilsjökull in Flugustaðadalur.

3: southern Iceland – including the ice caps of Mýrdalsjökull and Eyjafjallajökull, and extending as far north as Hekla. An important historical region, with a rich story of settlement and farm abandonment. Settlements are located on the coastal lowlands and farther inland along the Markarfljót valley towards Thórsmörk. Glaciation of the area is dominated by Mýrdalsjökull and Eyjafjallajökull, and there are smaller glaciers on Tindfjallajökull and Torfajökull.

II.6: Data collection methods

Presented below are the methods used in the collection of new empirical data. These include the collection and analysis of lacustrine sediments, geomorphological mapping, aeolian sediment analysis and tephrochronology.

II.6.1: Lacustrine sediments

Lacustrine sediments are an excellent source of palaeoenvironmental information and are widely used in Arctic environments to investigate environmental change (e.g. Karlén, 1976; Karlén, 1981; Kaplan *et al.*, 2002; Dahl *et al.*, 2003). They are sensitive ecosystems to both natural and anthropogenic changes (Karst-Riddoch, 2003) Lakes provide stable sediment traps which allow continuous accumulation of material within them. This material includes minerogenic and organic sediment from the catchment, pollen, diatoms and chironomids, which comes from both proximal and distal sources, and can all accumulate in the lake (Figure 2.4). Minerogenic and organic material can be washed into the lake from elsewhere in the catchment via the inflow stream. Pollen may be sourced from vegetation close to the lake or be blown in from beyond catchment boundaries, as can airborne sediment, most notably airfall tephra from distant volcanic eruptions. Diatoms often flourish in lacustrine environments, and are the most extensively used palaeoenvironmental indicators (Stoermer and Smol, 1999), due to both environmental sensitivity and to the preservation potential of their cell walls (Rühland *et al.*, 2003). Chironomidae are also used, though less commonly, due to species sensitivity to environmental changes. Each of these constituents can provide information on events and processes occurring in different locations within the catchment and in the lake, though not all will be present at every site.

In order to examine these characteristics, a core must be taken containing undisturbed sediments that have accumulated in the lake. A complete core will contain environmental data from the present day (top) to the date of lake formation (base). The following describes coring methods used to extract sediment cores from lakes in northern and eastern Iceland, and the techniques used in their analysis. The coring method used depends on lake size, water depth, and accessibility.

Background and general methodology

The methodology applied to a lacustrine site will depend upon the environmental question being investigated, and upon the local lacustrine conditions, and divides into two sections: sampling strategy and analysis strategy. Recently, Dahl *et al* (2003) published an evaluation of their methods for coring Holocene lakes. This paper provides a useful starting point for the coring methodology used in this thesis.

The sampling strategy used here break down into two main categories: (1) coring undertaken on frozen lakes during the winter, and (2) coring undertaken from a floating platform during summer months.

Sampling strategy

Dahl *et al* (2003) advise that the best sampling strategy involves coring from a series of lakes arrayed linearly downstream from the glacier. In addition, a core should be taken from a lake outwith the glaciated catchment, but in a similar general setting, in order to provide a control sample for the purposes of analysis. As Dahl *et al.* note, such a strategy is not always possible, as many catchments do not always have multiple lakes or sediment traps within them. In such cases, analysis has to be done from a single lake. Ideally, multiple cores should be taken from each lake in order to correlate the stratigraphy and check for sediment events and structures that are not lake-wide. Cores should be taken away from the lake's inflow and outflow points, and normally from the deepest point in the lake (e.g. Snowball, 1995; Dahl *et al.*, 2003).

Sampling strategies for coring Icelandic lakes can be divided into 'summer' and 'winter' strategies. The summer strategy involves coring being undertaken from a stable anchored floating platform on the lake surface. The winter strategy is to extract the core when the lake surface is frozen, once again allowing sampling from a stable platform. As discussed in Chapter II.5, two sites were chosen for the extraction of lacustrine sediments: Skeiðsvatn in Vatnsdalur, northern Iceland, and Hofsvötn in southeastern Iceland. A summer sampling strategy was undertaken at Skeiðsvatn, while a winter strategy was applied to Hofsvötn.

Skeiðsvatn is located close to the entrance of Vatnsdalur. It is dammed by a large landslide that has blocked the end of the valley. Although Dahl and Nesje (2003)

caution against using such lakes, it is possible to date the age of this landslide using tephrochronology, and therefore determine the lake's viability for this project. It is very unusual as a location within Tröllaskagi, as there are relatively few lakes in this region, and most of those lie within corrie hollows (for example Grænavatn; Kirkbride and Dugmore, 2001a). The maximum depth of the lake was measured at 2m in the centre of the lake.

The methodology applied at Skeiðsvatn was to extract the core during summer from a stable anchored floating platform. At both Skeiðsvatn and Hofsvötn, the decision was taken to take the cores from the middle of the lake, as both lakes are small (<500m), and therefore the middle of the lake provides the best opportunity for successful core extraction.

Anchors may be deployed within the lake, or at points along the lake shore, or a combination of the two. On a small lake such as Skeiðsvatn, it was possible to anchor the platform to the shore at three points using lengths of polypropylene (non-stretch) rope (Figure 2.18). When the ropes were tensioned against each other, a stable platform was created, which moved <20cm laterally during coring. Alternatively, anchors to the bed of the lake could be used in a similar way to provide stability, or a combination of ropes and anchors. A triangular floating platform, was used, and the core extracted from the middle of this triangle which was attached to the boat and one side of the triangle of rope anchors. The rope anchors were secured to each corner of the triangle, an to each other.

Two types of coring equipment were attempted at Skeiðsvatn. The first was a Kullenberg corer. the Kullenberg is a variety of piston/gravity corer where a sampling tube is driven into soft sediment by a series of weights, which are triggered on contact with the bed of the lake. This method of sampling limits the core length to the length of tube used, and is also heavily dependent on the stiffness of the sediment. The shallowness of Skeiðsvatn (~2m) made core extraction difficult with a Kullenberg, which ordinarily operates well in deeper waters. Additionally, the stiffness of the upper layers of sediment within Skeiðsvatn resulted in the loss of two coring tubes as it proved impossible to extract the core once the tube had been driven into the sediment. These initial challenges may be one explanation why this, and other similar, lakes have not previously been cored despite the history of research in Vatnsdalur and Tröllaskagi.

The shallowness of the lake and the nature of the sediment necessitated an alternative coring strategy. A Russian corer, commonly used in the extraction of cores from peat bogs and other terrestrial localities, was used given that the lake was sufficiently shallow. The Russian corer works by side-sampling: isolating a length of sample in a sampling chamber which is driven down to the relevant depth within the lake (Figure 2.19). A series of such samples will build up a full core.

Hofsvötn

The frozen surface of a lake during the winter can be a considerable advantage for sampling in some settings. The ice cover provides a completely stable platform for coring, and in Iceland vehicular access is also possible to otherwise inaccessible sites far from roads, given sufficiently complete snowcover. Extracting cores during the winter proved to be the most practical method for sampling lakes on the plateau surrounding Hofsjökull í Lón (Figure 2.20). There are no roads near the lakes on the plateau, and access from the nearest track, in Geithellnadalur, requires a strenuous hike with a 500m climb up steep ground to the plateau – difficult and dangerous in winter. Consequently it was much more effective to drive across the plateau during the winter, when a near-complete snowcover made access more straightforward using a vehicle to transport heavy coring equipment.

There are several potential problems with working on a lake surface during the winter. 1: The ice must be sufficiently thick to support the weight of the coring apparatus and people, as well as to support the additional forces required to extract the core. This was not an issue on the plateau at Hofsvötn, as the ice was >1m thick. 2: Snowcover and melting snow can be an issue. Deep dry snow above the ice surface is no problem, but where the snowpack is thawing, or stream water flowing over the frozen lake surface, it can cause difficulties for sampling. Where the snowcover is melting extensively in early spring, large deep pools of meltwater and an unstable snow surface can inhibit coring, as well as access to coring sites.

A hole was cut into the ice large enough for the corer to be inserted through, using a hand-operated ice drill (Figure 2.20). Prior to sampling, the lake depth was measured using a weighted 30m measuring tape. Building on experience at Skeiðsvatn, sampling was done using a custom-built modified piston corer. In this

case, the coring apparatus was lowered until the piston was just above the sediment-water interface at which point the piston, which was independently connected to the surface with a wire, was anchored to the scaffold on the surface (Figure 2.20). The core tube was subsequently hammered past the piston into the sediment, using a weighting system similar to that in a Kullenberg corer. The whole apparatus could then be retrieved with the piston providing an airtight seal to keep the sediment inside the coring tube.

Analysis Methods

As the primary aim is to determine the presence/absence of glacier ice in the catchment, this thesis will concentrate upon the physical sediment analysis, but in addition, analysis of the diatoms within Skeiðsvatn was carried out by Hill (2005). The primary methods applied were X-ray density analysis, percentage weight loss-on-ignition, magnetic susceptibility and particle size analysis.

The use of X-rays in core analysis was first described by Karlén (1981), and has subsequently be utilised in many lacustrine sediment studies, for example Snowball (1995), who has used thin sections of cores for very high resolution analysis. Karlén's method was to determine the density of the core by the level of X-ray radiation that could penetrate the sample. This resulted in a very high resolution quantification of core density that could be related to sediment type. Such a strategy was applied here to the cores collected from Skeiðsvatn and Hofsvötn, but with an original approach. The X-ray images were digitised using digital photography and the pixel values provided the relative density value. Absolute values were not determined due to the nature of the X-ray photography carried out. The cores were X-rayed as whole core segments, which reduces the potential resolution of the analysis (in comparison to thin sections), but still allows for relative density to be computed.

Magnetic susceptibility (χ) of sediments is a quantification of the amount of magnetic material within a sediment sample. It is commonly used in a variety of settings, including archaeological and environmental sediment studies. This will vary in relation to the sediment source, composition and grain size of the sample. At both of the lake sites, the bedrock is dominantly basaltic lava flows, which tend to produce high susceptibility readings due to the high levels of iron within them, the

homogeneity of the surrounding rocks in both catchments is useful for eliminating the potential for source rock variations being the cause of varations in susceptibility readings.

Two instruments were used, depending upon the source material. For the Skeiðsvatn core, there was only sufficient material to determine χ with the MS-2B magnetic susceptibility meter. Readings were taken at 1cm intervals, with air readings taken before and after each analysis for calibration. For Hofsvötn, there was enough sediment to carry out mass magnetic susceptibility analysis using the MS-2 meter. Two values of χ were recorded, low frequency (χ_{LF} and high frequency χ_{HF}. χ_{LF} is the same value recorded by the MS-2B. Once again, air readings before and after each analysis were taken.

Percentage weight loss on ignition (LOI) is a widely-applied proxy technique for measuring the level of organic and carbonate material within a sample (Lowe and Walker, 1997). The assumption is that subjecting a sample of material to 550°C will burn any organic material in the sample, leaving only inorganic residue. Although not applied in this study, ignition to 1000°C will remove carbonate material, leaving only silicic material. Samples are placed in crucibles, dried for two hours in an oven at 105°C to remove excess water, and weighed. The weight of the crucible is subtracted to get the sample weight. The samples were then placed in the furnace for two hours at 550°C, allowed to cool and weighed once more: the percentage difference in the two dry weights is the percentage loss on ignition.

Particle sizes are one of the most important indicators of environmental change in the catchment. In the context of this study, particle size analysis can quantify the changes in silt levels with and without a glacier being present in the catchment (Matthews and Karlén, 1992; Nesje *et al.*, 1994; Campbell, 1998). Greater levels of glacial erosion will result in an increase in the quantity of silt-sized particles, and a textural change in the sediments. Samples were taken for grain size analysis following LOI, and so are deemed to be inorganic in content. The instrument used to measure particle size was a Coulter LS230 laser diffraction coulometer. This is capable of differentiating size fractions from 4μm up to 2mm. Samples were washed through a 2mm sieve before analysis in the coulometer, though there were rarely any fragments greater than 2mm in the samples.

II.6.2: Geomorphology

Mapping

Mapping methods used in this study were a combination of aerial photograph interpretation and GPS field survey. Aerial photographs were used to provide not only the base maps for geomorphological mapping, but also to create a digital elevation model that could be used to orthorectify the original photos and provide a base for some of the modelling applied in Chapter IV.

Aerial photograph interpretation

Preliminary mapping was carried out using stereopairs of vertical aerial photographs from Landmælingar Íslands. Photograph coverage was identified that crossed the main locations, Vatnsdalur and Hofsjökull/Hofsvötn, as well as the Hraunsvatn site (Table 2.3). Identified features were drawn on a base map at approximately 1:10,000 scale prior to fieldwork. These included present-day ice masses, larger moraines, some large ice-marginal deposits, as well as major landslips. The presence of areas of late-lying snow, particularly above 900m on the 1990 and 1994 coverage, restricted identification of features close to the ice margins, and in some cases, determination of the extent of the ice masses themselves.

DEM construction

Digital elevation models (DEMs) and orthorectified aerial photographs were created for Vatnsdalur and Hofsjökull with ERDAS Imagine software from ESRI. 8 photographs were used for the Vatnsdalur map and 12 for the Hofsjökull map. This was done using Orthobase Pro 8.5.1, a module in Imagine designed to automatically generate DEMs from a series of overlapping aerial photographs and GPS ground control points. The aerial photographs were scanned at 400d.p.i. for a ground resolution of *c.*1.8m per pixel, and saved as TIFF files. During fieldwork, GPS positions were collected using a handheld Magellan SporTrak GPS unit, giving horizontal accuracy of ±5m. Altitude data was collected using a Suunto air pressure altimeter, as GPS altitudes are not as accurate as their X-Y positions. Reference altitudes at the start and end of each day gave a correction for air pressure changes,

and sea level reference elevation was used as a further check. Positions were averaged for 2 minutes at each site to maximise the point accuracy, and satellite coverage was rarely lower than 7 satellites above the local horizon (favourable orbital inclinations of the GPS satellites gives relatively good coverage around 60°N). Points were collected at landmarks visible on the aerial photos that have not moved in the intervening period. These were mostly stream confluences, stream/road crossings and road junctions. A map of the collected ground control points for the Hofsjökull area is in Figure 2.21. The DEMs were generated with the control points and an additional *c.*100 tie points per overlap (points identified as identical on two or more photographs), and output to a resolution of 10m and 14m for Vatnsdalur and Hofsjökull respectively.

Once the DEM was constructed, it could be used to generate an orthorectified image of each aerial photograph. This image is one of the original photograph without the distortions due to camera/ground geometry and offsets due to hills and valleys. Such an image allowed more accurate mapping of the major features visible on the photographs, mapping which was checked with additional GPS mapping of the same features. These orthorectified images were subsequently used for geomorphological map production.

Field mapping

Evidence of glacial activity (primarily glacial deposits) and postglacial slope movement was collected along with aeolian sediment sequences. Positions were identified using handheld GPS (minimum 6 satellites and averaged for 30 seconds), and recorded in Universal Transverse Mercator (UTM) coordinates. The GPS positions are accurate to within 10m horizontally; an altimeter was used to obtain heights to within 10m vertically. Features were sketched, and drawn onto enlarged aerial photographs in the field, and later transferred onto the base map.

Evidence of glacial activity primarily consisted of lateral and terminal moraines, and was associated with rock glaciers and other landforms involving debris-covered ice including protalus ramparts. Moraines were identified as linear or arcuate ridges, *c.*0.5-10m high, composed of angular to subrounded boulders, often (but not exclusively) in a finer matrix. Landslides, alluvial fans and screes could be checked in relation to their appearance on aerial photographs.

II.6.3: Stratigraphy and tephrochronology

Aeolian sediment sequences

One of the characteristics of the Icelandic landscape is the nature of the soil that accumulates beneath vegetated areas. The 'soil' is dominated by aeolian sediment accumulation, from both proximal and distal sources on the island. These aeolian soils accumulate quickly, due to the large amount of available loose sediment. Large areas of unvegetated upland and fresh sediment (particularly tephra) from volcanic eruptions results in significant volumes of mobile sediment. A further source of sediment is from the erosion of existing soils. In many parts of Iceland, the erosion of these sequences is clearly visible as a sediment escarpment – such escarpment both highlight the extent and severity of the erosion, and allow investigation of the sequence of sediments deposited in the surviving soil. Accumulations of peat also occur where there is poor drainage, though it is less common than aeolian silt, and tends to have a high proportion of silt within it.

The greatest value of these sequences comes from tephra deposited from volcanic eruptions of a known age. Tephra layers are very often visible within the sediment sequences, giving Icelandic soil it's characteristic 'stripy' appearance, and provide a powerful dating tool, both relative and absolute (Thorarinsson, 1944; 1974; Figure 2.22).

Additionally, aeolian sequences can record other environmental data. Evidence of disturbance to a sequence often indicates slopewash or solifluction, which is visible through evidence of layer disturbance, graded and sorted coarse sediments or by repetition and inversion of tephras. Episodes of disturbance may be present in only part of a sediment sequence, indicating that the source of the disturbance is not constant in time. Rarely, direct evidence of vegetation change is visible. Tree root/trunk casts have been found in sediments from Langanes (Figure 1.7), and where peat has accumulated, various macrofossils have been preserved. A change from peat to aeolian sediment indicates a drying out of the sequence, either through raising of the land surface by sediment accumulation, or by lowering of the water table.

Tephrochronology

Tephrochronology involves the dating of geomorphological features by their relationships to tephra layers, and the identification, correlation and dating of those

layers (Thorarinsson, 1944, 1974). Rapid accumulation of aeolian soil in conjunction with the proximity to volcanic eruptions producing large volumes of tephra makes tephrochronology a practical tool in many parts of Iceland (Thorarinsson, 1967; Larsen, 1984; Larsen *et al.*, 1999; 2001). Selected key tephras are listed in Table 2.4. In particular, major eruptions of Hekla throughout the Holocene provide markers across large parts of Iceland (Figure 2.23).

In some cases, the tephras can be unambiguously identified using their geochemical signature. Larsen (1981) and Larsen *et al* (1999) have shown that the use of Electron Probe Micro-Analysis (EPMA) can be used to resolve the identity of airfall tephra (Figure 2.24). EPMA can distinguish between tephras produced by major volcanoes, although two tephras produced by the same volcano may have a near-identical chemical signature. Important silicic tephra layers from the study area were analysed using EPMA, following the methods of Dugmore *et al* (1992; 1996; 2000) and Larsen *et al* (1999; 2001).

By integrating the geochemical data with stratigraphic data, the tephra layers can be unambiguously distinguished and correlated, providing exact marker horizons. It has also been shown that these glass shards maintain their geochemical signature for at least 4000 years without being degraded (Dugmore *et al*, 1992), negating this as a problem for virtually all tephras in this study.

Consequently, the tephras provide a powerful tool to improve the dating of geomorphic features (e.g. Bradwell, 2001; Casely, 2001; Kirkbride and Dugmore, 2001a; Kirkbride and Dugmore, 2001b; Casely and Dugmore, 2004). Those layers that appear in lake sediments can also be examined and tested.

Identification and sampling of tephra

Tephras were identified in soil sections in all study areas (e.g. Figure 2.22). These sequences are found in relatively sheltered locations, comprised of aeolian silt and tephra, and not include material washed in from elsewhere. Inwashing could distort the tephra record, or remove key layers, thus affecting the information about the age of the sequence and other environmental data contained within them. Ideally, the soils would be sub-horizontal or deposited parallel to the hillslope, allowing the true layer thickness to be identified. In cases where tephras are deposited on steep slopes, layer thicknesses are unreliable, but provided slopewash and solifluction are minimal,

the sequence of layers will be undisturbed. Important morphological characteristics identified for each tephra were layer thickness, colour and particle size.

Samples were taken of key indicator tephras using a clean knife and placed into separate sealed sample bags to avoid contamination. The most important layers were mostly silicic Hekla and Öræfajökull tephras and the Vö870 Landnam tephra. In sequences from both northern and eastern Iceland, dark/black basaltic tephras were generally not sampled except for a few cases (for example the Vö1477 tephra), as the geochemical signature would not be sufficiently distinct.

Geochemical analysis of key tephra layers

Sample preparation

In order to analyse tephra using the electron microprobe, clean tephra samples must be mounted on slides for the microprobe. In the case of samples taken from thick tephra layers, the original sample is frequently nearly 100% tephra and requires little pre-treatment. These samples were washed with dilute HCl and treated to 15 minutes ultrasound to separate the glass particles before mounting. Samples from thiner layers and layers mixed into the surrounding sediment would be likely to have a significant proportion of contaminant organic matter, and so were treated using acid digestion. This involved treating the samples with concentrated sulphuric acid and nitric acid to remove organic material before repeatedly washing, centrifuging and draining the samples to return them to a neutral pH.

The samples were then mounted in Araldite on glass slides with six samples to a slide, ground using a combination of hand-grinding and machine polishing to a thickness of ~50μm and carbon coated for the electron microprobe.

Microprobe analysis

Geochemical analysis was carried out using a Cameca SX100 electron microprobe and a standard WDS (wavelength dispersive) technique, an accelerating voltage of 20kV and a beam current of 4 or 10nA as measured across a Faraday cup. Analytical protocols follow Dugmore *et al.* (1995). The probe operates by accelerating a beam of electrons at a potential difference of 20kV towards the sample. X-rays are

backscattered from the sample at specific angles depending upon the elements present in the sample. These angles are calibrated using mineral standards, including andradite garnet [$Ca_3Fe^{3+}_2(SiO_4)_3$], wollastonite [$CaSiO_3$] and periclase [MgO]. The andradite standard was measured before and after each tephra sample, and the instrument recalibrated if the measured element values fall outside acceptable limits for the standard. The calibration quantifies the amount of backscatter at particular angles, and therefore the amount of the element present. Backscatter counts are made for ten elements (Si, Al, Fe, K, Ca, Mn, Mg, Na, Ti, P) – these elements comprise almost all of any Icelandic tephra sample. The results are converted into the weight percentage of the oxide to be used as comparison to existing published data or data available on the Tephrabase tephra database (http://www.geo.ed.ac.uk/tephra/).

Each probe analysis is carried out on a different individual tephra grain, visually identified in the sample under both reflected and transmitted light. Grains are checked for the quality of their surface, and for the presence of vesicles just beneath the surface which could be exposed during analysis by the beam. For silicic tephras, a beam current of 4nA is used, and for basaltic tephra a 10nA current was applied. The higher current gives a greater count rate of backscattered x-rays, but increases the risk of volatilisation of the reactive elements such as sodium, present in significant quantities in silicic samples. Counts of x-rays were done for 10 seconds for each analysed element, with a background count for 5 seconds.

Icelandic Glacier Type	Sensitivity Level
Valley	Rare in Iceland; sensitive to climate change
Corrie	Common in northern Iceland, and scattered elsewhere; sensitive to disappearance or growth into small valley glaciers.
Ice cap piedmont lobe	Drains most ice from the Icelandic ice caps; insensitive to climate, due to slow response times and loss of small-scale climate signals.
Ice cap outlet valley	Uncommon in Iceland, a few examples from Vatnajökull and Mýrdalsjökull; sensitive to climate, and particularly sensitive to retreat if ELA rises onto the ice cap.
Ice cap	Several large ice caps, a few scattered small examples; Large ice caps operate through the two above outlets, small ice caps are sensitive to melting if the ELA rises above their summits.

Table 2.1: Categories of glacier and their sensitivity in Iceland.

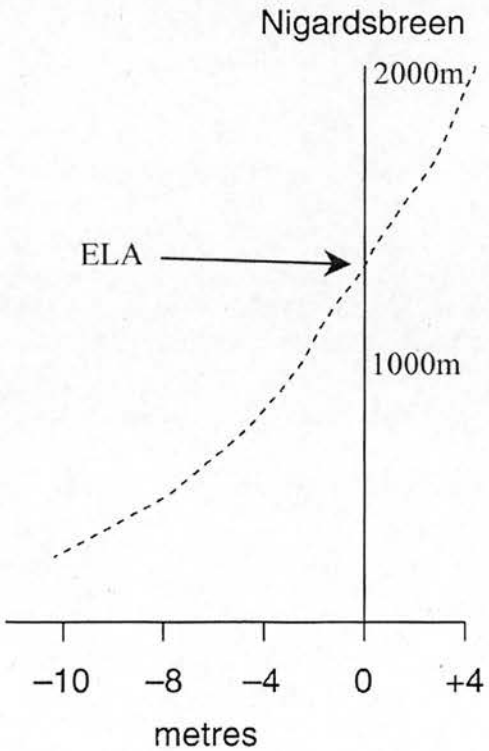

Figure 2.1: Glacier mass balance and equilibrium line altitude at Nigardsbreen, Norway (Benn and Evans, 1998). The ELA marks the elevation above which mass is gained over the course of a year.

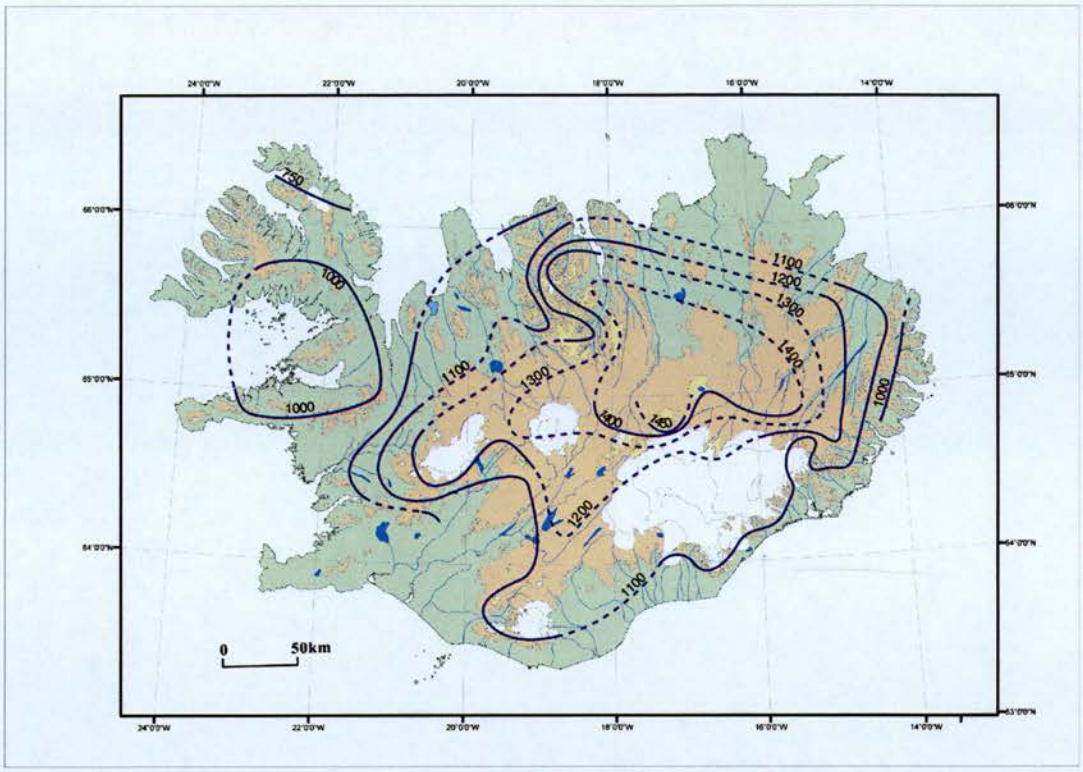

Figure 2.2: Equilibrium line altitude map as drawn by Ahlmann and Thorarinsson (1937) – the only such map to date.

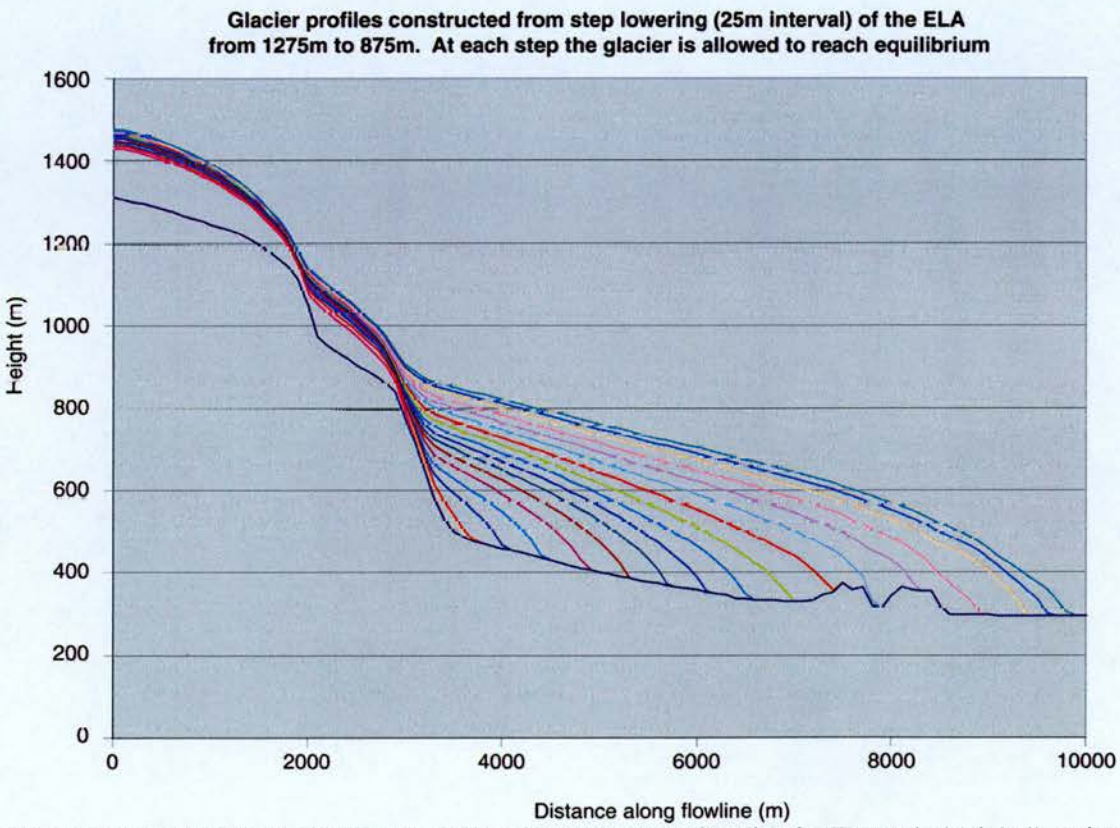

Figure 2.3: Equilibrium glacier model profiles along the central flowline for Tungnakvíslajökull, an ice cap outlet valley glacier in southern Iceland. Each successive profile represents a 25m ELA lowering, the equivalent of a 0.16°C temperature lowering (Casely, 2001), and produces a measurable change in glacier length.

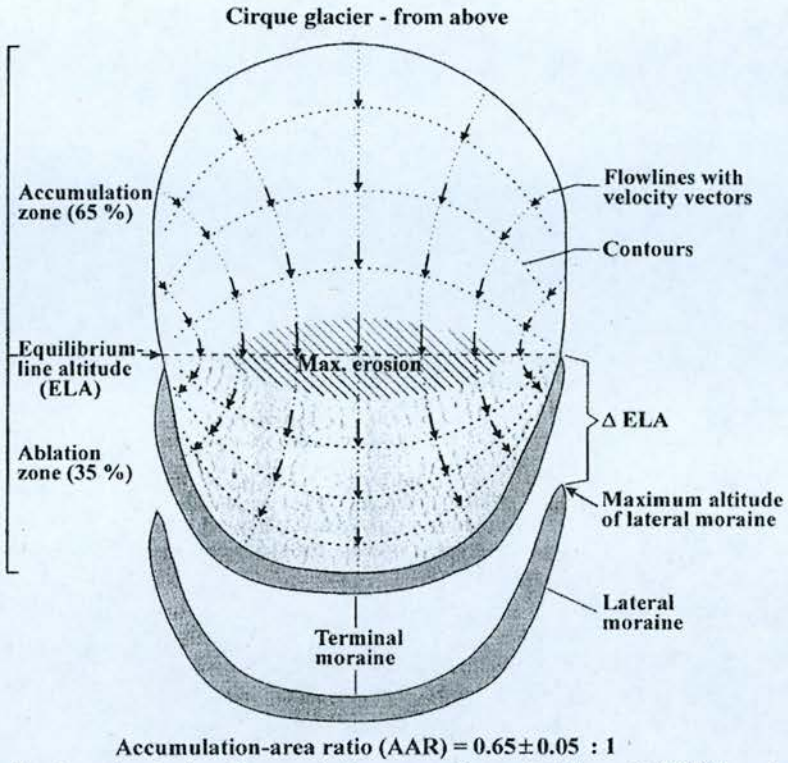

Figure 2.4: Idealised corrie glacier characteristics, according to Dahl *et al.* (2003), and estimation of glacier ELA from accumulation/ablation area or elevation of lateral moraines.

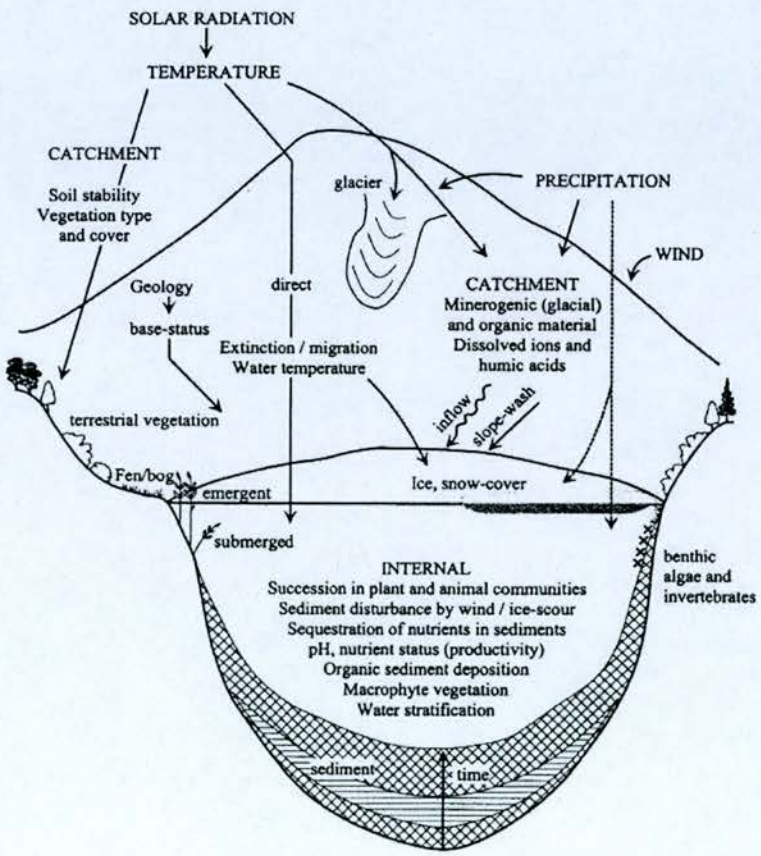

Figure 2.5: Inputs and processes affecting the environment of deposition in a lake (Birks *et al.* 2000).

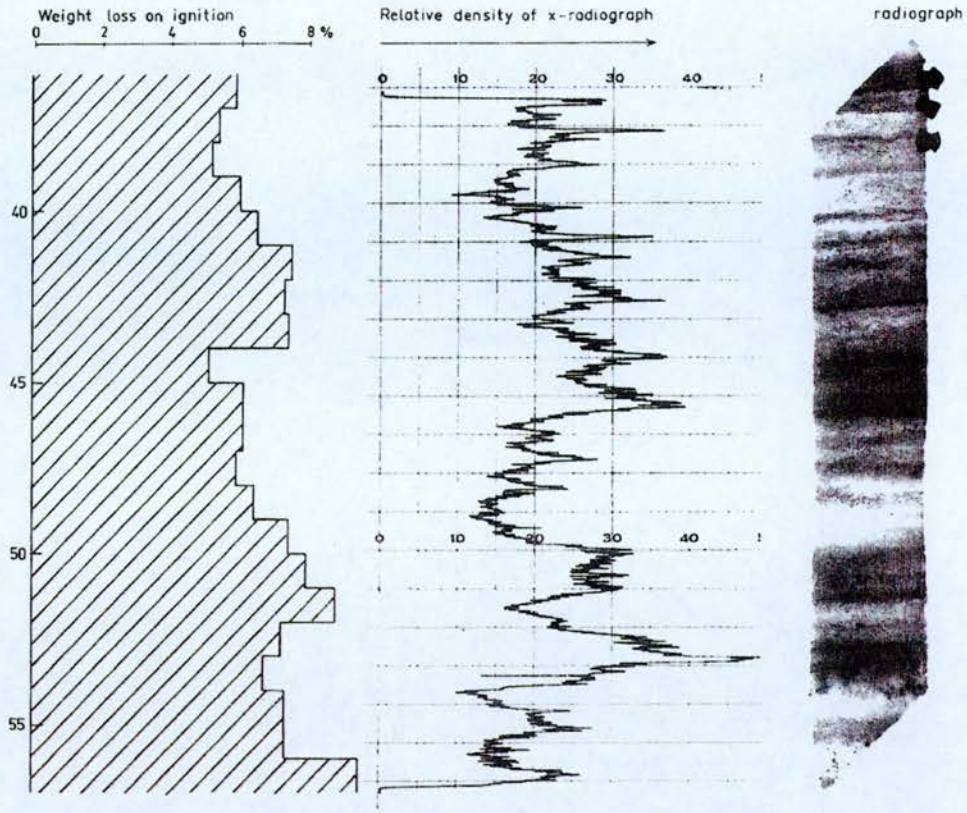

Figure 2.6: Sedimentological changes recorded in lacustrine sediments from Scandinavia (Karlén, 1981)

Site No.	*Lake or glacier name/location*	*Glacier type*	*Geothermal heat?*	*Clear drainage path?*	*Visible glacial geomorphology?*	*Chosen for further study?*
1	**Kerlingarfjöll**	short valley	Y	N		
2	**Hrutfell**	small ice cap	Y	N		
3	**Gláma**	ice cap (not present)	N	Y		
4	**Drangajökull**	ice cap	N	Y		
5	**Hofsvötn, Hofsjökull í Lón**	ice cap	N	Y	Y	Y (cored)
6	**Þrandarjökull**	ice cap	N	Y	N	N
7	**Skeiðsvatn, Tröllaskagi**	corrie	N	Y	Y	Y (cored)
8	**Hraunsvatn, Tröllaskagi**	corrie	N	Y	Y	Y
9	**Smjorfjoll, NE Iceland**	corrie	N	Y		
10	**Snæfell, E Iceland**	mountain ice cap	N	N	Y	N

Table 2.2: Potentially suitable lake coring sites: sites where small glaciers have proglacial lakes.

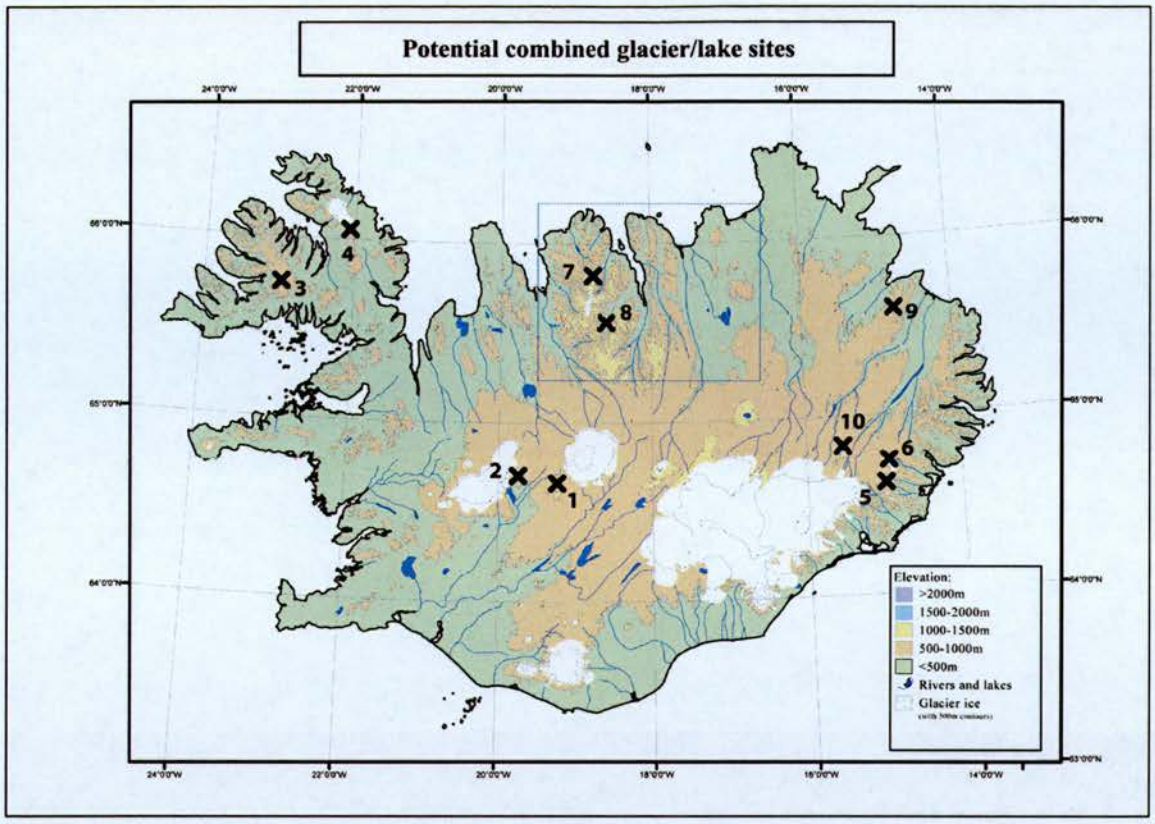

Figure 2.7: Potential sites identified for lake coring in Iceland, on the basis of the presence of a lake beneath small glaciers (numbers refer to Table 2.2). Sites 5 and 7 (Hofsvötn and Skeiðsvatn) were sampled.

Figure 2.8: 180 degree panorama of the proglacial lake at Snæfell, looking west. The ice is partially hidden by the clouds on the left (southwest). Drainage from the ice does presently partially enter the lake before exiting to the west, but the main drainage route is directly across the delta to the western exit.

Figure 2.9: Skeiðsvatn in Vatnsdalur from the shore of the lake, looking south towards the glaciers.

Figure 2.10: Panorama of Vatnsdalsjökull (right) and other glaciers at the southern end of Vatnsdalur.

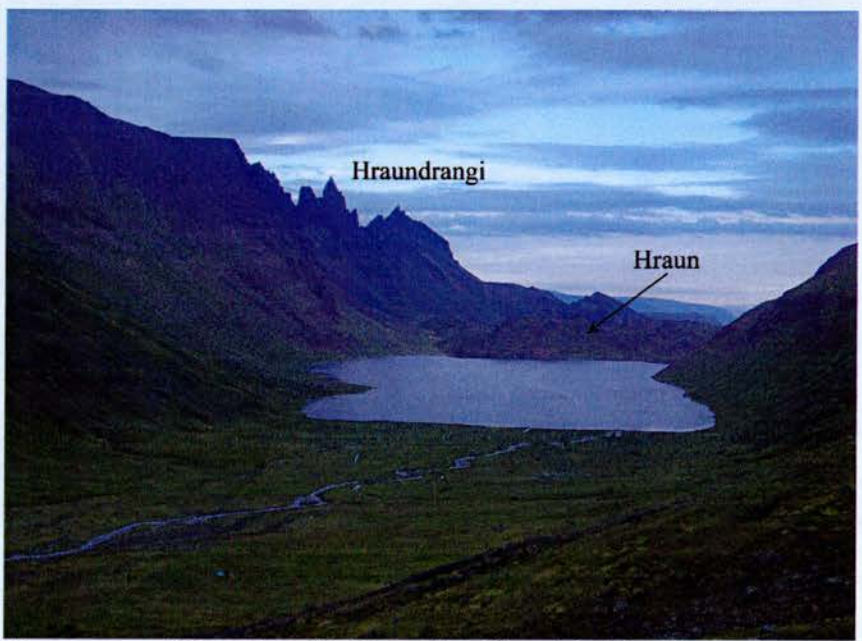

Figure 2.11: Hraunsvatn and Hraundrangi in Vatnsdalur (Öxnadalur), looking northeast. The landslide (Hraun) dams the lake.

Figure 2.12: Panorama of the upper Hofsvötn lake (looking south), with the northeastern edge of Hofsjökull visible to the upper right.

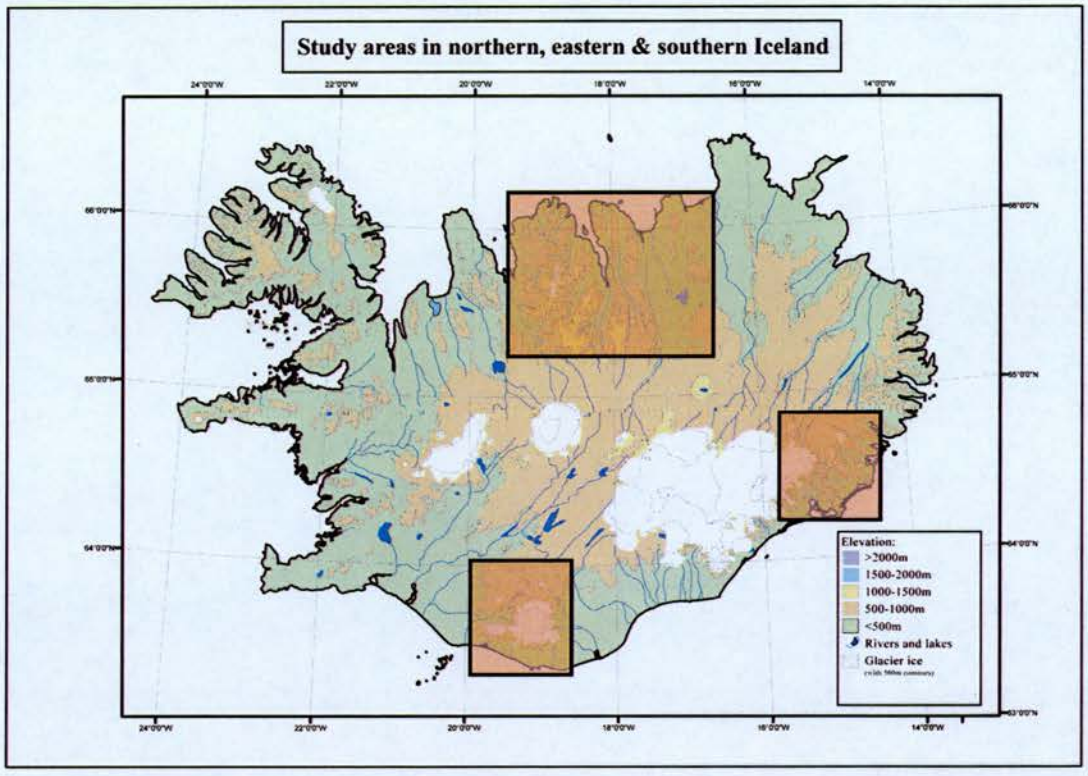

Figure 2.13: Location of the three primary study areas. Northern Iceland contains Tröllaskagi, with associated small glaciers and the lake Skeiðsvatn in Vatnsdalur, as well as the important archaeological sites around Lake Mývatn.

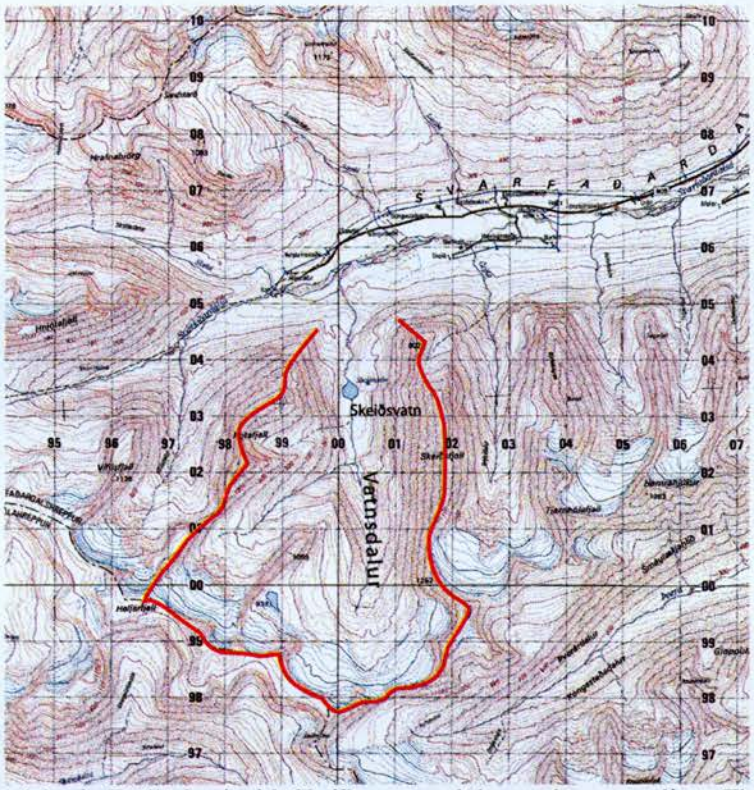

Figure 2.14: Vatnsdalur in Tröllaskagi with Skeiðsvatn, and the catchment outline. The small glaciers at the head of the valley, as well as some of those in nearby valleys, are visible. Grid squares are 1km^2 and north is up (Excerpt from the Landmælingar Íslands 1:50,000 topographic map).

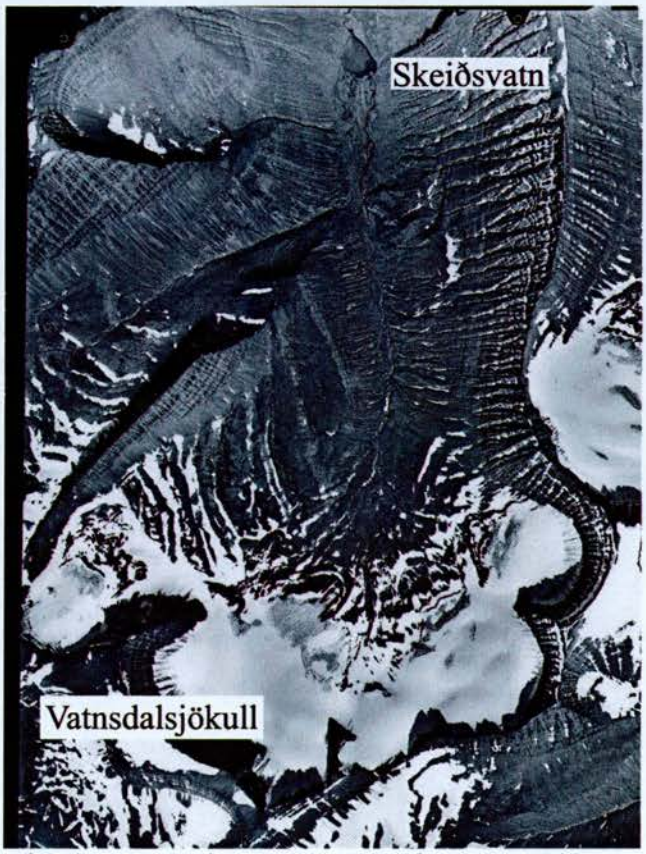

Figure 2.15: Vertical aerial photograph of Vatnsdalur, taken in August 1990.

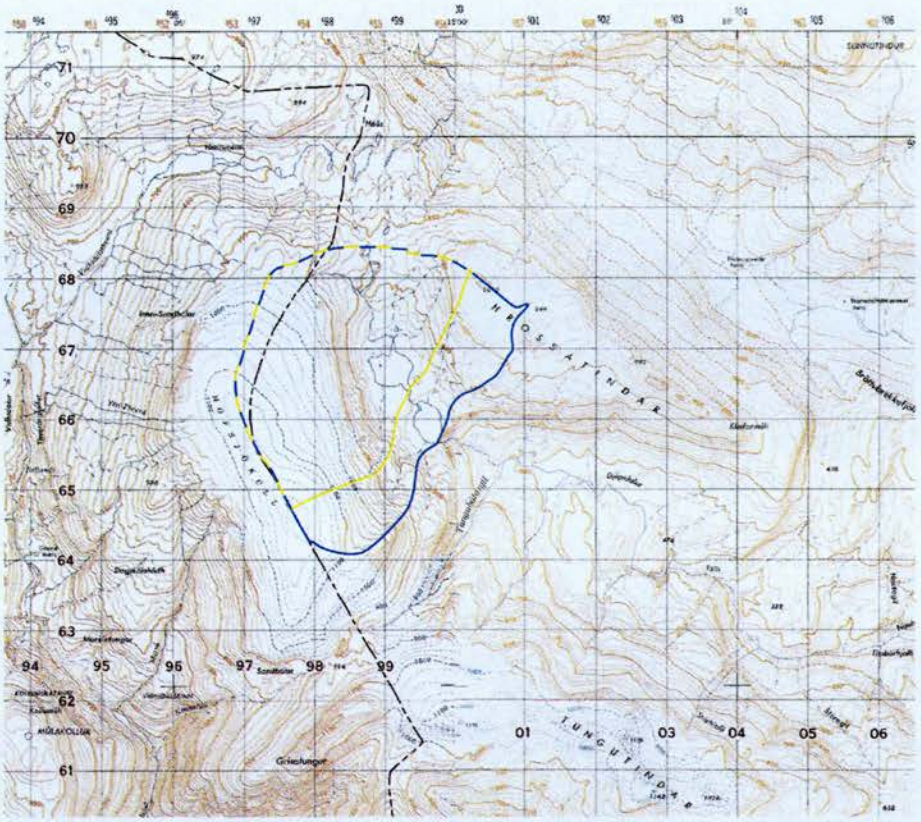

Figure 2.16: Hofsjökull í Lón, with the two Hofsvötn lakes. The lake catchments are in yellow and blue for the upper and lower lakes respectively. Geithellnadalur lies to the northeast of Hrossatindar. Grid squares are 1km^2 and north is up (Excerpt from two Landmælingar Íslands 1:50,000 topographic maps).

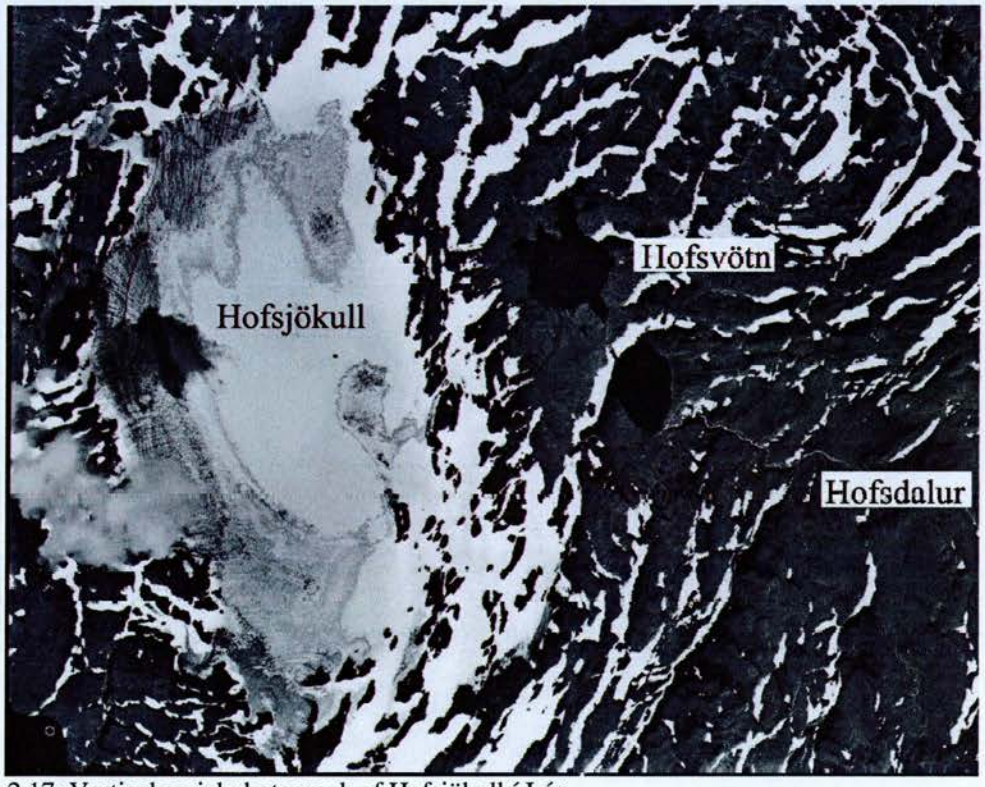

Figure 2.17: Vertical aerial photograph of Hofsjökull í Lón.

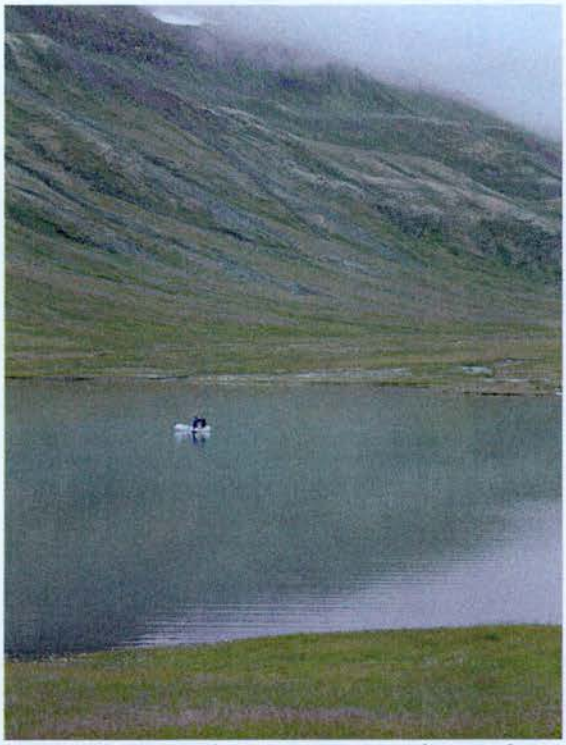

Figure 2.18: Summer coring at Skeiðsvatn, with the boat and coring platform anchored to the shore by 3 ropes.

Figure 2.19: An example of a core sample extracted with Russian corer. This particular sample was not used (taken near the lake margin), but shown as the camera was not risked on the lake coring platform!

Figure 2.20: Winter coring on Hofsvötn. (Top) Drilling through the ice. The ice drill is 2m long from handle to tip, and Jökull is demonstrating the total thickness of ice and overlying snow/slush! (middle) Piston coring: Skúli and Jökull are raising and lowering a weight to hammer the core tube into the

sediment, while the piston is attached to wires held by Thorkel. The coring platform is on 1m of snow the snow covering the ~1m of ice over Hofsvötn. (Bottom) Close-up of the coring platform.

Site	Date	Coverage Quality	Used / rejected?
Hofsjökull	1946	Clear skies, and little snow, but large shadows and limited coverage	Rejected – no camera information
	26/8/1976	Clear skies and full coverage, little snow	Used for geomorphology. Rejected for DEM construction – incorrect camera information
	2/8/1990	A few small clouds, more extensive snow, full coverage	Used for geomorphology. Used for DEM construction – but has minor cloud cover
Vatnsdalur	7/8/1994	Clear skies, full coverage, large snowpatches on/next to glaciers.	Used for DEM construction and geomorphology.
Hraunsvatn	7/8/1994	Clear skies, full coverage, large snowpatches on/next to glaciers. Target glacier on boundary of 4 photographs.	Used for geomorphology.

Table 2.3: Aerial photograph coverages used for geomorphological mapping and DEM construction.

Layer name and age if known	Volcano / volcanic system of origin	Colour
Vö1477	Veiðivötn	fine grey/black
Ö1362	Öræfajökull	fine white
H1104 (H1)	Hekla	fine white tephra
E935	Eldgjá	fine grey?
Vö871 (Landnám)	Veiðivötn	Two components – olive-green & white
H3 (~2900BP)	Hekla	yellow
H4 (~3800BP)	Hekla	white/grey
H5 (~6000BP)	Hekla	white/grey

Table 2.4: Key Holocene indicator tephras that are widespread in Iceland. Tephra presence, thickness and grain size is dependent on the particular locality.

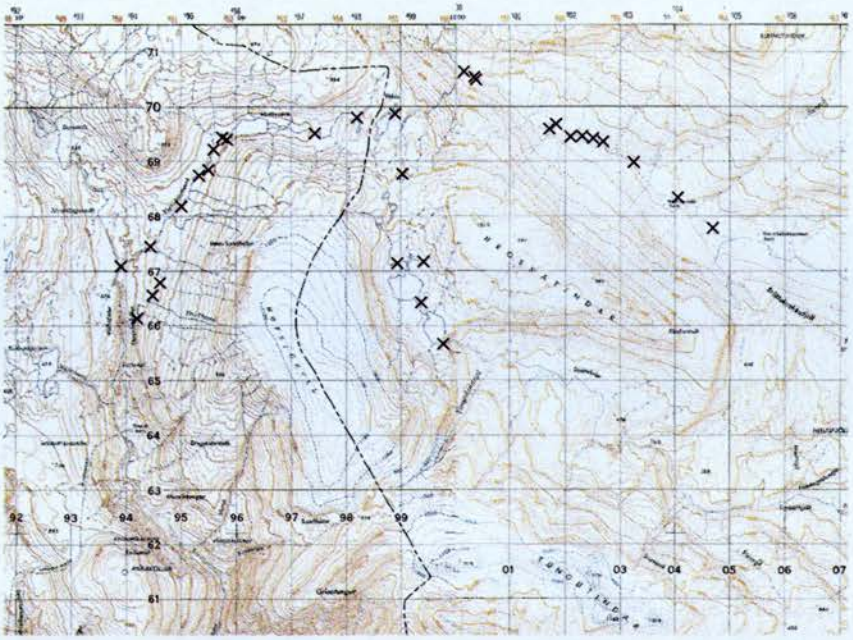

Figure 2.21: Map of the GPS points taken around Hofsjökull, Hofsvötn and Geithellnadalur as ground control points for DEM construction. Points are at features identifiable on both aerial photos and on the ground, and not all visible on the map.

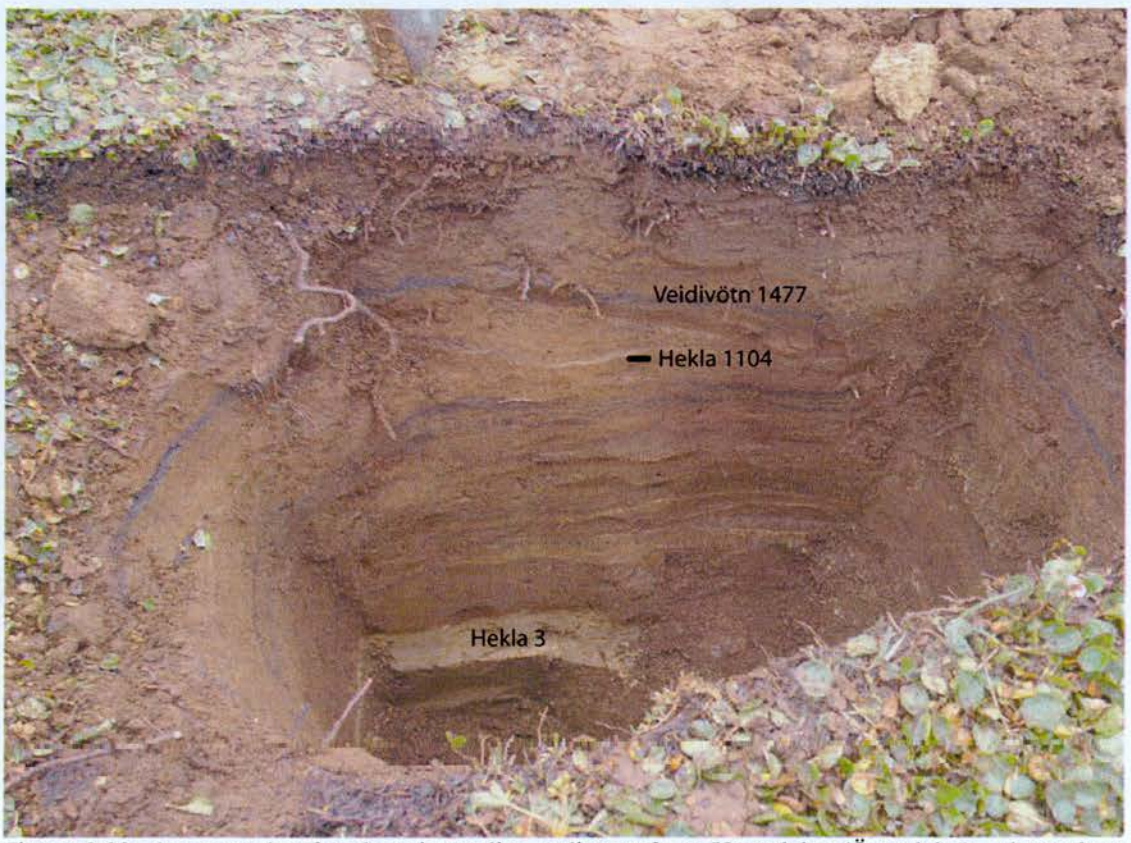

Figure 2.22: An example of tephras in aeolian sediment from Vatnsdalur (Öxnadalur): the tephras appear as distinct layers and are identifiable by colour, texture and relative stratigraphy. The black V1477 layer is ~1cm thick.

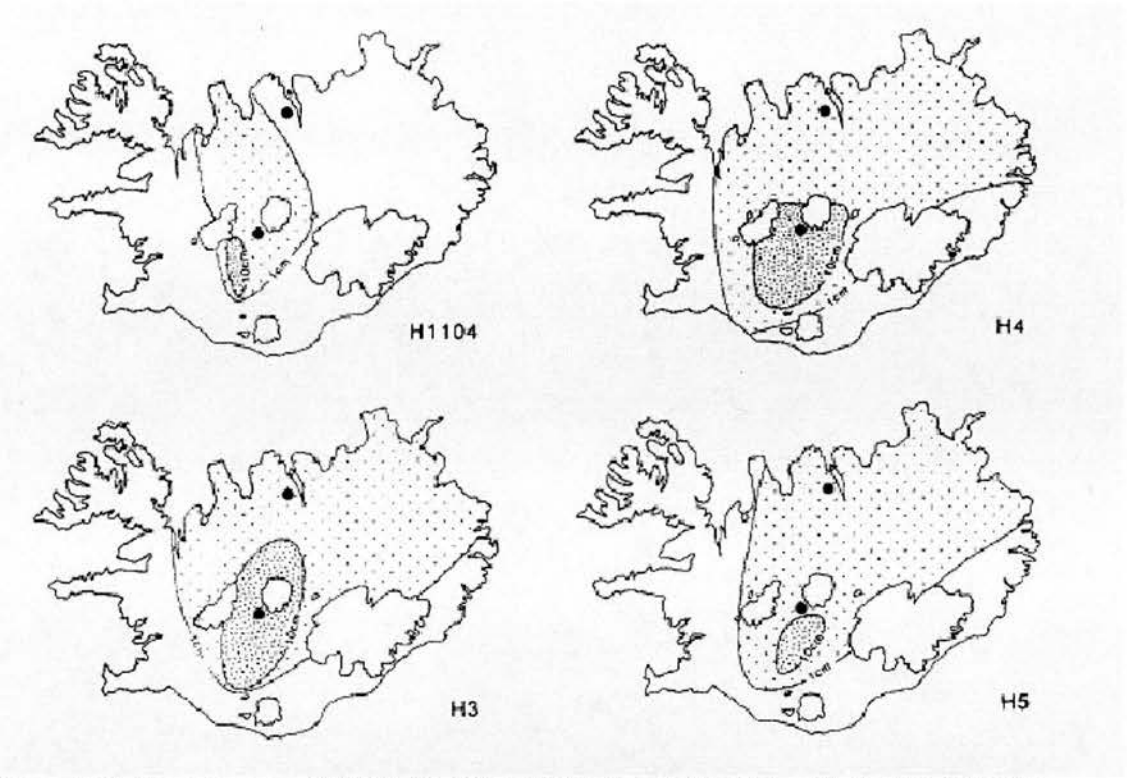

Figure 2.23: Isopach maps of H1104, H3, H4 and H5 tephra distribution, with shading indicating fallout thickness >10cm (darker) >1cm (lighter) represent fallout >1cm. These maps indicate that Tröllaskagi is the most favourably placed to receive the fallout from each of these eruptions.

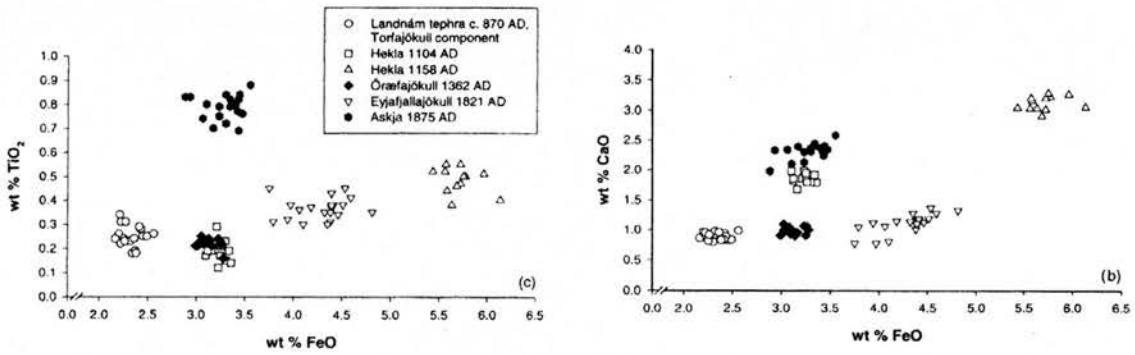

Figure 2.24: Comparison of selected chemical components of historical tephras by Larsen *et al* (1999). Individual clusters of points relate to individual eruptions. Comparison of different components can separate similar eruptions - although two eruptions can have similar compositions in one diagram, they can be distinguished in the second diagram.

Chapter III: Datasets

1 Introduction
2 Northern Iceland
3 Southeastern Iceland

III.1: Chapter Introduction

This chapter presents the datasets gathered in this project. As discussed in Chapter II they come from three sources: existing published data and reports, new empirical data and modelling data. The modelling data will be presented in Chapter IV. In this chapter the remaining empirical data will be presented for two of the three study regions in Iceland: Mývatn and Tröllaskagi in northern Iceland; and Hofsjökull, Snæfell and the valleys of southeastern Iceland.

Data types

There are three main types of empirical data to be presented in this chapter: geomorphic evidence, lacustrine sedimentary evidence and terrestrial sedimentary evidence. The data will be presented here as separate units which will then be integrated with the modelling experiments (Chapter IV) in the synthesis and discussion in Chapter V.

III.2: Northern Iceland

III.2.1: Introduction

The two key regions of northern Iceland are the Skeiðsvatn lake coring site in the Tröllaskagi mountains, and the settlements around Lake Mývatn. This section will present the relevant environmental data from geomorphological investigation in Tröllaskagi: II.3.2 will comprise the original geomorphological and sedimentological research carried out in Tröllaskagi, and III.2.3 presents additional geomorphological and sedimentological results from Vatnsdalur in Öxnadalur.

III.2.2: Skeiðsvatn, Vatnsdalur

Skeiðsvatn is the first site from which new data is presented concerning glacier fluctuations and Medieval environmental change. The data from Skeiðsvatn primarily concerns glacier fluctuations, as seen in the geomorphology and lacustrine sedimentology. Additional aeolian sediment sequences are presented from Skeiðsvatn and from Vatnsdalur in Öxnadalur. The Tröllaskagi region has been the subject of numerous studies of environmental reconstruction, primarily focusing upon glacier fluctuations in the area. Some of the evidence found from these studies is presented prior to the results for Vatsdalur.

Pre-LIA Holocene glacier fluctuations and geomorphology in Tröllaskagi

The mountains of Tröllaskagi have provided some excellent information relating to glacier fluctuations and climatic change through the reconstruction of former highstands of the glaciers. Within the Tröllaskagi mountains, the present-day ELA is approximately 1100m, producing about 115 small corrie and short valley glaciers, but no large ice masses (Ahlmann and Thorarinsson, 1937; Björnsson, 1979).

There are many dated moraines that were formed prior to the LIA: indeed the area includes the majority of such pre-LIA Neoglacial features in Iceland. This may be due to either the response rate of the glaciers, or to differing characteristics of the pre-LIA cooler phases (for example having increased precipitation in northern Iceland), thus pushing the glaciers slightly beyond their LIA maximum positions. There are several moraine groups that relate to significant ELA lowering (>220m), but these

appear to be older than the Preboreal Búði stage (Hjartarson and Ingólfsson, 1988; Häberle, 1994), and were likely to have formed during glacier retreat from Younger Dryas positions, and are not 'Neoglacial' in age. No subsequent Holocene advance has had an ELA substantially lower than that of the LIA – the older moraines in Table 3.1 and described in the following three paragraphs lie just beyond LIA moraine limits.

Häberle (1991) has carried out a study of moraines in Barkardalur, one of the sites in Tröllaskagi where there are several moraines lying outwith LIA limits. Häberele has postulated two dated advances outside the LIA limits – one 'shortly before' 1555±90BP, and on bracketed between 1835±120BP and 2240±120BP. The inner limit is based on a minimum age at a 'limit of soil development' within Häberle's profile B, and therefore may be comparable in age to the outer limit. The c.2000BP advance is Barkardalur I, and the 1555BP advance is Barkardalur II.

In Vatnsdalur, Stötter (1991) dated a glacier advance to between c.6000BP (the H5 tephra) and 4700±205BP. Stötter has interpreted a second advance of Vatnsdalsjökull subsequent to 3470±160BP, based on the end of peat formation between the two moraines, subsequent disturbance of the peat profile, and the lack of the H3 tephra within the peat. However, this is still not bracketed by a minimum age, and it is possible that the moraine is significantly younger. Stötter correlates these advances to Dugmore's (1989) mid-Holocene Drangagíl and Holsárgíl Sólheimajökull advances, and points to evidence that between these advances the climate was sufficiently warmer as to have and ELA c.150m higher than at present.

Evidence for a glacier advance in the Grænavatn cirque dating to between the H5 and H4 tephras (c.5950-c.3830BP) is presented by Kirkbride and Dugmore (2001a). Aeolian sediment accumulated within a hollow on the distal slope of a moraine fragment contains the H3 and H4 tephras. Despite favourable conditions for accumulating sediment, this hollow does not contain the H5 tephra, therefore bracketing the moraine age. However, there is still the caveat of 'absence of evidence'. Additionally, Kirkbride and Dugmore note that "...the distribution of H5 is patchy in the Tröllaskagi Peninsula."

LIA glacier fluctuations and rock glaciers in Tröllaskagi

Lichenometry is the most important tool for dating glacier advances during the Little Ice Age in Tröllaskagi. There are few documentary records to draw upon, and ther are no suitable tephras to distinguish LIA advances.

At Barkárdalsjökull, Häberle (1991) has determined, using largest lichen methods, a maximum LIA advance at c. AD1750. Häberle reports other lichenometric and documentary dates for maxima between 1801 and 1855 at Bægisárjökull, a corrie glacier in Skríðudalur, Vindheimajökull and Myrkarjökull. Kugelmann (1991) has generated a linear lichenometric dating 'curve' for the Tröllaskagi area using available surfaces in Skiðadalur and Svarvaðardalur, and using it has estimated the age of LIA glacier advances in the area. Kugelmann reports a range of maximum advance ages, the oldest being c.1810AD (in Burfellsdalur and Vatnsdalur) and with many glaciers advancing or readvancing between c.1870AD and c.1900AD. Caseldine (1985; 1987) has also used Skíðadalur for lichenometric work. Caseldine's moraines have been dated by lichenometry to the late LIA, falling into three age bands – 1896-1908AD, 1914-1922AD and 1930-1935AD, and are attributed to climatic amelioration at the end of the LIA (Caseldine, 1987). Also in Tröllaskagi, Caseldine and Stötter (1993) obtained ages for the LIA maximum back to 1813.

Within Iceland, rock glaciers are largely restricted to the north of Iceland, and there are many examples within Tröllaskagi. This is thought to be due to the lower precipitation, and generally more restricted ice volume within the area.

A number of recent papers have commented upon the alleged controversy regarding the classification of 'rock glaciers', and specifically whether proglacial debris areas with morphological characteristics of rock glaciers are, in fact, true rock glaciers (Barsch, 1996; Brazier *et al.*, 1998; Clark *et al.*, 1998). For the purposes of this study, 'rock glacier' covers a range of landforms whose genesis produces an equifinal outcome – including both proglacial and protalus types.

A range of morphological examples have been previously studied within the Tröllaskagi region. The rock glacier in Klængshóll (Meyer and Venzke, 1985) has been reevaluated by Martin *et al* (1991) as an ice-cored moraine. Many such ice-cored moraines exist, and may represent quite significant hidden ice volume in this part of Iceland (Martin *et al.*, 1991). The Nautárdalur rock glacier was mapped by Martin and Whalley (1987a, b) and represents an excellent example, though not the only one in the area. It shows no apparent advance, though there is slow forward

movement caused by ice input balanced by ice ablation which maintains a steep ice front (Martin and Whalley, 1987a).

Geomorphology

Geomorphological mapping of Vatnsdalur (Fig 3.1) reveals a wide variety of morphological features, some of glacial origin, and others with different origins. The entire catchment of Skeiðsvatn was visited for mapping in the summer of 2003 with three purposes:

1: mapping morphological evidence for glacier advances.
2: revisiting the evidence presented by Stötter (1991) for earlier Neoglacial glacier advances in the study area.
3: examination of aeolian sediment profiles from the catchment, to constrain the timing of geomorphological events in the catchment, and to investigate other environmental changes within the catchment.

The geomorphological mapping shows that of the seven small corries, six have ice masses within them, and one is ice-free. In front of the major corrie glaciers at the southern end of Vatnsdalur there is a suite of lateral and terminal moraines. The outermost of these clearly predate the Little Ice Age, as reported by Stötter (1991), and shown visually by the depth of soil cover on the moraine (Figure 3.2). Within these, there are a series of retreat postions marked by linear and arcuate lateral and terminal moraines (Figure 3.3), indicating amalgamation of ice from the two largest corries duing more advanced positions, and some indicating independent ice movement from the smaller eastern corrie. All of these younger moraines are unvegetated and comprised almost entirely of cobbles and boulders with little or no intervening sediment matrix, suggesting a relatively recent genesis.

There are two large rock glaciers situated within and on the margins of the main suite of moraines. Both rock glaciers appear to be related to Little Ice Age glacier limits, and are in positions marginal to the main LIA moraines. The western of the two has advanced beyond this position. The rock glaciers present a typical crenellated surface with flow structures and steep fronts 5 - 50m high (Figure 3.3; 3.4). Glacier ice is exposed at several places along the front of the western rock glacier which,

allied to its location in front of Vatnsdalsjökull, indicates a glacial origin for this feature. The western rock glacier flows beyond the pre-LIA ice limits onto terrain probably unglaciated during the Holocene, suggesting considerable movement of this feature (Figure 3.4). The eastern rock glacier is located within the outermost LIA moraine, and its position suggests formation due to the coalescing of debris sources from Vatnsdalsjökull and its eastern neighbour (ice limits from both glaciers are visible on Figure 3.3).

The southwestern tributary valley contains two of the glaciers draining meltwater into Skeiðsvatn (Figure 3.5). In front of the larger of these glaciers is a large ice-cored terminal moraine, which has a series of smaller moraines just beyond the main moraine. The ice within the larger moraine complex is clearly visible melting and collapsing in several positions. There is a small moraine fragment *c.*500m beyond the main moraine sequence, but this fragment is isolated, and there is no additional morphological evidence for an ice limit at this position.

The ice-free corrie situated beneath Kotafjall (west of Skeiðsvatn) contains a small series of arcuate terminal moraines. Within these moraines are several features indicitave of periglacial processes. On the southern corrie wall there are several fragments of protalus ramparts which have formed beneath the steep north-facing wall of the corrie. There are only small patches of ice present in this shaded part of the corrie today. The second feature is an unusual flow of cobbles on the valley floor, *c.*100m long (Figure 3.6). This flow is on a very shallow gradient (as low as <2°), originating from the northwestern side of the corrie. It resembles a series of solifluction lobes, but is much lower and longer than other examples found in the region.

Most of the glaciers in the catchment appear to be retreating, with gently-sloping ice fronts and moraine sequences in front of them, and very little snow left in the accumulation areas from the 2001 and 2002 winters (the southern glaciers are in Figure 3.7). One exception is the glacier to the northwest of Vatnsdalsjökull (Figure 3.8). This displays a steep ice front that descends directly into a small lake and onto a foreland with no evidence of terminal moraines, despite good preservation potential on this part of the foreland.

There are two major landslides within Vatnsdalur (figure 3.9). The larger landslide is located at the northern end of Vatnsdalur, and is the feature that dams Skeiðsvatn. The source of this slide is clearly the large scar on the northern end of the

Skeiðsfjall ridge, where a >550m high portion of the mountain has detatched in a catastrophic landslide. The debris from this slide has crossed the valley floor, and has flowed down into Svarfaðardalur where is lies as a chaotic series of hummocky ridges dissected by the tributary river. Following the damming event, Skeiðsvatn appears to have been twice its present size, and deltaic river inflow has filled in the southern part of the lake. Lake level readjustment after the landslide appears to have been rapid, as there is no evidence for substantial lake level lowering in the form of palaeoshorelines. The only shoreline above the present lake surface is c.0.5m higher and most likely relates to periods of high lake level during flooding.

The smaller landslide has occurred on the lower slopes of Skeiðsfjall. It appears to have been less catastrophic in nature. The origin of the slide is an arcuate escarpment 400m long, and the flow has extended 200m onto the valley floor. There is no evidence for present-day movement, and the southern portion of the landslide is slightly eroded by the main valley river.

Aeolian sediment sequences

Aeolian sediment sequences from inside, on top of, and outside geomorphic features help to constrain their age of formation. The relatively high altitude and varied geomorphic setting of the profiles has precluded the calculation of sediment accumulation rates between the sequences, as applied in previous studies (Dugmore and Buckland, 1991; Casely, 2001; Kirkbride and Dugmore, 2001b; Casely and Dugmore, 2004). The Veidivötn 871 (Landnám) tephra was not found, and H1104 was rarely found, and so the distinction between Settlement and post-Settlement sediment could not clearly be made. Hence the primary application of these, sequences was to constrain the chronology of geomorphic events in Vatnsdalur. The profile locations are in Figure 3.10, the key to these and subsequent profiles is in Figure 3.11, and the profiles themselves are in Figures 3.12-3.14.

One of the main targets in the analysis of aeolian sediments was the larger, lake damming landslide. The age of this landslide determines the maximum age of sediments accumulated in Skeiðsvatn. A profile was dug on one of the ridges on the southwestern side of the landslide, and is VDL-5 in Figure 3.18. The sequence includes two key indicator tephra, the thick layers of fine yellow and fine white tephra geochemically correlated with H3 and H4 tephras (samples VDL5-1 and VDL5-2 respectively). This gives the landslide a minimum age of 3800BP (the H4 tephra), but

the position of H3 and H4 nearly half-way up the profile strongly suggests that the landslide is early Holocene in age. There is no evidence for glacial modification of the slide, placing its formation age after glacial retreat from the Younger Dryas. The consequence of this is that the date of formation of Skeiðsvatn is in the early Holocene.

The second, smaller landslide located c.800m to the south of Skeiðsvatn is of geomorphological relevance, as it may have influenced the sediment input into the lake. A sedimentary profile taken from inside the outer lobe of the landslide indicates that the landslide was moving *c.*2800 ^{14}C years BP, as shown by the folding and thrust (compressive) faulting of the H3 tephra. There is little sediment beneath this tephra, suggesting that the slide took place shortly before tephra deposition. The tephra is overlain by a layer of convoluted sediments, and subsequently by smoothly laminated sediments, indicating cessation of large-scale movement by *c.*2200BP, assuming a constant sediment accumulation rate. If the convoluted sediment above H3 is the result of landscape readjustment following the landslide, the cessation may have occurred significantly earlier.

Profiles were dug in several locations to constrain the age of formation of the glaciogeomorphic features in the valley. Two profiles (VDL1 and VDL2) from within the arcuate moraines in the Kotafjall corrie contain both H3 and H4 tephras, indicating that these moraines are at least mid-Holocene in age. VDL3 from higher within the corrie shows greater evidence of disturbance.

In the tributary valley, profile VDL10 contains the H4 tephra (VDL10-1) inside the small arcuate moraine just beyond the main terminal moraine. It is possible that this position has been overridden by an early LIA advance, as the site is not presently vegetated and the soil profile may not be complete, but otherwise, this small moraine may be comparable to the pre-LIA Neoglacial moraines in front of Vatnsdalsjökull. VDL12 shows a typical sequence from within LIA limits: very shallow accumulation of aeolian silt and no tephras. VDL11 is located just inside the moraine fragment further down the tributary valley. As expected, the large mid-Holocene Hekla tephras are present, though the samples were not geochemically analysed.

In the main group of corries, the key section examined by Stötter (1991) was re-examined in order to check the stratigraphy and confirm the pre-LIA nature of the limit. The section has slumped, and presumably been substantially eroded laterally, since the visit of Stötter in 1991, but it is kept open by one of the proglacial streams

from Vatnsdalsjökull. Most of the same stratigraphy was found within the section, with the exception that slumping prevented the location of H5 at the base. VDL13-1 and VDL13-3 are geochemically consistent with H4 and H3 respectively, and VDL13-4 may be H1104 although the geochemistry is inconclusive and the layer has not been found in such thickness elsewhere. It is clear that the advance represented by the northernmost (left on Figure 3.2) moraine pre-dates H4. VDL8 was taken outside the outermost limits, but due to the location and depth of the section, the base was not reached. In a similar location, Stötter *et al.* (1999) refer to work indicating the presence of the Saksunarvatn tephra, though this information remains unpublished.

Sediment profiles VDL6 and VDL7 were taken <150m from the snout <150m from the glacier located northwest of Vatnsdalsjökull. There is no morphological evidence of advance positions between the profile and the present ice front. VDL6 contains the H3 tephra which (sample VDL6-1), as found visually in some other profiles has a distinctive dark layer immediately beneath it. Frozen ground prevented deeper examination of this profile, which is at an elevation of 900m, but it does indicate that this glacier has not advanced beyond its present position during the late Holocene.

Core Skeidsvatn III

The sediment core Skeiðsvatn III was extracted from the middle of Skeiðsvatn in a water depth of 1.9m. The core was extracted using a Russian corer with a 50cm long, 5cm diameter chamber. 5 sections of 50cm were taken, to give a total core length of 250cm. Due to time constraints and damage to coring rods, overlapping sections were not taken as originally planned. This leads to gaps at the top and base of the core sections due to disturbance of the sediments by the coring head. However, most of the core remains undamaged for analysis.

The core displays a straightforward sedimentary sequence, with the upper 65cm being grey laminated glacial silts, and the lower 1.85m being organic gyttja (Fig 3.15). Radiocarbon dating of $1cm^3$ of organic material immediately beneath the transition indicates an age of 1380 ±45 years BP, or 636-686 cal. AD to 1σ (GU-11230). The large H3 tephra does not appear in the sequence, constraining the maximum age of the profile base of 2800 ^{14}C BP. The damage to the corer occurred

in an attempt to penetrate a layer at 250cm, and this may reflect either a tephra layer or stiffer clay/silt sediments. If it is a lower layer of glacial silt, this may represent the advance prior to 400A.D. identified by Häberle (1991) in Barkardalur.

On of the first analyses attempted was the use of x-radiographs as a proxy for sediment density. This tool can clearly distinguish tephra layers and other large density changes within a core. The cores were x-rayed as entire 50cm core sections, and when these sections were digitised, subtle transitions were visible, as well as the more obvious stratigraphic changes. Unfortunately, the actual x-raying was carried out without calibration markers, and the 50cm x-ray segments could not be calibrated with each other for an indication of whole-core density change. Additionally, the slight thinning of the sediment section towards the ends of each core section leads to a darkening of the greyscale output near the core ends (visible in Figure 3.16). Therefore the x-rays were used individually to search for thin tephra layers and other sedimentological features of interest (e.g. Figure 3.16). No major tephras were found in the core, and one very minor layer was identified *c.*1cm above the gyttja/silt transition. Attempts to extract tephra grains from this minor layer using acid digestion failed, as no usable grains were found in the digested samples upon microprobe analysis.

Percentage weight loss-on-ignition was carried out on $1cm^3$ of dry sediment sample at 1cm intervals. The results are shown in Figure 3.15. The upper 65cm displays low LOI values of 3-5%. The visible textural change is highlighted by a sharp increase in loss-on-ignition from 4% to 11-12%. Loss-on-ignition then steadily decreases towards the base of the core, and is as low as 4-5% at the core base.

Low frequency magnetic susceptibility was also measured at 1cm intervals (Figure 3.15). Mass magnetic susceptibility was not done due to limited sample volume. Values (in SI units) are very high due to the basalt bedrock of the whole catchment. Once again, the largest change occurs at 68cm, with values decreasing from an average of 500 down to *c.*350. The large excursion around 34cm is due to a lump of organic material in the core, and is also responsible for the LOI excursion at this depth. Below 65cm, values increase gradually, reaching ~500 SI units at the base of the core, with one particular step around 165cm.

Grain size analysis was done at 5 and 10cm intervals (Figure 3.15). Throughout the core, the dominant grain size is that of silt (4 - 63μm). The visible textural transition from laminated clay to gyttja is less dramatic in terms of grain size change,

though there is a *c.*10% drop in the proportion of silt, balanced by increases in sand and organic fractions. There is a decrease in the proportion of sand from the base of the core towards the transition; after the transition the sand content remains constant. Individual graphs of weight percentage against particle size are shown in Figures 3.17 and 3.18. These highlight the shift in peak particle size towards finer grain sizes, but also shows the appearance of a small peak in coarser material (~0.2mm size), above the 65cm transition.

Diatom analyses on Skeiðsvatn III

Hill (2005) has carried out analysis of the diatoms within the lake in a collaborative study to this one in order to interpret the environmental history, as shown by the diatoms. Her findings are outlined below. The diatoms were sampled to at least 10cm resolution throughout the core, and the results are presented in Figure 3.19. Hill (2005) used principal components analysis (Figure 3.20) to characterise the diatoms of three distinct CONISS zones in the core:

Zone 1 (245-165cm): was characterised by low species diversity, and the relative abundance was dominated by the benthic *Pinnularia borealis*. Planktic species were also present, including *Aulacoseira distans* and *Cyclotella kützingiana*.

Zone 2 (165-65cm): benthic diatoms remained common, but there were increases in however there were some interesting shifts in the significance of planktonic Cyclotella species (*C. kützingiana* and *C. meneghiniana*), and acidophilous eunotioid diatoms (*E. exigua, E. praerupta var. inflata, E. tenella*) and Pinnularia taxa (*P. acoricola*), which both peaked between 95-105cm. Throughout this zone species diversity fluctuated substantially between <45 to ~90 species. Through this zone, the increase of these species, particularly the Eunotia species, suggest an increase in acidity associated with increased catchment biomass. One possibility that would be quite likely in this case is an increase in moss cover at higher elevations (e.g. Douglas *et al.*, 1994; Wolfe, 1995). The peaking in planktonic *Cyclotella* during a time of elevated acidity suggests that the diatoms in Skeiðsvatn are not sensitive to acidity changes (as these species would otherwise decrease at this time; Battarbee *et al.*, 1999).

Zone 3 (65-0cm): The transition from zone 2 to zone 3, at 65cm, signalled major changes within the diatom flora. The planktonic diatom population as a whole rapidly began to decline, falling to ~1% by 55cm. In contrast, benthic species increased in

significance, particularly small benthic species of *Fragilaria,* which represented 12.5% of the total diatom species at their peak at 45cm, shortly following a peak in the relative abundance of other benthic species. At 55cm several new species were introduced; *Surirella linearis, Surrirella. ovata* and *Amphora pediculus;* and the relative abundance of a number of existing benthic taxa increased dramatically (*Cymbella ventricosa*, *Diatoma hyemale* var. *mesodon* and *Navicula contenta*). These species remained significant throughout the remainder of zone 3. Conversely, a number of other benthic species declined in abundance throughout zone 3. The *Fragilaria* species have low light requirements and are likely to have been successful in turbid water and/or waster that is more often covered by ice.

Two further observations by Hill (2005) are noteworthy. There is a dramatic proliferation of diatoms in zone 1 (Figure 3.21), indicative of increased nutrient content and diatom productivity. Most studies show that proglacial lakes are unfavourable for diatom productivity (e.g. Hickman and Reasoner, 1998; Cremer *et al.*, 2001; Karst-Riddoch, 2003); however in Iceland, the situation appears reversed, with nutrient levels increasing following increased glacier activity. The same situation has been seen in glacigenic sediments from Hvítarvatn by Alex Wolfe (pers. comm.), so appears not to be a phenomenon isolated to Skeiðsvatn. The second observation is of the relative abundance of *Aulacoseira distans*, which does not decline as expected into zone 1. Several studies indicate that this species is more prevalent where LOI values are higher. Hill's explanation for this is that the species is responding to light level rather than organic content.

III.2.2: Hraunsvatn, Vatnsdalur in Öxnadalur

Presented here are the results from geomorphological and sedimentological work carried out in Vatnsdalur in Öxnadalur. This location is a potential site for lacustrine sediment coring, but was rejected in favour of Skeiðsvatn. A lichenometric study was carried out by Barr (2003) on the moraines and rock glaciers at the snout of the very small glacier in this valley. The geomorphology is shown in a map drawn by Barr in Fig 3.22. Sedimentological work was carried out in parallel with this study in order to attempt to constrain the age of pre-LIA moraines and a pre-LIA rock glacier in the valley. Additionally, as the lake is once again dammed by a landslide, a profile was

taken from the surface of the slide to determine a minimum age for lake formation. Six profiles were examined (HV1-5 and HV7), and are shown in Fig 3.23.

The geomorphology of the valley is straightforward. Immediately in front of the small glacier there is a large rock glacier. Lichenometry carried out on this rock glacier suggests that it was last strongly active in the early 20th Century (Barr, 2004). The rock glacier is clearly of the proglacial type, as large amounts of glacier ice are visible in hollows that are melting out at the present day. The rock glacier margin is steep, suggesting a small amount of activity continues to the present day. Beyond the rock glacier there are several lateral and terminal moraines. Tephra within aeolian sediment sequence HV3 confirms that the moraines are pre-LIA in age. It is located within a beaded arc of terminal moraine fragments, and contains two silicic tephras. The lower tephra is geochemically consistent with the H4 tephra. indicating a minimum age for these moraines of *c.*3800BP. The are likely to be older than that, given the quantity of sediment beneath the moraines, and the fact that the base of the section was not reached. The extent of the glacier traced out by these terminal/lateral moraines is considerably larger than that of the LIA or the present day, and it may be that this position represents an early Holocene readvance.

A profile was dug into the inactive rock glacier located to the northeast of Vatnsdalsjökull. HV7 shows two silicic tephras, geochemically consistent with H3 and H4 respectively. The position of H4 at the base of profile HV7 immediately on top of the surface of this rock glacier strongly suggests that this rock glacier was last active *c.*4000years BP, shortly before deposition of the tephra. There is no subsequent disturbance of this aeolian sediment sequence, indicating stabilisation of the land surface (and rock glacier) prior to the deposition H4. Moraine fragments close to, and just to the north of, the rock glacier must represent an early Holocene position of the main valley glacier, given the location of the rock glacier.

Two profiles were examined just beyond the glacial limits (HV1 and HV5), but could not constrain the maximum age of the glacial advances in the valley. HV5 was the only profile that clearly contained the Hekla 1104 tephra of all the profiles dug in Tröllaskagi (a thin fine white tephra in the section), and contains the H3 and H4 tephras. The sediments in this section showed no evidence of disturbance, suggesting that throughout this period glacier ice was not proximal, and the nearby slopes were not destabilised.

The profile HV4 was taken on the proximal (northwest) slope of the large lake-damming landslide. The sequence shows some evidence for disturbance, with cryoturbation near the top and visible thufur on the surface), as well as repetition of H3 tephra lower in the sequence. The H4 tephra was not found, and on this assumption, the landslide age may be constrained to between 3800BP and 2800BP. However, this is on the basis of a single disturbed profile and further work would need to confirm this age. The lake is likely to be >3000 years in age however.

III.3: Southeastern Iceland

III.3.1: Introduction

The key region of southeastern Iceland is the area around Hofsjökull í Lón. This includes the valleys Geithellnadalur and Hofsdalur, and the lake Hofsvötn. The region contains the transition between the inland plateau of Iceland (at the western side of the area) and the coastal fjords. Large glacially-cut valleys are incised into this trough, and in this area trend northwest to southeast. Mountains in the region reach a maximum elevation of 1343m, above the mean plateau elevation of 800-900m. Only the highest mountains are presently glaciated: these include the two small plateau ice caps of Hofsjökull í Lón and Þrandarjökull, and a short corrie glacier on Jökulgílstindar.

This section will present the relevant environmental data from geomorphological investigation in this area: III.3.2 comprises a discussion of the Hofsvötn lakes and III.3.3 presents geomorphological and sedimentological results from Geithellnadalur and surrounding areas.

III.3.2: Hofsvötn

Sediment cores were taken from Hofsvötn in winter 2003. The location of the cores taken is shown in Figure 3.24. Cores were extracted from the middle of Hofsvötn 'A' and 'B', and the longest core extracted was 1.4m in length from 13 and 8m of water depth respectively. During coring it was immediately apparent that the cores had reached the base of the lake sediments, as it was impossible to drive the core tube deeper, and the tube end was shattered on core retrieval, indicating that rocks were encountered.

Once the cores were opened, the sedimentology could be examined. Both cores showed clear laminations, composed of repeating varve units (coarse summer deposition and fine winter deposition; Figure 3.25). The varve units are dominated by the coarser dark brown summer deposition, and the paler winter deposits are usually thinner. Approximately 119 lamination pairs are visible in Hofsvötn A1, as defined by paler/darker laminations. This number is approximate, as some of the paler laminations are indistinct. There is an enhanced frequency of very well-defined laminations towards the core base, and fewer of these are present towards the upper

part of the core. These laminations are considered to be annual varves, and are nearly identical in structuare and appearance to those found by Black *et al* (2004) in the upper part of Hvítárvatn. As in Black's study, the interpretation is of the finer, paler clay material being deposited during winter when the lake is frozen. There are several occasions where the strongest laminations occur in closely spaced pairs. Such features have been seen in a varve sequence from northern Canada by Tomkins *et al* (2004), and have been interpreted as multiple freezing events in a single winter.

Magnetic susceptibility

The cores taken from Hofsvötn A and B have enough material to allow measurement of mass magnetic susceptibility, and allows the calculation of frequency dependent magnetic susceptibility using the Barrington MS-2B sensor (Walden *et al.*, 1999). The two instruments used to measure magnetic susceptibility in this thesis are compared in Figure 3.26: values obtained with the MS-2 sensor are directly comparable to those obtained using the MS-2B sensor, with the greatest variation being due to the sampling resolution obtainable using the MS-2B instrument.

Low frequency susceptibility (χ_{LF}) measurements using the MS-2B were taken at 0.5cm resolution in the core in order to more closely correspond to the laminations within the core (sampling resolution thus approximates to a 6 month time resolution). Unfortunately, layer distortion caused during coring reduces this somewhat, though an attempt was made to correct for this during sampling by sampling along the layers. In particular this was attempted towards the base of the core where the laminations are strongest. It was found that there was a poor relationship between χ_{LF}, and individual layers, suggesting either that the sampling resolution was insufficiently fine to distinguish these layers, or that the values if χ_{LF} were not varying on a subannual basis.

The overall pattern defined by the χ_{LF} is one of a decline from mean values of 600 SI units to values of ~440 SI units, along with a reduction in the variability of the values. These values are comparable to those found at Skeiðsvatn in the region of glacial silt, and would be expected if (assuming similar chemical composition of the underlying lithology) the catchment is reflecting deposition through active glaciation.

Frequency dependent susceptibility can be used to determine the relative quantity of ultrafine ferrimagnetic grains in a sample – high values of χ_{FD} indicate more of these superparamagnetic grains in a sample. Values from Hofsvötn A1 are shown in Figure 3.27, and are in the medium category according to Walden *et al.*, (1999),

indicating a reasonably even mixture of superparamagnetic grains and 'normal' multidomain grains (larger than 110μm). There is a gradual increase in frequency dependent MS (χ_{FD}), which may merely reflect a gradual decrease in the quantity of grains >110μm being deposited in Hofsvötn A1.

Low frequency susceptibility values from Hofsvötn B1 (the southeastern and more 'downstream' of the two lakes) has very little range over the period of the core (Figure 3.28). This may imply that Hofsvötn A1 is filtering out the signal of change from the main glacier outwash stream from Hofsjökull. This would be the case if the sediments in Hofsvötn A1 include higher proportions of non-magnetic coarser material, which increases in the upper part of the core. It may also suggest that the dominant grain size for the magnetic material is in the fine/ultrafine grade (but not wholly ultrafine), and is therefore deposited in both lakes, while the coarser non-magnetic material is primarily deposited in Hofsvötn A1.

Loss on ignition

LOI was calculated for samples from Hofsvötn A1 (Figure 3.29). Unfortunately an entire sample batch was lost during processing, and so there is a gap in the sequence. Relatively low values throughout were expected, as there is relatively low organic productivity in the catchment at 700m. The values that were calculated do however show an interesting pattern. The values are low towards the base of the core, and continue at comparable values to 22.5cm, where there is a sharp increase in LOI of *c.*2%. The top portion of the core represents approximately 30 years, so the higher LOI values correspond to the most recent 30 years in the record (since 1973). It is possible that the LOI values are responding not to glacier activity, but to organic productivity in the catchment.

Geithellnárvatn

A additional sample was taken from the lake Geithellnárvatn, which is outwith the potential glacial catchments of Hofsjökull and Þrandarjökull, and so can be used as a comparative study (Figure 3.30). The stratigraphy of this core is greatly disturbed, probably as the water depth was <1m, but should contain material representative of an unglaciated catchment. The age represented by the core is undetermined as there are no clear laminations as in the Hofsvötn cores. Magnetic susceptibility measurements on this core can be used as a material comparison to those obtained from Hofsvötn A

and B. The values are all noticeably lower, with an average of <200 SI units. This suggests that a greater concentration of magnetic (most likely magnetite; Walden *et al.*, 1999) minerals is being produced during phases of higher glacial erosion at Hofsjökull. In combination with the χ_{FD} results, this would indicate that fine grains (<110μm) produced by the glacial erosion process are responsible for the higher values.

X-rays of Geithellnárvatn 1 determined that there was one tephra layer present at Geithellnárvatn, but is of little value in a disturbed core. This tephra is the feature that produces the spike in χ_{LF} towards the base of the magnetic susceptibility graph (Figure 3.30).

Summary of results for Hofsvötn A and B

It was quickly apparent that the sequences in the Hofsvötn lakes represent a short period of accumulation, and are not pre-LIA in age and certainly do not extend into the Medieval. No tephra layers were found within the Hofsvötn cores (as determined by examination of X-rays), so additional constraint could not come from this source. The laminations present in the cores are most likely annual varves, and therefore layer counting places the age of the sequence at ~120years for Hofsvötn A1. As such, further investigation of these cores has been limited, with the aim being to utilise the cores to verify the magnitude of magnetic susceptibility readings obtained at Skeiðsvatn, and to determine the scale of recent change at the Hofsvötn/Hofsjökull system. These will be discussed in Chapter V.

III.3.3: Geithellnadalur and Hofsdalur

These two major troughs extend 20km inland from the coast, reaching the two plateau ice caps (Figure 3.31). At the present day, the vegetation cover is relatively complete up to *c.*600m, and fragments of birch woodland remain in both valleys. Birch woodland is particularly prevalent in Hofsdalur, with large stands of birch with a maximum height of 5m covering the upper part of the valley (Figure 3.32). Two abandoned farms are present in the upper part of Geithellnadalur, Hvannavellir (figure 3.33) and Þormóðshvammar, of unknown age. Present-day farms are close to the coast, with grazing of sheep and horses inland.

There has been one published record of farm settlement and abandonment in the region, that of Sveinbjarnardóttir (1991). The relevant part of Sveinbjarnardóttir's

work was carried out on farms in the region of Berufjörður, and in particular the valley of Fossárdalur that rends inland from the western end of Berufjörður. Abandonment of two sites in the inner part of Fossárdalur, at Móar and Engihlíð is constrained by the presence of the black Veidivötn 1477 (V1477) tephra and beneath it the Öræfi 1362 tephra. These were not interpreted as permanent settlements by Sveinbjarnardóttir, as four layers of charcoal and burnt turf indicate four separate episodes of settlement (or of use). The site was abandoned well before 1362.

The geomorphology of Geithellnadalur is controlled by the geology of the area. The area is dominated by the stacks of Tertiary basalt lavas that dip gently to the southwest, Erosion of these basalts give a terraced appearance to the landscape, with flat meadowlands on the valley floor separated from each other by the subsequent basalt lava flow that crosses the valley floor (Figure 3.34). Their resistance to erosion is highlighted by the number of impressive waterfalls on the large main valley river, the Geithellnaá. The consequence of the geology for settlement is that there are relatively few good flat areas for settlement, and the two abandoned farm sites lie in two of the only such sites for several miles on either side. The valley sides are similarly arranged, and there are some large cliffs and rock steps highlighting the more resistant sequences of lava flows. There are many large alluvial fans on the valley sides along Geithellnadalur and Hofsdalur. One particularly well developed fan is the location of the larger of the two farms, Hvannavellir on the southern side of the Geithellnaá (Figure 3.39). The site is covered with lush grassland today, and The smaller of the two sites lies on a smaller, more eroded fan on the northern side of the Geithellnaá.

There is very little visible glacial geomorphology within either Geithellnadalur or Hofsdalur. The locations with respect to present glaciation and the sharp elevation change from the plateau to the valley floor suggest that the valleys would only be glaciated during a major advance, beyond those experienced during the Holocene. Several possible moraines were observed high on the northern side of Geithellnadalur, and a small terminal moraine fragment exists in the upper part of the valley. These most likely represent a minor readvance during retreat from Younger Dryas limits (when ice reached the coast; Norðdahl and Einarsson, 2001), and have no bearing on the Holocene environmental history of the valley. Holocene geomorphological activity appears to be limited to fluvial activity and alluvial fan development, which is widespread in both Hofsdalur and Geithellnadalur.

Tephra stratigraphy in Geithellnadalur

Geithellnadalur and Hofsdalur were chosen as a site to investigate the tephrastratigraphy and rates of regional sediment accumulation. 27 profiles were examined from Geithellnadalur (map: Figure 3.35; profiles: Figures 3.36-3.42). In contrast to the sites in Tröllaskagi, there is a good local tephra stratigraphy for the Late Holocene and historical periods, with several key indicator tephra layers of relevance to Medieval settlement and abandonment. The valleys are some distance from the nearest volcanic centres and so all the tephra layers are a fine grain size, but as the site lies downwind from tephra blown on westerly winds many layers are present.

The most notable tephras are Veiðivötn 1477 and Öræfi 1362. V1477 is the largest eruption of tephra during the Holocene, and in this part of Iceland it is frequently as much as 5cm thick, and is distinct from some other 'black' tephras in having a slightly greyish colour. Ö1362 is a very distinctive fine white tephra from the major eruption of Öræfi that devastated parts of southern Iceland. It is sufficiently distinctive not to require further geochemical identification (Sveinbjarnardóttir, 1991) though samples were taken anyway. A further identifying characteristic in many profiles is a thin fine black tephra from an eruption of Grimsvötn in 1352 which consequently appears directly beneath Ö1362. The region lies on the margin of the fallout area of the V871 (Landnám) tephra, and the prehistoric H3 and H4 tephras may also be present. The characteristic two part Landnám tephra structure, formed from an olive-green silicic and a white basaltic component to the eruption, was clearly seen in GET-30, though in other profiles the silicic component was not visible.

There are also a number of dark tephras within the sequences. These have dominantly come from the Katla and Grimsvötn volcanic systems, and some of the layers are identifiable. The K1755 tephra has been found in the region (Sveinbjarnardóttir, 1991) as a ~5mm thick layer, and the K1625 layer may also be present.

Two dark layers were often observed beneath the Ö1362 tephra, and above the Landnám tephra. These were a distinctive 'grainy' layer, often brownish in appearance with grains up to 1mm diameter, below a blue-black layer. In the cases where Landnám was positively identified in the profiles (GET27 and GET30), sediment accumulation rates for this period could be determined. could be used to determine the age of these tephras (Howard, 2005; Figure 3.43). The SAR of

0.07mm/a was then used to estimate the age of these tephras in all profiles in relation to Ö1362. The estimates came out at around 1260A.D. for layer 'K' and 1150A.D. for layer 'G'. Layer 'K' is thus close in age to the Katla 1262 eruption, and 'G' to the V1159 eruption, whose fallout was likely to have reached the eastern fjords (A. Newton, pers. comm.; Howard, 2005). Consequently both in texture and in age correlation, these layers are likely to have originated from those two eruptions. The geochemistry of a sample from layer 'G' was also analysed, and supports Howard's argument, as the tephra is geochemically similar to, but not identical to the Landnám layer. The addition of these two tephras to the sequence of dated tephras gives an excellent sequence of historical-age tephras to use in Geithellnadalur. Some of these layers have also been correlated with layers found in neighbouring Hofsdalur.

The 27 profiles in Geithellnadalur have been taken from a range of settings within the valley, all within a 4km stretch to the west of Þormóðshvammar, and from a range of elevations between 137m and 705m. This allowed the examination of a variety of questions relating to the spatial distribution of sediment accumulation patterns. This in turn could help explore spatial and temporal questions relating to slope disturbance and erosion within the valley.

The aeolian sediment sequences were used to investigate the spatial distribution of sediment accumulation rates in the valley (SAR). Few profiles showed the full stratigraphy, and Table 3.2 shows the number of profiles considered to contain each layer, subsequently used for the SAR calculations. The overall SAR is presented in Figures 3.44. Regional patterns as presented by Howard (2005) are in Figure 3.45, and the altitudinal data is presented in Figure 3.46.

Overall, SAR increases noticeably following Landnám, as has been observed elsewhere. The highest SAR is 0.079mm/a, between 1262 and 1362, following which there is a sharp 43% drop to 0.045mm/a. The SAR then gradually increases towards the present day, but doesn't attain the high values of the 14th Century.

On the hillside above Þormóðshvammar, there is a very distinct peak in SAR occurring in the 13th Century, which was observed in 3 of the 4 profiles from that transect extending sufficiently far back in time. On the south side of the valley, and in the profiles from close to Þormóðshvammar, the highest values occur before 1262A.D., and close to Þormóðshvammar, the peak occurs between Landnám and 1159A.D.

The altitudinal distribution of SAR shows a distinct pattern. The most pronounced increase following Landnám appears to occur on the lowest ground, while the SAR peaks later on higher ground. However, this is an artefact of the construction of the SAR diagram, as the profiles in which Landnám was identified at low level (GET9 and GET10) did not contain V1159 or K1262, and so the peak may actually have occurred later, as seen at higher elevation. The SAR peak is pronounced above 200m, with a maximum value of 0.125mm/a between 1262 and 1362A.D. Also at this level, there is a significant rising trend towards the present day.

Additional reference profiles were dug in the area between Geithellnadalur and Hoffelsjökull (Figures 3.47-3.49) in order to compile a regional tephrastratigraphy for this area. The profile locations are shown in Figure 3.31, and cover the area between Höfn and Djúpivogur in southeastern Iceland. These profiles have not been analysed in great detail, but have been used for reference and aid in the understanding how the local tephra stratigraphy (for which there is very little present data) connects with the regional information from areas to the north and west.

Location	Age of advance		Source
	Minimum	Maximum	
Vatnsdalur	4700±205BP	*c.*6000BP	Stötter (1991)
Grænavatn cirque	*c.*3830BP	*c.*5950BP?	Kirkbride and Dugmore (2001)
Vatnsdalur		<3470±160BP	Stötter (1991)
Barkardalur	1835±120BP	2240±120BP	Häberle (1991)
Barkardalur	>1555±90BP		Häberle (1991)

Table 3.1: Dates of pre-LIA Neoglacial advances in Tröllaskagi

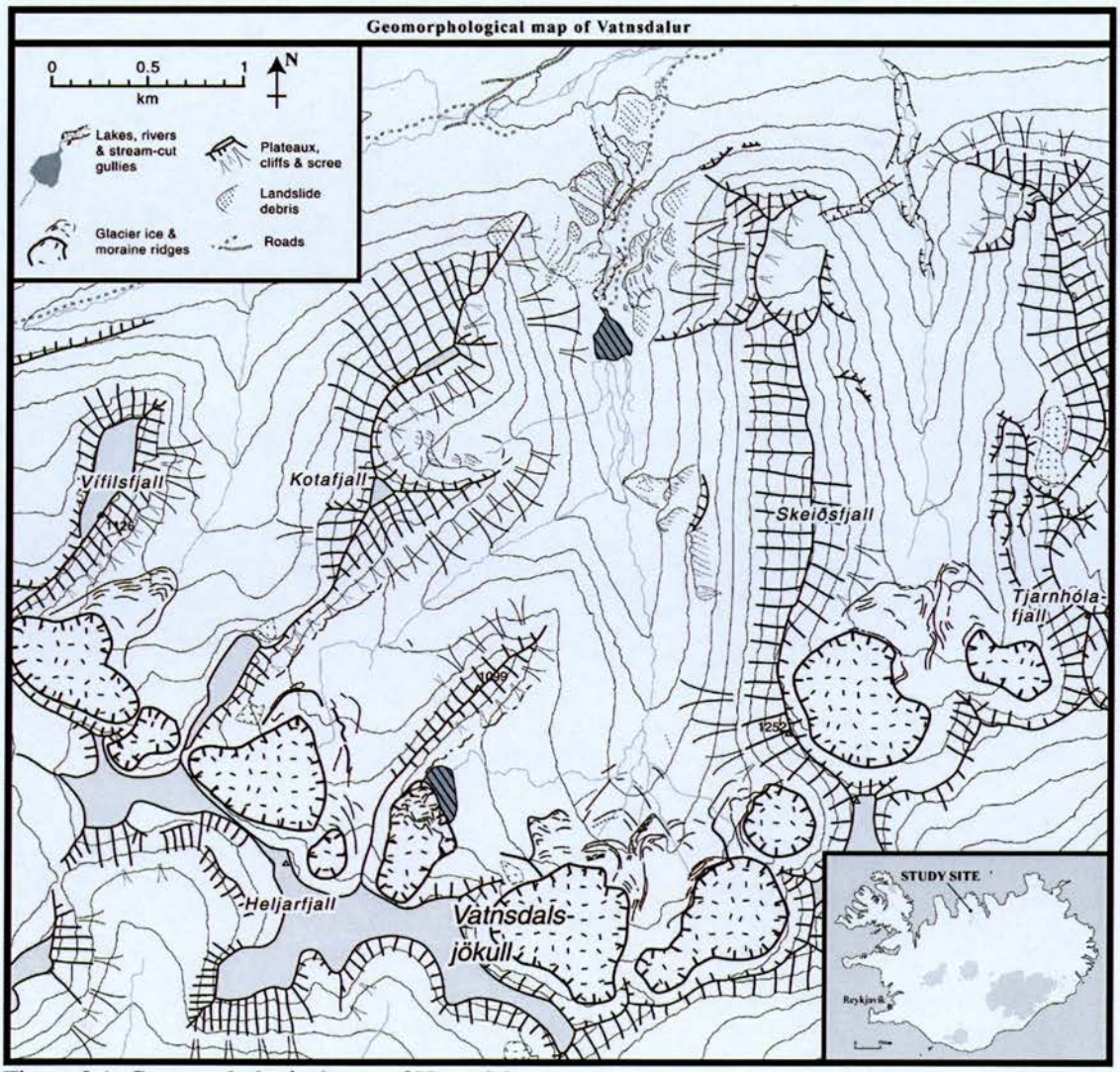

Figure 3.1: Geomorphological map of Vatnsdalur

Figure 3.2: Location of profile VDL13, and the site of the Vatnsdalur I and II advances reported by Stötter (1991). The profile is inside one moraine (rocks on the left), outside a second (rocks incorporated in soil on the right), and <*c*.3m from the LIA moraine limit (unvegetated rocks on the far right).

Figure 3.3: Panorama of the Vatnsdalsjökull foreland. Arcuate retreat moraines are visible, as is the transition between the vegetated pre-LIA foreland and the LIA foreland. The eastern rock glacier is showing the steep northern margin, with an apparent flow from the right (southwest).

Figure 3.4: The western rock glacier, which originates at the western margin of Vatnsdalsjökull. It flows across the Neoglacial moraines and onto foreland that appears to be unglaciated during the Holocene.

Figure 3.5: Panorama of the southwestern tributary valley of Vatnsdalur.

Figure 3.6: The interior of the Kotafjall corrie. The periglacial stone flows are in the foreground, and one of the terminal moraine fragments is on the right hand side of the photo.

Figure 3.7: Evidence of retreat – small retreat moraines formed within the last 20 years, and gently sloping ice fronts. The eastern rock glacier is on the right, and Vatnsdalsjökull is in the background. To the right of one of the small lakes is a cairn placed by Hans Stötter.

Figure 3.8: Evidence of glacier advance - the steep margin of the glacier northwest of Vatnsdalsjökull (note Teresa for scale on the right!). This is in marked contrast to the retreating margins in Figures 3.3 and 3.7.

Figure 3.9: Landslides in Vatnsdalur. The scar left by the large landslide that dams Skeiðsvatn is visible on the skyline to the right of the lake (partially behind the cloud), as is the debris mound behind the lake. In the foreground is the small landslide on the slopes of Skeiðsfjall.

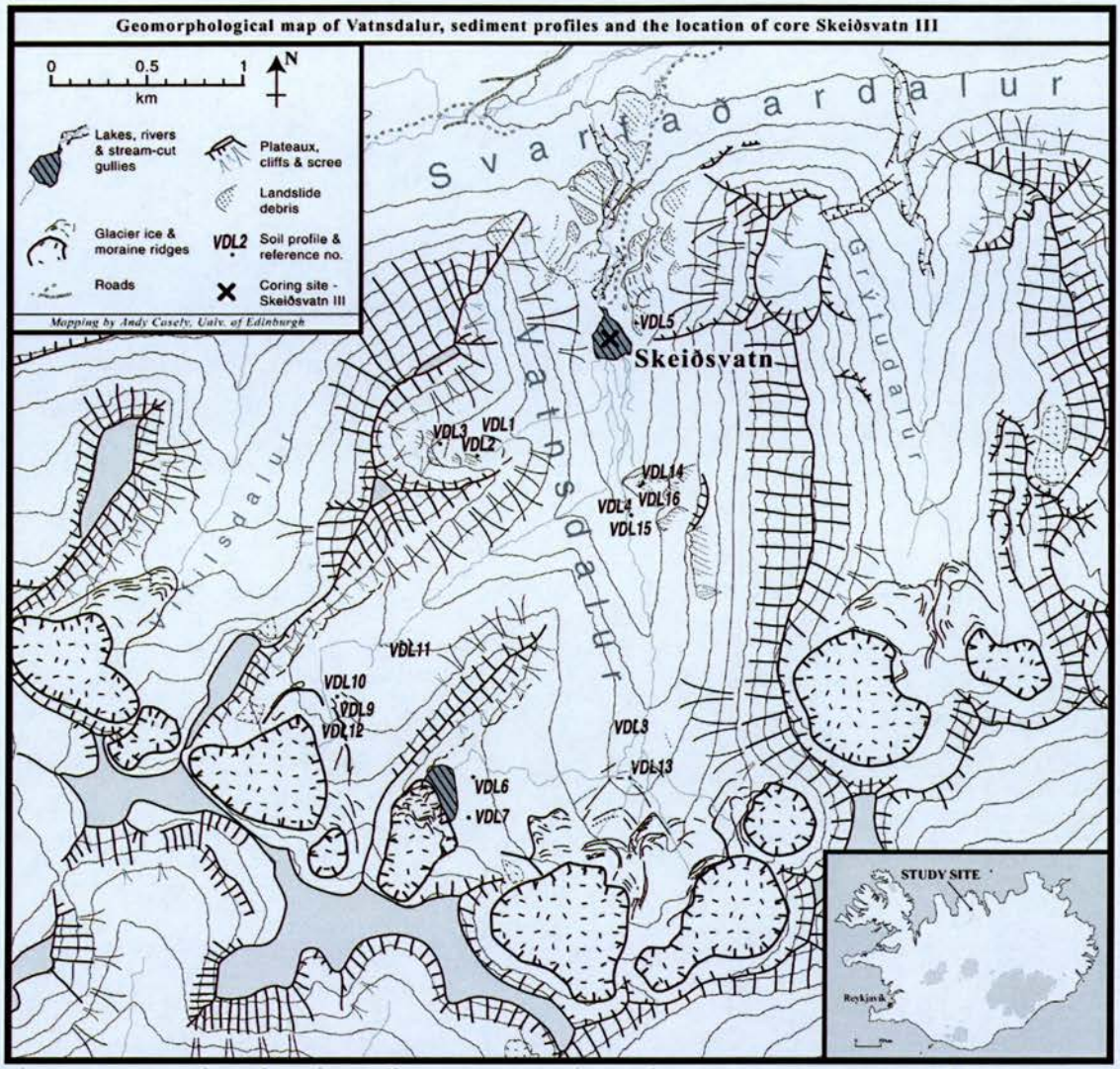

Figure 3.10: Location of aeolian sediment sequences in relation to the geomorphology of Vatnsdalur.

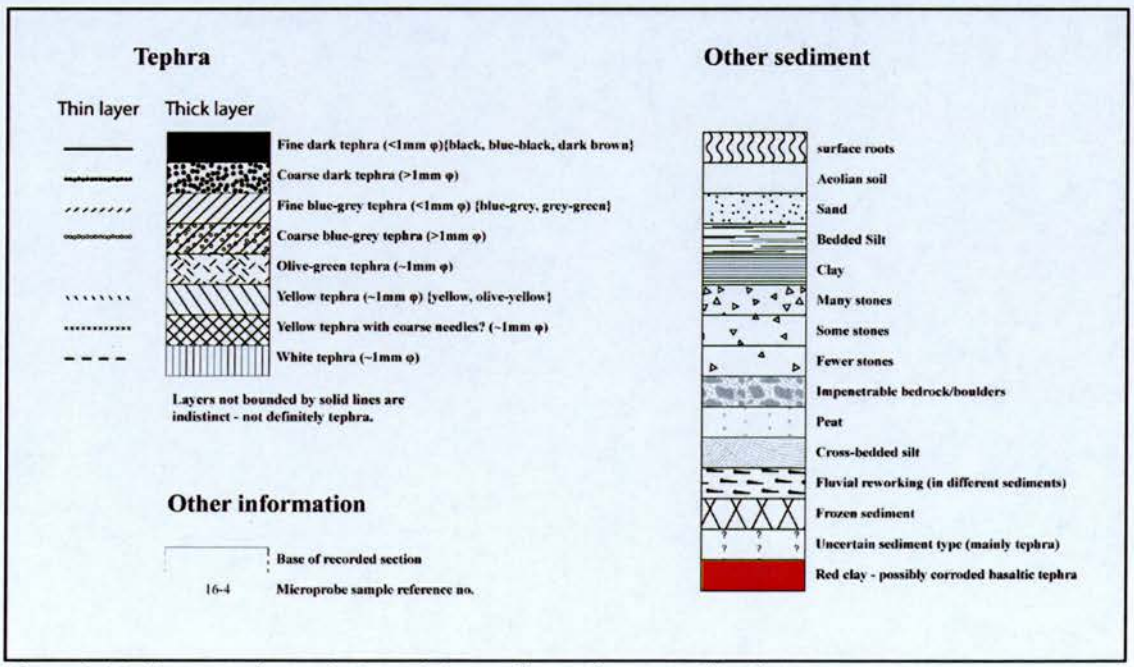

Figure 3.11: Key to main symbols used for aeolian sediment profile diagrams.

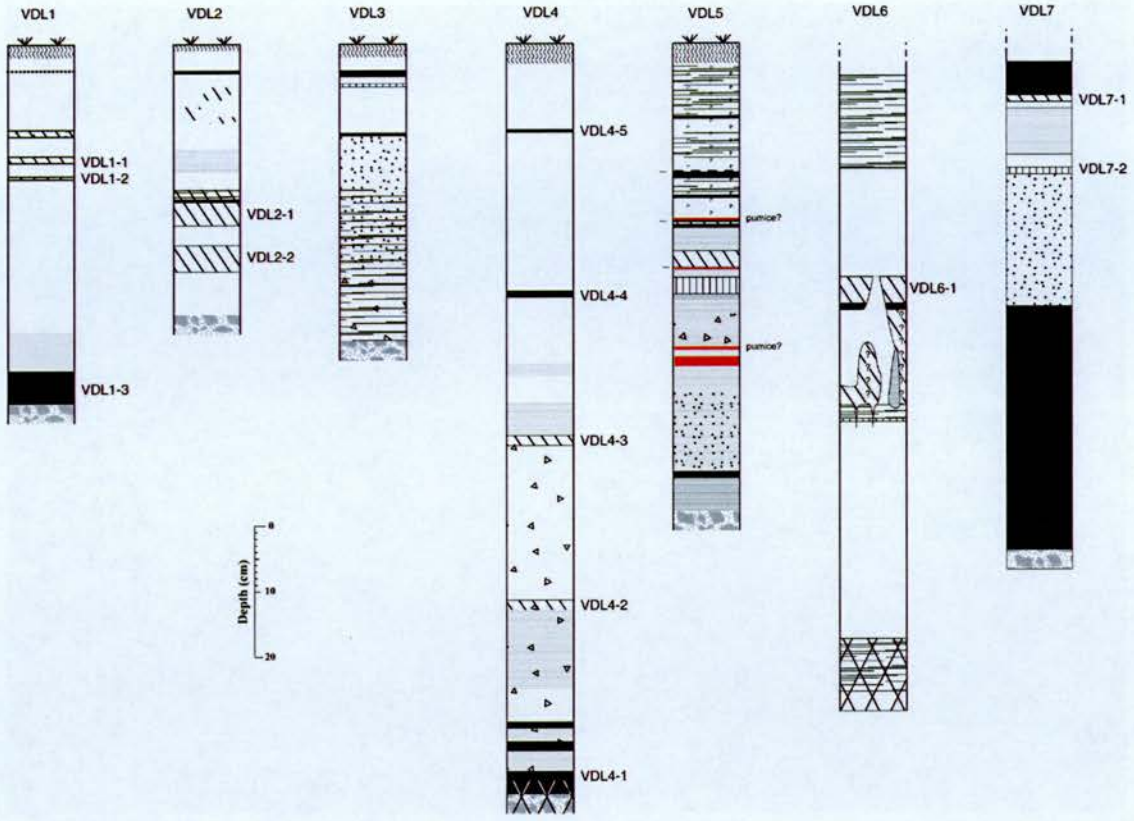

Figure 3.12: Vatnsdalur aeolian sediment sequences, VDL1-VDL7.

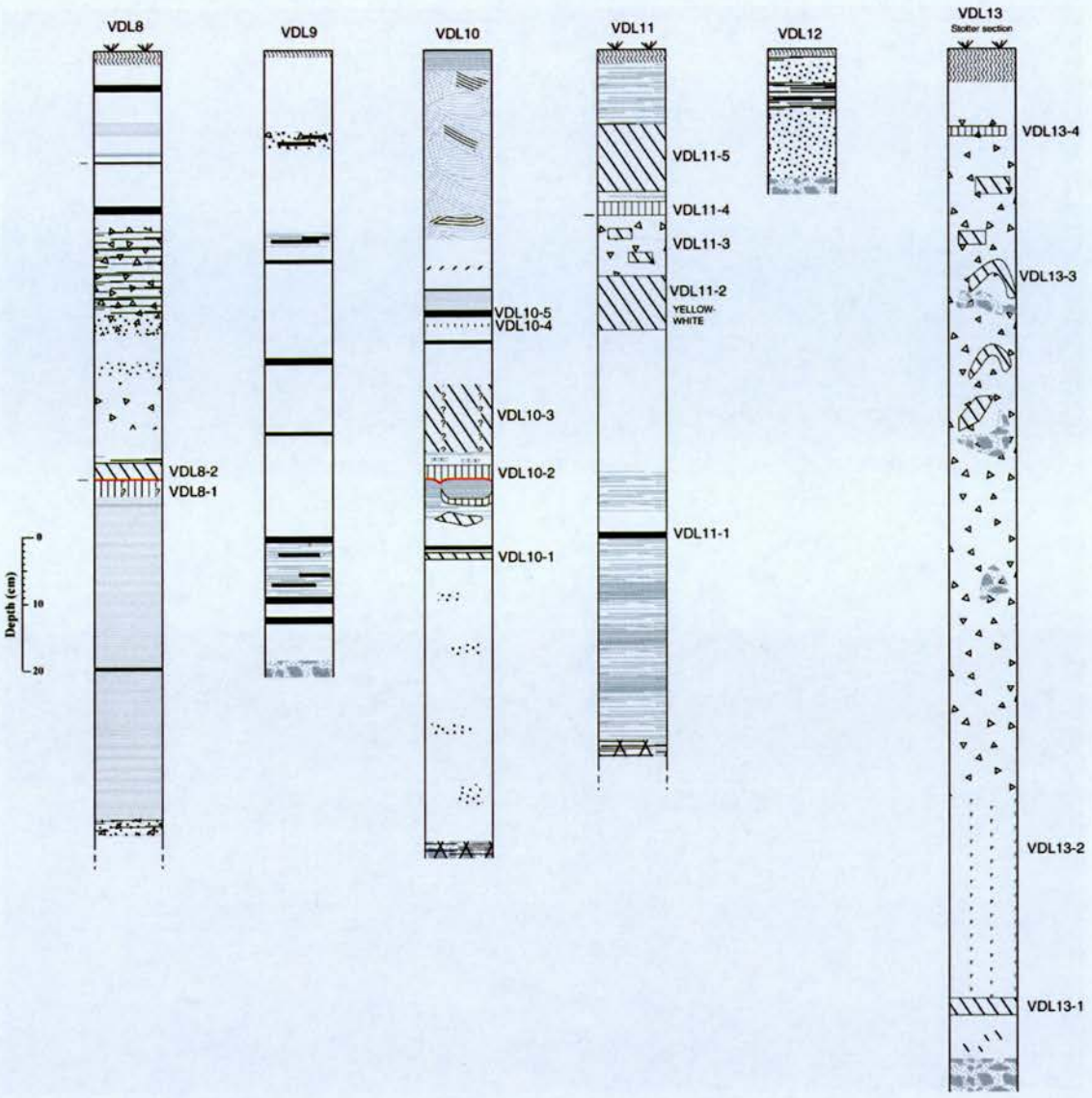

Figure 3.13: Vatnsdalur aeolian sediment sequences, VDL8-VDL13.

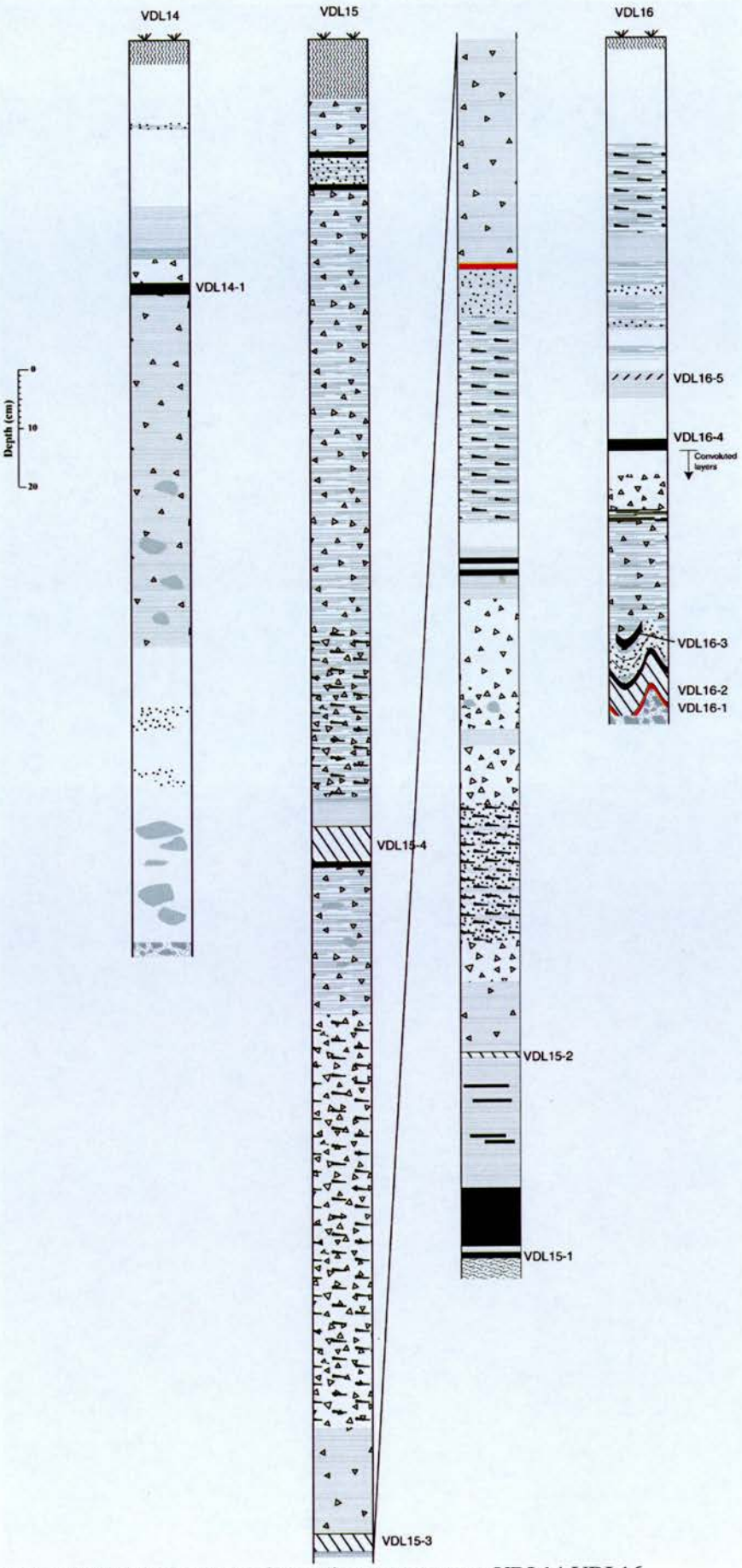

Figure 3.14: Vatnsdalur aeolian sediment sequences, VDL14-VDL16.

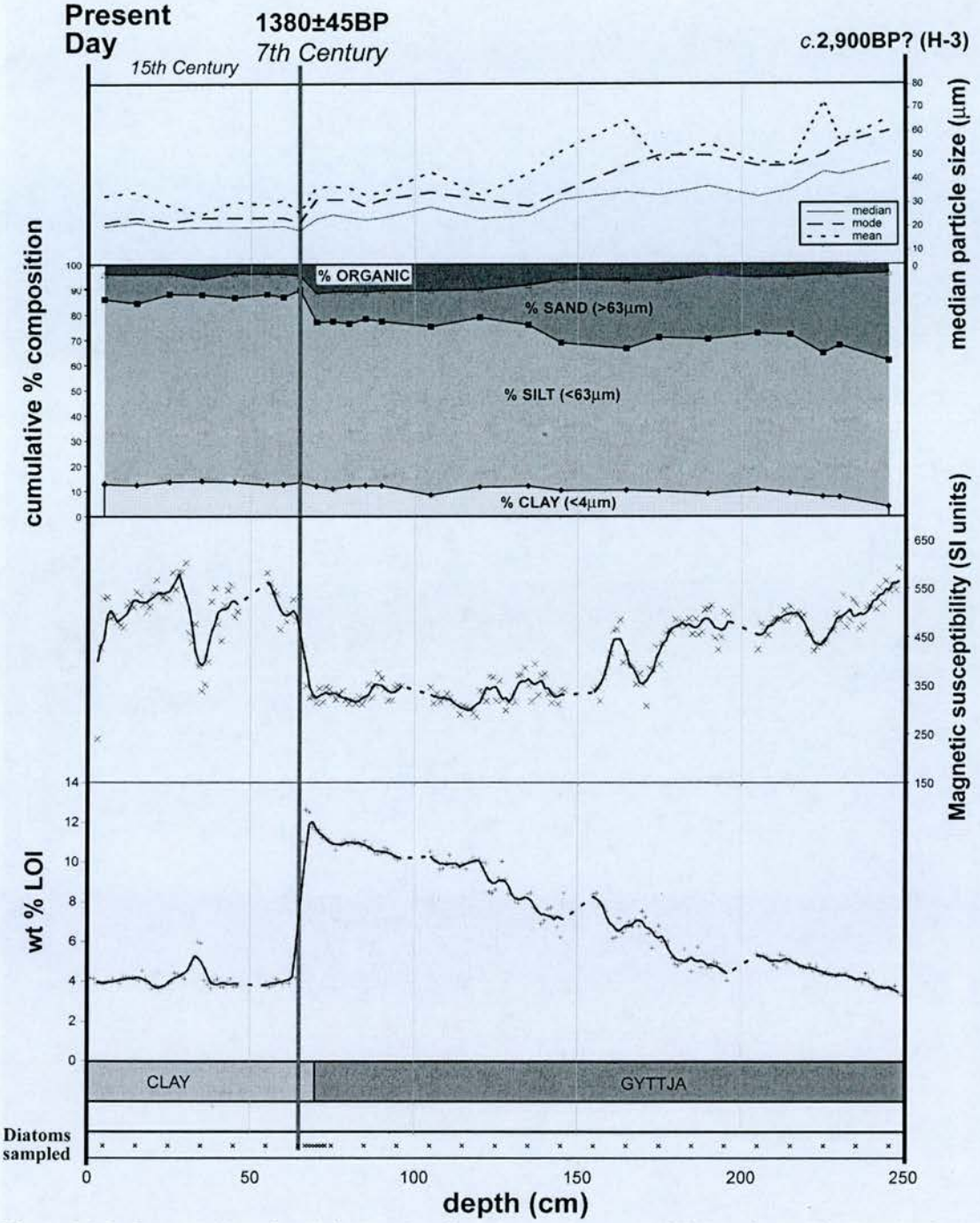

Figure 3.15: Core stratigraphy, % loss-on-ignition, magnetic susceptibility and particle size data from the core Skeiðsvatn III.

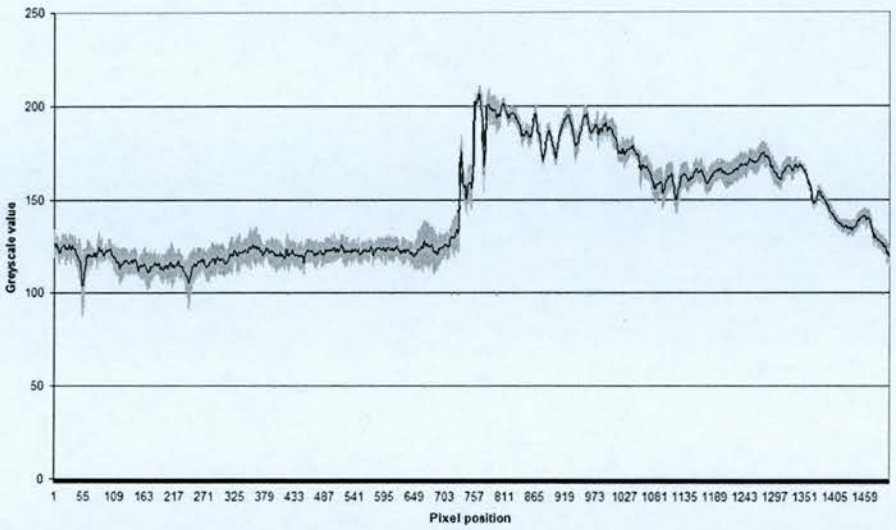

Figure 3.16: Greyscale analysis of an x-ray of the 77cm-50cm section in Skeiðsvatn III, covering the gyttja-silt transition (core base is to the left). The x-axis values are the pixel positions between 50cm and 77cm A greyscale value of 255 represents maximum density (white), and a value of 0 represents minimum density (black). The transition from gyttja (GS = ~120) to silt (GS = ~180) is clearly visible, as is the high resolution obtainable with X-radiography. The lowering of values to the right is an unavoidable effect of thinning of the core section.

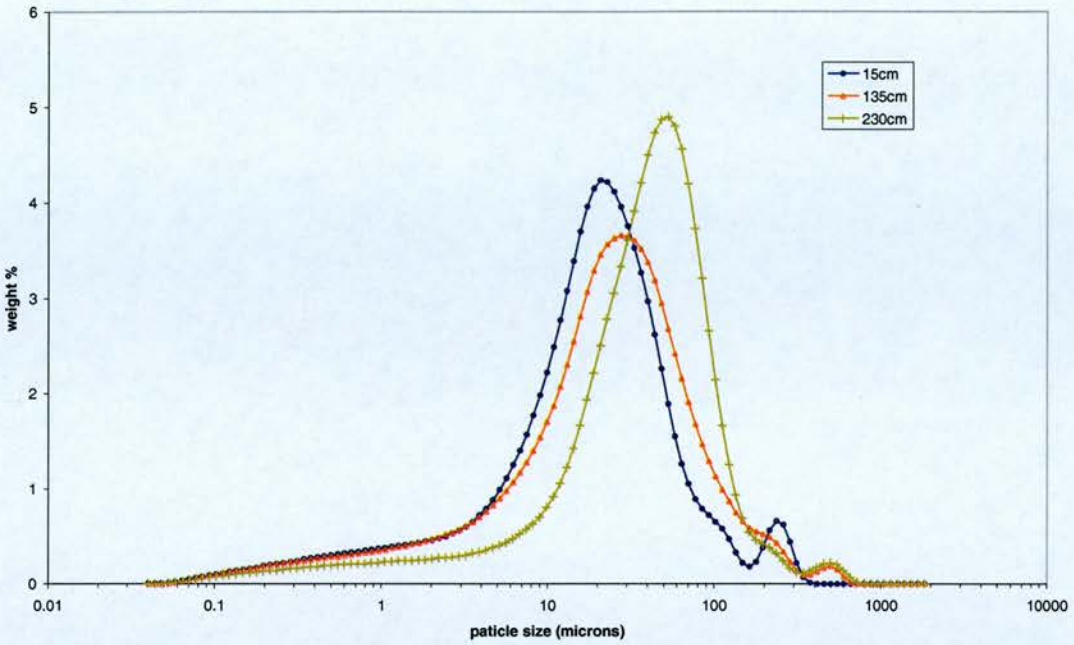

Figure 3.17: Particle size percentage frequency graphs for 3 selected depths – 15cm, 135cm and 230cm. The greatest similarity is between the 15cm and 230cm graphs, notably the subsidiary 0.2mm grain size peak.

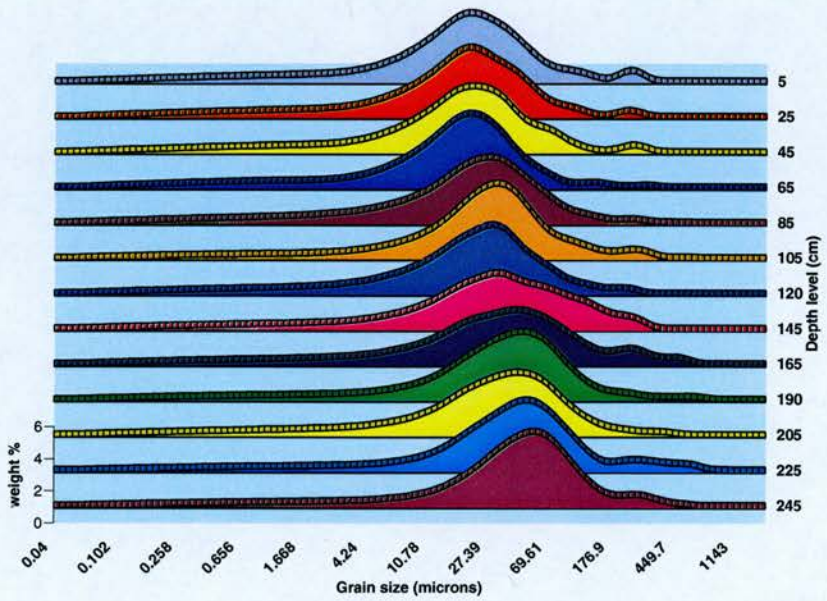

Figure 3.18: 3D perspective view of selected particle size distributions down the core from top to base.

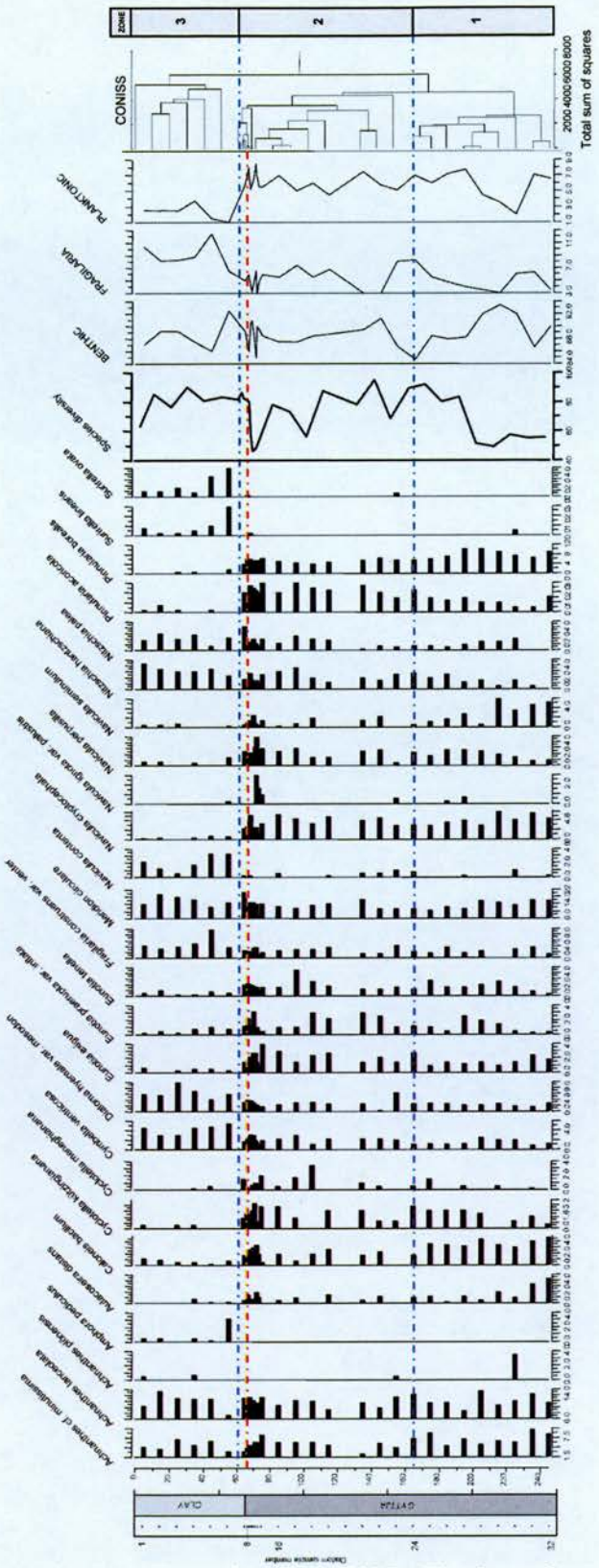

Figure 3.19: Diatom stratigraphy from analysis by Hill (2005). The three primary assemblage zones identified by CONISS are shown.

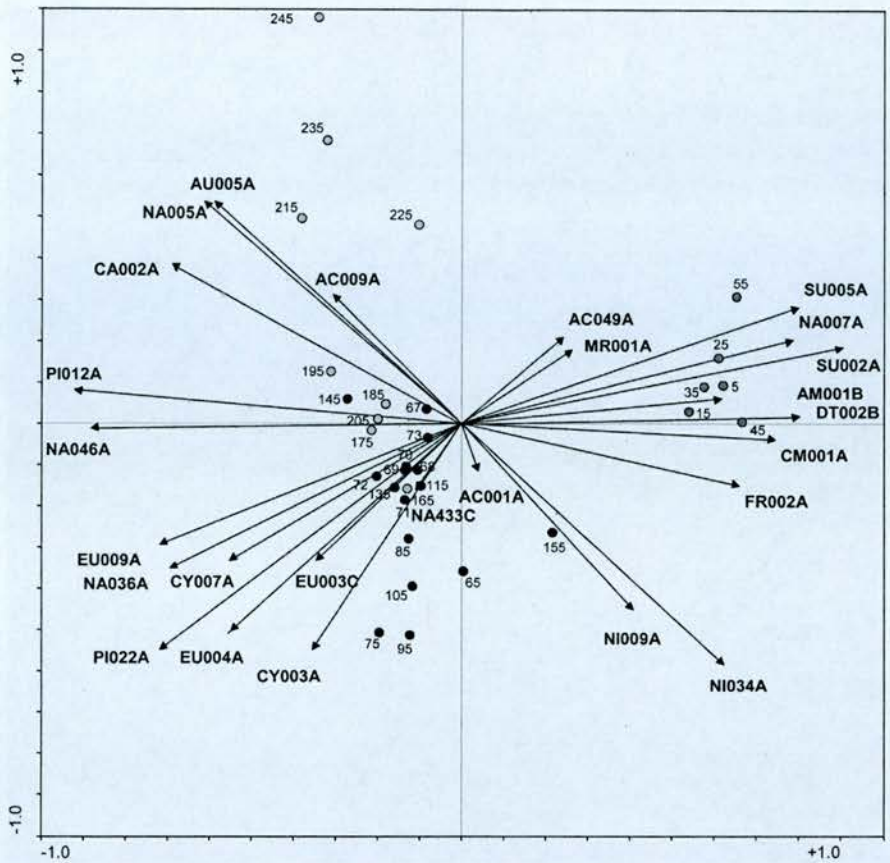

Figure 3.20: PCA biplot for diatoms in Skeiðsvatn (Hill, 2005). Arrows indicate diatom species preferences on the biplot, and filled circles are Skeiðsvatn samples, numbered by depth and coloured by CONISS zone. Samples further from the centre of the plot have stronger affinity with species along those axes (for example *Surirella* species, SU, correlate well with Zone 1 diatoms).

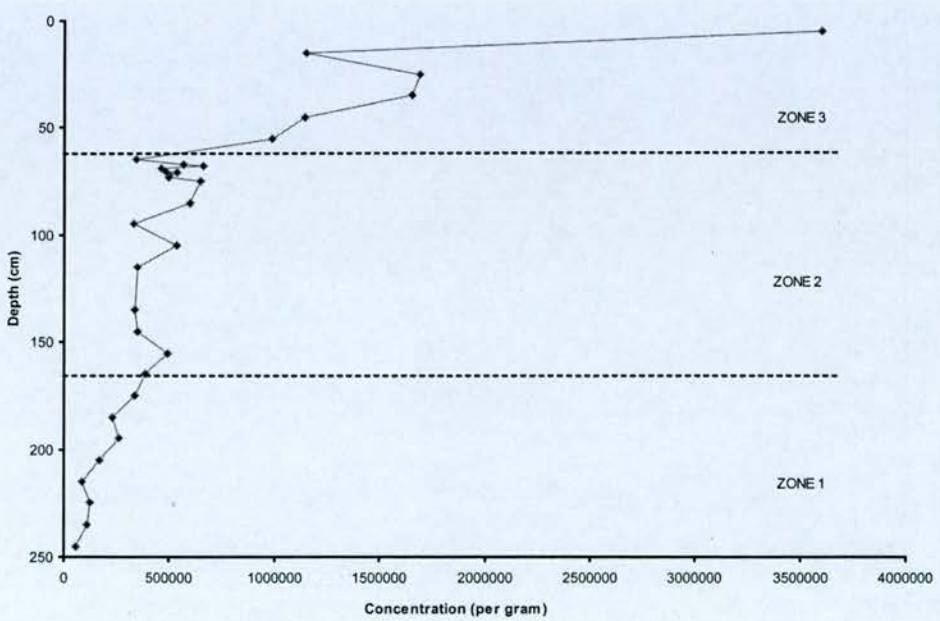

Figure 3.21: Diatom concentrations in Skeiðsvatn (Hill, 2005). There is a marked increase in concentration in the uppermost zone 3.

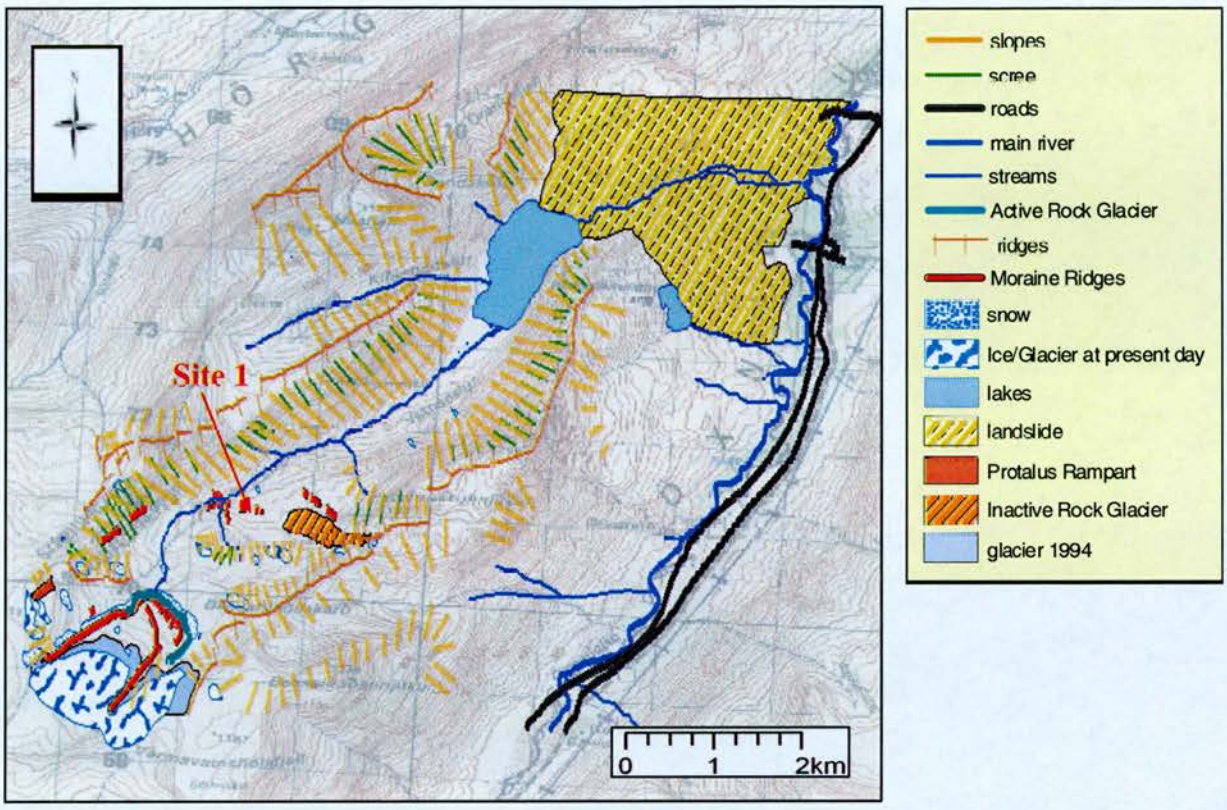

Figure 3.22: Geomorphological map of Vatnsdalur in Öxnadalur by Barr (2003). Mismatches between the map topography and the geomorphology are due to differences bwtween GPS and map coordinates. The LIA rock glacier is defined by the limit close to the glacier, and the pre-LIA limit is ~1.4km further down the valley.

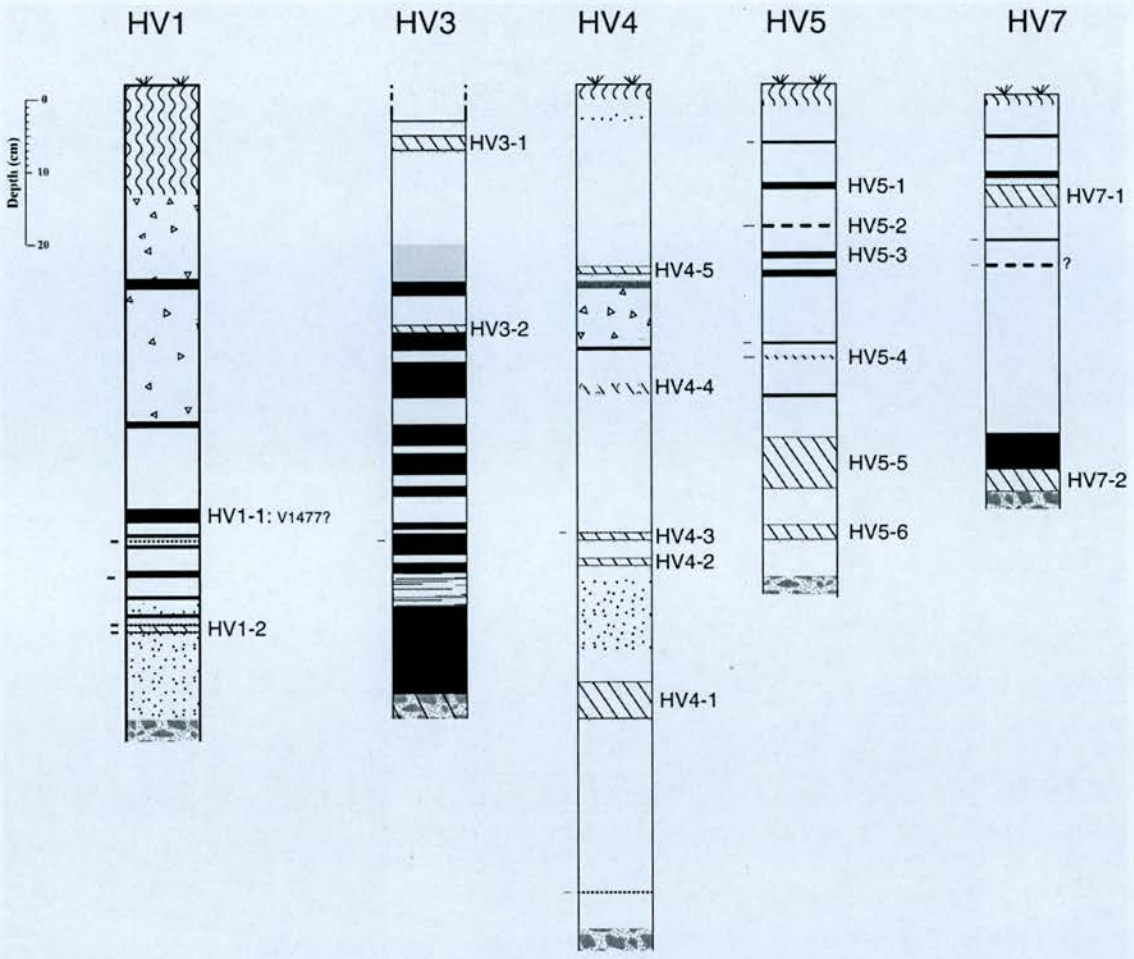

Figure 3.23: Aeolian sediment sequences from Vatnsdalur in Öxnadalur.

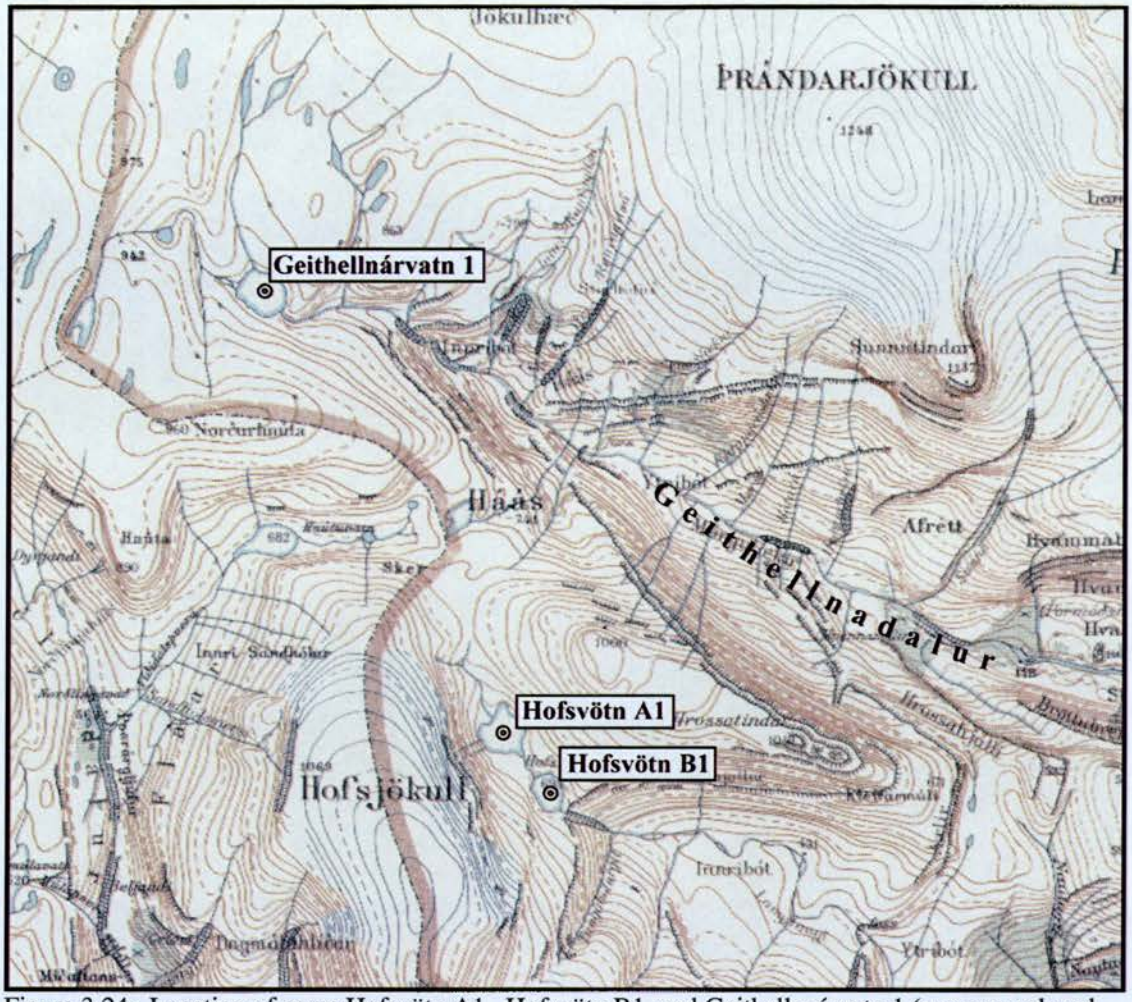

Figure 3.24: Location of cores Hofsvötn A1, Hofsvötn B1, and Geithellnaárvatn 1 (map reproduced from Landmælinger Íslands 1:100,000 series, no 52).

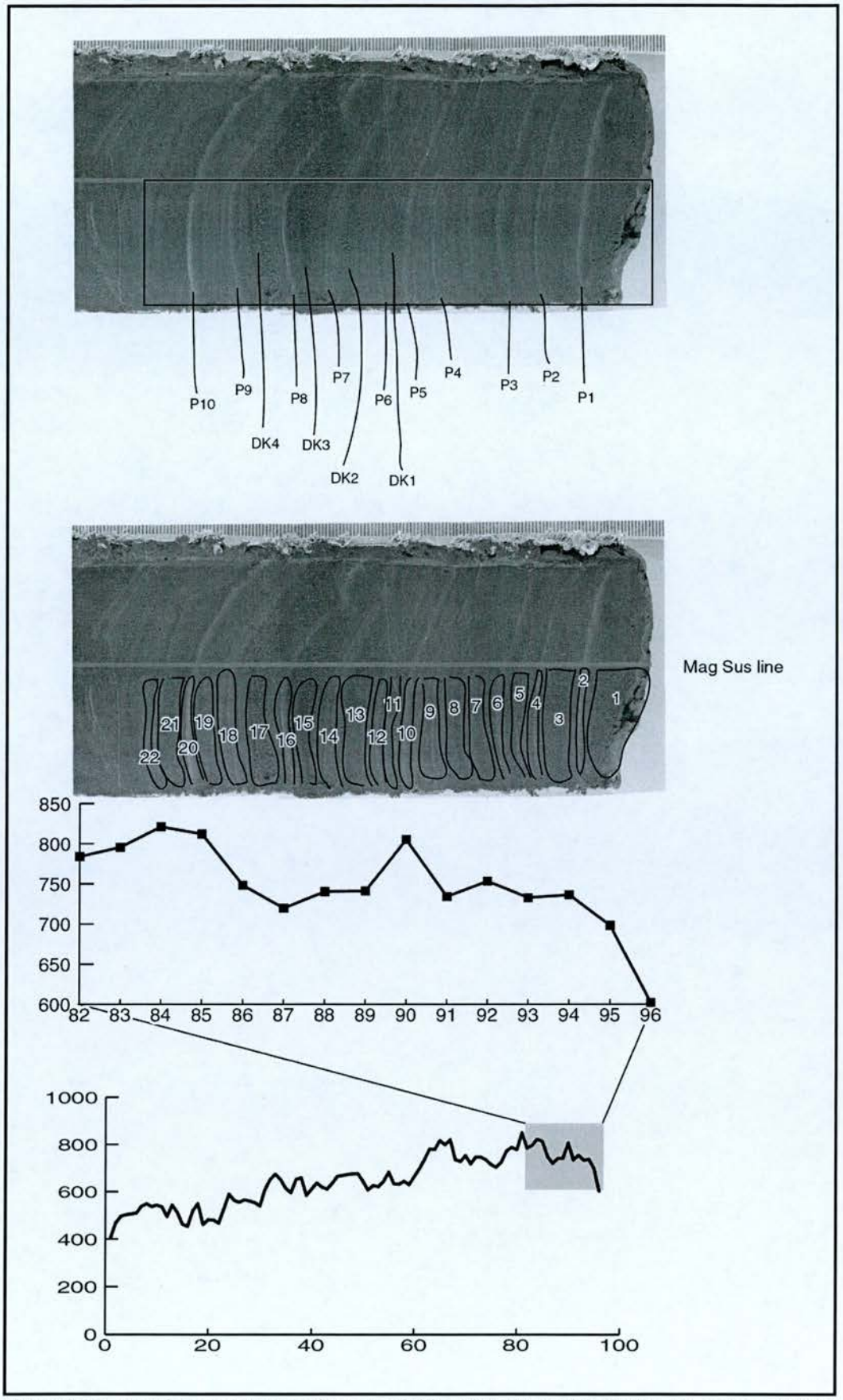

Figure 3.25: Low frequency magnetic susceptibility of the base of Hofsvötn A1, compared to the visible stratigraphy. The grey shaded box indicates the region of the graph covered by the 22 subsamples.

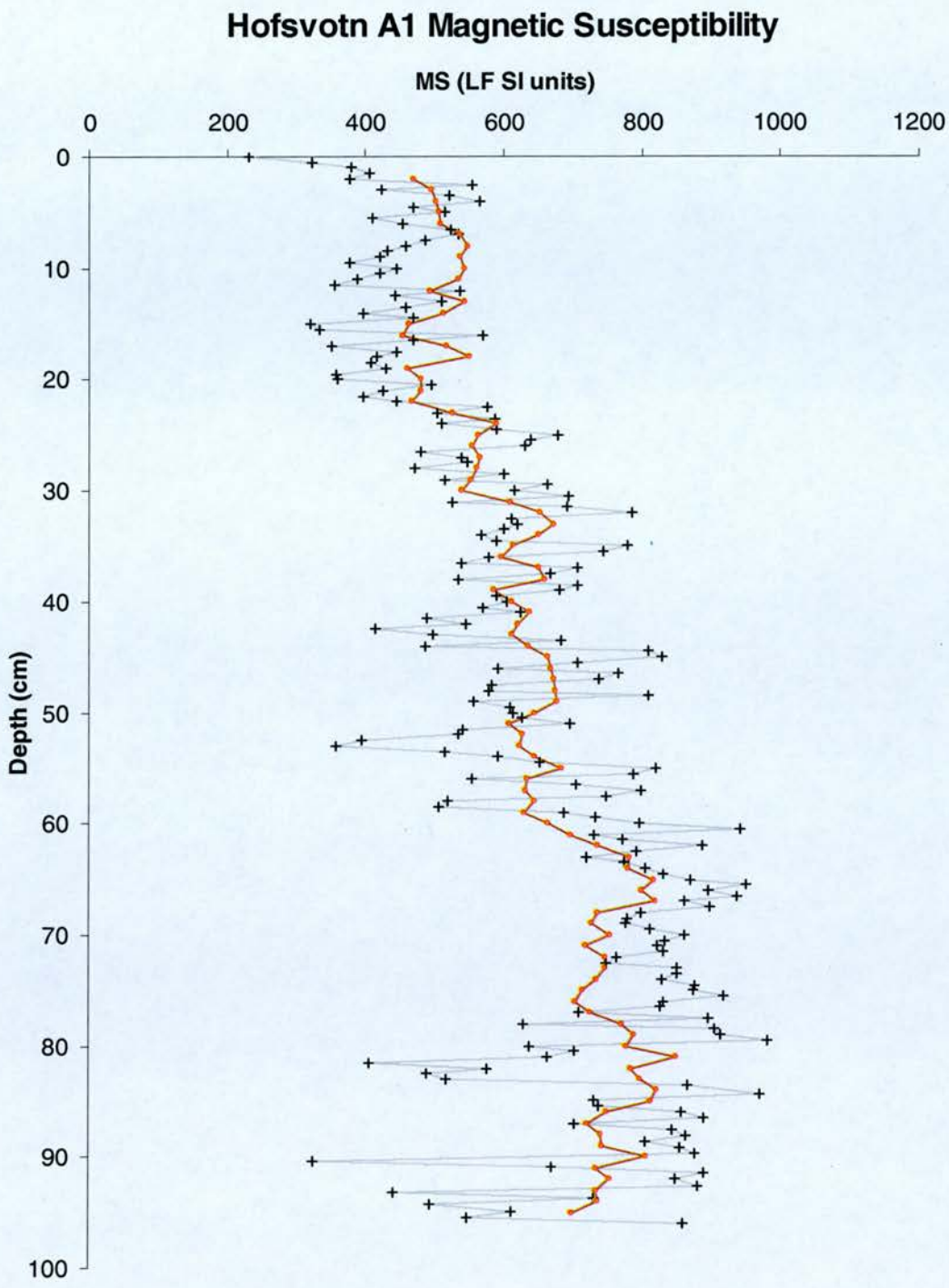

Figure 3.26: Comparison of low frequency magnetic susceptibility in core Hofsvötn A1 using the Bartington MS-2 (orange line) and MS-2B sensors (pale grey line). The absolute values are in good agreement, with the MS-2B values picling up more variability due to a higher sampling resolution.

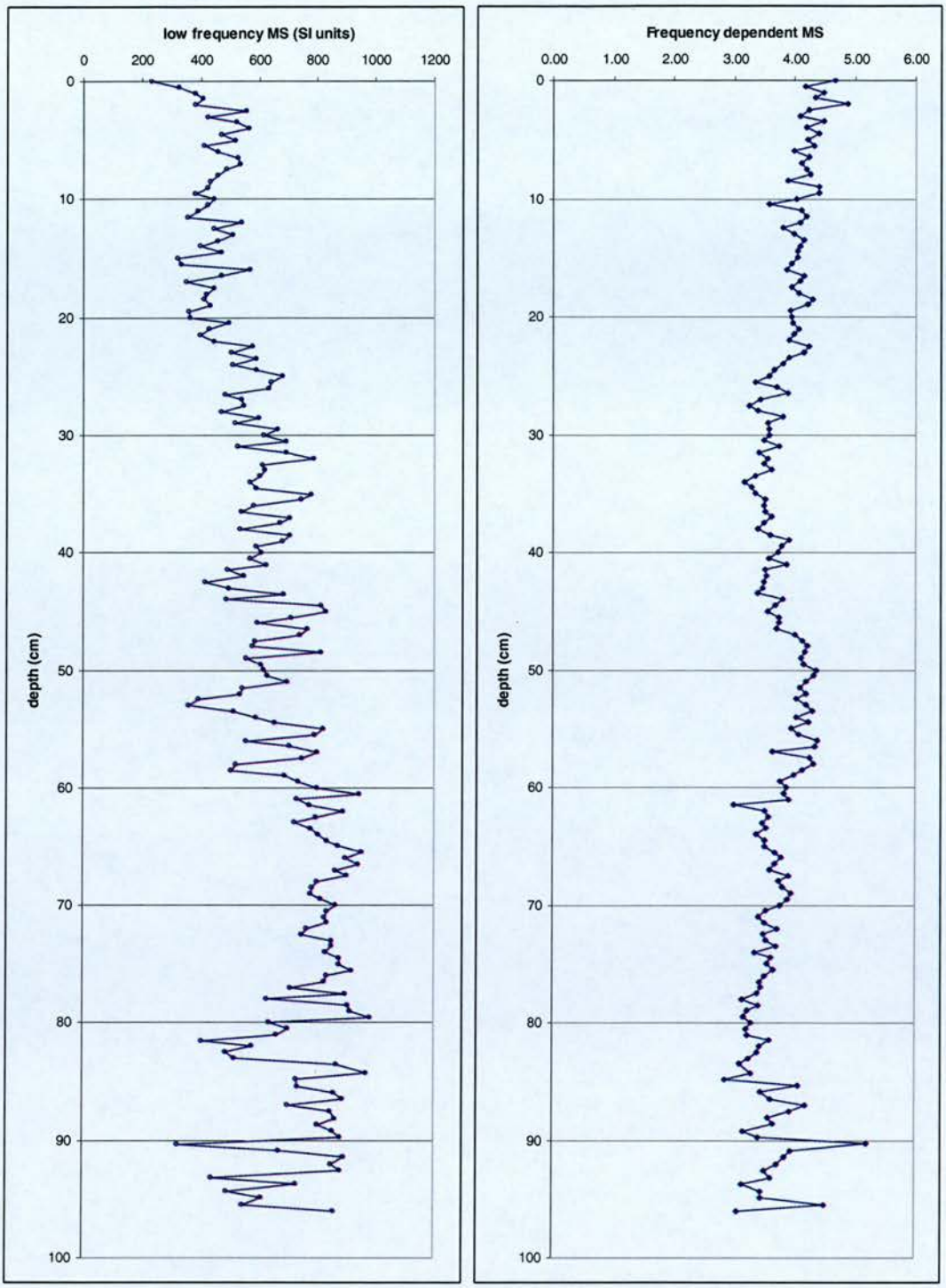

Figure 3.27: Mass mgnetic susceptibility results from core Hofsvötn A1, with χ_{LF} on the left and χ_{FD} on the right.

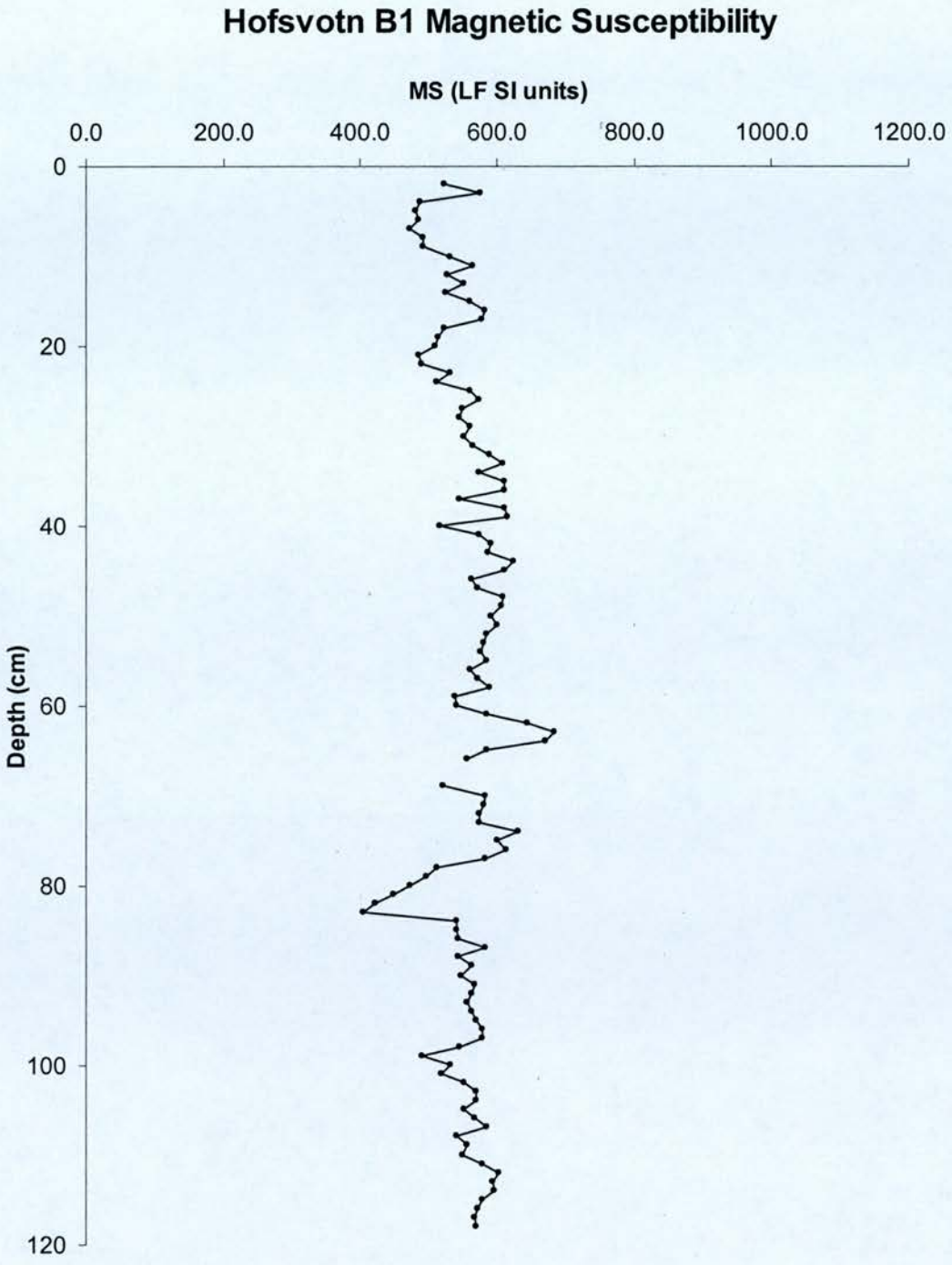

Figure 3.28: Low frequency magnetic susceptibility from core Hofsvötn B1. Very little variation is seen throughout the core.

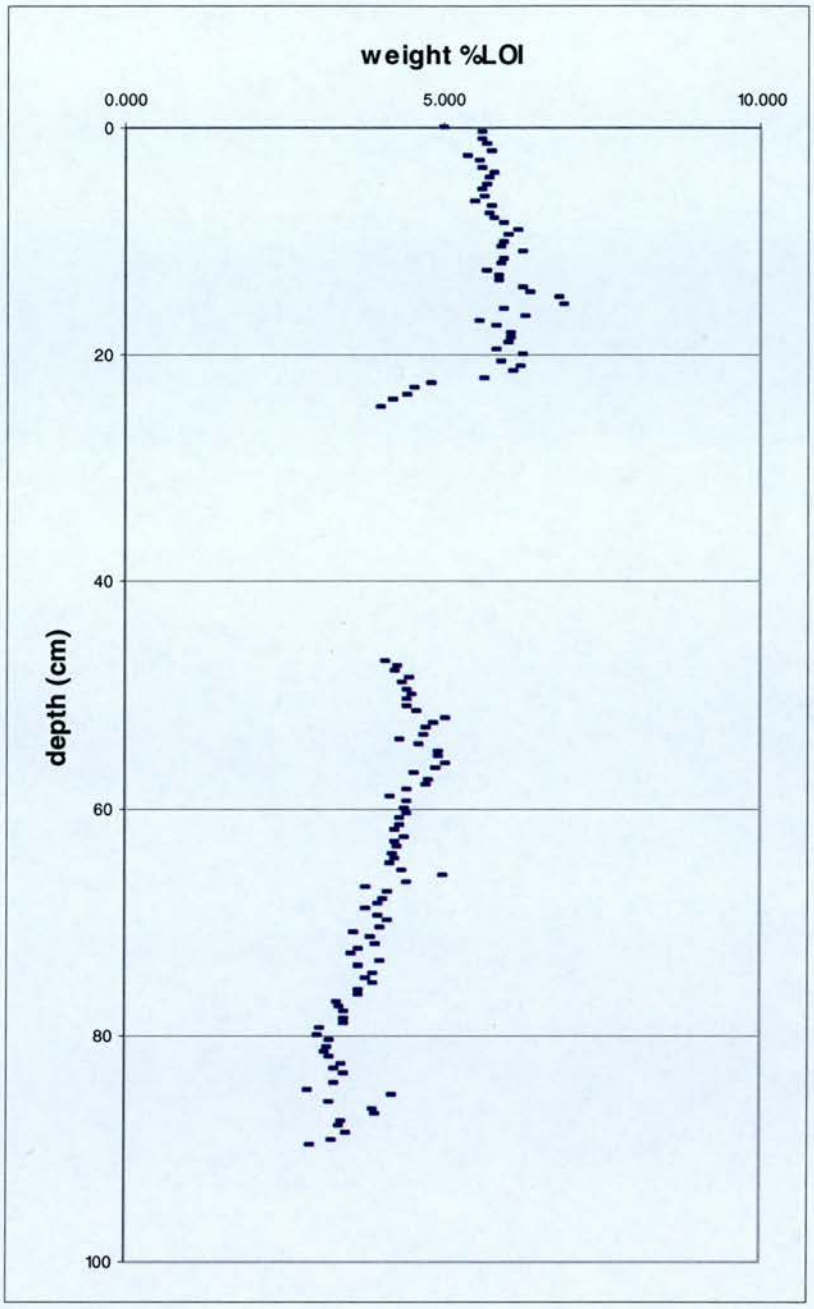

Figure 3.29: Loss on ignition values for Hofsvötn A1. Missing value between 25 and 46.5cm due to lost sample batch.

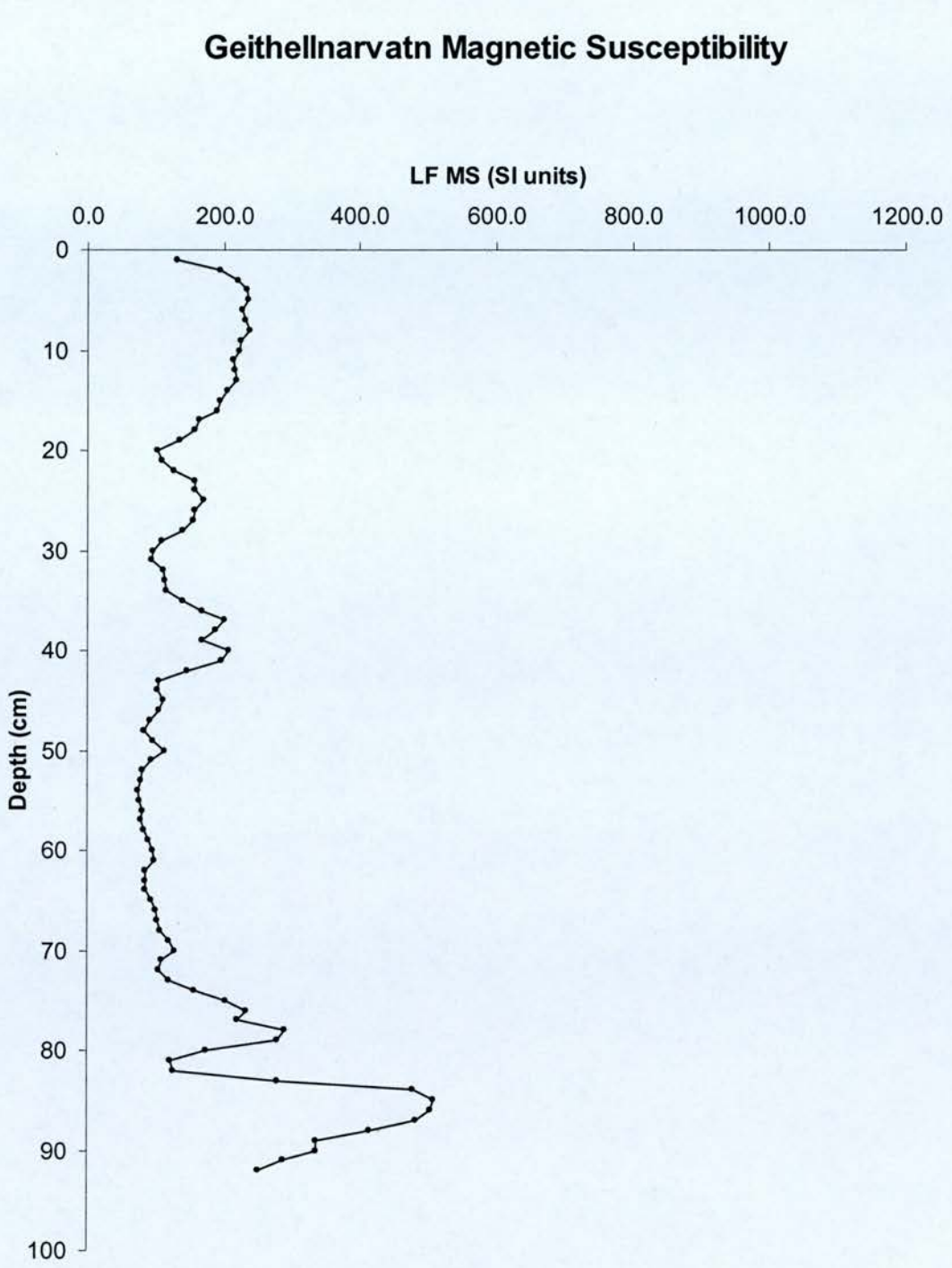

Figure 3.30: Low frequency magnetic susceptibility for Geithellnárvatn.

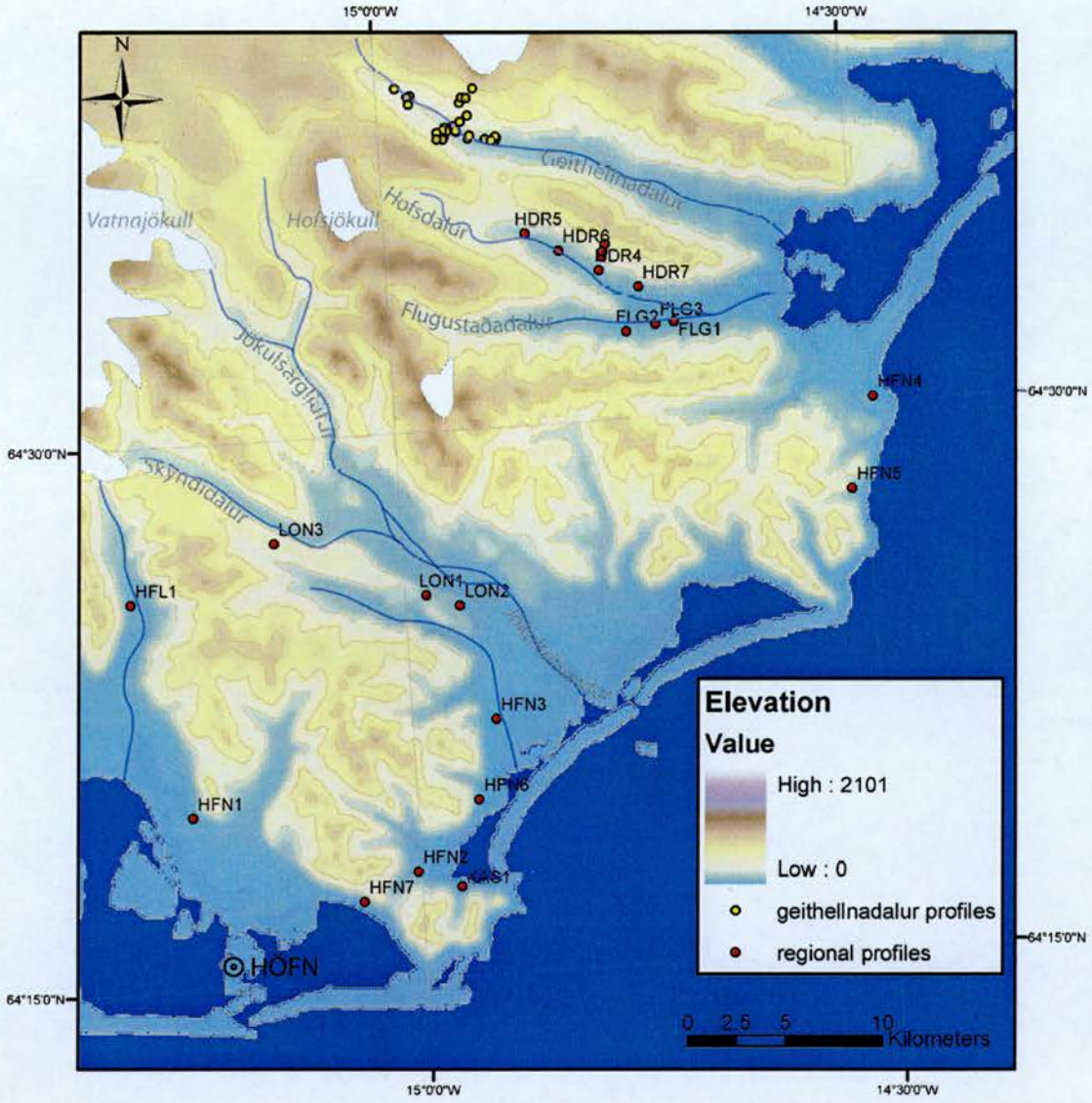

Figure 3.31: Map of southeastern Iceland, showing the locations of Hofsjökull, Geithellnadalur, Flugustaðadalur and Hofsdalur. It also has the location of the profiles illustrated in Figures 3.53-3.55, in the region to the south of Geithellnadalur.

Figure 3.32: Birch woodland in Hofsdalur. Woodland covers most of the lower ground below *c.*250m.

Figure 3.33: The landscape of Geithellnadalur from the northern side of the Geithellnaá. The Hvannavellir farm site is on the far side of the river beneath the large gully. The present day landscape shows much evidence of erosion (including tephrochronologists!), although there is also birch and willow present.

Figure 3.34: View southwest towards Hvannavellir. The green area beneath the large gully and fan is favourably palced in comparison to steep and rocky surrounding land. There is also access to the upper part of Hofsdalur along the bench in the lava flows (used by reindeer hunters today).

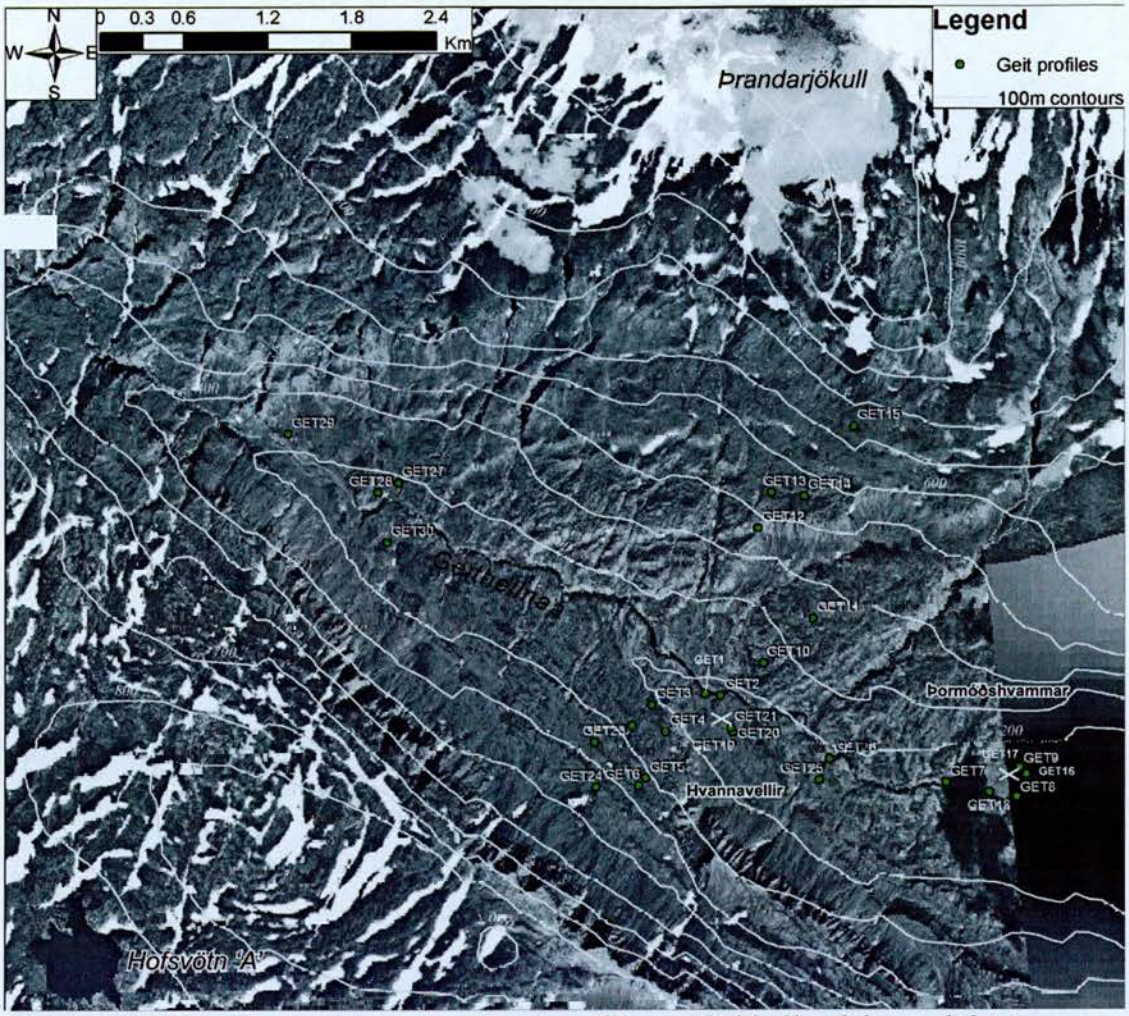

Figure 3.35: Location of aeolian sediment profiles in Geithellnadalur and the two abandoned farms, Þormóðshvammar and Hvannavellir.

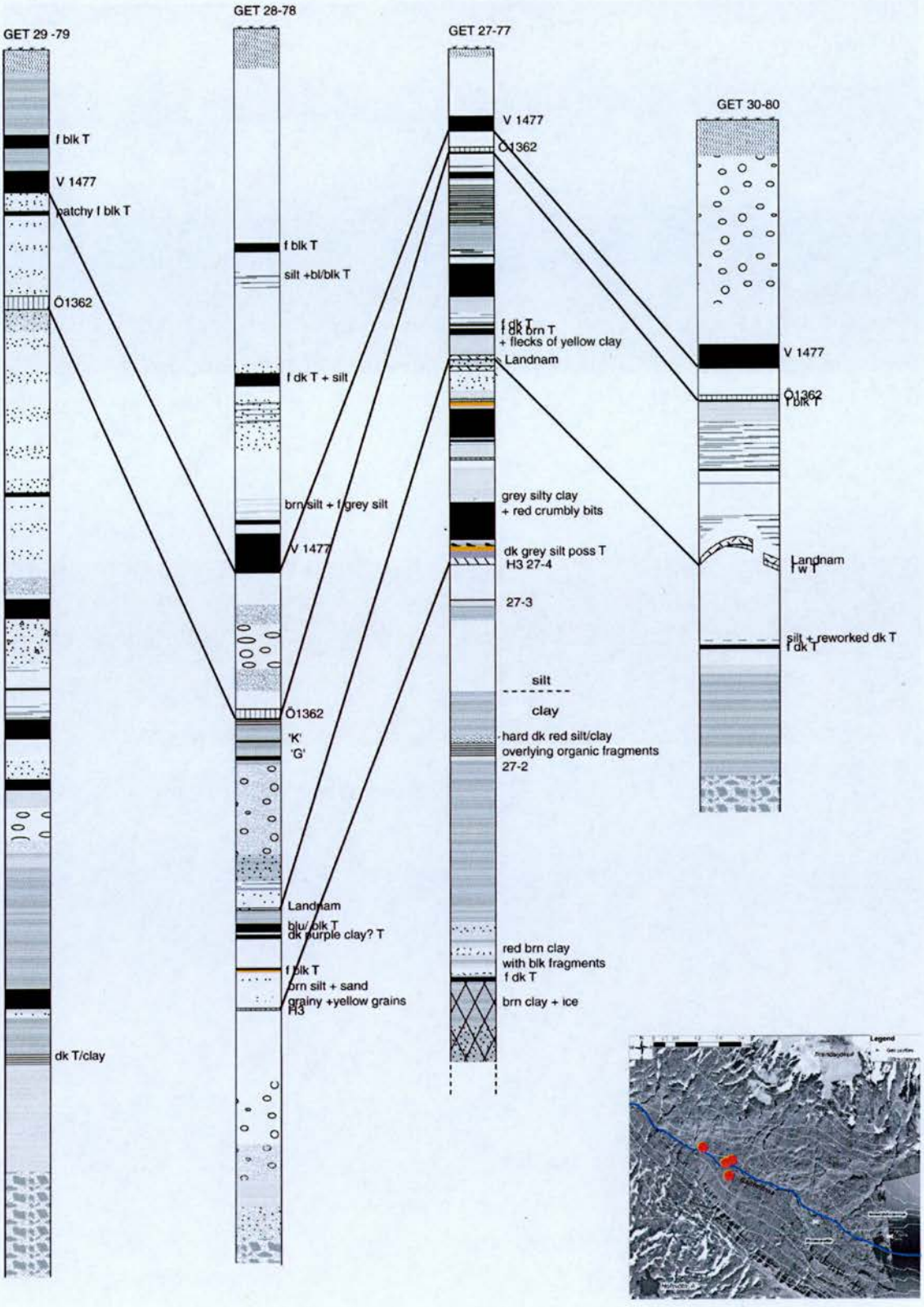

Figure 3.36: Aeolian sediment sequences from the upper part of Geithellnadalur.

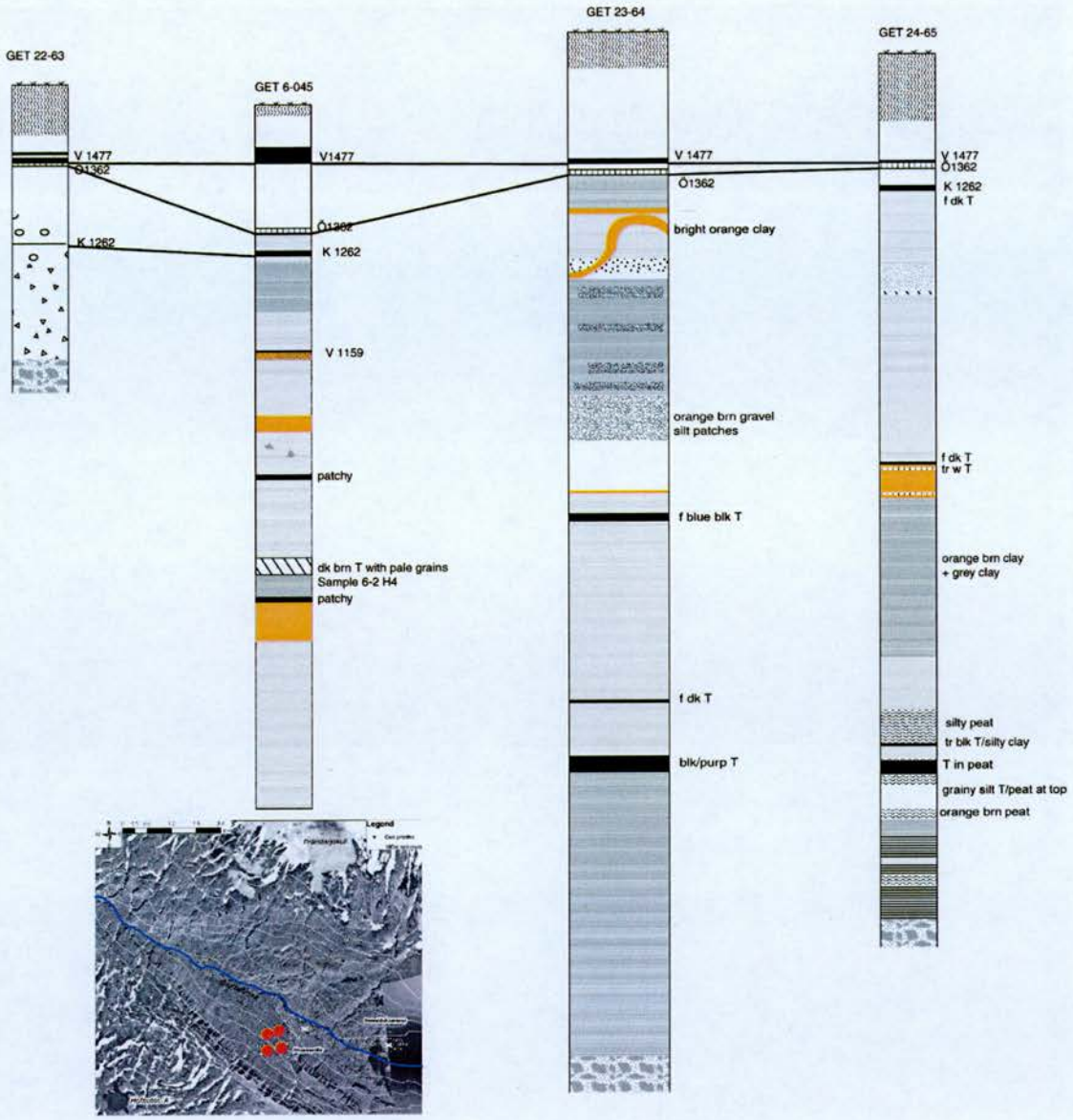

Figure 3.37: Aeolian sediment sequences from the hillside southwest of Hvannavellir.

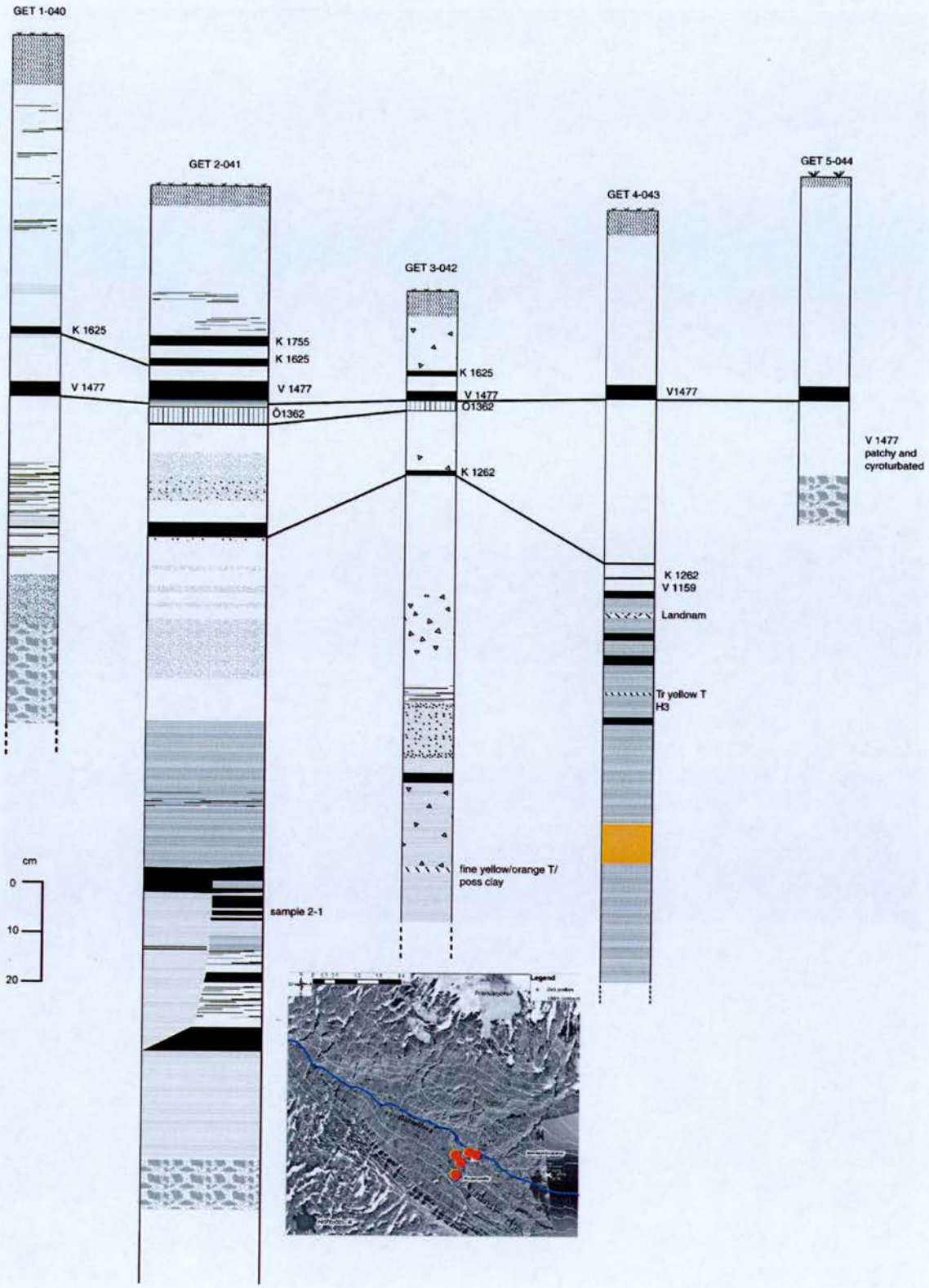

Figure 3.38: Aeolian sediment sequences from close to Hvannavellir.

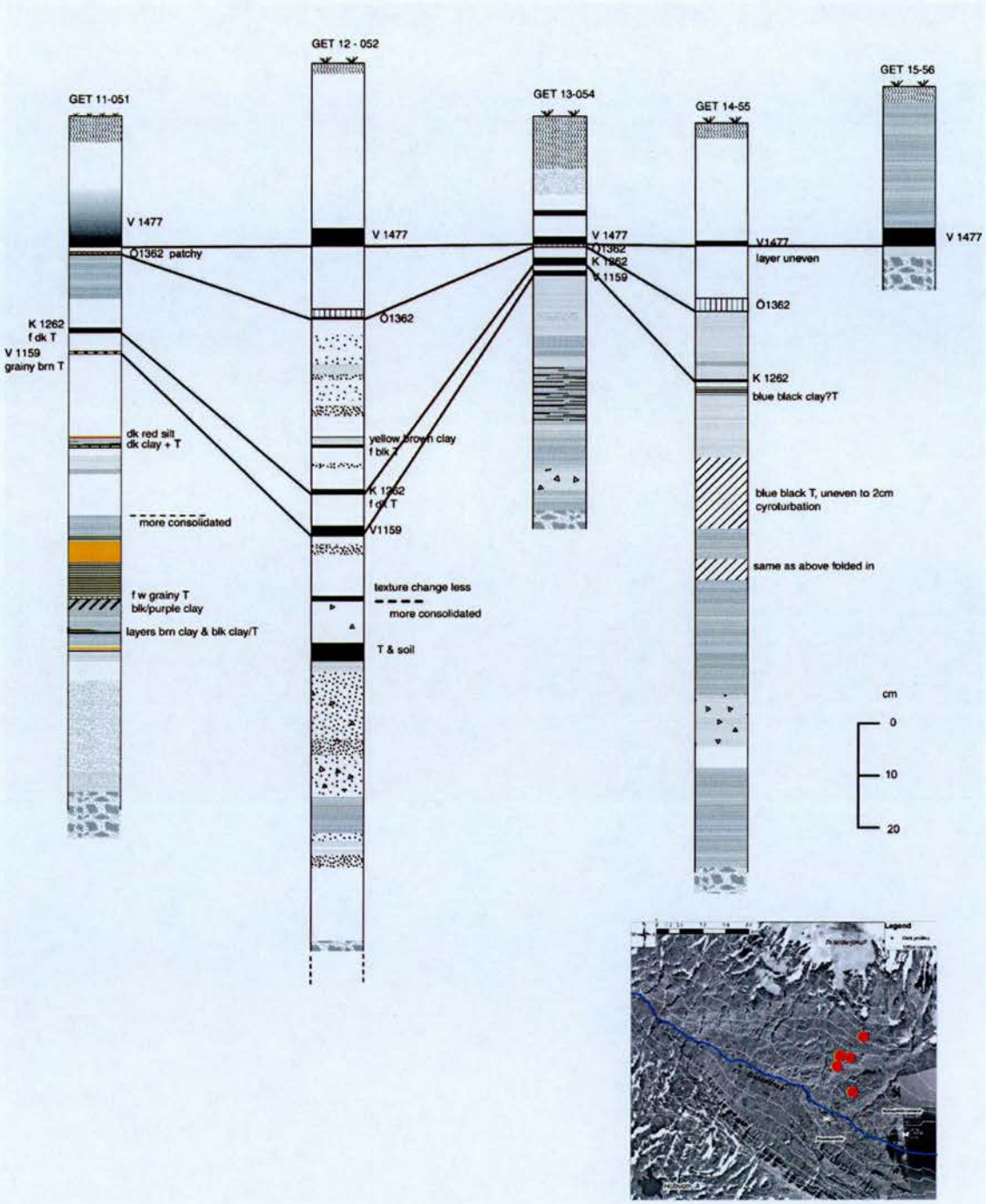

Figure 3.39: Aeolian sediment sequences from the hillside north of Þormóðshvammar.

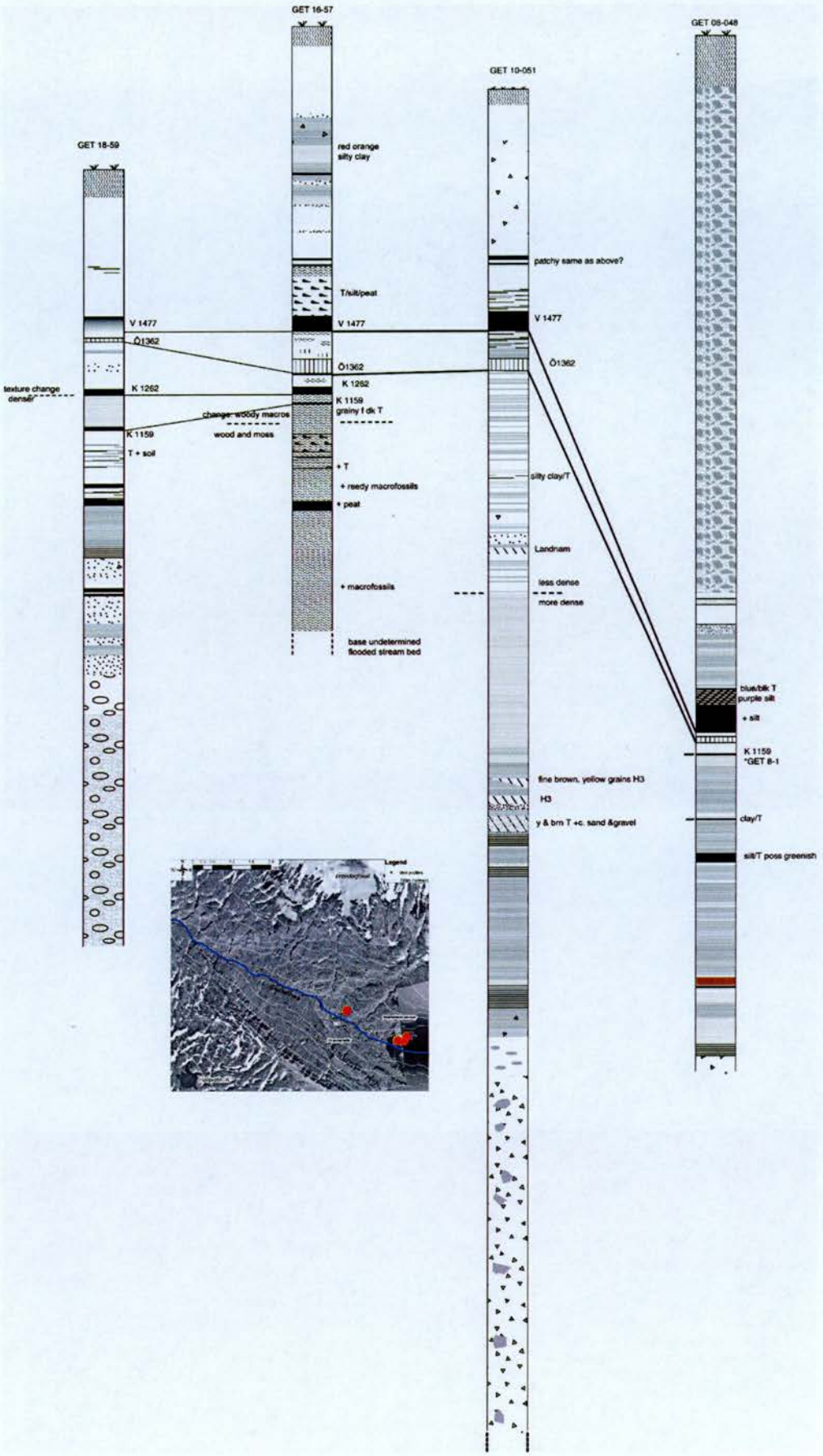

Figure 3.40: Aeolian sediment sequences from close to Þormóðshvammar.

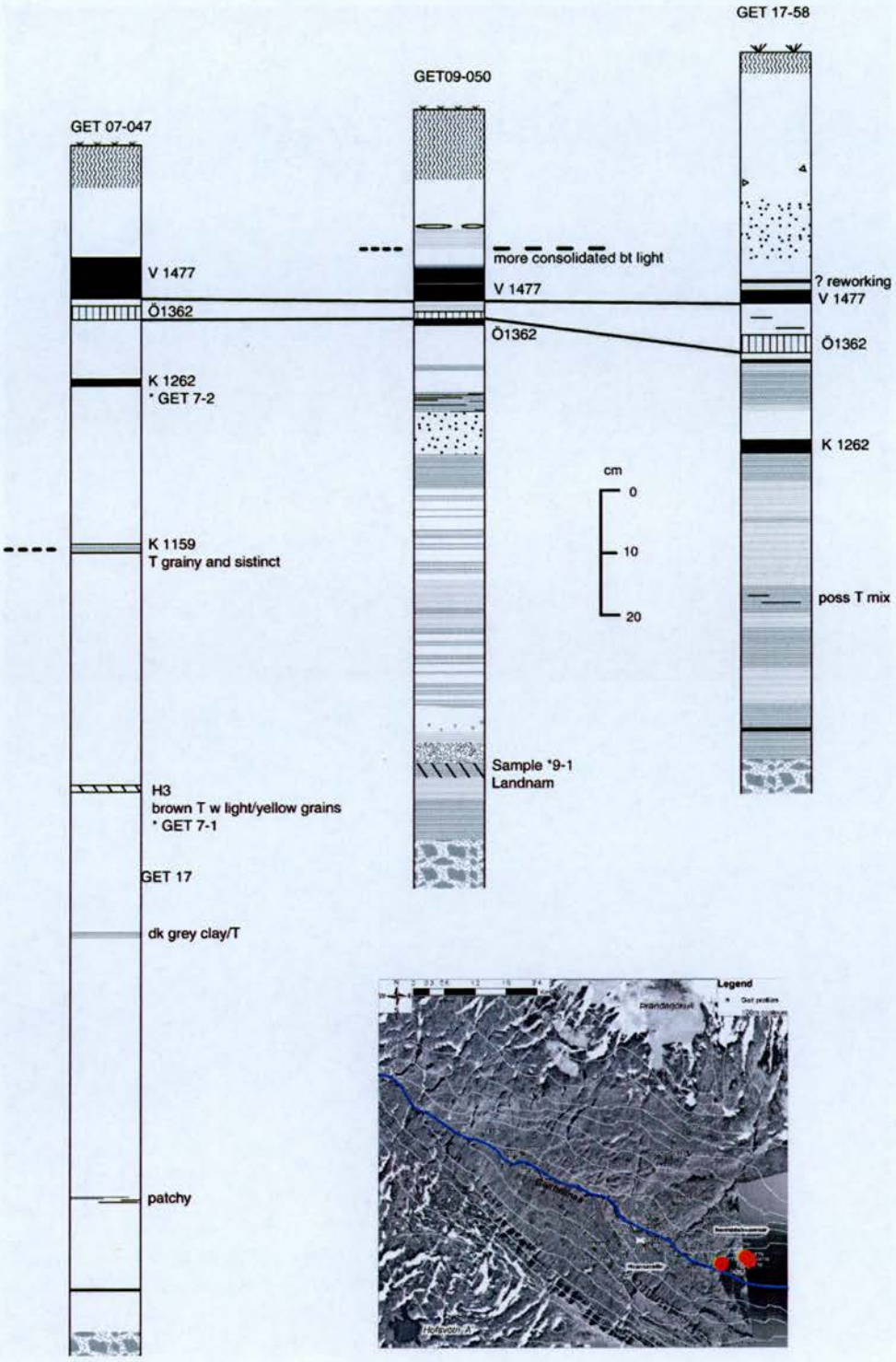

Figure 3.41: More aeolian sediment sequences from close to Þormóðshvammar.

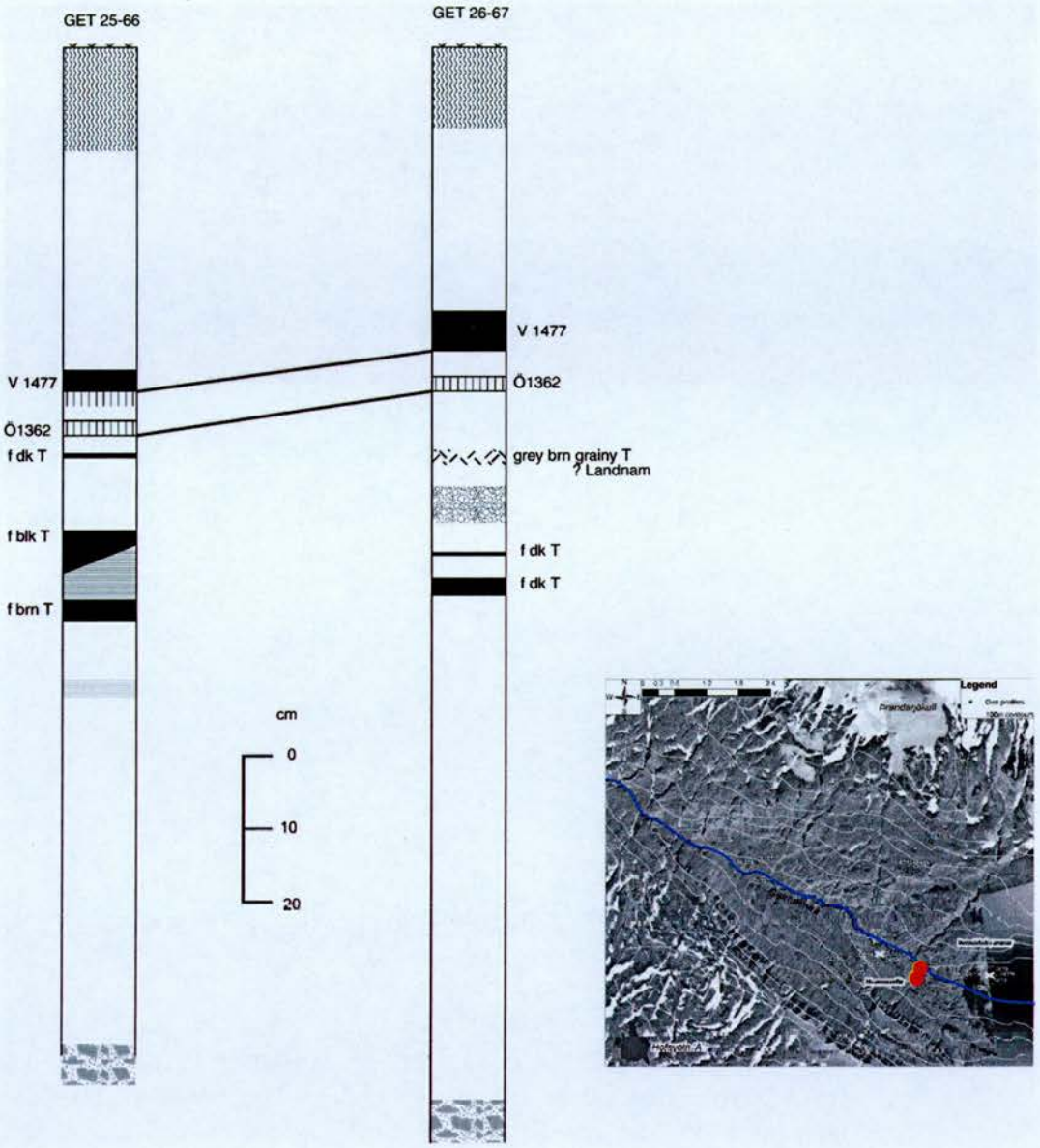

Figure 3.42: Aeolian sediment sequences from southeast of Hvannavellir.

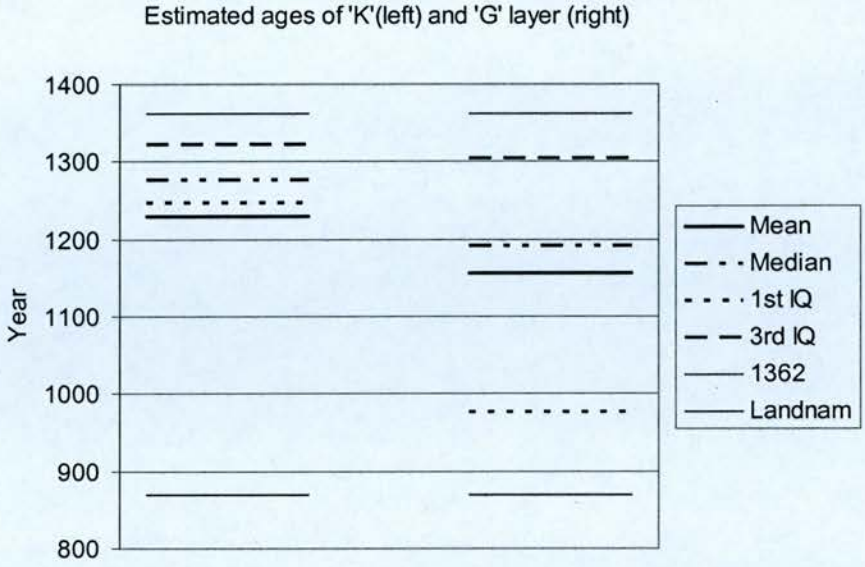

Figure 3.43: Estimating the age of two basaltic tephra layers 'K' and 'G' using the mean, median and interquartile ranges between Ö1362 and V871 (Howard, 2005).

Eruption	Number of occurrences
K1755	3
K1625	7
V1477	27
Ö1362	19
K1262	17
V1159	11
V871	7
H3	6
H4	2

Table 3.2: Total number of identifications of tephras in 27 profiles from Geithellnadalur and used for the SAR analysis.

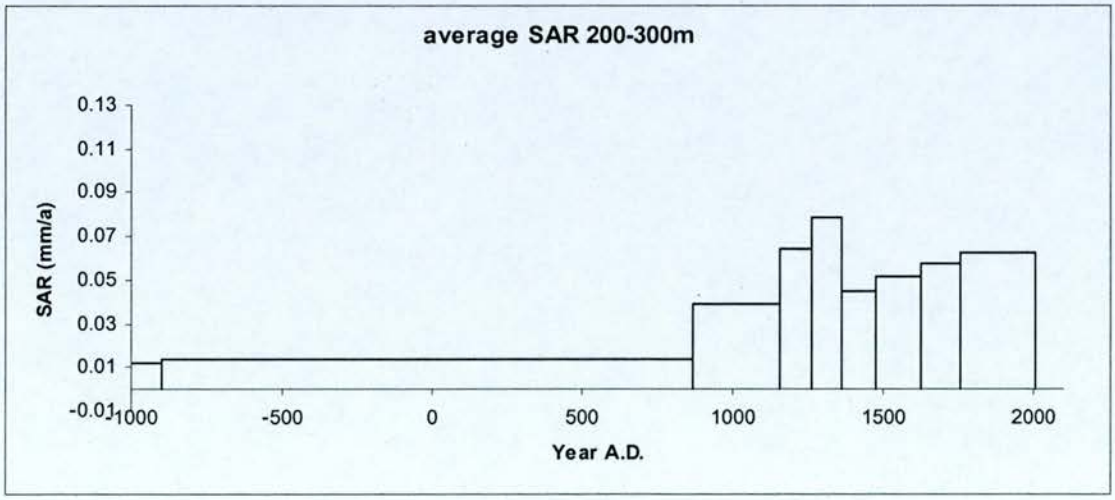

Figure 3.44: Overall pattern of SAR in Geithellnadalur

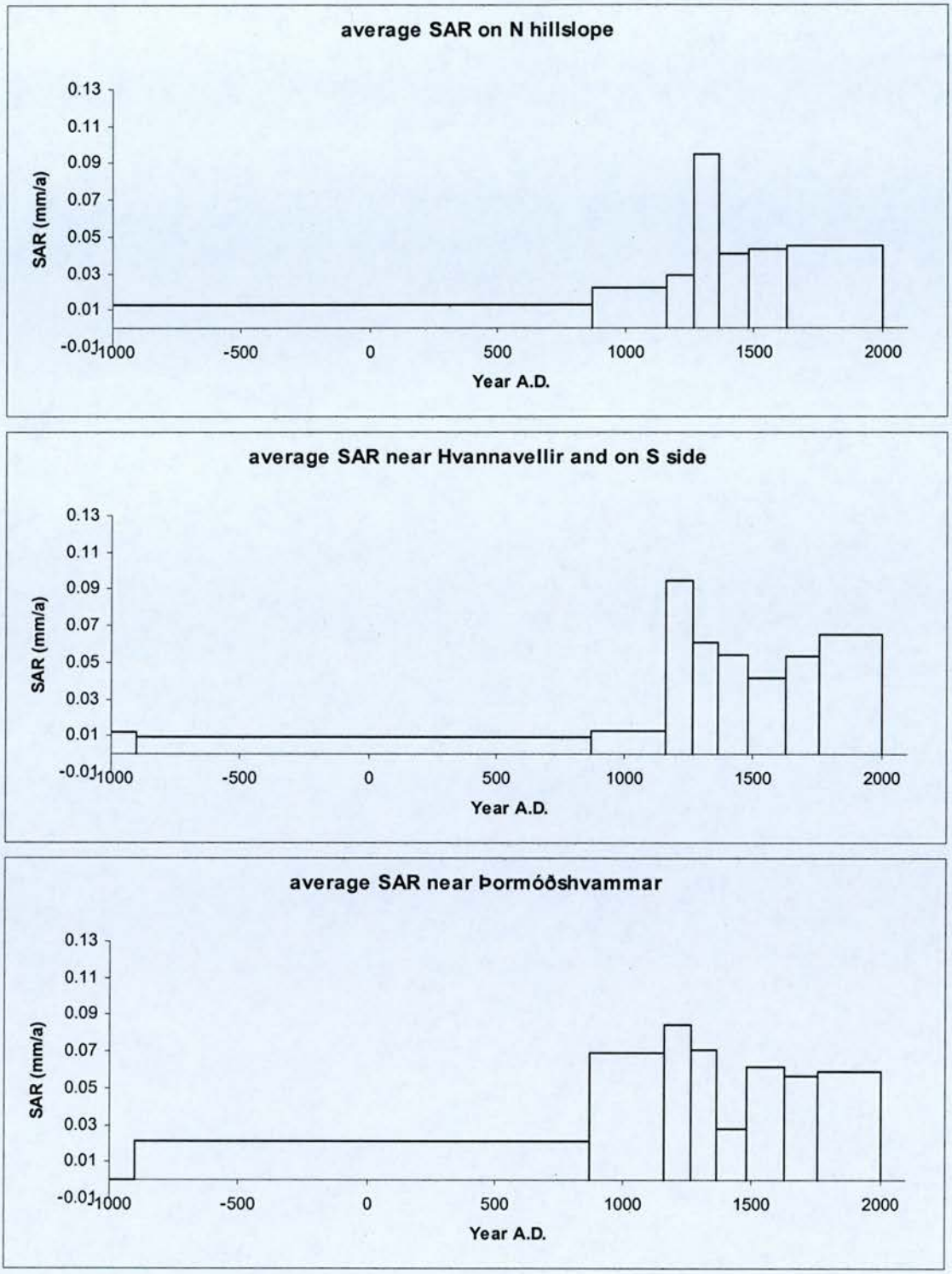

Figure 3.45: Spatial distribution of SAR – sediment accumulation in different parts of Geithellnadalur (calculations by Howard, 2005).

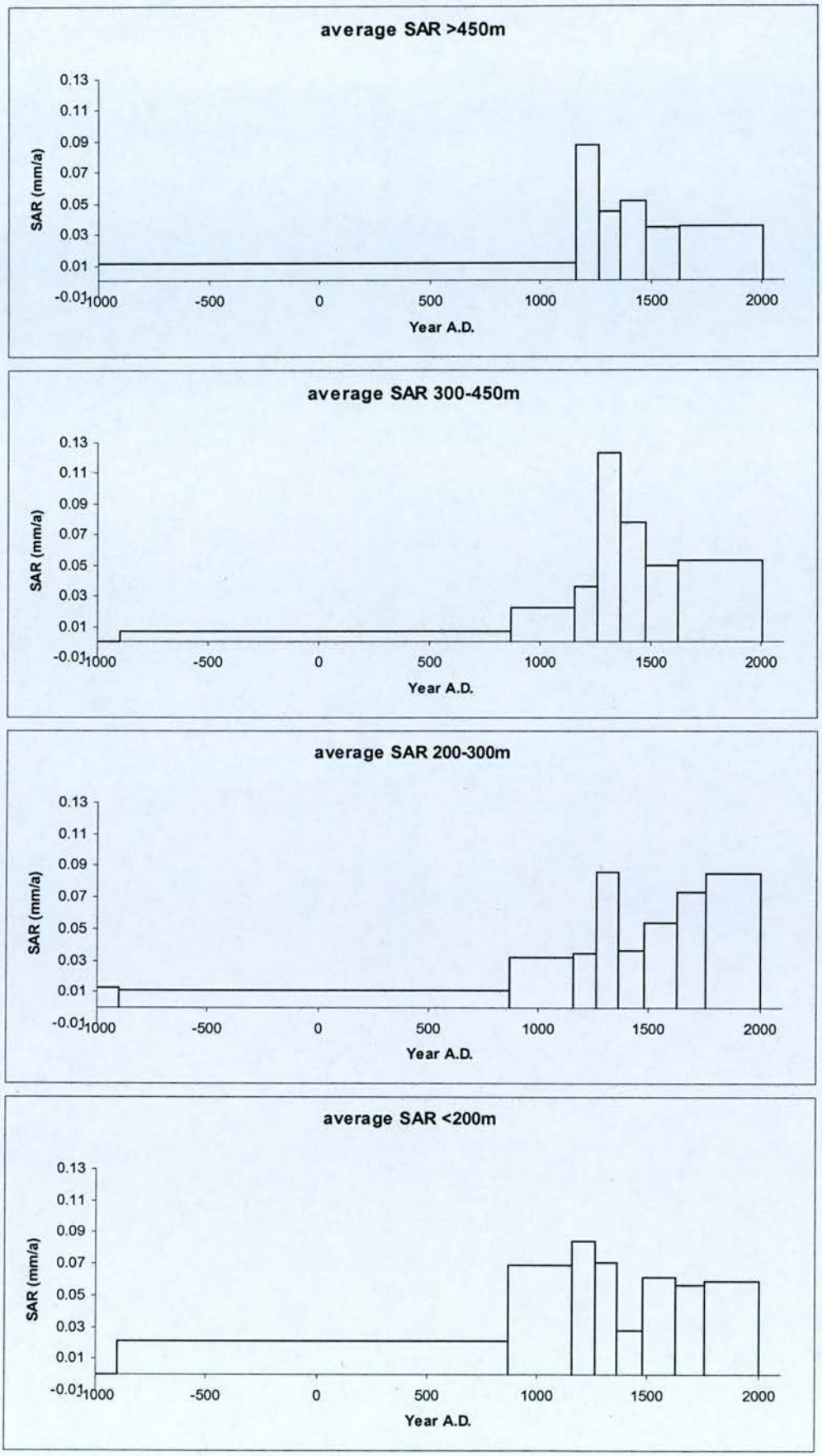

Figure 3.46: Distribution of sediment accumulation rates with altitude in Geithellnadalur (calculations by Howard, 2005).

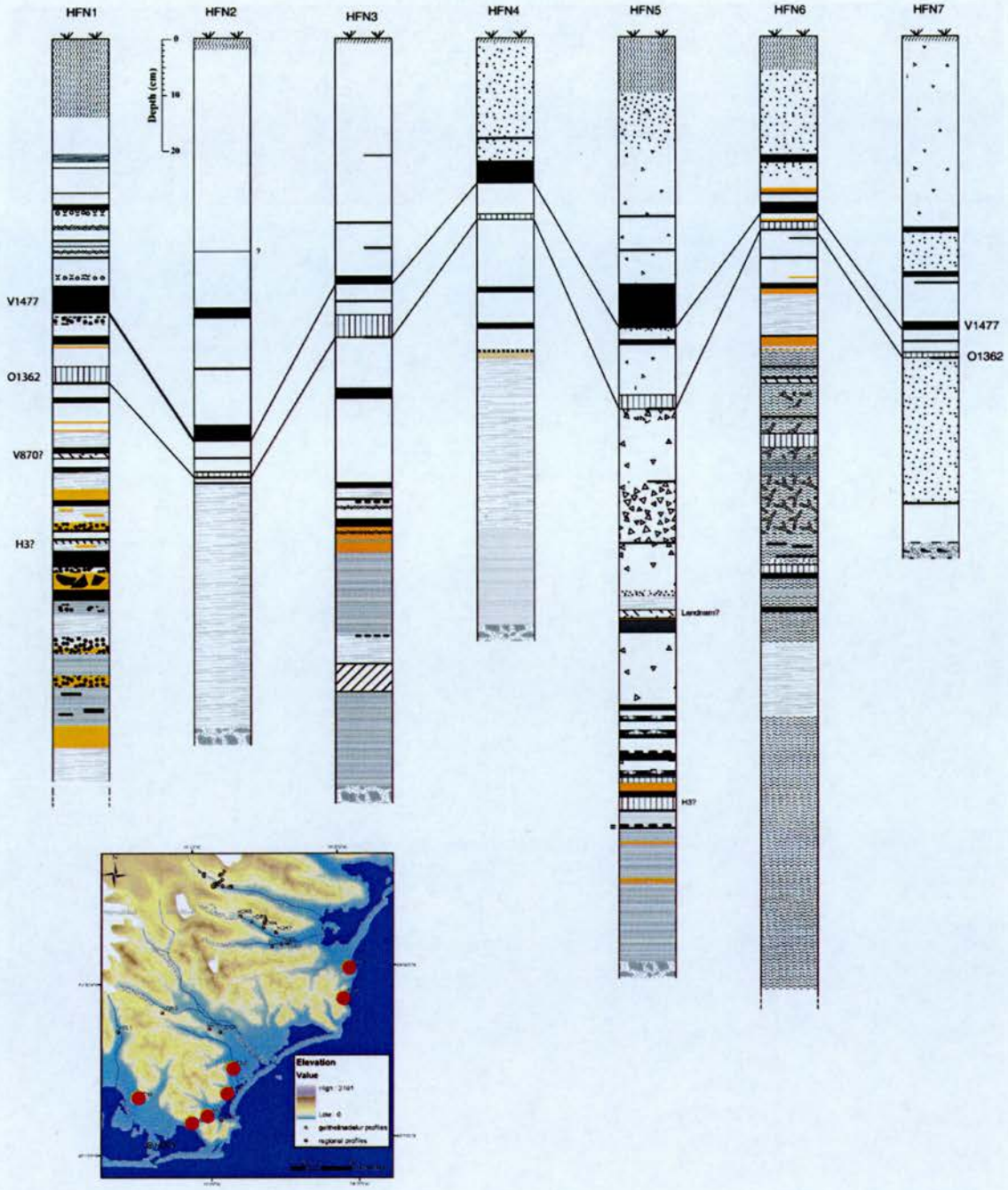

Figure 3.47: Regional tephra stratigraphy – profiles from the coast between Geithellnadalur and Höfn. A map of profile locations is in Figure 3.31.

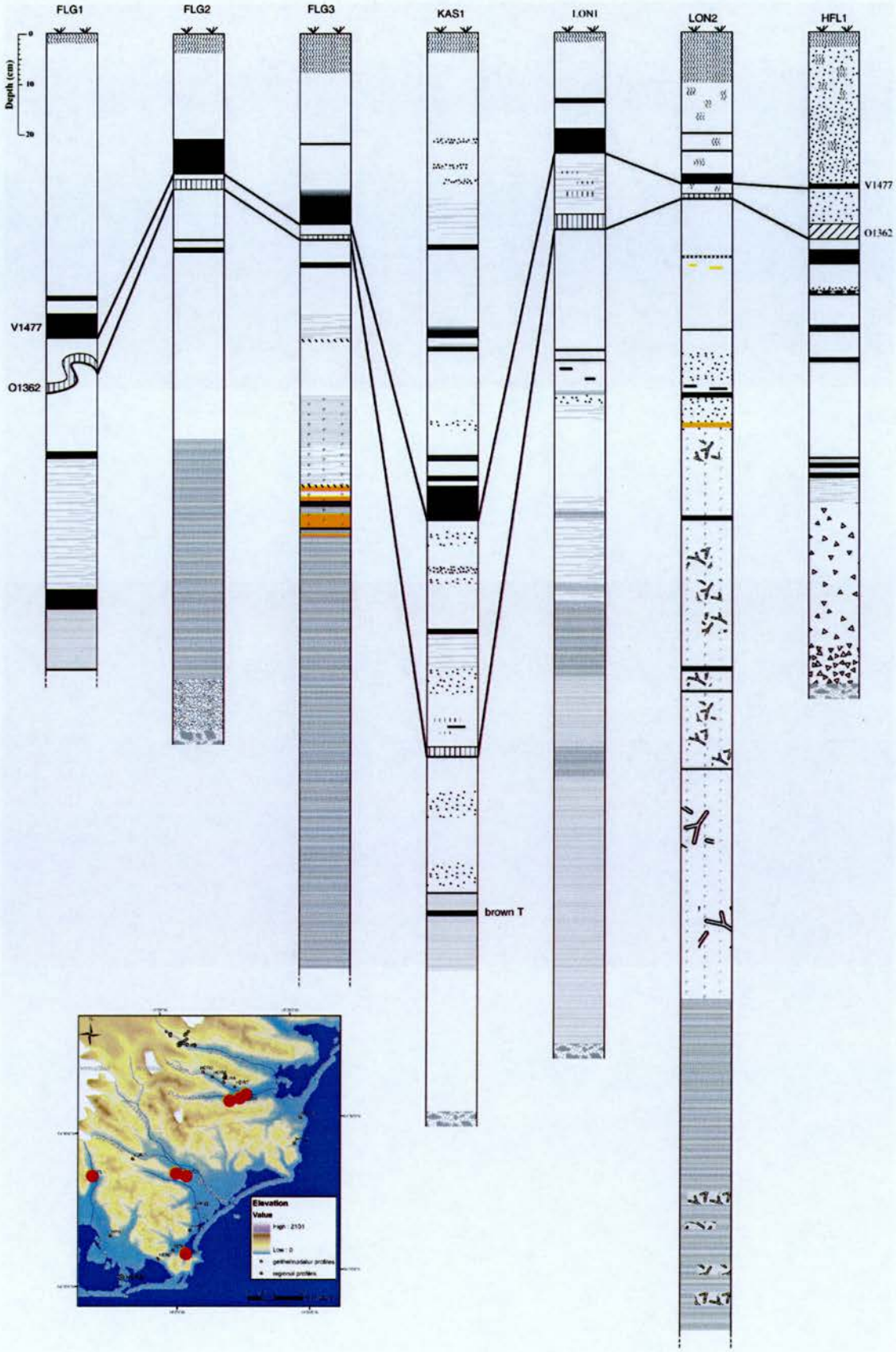

Figure 3.48: Regional tephra stratigraphy – profiles from Flugustaðadalur, Kastárdalur, Lónsoræfi and Hoffelsdalur. A map of profile locations is in Figure 3.31.

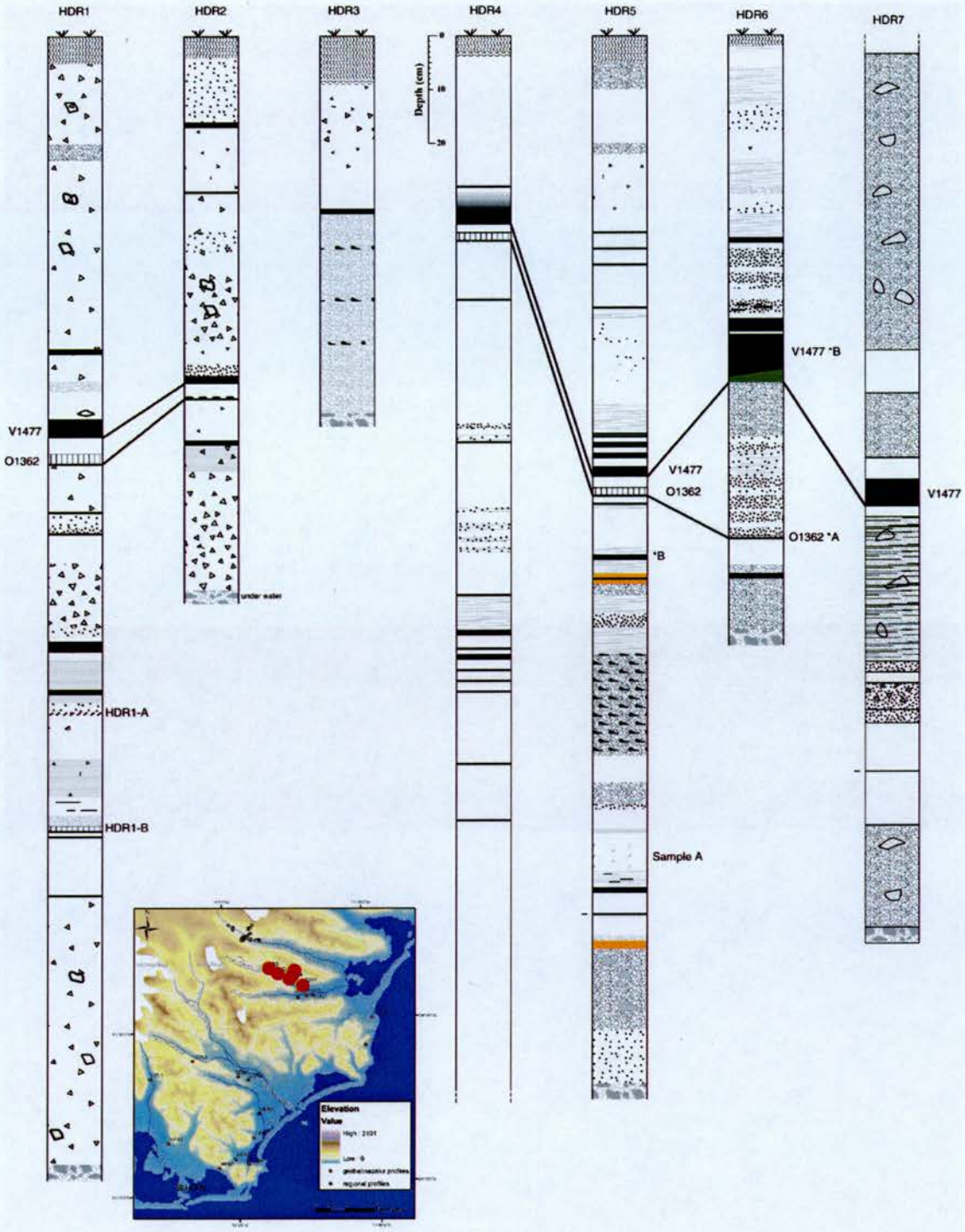

Figure 3.49: Regional tephra stratigraphy – profiles from Hofsdalur. A map of profile locations is in Figure 3.31.

Chapter IV: Modelling

1 Introduction

2 Numerical modelling of mass balance and vegetation in Iceland

3 Mass balance modelling – results

4 Catchment-scale glacier modelling – application of GLIMMER

IV.1: Chapter Introduction

The modelling in this thesis is designed to indicate envelopes of likely environmental change, constrained by the gathered empirical environmental data. It is separated into two streams.

The first modelling approach is an Iceland-wide equilibrium line altitude model based on recent temperature/precipitation data from weather stations around Iceland. This can be used to assess the sensitivity of different regions of Iceland, and of glaciers in Iceland, to climatic changes on the order of 1°C. From it, glaciers with marginal mass balance can be determined and investigated. Equilibrium-line altitude modelling can alaso be used to assess the sensitivity of marginal lands beneath the glaciers and elsewhere in the uplands to environmental change.

The second is to examine the specific glacierised catchments in which lake sediment cores have been taken. These are at Vatnsdalur in northern Iceland, and Hofsjökull in southeastern Iceland. The modelling used here will apply the GLIMMER Data from these specific models will constrain the extent of mass balance change required to produce geomorphological and sedimentary changes observed.

Glacier fluctuations provide a good, but partial record of change across a landscape, and glaciers with different environmental settings (eg aspect, hypsometry, altitude, plateau/corrie) will respond in differing ways. This multitude of related responses can cloud the causes of these changes. We cannot look at all glaciers, and we do not have a complete temporal dataset in any case. Numerical modelling is an alternative approach to exploring the glacier sensitivity to temperature and precipitation changes.

IV.2 Numerical modelling of mass balance and vegetation in Iceland

The objective of this section of the thesis is to construct and test a mass balance model for the whole of Iceland, using available temperature and precipitation data. The mass balance model can be used to test the sensitivity of both glaciers and landscape elements to environmental changes. Such changes not only include positive/negative mass balance shifts in areas close to the present limits of glaciation, but also associated changes to zones of severe periglaciation, and snowcover extent on marginal rangelands.

The key constraint upon whether the small glaciers in this study grow or disappear is their mass balance. The glaciers are too small for factors related to internal ice flow dynamics or additional complexities of large ice sheets to play a part. Consequently a well-constrained mass balance model is an important starting point for the modelling. The mass balance can be tested using available mass balance data, and so examine the validity of the model.

The mass balance for any location can be estimated using two pieces of information: the accumulation through snowfall when the temperature is below 1°C, and the ablation, through melting of snow at temperatures above 0°C. This is a simplified mass balance, and neglects other factors such as refreezing of meltwater, and avalanching of snow in corries. Using this information, as well as estimations of the rate of change of temperature with elevation (the lapse rate), and rate of change of precipitation with elevation, the equilibrium line altitude across Iceland can be estimated.

IV.2.1: Model architecture – primary functions of the model

In order to achieve the objective of a successful dynamic mass balance model for Iceland, we need to consider the basic structure of the model. The first consideration was the format of the model's inputs and outputs. Inputs are available in two formats:

1: time series (point) climatic data, available from specific weather stations around Iceland (Figure 4.1).

2: gridded (cell-based) climatic data, monthly temperature and annual precipitation. Topography is also available as gridded data.

Given the nature of the available input data, and the value of a strong spatial element to the output, a cell-based model structure was chosen. This could be based on the georeferenced grids of a GIS, and be compatible with GIS formats. The grid structure allows for a strong spatial component to the data analysis, and can cope with the introduction of a temporal change component.

The primary functions of the model are:

1: accept gridded input data (rasters).

2: process this data in order to compute a variety of parameters, including ablation, snowfall, mass balance, growing season, vegetation cover and snow cover.

3: display this information visually and compute output statistics, and allow cursor enquiry of individual cells.

4: allow customisation of the selected area in order to carry out a variety of experiments on different locations across Iceland.

5: allow evolution of the model through time.

A conceptual version of the model structure is shown in Figure 4.2. It shows the simple input, processing and output units as 'black boxes' – without the coding details. This design needs to incorporate the above aims, and be formed into a practical model architecture.

IV.2.2: Model construction – major Java classes in the model

Early on it was decided that the Java programming language would allow the achievement of the primary model objectives. The Java language is object oriented, meaning that different 'objects' are created by constructors (which are pieces of code that create specific types of object) and operated on by 'methods'. These methods can add input into the processing of an object, and return outputs to another part of the program. Java classes are sections of code that encapsulate an object and it's associated methods, and form the building blocks of the overall program. The Java language allows the programming of both the model and of the display and input/output, creating an integrated package. The classes allow the onstruction of the model to follow much of the original design, and additionally the design allows for model components to communicate with other elements of the program that will provide additional information (for example the addition of drainage onto display outputs). The actual model architecture is shown in simplified form in Figure 4.3. It is very similar to the

initial model design, with the exception that data processing is done by three separate program units – *PDDSum, PrecipModel* and *ELAFinderComplex.* There is also an additional management step in *ClimateControl.* This level is not strictly necessary, as the *ClimateControl* operations could be done within *Ice_model_Run*, but it reduces the size of *Ice_model_Run* to something a little more manageable! In terms of model structure, it limits *Ice_model_Run* to the role of the interface with the user, and delegates the climate model organisation to *ClimateControl.*

The key classes within the Iceland raster model are (including those in Figure 4.3), and are included in Appendix 2:

Iceland_model_Run: The umbrella class for the whole model. This class generates an Iceland_model_Run object which constructs a graphical user interface (GUI) for the model using the javax.swing toolkit. The GUI is a window and menu-based interface allowing for the control of many components of the model interactively. *Iceland-model_Run* manages inputs and outputs through the GUI, accepting user commands, sending model control commands to the ClimateControl object, and constructing display objects for the output windows.

ClimateControl: An object of this class accepts commands from Ice_model_Run to carry out different manipulations of the data. It stores some summary data, but primarily controls the operation of *PDDSum*, *PrecipModel* and *ELAFinderComplex.* A ClimateControl object is created or replaced each time a new version of the model is run (for example a new object is created if the location or the resolution is changed).

PDDSum: The object that generates the ablation data for the model. PDDSum calculates individual degree-day values using the pddCalc method and the mean temperature.

PrecipModel: PrecipModel controls the input of both temperature and precipitation into the model (using tools from SimpleIO). This includes generating monthly values from the annual precipitation grids, and importing the monthly temperature data. Values for the annual temperature curve eqution are calculated with this class. PrecipModel's third main function is to calculate snowfall, through integrating the temperature and precipitation data for a specific grid cell.

ELAFinderComplex: The ELAFinderComplex object has two functions. The first is to generate the equilibrium line altitude using precipitation and temperature data and snowfall calculation methods of *PrecipModel.* The second, which can be independent

of the first, is to calculate temperature lapse rates from the given temperature and topographic data.

Raster: The Raster class is one of the key components of the model. The mode is cell-based, and much of the data preparation and some of the data storage is done via this class. In effect the class is a mini-GIS, as some of its many methods have been inspired by analogues with the GIS architecture (including focal functions, reclassifying, calculating drainage networks, clipping and replacing sections of rasters). The layout of the raster class is based around the Ascii Grid file format for ESRI's ArcGIS software. There is a six line header with spatial referencing data, but no projection information, and then the data, which is arrayed row-by-row with spaces between the values. All inputs and output rasters in the model are in this format. Raster objects are created in most of the other classes when handling the various spatial datasets.

SimpleIO: An input/output toolkit used when objects need to communicate beyond the realm of the model program. This includes file handling, and output file creation (ascii files, comma separated data files and jpeg image files).

RasterPanelVik: manages the image preparation for output. This involves bringing together one or (frequently) more Rasters as well as occasionally vector data (points/lines/areas).

RasterScalePicker: An accessory class that controls the visualisation of the data output, including the colour scheme for particular raster outputs, and the minimum/maximum values within that scheme.

There are a variety of other classes that also contribute to the model, all within the 'afc' Java package – these are listed in Table 4.1. This includes a number of classes used for additional conceptual farm location modelling, which is being linked to the climate models.

The data pathways for the model are mapped out in Figure 4.4. It shows the progress of the three primary data sources in the model (temperature, precipitation and topography) through the processing modules in *PrecipModel, PDDSum* and *ELAFinderComplex* and some of the outputs that are created. The details of the collection and processing of temperature and precipitation data are described below in sections IV.2.3 and IV.2.4.

IV.2.3: Data sources

Raw meteorological data is available for 83 weather stations situated around Iceland (Figure 4.1). The data comprises of monthly observations of various weather phenomena, including precipitation and temperature. Not all stations have complete records, and stations were excluded that had fewer than 10 years of information, either for temperature or for precipitation. A Java class (WeatherAnalysis) was written to extract, collate and average the data for each station per month. The output of this includes (per month) mean temperature, mean daily maximum, mean daily minimum, and mean monthly precipitation.

With a suitable distribution of stations across Iceland, it would be possible to generate a precipitation and temperature model for Iceland directly from this data. Unfortunately, the proximity of most of the stations to the coast and the lack of data available for interior and high elevation sites made this task very difficult, as there would be large errors for medium to high elevation cells (Fig. 4.1). Therefore some alternative methods have been used to construct the mass balance model.

Precipitation data

Given the limitations of the weather station data, the best estimate for mean precipitation was the map of mean annual precipitation for 1971-2000 created by Veðurstófa Íslands (Figure 4.5). In order to be comparable with the available monthly temperature data, mothly precipitation was required. When graphs of precipitation are produced for the year for any one of the weather station locations, it can be seen that the variation in precipitation follows an approximately sinusoidal trend throughout the year (Figure 4.6). Consequently, for the basis of an accumulation model, constant precipitation could not be assumed as the variations are substantial (up to 50%) and will affect the modelled accumulation. Annual variation was estimated using the data gathered from the weather stations. A sine wave of the form:

$$y = P_m + v_p.\sin(x + \alpha_p) \quad (1)$$

can be used as an approximation for the precipitation variations during the year, where P_m is the mean monthly precipitation, v_p is the amplitude of the precipitation variation and α_p is the offset of the sine in radians if a year is represented by 2π. v_p increases in the wetter parts of Iceland, while α_p determines the timing of the wetter and drier times during the year. v_p and α_p were determined for the precipitation data from each of the

weather stations (using the Java class SineRegress). Using these values, v_p was modelled using a regression based on the mean precipitation, distance from coast and elevation. Percentage precipitation was used as opposed to absolute values, as the regression otherwise predicts extreme values (both wet and dry) in areas with few data points (Figure 4.7). The regression has a relatively low r^2 of 0.61, but modelled the variations more closely than the use of a spline or kriging. α_p was interpolated using a tension spline, as the equivalent regression function produced very low r^2 values (Figure 4.8).

The result of this modelling was an estimation of timing and magnitude of precipitation variation about the mean value determined from the map.

Temperature data

There is alternative data available for temperature, in the form of maps created by Gylfadóttir (2003; Figure 4.9). For each month, there is a 0.5' resolution raster map of mean temperature (this corresponds to a cellsize of approximately 0.4 x 0.9km). This represents modelling, described in Gylfadóttir (2003), of the temperature variations across Iceland during each month of the year, using data from 1961 – 1990. Interior locations are interpolated using a small number of automated stations, with data gathered during the last 15 years. Despite the interpolation, they remain the best available estimates of temperature for the whole of Iceland, given the lack of observational data for the interior. Gylfadóttir (2003) has also produced mean maximum and mean minimum temperature maps in the same form.

The 12 monthly mean maps were used to construct a model of temperature variation throughout the year. Average temperature closely follows a sinusoidal pattern, with lowest values in late winter, and highest values in late summer. For each cell, three curves were fitted to model these variations. Use of a modelled curve rather than interpolated data, allows greater ease in the calculation of mass balance and the equilibrium line, and potentially greater accuracy at the summer and winter turning points.

In the first case, a simple sine curve (similar to equation 1) was fitted for each cell of the map using the mean temperature as a baseline, and the resulting v_t and α_t values recorded. Such a model poorly represents the temperature variation, due to a late winter dip in temperature in Iceland. Mean temperatures are frequently similar (or

lower) in March when compared to January and February, with a steep rise between April and June. The fitted curve therefore overestimates the late winter and spring temperatures, while in compensation underestimates summer temperatures (Figure 4.10).

The second case was to use two sine curves of the form in equation 1, one for the summer data (May to September) and one for winter (October – April), with each curve varying around the same mean. This adjusts the best fit to more appropriately model each season, though there is the considerable problem of a hiatus in each curve. In particular, this provided a better estimation of summer temperature.

The third method was to use a different function which while still cyclical over one year, is a closer fit to the observed data. Embedding a second sine curve within the original has the effect of narrowing the peak and broadening the trough, therefore modelling the temperature variation significantly more accurately throughout the year, including the late winter dip. The equation is of the form:

(2)

$$f(x) = (T_{med} - T_{\lim}) - (T_{amp} + k_4).\cos(x - k_1 + k_3 \sin(x + k_2))$$

(3) $$T_{amp} = (T_{\max} - T_{\min})/2$$

(4) $$T_{med} = T_{\max} - T_{amp}$$

k_1 to k_4 are empirically-derived constants determining the shape of the curve; k_1 and k_2 control the relative offset of the respective curves, k_3 adjsusts the magnitude of the embedded sine wave, and k_4 is a small positive value to improve the fit of the overall curve whose amplitude is half of the temperature range. k_1 and k_2 are related to π, as they represent angles in radians. This curve produces a substantially improved fit, and therefore a better estimation of temperature variation for the model (Figure 4.11).

Combining the temperature and precipitation data creates the accumulation model. The objective of fitting the curve is to extract the roots of the equation, where the temperature drops below a threshold (in this case 1°C). Precipitation occurring during the period of the year where the average temperature is below this value will fall as snow, and therefore be accumulation. Using the roots derived from the temperature data, precipitation can be integrated between these roots to estimate total snowfall.

This process is carried out in the Java class PrecipModel, with the output being modelled annual snowfall for each cell in the grid.

IV.2.4: Calculating ablation and vegetation distribution

A simple degree-day model was created, using the monthly temperature maps. The equation is:

$$PDD = k_{pdd} \cdot \sum_{i=1} T_d : T_d \geq 0 \quad (5)$$

The equation used is similar to the most successful equation of Guðmunsson *et al* (2003), who tested various degree-day models against energy balance models for Vatnajökull, although the temperature values are taken from the maps of Gýlfadóttir (2003). Degree-day modelling requires daily temperature data, which is unavailable for either the weather stations or for the whole of Iceland, and so daily values were estimated using a probabilistic method. The model takes in as its inputs the 12 monthly mean temperature maps, and the monthly maximum and minimum maps.

Using these, the number of positive degree days can be calculated for each month. Daily variations can be estimated from the mean monthly temperature in Iceland, as there is a reasonable relationship between mean monthly temperature and the standard deviation of the daily monthly temperatures. Equation (6) was empirically determined from daily meteorlogical data from 6 weather stations for up to the last 24 years (Figure 4.12).

$$St.Dev = 3.3057e^{-0.0577T_m} \quad (6)$$

Consequently a standard deviation for the month can be determined depending upon the monthly mean temperature. For each month, the probability of a temperature value occurring within each month depends on the distance from the mean temperature. A z-table of probabilities (p-values) is used to calculate degree-day variations.

$$PDD = k_{pdd} \cdot \sum_{i=1} \left(T_i \times prob_i \times days_m\right) : prob_i \geq 0.01, T_i \geq 0 \quad (7)$$

Summing this for all the positive temperature classes (eg. 2-3°C) where the probability is above a minimum cut-off value of 0.01 gives the pdd sum (Figure 4.13). The cut-off value excludes the influence of unreasonably large temperatures, which, although

occurring at low probabilities (ie a probability of occurring on less that one day per month), will tend to slightly increase the PDD sum.

A degree-day factor, k_{pdd} is used to calibrate the pdd values against actual ablation, and so convert the degree days into metres of ablation. This varies depending upon whether ice or snow is being ablated. For the simple model, a k of 0.007m $°C^{-1}$ is used. In Iceland, values of 0.056 and 0.077m $°C^{-1}$ are reported for snow and ice respectively by Johannesson *et al*, (1995), with the higher value for ice being due to reduced reflectivity. Guðmundsson *et al* (2003) report a wide range of measured degree-day factors for Icelandic glaciers, mostly for medium to high elevation sites. For ice, they range from 0.004-0.0045m $°C^{-1}$ on the lower part of Breiðamerkurjökull to >0.007m $°C^{-1}$ on Tungnáarjökull. Corresponding values for snow are ~0.005m $°C^{-1}$ on Breiðamerkurjökull to >0.009m $°C^{-1}$ on Dyngjujökull. Hock (1999) introduced a direct solar radiation parameter, in order to improve high-resolution degree-day modelling for individual glaciers. This has not been done in this case, due to the coarse resolution of the input data, although some experiments were carried out using solar elevation and aspect as an input. In most of the results presented here, 0.0065m $°C^{-1}$ was used as an average, as the ablation of snow versus ice was not examined. Variability in Iceland is in part due to the incorporation of aeolian sediment and tephra into the ice, giving Icelandic glaciers their characteristic 'dirty' appearance.

For determining the distribuition of vegetation, a similar approach is applied through the PDDSum class. The degree-day approach to calculating vegetation depends upon there being sufficient annual degree-days over the period of a year, and this limit is coded as:

```
if((pm.getIgnoreVal(i,j)==false)&&(pddVegFile.getValue(i,j)<(vegLim*150)))
        pddVegFile.changeValue(i,j,0);
```

where the vegLim is the minimum growing temperature. A cell with an annual PDD below the vegLim*150 is assigned a zero value, as not having successful vegetation growing on it. This gives degree-day limits for grass and birch of 600 and 1025 degree-days respectively, as used in Ólafsdóttir, (2001). Monthly limits are similarly defined.

IV.2.5: Calculating Accumulation

Snow accumulation for each cell in the model was calculated using a combination of the mapped temperature data, the observed 1971-2000 precipitation, and the modelled precipitation curve parameters for each cell. Snow can fall and accumulate at

temperatures below 1°C, therefore the timing of the average temperature falling to below this threshold was calculated.

The calculation of accumulation is the integration of the precipitation model between limits defined by the threshold temperature. The limits are the roots to equation (2) where each month is represented by a fraction of 2π radians. The midpoint of each month is used, so January is π/12, and December is 23π/12 radians respectively. The roots (x_1 and x_2), and the integration of precipitation are found by solving for x_1 and x_2 in the following equations:

$$(8)\ \cos^{-1}\left(\frac{T_{med}-T_{\lim}}{T_{amp}+k_4}\right)=x_1-k_1+k_3.\sin(x_1+k_2)$$

$$(9)\ 2\pi-\cos^{-1}\left(\frac{T_{med}-T_{\lim}}{T_{amp}+k_4}\right)=x_2-k_1+k_3.\sin(x_2+k_2)$$

Snowfall is calculated as the integral between the roots, x_1 and x_2:

$$(10)\ \int_{x_1}^{x_2}\left(\overline{P}+v_P.\sin(x+\alpha_P)\right)dx=\left[\overline{P}.x-v_P.\cos(x+\alpha_P)\right]_{x_1}^{x_2}$$

IV.2.6: Calculating Mass Balance and ELA

The calculation of the mass balance for each cell in the model depends on the accumulation and ablation in each cell. Total snow accumulated at temperatures below 1°C using equation 2 is balanced against total ablation from equation 6 for each cell of the grid.

The equilibrium line altitude is calculated by adjusting the temperature limit T_{lim} in equation 2 and the temperature value T_i in equation (7) until precipitation and snowfall balance at a given temperature offset. Using the local temperature-altitude lapse rate (calculated using linear regression for a 50x50 cell window of the raster DEM and average temperature), the elevation of the ELA can be determined for each cell of the grid.

IV.2.7: Calculating the length of the growing season

The growing season length is defined as the time when the average temperature rises above the selected minimum limit for the given vegetation type (e.g. 4°C for grass). It would be possible to model the growing season through the calculation of the monthly PDD values, so that if a mean monthly temperature rises above the limit, the frowing season is defined to have started. However, this is of insufficient resolution for growing seasons in Iceland, where the difference of a week or two can be critical to the balance between fodder production and livestock numbers. The modelled temperature curve can be used, and so the exact position the mean temperature curve crosses the boundary can be determined. Clearly, the model does not take into account the stochastic variations inherent in the beginning and end of the real growing season, but indicates the mean length of the season, and can test the sensitivity of that value to change. The dates of start and end are also computed, and the sensitivity of these to change can also be tested.

IV.3 Mass balance modelling – results

The model was constructed with a graphical user interface that is shown in Figure 4.14. This interface allows the quick and effective customisation of model output. Additional configuration files can be provided to automate the control process in order to produce the time-dependent outputs as well as sequences of model scenarios.

IV.3.1: Model validation exercises Mass balance and vegetation compared to the present day

Comparison curves of mass balance and topography were tested for two locations, where there is existing mass balance data. These are at Breiðamerkurjökull on Vatnajökull and at Hagafellsjökull on Langjökull.

Breiðamerkurjökull

Mass balance at Breiðamerkurjökull is modelled reasonably well. The equilibrium line is around 1050-1100m according to Guðmundsson *et al.* (2003), which is reasonably close to that modelled (Figures 4.15 and 4.16). The slight difference may be explained by the time periods covered in the data: the measurements are 1992-2000, and the model is 1961-1990. The ablation at the snout of Breiðamerkurjökull of ~10m per annum is very close to the observed value.

Hagafellsjökull

At Hagafellsjökull, the equilibrium line is also well modelled, being within 50m of the observed value (Figures 4.17 and 4.18). However, when split into ablation and accumulation, it can be seen that ablation values are slightly high at high elevations, while accumulation is also slightly higher. This discrepancy is most likely to be due to the location of Hagafellsjökull near the central part of Iceland where there are fewer weather stations: hence the temperature and precipitation gradients on Hagafellsjökull are slightly too high. However, the mass balance is still well modelled ,and the ablation at lower elevations is in reasonable correspondence with the observations.

IV.3.2: Iceland's mass balance

A map of the areas of positive mass balance in Iceland is presented in Figure 4.19. All main glaciated areas of Iceland (ice caps and smaller glaciers) are successfully modelled with the exception of Drangajökull in the northwest. This includes Hofsjökull and Þrandarjökull, some Tröllaskagi glaciers, and positive mass balance for Snæfellsjökull, Hekla and Torfajökull. The negative mass balance for Drangajökull is considered to be an artefact of the input temperature and precipitation. The overall model fit is very good, suggesting that the spatial coverage of the mass balance solution is good.

Using this mass balance data and lapse rate information, the height of the equilibrium line can be estimated (Figure 4.20), and so compared to the 1937 map (Figure 4.21). The major trends are comparable, though the satial pattern of drier, higher ELAs in northern Iceland is clearer. Lowered ELAs over high ground and ice caps are caused by the topography/precipitation relationship, and so in the case of the major ice caps, the ELA would be higher if the ice was removed.

If mass balance is calculated on a monthly basis, snowcover amounts can be estimated. This only provides average values, but can indicate the regions most often covered by snow at a particular time of the year.

IV.3.3: Distribution of vegetation

The distribution of grass and birch has been modelled for Iceland using the degree-day model (Figure 4.22). The distribution shows large changes in the area of potential vegetation given a temperature shift of 1.5°C higher or 1.5°C lower than the 1961-2000 average. The details of such shifts will be assessed in Chapter V.

IV.3.4: Modelling the growing season

With the threshold temperature for grass defined at 4°C, the parameters of the average growing season can be identified: start, end and length (Figures 4.23 and 4.24). The sensitivity to date and length changes can therefore be modelled.

IV.4 Catchment-scale glacier modelling – application of GLIMMER

The two sites where lacustrine sediment cores were taken, Skeiðsvatn and Hofsjökull, provided the targets for application of the GLIMMER ice sheet model (Figure 4.25). The objective of this was to attempt to use the model to simulate the buildup of glaciers in the two study areas to examine the sensitivity of the glaciers to temperature changes. The model was run using local temperature and precipitation patterns derived from the Java model and local meteorological data, then run to equilibrium ice conditions at a variety of temperature offsets from present day conditions. Precipitation was not altered as a factor, due to the lack of calibration evidence for past precipitation changes in Iceland (such evidence, including the Stykkisholmur record, exists for temperature changes into the last century).

GLIMMER is a three-dimensional thermomechanical ice sheet model, which incorporates the GLIDE (General Land Ice Dynamic Elements) ice sheet model (Hagdorn *et al.*, 2005). It is designed to be interfaced to a range of global climate models, and is being used for a series of ice sheet reconstructions and experiments.

GLIMMER has been used to conduct several experiments relating to the buildup and removal of ice masses in these two areas. The model has taken as input data the topography of the area. Constant parameters are sea level temperature, precipitation, temperature lapse rate, and temperature and precipitation range, derived from the Java model. These parameters can be constant due to the small size of the areas in question, or they can be input as raster datasets. GLIMMER is designed to operate on larger spatial scales than those applied here, but was used as it was the only available model that could be used to estimate sensitivity to glacier buildup or melt.

IV.4.1: Vatnsdalur

Application of GLIMMER to Vatnsdalur initially indicated no ice buildup, even when the mean temperature was lowered by 5°C. What little ice was produced was perched on top of the mountains, as the mass balance was controlled by the level of ablation on lower ground. One consideration with the Tröllaskagi mountains is that mass balance is substantially affected by snow drifting and accumulating in the northern-facing corries. This effect could be approximately parameterised using the Iceland climate model to influence the accumulation through a simulated southerly

wind (accumulation is increased or decreased depending on the gradient and aspect of the cell; Figure 4.26). This is a coarse and uncalibrated parameterising of snow accumulation, which could clearly be improved through the application of a more sophisticated snow drifting model, for example that of Purves *et al.*, (1999). Using the simple approximation, there is an improved fit in the model, with a 2.5°C temperature lowering producing the glaciers as seen in Figure 4.27.

IV.4.2: Hofsjökull

A similar application of GLIMMER was applied to Hofsjökull to examine the sensitivity of Hofsjökull to buildup or removal of ice. In this case, a temperature lowering of 3°C with no adjustment for drifting produces glaciers as shown in Figure 4.28, with ice a little more extensive than at present.

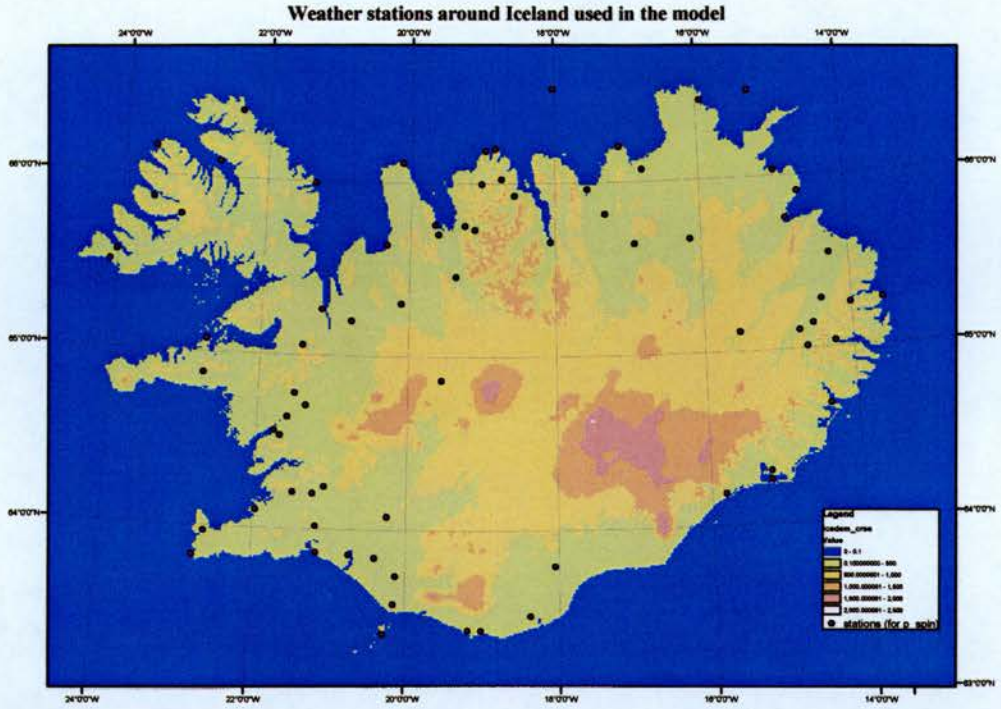

Figure 4.1: Location of weather stations around Iceland for which temperature/precipitation data is available.

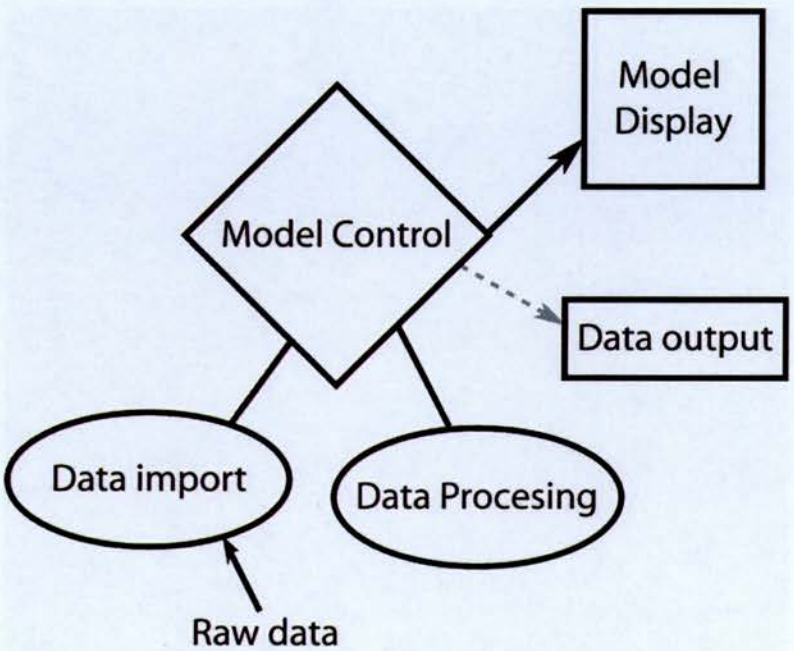

Figure 4.2: Conceptualised model structure – showing the pathway from raw data to model output.

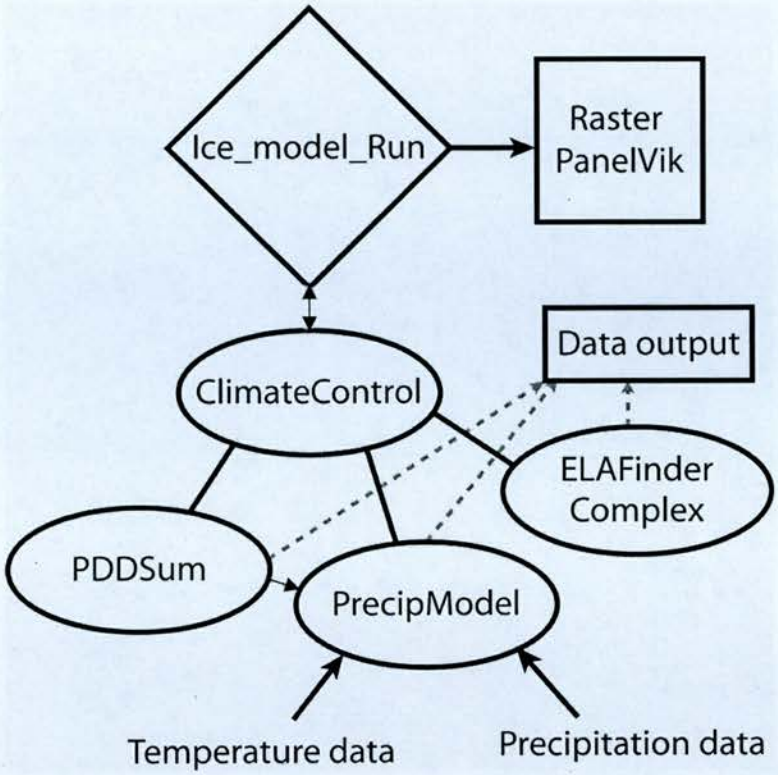

Figure 4.3: Actual model structure for the Iceland climate model – showing the pathways from raw data to model output.

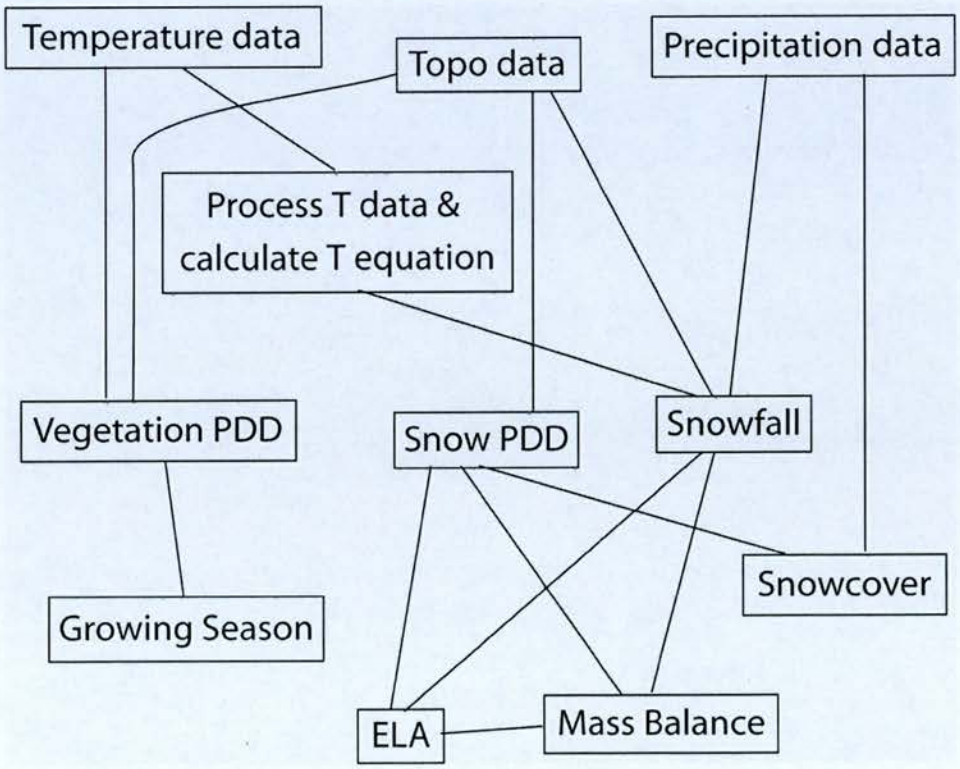

Figure 4.4: Data pathways in the model, from input data (top) to the various outputs.

Class Name	Package directory	Notes
GraphicsListener	afc.gui	update driver for animation
GradientPanel	afc.gui	display top layer's colour scheme
RasterPanelVik	afc.gui	model display
ClimateControl	afc.spatial	temp/precip management class
DrainageRaster	afc.spatial	drainage network calculation
ELAFinderComplex	afc.spatial	ELA and lapse rate calculation
Ice_model_Run	afc.spatial	Main model class
Line	afc.spatial	
PddSum	afc.spatial	ablation / vegetation PDD class
Point	afc.spatial	
PointsetRandom	afc.spatial	generate a random set of points
Polygon	afc.spatial	
PrecipModel	afc.spatial	extract, store and handles precip and temp data; calculates snowfall
Raster	afc.spatial	Handles and manipulates all rasters
Rectangle	afc.spatial	creates a Rectangle object – used for spatial footprints in raster manipulation
ThiessenPoly	afc.spatial	calculates Thiessen polygon lines for a series of points
ViewshedCalculator	afc.spatial	calculates the viewshed with in a given radius for use with Sun elevation data
Viking_Farms_Control	afc.spatial	manages a spatial model of 'farms' being integrated into the climate model
Viking_Farms_Run	afc.spatial	controls the GUI and i/o for the Viking farms model
GetTempCurve	afc.utilities	construct an annual temperature curve for one cell in the model
MonthsAndDays	afc.utilities	month information (days in the month & month names)
RasterScalePicker	afc.utilities	controls the colour scheme and scale for a Raster
Regression	afc.utilities	Calculate linear regression and correlation from two data series
SimpleIO	afc.utilities	input/output file handling: opening and saving files
SineRegress	afc.utilities	Fits a 1λ sine curve to a data series
GetWeather	afc.weather	Extract data from the raw Veðurstófa Íslands monthly data files for weather stations
WeatherAnalysis	afc.weather	select data types and construct *.csv files from the data in GetWeather

Table 4.1: A listing of the major classes designed and used in the model. The original Raster, Point, PointsetRandom and SimpleIO classes were first designed by Nick Hulton and Ross Purves, but have since been substantially modified and extended (particularly the Raster class).

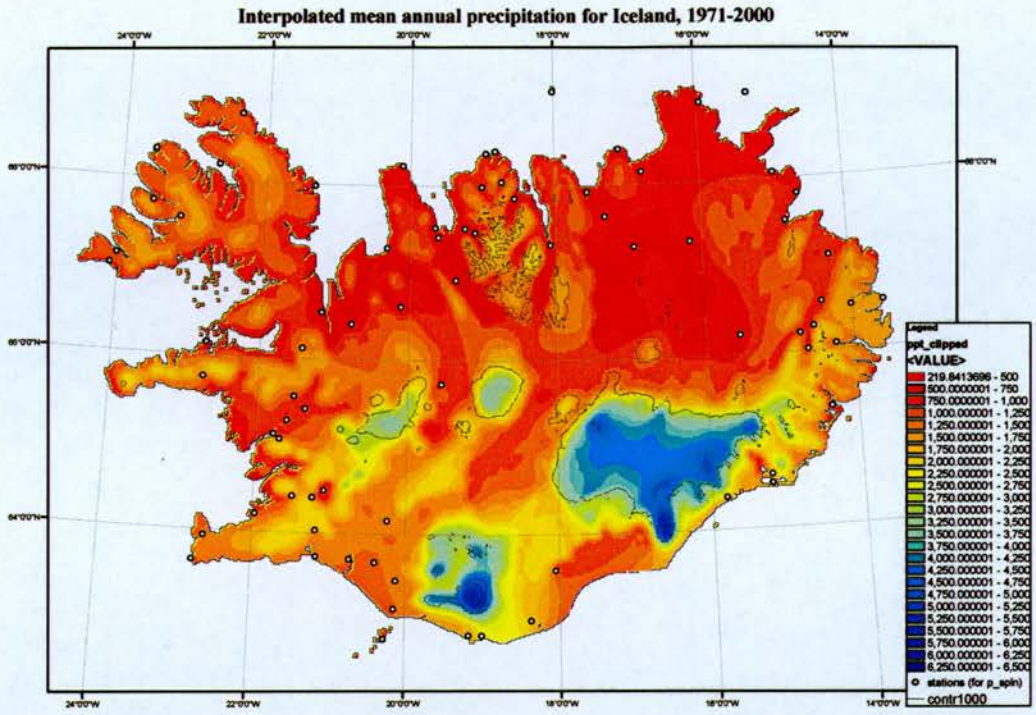

Figure 4.5: Map of mean annual precipitation in Iceland, 1971 – 2000 (digitised and gridded from a map provided by Halldor Björnsson, Veðurstofa Íslands).

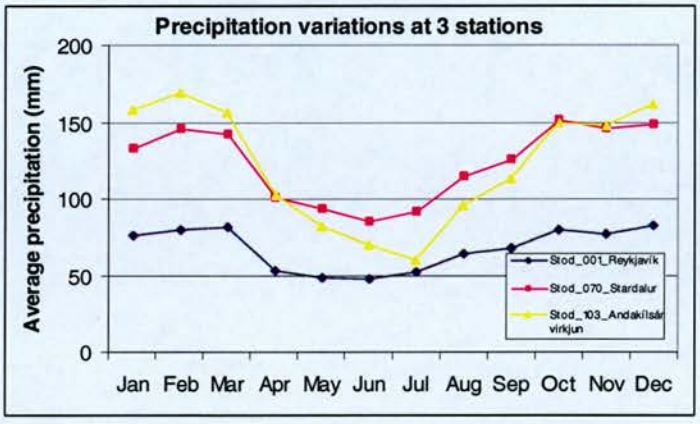

Figure 4.6: Selected precipitation data: Reykjavík, Stardalur and Andakílsárvirkjun, showing the approximately sinusoidal variation throughout the year.

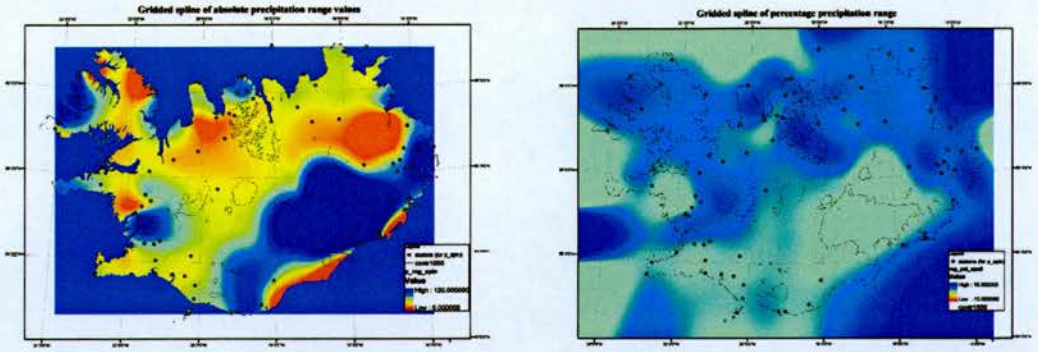

Figure 4.7: Modelled precipitation ranges (ie between maximum and minimum monthly values). They are based on station data, fitted with a spline. (a) Range modelled using absolute values, which leads to extreme excursions due to large absolute changes in total precipitation. (b) Range modelled with percentage precipitation change, producing a smoother surface with less extreme changes. High values (blue) are found more to the north and west, with lower values towards the southeast. Lack of data for central Iceland may cause an underestimation of the range for areas to the north and west of Vatnajökull. Extreme values offshore are due to the lack of points offshore, and are not used in the model anyway

.

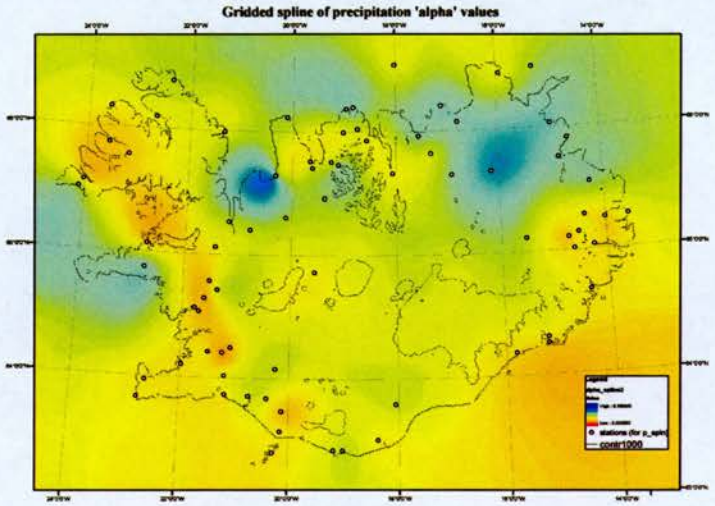

Figure 4.8: Modelled α_p values. The differences are relatively small across Iceland indicating the consistency of a winter precipitation maximum.

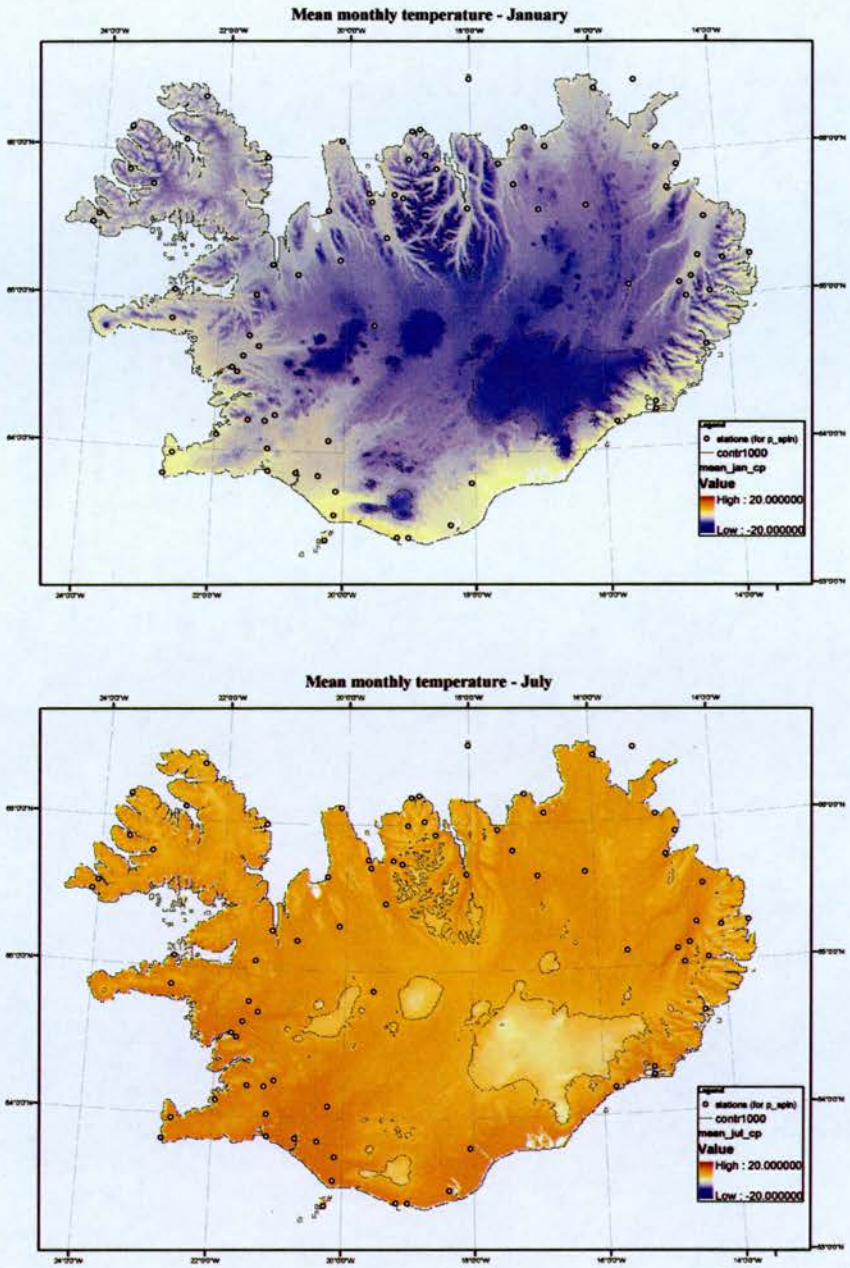

Figure 4.9: Gridded monthly mean temperature data, for (a) January and (b) July, from Gýlfadóttir (2003) and available online at http://www.vedur.is.

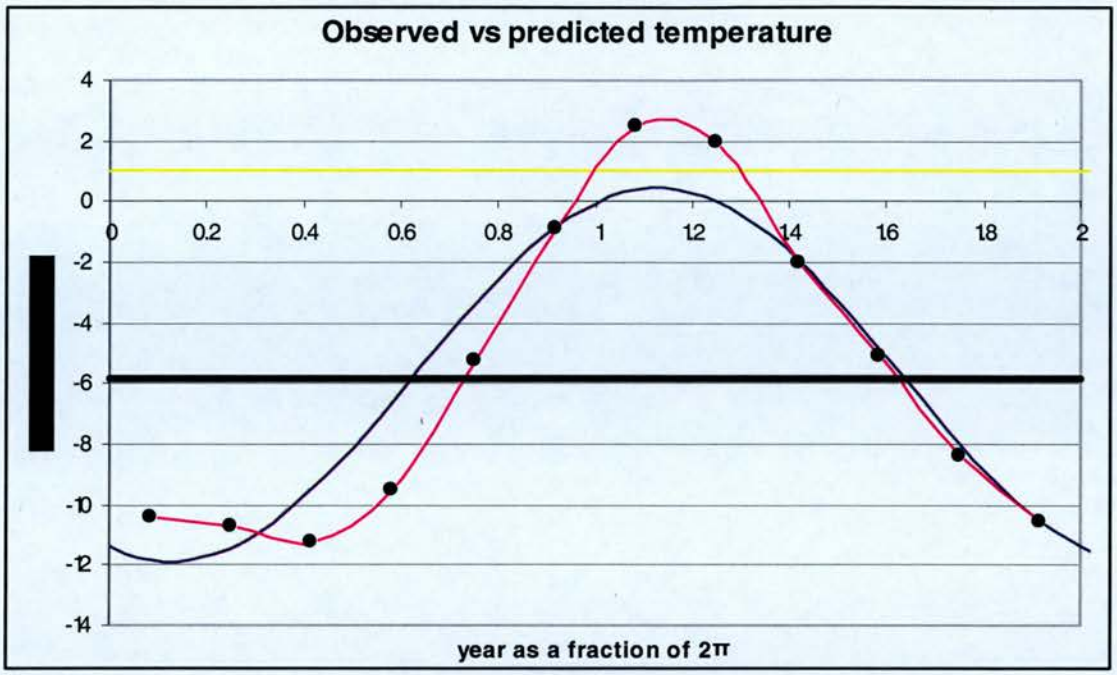

Figure 4.10: Fitting a simple sine curve does not adequately explain the temperature variations in Iceland, where there is a dip in mean temperature in March, and so temperatures are below average for more than 6 months of the year.

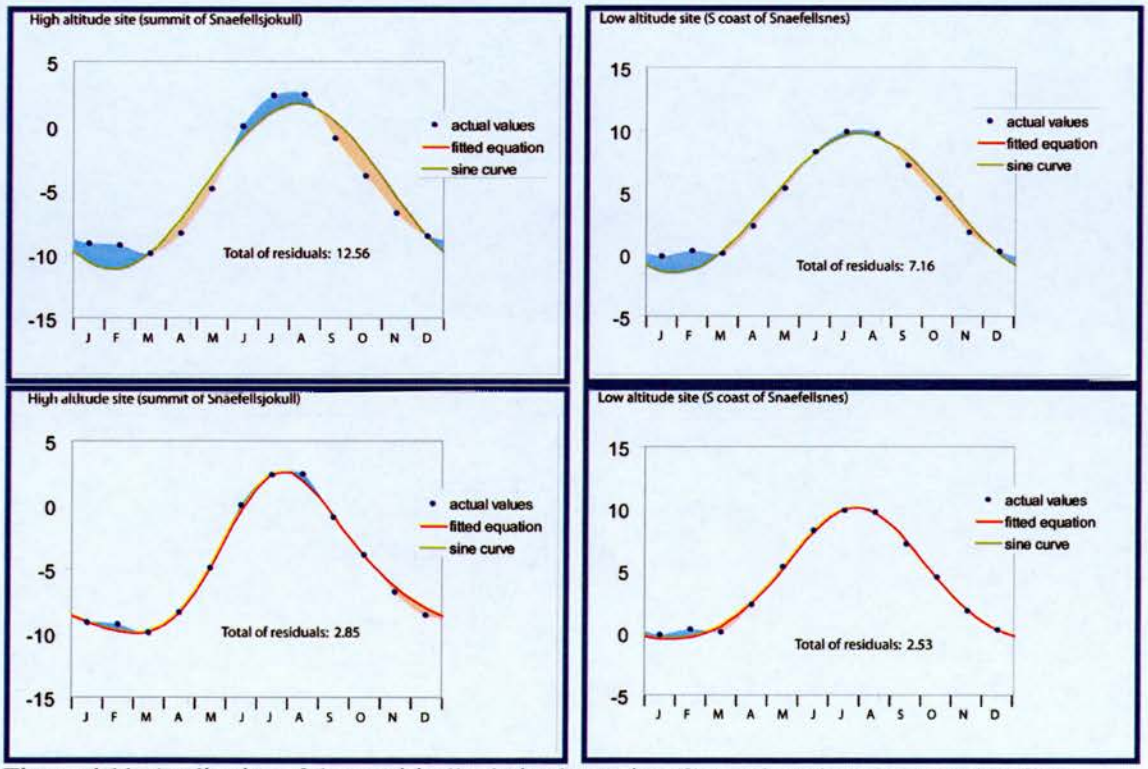

Figure 4.11: Application of the empirically-derived equation (2) to selected temperature data from Snæfellsnes: a high altitude site (left) and low altitude (right). The fit is considerably better than that in fig. 4.10. Residuals are blue and orange areas on the graph.

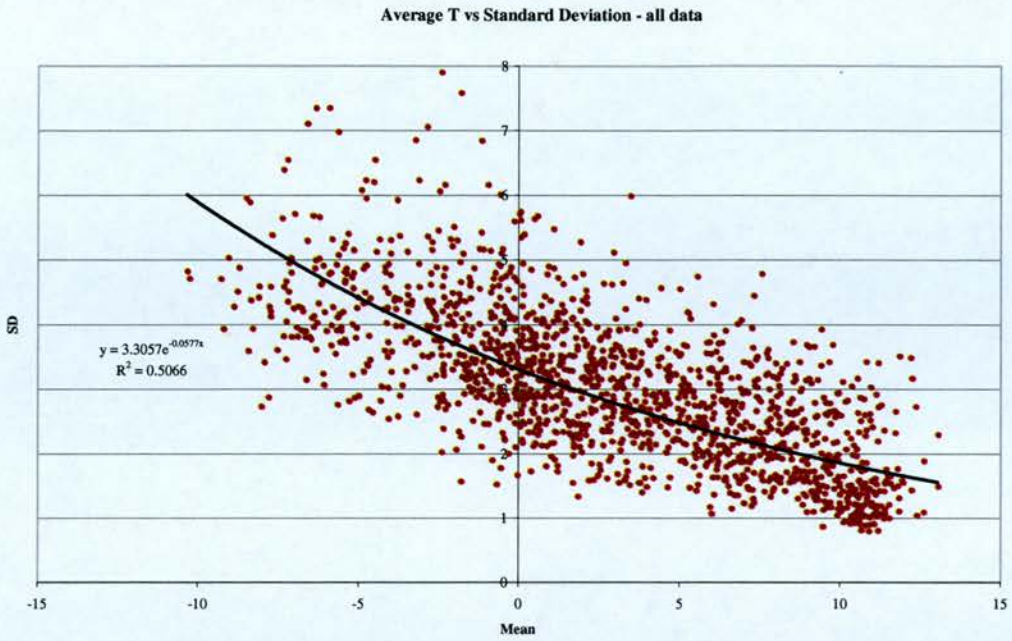

Figure 4.12: The relationship between mean monthly temperature and the standard deviation of daily temperatures.

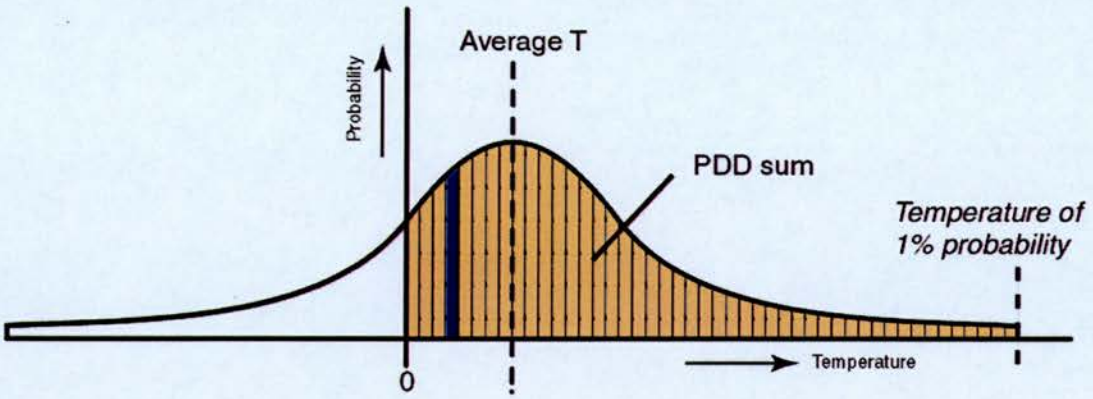

Figure 4.13: Calculation of PDD values using normal curve probability values and the average monthly temperature. The kurtosis of the curve is defined by the standard deviation, calculated from the monthly maximum and minimum temperature values.

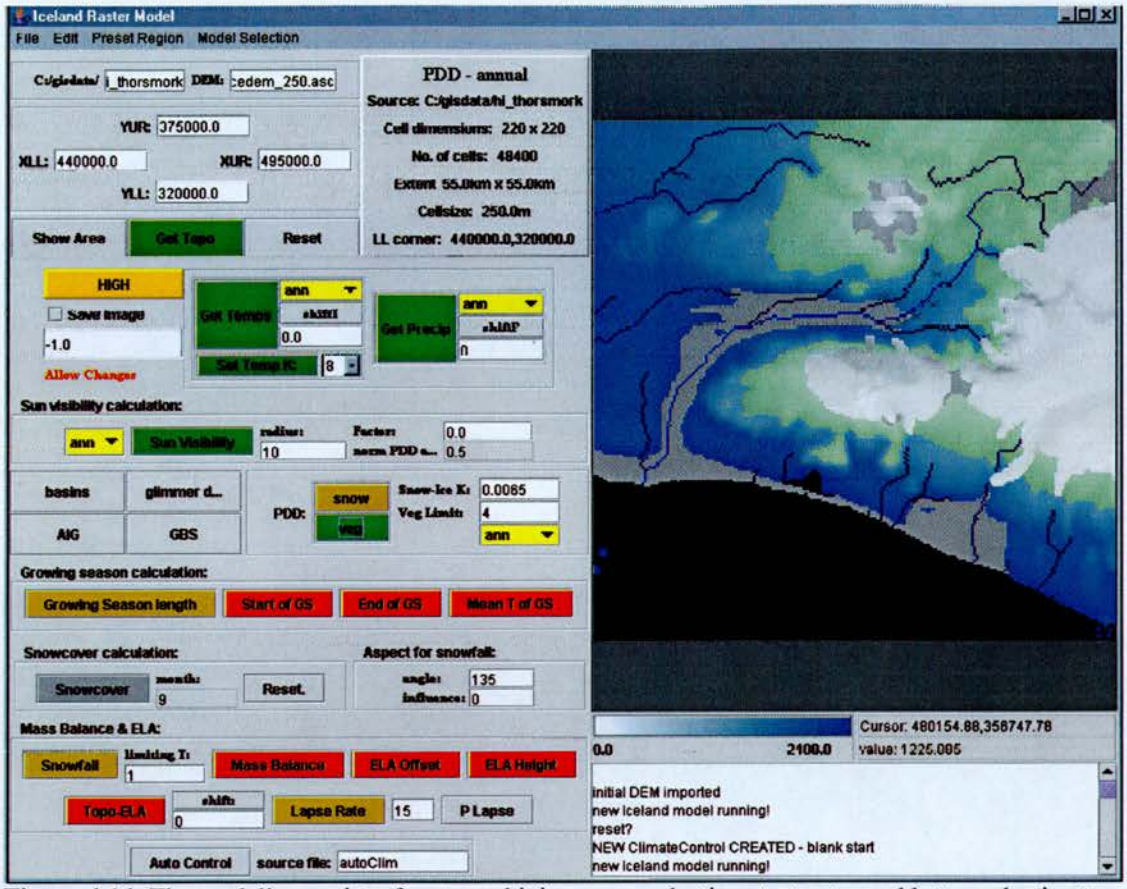

Figure 4.14: The model's user interface., combining menu selection, text entry and buton selection to produce outputs. Additional data processing occurs through files entered in the 'Auto Control' at the bottom of the screen.

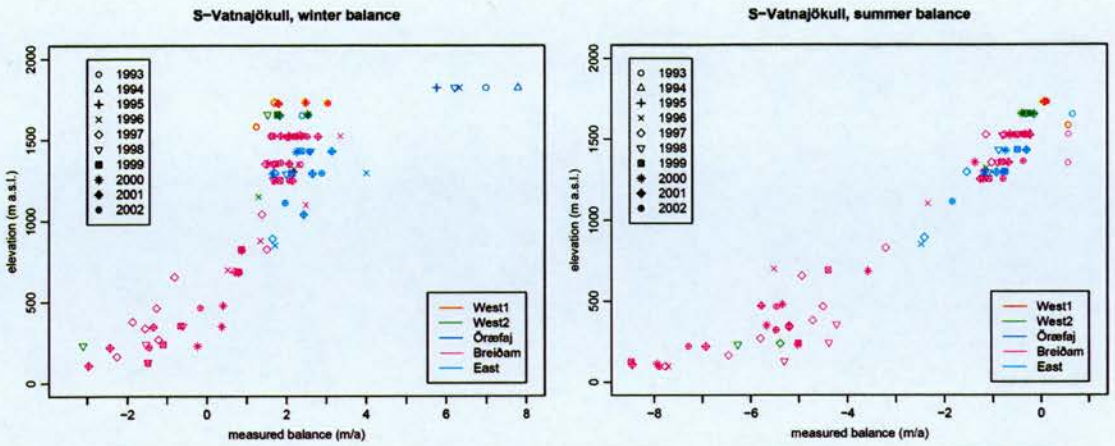

Figure 4.15: Winter/summer mass balance on the south side of Vatnajökull measured 1993-2002 (Aðalgeirsdóttir *et al.* 2004).

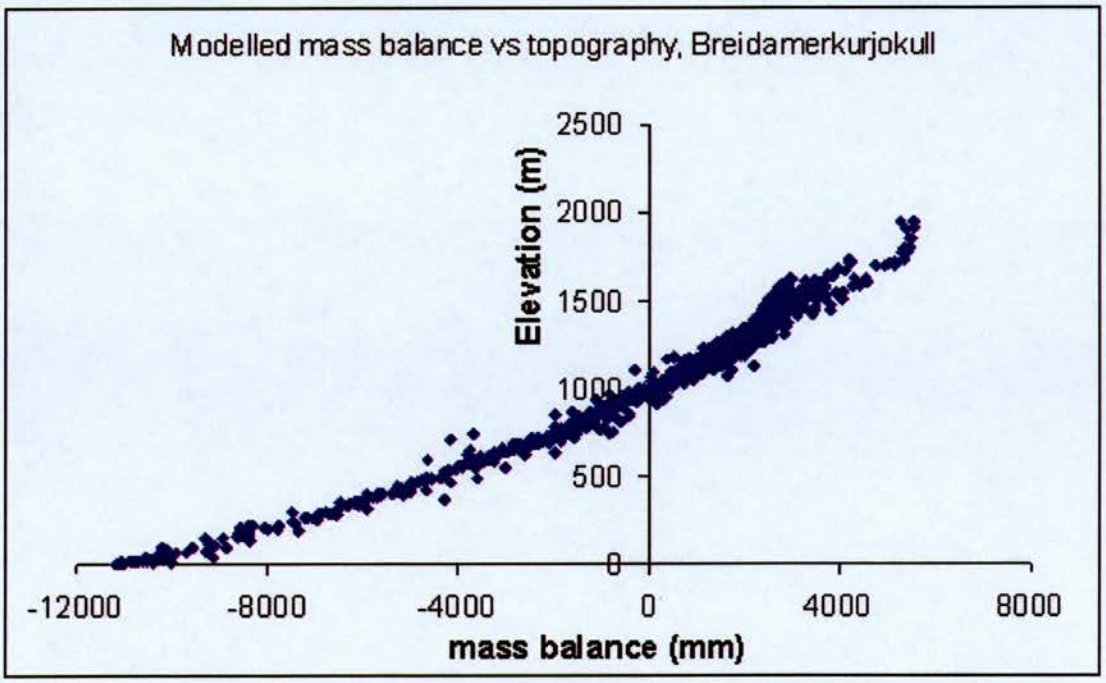

Figure 4.16: Annual mass balance predicted by the model in the Breiðamerkurjökull / Öræfi region of south Iceland. Although the point data from the graph in Figure 4.11 is scattered, the ELA can be determined to be around 1050-1100m, slightly higher than that predicted by the model. This may be explained by the time periods covered in the data: the measurements are from 1992-2000, and the model is 1961-1990, when average temperatures were slightly lower.

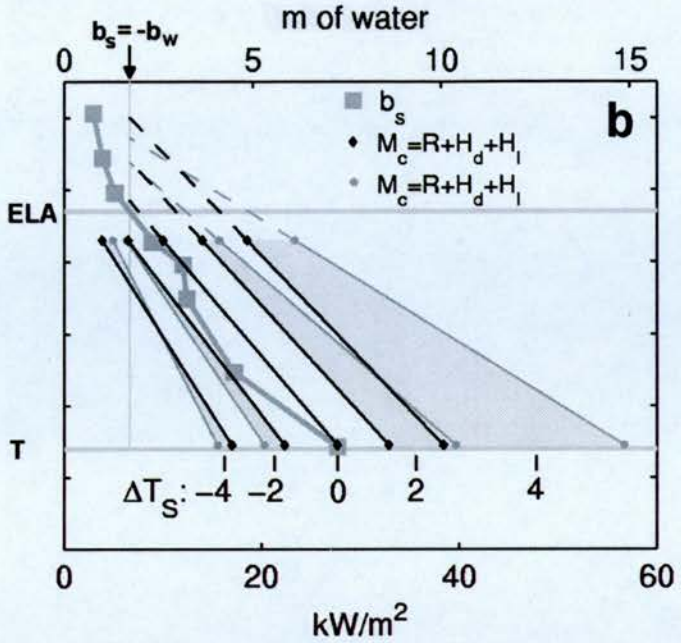

Figure 4.17: Ablation on Hagafellsjökull, Langjökull from Guðmundsson *et al.* (2003). The ELA is at 1150m.

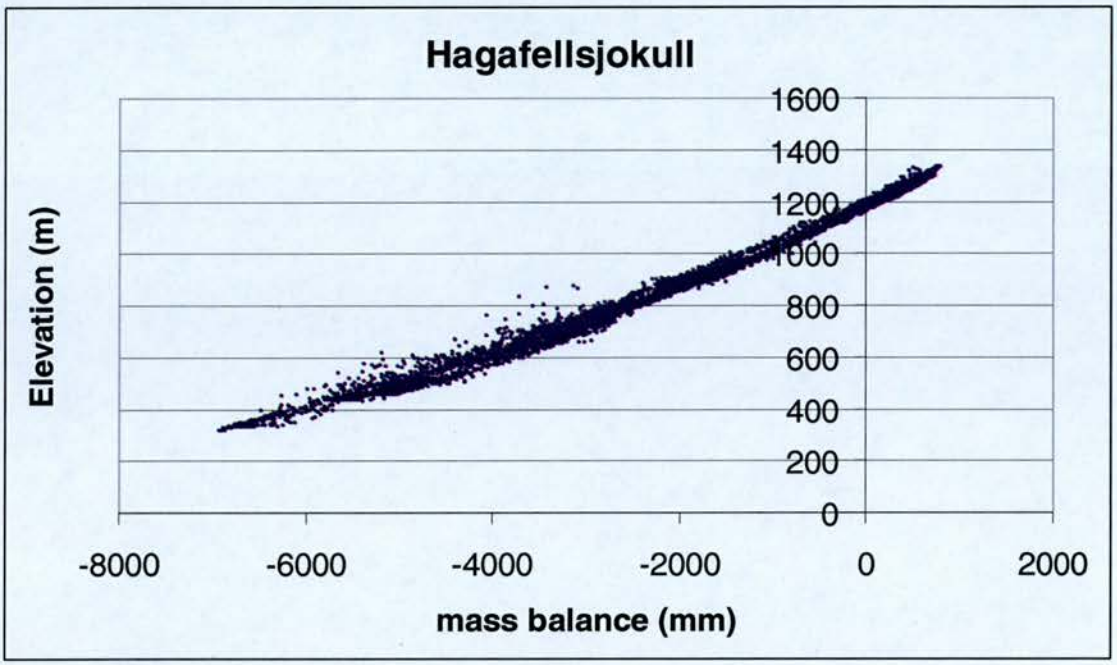

Figure 4.18: Modelled mass balance curve from Hagafellsjökull (Vestri and Eystri). The ELA is within 50m of the measured position.

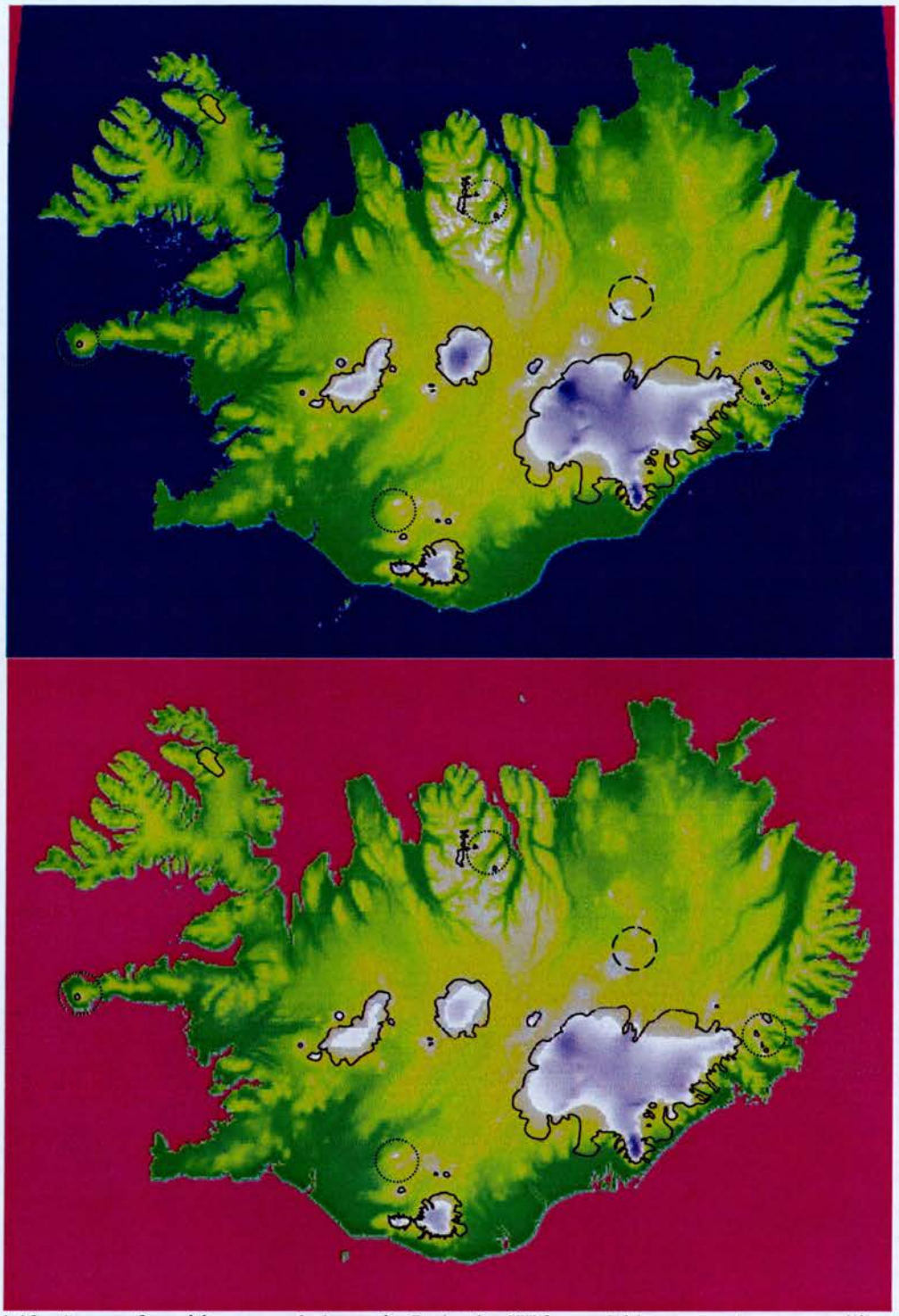

Figure 4.19: Areas of positive mass balance in Iceland. White and blue areas have a positive mass balance, and so should be glaciated; green areas have negative mass balance.
(top) Areas that would have a positive mass balance, given an even snowline of 1100m. This is the snowline on the south coast, but the interior is not well-modelled (for example Askja).
(bottom) The modelled snowline in the Iceland climate model. All main glaciated areas of Iceland (ice caps and smaller glaciers) are successfully modelled with the exception of Drangajökull in the northwest. This includes marginal glaciation at Hekla, Tröllaskagi, east of Vatnajökull and Snaefellsjökull (dotted circles), and high unglaciated mountains of Askja and Herðubreið (dashed circle).

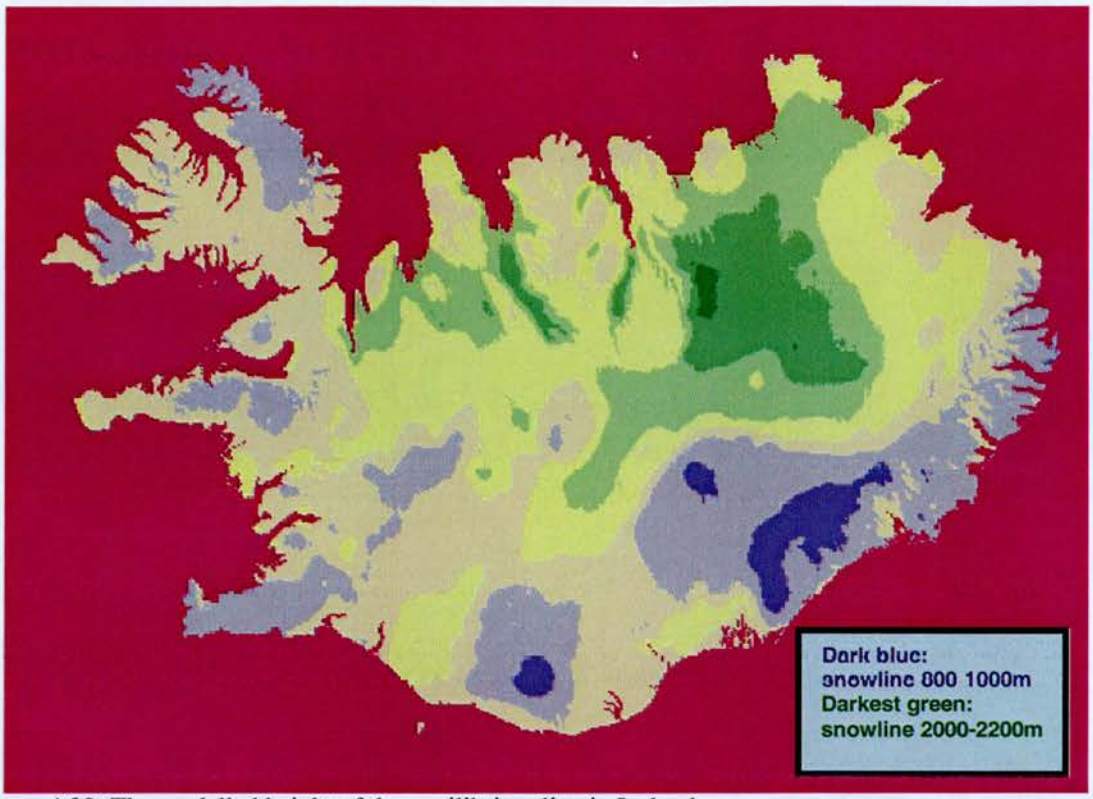

Figure 4.20: The modelled height of the equilibrium line in Iceland.

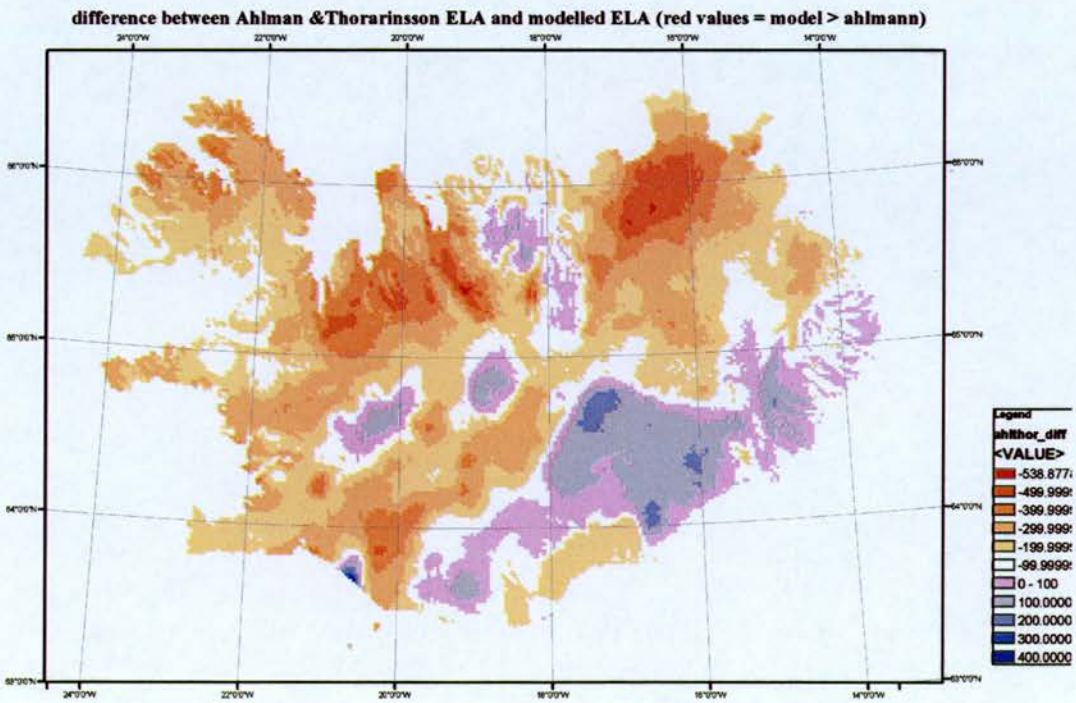

Figure 4.21: Comparison between the Ahlmann and Thorarinsson (1937) ELA and the modelled ELA. Red areas have a higher modelled ELA, and blue areas have a lower modelled ELA. In most areas, the difference is <100m, though larger discrepancies exist in northern Iceland (modelled ELA higher) and on Vatnajökull (modelled ELA lower).

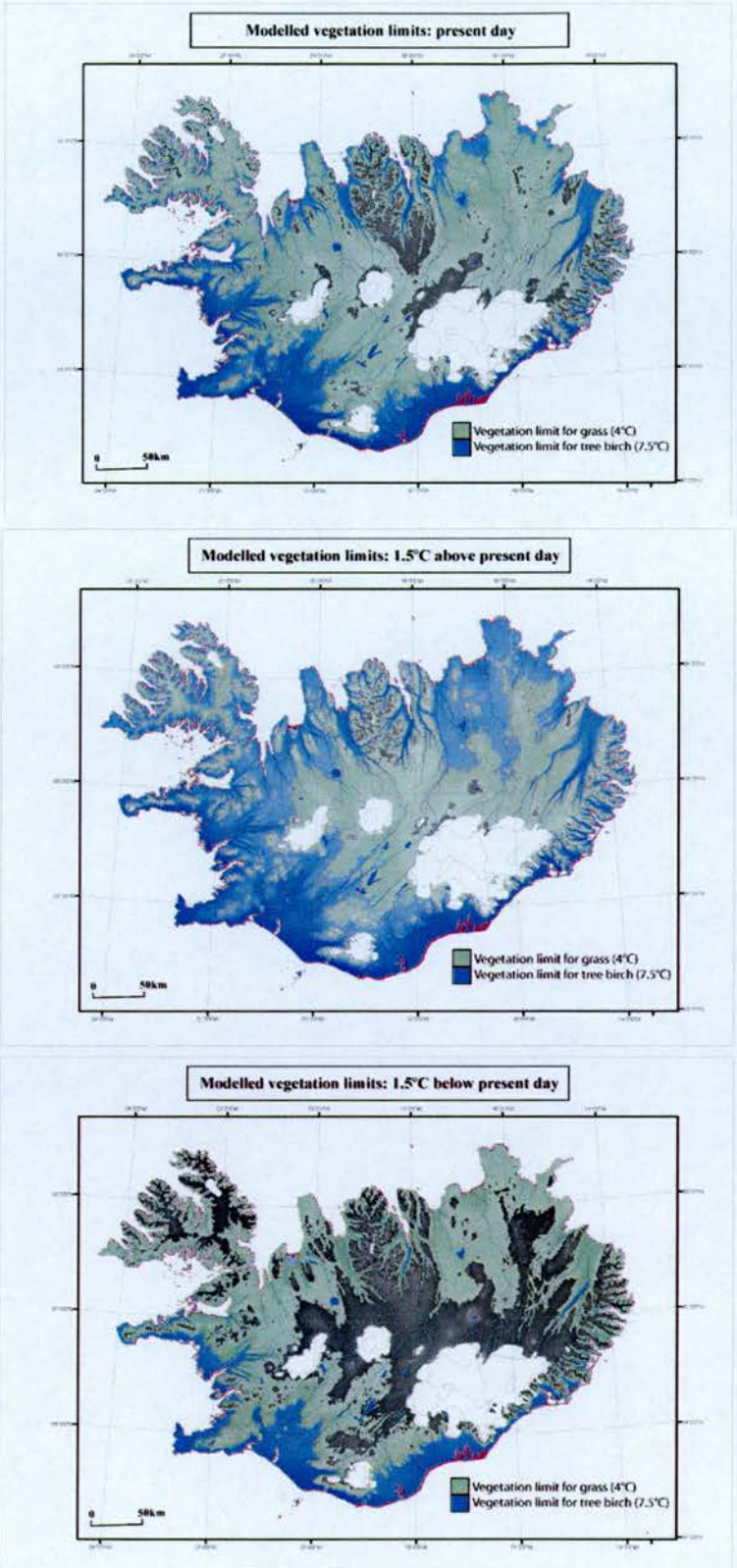

Figure 4.22: Modelled vegetation distribution for birch (blue) and grass (green), given temperatures at the 1961-2000 average (top); 1.5°C higher than average (middle) and 1.5°C lower than average (bottom).

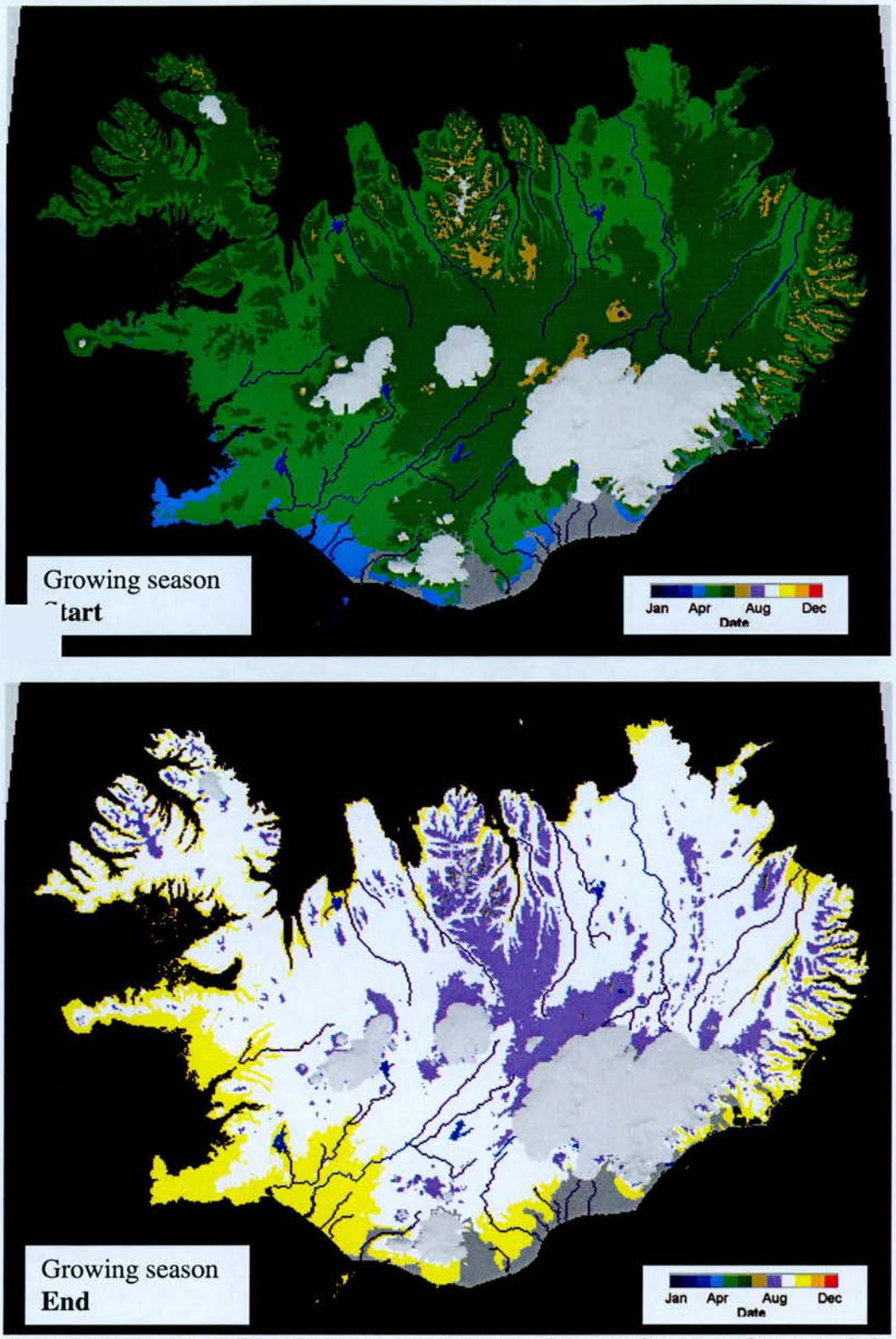

Figure 4.23: The date of the start (top) and end (bottom) of the growing season for grass. Colours relate to the month within which the event occurs, for example light green areas represent the growing season beginning in the month of May.

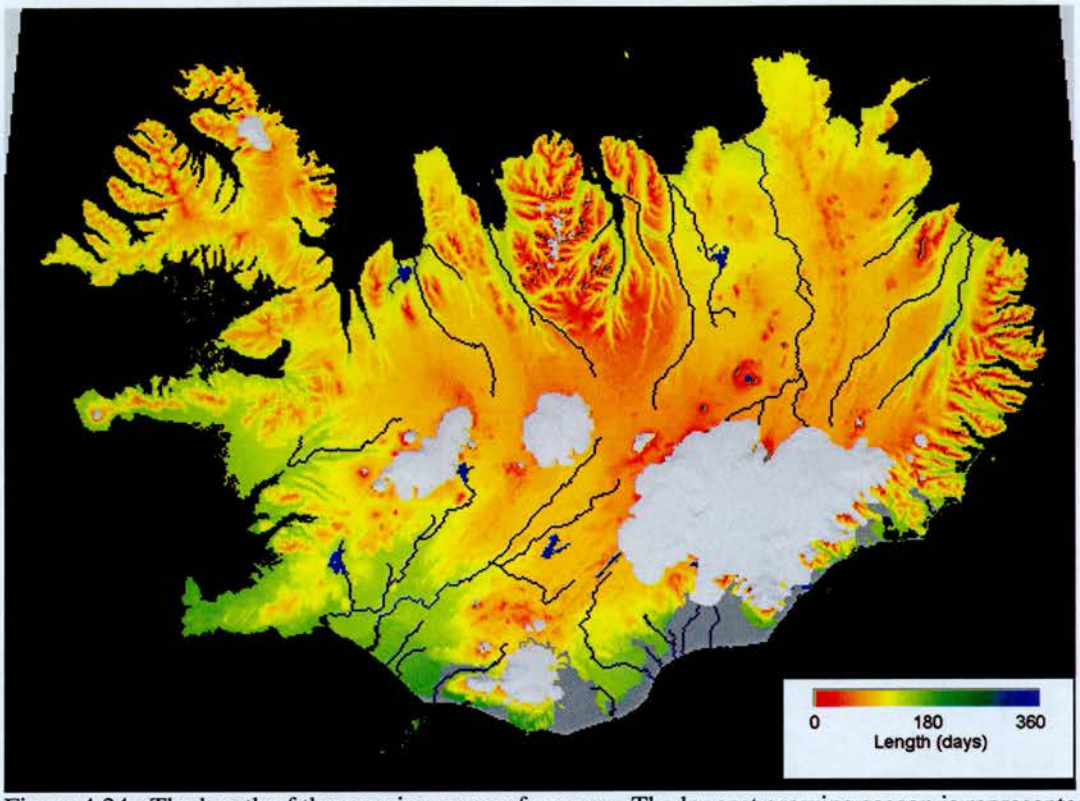

Figure 4.24: The length of the growing season for grass. The longest growing season is represented by yellow/green areas in southwest Iceland.

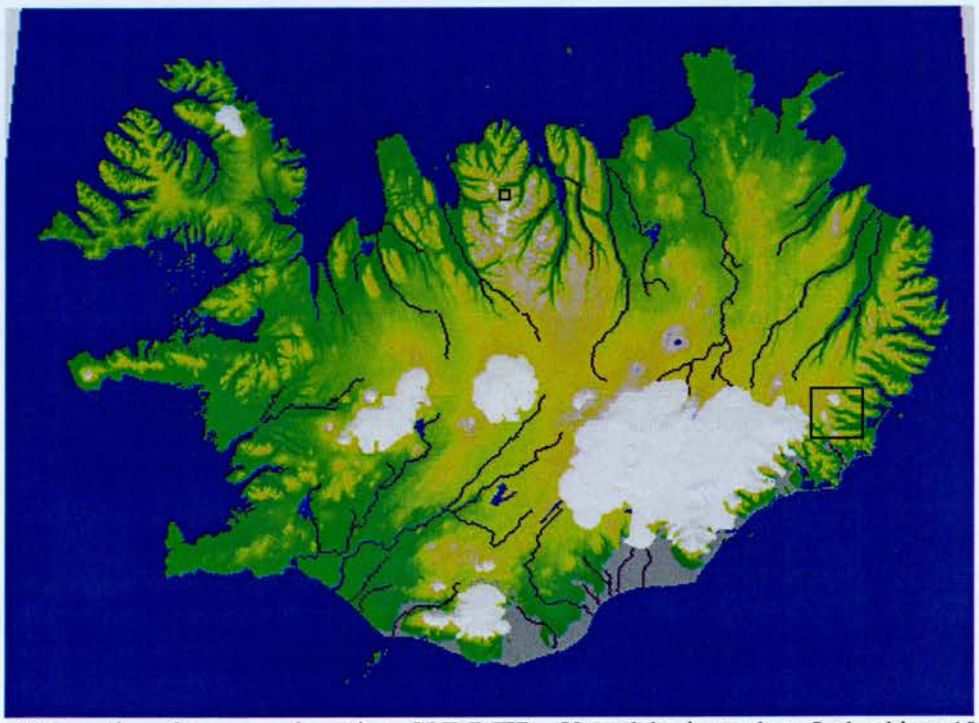

Figure 4.25: Location of two areas input into GLIMMER – Vatnsdalur in northern Iceland is at 100m resolution, the Hofsjökull area in southeastern Iceland is at 250m resolution.

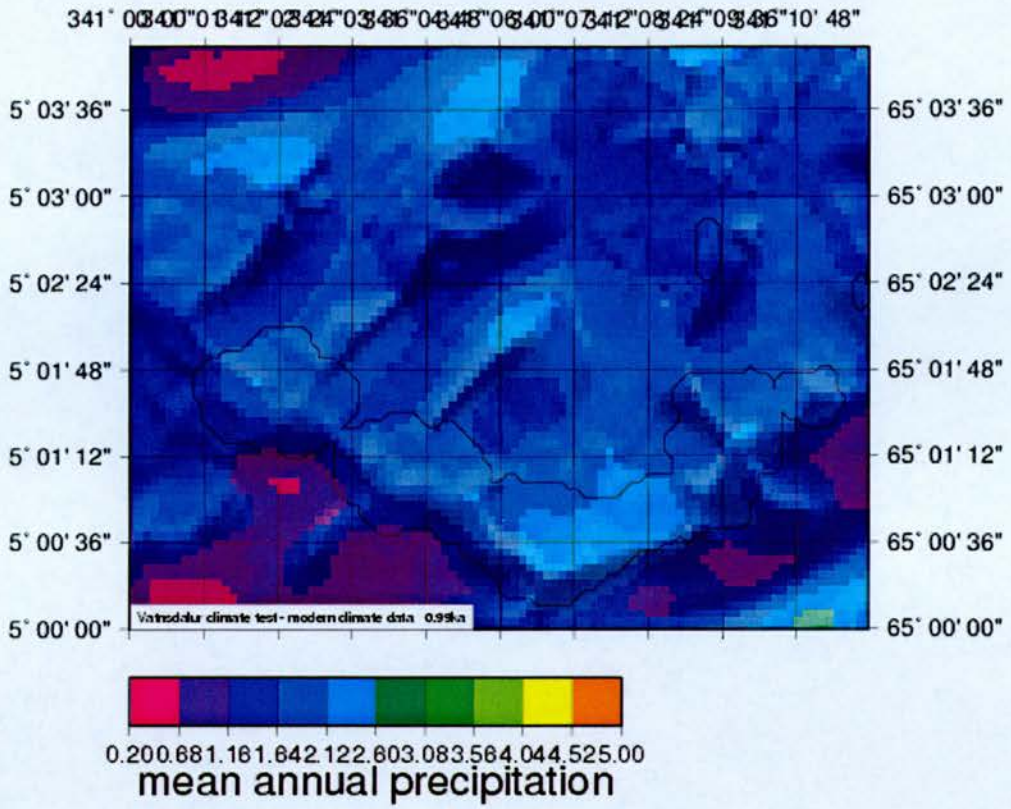

Figure 4.26: Modelled mean annual precipitation input into GLIMMER for the Vatnsdalur area. Precipitation has been shifted (using the mass balance model) to enhance accumulation to the north of high ground.

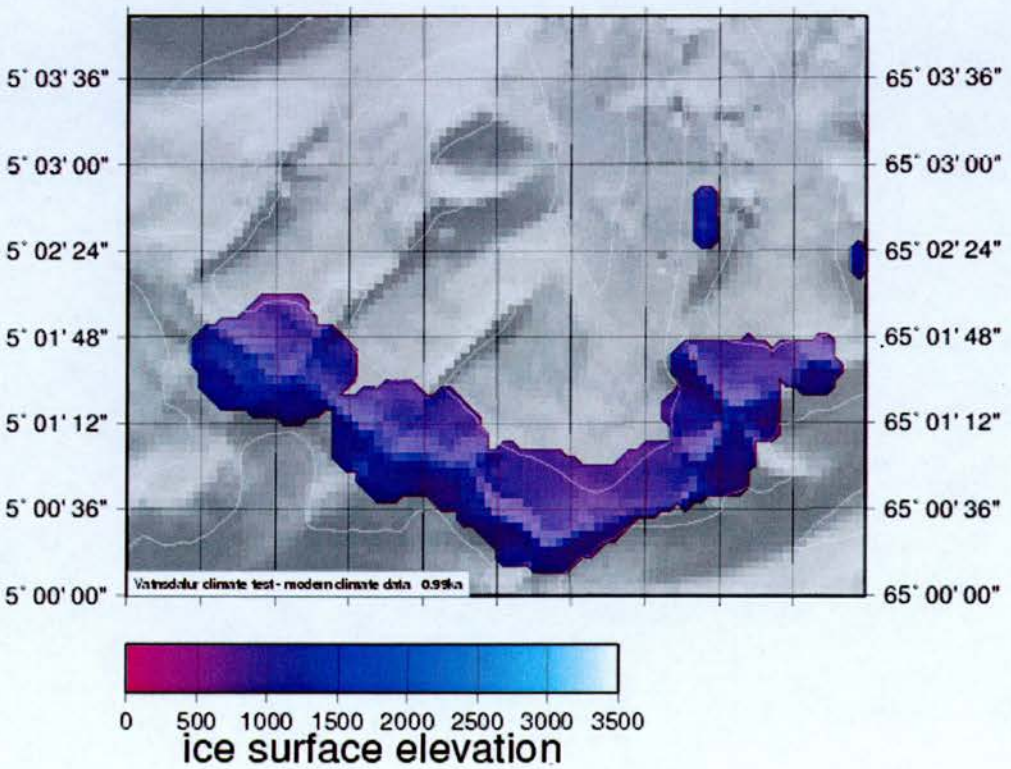

Figure 4.27: Glacier distribution in Vatnsdalur using modified precipitation and a temperature depression of 2.5°C from the Akureyri average.

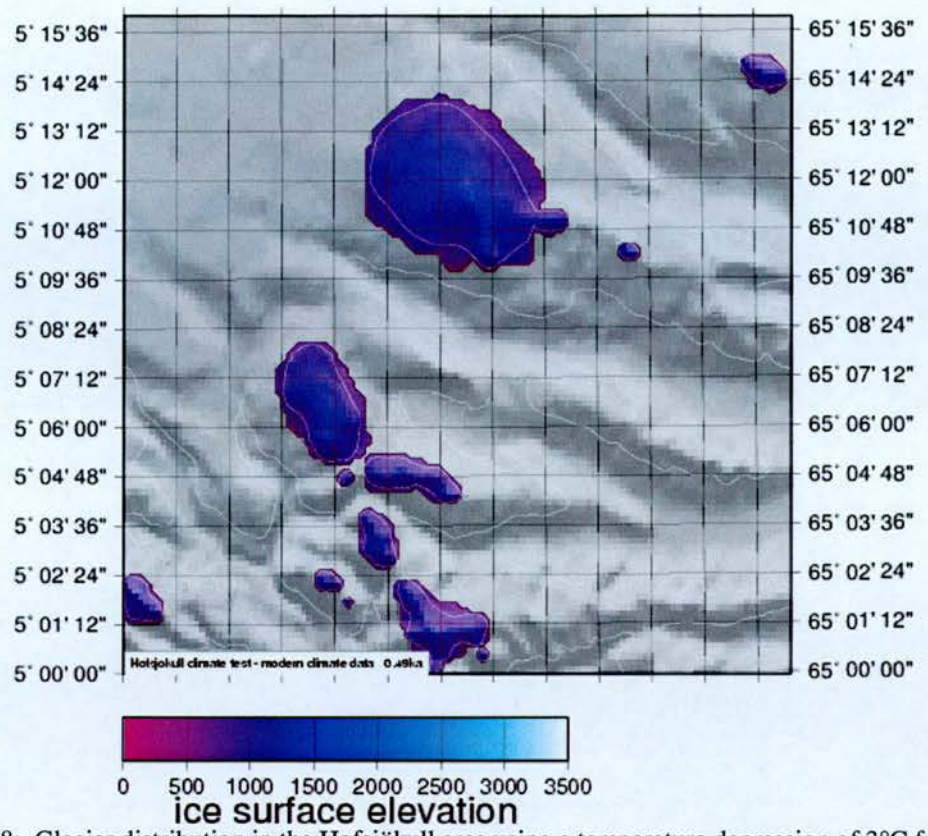

Figure 4.28: Glacier distribution in the Hofsjökull area using a temperature depression of 3°C from the Teigarhorn average.

Chapter V: Discussion

1 Introduction

2 Late Holocene climate

3 Southern Iceland – Climate and Settlement in Thórsmörk

4 Northern Iceland – Climate and Settlement around Mývatn

5 Eastern Iceland – Climate and Settlement in Geithellnadalur

V.1: Chapter Introduction

Chapters III and IV have described new empirical data and data from a new model of mass balance and vegetation in Iceland. This chapter combine these data with the existing archaeological and environmental story from the three study areas highlighted in Chapter II. The varying impacts of climate upon different regions of Iceland during the Medieval can now be discussed.

V.2: Late Holocene Climate

The overall aim of the thesis referred to the question of whether climate around the time of Norse settlement was "notably benign". Research objectives 4, 5 and 6 challenged the thesis to examine and determine the nature of Medieval climate.

The first examination of this point was the view in the literature. A general conclusion from the existing data sources was one of uncertainty surrounding the timing, magnitude and extent of the Medieval Warm Period, although I believe there is sufficient evidence that such an event existed in the 10^{th}-12^{th} Centuries, and was hemispherically significant. The conclusion was of a climate perhaps 1°C warmer than the 20^{th} Century average (though not perhaps warmer than present-day temperatures). One question surrounds the durability of this warmer climate. Documentary sources are almost invariably patchy, yet the impression is one of substantial variability even during the so-called warmer periods. How does this manifest itself in the glacier record?

In an area of marginal glaciation, persistence of any Medieval warmth can be examined. Consider the conceptual model in Figure 5.1. This model demonstrates the

difference between glaciers that are susceptible to melting (the 'variable' glaciers) and those that are not susceptible to melting (the 'steady state' glaciers). If the magnitude of temperature change is sufficiently large or the change is for a sufficiently long period, then the ice masses will disappear during the warm periods between colder episodes. Additionally, if there is increased snowfall at high elevation during warmer winters the ice masses can survive.

The record from Skeiðsvatn is one showing for the first time in Iceland the onset of Late Holocene-LIA glaciation. The onset occurs around 650A.D., prior to the 'Medieval Warm Period' and some 700 years prior to the beginning of the LIA. The record of glacier advance in Vatnsdalur dating through the second half of the Holocene indicates a sensitivity to climate not found in many other catchments, and this appears to be borne out by the loss of the Vatnsdalur glaciers. However, these glaciers do not disappear again. This constrains the magnitude and temporal extent of subsequent warming episodes: if the change is purely temperature-driven, then ablation is restricted by an amount related to glacier size. For example, a mass balance equivalent to 0.5m of annual ablation occurs in the Vatnsdalur area if the temperature is raised by 1°C on a piece of ice at the ELA. Therefore, 25m of ice will be removed after 50 years given an average of 1°C warming.

The record from Skeiðsvatn indicates that the threshold is not effectively crossed during the MWP. This may constrain the magnitude of warming to the order of 0.5 - 1°C, as larger warming would result in a large enough ablation rate to effectively remove the ice. The caveats to this are that debris cover may play a part during glacier retreat in this area. There is abundant evidence for rock glacier activity, and clear evidence for rock glaciers maintaining ice masses in regimes where they are not climatically sustainable. This may complicate the story, in that the formation of a rock glacier might allow the sustaining of ice masses in intervening warm periods beneath large debris covers. Similarly, precipitation increases could also have the same effect.

The record from Hofsvötn does not constrain Medieval climatic warming, as the record is limited to the time of the Little Ice Age. It does, however, show that a comparable record to that at Skeiðsvatn does not exist at this site. The limited extent of the record may be due to coarse material entering the lake from an ice position relatively close to the lake, or perhaps from avalanches bringing debris into the lake. There is no geomorphic evidence for a LIA ice margin close to or beyond the lakes,

which would require a particularly large extension of the ice, but the brevity of the core records indicates substantially enhanced geomorphic activity in the catchment ending in the late LIA. Consequently, although Hofsvötn was one of the very few sites identified as potentially suitable for this new approach in Iceland, it has proven not to contain a suitably long record.

V.3: Southern Iceland – Climate and Settlement in Thórsmörk

We can also therefore constrain the optimum for Thórsmörk to the same temperature increase, if we assume that temperature change is the climatic driver (as was done in Mackintosh, 2000; Casely, 2001). Model outputs for the upper part of the Thorsmörk valley are shown in Figures 5.3-5.7, with the relevant colour scale bars for all model outputs in Figure 5.2. Basic parameters for the area are in Figure 5.3, and show the topography of this model region, the extent of direct sunlight through the year, temperature, precipitation and snowfall. Figures 5.5 and 5.6 shows the Thórsmörk area under conditions of 0.5°C and 1°C warming. Clearly, the area around the Landnám settlements is climatically stable under these warming conditions, although as the maps show potential vegetation, sufficient time would be required to reach these levels of vegetation cover. The maps suggest that the analogies related to farming practice that the first settlers may have brought from their Scandinavian homelands could have been appropriate, given a substantial grass vegetation cover and nearby birch cover. The model indicates that the grass cover was good, and does not indicate marginal vegetation that is particularly vulnerable to climatic impacts. It is likely that although farming practices began with a Scandinavian style, there was a period of adaptation to the harsher and more fragile Icelandic environment (as shown in changes in animal bones from evidence of from Sveigakót at Mývatn).

If we consider the Little Ice Age in this area, it is possible to constrain the magnitude of cooling in the region. For the region shown in Figure 5.3, there is geomorphological and modelling evidence for the scale of LIA glacier advances. The geomorphological evidence is shown for advances of the glaciers Tungnakvíslajökull and Krossarjökull in Figure 5.8 (Casely, 2001), and the advances are reconstructed in Figure 5.9. Based on this evidence, an estimate of the magnitude of cooling was

made by Casely (2001). A flowline numerical model of Tungnakvíslajökull places the ice limit at the earlier and later LIA limits in step with temperature forcing according to Bergthórsson's (1969) sea ice temperatue curve, where the greatest cooling is *c.*1.5°C averaged over a period of 25 years.

Modelling of vegetation change in Thórsmörk in accordance with this scale of cooling is shown in Figure 5.6. Temperature is lowered in 0.5°C steps from 1.5°C above present-day climate to 2.5°C below present day. It shows two things. Even at a 1.5°C temperature lowering, areas ought to still be vegetated that are completely denuded today. It also shows that woodland survival in Thórsmörk would be expected given a 1.5°C lowering, albeit that woodland would be marginal at this point. The consequence of this is that the deforestation and denudation of this area is not an inevitable result of climatic change, although impacts of various types could be exacerbated by climate (c.f. Simpson *et al*, 2003). Winter snowcover (Figure 5.7) extends down to the low ground close to the farms, but is more restricted further down the Markarfljót. The present date of snowcover removal at the Thórsmörk farms is during April following extensive cover in Januray-March. Snow remains for approximately a further month if temperature is lowered 1°C, and so winter grazing is unlikely to have been a strong factor in erosion under present-day snow conditions, but may have been more significant in a warmer climate. Raising the temperature 1°C is enough to take the average snowline above the farms throughout winter, exposing vulnerable land to grazing.

The severity of the human impact in Thórsmörk argues for a powerful non-climatic driver, as grass cover is not threatened climatically. This may include a local sensitivity due to the presence of large volumes of tephra close to the Landnám surface. In profiles to the east of the farms, *c.*80% of the sediment within 50cm below the Landnám tephra is composed of up to five black Katla tephras (Casely, 2001). Hence the potentially inviting landscape may be 'booby trapped', in that extensive erosion, initiated from small vegetation breaches due to grazing, is exacerbated by the quantity of tephra. Climate modelling would therefore support the concept of woodland resource management in Thórsmörk as being a crucial factor in settlement change rather than a climatically determined abandonment (Dugmore *et al.*, in press). Even rangelands farther inland towards and beyond Einhyrningur are only substantially affected by the coldest temperatures of the LIA, although they show extensive erosion today (Figure 5.10).

There is further support for this story from the existing evidence of aeolian sediment accumulation collected by Dugmore (pers. comm.; Figure 5.11). Significant erosion at Thórsmörk occurs late in the Medieval period, and is followed by stabilisation of the landscape following farm abandonment. At Stakkholt, located south of Thórsmörk, there is disturbance in sediments from the period close to the time of abandonment, and stabilisation prior to increased SAR during the LIA. This LIA age SAR increase is comparable to that found in Geithellnadalur in timing, and may reflect the increased climatic stress at the time. Stakkholt is well placed to receive sediments from disturbance on the high ground towards Eyjafjallajökull, whereas Thórsmörk is on a small knoll at the apex of the Markarfljót and Krossá rivers, and does not have such a source of sediment from nearby higher ground (Figure 5.12).

V.4: Northern Iceland – Climate and Settlement around Mývatn

The story in northern Iceland is one of enduring woodland remnants, rangelands and settlement change. The landscape around Mývatn is upland, and progressively more marginal towards the interior (Figure 5.13). Firstly, the lingering presence of woodland in small pockets could have a climatic dimension. Low gradients mean that modest changes of tree line altitude result in particularly large spatial changes (Figures 5.15-5.20). Changes on the scale of the MWP and LIA have the consequences of a very large north/south shifts in vegetation boundaries, with survival of pockets in particularly sheltered or favoured locations (for example in the valleys north of Hofstaðir). Present day woodland south of Hrisheimar, consisting of stunted small trees that are barely surviving, appears to be at the limit of its distribution (Figure 5.17, lower left; farms are indicated in Figure 5.13).

There is the question relating to the marginality of settlement, and in particular settlements such as Sveigakót to the south of Mývatn. The first look at the marginality of Sveigakót comes from a look at the potential vegetation maps of the area (Figures 5.17 and 5.18). These show that, in comparison to Hofstaðir, the land around (and in particular south) of Sveigakót is distinctly marginal with a temperature of 1.5°C below present. The fragility of the higher ground in the area is highlighted by Simpson *et al.* (2003; Figure 5.21) at a local scale, but the modelling here shows the same pattern on a regional scale. The consequence for the rangelands is marked,

particularly in the area east and south of Sveigakót, while around Hofstaðir, marginal conditions do not approach so closely to the farm. There is therefore no 'firebreak' to prevent an eroding frontier approaching the site of Sveigakót.

There are some other factors to consider onto which the model can throw light. The first is snowcover (Figure 5.22). Winter grazing is restricted by snowcover, and so land that is more extensively covered or covered for a longer duration will have reduced winter grazing. The greatest impact is upon the land not covered by snow, yet which is most marginal. This is a time of year when winter fodder is at its lowest, and therefore there may be pressure to graze animals on rangelands as early as possible, or in as many areas as possible. Extremely marginal land (ie high up and farther inland) is of no use and the snowcover is irrelevant here. There is much less snowcover on the land west of the Kraká (the main river running south and west of Mývatn) and on the less marginal land towards Hofstaðir. The trouble lies with the cover on rangelands to the east of the Kraká – the late-lying snowcover inhibits growth and any land use (e.g. winter grazing) causes erosion on particularly vulnerable land. The impacts of this contrast are seen in Figures 5.23 and 5.24. SAR recorded in these profiles shows some of the degradation and complete erosion at Sveigakót associated with the time of farm occupation, but that the area towards Hrisheimar has been much less affected, with SAR increasing during the LIA. This later disturbance would connect with the time when snowcover may have been later lying and more extensive (Figure 5.22), and when vegetation was stressed by particularly cold sequences of years. In addition, non-climatic erosion triggered during the Medieval created a more fragile landscape for the impacts of the LIA to operate upon.

Growing season changes are the other relevant factor in this case. The length of the growing season (Figure 5.15 and 5.16) shows up the marginality of land farther from the coast. The change in the date of the end of the growing season, as shown by the contrast in Figure 5.16, is a crucial factor. The balance between livestock number and total fodder production makes this date critical for the success of the Norse pastoral lifestyle. The change in the end date is significant throughout the region, but once again, the land to the east of the Kraká, and close to Sveigakót is under the greatest pressure, as the date for the end of the growing season retreats into August (September dates are in white on the diagram).

The sum consequence for the Mývatn region is that the lands to the south of Hofstaðir are particularly vulnerable to large-scale environmental degradation brought on by climatic changes, in contrast to the story around Thórsmörk. This area is both upland and relatively flat lying, allowing the changes to roll across the landscape more extensively, and so threaten the rangelands of the Mývatn communities, as ther is no buffer between them and the denuded land farther inland. The large spatial changes involved may restrict recovery of vegetation, due to the physical removal of seed beds to locations distant from the recovering areas. The physical distance introduces a greater lag in the recovery time, inhibiting regeneration in short periods of more favourable climate. This is in contrast to an area such as Thórsmörk, where the seed bed will remain closer to denuded areas. Hofstaðir appears to have a sufficiently favourable location, in that it lies in more favourable land on all of the above counts, and so its survival is perhaps expected, despite the large-scale denudation of the Sveigakót area.

V.5: Eastern Iceland – Climate and Settlement in Geithellnadalur

The Geithellnadalur area has great deal more in common with the Thórsmörk region, than the Mývatn region in terms of its topography and maritime climate (Figure 5.25). Crucially it represents a third characteristic landscape type in Iceland, the deep valleys within relatively high, rolling montane plateaux. We might initially expect that the area would be climatically insensitive, if it is analogous to the Thórsmörk example. The first piece of evidence against this view comes in the sediment profiles from Geithellnadalur presented in Chapter III, which show a distinct and sharp peak in aeolian sediment accumulation in the $12^{th}/13^{th}$ Century. The question is about the origin of this peak in SAR.

The upland areas are presently marginal to grass cover. Under warmer conditions vegetation can extend over these upland slopes, adding considerable areas of upland grazing to the landholdings of farms in the valleys. With cooler, less favourable conditions, these uplands quickly become semi-vegetated, if not entirely barren (Figure 5.28). The valley bottom lands, in contrast are relatively unaffected, and remain broadly favourable to vegetation growth, even during periods of severe cooling (Figure 5.28). A similar pattern is shown in the growing season maps, which

show the plateaux as being susceptible to reduction of an already short season, while the bottom of the valleys keep a much longer season (Figure 5.27). In this context, the timing of the notable Medieval peak in SAR (and by definition land degradation) within the valleys seems unrelated to climate, as it is occurring in favourable areas. In this case however, it is most likely that the rapid climate-driven changes on the high ground remove substantial resources, and therefore intensify pressure on the surviving valley bottom areas of vegetation (an example is in Figure 5.20). In particular, the loss of summer grazing in the mountains would result in intensified impacts on the outfield areas of the valleys, and this is indeed where the impact is seen early on.

The much later changes that occurred during the peak of the LIA (17th-early 20th Century) are in general not associated with any greater impact within the valley systems, although there is an associated gradual rise in SAR. The suggestion here is that once the upland areas have been lost, the coping strategies are enacted and the sheltered environments of the valleys see comparatively little additional stress with LIA climate changes.

An additional issue in the deep valleys of eastern Iceland is the issue of sunlight hours (Figure 5.25). In Iceland the position close to, and just south of, the Arctic Circle means that the Sun follows a particularly wide variety of paths in the sky. These range from a very low arc for a few hours across the southern horizon in midwinter, to a nearly complete circuit of the sky, with greatest elevation of around 48°. In summer, direct sunlight reaches north-facing slopes during morning and evening, while in winter, very few locations receive any direct sunlight. The result is that maps of incident radiation (relating direct sunlight to the slope angle) show great variation as well as low levels in winter, spring and autumn months, while there is relatively less variation during the summer. This will have an effect on both snow and vegetation cover, but was not modelled in detail due to the lack of calibration data.

In this final case it would seem that climate can play an indirect role in landscape change and settlement viability through its effect on high altitude areas surrounding low valley floors.

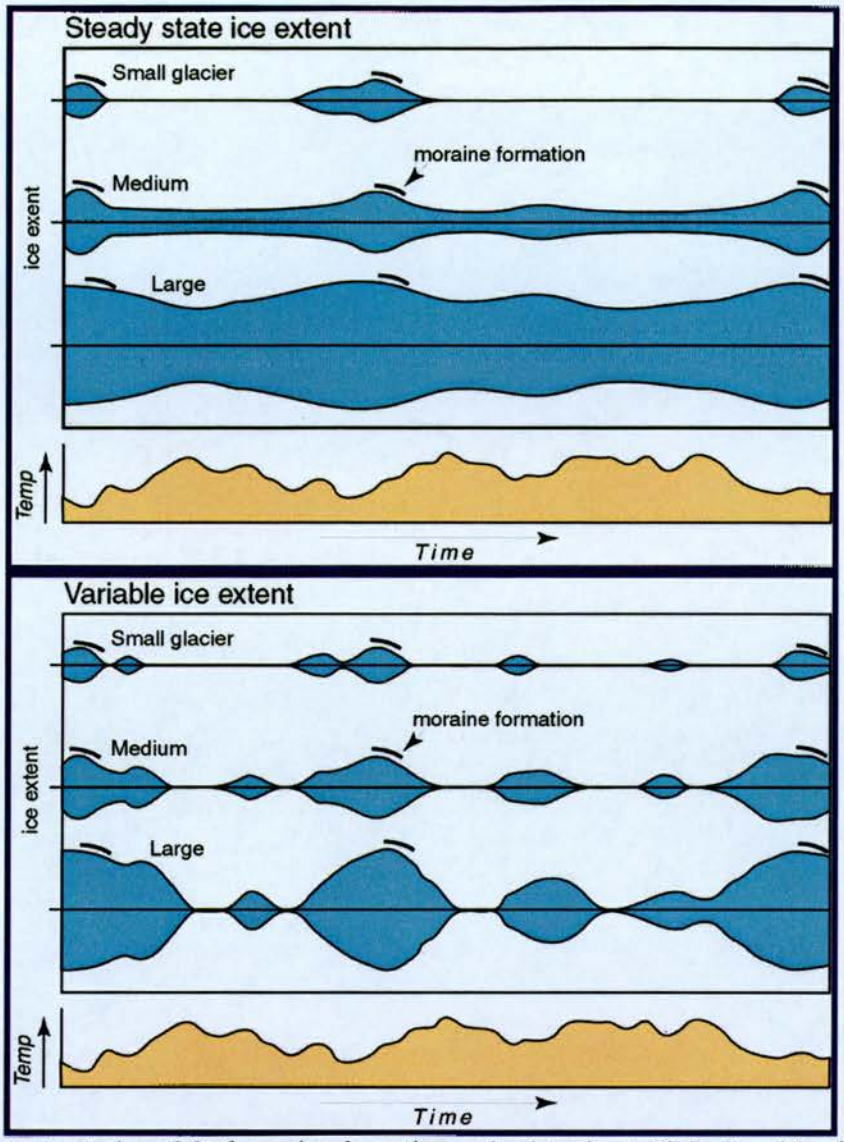

Figure 5.1: A conceptual model of moraine formation under 'steady state' (top) and 'variable' (bottom) glacier conditions. In this model, the pattern of temperature change is the same for both, and the record of moraines left behind will be the same.

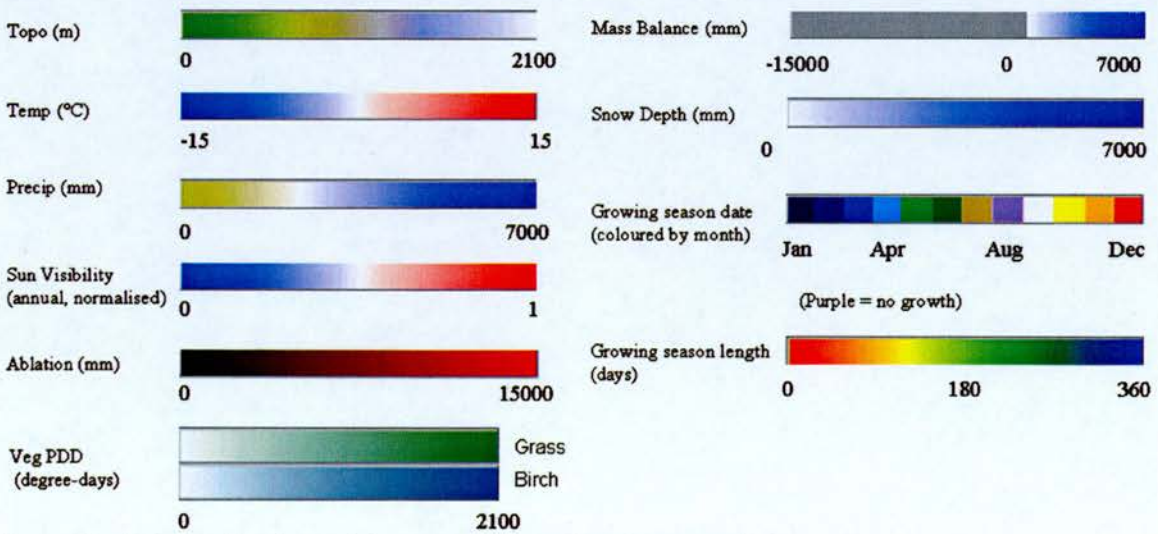

Figure 5.2: Scale barsfor the model outputs presented in subsequent figures.

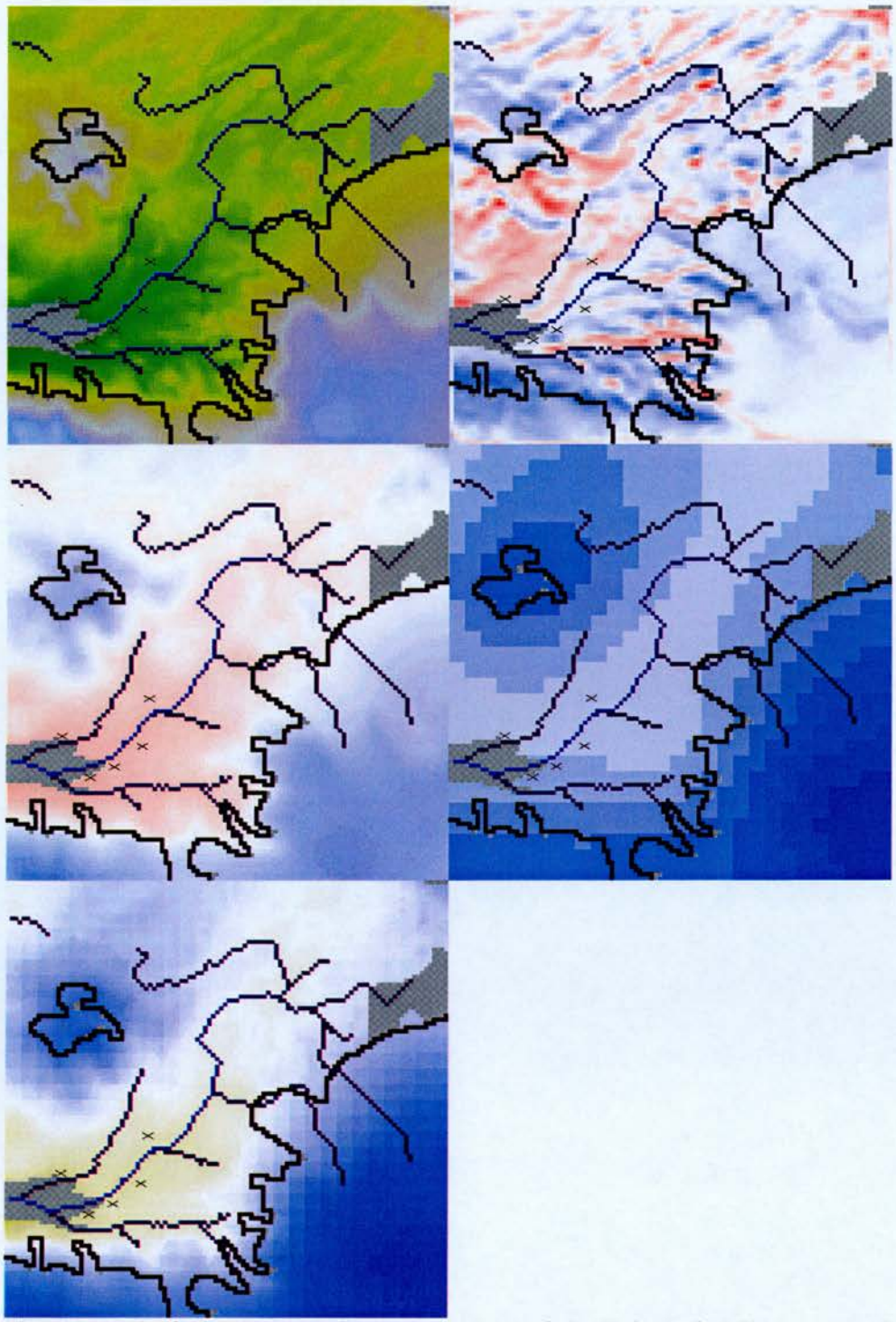

Figure 5.3: Physical parameters for the upper part of the Thórsmörk valley: topography (top left); sun visibility (top right); mean annual temperature (middle left); mean annual precipitation (middle right); Mean annual snowfall (bottom left). Diagrams are 28km x 28km, and present ice margins, rivers and sandar are in black, blue and grey respectively. Abandoned farms are marked by crosses.

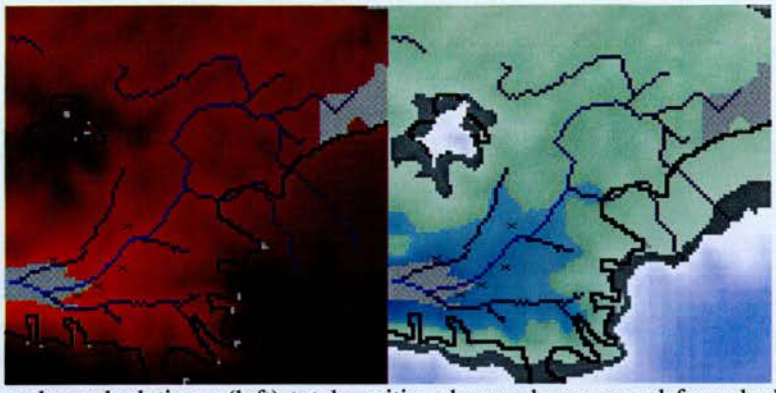

Figure 5.4: Degree-day calculations: (left) total positive degree-days as used for calculation of ablation; (right) a combined map showing the potential distribution of birch woodland (blue), grass (green) and areas of positive mass balance (white to pale blue).

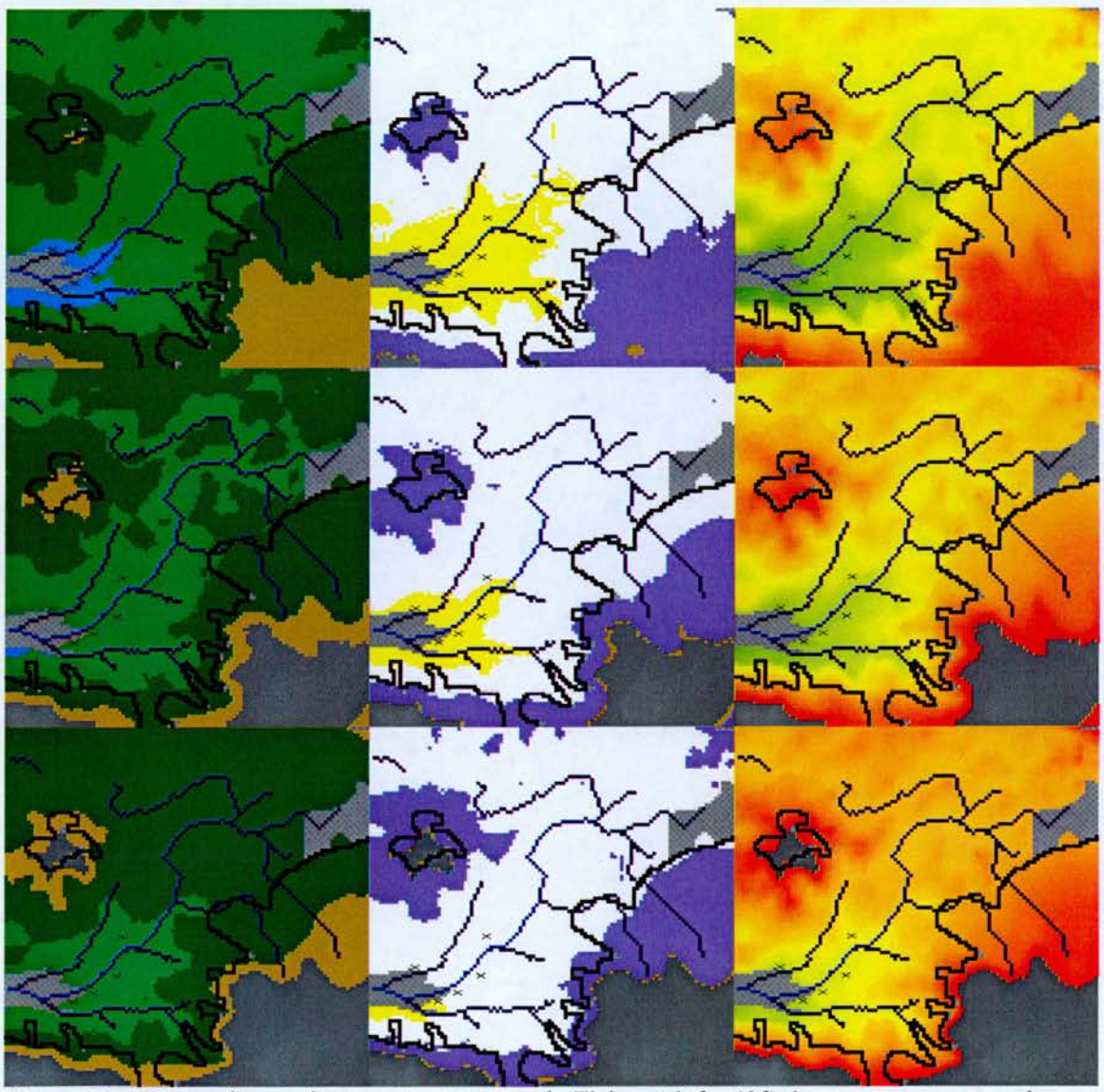

Figure 5.5: Changes in growing season parameters in Thórsmörk for 1°C above present; present day; and 1°C below present: (left) growing season start date, coloured by month, dark green is June; (centre) growing season end date, white is September; (right) growing season length, reduces from 160 days to 120 days in the area of the farms from top panel to bottom panel..

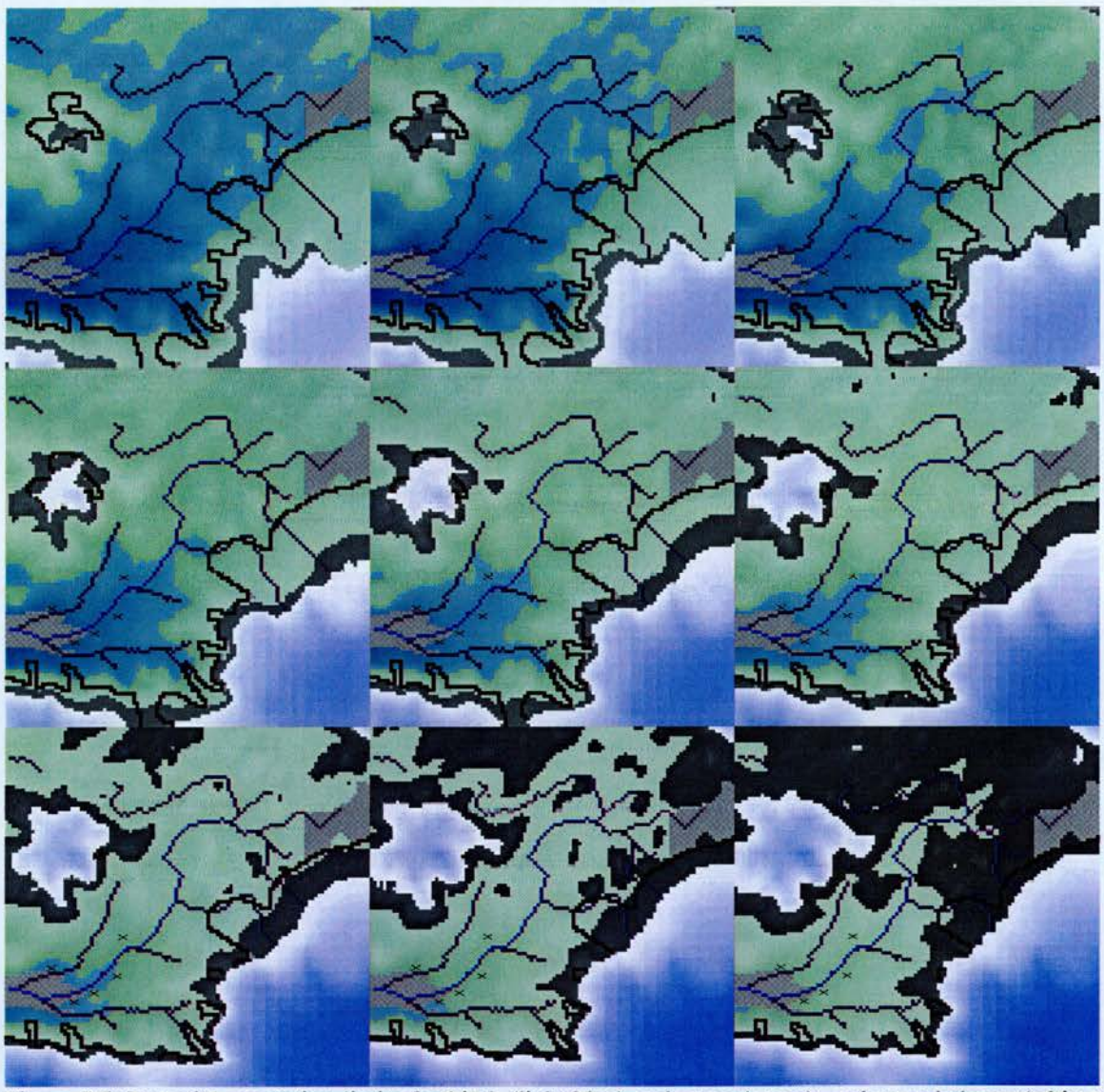

Figure 5.6: Potential vegetation limits for birch (light blue) and grass (green), and mass balance (white to pale blue) in Thórsmörk with temperature changes. Changes are stepwise in 0.5°C step from +1.5°C (top left) to -2.5°C (bottom right). Ice marginal positions are outlined in black.

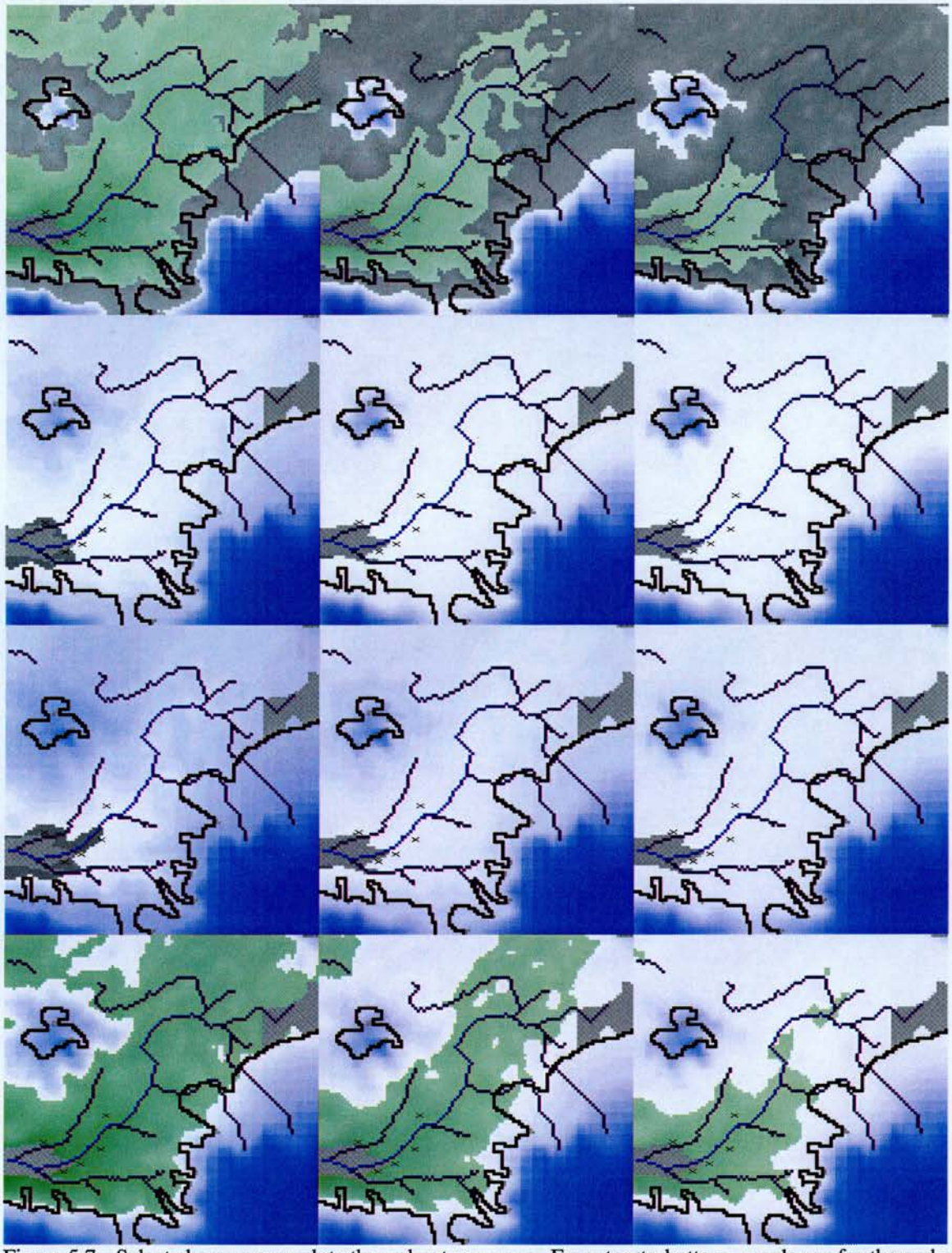

Figure 5.7: Selected snowcover plots throughout one year. From top to bottom, panels are for the end of September, December, March and June respectively. From left to right, panels are for temperatures of +1°C, 0°C and -1°C compared to the present. For each month, the area where the minimum degree day growth limit for grass is exceeded is shown (this can happen in areas outwith the total annual potential vegetation limit). Sandar areas show no vegetation or snowcover.

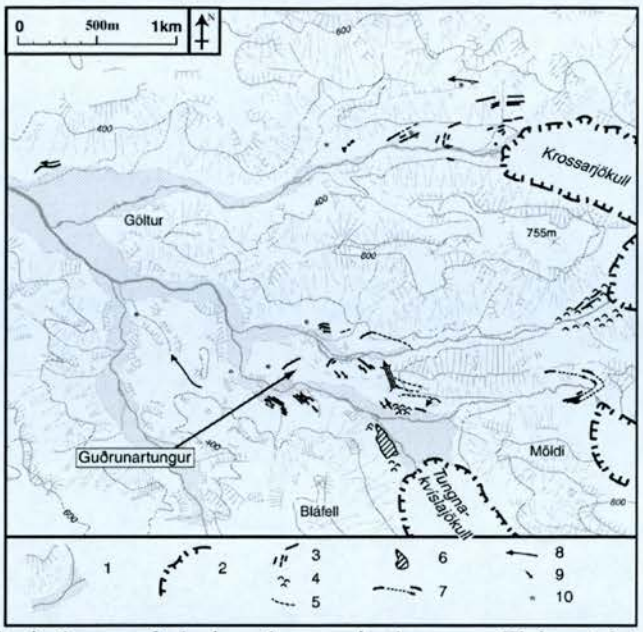

Figure 5.8: Geomorphological map of glacier advances in the upper Thórsmörk area (centre right part of the panels in Figure 5.5 (Casely and Dugmore, 2001).

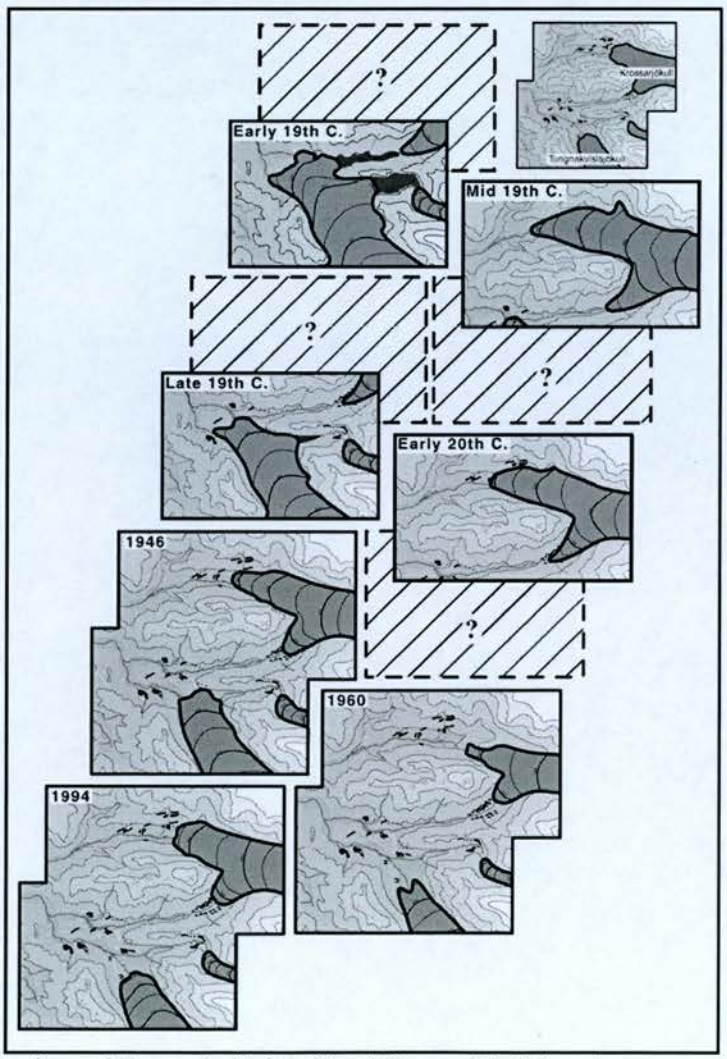

Figure 5.9: Reconstructions of Tungnakvíslajökull and Krossarjökull based on geomorphological evidence (Casely and Dugmore, 2001). Ice advance from 1994 positions is *c.*1km.

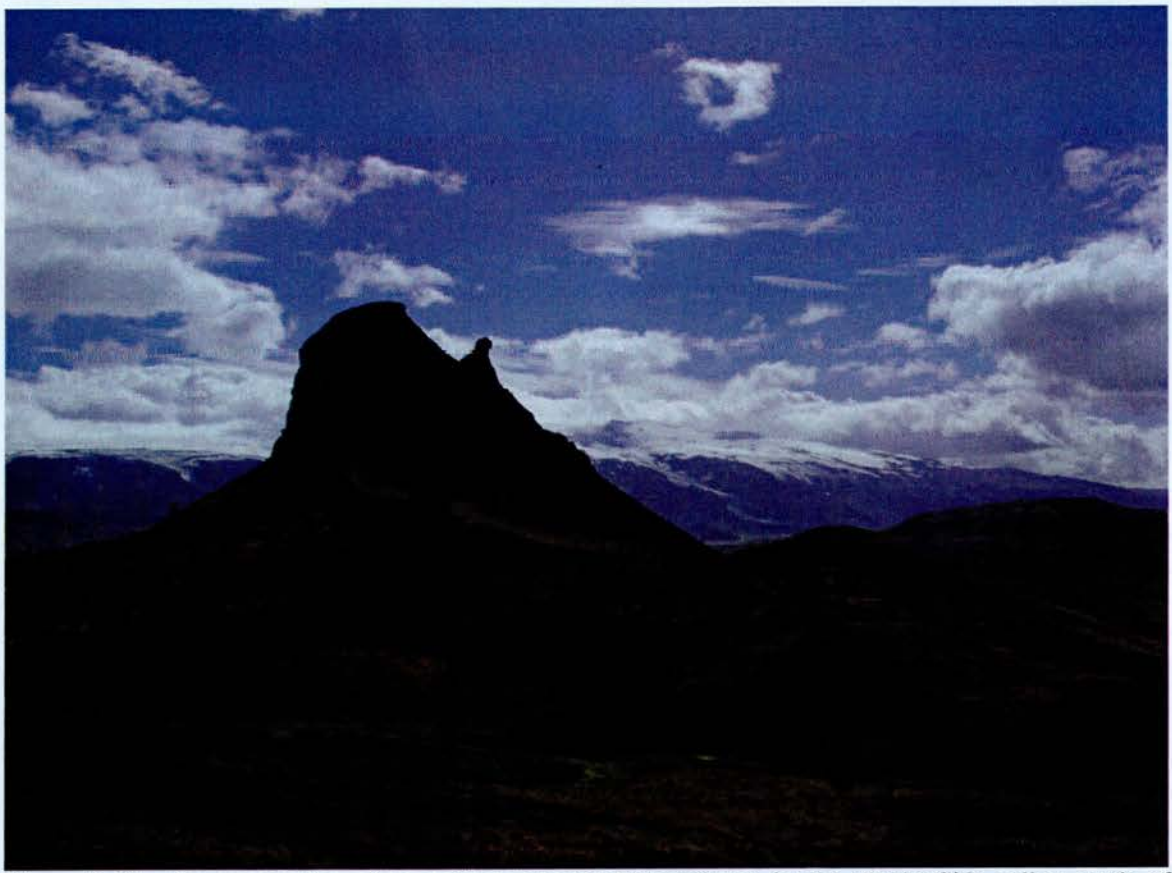

Figure 5.10: Severe erosion of rangelands around Einhyrningur in the Markarfljót valley north of Thórsmörk.

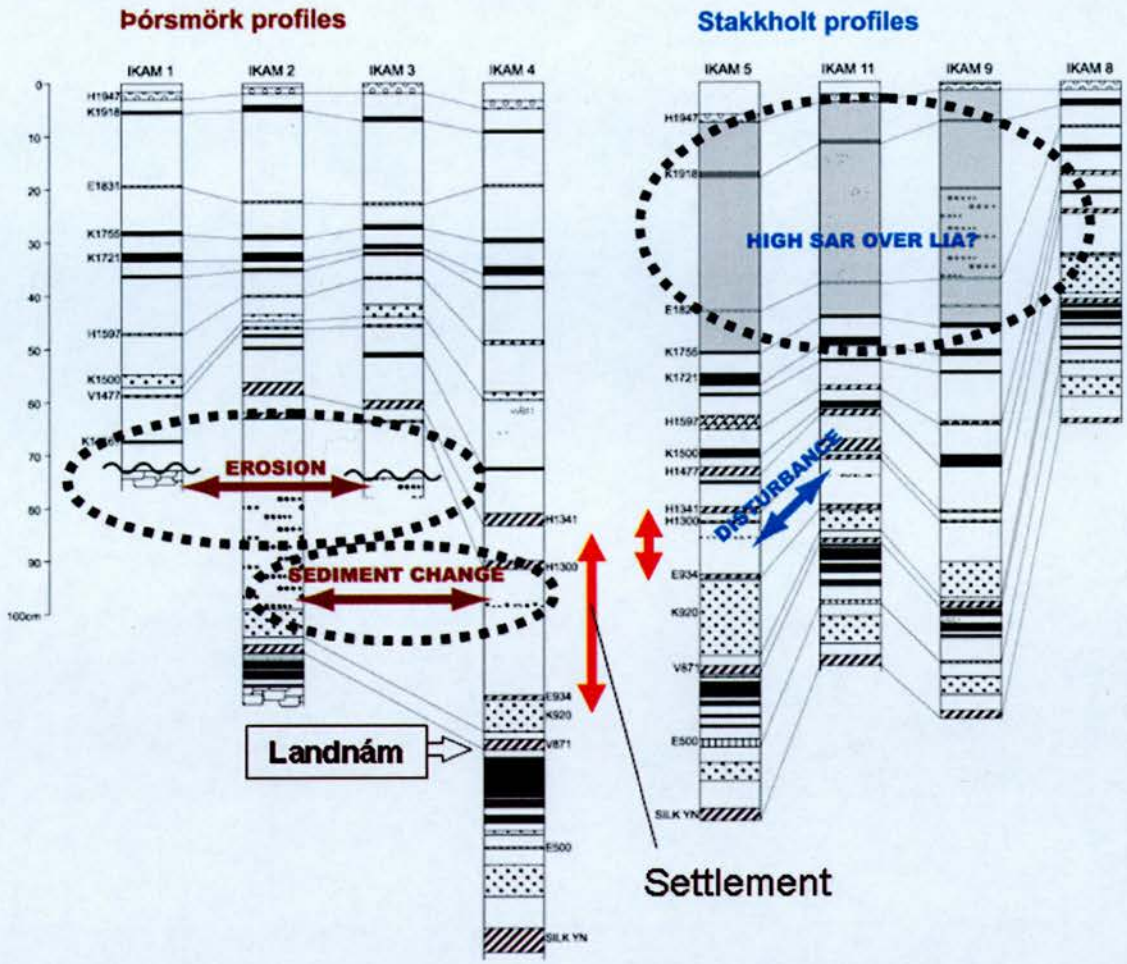

Figure 5.11: Aeolian sediment profiles from Thórsmörk and Stakkholt (Dugnore, pers. comm.). Stakkholt is located to the south of Thórsmörk on the other side of the valley.

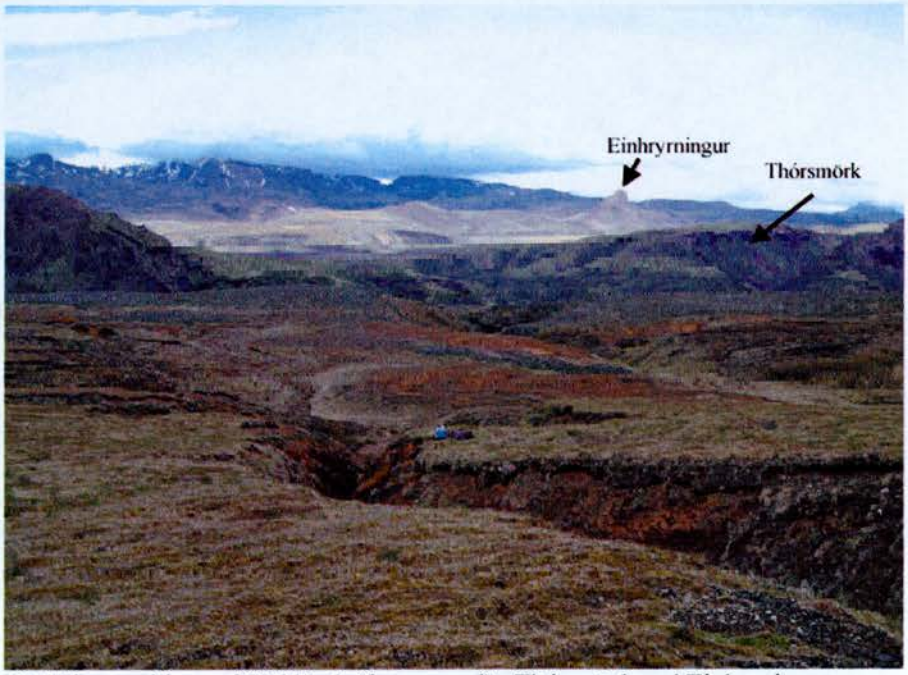

Figure 5.12: Relative positions of Stakkholt (foreground), Thórsmörk and Einhyrningur.

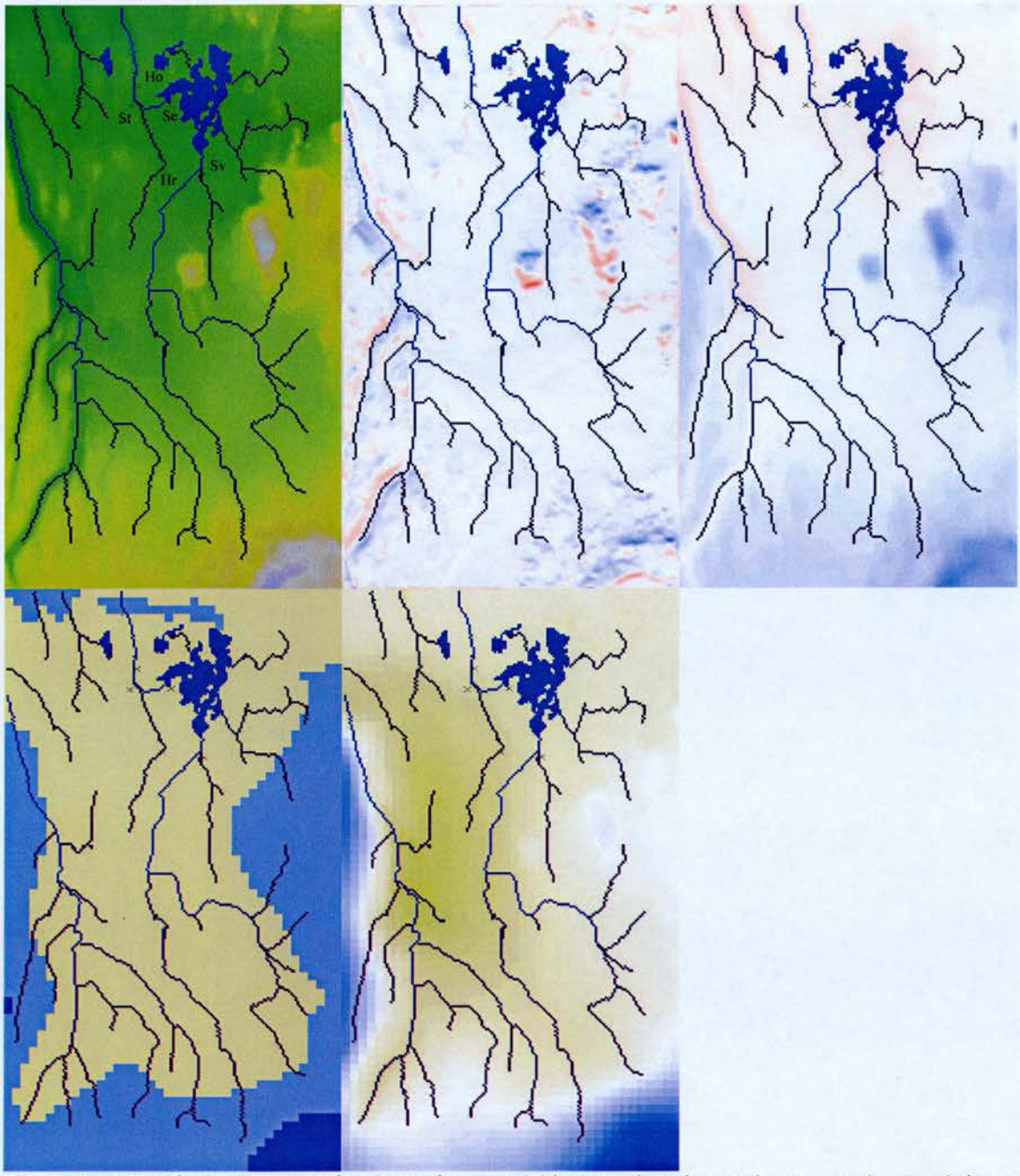

Figure 5.13: Physical parameters for the region near Mývatn and to the south: topography (top left); sun visibility (top middle); mean annual temperature (top right); mean annual precipitation (bottom left); mean annual snowfall (bottom right). Diagrams are 40km x 70km, and present ice margins, rivers and sandar are in black, blue and grey respectively. Abandoned farms are marked by crosses, and labelled: Hofstaðir (Ho), Selhagi (Se), Steinbogi (St), Hrisheimar (Hr), and Sveigakót (Sv).

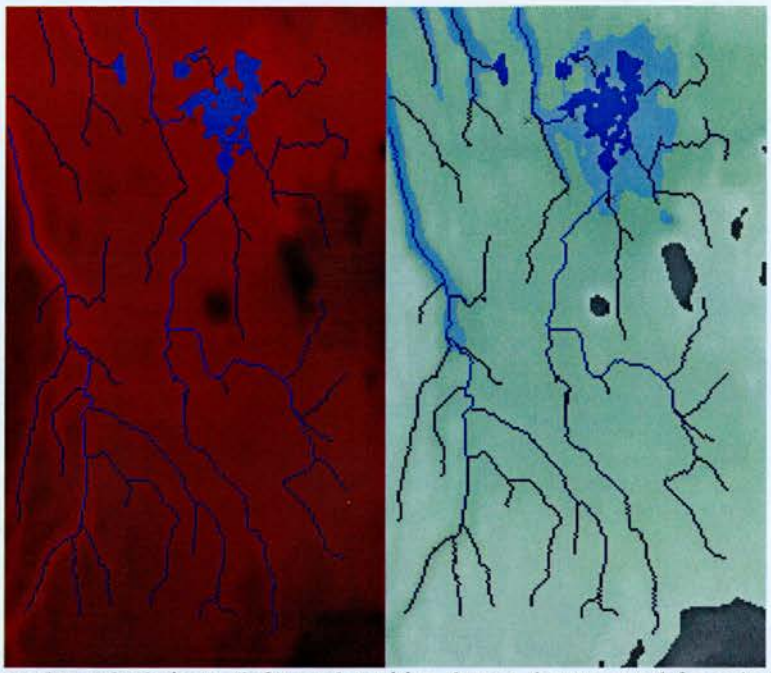

Figure 5.14: Degree-day calculations: (left) total positive degree-days as used for calculation of ablation; (right) a combined map showing the potential distribution of birch woodland (blue), grass (green) and areas of positive mass balance (white to pale blue).

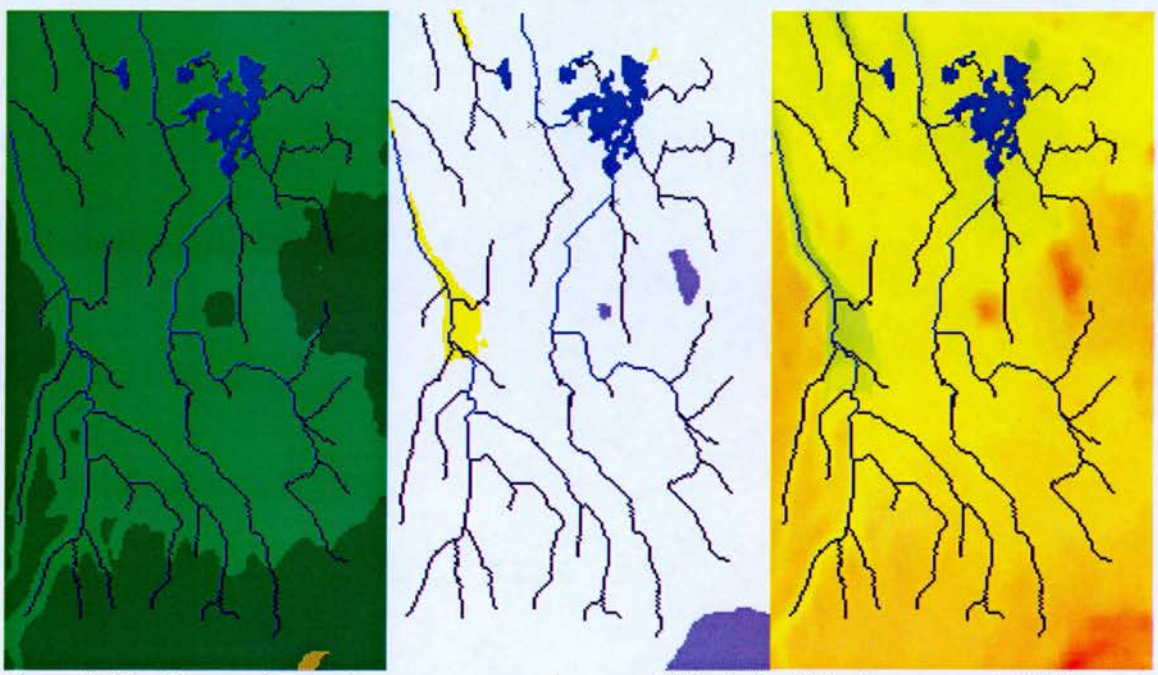

Figure 5.15: Changes in growing season parameters near Mývatn for 1°C above present: (left) growing season start date, coloured by month, dark green is June; (centre) growing season end date, white is September; (right) growing season length, reduces from 160 days to 120 days in the area of the farms from top panel to bottom panel..

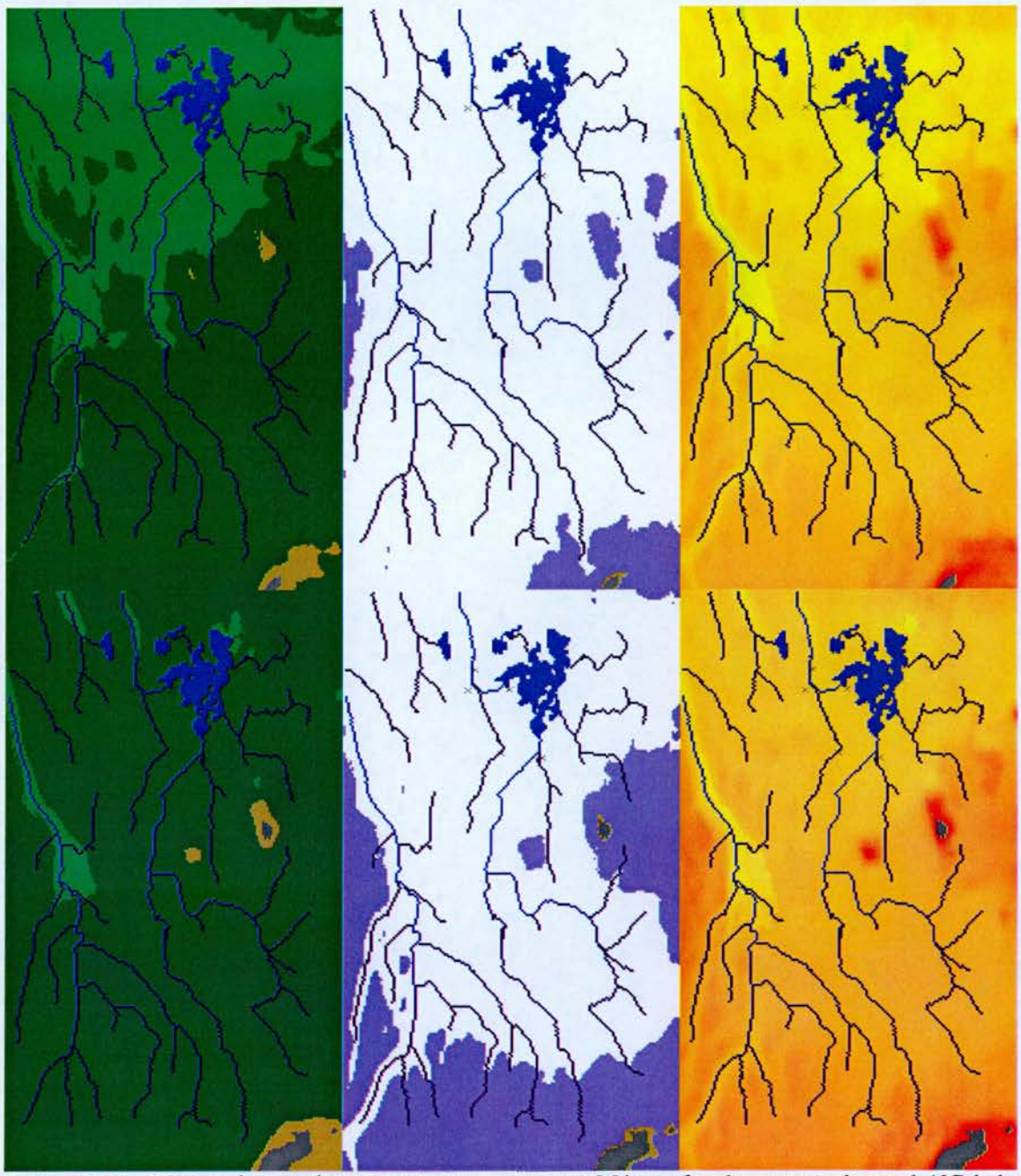

Figure 5.16: Changes in growing season parameters near Mývatn for the present day and 1°C below present: (left) growing season start date, coloured by month, dark green is June; (centre) growing season end date, white is September; (right). Growing season lengths decrease as you head inland across the rangelands towards Vatnajökull, but are reasonably high (>3 months) close to Mývatn, in river valleys and towards the north coast.

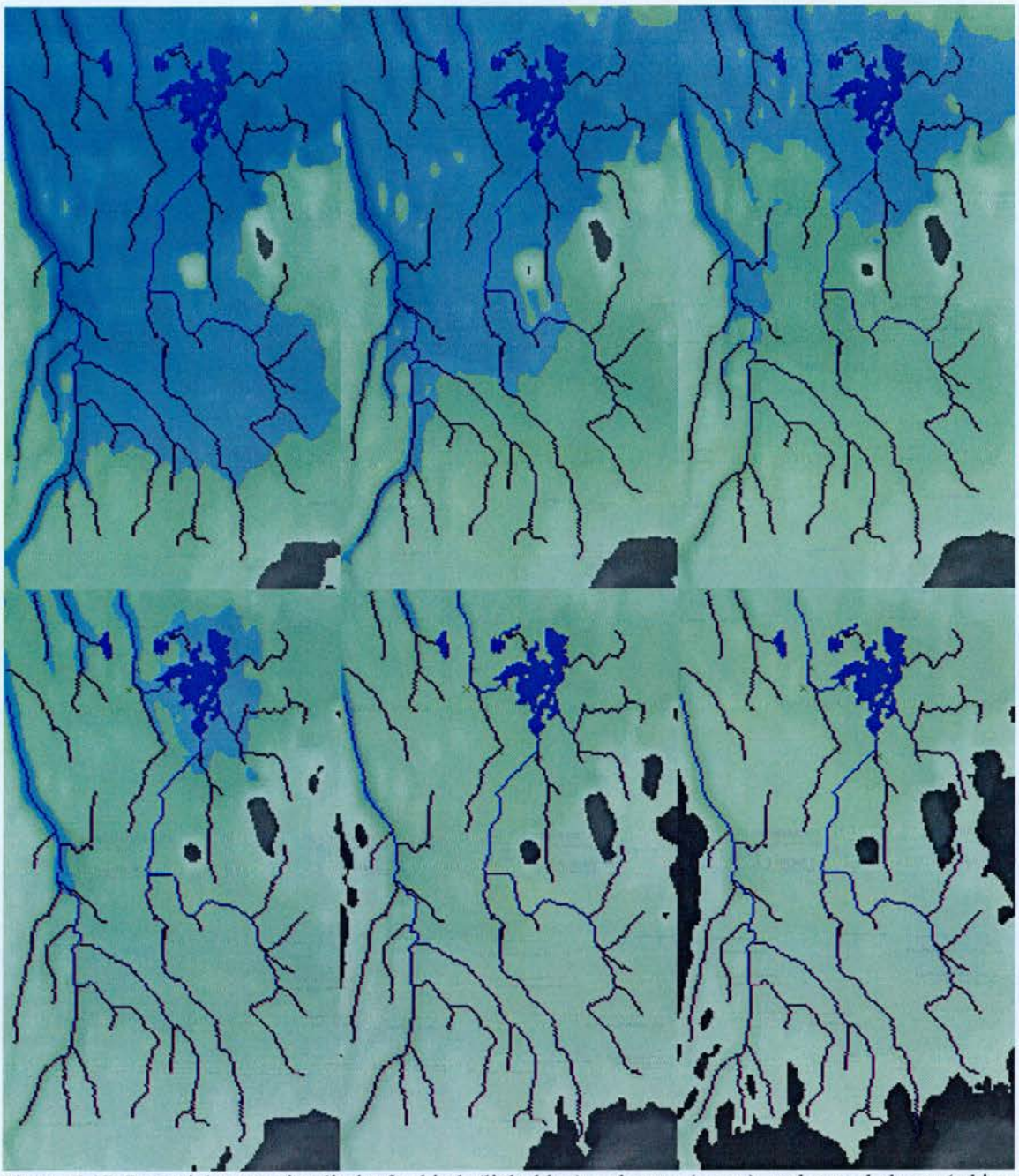

Figure 5.17: Potential vegetation limits for birch (light blue) and grass (green), and mass balance (white to pale blue) in Thórsmörk with temperature changes. Changes are stepwise in 0.5°C step from +1.5°C (top left) to -1°C (bottom right).

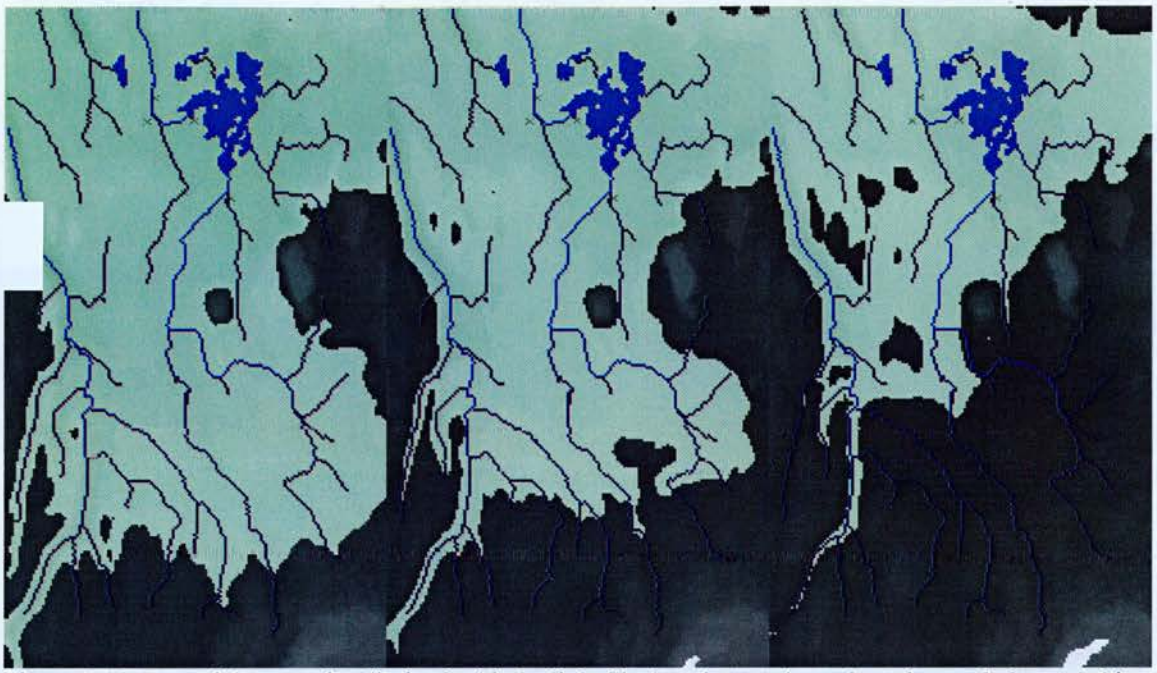

Figure 5.18: Potential vegetation limits for birch (light blue) and grass (green), and mass balance (white to pale blue) in Thórsmörk with temperature changes. Changes are stepwise in 0.5°C step from -1.5°C (left) to -2.5°C (right).

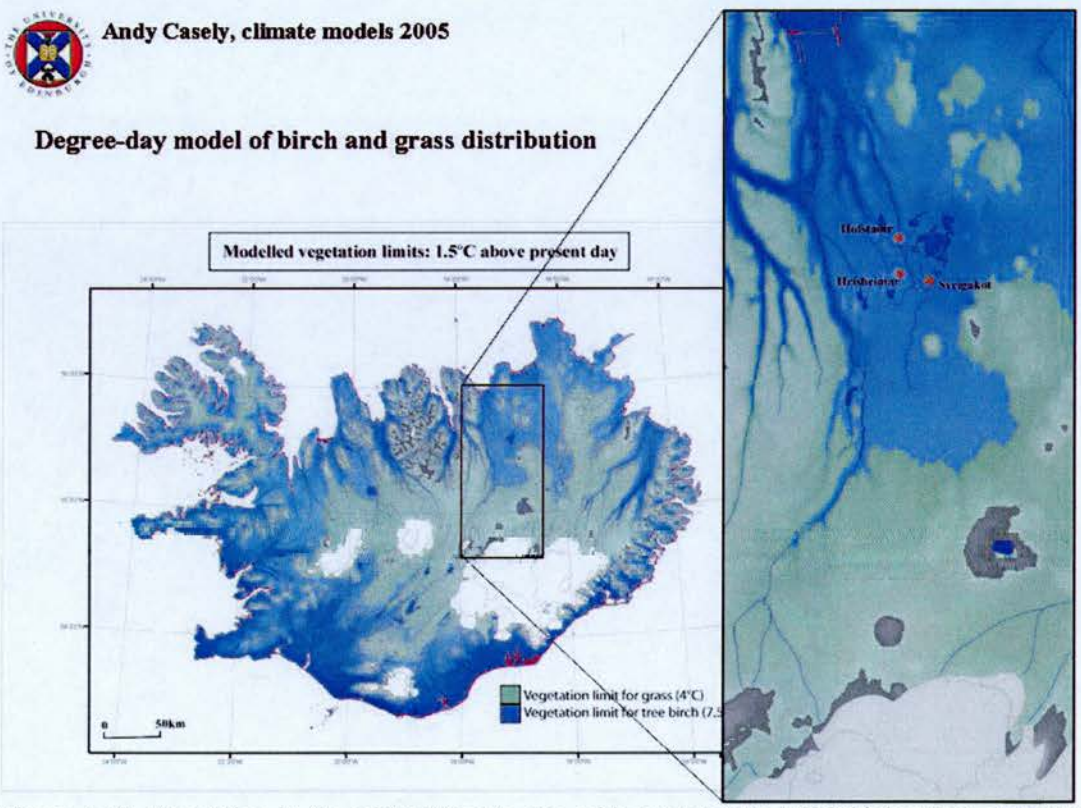

Figure 5.19: Vegetation in the wider Mývatn area with temperature 1.5°C higher than at present. Key archaeological sites are marked on the map.

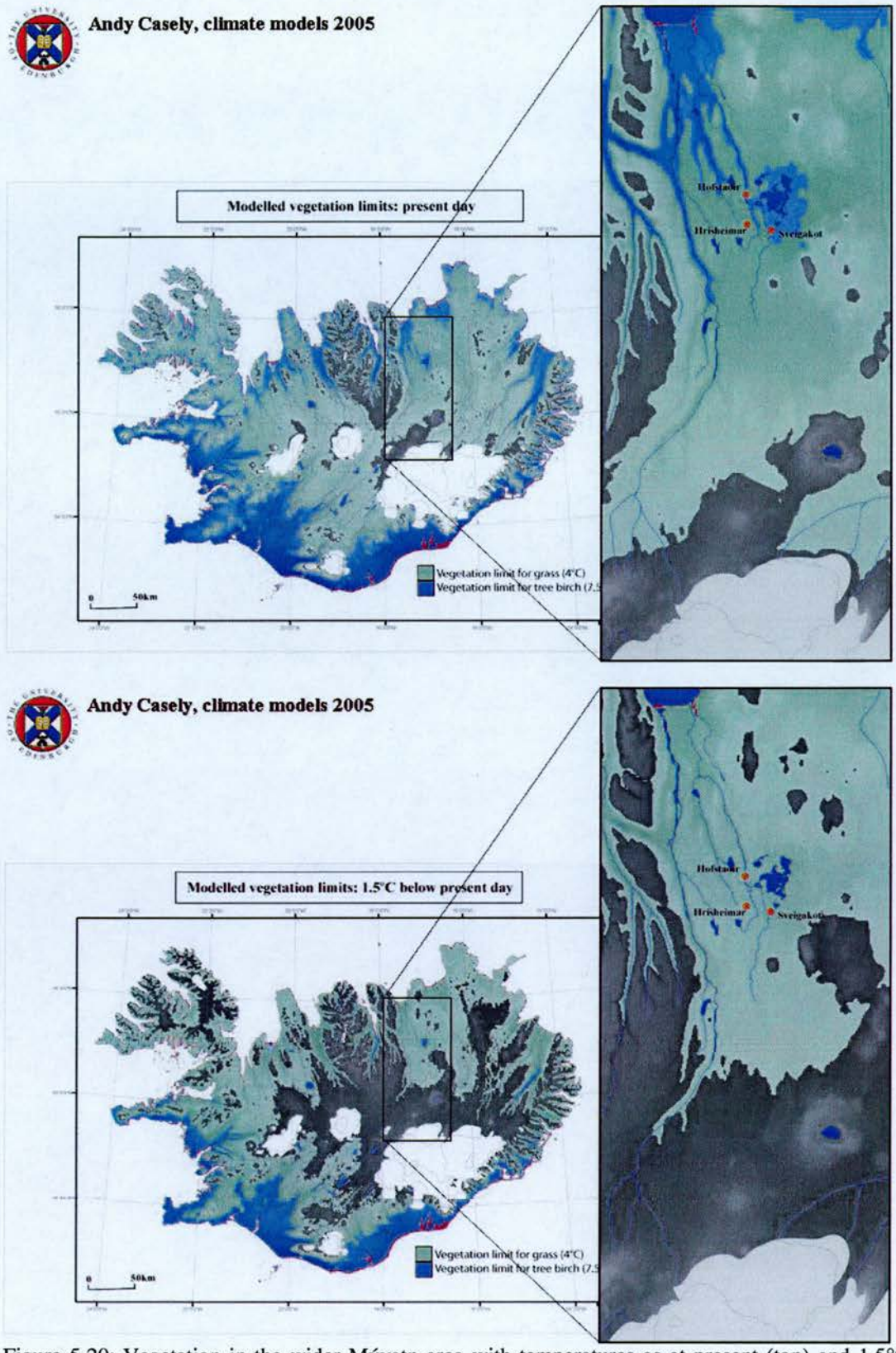

Figure 5.20: Vegetation in the wider Mývatn area with temperatures as at present (top) and 1.5°C lower than at present (bottom).

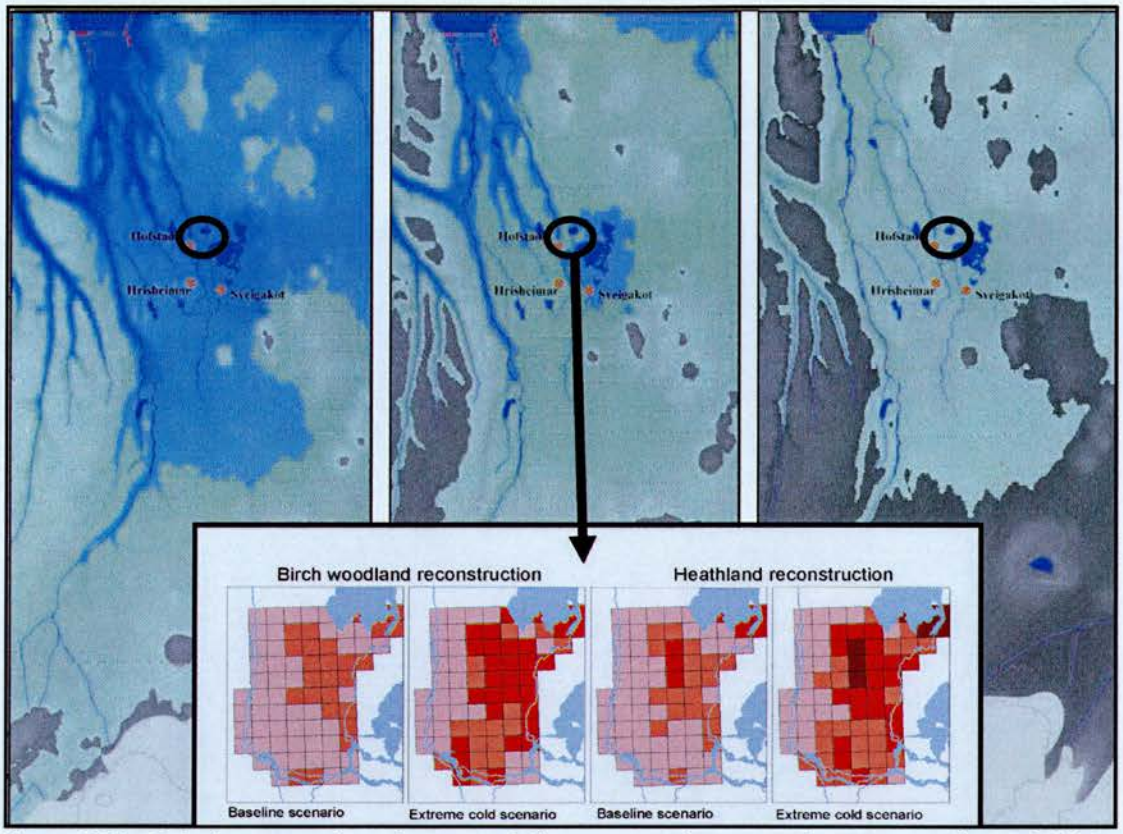

Figure 5.21: Relating vegetation change at Mývatn to settlement-scale reconstructions of birch and heathland by Simpson *et al* (2003). The panels are 1.5°C above, 0°C and 1.5°C below present day. Darker shades of red indicate enhanced vulnerability, and relate well to the temperature scenarios for present day and 1.5°C below present.

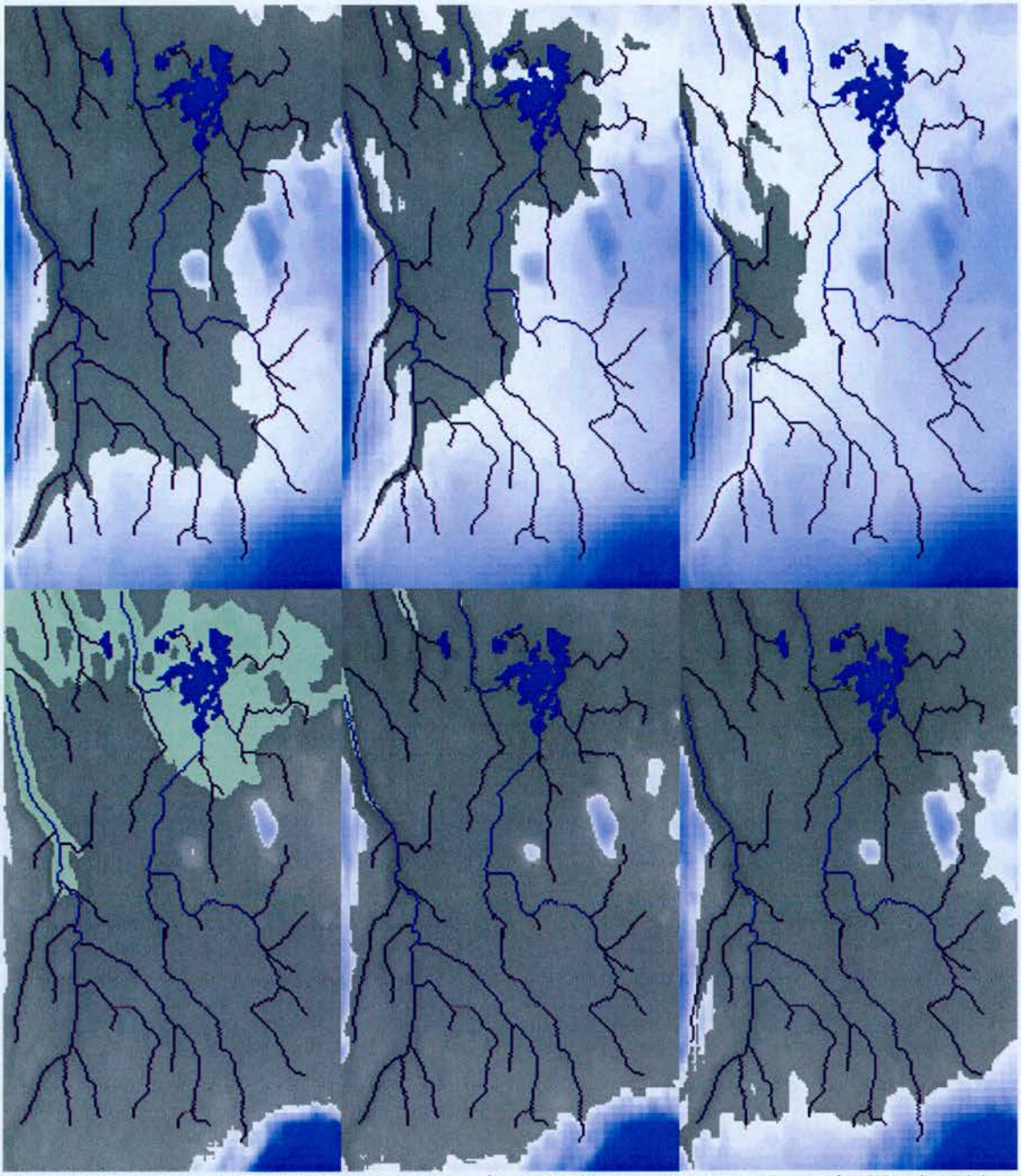

Figure 5.22: Selected snowcover plots for late winter at Mývatn. Upper panels are for March and lower panels are for May respectively. From left to right, panels are for temperatures of +1°C, 0°C and -1°C compared to the present. For each month, the area where the minimum degree day growth limit for grass is exceeded is shown (this can happen in areas outwith the total annual potential vegetation limit).

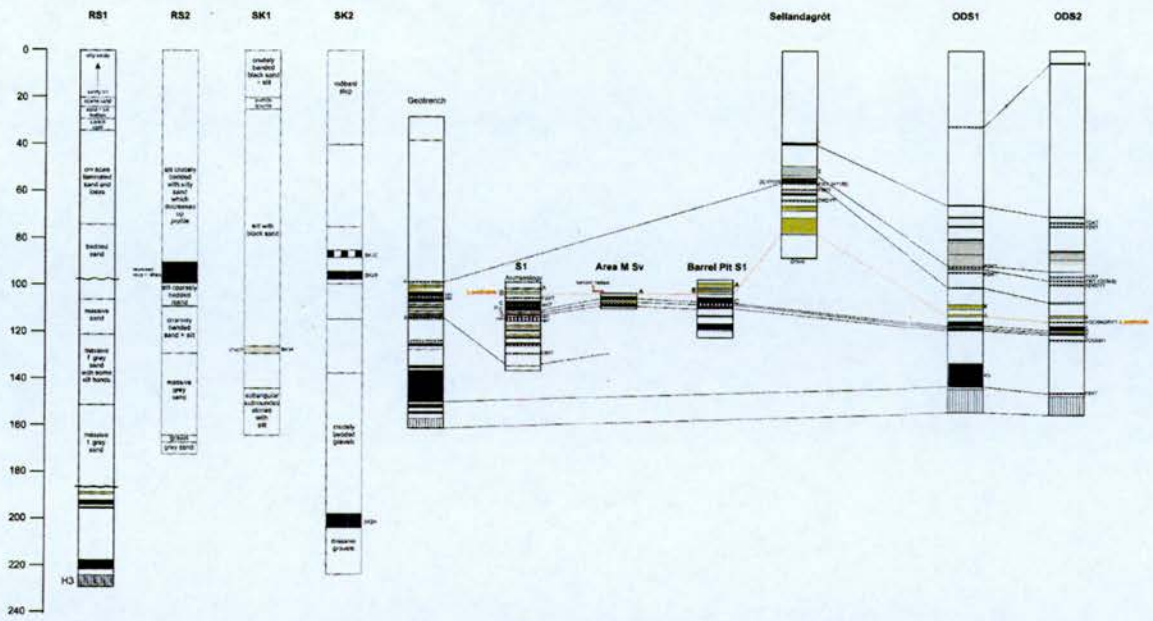

Figure 5.23: Aeolian sediment profiles from around Sveigakót. The profiles highlight the massiver degradation that took place following settlement (often complete soil removal) and high SAR following abandonment (Dugmore and Newton, pers. comm..).

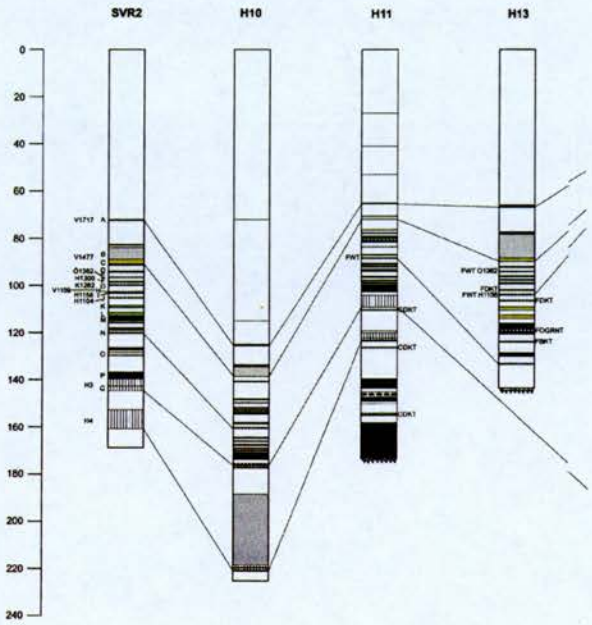

Figure 5.24: Aeolian sediment profiles from the western side of the Kraká and near Hrisheimar. These show that, in contrast to the Sveigakót area, there is much less disturbance around settlement, and that large SAR increases occur mostly after 1477 (Dugmore and Newton, pers. comm..).

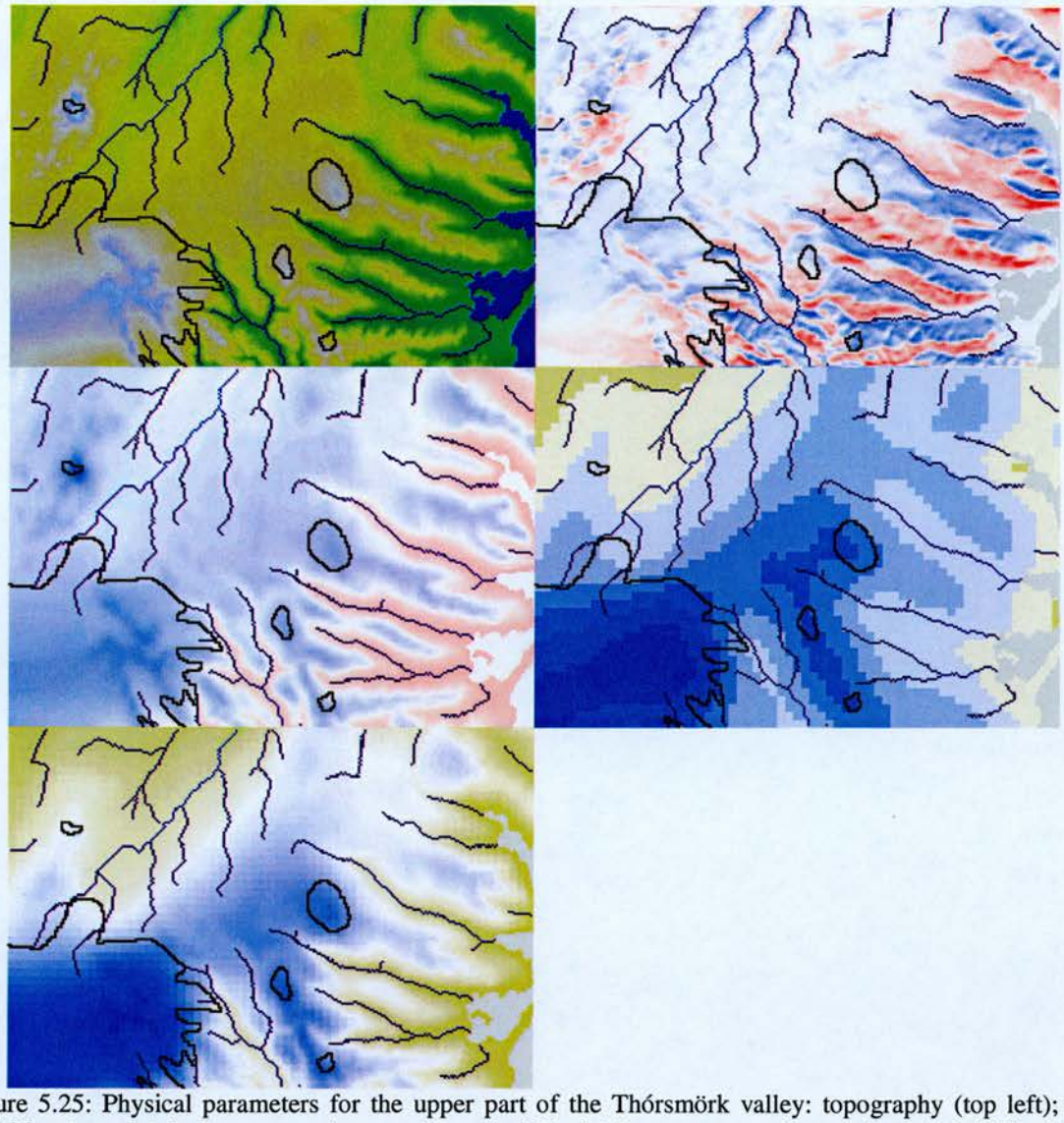

Figure 5.25: Physical parameters for the upper part of the Thórsmörk valley: topography (top left); sun visibility (top right); mean annual temperature (middle left); mean annual precipitation (middle right); Mean annual snowfall (bottom left). Diagrams are 65km x 45km, and present ice margins, rivers and sandar are in black, blue and grey respectively.

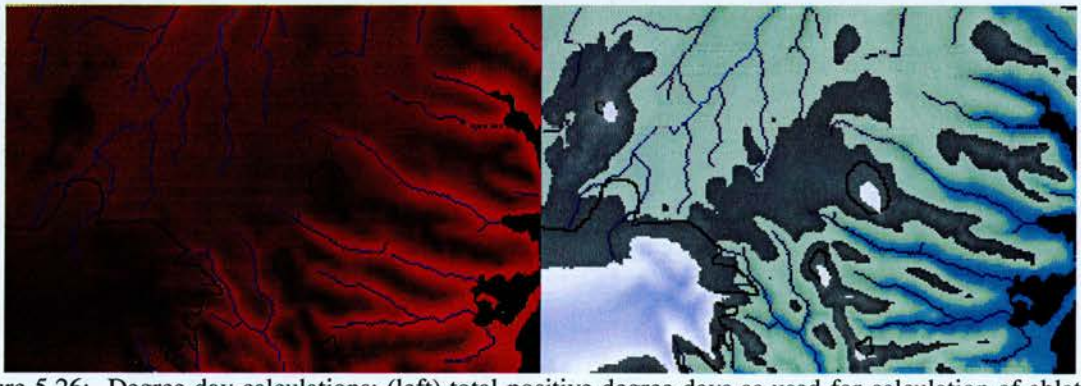

Figure 5.26: Degree-day calculations: (left) total positive degree-days as used for calculation of ablation; (right) a combined map showing the potential distribution of birch woodland (blue), grass (green) and areas of positive mass balance (white to pale blue).

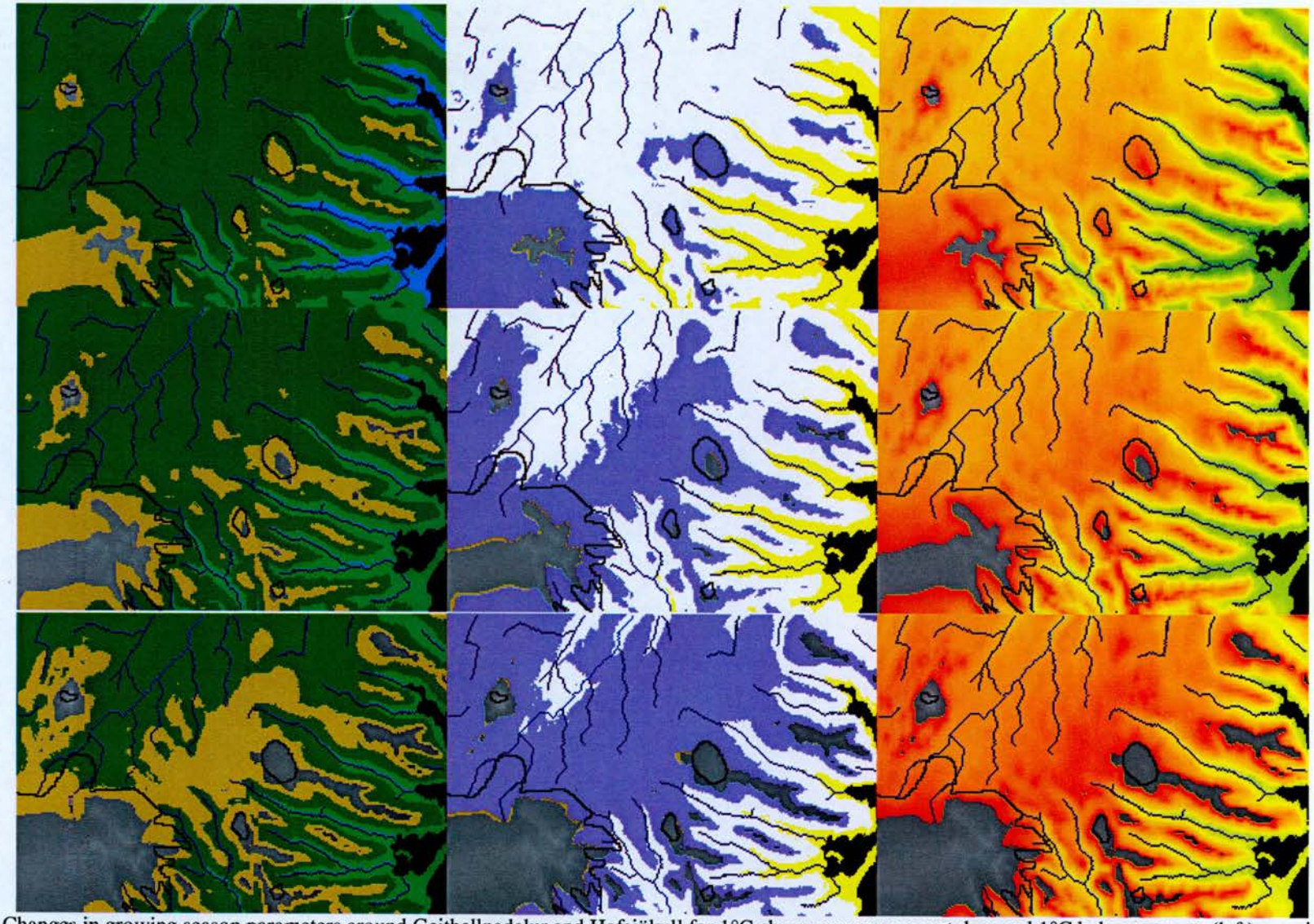

Figure 5.27: Changes in growing season parameters around Geithellnadalur and Hofsjökull for 1°C above present; present day; and 1°C below present: (left) growing season start date, coloured by month, dark green is June; (centre) growing season end date, white is September; (right) growing season length, reduces from 160 days to 120 days in the area of the farms from top panel to bottom panel. The top row is 1°C above, and the bottom row is 1°C below.

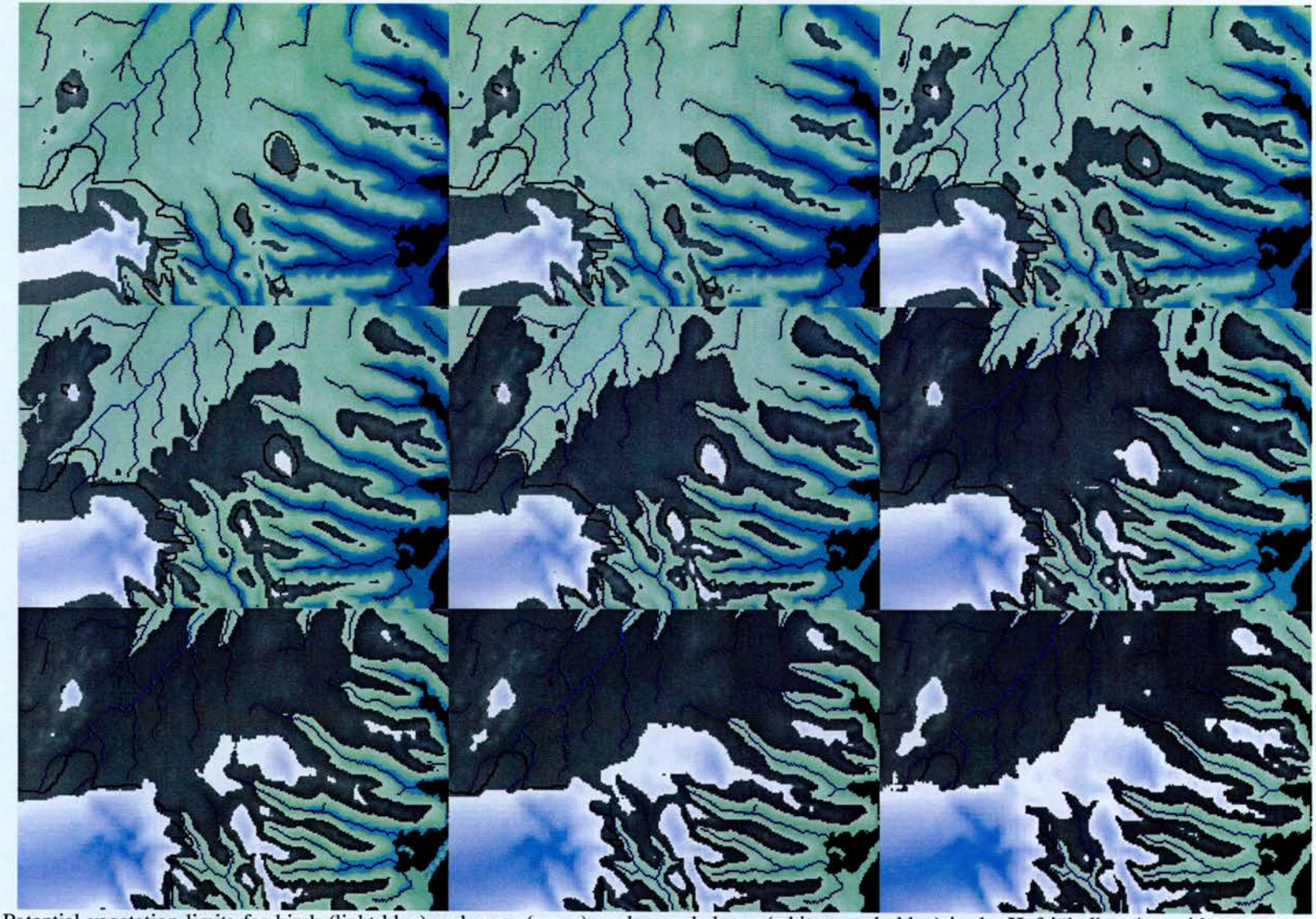

Figure 5.28: Potential vegetation limits for birch (light blue) and grass (green), and mass balance (white to pale blue) in the Hofsjökull region with temperature changes. Changes are stepwise in 0.5°C step from +1.5°C (top left) to -2.5°C (bottom right). Ice marginal positions are outlined in black.

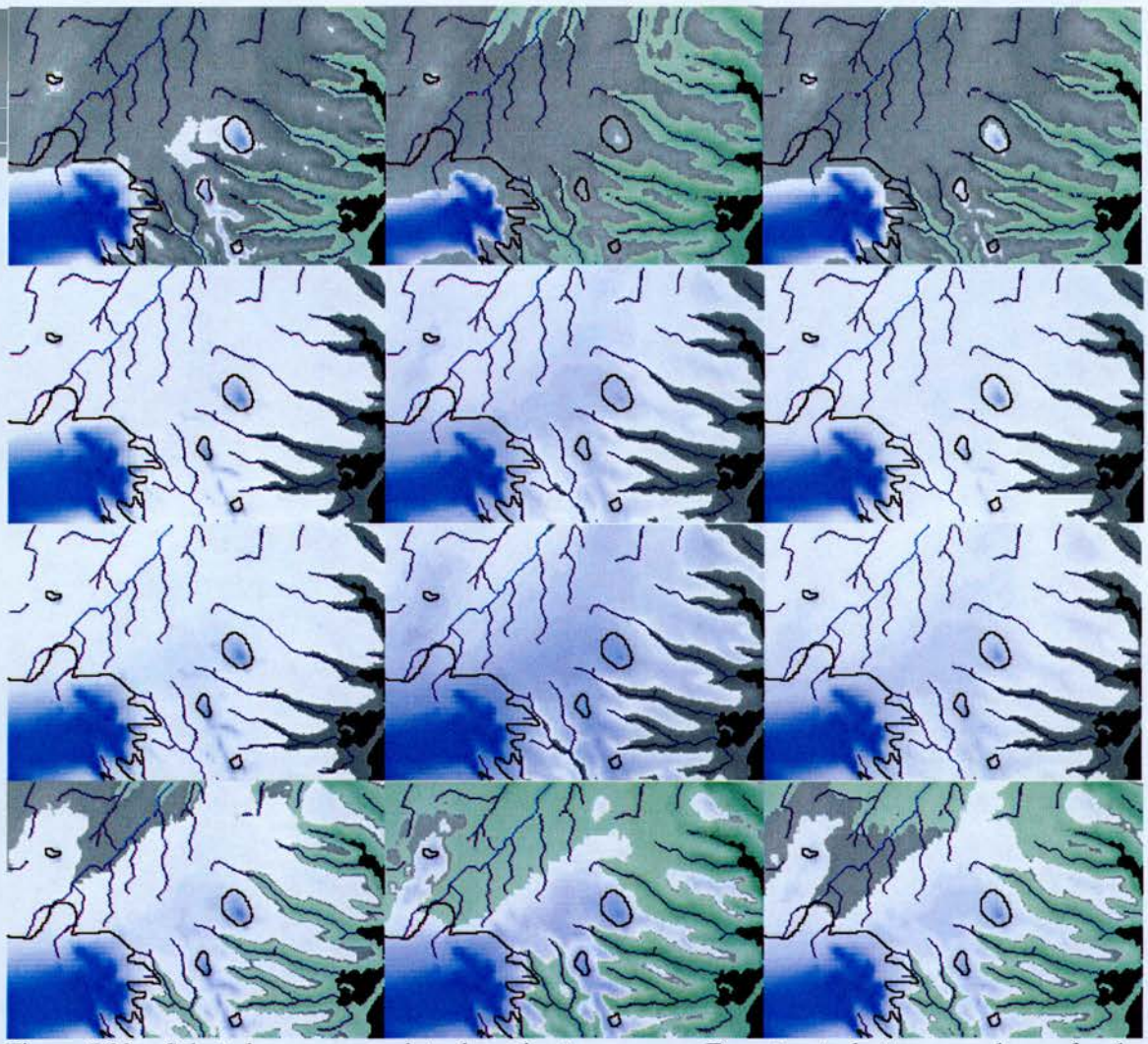

Figure 5.29: Selected snowcover plots throughout one year. From top to bottom, panels are for the end of September, December, March and June respectively. From left to right, panels are for temperatures of +1°C, 0°C and -1°C compared to the present. For each month, the area where the minimum degree day growth limit for grass is exceeded is shown (this can happen in areas outwith the total annual potential vegetation limit). Sandar areas show no vegetation or snowcover.

Figure 5.30: Hnutarvatn on the plateau north of Hofsjökull. Vegetation around the lake consists of mosses and some grass – this is the margin of grass survival at the present day.

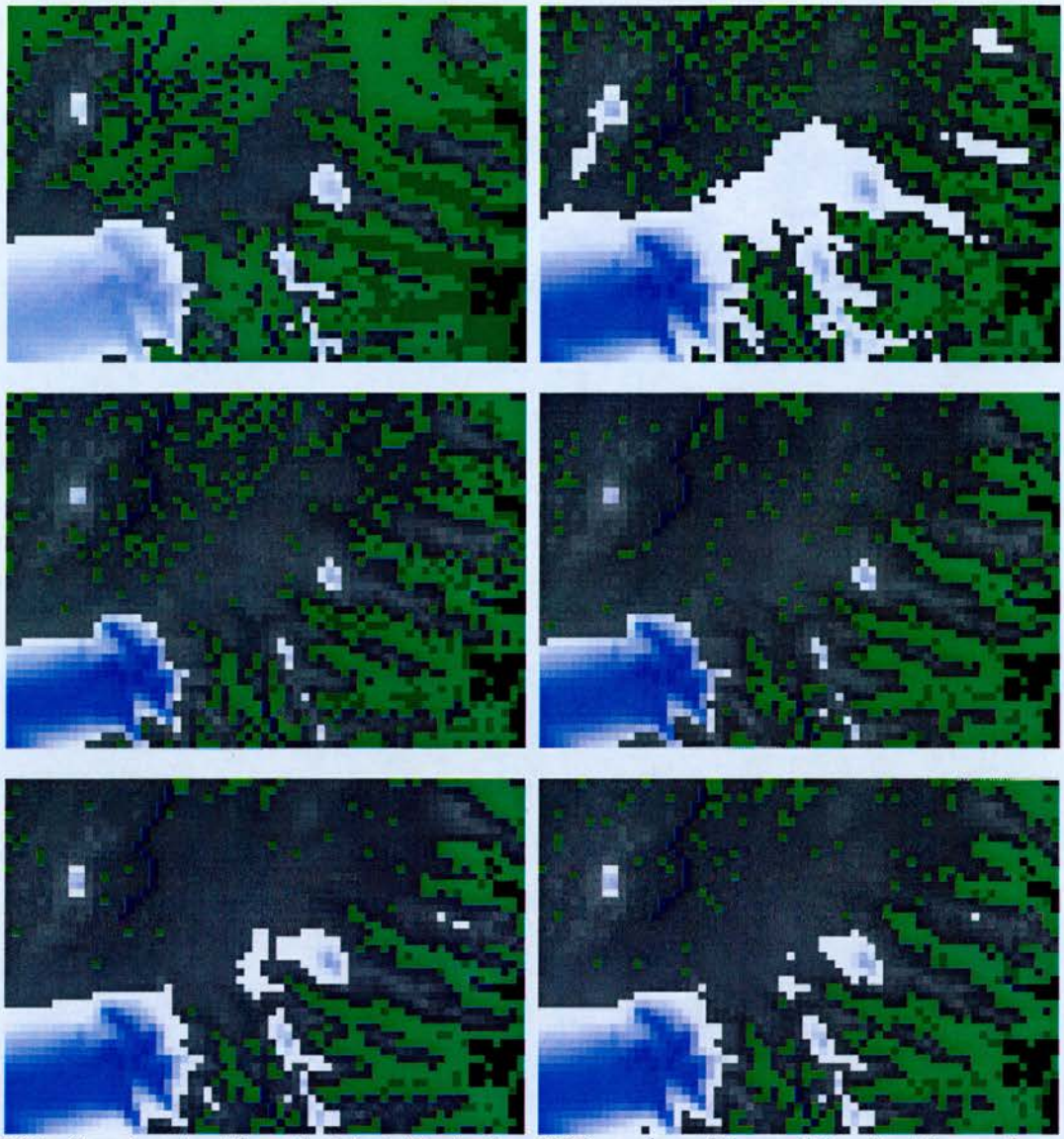

Figure 5.31: Sample output from the latest (September 2005) version of the model: output frames from a time-dependent model of the eastern Vatnajökull / Hofsjökull area. Temperature is varied each year (in this example, the Bergthórsson 1969 temperature curve was used). At each time step of 1 year, cooling brings a 30% chance of vegetation die-off if the PDD is below the growth threshold, and a 1% chance of new colonisation if the PDD is above the threshold. Although the colonisation and die-off rates are uncalibrated, it demonstrates the difficulties for recolonisation of the marginal plateau in a climate with slight cooling, while vegetation in the main valley systems survives relatively unscathed. Substantial sustained warmth is required to significantly revegetate the plateau.

Chapter VI: Conclusions

VI.1: Empirical data: notable findings

1: Across Iceland, 10 potential lake sites were identified for an original study into glacier retreat in Iceland. Two of these Skeiðsvatn and Hofsvötn, were analysed in detail, and of the remaining lakes, Hraunsvatn was identified as worthy of further investigation. The remaining sites were rejected for several reasons, and in particular the lake northeast of Snæfell was considered in detail before being rejected. Until this work, there has been no such investigation of glacier retreat in Iceland.

2: For the first time it is possible to constrain the magnitude of Medieval deglaciation in Iceland. The environmental record in the core Skeiðsvatn III was based on five parameters analysed at up to 250 levels in the core (1cm resolution). Within the catchment, 36km^2 was mapped, identifying many glacially-derived and non-glacial landforms. 16 aeolian sediment profiles were examined, identifying 83 tephra layers. In Skeiðsvatn III, a sedimentary transition has been identified at 636-686 cal. AD to 1σ, indicating the onset of glaciation in this catchment. Subsequent to the emplacement of ice in the catchment, this threshold is not recrossed during the subsequent 1350 years: the 'Medieval Warm Period', 'Little Ice Age' and recent periods. This is shown by the continuation if deposition of glacial silt towards the present day. There is a caveat to this in that although temperatures were not sufficiently warm for a sufficiently sustained period of time to melt the ice entirely, there may have been shorter episodes of ameliorated climate as is shown in some other records of Medieval climate. The second caveat is that this does not rule out increased precipitation during milder episodes, which would cancel the mass balance loss through temperature rise. However, Skeiðsvatn III provides the first lacustrine sedimentary constraint upon glacial retreat in Iceland.

3: Hofsvötn in eastern Iceland was also studied in detail. Three cores from the area were analysed for magnetic susceptibility (190 levels in Hofsvötn A1; >90 levels in Geithellnáarvatn and Hofsvötn B1), x-ray analysis and one for loss on ignition. The record from Hofsvötn indicated that there was very significant disturbance to the lakes during the late Little Ice Age, and so the lakes did not contain a record of Medieval glacier fluctuations.

4: Soil data, in combination with modelling data gives further insight into the relative importance of climate and human land management practices, and this can be used to identify threshold events. This can be seen in the new record from Geithellnadalur, based on 28 aeolian sediment profiles, including 185 identified tephras supplemented by 349 geochemical analyses of individual tephra grains. Geithellnadalur shows that environmental change in Iceland is not always directly related to climate, but that climate may have an indirect influence on landscape changes. In addition, 21 regional profiles of aeolian sediment accumulation, supported by further geochemical analyses, are presented to improve the tephrastratigraphy of southeastern Iceland and support the tephra identification in Geithellnadalur.

VI.2: Modelling: general conclusions

5: A new model of climate and vegetation has been created for Iceland. This is Java-based, and includes well over 12,000 lines of original code. The model shows that it is possible to model monthly and annual vegetation limits, snowlines and snowcover based on inputs of primarily topography, temperature and precipitation. The model is customisable to produce results for individual regions of Iceland, and to create maps either based on set input values for temperature and precipitation, or on a timeseries of annual input values.

6: Numerical modelling of mass balance and degree-days can evaluate climatic changes at a 250m resolution across Iceland (413,000 cells), and can constrain possible changes and spatial patterns. Models can be constrained by geomorphological (moraines / lake sediments) data and present-day ecological data, and they can be applied both to a regional scale and to a farm scale, unlike previous models that relate to one or the other.

7: Glacier models, mass balance models and ecological models can be integrated, showing the way forward for combined environmental modelling exercises. The principal benefit is that glacier change is testable in the palaeoenvironmental record, so tests can be applied to ecological reconstructions that cannot be tested through ecological data, which is vulnerable to anthropogenic influences. It can also be used in cases wher there is no existing ecological data. Crucially, the scale of this modelling is relevant to the scales of human development and impact: in Iceland this is down to the scale of individual landholdings, settlements and valleys.

VI.3: Integrated results

8: Thórsmörk: Integrated mass balance and ecological modelling combined with geomorphic data can be used to assess settlement changes in southern Iceland. Soil erosion in Thórsmörk was not directly driven by climatic change, and the degradation has a land management origin. The solution, as shown in the preservation of woodlands in Thórsmörk, was a land management decision, and not deterministically driven by a drop in temperature.

9: Mývatn: At Mývatn, and in particular in the environs of Sveigakót in the Kraká valley, climate does play a role in the extensive land degradation. The good quality of the land around Hofstaðir is a reflection of a non-marginal climate location, and is in contrast to the degraded land east of the Kraká river. The sensitivity of the landscape of the broad and flat Kraká valley to degradation is markedly higher than that found either closer to the north coast or in regions such as Thórsmörk. Erosion is likely to spread rapidly across the landscape due to shallow ecological gradients with no 'fire breaks' to halt the degradation.

10: Geithellnadalur: Climate change at Geithellnadalur is most marked on the plateau. The valley is non-marginal, though there are clearly impacts present in the $13^{th}/14^{th}$ Century. This impact is indirectly a climatic effect, as the upland rangelands are eliminated by the start of the 'Little Ice Age', as only a minor cooling is required to inhibit grass growth. The cascading impact is of increased grazing pressure on

lower rangelands and therefore sediment accumulations early on rival later Little Ice Age changes due to the impacts being played out at these early stages of the LIA.

In summary, these three case studies demonstrate the complexity of the climatic impact upon the Medieval Icelandic environment and on the Norse. The integrated geomorphological and numerical modelling approach has provided the tools to separate the twin impacts of climate and settlement on the Icelandic landscape, and shows that although climate is likely to have been the driver of some farm abandonments, others are much more likely to have had a human cause. Overall, it gives the potential for a more nuanced interpretation of landscape change that is integrated and testable.

References

Aðalgeirsdottir, G., Björnsson, H. and Jóhannessen, T. (2004). *Vatnajökull ice cap, results of computations with a dynamical model coupled with a degree-day mass-balance model.* Report Report: Science Institute, University of Iceland, RH-11-2004, 35 pp.

Aðalgeirsdottir, G., Guðmundsson, G. H. and Björnsson, H. (2003). A regression model for the mass-balance distribution of the Vatnajokull ice cap, Iceland. *Annals of Glaciology*, 37, 189-193.

Ahlmann, H. W. and Thorarinsson, S. (1937). Vatnajökull. Scientific results of Swedish - Icelandic investigations. *Geografiska Annaler*, 19, 146-231.

Alley, R. B., Mayewski, P. A., Sowers, T., Stuiver, M., Taylor, K. C. and Clark, P. U. (1997). Holocene climatic instability: A prominent, widespread event 8200 yr ago. *Geology*, **25**(6), 483-486.

Andrews, J. T. and Giraudeau, J. (2003). Multi-proxy records showing significant Holocene environmental variability: the inner N. Iceland shelf (Hunafloi). *Quaternary Science Reviews*, **22**(2-4), 175-193.

Andrews, J. T., Helgadottir, G., Geirsdottir, A. and Jennings, A. E. (2001a). Multicentury-scale records of carbonate (hydrographic?) variability on the northern Iceland margin over the last 5000 years. *Quaternary Research*, **56**(2), 199-206.

Andrews, J. T., Caseldine, C., Weiner, N. J. and Hatton, J. (2001b). Late Quaternary (~4 ka) marine and terrestrial environmental change in Reykjarfjördur, N. Iceland: climate and/or settlement? *Journal of Quaternary Science*, **16**, 133-144.

Andrews, J. T., Hardardo, R., Hardardottir, J., Kristjansdottir, G. B., Gronvold, K. and Stoner, J. S. (2003a). A high-resolution Holocene sediment record from Húnaflóaáll, N Iceland margin: century- to millennial-scale variability since the Vedde tephra. *Holocene*, **13**(5), 625-638.

Andrews, J. T., Hardadottir, J., Stoner, J. S., Mann, M. E., Kristjansdottir, G. B. and Koc, N. (2003b). Decadal to millennial-scale periodicities in North Iceland shelf sediments over the last 12000 cal yr: long-term North Atlantic oceanographic variability and solar forcing. *Earth and Planetary Science Letters*, **210**(3-4), 453-465.

Arnalds, Ó., Aradóttir, A. L. and Thorsteinsson, I. (1987). The nature and restoration of denuded areas in Iceland. *Arctic and Alpine Research*, 19(4), 518-525.

Arnalds, Ó., Thorarinsdóttir, E. F., Metusalemsson, S., Jonsson, A., Gretarsson, E. and Arnarson, A. (1997). *Soil Erosion in Iceland.* Icelandic SCS and the Agricultural Research Institute, Reykjavík pp.

Arneborg, J., Heinemeier, J., Lynnerup, N., Nielsen, H. L., Rud, N. and Sveinbjornsdottir, A. E. (1999). Change of diet of the Greenland Vikings determined from stable carbon isotope analysis and C-14 dating of their bones. *Radiocarbon*, **41**(2), 157-168.

Barlow, L. K., Sadler, J. P., Ogilvie, A. E. J., Buckland, P. C., Amorosi, T., Ingimundarson, J. H., Skidmore, P., Dugmore, A. J. and McGovern, T. H. (1997). Interdisciplinary investigations of the end of the Norse western settlement in Greenland. *Holocene*, **7**(4), 489-499.

Barr, I. (2003). *Mapping and Dating the 'Little Ice Age' Extent of Vatnsdalsjökull, Öxnadalur, Northern Iceland.* unpublished Undergraduate Dissertation, Department of Geography, Lancaster University. 74 pp.

Barsch, D. (1996). *Rockglaciers: indicators for the Present and Former Geocryology in High Mountain Environments.* Springer, Berlin. 331pp.

Beck, M. B., Jakeman, A. J. and McAleer, M. J. (1993). Construction and evaluation of models of environmental systems. In: Jakeman, A. J., Beck, M. B. and McAleer, M. J. (Eds.) *Modelling change in environmental systems.* Chichester, Wiley. 3-35.

Bell, W. T. and Ogilvie, A. E. J. (1978). Weather compilations as a source of data for the reconstruction of European climate during the Medieval Period. *Climatic Change*, **1**, 331-348.

Benn, D. I. and Evans, D. J. A. (1998). *Glaciers and Glaciation*. Arnold, London. 734pp.

Bentley, M. J. and Dugmore, A. J. (1998). Landslides and the rate of glacial trough formation. *Mountain Glaciation: Journal of Quaternary Science - Quaternary Proceedings*, 6, (ed. Owen, L.A.) 11-16.

Bergþórsson, P. (1969). An estimate of drift ice and temperature in Iceland in 1000 years. *Jökull*, **19**, 94-101.

Bergþórsson, P. (1985). Sensitivity of Icelandic agriculture to climatic variations. *Climatic Change*, 7(1), 111-127.

Bianchi, G. G. and McCave, I. N. (1999). Holocene periodicity in North Atlantic climate and deep-ocean flow south of Iceland. *Nature*, **397**, 515-517.

Bigg, G. R. (1996). *The Oceans and Climate.* Cambridge University Press, Cambridgepp.

Bingham, R. G. (1999). *Modelling ice sheet dynamics in response to climatic changes in southern Iceland.* Unpublished MRes thesis, Department of Geology and Geophysics, University of Edinburgh. 154 pp.

Bingham, R. G., Hulton, N. R. J. and Dugmore, A. J. (2003). Modelling the southern extent of the last Icelandic ice-sheet. *Journal of Quaternary Science*, 18(2), 169-181.

Birks, H. H., Battarbee, R. W. and Birks, H. J. B. (2000). The development of the aquatic ecosystem at Krakenes Lake, western Norway, during the late glacial and early Holocene - a synthesis. *Journal of Paleolimnology*, 23(1), 91-114.

Björnsson, H. (1979). Glaciers in Iceland. *Jökull*, 29, 74-80.

Black, J. L., Miller, G. H. and Geirsdóttir, Á. (2004). *Investigating Holocene climate change in glacial lake Hvítárvatn, Iceland.* 34th International Arctic Workshop, Program and Abstracts 2004., Institute of Arctic and Alpine Research, University of Colorado at Boulder. 33-34.

Bond, G., Showers, W., Cheseby, M., Lotti, R., Almasi, P., deMenocal, P., Priore, P., Cullen, H., Hajdas, I. and Bonani, G. (1997). A pervasive millennial-scale cycle in North Atlantic Holocene and glacial climates. *Science*, 278, 1257-1266.

Boulton, G. S., Hagdorn, M. and Hulton, N. R. J. (2003). Streaming flow in an ice sheet through a glacial cycle. In: *Annals of Glaciology, Vol 36.* Annals of Glaciology. 117-128.

Boygle, J. E. (1994). *Tephra in lake sediments: an unambiguous geochronological marker?* Unpublished PhD thesis, Department of Geography, University of Edinburgh.

Bradley, R. S. and Jones, P. D. (Eds.) (1995). *Climate Since A.D.1500*, Routledge. 724pp.

Bradley, R. S., Briffa, K. R., Crowley, T. J. and Hughes, M. K. (2001). The Scope of Medieval Warming. *Science*, **292**, 2011.

Bradwell, T. (2001a). *Glacier Fluctuations, Lichenometry and Climatic Change in Iceland.* Unpublished PhD Thesis, Department of Geography, University of Edinburgh. 359 pp.

Bradwell, T. (2001b). A new lichenometric dating curve for southeast Iceland. *Geografiska Annaler*, **83A**, 91-101.

Braithwaite, R. J. (1995). Positive degree-day factors for ablation on the Greenland ice sheet studied by energy balance modelling. *Journal of Glaciology*, 41(137), 153-160.

Brazier, V., Kirkbride, M. P. and Owens, I. F. (1998). The relationship between climate and rock glacier distribution in the Ben Ohau Range, New Zealand. *Geografiska Annaler Series a-Physical Geography*, 80A(3-4), 193-207.

Briffa, K. R., Bartholin, T. S., Eckstein, D., Jones, P. D., Karlen, W., Schweingruber, F. H. and Zetterberg, P. (1990). A 1,400-Year Tree-Ring Record of Summer Temperatures in Fennoscandia. *Nature*, **346**(6283), 434-439.

Broecker, W. (2001). Was the Medieval Warm Period Global? *Science*, **291**, 1497-1499.

Buckland, P. and Dugmore, A. (1991). 'If this is a refugium, why are my feet so bloody cold?' The origins of the Icelandic biota in the light of recent research. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland, Past and Present*, Kluwer Academic Publishers. 107-125.

Buckland, P. C., Gerrard, A. J., Sadler, J. P. and Sveinbjarnardóttir, G. (1994). Farmers, farm mounds and environmental change. In: Stötter, J. and Wilhelm, F. (Eds.) *Environmental Change in Iceland.* Münchener Geographische Abhandlungen, Reihe B, Band B12. 7-29.

Campbell, C. (1998). Late Holocene lake sedimentology and climate change in southern Alberta, Canada. *Quaternary Research*, 49(1), 96-101.

Caseldine, C. (1985). The extent of some glaciers in Northern Iceland during the Little Ice Age and the nature of recent deglaciation. *Geographical Journal*, 151, 215-227.

Caseldine, C. and Hatton, J. (1994). Interpretation of Holocene climatic change for the Eyjafjördur area of northern Iceland from pollen-analytical data: comments and preliminary results. In: Stötter, J. and Wilhelm, F. (Eds.) *Environmental Change in Iceland.* Münchener Geographische Abhandlungen, Reihe B, Band B12. 41-62.

Caseldine, C. and Stötter, J. (1993). 'Little Ice Age' glaciation of Tröllaskagi peninsula, northern Iceland: climatic implications for reconstructed Equilibrium Line Altitudes (ELAs). *The Holocene*, 3, 357-366.

Caseldine, C. J. (1987). Neoglacial Glacier Variations in Northern Iceland - Examples from the Eyjafjordur Area. *Arctic and Alpine Research*, 19(3), 296-304.

Casely, A. F. (2001). *'Little Ice Age' Glacier Fluctuations And Climatic Change In Iceland - A Multidisciplinary Approach*. Unpublished MRes Thesis, Department of Geology and Geophysics, University of Edinburgh. 113 pp.

Casely, A. F. and Dugmore, A. J. (2004). Climate change and 'anomalous' glacier fluctuations: the southwest outlets of Mýrdalsjökull, Iceland. *Boreas*, **33**(2), 108-122.

Clark, D. H., Steig, E. J., Potter, N. and Gillespie, A. R. (1998). Genetic variability of rock glaciers. *Geografiska Annaler Series a-Physical Geography*, 80A(3-4), 175-182.

Cook, E. R., Briffa, K. R., Meko, D. M., Graybill, D. A. and Funkhouser, G. (1995). The Segment Length Curse in Long Tree-Ring Chronology Development for Paleoclimatic Studies. *Holocene*, **5**(2), 229-237.

Crowley, T. J. (2000). Causes of climate change over the past 1000 years. *Science*, **289**(5477), 270-277.

Crowley, T. J. and Lowery, T. S. (2000). How warm was the medieval warm period? *Ambio*, **29**, 51-54.

Dahl, S. O. and Nesje, A. (1994). Holocene glacier fluctations at Hardangerjøkulen, central-southern Norway: a high-resolution composite chronology from lacustrine and terrestrial deposits. *The Holocene*, **4**(3), 269-277.

Dahl, S. O., Bakke, J., Lie, Ø. and Nesje, A. (2003). Reconstruction of former glacier equilibrium-line altitudes based on proglacial sites: an evaluation of approaches and selection of sites. *Quaternary Science Reviews*, 22, 275-287.

Dahl-Jensen, D., Mosegaard, K., Gundestrup, N., Clow, G. D., Johnsen, S. J., Hansen, A. W. and Balling, N. (1998). Past Temperatures Directly from the Greenland Ice Sheet. *Science*, **282**, 268-271.

Dansgaard, W., Johnsen, S. J., Reeh, N., Gundestrup, N., Clausen, H. B. and Hammer, C. U. (1975). Climatic changes, Norsemen and modern man. *Nature*, **255**, 24-28.

Dansgaard, W., White, J. W. C. and Johnsen, S. J. (1989). The Abrupt Termination of the Younger Dryas Climate Event. *Nature*, **339**(6225), 532-534.

Dawson, A. G., Hickey, K., Holt, T., Elliott, L., Dawson, S., Foster, I. D. L., Wadhams, P., Jonsdottir, I., Wilkinson, J., McKenna, J., Davis, N. R. and Smith, D. E. (2002). Complex North Atlantic Oscillation (NAO) Index signal of historic North Atlantic storm-track changes. *Holocene*, 12(3), 363-369.

Dawson, A. G., Elliott, L., Mayewski, P., Lockett, P., Noone, S., Hickey, K., Holt, T., Wadhams, P. and Foster, I. (2003). Late-Holocene North Atlantic climate 'seesaws', storminess changes and Greenland ice sheet (GISP2) palaeoclimates. *Holocene*, **13**(3), 381-392.

Dawson, A., Elliott, L., Noone, S., Hickey, K., Holt, T., Wadhams, P. and Foster, I. (2004). Historical storminess and climate 'see-saws' in the North Atlantic region. *Marine Geology*, 210(1-4), 247-259.

Diucilus (1967). *Diucili Liber de mensura orbis terrae [Diucil's book on the measure of the sphere of the earth]*. Edited by Tierney, J.J.: Scriptores Latini Hiberniae, **6**. Dublin, Dublin Institute for Advanced Studies.

Dowdeswell, J. A., Hagen, J. O., Björnsson, H., Glazovsky, A. F., Harrison, W. D., Holmlund, P., Jania, J., Koerner, R. M., Lefauconnier, B., Ommanney, C. S. L. and R.H., T. (1997). The mass balance of circum-Arctic glaciers and recent climate change. *Quaternary Research*, **48**, 1-14.

Dugmore, A. J. (1987). *Holocene glacier fluctuations around Eyjafjallajökull, south Iceland: A tephrochronological study.* Unpublished PhD Thesis, Department of Geography, University of Aberdeen. 118 pp.

Dugmore, A. J. (1989). Tephrochronological studies of Holocene glacier fluctuations in southern Iceland. In: Oerlemans, J. (Ed.) *Glacier Fluctuations and Climatic Change* Dordrecht, Kluwer Academic Publishers. 37-55.

Dugmore, A. J. and Buckland, P. (1991). Tephrochronology and Late Holocene soil erosion in south Iceland. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present* Dordrecht, Kluwer Academic Publishers. 147-159.

Dugmore, A. J. and Erskine, C. C. (1994). Local and regional patterns of soil erosion in southern Iceland. In: Stötter, J. and Wilhelm, F. (Eds.) *Environmental Change in Iceland.* 63-78.

Dugmore, A. J. and Sugden, D. E. (1991). Do the anomalous fluctuations of Sólheimajökull reflect ice-divide migration? *Boreas*, **20**, 105-113.

Dugmore, A. J., Church, M. J., Mairs, K., Newton, A. J. and Sveinbjarnardóttir, G. (in press). *An over-optimistic pioneer fringe? Environmental perspectives on medieval settlement abandonment in Þórsmörk, south Iceland.* Dynamics of Northern Societies: Proceedings of the 2004 SILA-NABO conference, Copenhagen, National Museum of Denmark.

Dugmore, A. J., Keller, C. and McGovern, T. H. (manuscript). Go quietly into the night? Conflict, climate change, adaptation and the end of Norse settlement in Greenland. *for submission to Journal of Peace Research.*

Dugmore, A. J., Larsen, G. and Newton, A. J. (1995). Seven tephra isochrones in Scotland. *The Holocene*, 5, 257-266.

Dugmore, A. J., Newton, A. J., Edwards, K. J., Larsen, G., Blackford, J. J. and Cook, G. T. (1996). Long-distance marker horizons from small-scale eruptions: British tephra deposits from the AD1510 eruption of Hekla, Iceland. *Journal of Quaternary Science*, 11, 511-516.

Dugmore, A. J., Newton, A. J., Larsen, G. and Cook, G. T. (2000). Tephrochronology, Environmental Change and the Norse Settlement of Iceland. *Environmental Archaeology*, 5, 21-34.

Dugmore, A. J., Newton, A. J., Sugden, D. E. and Larsen, G. (1992). Geochemical stability of fine-grained silicic Holocene tephra in Iceland and Scotland. *Journal of Quaternary Science*, 7, 173-183.

Eide, T. O. (1955). Breden og bygda. In: Lid, N. (Ed.) *Norveg 5*, H. Aschehoug. 1-42.

Eiríksson, J., Knudsen, K. L., Haflidason, H. and Henriksen, P. (2000a). Late-glacial and Holocene paleoceanography of the North Iceland Shelf. *Journal of Quaternary Science*, **15**, 23–42.

Eiríksson, J., Knudsen, K. L., Haflidason, H. and Heinemeier, H. (2000). Chronology of the late Holocene climatic events in the northern North Atlantic based on AMS 14C dates and tephra markers from the volcano, Hekla, Iceland. *Journal of Quaternary Science*, **15**, 573–580.

Esper, J., Cook, E. R. and Schweingruber, F. H. (2002). Low-frequency signals in long tree-ring chronologies for reconstructing past temperature variability. *Science*, **295**(5563), 2250-2253.

Evans, D. J. A., Archer, S. and Wilson, D. J. H. (1999). A comparison of the lichenometric and Schmidt hammer dating techniques based on data from the proglacial areas of some Icelandic glaciers. *Quaternary Science Reviews*, **18**, 13-41.

Feeny, D., Berkes, F. and others, a. (1990). The tragedy of the commons: twenty two years later. *Human Ecology*, 18(1), 1-19.

Fitzhugh, W. W. (2000). Puffins, ringed pins, and runestones: The Viking passage to America. In: Fitzhugh, W. W. and Ward, E. I. (Eds.) *Vikings: The North Atlantic Saga.* Washington and London, Smithsonian Institution Press. 10-25.

Flowers, G. E., Marshall, S. J., Bjornsson, H. and Clarke, G. K. C. (2005). Sensitivity of Vatnajokull ice cap hydrology and dynamics to climate warming over the next 2 centuries. *Journal of Geophysical Research-Earth Surface*, 110(F2), art. no.-F02011.

Friðriksson, S. and Sigurðsson, F. H. (1983). Áhrif lofthita á grassprettu [The influence of temperature on vegetation growth]. *Journal of Icelandic Agricultural Research*, 15, 41-54.

Furbish, D. J. and Andrews, J. T. (1984). The Use of Hyposometry to Indicate Long-Term Stability and Response of Valley Glaciers to Changes in Mass-Transfer. *Journal of Glaciology*, 30(105), 199-211.

Grönvold, K., Oskarsson, N., Johnsen, S. J., Clausen, H. B., Hammer, C. U., Bond, G. and Bard, E. (1995). Ash layers from Iceland in the Greenland GRIP ice core correlated with oceanic and land sediments. *Earth and Planetary Science Letters*, 135, 149-155.

Grove, J. M. (1988). *The Little Ice Age*. Methuen, London. 498pp.

Grove, J. M. (2001). The initiation of the "Little Ice Age" in regions round the North Atlantic. *Climatic Change*, **48**, 53-82.

Grove, J. M. and Switsur, R. (1994). Glacial geologic evidence for the Medieval Warm Period. *Climatic Change*, **30**, 1-27.

Grudd, H., Briffa, K. R., Karlen, W., Bartholin, T. S., Jones, P. D. and Kromer, B. (2002). A 7400-year tree-ring chronology in northern Swedish Lapland: natural climatic variability expressed on annual to millennial timescales. *The Holocene*, **12**, 657-665.

Guðmundsson, S., Björnsson, H., Pálsson, F. and Haraldsson, H. H. (2003). *Comparison of physical and regression models of summer ablation on ice caps in Iceland*. Report: Science Institute, University of Iceland, National Power Company of Iceland, RH-15-2003, 31 pp.

Guðmundsson, S., Björnsson, H., Pálsson, F. and Haraldsson, H. H. (2003). *Physical energy balance and degree-day models of summer ablation on Langjökull ice cap, SW-Iceland*. Report: Science Institute, University of Iceland, National Power Company of Iceland, RH-20-2003, 20 pp.

Guðmunsson, H. J. (1997). A review of the Holocene environmental history of Iceland. *Quaternary Science Reviews*, **16**, 81-92.

Gylfadóttir, S. (2003). *Spatial interpolation of Icelandic monthly mean temperature data*. Report Report: Veðurstofa Íslands, Reykjavík. 27 pp.

Häberle, T. (1991). Holocene glacial history of the Hörgardalur area, Tröllaskagi, northern Iceland. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present*. Dordrecht, Kluwer Academic Publishers. 193-202.

Häberle, T. (1991). Holocene glacial history of the Hörgardalur area, Tröllaskagi, northern Iceland. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present*. Dordrecht, Kluwer Academic Publishers. 193-202.

Häberle, T. (1994). Glacial, Late Glacial and Holocene history of the Hörgárdalur area, Tröllaskagi, Northern Iceland. In: Stötter, J. and Wilhelm, F. (Eds.) *Environmental Change in Iceland*. 41-63.

Hagdorn, M., Rutt, I. and Payne, T. (2005). *GLIMMER 0.5.2 Documentation*. http://xweb.geos.ed.ac.uk/~mhagdorn/glide/glimmer.pdf

Haines-Young, R. H. and Petch, J. R. (1986). *Physical Geography: Its Nature and Methods*. Chapman, Londonpp.

Hallsdóttir, M. (1987). *Pollen analytical studies of human influence on vegetation in relation to the Landnám tephra layer in southwest Iceland.* Lundqua Thesis. 18. Lund University, Lund.

Haraldsson, H. (1981). The Markarfljót sandur area, southern Iceland. *Striae*, 15, 1-58.

Haraldsson, H. V. and Ólafsdóttir, R. (2003). Simulating vegetation cover dynamics with regards to long-term climatic variations in sub-arctic landscapes. *Global and Planetary Change*, 38(3-4), 313-325.

Harbor, J. M. (1993). Glacial geomorphology - modeling processes and landforms. *Geomorphology*, 7, 129-140.

Hill, K. M. (2005). *Late Holocene environmental change as recorded in sedimentary diatom assemblages from a glacier-fed lake in northern Iceland.* unpublished BSc Dissertation, School of Geography, University of Nottingham. 71 pp.

Hjartarson, Á. and Ingólfsson, Ó. (1988). Preboreal glaciation of southern Iceland. *Jökull*, 38, 1-16.

Hodell, D. A., Curtis, J. H., Jones, G. A., Higueragundy, A., Brenner, M., Binford, M. W. and Dorsey, K. T. (1991). Reconstruction of Caribbean Climate Change over the Past 10,500 Years. *Nature*, **352**(6338), 790-793.

Holzhauser, H. (1984). Zur Geschichte der Aletschgletscher und Fieshgletscher. *Physiche Geographie (Geographisches Institut der Universität Zürich)*, **13**, 1-448.

Holzhauser, H. (1997). Fluctuations of the Grosser Aletsch Glacier and Gorner Glacier during the last 3200 years: new results. *Paläoklimaforschung/Palaeoclimate Research*, **24 (special issue 16)**, 36-58.

Howard, T. M. (2005). *Environmental change in Geithelladalur, east Iceland. Anthropogenic and climatic impacts on soil erosion and landscape change.* Unpublished Undergraduate Dissertation, Institute of Geography, University of Edinburgh. 75 pp.

Hughes, M. K. and Diaz, H. F. (1994). Was there a "Medieval Warm Period", and if so, where and when? *Climatic Change*, **26**, 109-142.

Hulton, N. R. J., Purves, R. S., McCulloch, R. D., Sugden, D. E. and Bentley, M. J. (2002). The Last Glacial Maximum and deglaciation in southern South America. *Quaternary Science Reviews*, 21(1-3), 233-241.

Hurrell, J. W. (1995). Decadal Trends in the North-Atlantic Oscillation - Regional Temperatures and Precipitation. *Science*, **269**(5224), 676-679.

Hurrell, J. W. and van Loon, H. (1997). Decadal variations in climate associated with the north Atlantic oscillation. *Climatic Change*, 36(3-4), 301-326.

Ingram, M. J., Underhill, D. J. and Wigley, T. M. L. (1978). Historical Climatology. *Nature*, **276**(5686), 329-334.

IPCC (2001). *Climate Change 2001: The Scientific Basis. Contribution of Working Group I to the Third Assessment Report of the Intergovernmental Panel on Climate Change.* Report: Cambridge University Press, [Houghton, J.T.,Y. Ding, D.J. Griggs, M. Noguer, P.J. van der Linden, X. Dai, K. Maskell, and C.A. Johnson (eds.)] Cambridge, United Kingdom and New York, NY, USA. 881 pp.

Maskell, and C.A. Johnson (eds.)]. Report Report: Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA. 881pp pp.

Jamieson, S. S. R., Hulton, N. R. J., Sugden, D. E., Payne, A. J. and Taylor, J. (2005). Cenozoic landscape evolution of the Lambert basin, East Antarctica: the

relative role of rivers and ice sheets. *Global and Planetary Change*, 45(1-3), 35-49.

Jennings, A. E. and Weiner, N. J. (1996). Environmental change in eastern Greenland during the last 1300 years: Evidence from foraminifera and lithofacies in Nansen Fjord, 68 degrees N. *Holocene*, **6**(2), 179-191.

Jensen, K. G., Kuijpers, A., Koc, N. and Heinemeier, J. (2004). Diatom evidence of hydrographic changes and ice conditions in Igaliku Fjord, South Greenland, during the past 1500 years. *Holocene*, **14**(2), 152-164.

Jiang, H., Seidenkrantz, M. S., Knudsen, K. L. and Eiriksson, J. (2002). Late-Holocene summer sea-surface temperatures based on a diatom record from the north Icelandic shelf. *Holocene*, **12**(2), 137-147.

Jones, G. (1984). *A history of the Vikings*. Oxford University Press., Oxfordpp.

Jones, P. D. and Bradley, R. S. (1995). Climatic variations over the last 500 years. In: Bradley, R. S. and Jones, P. D. (Eds.) *Climate Since A.D.1500 (Chapter 33)*, Routledge. 649-665.

Jones, P. D., Briffa, K. R., Barnett, T. P. and Tett, S. F. B. (1998). High-resolution palaeoclimatic records for the last millennium: interpretation, integration and comparison with General Circulation Model control-run temperatures. *The Holocene*, **8**(4), 455-471.

Kaplan, M. R., Wolfe, A. P. and Miller, G. H. (2002). Holocene environmental variability in southern Greenland inferred from lake sediments. *Quaternary Research*, 58(2), 149-159.

Karlén, W. (1976). Lacustrine sediments and tree-limit variations as indicators of Holocene climatic fluctuations in Lappland, Northern Sweden. *Geografiska Annaler*, 58A(1-2), 1-34.

Karlén, W. (1981). Lacustrine sediment studies. *Geografiska Annaler*, 63A(3-4), 273-281.

Karlén, W. (1988). Scandinavian glacial and climatic fluctuations during the Holocene. *Quaternary Science Reviews*, 7, 199-209.

Kayastha, R. B., Ageta, Y., Nakawo, M., K., F., Sakai, A. and Matsuda, Y. (2003). Positive degree-day factors for ice ablation on four glaciers in the Nepalese Himalayas and Qinghai-Tibetan Plateau. *Bulletin of Glaciological Research*, 20, 7-14.

Keigwin, L. D. (1996). The Little Ice Age and Medieval Warm Period in the Sargasso Sea. *Science*, **274**, 1503-1508.

Kirkbride, M. P. and Brazier, V. (1998). A critical evaluation of the use of glacier chronologies in climatic reconstruction, with reference to New Zealand. *Quaternary Proceedings*, **6**, 55-64.

Kirkbride, M. P. and Dugmore, A. J. (2001a). Timing and significance of mid-Holocene glacier advances in northern and central Iceland. *Journal of Quaternary Science*, 16(2), 145-153.

Kirkbride, M. P. and Dugmore, A. J. (2001b). Can lichenometry be used to date the "Little Ice Age" glacial maximum in Iceland? *Climatic Change*, 48, 151-167.

Kirkbride, M. P. and Dugmore, A. J. (2003). Glaciological response to distal tephra fallout from the 1947 eruption of Hekla, south Iceland. *Journal of Glaciology*, 49(166), 420-428.

Koch, L. (1945). *The East Greenland Ice.* Meddelelser om Grønland. 130. Copenhagen. 373 pp.

Kugelmann, O. (1991). Dating recent glacier advances in the Svarvaðardalur/Skíðadalur area of northern Iceland by means of a new lichen

curve. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present.* Dordrecht, Kluwer Academic Publishers. 203-217.

Lamb, H. H. (1977). *Climate, Past, Present and Future. Volume 2: Climatic History and the Future.* Methuen, London pp.

Lamb, H. H. (1979). Climate variation and changes in the wind and ocean circulation: the Little Ice Age in the northeast Atlantic. *Quaternary Research*, 11, 1-20.

Larsen, G. (1981). Tephrochronology by microprobe glass analysis. In: Self, S. and Sparks, R. S. J. (Eds.) *Tephra Studies* Dordrecht, D. Reidel. 95-102.

Larsen, G. (1984). Recent volcanic history of the Veidivötn fissure swarm, southern Iceland - an approach to volcanic risk assessment. *Journal of Volcanology and Geothermal Research*, 22, 33-58.

Larsen, G., Dugmore, A. and Newton, A. (1999). Geochemistry of historical-age silicic tephras in Iceland. *The Holocene*, 9, 463-471.

Larsen, G., Newton, A. J., Dugmore, A. J. and Vilmundardóttir, E. G. (2001). Geochemistry, dispersal volumes and chronology of Holocene silicic tephra layers from the Katla volcanic system, Iceland. *Journal of Quaternary Science*, 16, 119-132.

Lassen, S. J., Kuijpers, A., Kunzendorf, H., Hoffmann-Wieck, G., Mikkelsen, N. and Konradi, P. (2004). Late-Holocene Atlantic bottom-water variability in Igaliku Fjord, South Greenland, reconstructed from foraminifera faunas. *Holocene*, **14**(2), 165-171.

Lawson, I., Gathorne-Hardy, F. J., Church, M. J., Einarsson, A., Edwards, K. J., Perdikaris, S., McGovern, T. H., Amundsen, C. and Svebjarnardóttir, G. (in press). *Human impacts on freshwater environments in Norse and early medieval Iceland.* Dynamics of Northern Societies: Proceedings of the 2004 SILA-NABO conference, Copenhagen, National Museum of Denmark.

Lowe, J. J. and Walker, M. J. C. (1997). *Reconstructing Quaternary Environments.* Longman, Harlow pp.

Mackintosh, A. N. (2000). *Glacier fluctuations and climatic change in Iceland.* Unpublished PhD Thesis, Department of Geography, University of Edinburgh.

Mackintosh, A. N. and Dugmore, A. J. (2000). Modelling Holocene glacier fluctuations and climatic change in Iceland. *GeoLines*, 11(142-146).

Mackintosh, A. N., Dugmore, A. J. and Hubbard, A. L. (2002). Holocene climatic changes in Iceland: evidence from modelling glacier length fluctuations at Solheimajokull. *Quaternary International*, **91**, 39-52.

Magnusson, E., Bjornsson, H., Dall, J. and Palsson, F. (2005). Volume changes of Vatnajokull ice cap, Iceland, due to surface mass balance, ice flow, and subglacial melting at geothermal areas. *Geophysical Research Letters*, 32(5), art. no.-L05504.

Mann, M. E., Bradley, R. S. and Hughes, M. K. (1999). Northern hemisphere temperatures during the past millennium: Inferences, uncertainties, and limitations. *Geophysical Research Letters*, **26**, 759-762.

Martin, H. E. and Whalley, W. B. (1987a). A glacier ice-cored rock glacier in Tröllaskagi, Northern Iceland. *Jökull*, 37, 49-55.

Martin, H. E. and Whalley, W. B. (1987b). Rock Glaciers .1. Rock Glacier Morphology - Classification and Distribution. *Progress in Physical Geography*, 11(2), 260-282.

Martin, H. E., Whalley, W. B. and Caseldine, C. (1991). Glacier fluctuations and rock glaciers in Tröllaskagi, Northern Iceland, with special reference to 1946-1986.

In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present* Dordrecht, Kluwer Academic Publishers. 255-265.

Matthews, J. A. and Karlén, W. (1992). Asynchronous Neoglaciation and Holocene Climatic-Change Reconstructed from Norwegian Glaciolacustrine Sedimentary Sequences. *Geology*, 20(11), 991-994.

Mayewski, P. A., Meeker, L. D., Morrison, M. C., Twickler, M. S., Whitlow, S., Ferland, K. K., Meese, D. A., Legrand, M. R. and Steffenson, J. P. (1993). Greenland ice core 'signal' characteristics: an expanded view of climate change. *Journal of Geophysical Research*, **98**(D7), 12839-12847.

McGovern, T. H., Bigelow, G. F., Amorosi, T. and Russell, D. (1988). Northern Islands, human error and environmental degradation: a view of social and ecological change in the Medieval North Atlantic. *Human Ecology*, 16, 225-270.

McGovern, T. H., Perdikaris, S., Einarsson, A. and Sidell, J. (in press). Inland cod and sustainable waterfowl: wild species use in the Viking age, Mývatnssveit, N Iceland. *Environmental Archaeology*.

McGovern, T. H., Vésteinsson, O., Fridriksson, A., Church, M., Lawson, I., Simpson, I. A., Einarsson, A., Dugmore, A., Cook, G., Perdikaris, S., Edwards, K., Thompson, A. M., Adderley, W. P., Newton, A., Lucas, G. and Aldred, O. (2005). Landscapes of Settlement in Northern Iceland: historical ecology of human impact and climate fluctuation on the millennial scale. *American Anthropologist*, submitted.

Meese, D. A., Gow, A. J., Grootes, P., Mayewski, P. A., Ram, M., Stuiver, M., Taylor, K. C., Waddington, E. D. and Zielinski, G. A. (1994). The Accumulation Record from the Gisp2 Core as an Indicator of Climate-Change Throughout the Holocene. *Science*, **266**(5191), 1680-1682.

Meyer, H. H. and Venzke, J. F. (1985). Der Klængshóll Kargletscher in Nordisland. *Natur und Museum*, 115, 29-46.

Nesje, A. and Kvamme, M. (1991). Holocene glacier and climate variations in western Norway: evidence for early Holocene glacier demise and multiple Neoglacial events. *Geology*, **19**, 610-612.

Nesje, A., Dahl, S. O., Løvlie, R. and Sulebak, J. R. (1994). Holocene glacier activity at the southwestern part of Hardangerjøkulen, central-southern Norway: evidence from lacustrine deposits. *The Holocene*, **4**(4), 377-382.

Nesje, A., Kvamme, M., Rye, N. and Lovlie, R. (1991). Holocene Glacial and Climate History of the Jostedalsbreen Region, Western Norway - Evidence from Lake-Sediments and Terrestrial Deposits. *Quaternary Science Reviews*, **10**(1), 87-114.

Nesje, A., Lie, O. and Dahl, S. O. (2000). Is the North Atlantic Oscillation reflected in Scandinavian glacier mass balance records? *Journal of Quaternary Science*, 15(6), 587-601.

Nesje, A., Matthews, J. A., Dahl, S. O., Berrisford, M. S. and Andersson, C. (2001). Holocene glacier fluctuations of Flatebreen and winter-precipitation changes in the Jostedalsbreen region, western Norway, based on glaciolacustrine sediment records. *The Holocene*, **11**(3), 267-280.

Norðdahl, H. and Einarsson, T. (2001). Concurrent changes of relative sea-level and glacier extent at the Weichselian-Holocene boundary in Berufjordur, Eastern Iceland. *Quaternary Science Reviews*, 20(15), 1607-1622.

O'Brien, S., Mayewski, P., Meeker, L., Meese, D., Twickler, M. and Whitlow, S. (1995). Complexity of Holocene climate as reconstructed from a Greenland Ice Core. *Science*, **270**, 1962-1964.

Oerlemans, J. (1992). Climate sensitivity of glaciers in southern Norway: application of an energy-balance model to Nigardsbreen, Hellstugubreen and Alftobreen. *Journal of Glaciology*, 38, 223-243.

Oerlemans, J. (1996). Modelling the response of valley glaciers to climatic change. In: Boutron, C. (Ed.) *Physics and Chemistry of the Earth and other Objects of the Solar System* Les Ulis, Les Editions de Physique. 91-123.

Oerlemans, J. (1997). A flowline model for Nigardsbreen, Norway: projection of future glacier length based on dynamic calibration with the historical record. *Annals of Glaciology*, 24, 382-389.

Ogilvie, A. E. J. (1984). The past climate and sea-ice record from Iceland, part 1: data to A.D.1780. *Climatic Change*, **6**, 131-152.

Ogilvie, A. E. J. (1991). Climatic changes in Iceland, A.D. c.865 to 1598. *Acta Archaeologica*, **61**, 233-251.

Ogilvie, A. E. J. (1992). Documentary evidence for changes in the climate of Iceland, AD1500 to 1800. In: Bradley, R. S. and Jones, P. D. (Eds.) *Climate Since A.D. 1500.* London, Routledge. 92-117.

Ogilvie, A. E. J. (2000). North Atlantic climate c. AD 1000: Millennial reflections on the Viking discoveries of Iceland, Greenland and North America. *Weather*, **55**, 34-45.

Ogilvie, A. E. J. and Farmer, G. (1997). Documenting the medieval climate. In: Hulme, M. and Barrow, E. (Eds.) *Climates of the British Isles. Present, past and future* London, Routledge. 112–133.

Ogilvie, A. E. J. and Jonsson, T. (2001). "Little Ice Age" research: A perspective from Iceland. *Climatic Change*, **48**(1), 9-52.

Ogilvie, A. E. J. and McGovern, T. H. (2000). Sagas and science: climate and human impacts in the North Atlantic. In: Fitzhugh, W. W. and Ward, E. I. (Eds.) *Vikings: The North Atlantic Saga.* Washington and London, Smithsonian Institution Press. 385-393.

Ólafsdóttir, R. (2001). *Land degradation and climate in Iceland: a spatial and temporal assessment*. Meddelanden från Lunds Universitets Geografiska Institutioner. Avhandlingar 143. Lund. 128 pp.

Ólafsdóttir, R. and Júlíusson, Á. D. (2000). Farmers' perception of land-cover changes in NE Iceland. *Land Degradation and Development*, 11, 439-458.

Ólafsdóttir, R. and Schlyter, P. (2001). *Erosion patches - their spatial distribution and morphometry - in the Mývatnsheiði region, NE Iceland.* In: Ólafsdóttir, R.: Land degradation and climate in Iceland: a spatial and temporal assessmentMeddelanden från Lunds Universitets Geografiska Institutioner. Lund. Avhandlingar 143. Paper II. 12 pp.

Ólafsdóttir, R., Schlyter, P. and Haraldsson, H. V. (2001). Simulating Icelandic vegetation cover during the Holocene - Implications for long-term land degradation. *Geografiska Annaler Series a-Physical Geography*, 83A(4), 203-215.

Ólafsdóttir, R. and Guðmundsson, H. J. (2002). Holocene land degradation and climatic change in northeastern Iceland. *Holocene*, **12**(2), 159-167.

Ólafsson, H. (2000). Sagas of Western Expansion. In: Fitzhugh, W. W. and Ward, E. I. (Eds.) *Vikings: The North Atlantic Saga.* Washington and London, Smithsonian Institution Press. 143-145.

Parry, M. L. (1978). *Climate change, agriculture and settlement.* Dawson, Folkestone. 214pp.

Parry, M. L. (1990). *Climate Change and World Agriculture.* Earthscan Publications Limited, London. 157pp.

Paterson, W. S. B. (1994). *The Physics of Glaciers, 3rd Edition.* Pergammon, Oxfordpp.

Pfister, C., Luterbacher, L., Schwartz-Zanetti, G. and Wegmann, M. (1998). Winter air temperature variations in western Europe during the Early and High Middle Ages (AD750-1300). *The Holocene*, **8**, 535-552.

Pfister, C., Schwarz-Zanetti, G., Hochstrasser, F. and Wegmann, M. (1996). Winter severity in Europe: the Fourteenth century. *Climatic Change*, **34**(1), 91-108.

Pohjola, V. A. and Rogers, J. C. (1997). Atmospheric circulation and variations in Scandinavian glacier mass balance. *Quaternary Research*, **47**(1), 29-36.

Ruddiman, W. F. (2003). The anthropogenic greenhouse era began thousands of years ago. *Climatic Change*, 61(3), 261-293.

Ruddiman, W. F. and Thomson, J. S. (2001). The case for human causes of increased atmospheric CH4. *Quaternary Science Reviews*, 20(18), 1769-1777.

Schledermann, P. (2000). A.D. 1000: East meets West. In: Fitzhugh, W. W. and Ward, E. I. (Eds.) *Vikings: The North Atlantic Saga.* Washington and London, Smithsonian Institution Press. 188-192.

Sharp, M. J. and Dugmore, A. J. (1985). Holocene glacier fluctuations in East Iceland. *Zeitschrift für Gletscherkunde un Glazialgeologie*, **21**, 341-349.

Sigtryggsson, H. (1972). An outline of sea ice conditions in the vicinity of Iceland. *Jökull*, **19**, 7-10.

Simpson, I. A., Dugmore, A. J., Thomson, A. and Vesteinsson, O. (2001). Crossing the thresholds: human ecology and historical patterns of landscape degradation. *Catena*, **42**(2-4), 175-192.

Simpson, I. A., Vesteinsson, O., Adderley, W. P. and McGovern, T. H. (2003). Fuel resource utilisation in landscapes of settlement. *Journal of Archaeological Science*, **30**(11), 1401-1420.

Snowball, I. and Sandgren, P. (1996). Lake sediment studies of Holocene glacial activity in the Karsa valley, northern Sweden: Contrasts in interpretation. *Holocene*, 6(3), 367-372.

Snowball, I. F. (1993). Mineral Magnetic-Properties of Holocene Lake-Sediments and Soils from the Karsa Valley, Lappland, Sweden, and Their Relevance to Paleoenvironmental Reconstruction. *Terra Nova*, **5**(3), 258-270.

Snowball, I. F. (1995). *Mineral magnetic and geochemical properties of Holocene sediment and soils in the Abisko region of northern Sweden*, LUNDQUA Thesis 34, Department of Quaternary Geology, Lund University.

Snowball, I. F. (1996). Holocene environmental change in the Abisko region of northern Sweden recorded by the mineral magnetic stratigraphy of lake sediments. *Gff*, **118**, 9-17.

Snowball, I., Zillen, L. and Gaillard, M. J. (2002). Rapid early-Holocene environmental changes in northern Sweden based on studies of two varved lake-sediment sequences. *Holocene*, 12(1), 7-16.

Stötter, J. (1991). New observations on the postglacial glacial history of Tröllaskagi, Northern Iceland. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present* Dordrecht, Kluwer Academic Publishers. 181-192.

Stötter, J. and Wilhelm, F. (1994). Iceland's role for investigating environmental changes. In: Stötter, J. and Wilhelm, F. (Eds.) *Environmental Change in Iceland.* Münchener Geographische Abhandlungen, Reihe B, Band B12. 1-5.

Stotter, J., Wastl, M., Caseldine, C. and Haberle, T. (1999). Holocene palaeoclimatic reconstruction in northern Iceland: approaches and results. *Quaternary Science Reviews*, **18**(3), 457-474.

Sugden, D. E., Hulton, N. R. J. and Purves, R. S. (2002). Modelling the inception of the Patagonian icesheet. *Quaternary International*, 95-6, 55-64.

Sveinbjarnardóttir, G. (1991). A study of farm abandonment in two regions of Iceland. In: Maizels, J. K. and Caseldine, C. (Eds.) *Environmental Change in Iceland: Past and Present.* Dordrecht, Kluwer Academic Publishers. 161-177.

Sveinbjarnardóttir, G. (1992). *Farm abandonment in Medieval and post-Medieval Iceland: an interdisciplinary study.* Oxbow Monograph. 17. Oxford. 192 pp.

Thompson, L. (1991). Ice core records with emphasis on the global record of the last 2000 years. In: Bradley, R. (Ed.) *Global Changes of the Past.* Boulder, CO, UCAR/Office for Interdisciplinary Earth Studies. 201-224.

Thorarinsson, S. (1943). Oscillations of the Iceland glaciers in the last 250 years. *Geografiska Annaler*, 25, 1-54.

Thorarinsson, S. (1944). Tefrokronologiska studier på Island (Tephrochronological studies in Iceland). *Geografiska Annaler*, 26, 1-217.

Thórarinsson, S. (1956). *The thousand years struggle against ice and fire.* Museum of Natural History Occasional Papers. 14. National Museum of Iceland, Reykjavík.

Thórarinsson, S. (1966). The age of the maximum post-glacial advance of Hagafellsjökull Eystri. *Jökull*, **16**, 207-210.

Thorarinsson, S. (1967). The eruptions of Hekla in historical times. In: *The eruption of Hekla 1947-1948 II(3).* 1-68.

Thorarinsson, S. (1974). The terms Tephra and Tephrochronology. In: Westgate, J. A. and Gold, C. M. (Eds.) *The World Bibliography and Index of Quaternary Tephrochronology.* Edmonton, University of Alberta. 27-28.

Thoroddsen, T. (1916-17). Aferði a Íslandi í þusund ar. 1-2.

Tomkins, J. D. and Lamoureux, S. F. (2004). *Decadal hydroclimatic variability as evidenced by the sedimentary record of Mirror Lake, Northwest Territories, Canada.* 34th International Arctic Workshop, Program and Abstracts 2004., Institute of Arctic and Alpine Research, University of Colorado at Boulder. 166.

UKMO (2004). *Archiv der 00 UTC UKMO-Bracknell-Bodenanalysen (ab 27.01.1998) [Archive of 00 UTC UK Meterological Office - Bracknell surface pressure charts (from 27/1/1998)]*, Wetterzentrale.

Vésteinsson, O. (2000). The archaeology of landnám: Early Settlement in Iceland. In: Fitzhugh, W. W. and Ward, E. I. (Eds.) *Vikings: The North Atlantic Saga.* Washington and London, Smithsonian Institution Press. 164-174.

Vésteinsson, O. (Ed.) (2001). *Archaeological investigations at Sveigakot 1998–2000.* Reykjavík, Fornleifastofnun Iuslandspp.

Walden, J., Oldfield, F. and Smith, J. (1999). *Environmental magnetism: a practical guide.* Technical Guide No 6. Quaternary Research Association, London. 243pp.

Wallace, B. L. (1991). L'Anse Aux Meadows: Gateway to Vinland. *Acta Archaeologica*, **61**, 166-198.

Wanner, H., Bronnimann, S., Casty, C., Gyalistras, D., Luterbacher, J., Schmutz, C., Stephenson, D. B. and Xoplaki, E. (2001). North Atlantic Oscillation - Concepts and studies. *Surveys in Geophysics*, 22(4), 321-382.

Appendix 1

Microprobe data:

VDL: Vatnsdalur samples

HV: Samples from Hraunsvatn valley

HDR/HOF: Hofsdalur samples

GET: Geithellnadalur samples

FGD: Flugustaðadalur samples

LON: Lónsoraefi samples

Oxide	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
1 / 1 .	35.354	0.038	1.743	27.18	0.437	0.116	32.656	0.03	0.003	0.249	97.806	andradite
2 / 1 .	35.47	0.015	1.653	27.933	0.397	0.086	32.181	0.007	0	0.253	97.995	andradite 12:37
3 / 1 .	35.334	0.048	1.719	27.468	0.415	0.06	32.318	0	0.046	0.256	97.665	andradite 12:38
4 / 1 .	69.933	0.16	14.01	2.984	0.111	0.135	1.995	5.141	2.715	0.077	97.26	VDL2-1 1
5 / 1 .	70.334	0.2	14.113	3.021	0.095	0.117	2.08	5.174	2.45	0.035	97.618	VDL2-1 2
6 / 1 .	65.364	0.364	14.88	5.625	0.171	0.369	3.398	4.996	1.928	0.094	97.189	VDL2-1 3
8 / 1 .	69.907	0.185	14.037	3.253	0.098	0.139	1.882	5.233	2.491	0.004	97.23	VDL2-1 5
9 / 1 .	70.802	0.167	14.367	3.146	0.099	0.12	1.99	5.24	2.437	0.005	98.371	VDL2-1 6
11 / 1 .	71.135	0.153	14.118	3.139	0.118	0.146	2.1	5.048	2.42	0.066	98.443	VDL2-1 8
13 / 1 .	70.996	0.191	14.171	3.202	0.118	0.128	2.1	5.23	2.454	0.051	98.64	VDL2-1 10
14 / 1 .	70.328	0.136	14.109	2.961	0.108	0.132	1.953	5.155	2.419	0.054	97.356	VDL2-1 11
15 / 1 .	66.528	0.408	14.791	5.259	0.143	0.382	3.165	5.034	2.165	0.098	97.973	VDL2-1 12
16 / 1 .	70.736	0.17	14.118	3.101	0.115	0.156	1.903	5.267	2.416	0	97.982	VDL2-1 13
17 / 1 .	69.581	0.135	14.135	2.906	0.08	0.109	2.012	5.133	2.522	0.035	96.649	VDL2-1 14
18 / 1 .	70.129	0.147	14.102	2.999	0.089	0.132	2.023	5.218	2.291	0.028	97.157	VDL2-1 15
19 / 1 .	70.568	0.179	14.02	3.099	0.073	0.106	2.006	5.391	2.463	0	97.905	VDL2-1 16
20 / 1 .	71.127	0.174	14.347	3.194	0.07	0.154	2.012	5.303	2.319	0.025	98.727	VDL2-1 17
21 / 1 .	35.34	0.027	1.849	27.676	0.493	0.058	32.505	0.049	0	0.317	98.314	andradite
22 / 1 .	37.172	0	1.77	27.429	0.489	0.064	6.72	0.035	0	0.306	73.984	andradite
23 / 1 .	0	0	0	0	0	0	0	0	0	0	0	andradite
24 / 1 .	34.691	0.008	1.618	26.838	0.507	0.048	31.538	0	0	0.331	95.58	andradite
25 / 1 .	34.862	0.032	1.65	27.291	0.46	0.076	31.497	0.022	0.009	0.343	96.241	andradite
26 / 1 .	34.911	0.071	1.882	27.67	0.555	0.078	31.447	0.03	0.009	0.326	96.979	andradite
27 / 1 .	51.225	0	0.023	0.373	0.065	0.194	47.02	0.014	0	0.496	99.41	wollastonite
28 / 1 .	36.344	0.031	1.73	27.09	0.545	0.085	32.603	0	0.05	0.341	98.818	andradite
29 / 1 .	35.471	0.039	1.598	27.251	0.427	0.073	31.979	0	0	0.417	97.255	andradite
30 / 1 .	35.738	0.032	1.8	27.684	0.454	0.102	32.409	0	0.006	0.276	98.501	andradite
31 / 1 .	73.907	0.106	13	1.858	0.093	0.024	1.344	4.978	2.616	0	97.925	VDL2-2 1
32 / 1 .	73.781	0.086	12.987	2.104	0.112	0.063	1.328	5.223	2.597	0.01	98.29	VDL2-2 2
33 / 1 .	73.899	0.075	13.002	1.925	0.144	0.025	1.363	4.847	2.61	0.017	97.909	VDL2-2 3
34 / 1 .	73.977	0.083	13.37	2.033	0.067	0.003	1.367	5.067	2.471	0.002	98.441	VDL2-2 4
35 / 1 .	73.294	0.08	13.419	2.126	0.089	0.032	1.252	5.061	2.442	0	97.795	VDL2-2 5
36 / 1 .	73.746	0.1	13.222	1.985	0.105	0.032	1.264	5.088	2.487	0.068	98.095	VDL2-2 6
37 / 1 .	74.18	0.076	13.212	2.173	0.095	0.037	1.341	5.195	2.441	0.031	98.782	VDL2-2 7
38 / 1 .	72.544	0.102	12.821	1.957	0.086	0.059	1.294	5.11	2.523	0.004	96.5	VDL2-2 8
40 / 1 .	74.134	0.09	12.963	2.036	0.105	0.032	1.305	4.881	2.425	0.034	98.006	VDL2-2 10
41 / 1 .	73.105	0.121	13.499	1.938	0.057	0.019	1.287	4.81	2.63	0	97.466	VDL2-2 11
42 / 1 .	73.349	0.049	12.951	2.001	0.099	0.022	1.314	4.911	2.655	0.065	97.416	VDL2-2 12
43 / 1 .	35.54	0.042	1.641	27.576	0.453	0.036	32.338	0.027	0.026	0.389	98.069	andradite
44 / 1 .	70.123	0.196	13.652	2.981	0.083	0.124	1.954	5.215	2.289	0.06	96.676	VDL10-1 1
45 / 1 .	71.072	0.149	14.102	2.981	0.089	0.15	2	5.09	2.221	0.006	97.861	VDL10-1 2
46 / 1 .	71.743	0.152	14.081	3.177	0.092	0.135	1.987	5.258	2.212	0	98.837	VDL10-1 3
47 / 1 .	70.729	0.192	13.861	3.24	0.124	0.128	1.95	5.316	2.435	0.034	98.008	VDL10-1 4
48 / 1 .	71.007	0.125	13.975	2.978	0.165	0.126	1.909	5.386	2.232	0	97.903	VDL10-1 5
49 / 1 .	69.963	0.259	14.622	3.743	0.143	0.157	2.411	5.022	2.113	0.09	98.523	VDL10-1 6
51 / 1 .	71.6	0.189	14.227	3.19	0.111	0.158	2.068	5.259	2.261	0	99.063	VDL10-1 8
53 / 1 .	71.143	0.164	14.249	2.856	0.21	0.121	1.98	4.961	2.334	0.06	98.078	VDL10-1 10
54 / 1 .	70.865	0.141	13.656	2.979	0.057	0.127	2.089	4.97	2.39	0.073	97.347	VDL10-1 11
56 / 1 .	35.516	0.035	1.904	27.51	0.555	0.103	32.285	0.03	0	0.362	98.298	andradite
57 / 1 .	35.679	0.011	1.667	27.464	0.425	0.042	32.149	0.027	0.035	0.329	97.828	andradite
58 / 1 .	61.029	0.023	24.262	0.174	0	0.007	5.857	8.777	0.398	0.058	100.585	VDL15-3 1
60 / 1 .	73.364	0.101	13.131	1.961	0.032	0.039	1.31	4.624	2.704	0.025	97.292	VDL15-3 3
61 / 1 .	73.884	0.057	13.392	1.886	0.083	0.018	1.29	4.853	2.547	0.061	98.071	VDL15-3 4
63 / 1 .	74.077	0.068	13.175	1.882	0.08	0	1.296	4.927	2.654	0.014	98.173	VDL15-3 6
65 / 1 .	73.285	0.106	13.32	1.86	0.099	0.046	1.256	4.739	2.73	0	97.44	VDL15-3 8
67 / 1 .	72.916	0.086	12.63	1.912	0.112	0.01	1.311	4.928	2.488	0	96.393	VDL15-3 10
68 / 1 .	73.371	0.077	13.117	1.988	0.067	0.037	1.256	4.89	2.687	0	97.49	VDL15-3 11
69 / 1 .	73.125	0.091	13.535	2.105	0.096	0.027	1.328	4.868	2.791	0	97.966	VDL15-3 12
70 / 1 .	35.857	0.046	1.673	27.669	0.485	0.042	32.248	0.012	0.029	0.323	98.383	andradite
71 / 1 .	35.83	0.058	1.648	27.497	0.386	0.068	32.303	0	0.041	0.378	98.208	andradite
72 / 1 .	35.9	0.019	1.801	27.644	0.485	0.029	32.232	0.062	0.07	0.358	98.6	andradite
73 / 1 .	70.662	0.137	13.792	3.12	0.093	0.153	1.951	5.127	2.359	0.007	97.4	VDL16-3 1
75 / 1 .	70.484	0.181	14.241	2.894	0.102	0.114	2.065	5.179	2.401	0	97.66	VDL16-3 3
76 / 1 .	70.418	0.144	14.166	2.882	0.131	0.143	1.963	5.263	2.405	0.049	97.562	VDL16-3 4
77 / 1 .	70.595	0.143	13.923	3.167	0.013	0.16	2.011	5.019	2.45	0	97.481	VDL16-3 5
78 / 1 .	70.972	0.179	14.052	3.28	0.143	0.144	2.077	5.182	2.256	0	98.284	VDL16-3 6
79 / 1 .	71.457	0.16	13.99	2.941	0.153	0.132	1.962	5.021	2.361	0.1	98.277	VDL16-3 7
80 / 1 .	70.664	0.154	13.861	3.019	0.105	0.118	2.013	4.936	2.433	0.063	97.367	VDL16-3 8
81 / 1 .	71.63	0.167	14.113	3.034	0.112	0.153	1.948	4.719	2.404	0.09	98.372	VDL16-3 9
82 / 1 .	70.111	0.188	14.491	3.67	0.118	0.148	2.386	4.932	2.292	0.048	98.385	VDL16-3 10
83 / 1 .	70.607	0.146	13.791	3.218	0.083	0.126	1.973	4.978	2.23	0.034	97.186	VDL16-3 11
84 / 1 .	35.62	0.06	1.578	27.689	0.504	0.081	32.335	0.01	0.006	0.343	98.225	andradite

Oxide	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
85 / 1 .	71.025	0.173	14.429	2.972	0.163	0.151	1.861	5.068	2.451	0	98.293	VDL15-4 1
87 / 1 .	71.281	0.201	14.057	3.012	0.118	0.148	2.066	5.287	2.384	0.074	98.629	VDL15-4 3
89 / 1 .	70.652	0.168	14.106	3.032	0.131	0.108	1.95	4.912	2.405	0.009	97.472	VDL15-4 5
90 / 1 .	70.613	0.196	14.134	3.041	0.109	0.148	1.974	4.919	2.424	0.068	97.626	VDL15-4 6
92 / 1 .	71.643	0.155	14.152	3.193	0.144	0.135	2.122	4.9	2.359	0	98.803	VDL15-4 8
93 / 1 .	71.691	0.166	14.328	3.001	0.131	0.123	1.92	4.998	2.469	0.032	98.86	VDL15-4 9
94 / 1 .	70.943	0.158	14.172	2.952	0.077	0.107	2.042	4.706	2.379	0.042	97.578	VDL15-4 10
95 / 1 .	70.726	0.174	13.709	3.104	0.128	0.131	2.016	4.83	2.549	0.038	97.403	VDL15-4 11
96 / 1 .	70.859	0.172	14.144	3.232	0.048	0.122	2.054	5.087	2.376	0.023	98.118	VDL15-4 12
99 / 1 .	70.948	0.178	13.914	3.069	0.089	0.115	1.972	4.898	2.565	0.035	97.783	VDL15-4 15
100 / 1 .	35.453	0.012	1.798	26.868	0.458	0.041	32.263	0.027	0.009	0.301	97.231	andradite
101 / 1 .	36.068	0	1.704	27.086	0.44	0.032	32.511	0	0.026	0.335	98.202	andradite
102 / 1 .	34.39	0.045	1.669	27.322	0.415	0.034	32.259	0.032	0.006	0.331	96.503	andradite
103 / 1 .	35.918	0.036	1.763	27.43	0.43	0.071	32.413	0	0	0.379	98.44	andradite
104 / 1 .	35.73	0	1.783	27.647	0.405	0.055	31.895	0.067	0	0.32	97.903	andradite
105 / 1 .	35.751	0.021	1.518	27.366	0.392	0.058	32.303	0.022	0	0.269	97.7	andradite 23/6
106 / 1 .	35.624	0.029	1.666	27.155	0.485	0.046	32.201	0	0	0.276	97.482	andradite 23/6
107 / 1 .	71.773	0.198	13.96	2.992	0.153	0.14	1.978	4.846	2.501	0.026	98.567	VDL5-1
109 / 1 .	71.759	0.147	13.684	3.038	0.185	0.17	1.979	5.157	2.392	0.103	98.613	VDL5-1 3
110 / 1 .	70.882	0.175	13.98	3.042	0.112	0.164	1.978	5.124	2.339	0.085	97.88	VDL5-1 4
112 / 1 .	71.027	0.143	13.409	2.944	0.147	0.138	2.032	4.857	2.416	0.052	97.166	VDL5-1 6
113 / 1 .	72.416	0.188	14.168	3.057	0.111	0.115	1.928	4.775	2.492	0	99.25	VDL5-1 7
114 / 1 .	71.813	0.213	14.034	3.009	0.157	0.167	1.996	5.032	2.627	0.036	99.084	VDL5-1 8
116 / 1 .	71.245	0.159	13.664	3.065	0.144	0.142	2.013	5.02	2.541	0.075	98.068	VDL5-1 10
117 / 1 .	69.943	0.193	13.847	3.05	0.102	0.137	1.868	4.798	2.453	0.02	96.408	VDL5-1 11
118 / 1 .	72.183	0.233	14.02	3.091	0.08	0.126	1.93	4.964	2.547	0.041	99.215	VDL5-1 12
119 / 1 .	70.889	0.206	13.581	2.743	0.013	0.113	1.808	4.965	2.511	0	96.83	VDL5-1 13
120 / 1 .	72.166	0.138	14.455	3.002	0.125	0.161	2.078	5.285	2.514	0.039	99.964	VDL5-1 14
121 / 1 .	70.783	0.178	13.708	3.16	0.125	0.116	1.959	5.073	2.508	0	97.609	VDL5-1 15
122 / 1 .	36.229	0.026	1.627	27.069	0.505	0.076	32.354	0	0.012	0.346	98.244	andradite
123 / 1 .	36.084	0.059	1.694	27.3	0.509	0.018	32.314	0.062	0	0.356	98.396	andradite
124 / 1 .	75.688	0.069	13.146	1.83	0.08	0.042	1.382	4.806	2.907	0.023	99.975	VDL5-2 1
125 / 1 .	73.769	0.093	13.029	1.903	0.141	0.044	1.349	4.796	2.79	0.003	97.916	VDL5-2 2
126 / 1 .	73.076	0.094	12.699	2.192	0.054	0.025	1.347	4.883	2.856	0.034	97.261	VDL5-2 3
127 / 1 .	73.613	0.089	13.044	1.984	0.122	0.042	1.291	4.869	2.772	0.035	97.863	VDL5-2 4
128 / 1 .	73.724	0.095	12.784	2.093	0.15	0	1.337	4.948	2.896	0	98.027	VDL5-2 5
129 / 1 .	74.349	0.074	12.737	1.931	0.201	0.02	1.285	5.117	2.667	0.029	98.41	VDL5-2 6
130 / 1 .	74.708	0.089	13.152	1.984	0.061	0.014	1.282	4.869	2.6	0.048	98.807	VDL5-2 7
131 / 1 .	73.75	0.054	12.886	2.05	0.086	0.024	1.392	5.02	3.028	0	98.291	VDL5-2 8
132 / 1 .	73.942	0.069	12.667	1.915	0.054	0.055	1.282	4.925	2.975	0.035	97.919	VDL5-2 9
133 / 1 .	73.734	0.093	12.494	1.957	0.058	0.047	1.327	4.916	2.868	0.016	97.51	VDL5-2 10
134 / 1 .	73.456	0.098	13.025	1.999	0.077	0.064	1.285	4.957	2.833	0	97.794	VDL5-2 11
135 / 1 .	73.698	0.041	12.856	2.059	0.042	0.007	1.309	4.948	2.797	0.009	97.764	VDL5-2 12
136 / 1 .	73.769	0.111	12.572	1.945	0.121	0.007	1.39	5.124	2.727	0.008	97.775	VDL5-2 13
137 / 1 .	35.764	0.02	1.653	27.427	0.445	0.023	32.56	0.015	0.023	0.261	98.192	andradite
139 / 1 .	71.712	0.179	13.619	3.016	0.163	0.128	1.953	5.065	2.463	0.013	98.309	VDL16-1 2
140 / 1 .	53.994	0.085	27.719	0.746	0.102	0.066	11.544	5.452	0.162	0.089	99.959	VDL16-1 3
141 / 1 .	50.29	2.985	13.124	14.664	0.262	4.049	8.333	3.271	0.747	0.586	98.312	VDL16-1 4
142 / 1 .	50.295	3.058	12.768	14.104	0.316	4.415	8.564	3.419	0.64	0.52	98.1	VDL16-1 5
143 / 1 .	50.498	3.133	12.878	14.988	0.297	4.169	8.623	3.17	0.718	0.46	98.935	VDL16-1 6
144 / 1 .	50.297	3.01	12.748	14.716	0.3	4.305	8.486	3.282	0.636	0.541	98.322	VDL16-1 7
146 / 1 .	49.21	3.859	12.283	15.204	0.19	4.754	8.882	2.895	0.788	0.467	98.532	VDL16-1 9
147 / 1 .	47.658	0.05	31.573	0.788	0	0.207	15.919	2.729	0.048	0.125	99.097	VDL16-1 10
151 / 1 .	53.822	0.113	28.386	0.924	0.045	0.139	11.912	4.013	0.213	0.109	99.674	VDL16-1 14
152 / 1 .	70.011	0.308	13.172	3.616	0.098	0.214	1.261	5.828	3.328	0.023	97.86	VDL16-1 15
155 / 1 .	48.527	2.447	13.415	12.768	0.2	6.714	10.862	2.719	0.305	0.347	98.303	VDL16-1 18
156 / 1 .	35.488	0.086	1.706	26.846	0.417	0.057	32.75	0.017	0	0.318	97.685	andradite
157 / 1 .	71.91	0.157	14.005	3.092	0.099	0.14	2.006	5.308	2.42	0.051	99.187	VDL16-1 19
158 / 1 .	48.631	3.128	13.021	12.806	0.188	5.798	10.831	2.695	0.533	0.431	98.063	VDL16-1 20
159 / 1 .	65.873	1.231	13.948	5.59	0.216	1.21	3.13	5.149	2.671	0.246	99.264	VDL16-1 21
164 / 1 .	35.325	0.036	1.579	27.093	0.494	0.059	32.349	0.015	0.081	0.371	97.402	andradite
165 / 1 .	50.114	2.604	13.267	13.149	0.172	5.548	10.008	3.011	0.524	0.355	98.752	VDL16-3 1
166 / 1 .	65.639	0.525	16.029	4.92	0.225	0.461	2.22	5.772	3.895	0.1	99.786	VDL16-3 2
167 / 1 .	65.01	0.491	15.833	4.889	0.228	0.468	2.272	6.01	3.764	0.134	99.098	VDL16-3 3
168 / 1 .	49.949	2.658	13.079	13.197	0.185	5.519	9.961	2.932	0.493	0.394	98.366	VDL16-3 4
169 / 1 .	65.698	0.535	15.613	4.963	0.212	0.466	2.349	5.816	4.031	0.083	99.767	VDL16-3 5
171 / 1 .	64.816	0.506	15.521	4.992	0.232	0.463	2.281	5.856	3.788	0.112	98.566	VDL16-3 7
173 / 1 .	65.477	0.534	16.096	4.963	0.133	0.45	2.217	5.414	3.844	0.093	99.22	VDL16-3 9
174 / 1 .	65.285	0.493	15.746	5.1	0.159	0.507	2.326	5.457	3.637	0.142	98.852	VDL16-3 10
175 / 1 .	49.61	1.91	14.505	11.341	0.183	7.261	11.933	2.575	0.283	0.267	99.867	VDL16-3 11
176 / 1 .	65.71	0.467	16.032	5.022	0.253	0.428	2.113	5.271	3.71	0.095	99.1	VDL16-3 12

Oxide	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
178 / 1 .	66.074	0.494	16.131	4.773	0.207	0.449	2.188	5.529	3.703	0.092	99.64	VDL16-3 14
180 / 1 .	65.023	0.506	15.902	5.181	0.168	0.481	2.297	5.471	3.761	0.154	98.944	VDL16-3 16
181 / 1 .	35.331	0.043	1.661	26.904	0.412	0.075	32.458	0.015	0.053	0.296	97.247	andradite
182 / 1 .	35.949	0.003	1.804	27.296	0.372	0.065	32.497	0.017	0.015	0.324	98.342	andradite
183 / 1 .	49.623	2.655	12.857	12.989	0.173	5.868	10.207	2.853	0.409	0.395	98.029	HDR-1A 1
184 / 1 .	49.769	1.857	13.427	13.332	0.295	6.631	11.295	2.631	0.196	0.271	99.705	HDR-1A 2
185 / 1 .	49.037	2.983	13.035	14.163	0.179	5.272	9.635	2.86	0.516	0.383	98.062	HDR-1A 3
186 / 1 .	49.291	2.919	12.677	14.144	0.169	5.253	9.479	3.09	0.488	0.407	97.917	HDR-1A 4
187 / 1 .	49.241	2.644	13.413	12.991	0.195	5.905	10.544	2.825	0.428	0.348	98.534	HDR-1A 5
188 / 1 .	49.068	1.807	13.21	13.024	0.258	6.51	11.234	2.664	0.26	0.29	98.324	HDR-1A 6
189 / 1 .	49.847	1.833	13.464	12.676	0.208	6.408	11.21	2.757	0.238	0.285	98.926	HDR-1A 7
190 / 1 .	49.376	2.483	13.516	12.556	0.236	6.338	10.892	2.738	0.402	0.468	99.006	HDR-1A 8
191 / 1 .	49.286	1.845	14.094	11.498	0.224	7.436	12.071	2.467	0.276	0.244	99.44	HDR-1A 9
192 / 1 .	50.03	2.958	12.854	13.952	0.21	5.354	9.604	3.166	0.475	0.395	98.999	HDR-1A 10
193 / 1 .	49.083	2.956	13.205	13.938	0.295	5.184	9.606	3.138	0.488	0.396	98.289	HDR-1A 11
194 / 1 .	49.909	2.448	13.393	12.998	0.236	6.102	10.618	2.554	0.368	0.414	99.04	HDR-1A 12
195 / 1 .	35.228	0.018	1.775	27.282	0.396	0.055	32.473	0	0.044	0.267	97.538	andradite
196 / 1 .	49.012	2.54	13.914	12.665	0.211	5.807	10.54	2.87	0.437	0.424	98.421	HDR-1B 1
197 / 1 .	49.529	3.333	12.412	15.288	0.282	4.732	8.994	3.259	0.579	0.432	98.839	HDR-1B 2
198 / 1 .	50.396	2.994	13.156	13.74	0.301	5.515	9.808	2.94	0.44	0.445	99.734	HDR-1B 3
200 / 1 .	71.357	0.197	12.956	3.286	0.102	0.029	0.99	5.548	3.305	0.018	97.787	HDR-1B 5
201 / 1 .	72.468	0.212	13.5	3.011	0.137	0.039	0.989	5.661	3.363	0	99.381	HDR-1B 6
202 / 1 .	72.514	0.209	13.198	3.117	0.08	0.027	0.956	5.508	3.771	0	99.379	HDR-1B 7
203 / 1 .	71.545	0.216	13.19	3.147	0.029	0.031	0.96	5.48	3.449	0	98.048	HDR-1B 8
204 / 1 .	48.907	2.407	13.235	12.632	0.245	6.384	10.804	2.809	0.326	0.361	98.109	HDR-1B 9
205 / 1 .	50.153	1.57	13.799	11.974	0.252	7.058	11.914	2.564	0.117	0.272	99.672	HDR-1B 10
206 / 1 .	71.253	0.221	12.711	3.236	0.131	0.041	0.965	5.538	3.224	0.035	97.355	HDR-1B 11
207 / 1 .	49.459	1.586	13.382	12.095	0.217	7.224	12.221	2.451	0.177	0.287	99.1	HDR-1B 12
208 / 1 .	49.716	1.811	13.515	13.078	0.308	6.695	11.124	2.595	0.25	0.28	99.371	HDR-1B 13
209 / 1 .	74.904	0.279	12.555	2.728	0.172	0.212	1.712	4.284	2.477	0.039	99.364	HDR-1B 14
210 / 1 .	71.926	0.18	13.153	3.153	0.105	0.038	1.017	5.37	3.519	0.01	98.471	HDR-1B 15
211 / 1 .	71.811	0.218	13.101	3.232	0.057	0.012	0.993	5.439	3.311	0.007	98.181	HDR-1B 16
212 / 1 .	48.573	2.295	13.46	11.629	0.217	6.845	11.41	2.52	0.253	0.32	97.521	HDR-1B 17
213 / 1 .	49.121	2.49	13.251	12.717	0.166	6.296	10.724	2.958	0.313	0.308	98.345	HDR-1B 18
214 / 1 .	49.123	1.59	13.812	12.371	0.189	7.412	11.908	2.416	0.259	0.284	99.365	HDR-1B 19
215 / 1 .	50.11	2.664	13.37	13.391	0.307	5.949	10.268	2.952	0.395	0.346	99.752	HDR-1B 20
216 / 1 .	49.86	2.523	13.422	13.148	0.191	5.812	10.281	2.727	0.367	0.308	98.639	HDR-1B 21
217 / 1 .	71.191	0.246	12.858	3.224	0.048	0	1.028	5.205	3.344	0	97.143	HDR-1B 22
218 / 1 .	71.976	0.266	13.374	3.293	0.118	0.014	1.024	5.319	3.293	0.013	98.689	HDR-1B 23
219 / 1 .	49.35	1.509	13.545	11.769	0.205	7.576	12.332	2.342	0.177	0.237	99.043	HDR-1B 24
220 / 1 .	49.645	2.641	13.087	13.434	0.198	5.229	9.765	3.183	0.431	0.366	97.978	HDR-1B 25
221 / 1 .	35.474	0.039	1.727	27.189	0.451	0.031	32.396	0.044	0	0.318	97.669	andradite

DataSet/Pt	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
5 / 1 .	48.834	2.211	13.454	11.365	0.252	6.933	11.216	2.936	0.312	0.189	97.702	get2-1 1
14 / 1 .	48.388	2.013	13.664	11.694	0.256	6.701	11.227	2.805	0.34	0.241	97.329	get2-1 10
15 / 1 .	49.077	2.071	13.37	11.788	0.173	6.729	10.916	2.68	0.348	0.08	97.232	get2-1 11
16 / 1 .	48.645	2.4	13.735	11.828	0.197	6.738	11.154	2.843	0.346	0.188	98.075	get2-1 12
17 / 1 .	48.499	2.193	13.867	11.846	0.146	6.675	11.005	2.857	0.357	0.159	97.604	get2-1 13
6 / 1 .	48.793	2.241	13.583	11.689	0.23	6.743	11.273	2.791	0.303	0.312	97.959	get2-1 2
7 / 1 .	48.609	2.152	13.623	11.931	0.263	6.723	11.116	2.949	0.279	0.182	97.826	get2-1 3
8 / 1 .	48.781	2.207	13.934	11.842	0.16	6.841	11.489	2.748	0.334	0.211	98.547	get2-1 4
9 / 1 .	49.451	2.013	14.073	12.022	0.24	6.92	11.154	2.717	0.336	0.175	99.101	get2-1 5
10 / 1 .	48.608	2.203	13.686	11.708	0.259	6.772	11.304	2.742	0.318	0.219	97.819	get2-1 6
12 / 1 .	49.005	2.073	13.651	11.938	0.164	6.773	11.202	2.689	0.331	0.19	98.016	get2-1 8
13 / 1 .	49.09	2.178	13.922	11.743	0.228	6.83	11.45	2.779	0.331	0.073	98.622	get2-1 9
108 / 1 .	69.582	0.022	14.081	0.063	0.02	0.083	4.389	0.01	0	0.031	88.28	get27-3 1
109 / 1 .	71.22	0.287	13.179	4.377	0.213	0.136	1.319	5.689	3.33	0.069	99.818	get27-3 2
110 / 1 .	61.177	0	14.721	0.004	0.023	0.002	4.803	0.061	0.004	0	80.795	get27-3 3
99 / 1 .	61.559	0.753	13.894	9.296	0.244	0.686	4.166	4.688	1.744	0.227	97.256	get27-4 1
105 / 1 .	70.412	0.152	13.92	3.45	0.172	0.04	2.271	5.27	2.532	0	98.219	get27-4 11-12
105 / 3 .	63.602	0.634	14.751	8.497	0.312	0.657	4.546	5.116	1.391	0.114	99.622	get27-4 11-12
106 / 2 .	67.487	0.272	13.775	5.29	0.255	0.097	2.709	5.248	2.056	0	97.187	get27-4 13-14
106 / 1 .	62.12	0.545	16.518	7.424	0.279	0.645	5.316	5.217	1.279	0.197	99.54	get27-4 13-14
100 / 1 .	61.818	0.553	14.146	9.156	0.276	0.606	3.92	4.538	1.663	0.099	96.773	get27-4 2
101 / 2 .	69.796	0.058	12.351	1.893	0.085	0.006	1.206	4.298	2.604	0.063	92.358	get27-4 3-4
101 / 1 .	60.707	0.763	14.282	9.741	0.298	0.88	4.15	4.139	1.982	0.355	97.295	get27-4 3-4
102 / 1 .	60.756	0.836	14.626	9.609	0.304	0.917	4.694	4.556	1.678	0.173	98.149	get27-4 5-6
102 / 2 .	62.415	0.801	14.657	9.068	0.288	0.741	4.474	4.451	1.71	0.06	98.665	get27-4 5-6
103 / 2 .	61.087	0.8	14.355	9.522	0.218	0.948	4.668	4.792	1.569	0.302	98.262	get27-4 7-8
104 / 2 .	48.297	1.759	13.227	12.05	0.265	7.176	11.779	2.483	0.165	0.159	97.36	get27-4 9-11
104 / 1 .	62.166	0.8	14.597	8.946	0.281	0.79	4.359	4.904	1.676	0.241	98.76	get27-4 9-11
46 / 1 .	49.543	1.731	13.512	12.589	0.249	6.94	11.811	2.612	0.2	0.196	99.385	get30-1 1
55 / 1 .	49.905	1.845	13.298	12.341	0.195	6.579	11.306	2.474	0.229	0.24	98.412	get30-1 10
56 / 1 .	50.076	2.651	13.35	13.279	0.229	5.661	10.169	2.904	0.438	0.363	99.121	get30-1 11
57 / 1 .	49.431	1.695	13.747	11.946	0.163	7.111	11.92	2.315	0.168	0	98.497	get30-1 12
59 / 1 .	49.206	1.875	13.662	11.807	0.243	7.157	11.351	2.446	0.219	0.146	98.112	get30-1 14
47 / 1 .	48.982	1.597	13.93	12.034	0.252	7.221	11.544	2.375	0.21	0.24	98.384	get30-1 2
49 / 1 .	50.108	1.908	13.699	12.467	0.262	6.533	11.104	2.605	0.242	0.175	99.103	get30-1 4
50 / 1 .	49.227	1.743	13.646	12.216	0.163	7.151	11.434	2.444	0.192	0.197	98.414	get30-1 5
51 / 1 .	49.411	1.798	13.607	12.13	0.233	7.146	11.512	2.334	0.185	0.138	98.494	get30-1 6
52 / 1 .	49.309	1.642	13.624	12.026	0.268	6.848	11.453	2.523	0.176	0.211	98.08	get30-1 7
53 / 1 .	49.09	1.874	13.311	12.208	0.077	6.972	11.755	2.404	0.182	0.211	98.084	get30-1 8
54 / 1 .	48.67	1.415	14.089	10.604	0.199	7.988	12.349	2.22	0.115	0.182	97.83	get30-1 9
19 / 1 .	72.911	0.047	12.118	1.737	0.084	0.044	1.219	4.717	2.749	0	95.628	get6-1 1
29 / 1 .	73.703	0.054	13.256	1.917	0.023	0.032	1.353	5.044	2.817	0	98.199	get6-1 10
31 / 1 .	62.173	0.627	14.48	8.934	0.359	0.656	4.229	4.81	1.675	0.181	98.124	get6-1 12
127 / 1 .	61.866	0.687	14.356	9.128	0.304	0.848	4.243	4.601	1.656	0.279	97.969	get6-1 13-14
128 / 1 .	64.33	0.673	14.442	7.429	0.286	0.369	3.77	5.083	1.785	0.038	98.204	get6-1 15-16
128 / 2 .	62.062	0.762	14.255	9.02	0.269	0.864	4.373	4.689	1.78	0.249	98.322	get6-1 15-16
129 / 1 .	60.636	1.106	14.114	9.629	0.17	1.58	4.852	4.146	1.669	0.518	98.419	get6-1 17-18
130 / 2 .	72.07	0.155	12.899	2.058	0.081	0.08	1.219	4.432	2.876	0.016	95.885	get6-1 19-20
130 / 1 .	72.691	0.144	12.635	1.798	0.088	0.019	1.324	5.181	2.767	0	96.647	get6-1 19-20
20 / 1 .	73.957	0.065	12.443	1.774	0.023	0.053	1.22	4.63	2.886	0	97.051	get6-1 1a
22 / 1 .	62.522	0.544	13.924	8.818	0.282	0.554	4.288	5.019	1.81	0.076	97.836	get6-1 3
23 / 1 .	45.943	3.912	13.354	13.551	0.162	5.849	10.33	3.332	0.568	0.388	97.391	get6-1 4
24 / 1 .	57.981	1.147	14.91	9.524	0.224	1.733	5.32	4.647	1.444	0.673	97.604	get6-1 5
26 / 1 .	45.065	3.068	14.454	14.251	0.213	6.381	10.242	3.02	0.491	0.295	97.48	get6-1 7
61 / 1 .	61.779	0.751	14.458	8.6	0.317	0.89	4.262	5.237	1.709	0.211	98.215	get7-1 1
71 / 1 .	60.963	0.779	14.315	8.967	0.28	0.795	4.308	4.659	1.727	0.174	96.967	get7-1 11
73 / 1 .	60.79	0.835	14.501	9.55	0.321	1.046	4.803	4.566	1.591	0.098	98.1	get7-1 13
74 / 1 .	61.313	0.722	13.943	9.053	0.244	0.701	4.343	4.752	1.783	0.159	97.015	get7-1 14
62 / 1 .	62.367	0.775	14.379	8.737	0.26	0.86	4.357	4.961	1.805	0.227	98.728	get7-1 2
63 / 1 .	73.721	0.083	12.438	1.53	0	0.042	1.303	4.734	2.757	0.063	96.671	get7-1 3
64 / 1 .	65.37	0.462	13.983	6.674	0.294	0.277	3.436	4.997	1.94	0	97.432	get7-1 4
65 / 1 .	61.873	0.765	14.804	8.815	0.349	0.817	4.309	4.713	1.546	0.204	98.194	get7-1 5
66 / 1 .	60.837	0.974	14.51	9.195	0.247	0.966	4.507	4.663	1.595	0.242	97.738	get7-1 6
67 / 1 .	63.71	0.572	14.703	7.562	0.244	0.602	3.838	4.841	1.912	0.235	98.217	get7-1 7
68 / 1 .	63.507	1.396	13.897	6.368	0.187	1.488	3.427	4.841	2.564	0.266	97.94	get7-1 8
69 / 1 .	59.287	1.236	15.128	9.782	0.302	1.761	5.158	4.346	1.434	0.511	98.945	get7-1 9
33 / 1 .	46.657	4.497	12.509	14.572	0.178	5.22	9.554	3.261	0.8	0.496	97.745	get7-2 1
42 / 1 .	46.076	4.411	12.582	14.722	0.273	5.031	9.55	3.384	0.784	0.845	97.659	get7-2 10
43 / 1 .	46.541	4.384	12.444	14.68	0.216	5.052	9.335	3.368	0.771	0.546	97.336	get7-2 11
44 / 1 .	47.375	4.505	12.58	14.637	0.266	4.892	9.528	3.254	0.734	0.531	98.303	get7-2 12
34 / 1 .	46.482	4.608	12.541	14.838	0.289	4.889	9.301	3.335	0.78	0.581	97.645	get7-2 2
35 / 1 .	48.774	2.959	12.81	13.504	0.197	5.492	9.528	3.03	0.432	0.232	96.959	get7-2 3

DataSet/Pt	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
36 / 1 .	46.968	4.539	12.764	14.67	0.172	4.921	9.086	3.17	0.789	0.533	97.612	get7-2 4
37 / 1 .	47.056	4.634	12.782	14.793	0.105	4.938	9.443	3.366	0.736	0.524	98.377	get7-2 5
38 / 1 .	47.034	4.667	12.399	14.409	0.242	5.051	9.251	3.136	0.795	0.584	97.569	get7-2 6
39 / 1 .	46.725	4.642	12.392	14.798	0.26	5.009	9.225	3.215	0.758	0.409	97.435	get7-2 7
40 / 1 .	47.234	4.2	12.606	13.758	0.276	4.517	8.813	3.466	0.896	0.627	96.393	get7-2 8
78 / 1 .	48.88	2.478	13.205	12.013	0.188	6.736	11.045	2.787	0.312	0.312	97.955	get8-1 1
84 / 1 .	48.71	2.897	12.87	13.281	0.255	5.851	10.195	2.965	0.412	0.32	97.756	get8-1 10-11
84 / 2 .	49.205	2.41	13.263	12.23	0.272	6.696	10.983	2.793	0.345	0.204	98.4	get8-1 10-11
85 / 1 .	48.727	2.28	13.531	12.275	0.173	6.532	10.791	2.873	0.34	0.073	97.595	get8-1 12-13
85 / 2 .	48.882	2.883	13.075	13.372	0.278	5.915	10.373	2.819	0.401	0.283	98.281	get8-1 12-13
86 / 1 .	48.948	2.413	13.171	12.365	0.137	6.603	10.989	2.763	0.34	0.189	97.919	get8-1 14-15
86 / 2 .	49.112	2.332	13.29	12.27	0.195	6.777	11.188	2.69	0.32	0.182	98.356	get8-1 14-15
79 / 1 .	49.399	2.373	13.316	12.242	0.246	6.758	10.862	2.895	0.337	0.277	98.704	get8-1 2
80 / 1 .	49.055	2.574	13.488	12.256	0.192	6.433	10.921	2.986	0.349	0.233	98.486	get8-1 3
81 / 1 .	49.117	2.325	13.004	12.512	0.23	6.419	10.52	2.852	0.378	0.269	97.628	get8-1 4-5
81 / 2 .	48.263	2.487	13.449	12.048	0.217	6.686	11.187	2.864	0.332	0.298	97.83	get8-1 4-5
82 / 1 .	48.943	2.881	13.011	12.921	0.268	5.801	10.185	2.795	0.393	0.24	97.436	get8-1 6-7
82 / 2 .	48.904	2.545	13.765	12.305	0.154	6.64	11.175	2.845	0.332	0.218	98.882	get8-1 6-7
83 / 2 .	49.083	2.505	13.206	12.768	0.195	6.279	10.569	2.856	0.379	0.335	98.176	get8-1 8-9
93 / 2 .	49.783	1.819	13.235	13.054	0.271	6.537	11.428	2.596	0.234	0.175	99.131	get9-1 11-12
94 / 1 .	48.248	3.434	12.505	14.782	0.241	5.148	9.645	2.922	0.448	0.354	97.728	get9-1 13-14
96 / 1 .	48.259	2.63	13.121	12.652	0.192	6.673	11.182	2.75	0.301	0.203	97.963	get9-1 16
90 / 1 .	48.865	1.425	13.968	10.562	0.189	8.188	12.698	2.23	0.167	0.051	98.343	get9-1 5-6
91 / 1 .	49.519	2.621	12.943	12.812	0.313	6.073	10.294	2.952	0.38	0.269	98.176	get9-1 7-8
91 / 2 .	48.716	1.54	14.098	11.198	0.243	7.448	12.082	2.547	0.177	0.175	98.224	get9-1 7-8
92 / 1 .	48.349	2.047	13.759	11.798	0.233	7.083	11.477	2.801	0.264	0.218	98.028	get9-1 9-10
92 / 2 .	49.411	2.291	13.318	12.776	0.204	6.129	10.606	2.774	0.352	0.335	98.197	get9-1 9-10
113 / 2 .	48.526	2.435	12.85	12.384	0.233	6.19	10.49	2.767	0.335	0.233	96.444	hof5-B 1
113 / 1 .	49.276	2.323	13.342	12.068	0.24	6.556	10.875	2.561	0.342	0.204	97.786	hof5-B 1
112 / 1 .	49.462	2.322	13.475	12.288	0.169	6.84	11.257	2.772	0.35	0.233	99.168	hof5-B 1
117 / 2 .	49.633	2.052	13.289	11.256	0.202	6.122	10.673	2.457	0.332	0.212	96.228	hof5-B 10-11
117 / 1 .	49.792	2.347	13.216	12.02	0.214	6.507	11.146	2.792	0.324	0.197	98.554	hof5-B 10-11
118 / 1 .	45.426	2.061	12.155	10.736	0.15	6.052	9.886	2.602	0.299	0.219	89.586	hof5-B 12-13
118 / 2 .	48.928	2.219	13.091	12.166	0.211	6.514	11.14	2.551	0.321	0.24	97.381	hof5-B 12-13
114 / 2 .	47.86	2.461	13.455	11.875	0.227	6.561	11.373	2.844	0.306	0.254	97.215	hof5-B 4-5
114 / 1 .	49.524	2.435	13.555	12.125	0.192	6.703	11.192	2.664	0.392	0.211	98.993	hof5-B 4-5
115 / 2 .	48.913	2.438	13.502	11.928	0.144	6.644	11.419	2.655	0.344	0.247	98.235	hof5-B 6-7
115 / 1 .	49.352	2.581	13.363	12.028	0.224	6.717	10.805	2.786	0.339	0.342	98.537	hof5-B 6-7
116 / 1 .	49.303	2.566	13.363	12.464	0.214	3.716	10.828	1.23	0.377	0.138	94.198	hof5-B 8-9
116 / 2 .	49.312	2.356	13.339	12.207	0.265	6.826	11.229	2.612	0.345	0.174	98.666	hof5-B 8-9
125 / 1 .	48.839	2.458	12.638	12.457	0.185	6.484	10.625	2.827	0.323	0.189	97.024	lon3-5 11-12
125 / 2 .	49.335	2.442	13.052	12.394	0.246	6.373	10.846	2.685	0.355	0.305	98.032	lon3-5 11-12
120 / 2 .	48.582	2.454	13.327	12.423	0.262	6.385	11.01	2.784	0.384	0.58	98.192	lon3-5 1-2
120 / 1 .	49.332	2.673	13.732	12.235	0.166	6.578	11.34	2.72	0.337	0.29	99.402	lon3-5 1-2
121 / 1 .	49.028	2.173	13.697	11.852	0.214	6.521	11.475	2.73	0.286	0.058	98.034	lon3-5 3-4
121 / 2 .	48.838	2.225	13.488	12.175	0.233	6.682	11.486	2.851	0.332	0.312	98.621	lon3-5 3-4
122 / 2 .	49.148	2.426	13.413	12.086	0.176	6.663	10.984	2.696	0.309	0.233	98.132	lon3-5 5-6
122 / 1 .	48.997	2.192	13.609	12.08	0.198	6.791	11.574	2.797	0.327	0.218	98.783	lon3-5 5-6
123 / 1 .	48.592	2.37	13.598	11.875	0.29	6.717	11.028	2.759	0.32	0.319	97.868	lon3-5 7-8
123 / 2 .	48.733	2.327	13.568	12.257	0.255	6.616	10.923	2.713	0.345	0.254	97.992	lon3-5 7-8
124 / 2 .	48.963	2.401	13.75	11.946	0.185	6.567	10.953	2.603	0.361	0.247	97.977	lon3-5 9-10
124 / 1 .	49.456	2.627	13.187	12.493	0.227	6.254	10.703	2.67	0.303	0.204	98.123	lon3-5 9-10
2 / 2 .	50.508	0.011	0.07	0.346	0.116	0.255	47.765	0.014	0	0.092	99.177	SIK3 Std
2 / 1 .	51.431	0	0.038	0.37	0	0.232	47.793	0	0	0.013	99.878	SIK3 Std
3 / 2 .	51.466	0.008	0.034	0.337	0.116	0.272	47.728	0	0.002	0	99.961	SIK3 Std
3 / 1 .	51.911	0.004	0	0.386	0.05	0.175	47.638	0.017	0	0.092	100.272	SIK3 Std
107 / 2 .	35.58	0.05	1.697	27.386	0.51	0.053	31.702	0.017	0	0	96.995	andradite 00:25
107 / 1 .	35.523	0.004	1.821	27.655	0.384	0.077	32.314	0.002	0.007	0.085	97.871	andradite 00:25
111 / 1 .	35.697	0.057	1.648	27.438	0.384	0.074	32.193	0	0.002	0.072	97.566	andradite 00:45
119 / 2 .	35.308	0.053	1.769	27.014	0.343	0.048	32.064	0.036	0.023	0	96.658	andradite 01:24
119 / 1 .	35.287	0.035	1.776	27.245	0.491	0.045	32.462	0	0.01	0.013	97.364	andradite 01:24
126 / 2 .	35.188	0.057	1.83	27.311	0.276	0.091	32.113	0.002	0.006	0.046	96.92	andradite 01:56
126 / 1 .	35.741	0.021	1.691	27.268	0.471	0.098	32.115	0	0	0.033	97.438	andradite 01:56
132 / 1 .	35.573	0.018	1.86	27.256	0.421	0.065	31.752	0.048	0.021	0	97.014	andradite 02:26
131 / 1 .	35.452	0.089	1.585	27.678	0.378	0.104	32.166	0.017	0.001	0.039	97.507	andradite 02:26
4 / 1 .	35.583	0.036	1.647	26.869	0.461	0.079	32.271	0.007	0	0	96.952	andradite 12:11
18 / 2 .	35.022	0.018	1.795	27.25	0.407	0.06	32.185	0.029	0.011	0.007	96.783	andradite 12:55
18 / 1 .	35.462	0.007	1.74	27.286	0.435	0.088	32.066	0	0	0	97.083	andradite 12:55
32 / 1 .	35.153	0	1.732	27.282	0.386	0.098	32.261	0	0	0.059	96.971	andradite 14:05
45 / 1 .	35.609	0.06	1.868	27.176	0.464	0.076	32.298	0.077	0.025	0.078	97.733	andradite 14:46
60 / 2 .	35.092	0.039	1.734	27.519	0.38	0.092	32.135	0.056	0.014	0.039	97.099	andradite 15:40
60 / 1 .	35.194	0	1.799	27.211	0.443	0.111	32.338	0	0	0.046	97.142	andradite 15:40
75 / 2 .	35.466	0.106	1.751	27.342	0.491	0.078	31.425	0	0.018	0.039	96.716	andradite 16:30

DataSet/Pt	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
75 / 1 .	35.424	0.014	1.703	26.903	0.545	0.057	32.109	0.022	0.002	0	96.779	andradite 16:30
76 / 1 .	35.44	0.043	1.764	27.436	0.453	0.076	31.774	0.01	0	0.059	97.054	andradite 16:35
76 / 2 .	35.68	0.057	1.492	27.057	0.441	0.088	32.241	0.034	0	0	97.09	andradite 16:35
77 / 2 .	35.585	0.078	1.88	27.259	0.444	0.052	31.775	0.031	0.002	0.111	97.218	andradite 20:40
77 / 1 .	35.889	0	1.74	27.148	0.397	0.079	32.717	0.034	0.011	0.046	98.061	andradite 20:40
87 / 1 .	34.987	0.004	1.766	27.162	0.444	0.077	32.329	0	0.008	0	96.775	andradite 21:40
87 / 2 .	35.909	0	1.887	27.21	0.428	0.06	31.967	0.031	0	0	97.492	andradite 21:40
97 / 1 .	35.838	0.064	1.838	27.614	0.384	0.062	31.881	0.068	0.007	0.118	97.873	andradite 22:50
98 / 1 .	35.422	0.057	1.69	27.645	0.49	0.059	32.21	0	0.008	0.026	97.607	andradite 22:55
1 / 4 .	35.072	0.028	1.619	27.032	0.405	0.057	31.936	0.027	0.002	0.013	96.192	Andradite Check
1 / 3 .	34.896	0	1.721	27.495	0.355	0.082	32.189	0	0.019	0	96.757	Andradite Check
1 / 2 .	35.038	0.163	1.746	27.394	0.383	0.094	32.232	0	0	0.007	97.056	Andradite Check
1 / 1 .	35.675	0.057	1.508	26.744	0.44	0.087	32.54	0.051	0	0.033	97.134	Andradite Check

Oxide	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
1 / 1 .	35.61	0.01	1.81	27.36	0.43	0.08	32.32	0.03	0.00	0.34	97.98	
3 / 1 .	72.81	0.08	12.95	2.01	0.10	0.05	1.36	4.52	2.92	0.03	96.83	FGD3b-1
4 / 1 .	60.66	0.91	14.66	9.09	0.28	0.88	4.61	4.90	1.72	0.43	98.12	FGD3b-2
5 / 1 .	58.80	1.14	14.78	9.65	0.31	1.59	5.16	3.91	1.54	0.59	97.47	FGD3b-3
6 / 1 .	60.34	0.78	14.96	8.96	0.25	0.93	4.73	4.61	1.58	0.35	97.49	FGD3b-4
7 / 1 .	60.15	0.88	14.36	9.49	0.26	1.03	4.84	4.68	1.47	0.39	97.55	FGD3b-5
8 / 1 .	60.45	0.81	14.17	9.33	0.30	0.92	4.65	4.56	1.66	0.34	97.19	FGD3b-6
9 / 1 .	60.80	0.85	14.48	9.14	0.29	1.09	5.04	4.98	1.42	0.38	98.45	FGD3b-7
10 / 1 .	61.16	0.80	14.45	9.25	0.29	0.92	4.45	4.51	1.83	0.31	97.97	FGD3b-8
11 / 1 .	69.91	0.14	13.61	2.74	0.11	0.03	1.74	4.48	2.57	0.04	95.36	FGD3b-9
13 / 1 .	59.76	0.88	14.63	9.48	0.25	1.07	4.74	4.63	1.65	0.41	97.50	FGD3b-11
14 / 1 .	59.17	1.13	13.88	9.39	0.34	1.57	4.72	4.48	1.62	0.57	96.85	FGD3b-12
15 / 1 .	67.44	0.26	13.59	4.40	0.16	0.08	2.59	5.26	2.30	0.02	96.11	FGD3b-13
16 / 1 .	69.39	0.17	13.37	3.31	0.14	0.06	2.20	5.24	2.59	0.04	96.50	FGD3b-14
17 / 1 .	60.23	0.90	14.58	9.22	0.26	1.19	4.70	4.60	1.63	0.42	97.72	FGD3b-15
18 / 1 .	60.18	0.82	14.42	9.42	0.28	1.01	4.56	4.57	1.57	0.36	97.19	FGD3b-16
21 / 1 .	59.95	0.83	15.35	8.80	0.23	1.39	5.59	5.20	1.08	0.40	98.82	FGD3b-19
22 / 1 .	34.87	0.02	1.73	27.13	0.39	0.08	31.93	0.04	0.00	0.41	96.59	andradite 31/5 12:31
23 / 1 .	35.35	0.01	1.67	27.05	0.44	0.07	32.10	0.00	0.02	0.35	97.06	andradite 31/5 12:33
24 / 1 .	48.92	1.88	13.21	12.34	0.22	6.77	11.18	2.57	0.21	0.30	97.59	FGD3c-1
25 / 1 .	74.02	0.12	12.34	1.54	0.01	0.08	0.65	4.49	3.81	0.01	97.07	HV3-2 1
26 / 1 .	73.01	0.11	12.38	1.69	0.02	0.10	0.74	4.37	3.94	0.00	96.36	HV3-2 2
29 / 1 .	35.12	0.02	1.83	27.21	0.44	0.07	32.20	0.02	0.00	0.34	97.24	andradite 31/5 14:50
32 / 1 .	73.48	0.10	12.38	1.63	0.04	0.06	0.74	4.54	3.79	0.00	96.74	HV3-2 6
33 / 1 .	73.52	0.09	12.27	1.65	0.07	0.08	0.69	4.45	3.88	0.02	96.71	HV3-2 7
34 / 1 .	73.42	0.10	12.19	1.60	0.04	0.07	0.68	4.50	3.85	0.00	96.46	HV3-2 8
35 / 1 .	73.25	0.12	12.52	1.66	0.03	0.08	0.72	4.54	3.85	0.00	96.76	HV3-2 9
38 / 1 .	73.32	0.12	12.14	1.65	0.01	0.07	0.69	4.36	3.87	0.00	96.22	HV3-2 12
39 / 1 .	73.81	0.10	12.41	1.69	0.06	0.08	0.70	4.45	3.87	0.03	97.21	HV3-2 13
40 / 1 .	70.35	0.15	13.80	3.09	0.11	0.12	2.01	5.11	2.40	0.04	97.17	HV4-2 1
43 / 1 .	70.26	0.18	13.96	3.12	0.10	0.15	2.02	4.49	2.52	0.01	96.79	HV4-2 4
47 / 1 .	70.67	0.16	13.83	2.91	0.15	0.13	1.95	4.95	2.61	0.01	97.36	HV4-2 8
49 / 1 .	70.40	0.18	13.94	2.97	0.08	0.13	1.89	4.92	2.50	0.03	97.05	HV4-2 10
50 / 1 .	70.78	0.17	13.65	2.99	0.07	0.13	1.95	4.83	2.57	0.06	97.19	HV4-2 11
52 / 1 .	69.88	0.17	13.67	3.08	0.11	0.14	1.94	5.01	2.47	0.02	96.47	HV4-2 13
54 / 1 .	70.12	0.15	13.80	3.02	0.11	0.13	2.05	4.77	2.52	0.06	96.73	HV4-2 14
55 / 1 .	68.34	0.13	16.01	2.47	0.08	0.11	2.94	5.55	2.13	0.05	97.80	HV4-2 15
58 / 1 .	35.54	0.03	1.69	27.02	0.40	0.05	32.15	0.02	0.00	0.27	97.16	andradite 31/5 16:35
60 / 1 .	57.05	0.00	25.85	0.28	0.04	0.01	8.32	7.08	0.24	0.10	98.96	HV7-1 2
63 / 1 .	69.28	0.16	14.34	2.83	0.09	0.14	2.18	5.22	2.40	0.03	96.67	HV7-1 5
64 / 1 .	69.75	0.15	13.76	2.98	0.11	0.15	1.97	5.08	2.45	0.05	96.43	HV7-1 6
67 / 1 .	69.43	0.16	13.75	3.02	0.14	0.14	1.99	4.93	2.64	0.03	96.23	HV7-1 9
68 / 1 .	70.27	0.20	13.72	3.14	0.14	0.14	1.95	4.46	2.60	0.06	96.68	HV7-1 10
69 / 1 .	70.18	0.16	13.84	3.01	0.16	0.14	1.97	4.58	2.47	0.01	96.52	HV7-1 11
70 / 1 .	70.48	0.18	13.81	3.19	0.09	0.15	1.97	4.93	2.57	0.02	97.38	HV7-1 12
71 / 1 .	70.51	0.19	13.68	2.97	0.08	0.15	1.93	4.97	2.68	0.01	97.16	HV7-1 13
73 / 1 .	69.20	0.14	15.33	2.68	0.14	0.12	2.59	5.31	2.17	0.03	97.72	HV7-1 15
74 / 1 .	69.82	0.17	14.01	3.08	0.13	0.16	2.02	4.47	2.52	0.03	96.40	HV7-1 16
75 / 1 .	35.01	0.02	1.76	26.97	0.45	0.05	32.13	0.01	0.00	0.30	96.70	andradite 31/5 17:31
76 / 1 .	35.15	0.00	1.77	27.32	0.39	0.07	32.02	0.02	0.00	0.30	97.03	andradite 31/5 17:34
77 / 1 .	73.07	0.07	13.00	2.00	0.08	0.03	1.29	4.49	2.74	0.00	96.78	HV5-6 1
78 / 1 .	72.64	0.11	12.96	2.02	0.11	0.04	1.33	4.84	2.88	0.00	96.92	HV5-6 2
79 / 1 .	72.87	0.08	12.78	2.02	0.11	0.02	1.32	4.87	2.94	0.00	97.01	HV5-6 3
80 / 1 .	71.34	0.05	13.50	1.66	0.08	0.03	1.49	5.43	2.55	0.01	96.12	HV5-6 4
81 / 1 .	57.93	1.07	15.63	8.72	0.20	1.40	5.37	4.96	1.40	0.61	97.29	HV5-6 5
82 / 1 .	72.57	0.07	13.06	2.06	0.08	0.02	1.31	4.93	2.84	0.00	96.95	HV5-6 6
83 / 1 .	72.54	0.08	12.88	1.95	0.08	0.03	1.33	4.77	2.85	0.00	96.51	HV5-6 7
84 / 1 .	72.46	0.07	12.87	2.06	0.10	0.04	1.31	4.83	2.89	0.01	96.64	HV5-6 8
87 / 1 .	72.41	0.09	13.02	2.08	0.14	0.03	1.27	4.53	2.83	0.01	96.40	HV5-6 11
89 / 1 .	59.99	0.85	14.95	8.44	0.22	1.05	5.09	4.47	1.54	0.68	97.27	HV5-6 13
90 / 1 .	72.84	0.11	13.01	1.91	0.09	0.01	1.29	4.54	2.79	0.00	96.60	HV5-6 14
91 / 1 .	35.22	0.01	1.77	27.19	0.46	0.04	32.28	0.02	0.01	0.31	97.31	andradite 31/5 18:24
94 / 1 .	70.96	0.24	13.24	3.23	0.12	0.02	1.00	5.18	3.31	0.00	97.28	HOF1c-1
95 / 1 .	70.76	0.24	13.28	3.29	0.10	0.04	0.98	5.78	3.36	0.00	97.83	HOF1c-2
97 / 1 .	48.11	2.49	13.03	12.25	0.26	6.37	10.81	2.81	0.39	0.40	96.90	HOF1c-4
98 / 1 .	70.18	0.20	13.02	3.31	0.11	0.02	0.99	5.86	3.44	0.00	97.14	HOF1c-5
99 / 1 .	69.55	0.22	12.90	3.25	0.12	0.02	0.97	5.87	3.38	0.02	96.31	HOF1c-6
100 / 1 .	70.03	0.20	13.16	3.25	0.08	0.03	1.04	5.84	3.40	0.01	97.04	HOF1c-7

101 / 1 .	70.62	0.21	12.99	3.20	0.12	0.05	0.99	5.78	3.38	0.00	97.34	HOF1c-8
102 / 1 .	70.18	0.22	13.01	3.15	0.09	0.01	1.03	5.15	3.43	0.01	96.28	HOF1c-9
103 / 1 .	70.19	0.23	12.83	3.16	0.11	0.01	0.92	5.31	3.41	0.00	96.18	HOF1c-10
104 / 1 .	69.88	0.21	12.92	3.27	0.13	0.01	0.97	5.07	3.44	0.01	95.91	HOF1c-11
105 / 1 .	69.86	0.21	12.89	3.18	0.09	0.01	0.94	5.53	3.27	0.02	96.00	HOF1c-12
106 / 1 .	70.34	0.23	13.05	3.04	0.09	0.03	0.96	5.75	3.48	0.04	97.00	HOF1c-13
107 / 1 .	65.67	0.05	19.62	0.68	0.00	0.02	1.95	9.46	1.53	0.00	98.98	HOF1c-14
109 / 1 .	70.19	0.22	12.81	3.16	0.11	0.02	0.97	5.30	3.44	0.00	96.21	HOF1c-16

Oxide	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
110 / 1 .	35.45	0.03	1.77	27.07	0.45	0.07	32.06	0.03	0.01	0.29	97.21	andradite 31/5 19:23
111 / 1 .	35.12	0.04	1.65	26.98	0.43	0.07	32.16	0.00	0.01	0.33	96.77	andradite 1/6 9:24
112 / 1 .	35.06	0.02	1.76	26.95	0.43	0.07	32.09	0.01	0.03	0.34	96.75	andradite 1/6 9:32
113 / 1 .	35.09	0.00	1.52	27.09	0.38	0.05	32.31	0.01	0.01	0.37	96.82	andradite 1/6 9:35
114 / 1 .	35.54	0.05	1.74	27.42	0.49	0.06	32.05	0.01	0.00	0.34	97.69	andradite 1/6 9:35 afte
115 / 1 .	49.13	1.59	13.45	11.62	0.24	7.52	12.19	2.36	0.16	0.27	98.53	FGD3a-1
116 / 1 .	48.48	1.87	13.06	12.92	0.22	6.63	11.07	2.60	0.27	0.27	97.39	HOF1d-1
121 / 1 .	73.99	0.08	12.28	1.72	0.08	0.01	1.21	4.54	2.76	0.05	96.73	HV3-1 2
122 / 1 .	75.01	0.08	12.61	1.74	0.06	0.04	1.25	4.75	3.09	0.03	98.64	HV3-1 3
123 / 1 .	74.53	0.05	12.18	1.66	0.10	0.05	1.27	4.60	2.89	0.04	97.36	HV3-1 4
124 / 1 .	74.58	0.05	12.86	1.64	0.11	0.06	1.17	4.63	2.95	0.00	98.05	HV3-1 5
125 / 1 .	74.04	0.07	12.30	1.64	0.00	0.04	1.26	4.57	2.93	0.01	96.84	HV3-1 6
127 / 1 .	73.34	0.11	12.44	1.76	0.01	0.03	1.23	4.37	2.78	0.04	96.11	HV3-1 8
128 / 1 .	74.67	0.09	12.44	1.81	0.12	0.06	1.19	4.41	2.97	0.05	97.82	HV3-1 9
129 / 1 .	74.35	0.06	12.56	1.95	0.00	0.08	1.28	4.56	2.72	0.00	97.55	HV3-1 10
130 / 1 .	73.50	0.08	12.55	1.76	0.01	0.02	1.29	4.53	2.85	0.00	96.58	HV3-1 11
131 / 1 .	74.25	0.08	12.24	1.74	0.10	0.07	1.26	4.52	2.91	0.00	97.16	HV3-1 12
132 / 1 .	74.34	0.08	12.49	1.77	0.09	0.03	1.39	4.61	2.68	0.03	97.51	HV3-1 13
133 / 1 .	34.54	0.06	1.77	26.91	0.49	0.08	32.12	0.08	0.00	0.29	96.34	andradite 1/6 10:43
134 / 1 .	35.53	0.00	1.76	27.37	0.56	0.11	32.28	0.02	0.05	0.32	97.99	andradite 1/6 11:00
135 / 1 .	35.42	0.06	1.60	27.50	0.43	0.04	32.35	0.01	0.00	0.32	97.74	andradite 1/6 12:17
136 / 1 .	69.77	0.14	13.76	3.13	0.11	0.18	2.02	4.96	2.50	0.03	96.60	HV4-1 1
137 / 1 .	70.17	0.17	13.58	3.19	0.14	0.14	1.98	5.00	2.59	0.03	96.99	HV4-1 2
140 / 1 .	70.52	0.16	13.90	3.16	0.12	0.15	1.98	4.94	2.59	0.05	97.56	HV4-1 5
142 / 1 .	65.86	0.35	14.68	5.41	0.22	0.39	3.28	4.88	2.09	0.15	97.28	HV4-1 7
143 / 1 .	58.03	0.00	25.45	0.31	0.02	0.05	7.99	7.15	0.26	0.02	99.27	HV4-1 8
144 / 1 .	56.79	1.38	14.31	9.67	0.25	2.12	5.31	4.27	1.54	0.75	96.39	HV4-1 9
146 / 1 .	71.91	0.16	14.02	3.25	0.11	0.12	1.95	4.84	2.34	0.07	98.76	HV4-1 11
147 / 1 .	62.61	0.64	14.49	7.57	0.25	0.80	4.06	4.62	1.81	0.22	97.08	HV4-1 12
149 / 1 .	69.68	0.16	13.68	2.87	0.13	0.17	1.91	4.88	2.52	0.01	96.01	HV4-1 14
150 / 1 .	71.60	0.16	14.17	2.96	0.12	0.15	1.98	4.75	2.62	0.04	98.53	HV4-1 15
151 / 1 .	70.40	0.19	13.96	2.93	0.11	0.13	1.99	4.86	2.78	0.04	97.38	HV4-1 16
152 / 1 .	69.98	0.18	13.69	3.18	0.11	0.11	1.90	4.74	2.68	0.01	96.57	HV4-1 17
153 / 1 .	70.71	0.16	14.05	3.07	0.09	0.11	1.99	4.80	2.53	0.01	97.52	HV4-1 18
154 / 1 .	71.26	0.16	13.63	3.02	0.00	0.15	1.95	4.88	2.51	0.05	97.62	HV4-1 19
155 / 1 .	35.35	0.03	1.89	27.60	0.39	0.09	32.35	0.03	0.00	0.34	98.05	andradite 1/6 13:53
156 / 1 .	75.01	0.07	12.48	1.67	0.14	0.05	1.29	4.61	3.05	0.01	98.39	HV7-2 1
158 / 1 .	74.33	0.03	12.37	1.70	0.06	0.04	1.33	4.69	2.78	0.04	97.37	HV7-2 3
159 / 1 .	55.89	0.22	26.03	1.21	0.07	0.12	9.15	5.79	0.22	0.14	98.83	HV7-2 4
160 / 1 .	75.33	0.06	12.51	1.79	0.03	0.05	1.32	4.55	2.64	0.03	98.31	HV7-2 5
161 / 1 .	48.52	2.29	13.41	12.88	0.30	6.23	10.89	2.77	0.30	0.31	97.90	HV7-2 6
163 / 1 .	74.19	0.06	12.68	1.77	0.09	0.05	1.24	4.61	2.89	0.00	97.58	HV7-2 8
164 / 1 .	65.42	0.05	19.40	0.75	0.06	0.03	4.20	7.08	1.17	0.05	98.22	HV7-2 9
165 / 1 .	73.70	0.09	12.38	1.71	0.08	0.06	1.33	4.62	2.99	0.02	96.97	HV7-2 10
166 / 1 .	74.55	0.06	12.57	1.69	0.12	0.02	1.23	4.58	2.87	0.00	97.69	HV7-2 11
167 / 1 .	74.03	0.07	12.23	1.82	0.07	0.04	1.28	4.54	2.71	0.02	96.81	HV7-2 12
168 / 1 .	74.62	0.10	12.59	1.67	0.05	0.04	1.27	4.57	2.87	0.06	97.84	HV7-2 13
169 / 1 .	73.82	0.07	12.19	1.82	0.03	0.07	1.20	4.57	3.04	0.00	96.82	HV7-2 14
170 / 1 .	35.37	0.02	1.67	27.09	0.40	0.07	31.93	0.01	0.00	0.33	96.88	andradite 1/6 15:05
171 / 1 .	35.64	0.04	1.71	27.23	0.43	0.09	32.39	0.00	0.02	0.34	97.87	andradite 1/6 15:08
172 / 1 .	35.34	0.03	1.74	27.82	0.44	0.08	31.99	0.00	0.00	0.41	97.85	andradite 1/6 15:11 fo
173 / 1 .	35.49	0.03	1.78	27.65	0.47	0.06	32.14	0.02	0.03	0.37	98.04	andradite 1/6 15:18
174 / 1 .	49.01	1.88	13.46	12.75	0.25	6.97	11.23	2.72	0.22	0.29	98.78	FGD3c-2
175 / 1 .	49.01	1.89	13.19	12.36	0.25	6.83	11.25	2.69	0.23	0.28	97.98	FGD3c-3
176 / 1 .	49.05	2.72	13.08	13.23	0.27	5.69	9.87	3.09	0.39	0.38	97.77	FGD3c-4
177 / 1 .	49.29	2.76	12.95	13.44	0.21	5.58	9.78	3.05	0.45	0.43	97.96	FGD3c-5
178 / 1 .	48.22	2.76	13.12	12.98	0.23	6.42	10.48	2.79	0.39	0.39	97.78	FGD3c-6

179 / 1 .	49.32	2.70	13.21	13.73	0.23	5.79	9.97	3.10	0.41	0.42	98.88	FGD3c-7
180 / 1 .	49.08	2.72	12.84	13.40	0.27	5.76	9.90	3.01	0.41	0.37	97.76	FGD3c-8
181 / 1 .	48.67	2.64	12.88	13.24	0.27	6.09	10.30	2.90	0.40	0.40	97.80	FGD3c-9
182 / 1 .	49.26	1.91	13.45	12.75	0.26	6.70	11.24	2.75	0.23	0.30	98.84	FGD3c-10
183 / 1 .	49.38	2.74	12.94	13.73	0.25	5.75	10.02	2.90	0.38	0.39	98.48	FGD3c-11
184 / 1 .	48.94	2.59	13.22	13.26	0.25	6.10	10.68	2.83	0.30	0.38	98.54	FGD3c-12
185 / 1 .	35.27	0.04	1.71	27.62	0.37	0.10	32.20	0.02	0.00	0.38	97.70	andradite 1/6 15:55
186 / 1 .	48.81	1.59	13.59	11.64	0.21	7.70	12.38	2.32	0.13	0.26	98.64	FGD3a-2
187 / 1 .	48.54	1.60	13.74	11.38	0.20	7.66	12.21	2.27	0.15	0.23	97.97	FGD3a-3
188 / 1 .	49.70	1.75	13.29	11.93	0.19	6.91	10.99	2.70	0.30	0.28	98.05	FGD3a-4
190 / 1 .	48.52	1.65	13.44	11.88	0.23	7.29	11.61	2.45	0.18	0.27	97.53	FGD3a-6
191 / 1 .	49.25	1.69	13.47	11.62	0.19	7.29	11.77	2.49	0.19	0.25	98.21	FGD3a-7
192 / 1 .	49.90	1.75	13.26	11.93	0.21	6.83	11.14	2.65	0.24	0.27	98.17	FGD3a-8
193 / 1 .	49.31	1.68	13.50	11.98	0.21	7.30	11.76	2.49	0.25	0.29	98.76	FGD3a-9
194 / 1 .	49.03	1.58	13.58	11.76	0.22	7.66	12.15	2.37	0.14	0.29	98.77	FGD3a-10
195 / 1 .	48.77	1.55	13.58	11.61	0.25	7.70	12.20	2.32	0.13	0.28	98.38	FGD3a-11
196 / 1 .	49.17	1.58	13.43	11.58	0.24	7.71	12.27	2.33	0.18	0.28	98.77	FGD3a-12
197 / 1 .	48.96	1.57	13.60	11.76	0.19	7.68	12.24	2.33	0.16	0.28	98.77	FGD3a-13
198 / 1 .	35.48	0.03	1.78	27.71	0.45	0.07	32.20	0.01	0.00	0.42	98.14	andradite 1/6 16:32
199 / 1 .	35.29	0.04	1.71	27.51	0.48	0.06	32.33	0.02	0.02	0.33	97.78	andradite 1/6 16:35

Oxide	SiO2	TiO2	Al2O3	FeO	MnO	MgO	CaO	Na2O	K2O	P2O5	Total	Comment
200 / 1 .	48.86	2.20	12.57	13.59	0.26	6.22	11.06	2.56	0.27	0.33	97.90	HOF1d-2
201 / 1 .	48.82	1.91	13.02	12.70	0.23	6.77	11.27	2.33	0.27	0.29	97.61	HOF1d-3
202 / 1 .	49.20	1.89	13.17	12.59	0.24	6.72	11.12	2.57	0.23	0.31	98.03	HOF1d-4
203 / 1 .	49.21	2.50	13.28	13.24	0.27	5.96	10.22	2.79	0.40	0.37	98.25	HOF1d-5
204 / 1 .	48.84	1.93	13.18	12.69	0.26	6.91	11.31	2.66	0.27	0.31	98.36	HOF1d-6
206 / 1 .	48.80	1.82	13.39	12.33	0.22	6.88	11.57	2.49	0.23	0.29	98.01	HOF1d-7
207 / 1 .	49.27	1.83	13.36	12.94	0.24	6.83	11.35	2.53	0.23	0.29	98.86	HOF1d-8
208 / 1 .	48.96	1.93	13.06	13.11	0.19	6.67	11.22	2.59	0.28	0.33	98.33	HOF1d-9
209 / 1 .	48.62	1.86	13.20	12.77	0.16	6.81	11.33	2.58	0.22	0.29	97.85	HOF1d-10
210 / 1 .	48.58	1.88	13.23	12.74	0.24	6.99	11.26	2.63	0.23	0.31	98.09	HOF1d-11
211 / 1 .	48.88	1.92	13.12	12.90	0.22	6.83	11.33	2.55	0.26	0.27	98.27	HOF1d-12
212 / 1 .	35.11	0.03	1.71	27.62	0.49	0.07	32.03	0.00	0.00	0.39	97.44	andradite 1/6 17:12
213 / 1 .	35.26	0.03	1.69	27.77	0.45	0.07	32.14	0.02	0.01	0.37	97.81	andradite 1/6 17:15

Appendix 2

Model code:

1: Ice_model_Run.java

2: ClimateControl.java

3: PDDSum.java

4: PrecipModel.java

5: ELAFinderComplex.java

6: Raster.java

7: RasterScalePicker.java

8: RasterPanelVik.java

```
1
2   /*Top class for Iceland climate and vegetation model.  Set up to run as an
3    *application.
4    *© Andy Casely 2005.  Last updated 7th April.
5    *This class draws the GUI and manages GUI inputs and outputs.*/
6
7   package afc.spatial;
8
9   import java.awt.*;
10  import java.util.*;
11  import java.awt.event.*;
12  import java.awt.geom.*;
13  import javax.swing.*;
14  import javax.swing.text.*;
15  import afc.gui.*;
16  import afc.utilities.*;
17  import javax.swing.event.MouseInputAdapter;
18  import java.awt.image.BufferedImage;
19  import java.awt.image.MemoryImageSource;
20
21  public class Ice_model_Run extends JFrame implements ActionListener
22  {
23     private Container contentPane;
24     private JButton topoB,highlightB,tempB,resetB,pddGoB,fillB,fluvB,
25           precipB,setKB,snowCalcB,elaOffCalcB,elaCalcB,lapseB,pLapseB,
26           massBalB,ela_TopoB,lowB,pddVegB,growSCalcB,autoControlB,
27           growSStartB,growSEndB,growSMeanTB,shiftela,tempshift,
28           snowCCalcB,snowCResetB,precipshift,sunShedB,glimmerB,aigB,gbsB;
29     private JTextField xllInput,yllInput,xurInput,yurInput,tfileinput,
30           tdirinput,kCoarse,focalField,limT,pddSnowIce,autoFileInput,
31           statusValueField,statusField,shiftamt,shiftTamt,shiftPamt,
32
    snowcMonth,pddVegLim,aspectA,aspectI,sSRadius,pddSShed,fSShed,clickVal,
33           drainageMin,basinMin,basinMax;
34     private JTextArea messageOutput;
35     private JLabel
    dataTitle,dataName,dataCells,dataNumCells,dataKm,dataCellsize,
36           dataReg,oldMinRsc,oldMaxRsc,allowC;
37     private JCheckBox saveCB;
38     private JComboBox tempCB,precipCB,pddCB,sunsCB;
39     private MenuItem menuEditChanges;
40     private Choice tkChoice;
41     private BufferedImage bi;
42     private Color border_colour,fill_colour,point_colour;
43     private BasicStroke stroke = new BasicStroke(1);
44     private JPanel p0,pInfo,gradPanel,rastSFV,
45           rastInfoPanel,rastScaleP,rastStatusPanel,pInfoIcelandModel;
46     private Raster dispR,paintedRaster,editsR;
47     private Vector rasters,strings,displayLines,displayPoints,mouseCoords;
48     private String root_dir,defRoot_dir,dem,defDEM,dirName,defDirName;
49     private String[] months = new String[13];
50     private boolean firstModelRun,highlightCheck,tempOrPrecipImported,
51           multiRasterFlag,snowVegFlag,hiLoFlag,saveFileFlag,kSet,precipIndex,
52           pddIndex,sIndex,vikingFarmFlag,allowChanges,editsLayer,mPressed,
53           fileSaved,drainagePath,editPoints,editLines,editAreas,
54           showRiversAndLakes,showIceArea,showIceMargins,showSandar,
55           dChanged,newDrainage,trimToIce,trimToSandar,showDrainageVectors,
56           largerDrainage,blackAndWhite;
57
58     private int infoPWidth,infoPHeight,buttonHeight,textFieldHeight,
59
    pCAPWidth,rasterPWidth,selectDispHeight,noData,ps,upperLayer,dragNo,
60           basMin,basMax,drainMin,optionIndex;
61     private float oldMin,oldMax,ts,clickValue;
62     private double xllD,yllD,xurD,yurD,xll,yll,xur,yur,imageScaling;
63     private Object[] optionObjects;
64     private int[] optionResults;
65     private RasterPanelVik rastPanel;
66     private GradientPanel rastGradientPanel;
67     private ClimateControl climateControl;
68     private Viking_Farms_Run vfr;
```



```
69     private ModelPanel modelPanel;
70     private afc.spatial.Point start,end;
71     private static final int BORDER_SIZE = 0;
72
73     public Ice_model_Run()
74     {
75  //---------create a JPanel container in a new window with a title-------
76        super("Iceland Raster Model");
77  //    rt=Runtime.getRuntime();
78  //--------preset coordinates for selected test area (Thorsmork approx.)
79        xllD = 445000;
80        yllD = 337000;
81        xurD = 478000;
82        yurD = 360000;
83        oldMin = 0;
84        oldMax = 2100;
85  //--------preset files and folders
86        defRoot_dir = "C:/gisdata/";
87        root_dir = defRoot_dir;
88        defDEM = "icedem_cp.asc";
89        dem = defDEM;
90        defDirName = "testfolder";
91        dispR = new Raster(0,0,1,1,1,0,0);
92
93  //--------preset Panel settings
94        buttonHeight=39;
95        textFieldHeight=34;
96        infoPWidth=500;
97        selectDispHeight = (textFieldHeight*4)+buttonHeight;
98        rasterPWidth=452;
99        infoPHeight=720;
100       pCAPWidth = 300;
101 //other presets
102       imageScaling =4;
103       clickValue = -1;
104       months = MonthsAndDays.getMonths();
105       drainMin=400;
106       basMin=100;
107       basMax=200000;
108       strings = new Vector();
109       highlightCheck=false;
110       firstModelRun=true;
111       tempOrPrecipImported=false;
112       kSet=false;
113       multiRasterFlag=false;
114       snowVegFlag=true;
115       hiLoFlag=false;
116       saveFileFlag=false;
117       fileSaved=false;
118       vikingFarmFlag = false;
119       allowChanges=false;
120       dChanged=true;newDrainage=true;
121       blackAndWhite=false;
122       hideAllOverlays();
123
124 //-----------Menu Bar Construction---------
125       MenuBar menuBar = new MenuBar();
126       Menu menuFile = new Menu("File");
127       menuFile.addActionListener(this);
128       menuBar.add(menuFile);
129       Menu menuEdit = new Menu("Edit");
130       menuEdit.addActionListener(this);
131       menuBar.add(menuEdit);
132       Menu menuRegion = new Menu("Preset Region");
133       menuRegion.addActionListener(this);
134       menuBar.add(menuRegion);
135       Menu menuModel = new Menu("Model Selection");
136       menuModel.addActionListener(this);
137       menuBar.add(menuModel);
138       setMenuBar(menuBar);
139
```

```
140         MenuItem menuViewParams = new MenuItem("View Settings");
141         menuFile.add(menuViewParams);
142         MenuItem menuDrainageParams = new MenuItem("Drainage and Basin
    Settings");
143         menuFile.add(menuDrainageParams);
144         MenuItem menuShowVectors = new MenuItem("Show Drainage Vectors");
145         menuFile.add(menuShowVectors);
146         menuFile.addSeparator();
147         MenuItem menuFileExit = new MenuItem("Exit");
148         menuFile.add(menuFileExit);
149
150         menuEditChanges = new MenuItem("Allow Raster Changes");
151         menuEdit.add(menuEditChanges);
152         MenuItem menuEditsSaveT = new MenuItem("Save Edits to Topo");
153         menuEdit.add(menuEditsSaveT);
154         MenuItem menuEditsRevert = new MenuItem("Revert to original DEM");
155         menuEdit.add(menuEditsRevert);
156         MenuItem menuEditsReset = new MenuItem("Reset saves to original DEM"
157         menuEdit.add(menuEditsReset);
158         menuEdit.addSeparator();
159         MenuItem menuEditsLayer = new MenuItem("Show Edits Layer");
160         menuEdit.add(menuEditsLayer);
161         MenuItem menuEditsSaveL = new MenuItem("Save Changes to Edits Layer"
162         menuEdit.add(menuEditsSaveL);
163         MenuItem menuEditsClear = new MenuItem("Clear Edits Layer");
164         menuEdit.add(menuEditsClear);
165         MenuItem menuFileIceland = new MenuItem("All Iceland");
166         menuRegion.add(menuFileIceland);
167         MenuItem menuFileB = new MenuItem("Breida");
168         menuRegion.add(menuFileB);
169         MenuItem menuFileE = new MenuItem("East Fjords");
170         menuRegion.add(menuFileE);
171         MenuItem menuFileVatna = new MenuItem("W Vatnajokull");
172         menuRegion.add(menuFileVatna);
173         MenuItem menuFileEVatn = new MenuItem("E Vatnajokull");
174         menuRegion.add(menuFileEVatn);
175         MenuItem menuFileSEVatn = new MenuItem("SE Vatnajokull");
176         menuRegion.add(menuFileSEVatn);
177         MenuItem menuFileHofsjokull = new MenuItem("Hofsjokull");
178         menuRegion.add(menuFileHofsjokull);
179         MenuItem menuFileHofsjokullm = new MenuItem("Hofsjokull-m");
180         menuRegion.add(menuFileHofsjokullm);
181         MenuItem menuFileMyrdalsjokull = new MenuItem("Myrdalsjokull");
182         menuRegion.add(menuFileMyrdalsjokull);
183         MenuItem menuFileMyvatn = new MenuItem("Myvatn");
184         menuRegion.add(menuFileMyvatn);
185         MenuItem menuFileMyvatnS = new MenuItem("Myvatn & S");
186         menuRegion.add(menuFileMyvatnS);
187         MenuItem menuFileNW = new MenuItem("NW Fjords");
188         menuRegion.add(menuFileNW);
189         MenuItem menuFileReykholt = new MenuItem("Reykholtsdalur");
190         menuRegion.add(menuFileReykholt);
191         MenuItem menuFileOraefi = new MenuItem("Oraefi");
192         menuRegion.add(menuFileOraefi);
193         MenuItem menuFileSnae = new MenuItem("Snaefellsnes");
194         menuRegion.add(menuFileSnae);
195         MenuItem menuFileSvarfadardalur = new MenuItem("Svarfadardalur");
196         menuRegion.add(menuFileSvarfadardalur);
197         MenuItem menuFileThorsmork = new MenuItem("Thorsmork");
198         menuRegion.add(menuFileThorsmork);
199         MenuItem menuFileThorsmork2 = new MenuItem("Thorsmork2");
200         menuRegion.add(menuFileThorsmork2);
201         MenuItem menuFileTrollaskagi = new MenuItem("Trollaskagi");
202         menuRegion.add(menuFileTrollaskagi);
203         MenuItem menuFileVatnsdalur = new MenuItem("Vatnsdalur");
204         menuRegion.add(menuFileVatnsdalur);
205         MenuItem menuFiletest = new MenuItem("testarea");
206         menuRegion.add(menuFiletest);
207
208         MenuItem menuFileVikingFarms = new MenuItem("Viking Farms Model");
209         menuModel.add(menuFileVikingFarms);
```



```
210         MenuItem menuFileIcelandModel = new MenuItem("Iceland Climate Model"
211         menuModel.add(menuFileIcelandModel);
212
213   /*---initialise the top-level Container and add a WindowListener to manage
214    *closing events*/
215         contentPane = getContentPane();
216         contentPane.setLayout(new BorderLayout());
217         setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);
218         addWindowListener(new WinMonitor());
219
220   /*-------top container pTopLevel divided into 2: LHS will be info and
      ontrols
221    *(pInfo), RHS will be the raster panel, scale bar and mouse information *
222         JPanel pTopLevel = new JPanel(new BorderLayout());
223         pTopLevel.setBorder(BorderFactory.createEmptyBorder(0,0,0,0));
224         pTopLevel.setPreferredSize(new
      Dimension(infoPWidth+rasterPWidth,infoPHeight));
225         contentPane.add(pTopLevel);
226
227   /*-------pInfo panel includes all the buttons and controls for model area
228    *selection (selectAndDisplay) and for climate model control
229    *(pInfoIcelandModel)*/
230         pInfo = new JPanel();
231         pInfo.setPreferredSize(new Dimension(infoPWidth,infoPHeight));
232         pInfo.setBorder(BorderFactory.createBevelBorder(2));
233         pTopLevel.add(pInfo,BorderLayout.WEST);
234
235   //-----p0 is the panel containing the raster and raster information
236         p0=new JPanel(new BorderLayout());
237         p0.setPreferredSize(new Dimension(rasterPWidth,infoPHeight));
238         pTopLevel.add(p0,BorderLayout.EAST);
239
240   //pSelectAndDisplay contains the area selection and info display panels
241         JPanel pSelectAndDisplay = new JPanel(new BorderLayout());
242         pSelectAndDisplay.setPreferredSize(new
      Dimension(infoPWidth-4,selectDispHeight));
243         pInfo.add(pSelectAndDisplay,BorderLayout.SOUTH);
244
245   //pInfoIcelandModel has the controls for the climate  and vegetation model
246         pInfoIcelandModel = buildClimateControls();
247         pInfo.add(pInfoIcelandModel);
248
249   //areaSelectPanel contains the file settings, coordinates, and the area
      buttons
250         JPanel areaSelectPanel = new JPanel();
251         areaSelectPanel.setPreferredSize(new
      Dimension(pCAPWidth,selectDispHeight));
252         pSelectAndDisplay.add(areaSelectPanel,BorderLayout.WEST);
253
254   //fileInfo panel contains the directory and DEM names
255         JPanel fileInfo = new JPanel();
256         fileInfo.setBorder(BorderFactory.createEtchedBorder());
257         fileInfo.setPreferredSize(new Dimension(pCAPWidth,textFieldHeight));
258         JLabel rootD = new JLabel(defRoot_dir,SwingConstants.RIGHT);
259         rootD.setFont(new Font("Times New Roman",Font.BOLD,11));
260         fileInfo.add(rootD);
261         tdirinput = new JTextField(6);
262         tdirinput.setText(defDirName);
263         fileInfo.add(tdirinput);
264         JLabel demD = new JLabel("DEM:",SwingConstants.RIGHT);
265         demD.setFont(new Font("Times New Roman",Font.BOLD,11));
266         fileInfo.add(demD);
267         tfileinput = new JTextField(8);
268         tfileinput.setText(defDEM);
269         fileInfo.add(tfileinput);
270         areaSelectPanel.add(fileInfo,0);
271
272   //pCoord panel has th manual input raster coordinate boxes
273         JPanel pCoord = new JPanel(new BorderLayout());
274         pCoord.setPreferredSize(new
```

```
274   Dimension(pCAPWidth,(textFieldHeight*3)-10));
275         pCoord.setBorder(BorderFactory.createEtchedBorder());
276         JPanel pInput4 = new JPanel();
277         pInput4.add(new JLabel("YUR:"));
278         yurInput = new JTextField(7);
279         yurInput.setText(Double.toString(yurD));
280         pInput4.add(yurInput);
281         pCoord.add(pInput4,BorderLayout.NORTH);
282         JPanel pInput1 = new JPanel();
283         pInput1.add(new JLabel("XLL:"));
284         xllInput = new JTextField(7);
285         xllInput.setText(Double.toString(xllD));
286         pInput1.add(xllInput);
287         pCoord.add(pInput1,BorderLayout.WEST);
288         JPanel pInput3 = new JPanel();
289         pInput3.add(new JLabel("XUR:"));
290         xurInput = new JTextField(7);
291         xurInput.setText(Double.toString(xurD));
292         pInput3.add(xurInput);
293         pCoord.add(pInput3,BorderLayout.EAST);
294         JPanel pInput2 = new JPanel();
295         pInput2.add(new JLabel("YLL:"));
296         yllInput = new JTextField(7);
297         yllInput.setText(Double.toString(yllD));
298         pInput2.add(yllInput);
299         pCoord.add(pInput2,BorderLayout.SOUTH);
300         areaSelectPanel.add(pCoord,1);
301
302   //pArea contains buttons to reset the raster and display the general area
303         JPanel pAreaButtons = new JPanel(new GridLayout(1,3));
304         pAreaButtons.setPreferredSize(new
      Dimension(pCAPWidth,buttonHeight-4));
305         highlightB = new JButton("Show Area");
306         highlightB.addActionListener(this);
307         pAreaButtons.add(highlightB,0);
308         topoB = new JButton("Get Topo");
309         topoB.addActionListener(this);
310         pAreaButtons.add(topoB,1);
311         resetB = new JButton("Reset");
312         resetB.addActionListener(this);
313         pAreaButtons.add(resetB,2);
314         areaSelectPanel.add(pAreaButtons,2);
315
316   //displayInfoPanel contains the header information about the raster and is
      located to
317   //the right of the areaSelectPanel panel
318         JPanel displayInfoPanel = new JPanel(new GridLayout(7,1));
319         displayInfoPanel.setFont(new Font("Times New Roman",Font.BOLD,12));

320         displayInfoPanel.setBorder(BorderFactory.createBevelBorder(0));
321         displayInfoPanel.setPreferredSize(new
      Dimension((infoPWidth-4)-pCAPWidth,selectDispHeight));
322         pSelectAndDisplay.add(displayInfoPanel,BorderLayout.EAST);
323
324         JPanel displayInfo0 = new JPanel();
325         dataName = new JLabel("Raster");
326         dataName.setFont(new Font("Times New Roman",Font.BOLD,16));
327         displayInfo0.add(dataName);
328         displayInfoPanel.add(displayInfo0,0);
329         JPanel displayInfo1 = new JPanel();
330         displayInfo1.add(new JLabel("Source:"));
331         dataTitle = new JLabel(defRoot_dir+defDEM);
332         displayInfo1.add(dataTitle);
333         displayInfoPanel.add(displayInfo1,1);
334         JPanel displayInfo2 = new JPanel();
335         displayInfo2.add(new JLabel("Cell dimensions: "));
336         dataCells = new JLabel("Cols x Rows");
337         displayInfo2.add(dataCells);
338         displayInfoPanel.add(displayInfo2,2);
339         JPanel displayInfo6 = new JPanel();
340         displayInfo6.add(new JLabel("No. of cells: "));
```



```
341         dataNumCells = new JLabel("# Cells");
342         displayInfo6.add(dataNumCells);
343         displayInfoPanel.add(displayInfo6,3);
344         JPanel displayInfo3 = new JPanel();
345         displayInfo3.add(new JLabel("Extent"));
346         dataKm = new JLabel("km x km");
347         displayInfo3.add(dataKm);
348         displayInfoPanel.add(displayInfo3,4);
349         JPanel displayInfo4 = new JPanel();
350         displayInfo4.add(new JLabel("Cellsize: "));
351         dataCellsize = new JLabel("Cells");
352         displayInfo4.add(dataCellsize);
353         displayInfoPanel.add(displayInfo4,5);
354         JPanel displayInfo5 = new JPanel();
355         displayInfo5.add(new JLabel("LL corner: "));
356         dataReg = new JLabel("xll,yll");
357         displayInfo5.add(dataReg);
358         displayInfoPanel.add(displayInfo5,6);
359
360   //rastPanel is the main display panel for the model
361         modelPanel = new ModelPanel();
362         modelPanel.initialise();
363         rastPanel = new RasterPanelVik(this,new
      Dimension(rasterPWidth,infoPHeight-144));
364         rastPanel.addGraphicsListener(modelPanel);
365
366         modelPanel.setBorder(BorderFactory.createEtchedBorder());
367         p0.add(modelPanel,BorderLayout.NORTH);
368
369   //dataPanel includes the status and progress panels
370         JPanel dataPanel = new JPanel(new BorderLayout());
371         dataPanel.setPreferredSize(new Dimension(rasterPWidth,144));
372         p0.add(dataPanel,BorderLayout.SOUTH);
373
374   //the panel for status messages and progress
375         messageOutput = new JTextArea();
376         messageOutput.setLineWrap(true);
377         messageOutput.setEditable(false);
378         JScrollPane scrollPane = new JScrollPane(messageOutput);
379         scrollPane.setPreferredSize(new Dimension(rasterPWidth,100));
380         dataPanel.add(scrollPane,BorderLayout.SOUTH);
381
382   //the panel containing the scale bar(rastInfoPanel) and the mouse info
383         rastStatusPanel = new JPanel(new GridLayout(1,2));
384         rastStatusPanel.setPreferredSize(new Dimension(rasterPWidth,44));
385         dataPanel.add(rastStatusPanel,BorderLayout.NORTH);
386
387         rastInfoPanel = new JPanel(new GridLayout(2,1));
388         rastInfoPanel.setBorder(BorderFactory.createEtchedBorder());
389         rastInfoPanel.setPreferredSize(new Dimension(100,44));
390         rastStatusPanel.add(rastInfoPanel,BorderLayout.EAST);
391
392         gradPanel = new JPanel(new GridLayout(1,1));
393         gradPanel.add(new JLabel("gradient"));
394         gradPanel.setBorder(BorderFactory.createEtchedBorder());
395         rastScaleP = new JPanel(new GridLayout(1,3));
396         oldMinRsc = new JLabel(Float.toString(oldMin));
397         rastScaleP.add(oldMinRsc);
398         rastScaleP.add(new JLabel(" --- "));
399         oldMaxRsc = new JLabel(Float.toString(oldMax));
400         rastScaleP.add(oldMaxRsc);
401         rastInfoPanel.add(gradPanel,0);
402         rastInfoPanel.add(rastScaleP,1);
403
404         statusField = new JTextField("cursor:");
405         statusField.setEditable(false);
406         statusValueField = new JTextField("value:");
407         statusValueField.setEditable(false);
408         rastSFV = new JPanel(new GridLayout(2,1));
409         rastSFV.setBorder(BorderFactory.createEtchedBorder());
410         rastSFV.add(statusField,0);
```

```
411         rastSFV.add(statusValueField,1);
412         rastStatusPanel.add(rastSFV,BorderLayout.WEST);
413
414   //-----size everything in the window and make visible
415         pack();
416         setVisible(true);
417      }
418
419      public JPanel buildClimateControls()
420      {
421   //pInfoIcelandModel has the controls for the climate  and vegetation model
422         pInfoIcelandModel = new JPanel();
423         pInfoIcelandModel.setPreferredSize(new
      Dimension(infoPWidth,infoPHeight-selectDispHeight-8));
424         pInfoIcelandModel.setBorder(BorderFactory.createBevelBorder(2));
425
426         //pBegin2 - wraps the begin panel and the resolution panel
427         JPanel pBegin2 = new JPanel(new BorderLayout());
428         pInfoIcelandModel.add(pBegin2,BorderLayout.SOUTH);
429
430   //Resolution switch and selection of whether to save a JPEG
431         JPanel resSavePanel = new JPanel(new GridLayout(4,1));
432         resSavePanel.setPreferredSize(new Dimension(120,90));
433         pBegin2.add(resSavePanel,BorderLayout.WEST);
434         lowB = new JButton("LOW");
435         lowB.setBorder(BorderFactory.createBevelBorder(0));
436         lowB.addActionListener(this);
437         resSavePanel.add(lowB);
438         saveCB = new JCheckBox("Save Image",saveFileFlag);
439         resSavePanel.add(saveCB);
440         clickVal = new JTextField(3);
441         clickVal.setText(Float.toString(-1));
442         resSavePanel.add(clickVal);
443         allowC = new JLabel("Allow Changes");
444         allowC.setFont(new Font("Times New Roman",Font.BOLD,11));
445         allowC.setForeground(Color.RED);
446         resSavePanel.add(allowC);
447
448   /*pBegin is the 'Start' panel - getting temperature or precipitation will
449    *initialise the model with whatever coordinates are in the input panel, a
450    *save results to the appropriate folder*/
451         JPanel pBegin = new JPanel();
452         pBegin.setBorder(BorderFactory.createEtchedBorder());
453         pBegin2.add(pBegin,BorderLayout.EAST);
454
455   //tPanel is the temperature control panel
456         JPanel tPanel = new JPanel(new BorderLayout());
457         tPanel.setBorder(BorderFactory.createEtchedBorder());
458         tPanel.setPreferredSize(new Dimension(150,91));
459         pBegin.add(tPanel);
460
461         tempB = new JButton("Get Temps");
462         tempB.addActionListener(this);
463         tempB.setMargin(new Insets(3,3,3,3));
464         tPanel.add(tempB,BorderLayout.WEST);
465
466         JPanel tButtPanel = new JPanel(new GridLayout(3,1));
467         tButtPanel.setPreferredSize(new Dimension(73,66));
468         tPanel.add(tButtPanel,BorderLayout.EAST);
469
470         tempCB = new JComboBox(months);
471         tempCB.addActionListener(this);
472         tButtPanel.add(tempCB,0);
473         tempshift = new JButton("shiftT");
474         tempshift.setFont(new Font("Times New Roman",Font.BOLD,11));
475         tempshift.addActionListener(this);
476         tButtPanel.add(tempshift,1);
477         shiftTamt = new JTextField(2);
478         shiftTamt.setText(Integer.toString(0));
479         tButtPanel.add(shiftTamt,2);
480
```



```
481   //temp K calculation Panel
482         JPanel kPanel = new JPanel(new BorderLayout());
483         kPanel.setBorder(BorderFactory.createEtchedBorder());
484         kPanel.setPreferredSize(new Dimension(70,25));
485         tPanel.add(kPanel,BorderLayout.SOUTH);
486         setKB = new JButton("Set Temp K:");
487         setKB.addActionListener(this);
488         setKB.setMargin(new Insets(2,2,2,2));
489         setKB.setPreferredSize(new Dimension(105,25));
490         kPanel.add(setKB,BorderLayout.WEST);
491         tkChoice = new Choice();
492         for(int i=0;i<9;i++)
493            tkChoice.addItem(""+(i+1));
494         kPanel.add(tkChoice,BorderLayout.EAST);
495
496   //pptPanel is the precipitation control panel
497         JPanel pptPanel = new JPanel(new BorderLayout());
498         pptPanel.setBorder(BorderFactory.createEtchedBorder());
499         pptPanel.setPreferredSize(new Dimension(150,66));
500         pBegin.add(pptPanel);
501
502         precipB = new JButton("Get Precip");
503         precipB.addActionListener(this);
504         precipB.setMargin(new Insets(3,3,3,3));
505         pptPanel.add(precipB,BorderLayout.WEST);
506
507         JPanel pButtPanel = new JPanel(new GridLayout(3,1));
508         pButtPanel.setPreferredSize(new Dimension(75,66));
509         pptPanel.add(pButtPanel,BorderLayout.EAST);
510         precipCB = new JComboBox(months);
511         precipCB.addActionListener(this);
512         pButtPanel.add(precipCB,0);
513         precipshift = new JButton("shiftP");
514         precipshift.setFont(new Font("Times New Roman",Font.BOLD,11));
515         precipshift.addActionListener(this);
516         pButtPanel.add(precipshift,1);
517         shiftPamt = new JTextField(2);
518         shiftPamt.setText(Integer.toString(0));
519         pButtPanel.add(shiftPamt,2);
520
521   //panel for sun visibility calculation
522         JPanel sunShedContainer = new JPanel(new BorderLayout());
523         sunShedContainer.setPreferredSize(new Dimension(infoPWidth-4,60));
524         pInfoIcelandModel.add(sunShedContainer,BorderLayout.SOUTH);
525         sunShedContainer.add(new JLabel("   Sun visibility
      calculation:"),BorderLayout.NORTH);
526
527         JPanel sunShedPanel = new JPanel();
528         sunShedPanel.setBorder(BorderFactory.createEtchedBorder());
529         sunShedPanel.setPreferredSize(new Dimension(infoPWidth-4,45));
530         sunShedContainer.add(sunShedPanel,1);
531
532         sunShedB = new JButton("Sun Visibility");
533         sunShedB.addActionListener(this);
534         sunShedPanel.add(sunShedB);
535
536         JPanel sunShedRPanel = new JPanel(new GridLayout(2,1));
537         sunShedRPanel.setPreferredSize(new Dimension(75,35));
538         sunShedPanel.add(sunShedRPanel);
539         JLabel sunRadius = new JLabel("radius:");
540         sunRadius.setFont(new Font("Times New Roman",Font.BOLD,11));
541         sunShedRPanel.add(sunRadius,0);
542         sSRadius = new JTextField(3);
543         sSRadius.setText(Integer.toString(12));
544         sunShedRPanel.add(sSRadius,1);
545         sunsCB = new JComboBox(months);
546         sunsCB.addActionListener(this);
547         sunShedPanel.add(sunsCB,0);
548
549         JPanel sunShedFPanel = new JPanel(new GridLayout(2,2));
550         sunShedFPanel.setPreferredSize(new Dimension(155,35));
```

```
551         sunShedPanel.add(sunShedFPanel);
552         JLabel fLabel = new JLabel("Factor:");
553         fLabel.setFont(new Font("Times New Roman",Font.BOLD,11));
554         sunShedFPanel.add(fLabel,0);
555         fSShed = new JTextField(3);
556         fSShed.setText(Float.toString(0f));
557         sunShedFPanel.add(fSShed,1);
558         JLabel normPDD = new JLabel("norm PDD amt:");
559         normPDD.setFont(new Font("Times New Roman",Font.BOLD,11));
560         sunShedFPanel.add(normPDD,2);
561         pddSShed = new JTextField(3);
562         pddSShed.setText(Float.toString(0.5f));
563         pddSShed.setEditable(false);
564         sunShedFPanel.add(pddSShed,3);
565
566   //---------Buttons relating to the PDD sum and K calculation
567         JPanel pddAndKPanel = new JPanel(new BorderLayout());
568         pddAndKPanel.setPreferredSize(new Dimension(infoPWidth-2,75));
569         pInfoIcelandModel.add(pddAndKPanel,BorderLayout.SOUTH);
570
571         JPanel rasterP = new JPanel(new GridLayout(2,2));
572         rasterP.setPreferredSize(new Dimension(200,75));
573         fillB = new JButton("basins");
574         fillB.addActionListener(this);
575         rasterP.add(fillB);
576         glimmerB = new JButton("glimmer data");
577         glimmerB.addActionListener(this);
578         rasterP.add(glimmerB);
579         aigB = new JButton("AIG");
580         aigB.addActionListener(this);
581         rasterP.add(aigB);
582         gbsB = new JButton("GBS");
583         gbsB.addActionListener(this);
584         rasterP.add(gbsB);
585         pddAndKPanel.add(rasterP,BorderLayout.WEST);
586
587   //PDD calculation Panel
588         JPanel pddPanel = new JPanel();
589         pddPanel.setPreferredSize(new Dimension((int)infoPWidth-205,75));
590         pddPanel.setBorder(BorderFactory.createEtchedBorder());
591         pddAndKPanel.add(pddPanel,BorderLayout.EAST);
592
593         pddPanel.add(new JLabel("PDD:"));
594         JPanel pddBPanel = new JPanel(new BorderLayout());
595         pddBPanel.setPreferredSize(new Dimension(135,40));
596         JPanel pddGoPanel = new JPanel(new GridLayout(2,1));
597         pddGoPanel.setBorder(BorderFactory.createEtchedBorder());
598         pddGoB = new JButton("snow");
599         pddGoB.addActionListener(this);
600         pddGoPanel.add(pddGoB);
601         pddVegB = new JButton("veg");
602         pddVegB.addActionListener(this);
603         pddGoPanel.add(pddVegB,1);
604         pddPanel.add(pddGoPanel);
605
606         JPanel pddParamPanel = new JPanel(new GridLayout(3,2));
607         pddParamPanel.setPreferredSize(new Dimension(140,60));
608         pddPanel.add(pddParamPanel);
609         JLabel si = new JLabel("Snow-Ice K:");
610         si.setFont(new Font("Times New Roman",Font.BOLD,11));
611         pddParamPanel.add(si,0);
612         pddSnowIce = new JTextField(3);
613         pddSnowIce.setText(Float.toString(0.007f));
614         pddParamPanel.add(pddSnowIce,1);
615         JLabel vg = new JLabel("Veg Limit:");
616         vg.setFont(new Font("Times New Roman",Font.BOLD,11));
617         pddParamPanel.add(vg,2);
618         pddVegLim = new JTextField(3);
619         pddVegLim.setText(Integer.toString(4));
620         pddParamPanel.add(pddVegLim,3);
621         pddCB = new JComboBox(months);
```



```
622         pddCB.addActionListener(this);
623         pddParamPanel.add(new JPanel(),4);
624         pddParamPanel.add(pddCB,5);
625
626   //panel for growing season calculation
627         JPanel growSContainer = new JPanel(new BorderLayout());
628         growSContainer.setPreferredSize(new Dimension(infoPWidth-4,60));
629         pInfoIcelandModel.add(growSContainer,BorderLayout.SOUTH);
630
631         growSContainer.add(new JLabel("   Growing season
    calculation:"),BorderLayout.NORTH);
632         JPanel growSPanel = new JPanel();
633         growSPanel.setBorder(BorderFactory.createEtchedBorder());
634         growSPanel.setPreferredSize(new Dimension(infoPWidth-4,45));
635         growSContainer.add(growSPanel,1);
636
637         growSCalcB = new JButton("Growing Season length");
638         growSCalcB.addActionListener(this);
639         growSPanel.add(growSCalcB);
640         growSStartB = new JButton("Start of GS");
641         growSStartB.addActionListener(this);
642         growSPanel.add(growSStartB);
643         growSEndB = new JButton("End of GS");
644         growSEndB.addActionListener(this);
645         growSPanel.add(growSEndB);
646         growSMeanTB = new JButton("Mean T of GS");
647         growSMeanTB.addActionListener(this);
648         growSPanel.add(growSMeanTB);
649
650   //panel for snowcover and aspect calculation
651         JPanel snowAspect = new JPanel();
652         snowAspect.setPreferredSize(new Dimension(infoPWidth-4,66));
653         pInfoIcelandModel.add(snowAspect,BorderLayout.SOUTH);
654
655         JPanel snowCContainer = new JPanel(new BorderLayout());
656         snowCContainer.setPreferredSize(new Dimension(infoPWidth-214,60));
657         snowAspect.add(snowCContainer,BorderLayout.WEST);
658         snowCContainer.add(new JLabel("   Snowcover
    calculation:"),BorderLayout.NORTH);
659
660         JPanel snowCPanel = new JPanel();
661         snowCPanel.setBorder(BorderFactory.createEtchedBorder());
662         snowCPanel.setPreferredSize(new Dimension(infoPWidth-214,45));
663         snowCContainer.add(snowCPanel,1);
664
665         JPanel aspectContainer = new JPanel(new BorderLayout());
666         aspectContainer.setPreferredSize(new Dimension(200,60));
667         snowAspect.add(aspectContainer,BorderLayout.EAST);
668         aspectContainer.add(new JLabel("   Aspect for
    snowfall:"),BorderLayout.NORTH);
669
670         JPanel aspectPanel = new JPanel();
671         aspectPanel.setBorder(BorderFactory.createEtchedBorder());
672         aspectPanel.setPreferredSize(new Dimension(200,45));
673         aspectContainer.add(aspectPanel,1);
674
675         snowCCalcB = new JButton("Snowcover");
676         snowCCalcB.addActionListener(this);
677         snowCPanel.add(snowCCalcB);
678
679         JPanel snowcMPanel = new JPanel(new GridLayout(2,1));
680         snowcMPanel.setPreferredSize(new Dimension(70,35));
681         JLabel mnsnowc = new JLabel("month:");
682         mnsnowc.setFont(new Font("Times New Roman",Font.BOLD,11));
683         snowcMPanel.add(mnsnowc,0);
684         snowcMonth = new JTextField(2);
685         snowcMonth.setEditable(false);
686         snowcMonth.setText(Integer.toString(9));
687         snowcMPanel.add(snowcMonth,1);
688         snowCPanel.add(snowcMPanel);
689
```

```
690         snowCResetB = new JButton("Reset.");
691         snowCResetB.addActionListener(this);
692         snowCPanel.add(snowCResetB);
693
694         JPanel aspectControlPanel = new JPanel(new GridLayout(2,2));
695         aspectControlPanel.setPreferredSize(new Dimension(115,35));
696         JLabel aspAng = new JLabel("angle:");
697         aspAng.setFont(new Font("Times New Roman",Font.BOLD,11));
698         aspectControlPanel.add(aspAng,0);
699         aspectA = new JTextField(3);
700         aspectA.setText(Integer.toString(135));
701         aspectControlPanel.add(aspectA,1);
702         JLabel aspInf = new JLabel("influence:");
703         aspInf.setFont(new Font("Times New Roman",Font.BOLD,11));
704         aspectControlPanel.add(aspInf,2);
705         aspectI = new JTextField(3);
706         aspectI.setText(Integer.toString(0));
707         aspectControlPanel.add(aspectI,3);
708         aspectPanel.add(aspectControlPanel);
709
710     /*------Panel relating to Mass Balance & ELA calculation,
711      *including an independent lapse rate calculator*/
712         JPanel massbContainer = new JPanel(new BorderLayout());
713         massbContainer.setPreferredSize(new Dimension(infoPWidth-4,100));
714         pInfoIcelandModel.add(massbContainer,BorderLayout.SOUTH);
715
716         massbContainer.add(new JLabel("   Mass Balance &
    ELA:"),BorderLayout.NORTH);
717         JPanel massbPanel = new JPanel();
718         massbPanel.setBorder(BorderFactory.createEtchedBorder());
719         massbPanel.setPreferredSize(new Dimension(infoPWidth-4,45));
720         massbContainer.add(massbPanel,1);
721
722         snowCalcB = new JButton("Snowfall");
723         snowCalcB.addActionListener(this);
724         massbPanel.add(snowCalcB);
725         JPanel snowTPanel = new JPanel(new GridLayout(2,1));
726         snowTPanel.setPreferredSize(new Dimension(70,35));
727         JLabel stp = new JLabel("limiting T:");
728         stp.setFont(new Font("Times New Roman",Font.BOLD,11));
729         snowTPanel.add(stp,0);
730         limT = new JTextField(5);
731         limT.setText(Float.toString(1f));
732         snowTPanel.add(limT,1);
733         massbPanel.add(snowTPanel);
734
735         massBalB = new JButton("Mass Balance");
736         massBalB.addActionListener(this);
737         massbPanel.add(massBalB);
738         elaOffCalcB = new JButton("ELA Offset");
739         elaOffCalcB.addActionListener(this);
740         massbPanel.add(elaOffCalcB);
741         elaCalcB = new JButton("ELA Height");
742         elaCalcB.addActionListener(this);
743         massbPanel.add(elaCalcB);
744         ela_TopoB = new JButton("Topo-ELA");
745         ela_TopoB.addActionListener(this);
746         massbPanel.add(ela_TopoB);
747         JPanel elaToffPanel = new JPanel(new GridLayout(2,1));
748         elaToffPanel.setPreferredSize(new Dimension(80,35));
749         shiftela = new JButton("shift:");
750         shiftela.setFont(new Font("Times New Roman",Font.BOLD,11));
751         shiftela.addActionListener(this);
752         elaToffPanel.add(shiftela,0);
753         shiftamt = new JTextField(2);
754         shiftamt.setText(Integer.toString(0));
755         elaToffPanel.add(shiftamt,1);
756         massbPanel.add(elaToffPanel);
757
758         lapseB = new JButton("Lapse Rate");
759         lapseB.addActionListener(this);
```



```
760         massbPanel.add(lapseB);
761         focalField = new JTextField(3);
762         focalField.setText(Integer.toString(15));
763         massbPanel.add(focalField);
764         pLapseB = new JButton("P Lapse");
765         pLapseB.addActionListener(this);
766         massbPanel.add(pLapseB);
767
768         JPanel autoControl = new JPanel();
769         autoControl.setBorder(BorderFactory.createEtchedBorder());
770         autoControl.setPreferredSize(new
    Dimension(pCAPWidth,textFieldHeight));
771         autoControlB = new JButton("Auto Control");
772         autoControlB.addActionListener(this);
773         autoControl.add(autoControlB);
774         autoControl.add(new JLabel("source file:"));
775         autoFileInput = new JTextField(10);
776         autoFileInput.setText("autoClim");
777         autoControl.add(autoFileInput);
778         pInfoIcelandModel.add(autoControl,BorderLayout.SOUTH);
779
780         return pInfoIcelandModel;
781     }
782
783     //method only works the first time, to begin the model from scatch
784     public void start() throws Exception
785     {
786         if(firstModelRun==true)
787         {
788             messageOutput.append("\n"+"initial DEM imported");
789             getAreaValues();
790             resetModel("testfolder",xllD,yllD,xurD,yurD,4);
791             messageOutput.append("\n"+"NEW ClimateControl CREATED - blank
    start");
792             firstModelRun=false;
793         }
794     }
795
796     public void actionPerformed(ActionEvent ae)
797     {
798         String arg = ae.getActionCommand();
799         Object source = ae.getSource();
800         float lt,pddK,pddVeg,ai,ssf,ssp;
801         double area;
802         int mn,kc,fr,sa,aa,ssm,ssr;
803         saveFileFlag = saveCB.isSelected();
804         presetReset(arg);
805         if ("Exit".equals(arg))
806         {
807             closeDown();
808         }
809         else if ("View Settings".equals(arg))
810         {
811             Object[] possibleValues = { "None", "Rivers","Rivers & Sandar","I
    Margins","Ice Area","Rivers & Ice Margins","Rivers & Ice Area","All
    Features"};
812             Object selectedValue = JOptionPane.showInputDialog(null,"Select
    view setting",
813             "View Settings",JOptionPane.INFORMATION_MESSAGE,
    null,possibleValues, possibleValues[1]);
814             System.out.println("Selected Object: "+(String)selectedValue);
815             if((selectedValue.equals("Rivers"))||(selectedValue.equals("River
    & Ice Margins"))||(selectedValue.equals("Rivers & Ice Area"))
816             ||(selectedValue.equals("Rivers &
    Sandar"))||(selectedValue.equals("All Features")))
817             {
    showRiversAndLakes=true;showSandar=false;showIceMargins=false;showIceArea=
    alse;}
818             if((selectedValue.equals("Ice
    Margins"))||(selectedValue.equals("Rivers & Ice Margins"))
819             ||(selectedValue.equals("All Features")))
```

```
820             {  showIceMargins=true;showIceArea=false;}
821             if((selectedValue.equals("Ice
    Area"))||(selectedValue.equals("Rivers & Ice Area"))
822             ||(selectedValue.equals("All Features")))
823             {  showIceArea=true;showIceMargins=false;}
824             if((selectedValue.equals("Rivers & Sandar"))
825             ||(selectedValue.equals("All Features")))
826             showSandar=true;
827             if(selectedValue.equals("None"))
828             {
    showRiversAndLakes=false;showIceMargins=false;showIceArea=false;showSandar
    false;}
829
830         }
831         else if ("Drainage and Basin Settings".equals(arg))
832         {
833             JFrame jf = new JFrame("Drainage and Basin Settings");
834             Container cp = jf.getContentPane();
835             optionIndex=1;
836             optionObjects = new Object[3];
837             cp.add(new JLabel("Enter Drainage and Basin Parameters:"));
838             cp.setLayout(new BorderLayout());
839             jf.addWindowListener(new OptionMonitor());
840             JPanel iTopLevel = new JPanel(new GridLayout(3,2));
841             iTopLevel.setPreferredSize(new Dimension( 200,textFieldHeight*3))
842             cp.add(iTopLevel);
843             iTopLevel.add(new JLabel("Min River Area:"),0);
844             drainageMin = new JTextField(6);
845             drainageMin.setText(Integer.toString(drainMin));
846             iTopLevel.add(drainageMin,1);
847             optionObjects[0] = drainageMin;
848             iTopLevel.add(new JLabel("Min Basin Area:"),2);
849             basinMin = new JTextField(6);
850             basinMin.setText(Integer.toString(basMin));
851             iTopLevel.add(basinMin,3);
852             optionObjects[1] = basinMin;
853             iTopLevel.add(new JLabel("Max Basin Area:"),4);
854             basinMax = new JTextField(6);
855             basinMax.setText(Integer.toString(basMax));
856             iTopLevel.add(basinMax,5);
857             optionObjects[2] = basinMax;
858             jf.pack();
859             jf.setVisible(true);
860
861         }
862         else if ("Show Drainage Vectors".equals(arg))
863         {
864             if(showDrainageVectors==false)
865                 showDrainageVectors=true;
866             else
867                 showDrainageVectors=false;
868             System.out.println("drain v: "+showDrainageVectors);
869         }
870         else if ("basins".equals(arg))
871         {
872             messageOutput.append("\n"+"generating basins...");
873             try
874             {
875                 dispR = getDrainageBasins(drainMin,false,basMin,basMax);
876                 Raster tRaster = new
    Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
877                 tRaster.setScaleType("iceland_topo4");
878                 tRaster.setSea(true);
879                 overlay(tRaster,dispR,1);
880                 updateInfo(dispR,"basins");
881             }
882             catch(Exception e2)
883             {
884                 messageOutput.append("\n"+"Fill failed.");
885             }
886         }
```



```
887         else if ("glimmer data".equals(arg))
888         {
889             messageOutput.append("\n"+"saving glimmer data...");
890             try
891             {
892                 if(tempOrPrecipImported==false)
893                 {
894                     climateControl.initialiseIMS();
895                     tempOrPrecipImported=true;
896                 }
897                 fr=Integer.parseInt(focalField.getText());
898                 dispR = climateControl.getGlimmerData(fr,hiLoFlag);
899                 messageOutput.append("\n"+"glimmer data saved...");
900                 updateInfo(dispR,"Lapse Rate");
901             }
902             catch(Exception e2)
903             {
904                 messageOutput.append("\n"+"Failed to save glimmer data.");
905             }
906         }
907         else if ("Show Area".equals(arg))
908         {
909             messageOutput.append("\n"+"Highlight area on map...");
910             try
911             {
912                 start();
913                 if(areaValuesChanged())
914                 {
915                     getAreaValues();
916                     resetModel("custom_area",xll,yll,xur,yur,2);
917                     messageOutput.append("\n"+"NEW ClimateControl CREATED - new
    area selected");
918                 }
919                 dispR = climateControl.displayArea(true);
920                 updateInfo(dispR,dirName);
921                 climateControl.restoreTopoRaster();
922             }
923             catch(Exception e2)
924             {
925                 messageOutput.append("\n"+"Show Area failed.");
926             }
927         }
928         else if ("Get Topo".equals(arg))
929         {
930             try
931             {
932                 start();
933                 if(areaValuesChanged())
934                 {
935                     climateControl = new
    ClimateControl(this,hiLoFlag,messageOutput);
936                     getAreaValues();
937                     resetModel("custom_area",xll,yll,xur,yur,2);
938                     messageOutput.append("\n"+"NEW ClimateControl CREATED - new
    area selected");
939                 }
940                 dispR = climateControl.displayArea(false);
941                 updateInfo(dispR,dirName);
942                 topoB.setBackground(Color.GREEN);
943 //              this.iTrans = rpv.getTrans();
944 //              rpv.addMouseMotionListener(new MouseMoveMonitor());
945             }
946             catch(Exception e2)
947             {
948                 messageOutput.append("\n"+"Topography display failed.");
949             }
950         }
951         else if (("Get Temps".equals(arg))||("shiftT".equals(arg)))
952         {
953             messageOutput.append("\n"+"Extracting temperature data...");
954             try
```

```
955             {
956                start();
957                if(areaValuesChanged())
958                {
959                   climateControl = new
    ClimateControl(this,hiLoFlag,messageOutput);
960                   getAreaValues();
961                   resetModel("custom_area",xll,yll,xur,yur,2);
962
963                   messageOutput.append("\n"+"NEW ClimateControl CREATED - n
    area selected");
964                }
965             //   ****** Final initialisation of the model - area selection finalise

966                if(tempOrPrecipImported==false)
967                {
968                   climateControl.initialiseIMS();
969                   tempOrPrecipImported=true;
970                }
971                ts = Float.parseFloat(shiftTamt.getText());

972                dispR = climateControl.dispTemp(0,ts,hiLoFlag);
973                if("shiftT".equals(arg))
974                {
975                   if(ts<6)
976                      ts+=1;
977                   else if(ts==6)
978                      ts=-5;
979                }
980                updateInfo(dispR,"Temp "+months[0]+": "+ts+"C");
981                shiftTamt.setText(Float.toString(ts));
982                tempB.setBackground(Color.GREEN);
983                setMonthComboBox(tempCB,0);
984                tempCB.setBackground(Color.YELLOW);
985             }
986             catch(Exception e2)
987             {
988                messageOutput.append("\n"+"Get Temps failed");
989             }
990          }
991          else if(source.equals(tempCB))
992          {
993             try
994             {
995                mn=tempCB.getSelectedIndex();
996                ts = Float.parseFloat(shiftTamt.getText());

997                dispR = climateControl.dispTemp(mn,ts,hiLoFlag);
998                if((mn>=1)&&(mn<=12))
999                {
1000                  messageOutput.append("\n"+"Displaying Temperature for
    "+months[mn]);
1001                  updateInfo(dispR,"Temp: "+ts+"C, "+months[mn]);
1002               }
1003               else
1004               {
1005                  messageOutput.append("\n"+"Displaying Annual Temp");
1006                  updateInfo(dispR,"Annual Temp: "+ts+"C");
1007               }
1008               if(mn==12)
1009                  mn=0;
1010               tempCB.setSelectedIndex(mn);
1011            }
1012            catch(Exception e2)
1013            {
1014               messageOutput.append("\n"+"Display Temp failed");
1015            }
1016         }
1017         else if (source.equals(setKB))
1018         {
1019            messageOutput.append("\n"+"Setting complex equation K values...
```



```
1020           try
1021           {
1022              kc=tkChoice.getSelectedIndex()+1;
1023              kc = climateControl.setKVals(kc);
1024              tkChoice.select(kc-1);
1025              kSet=true;
1026              setKB.setBackground(Color.GREEN);
1027           }
1028           catch(Exception e2)
1029           {
1030              messageOutput.append("\n"+"Setting K values failed");
1031           }
1032        }
1033
1034        else if (("Get Precip".equals(arg))||("shiftP".equals(arg)))
1035        {
1036           messageOutput.append("\n"+"Extracting precipitation data...");
1037           try
1038           {
1039              start();
1040              if(areaValuesChanged())
1041              {
1042                 climateControl = new
    ClimateControl(this,hiLoFlag,messageOutput);
1043                 getAreaValues();
1044                 resetModel("custom area",xll,yll,xur,yur,2);
1045                 messageOutput.append("\n"+"NEW ClimateControl CREATED - n
    area selected");
1046              }
1047              if(tempOrPrecipImported==false)
1048              {
1049                 climateControl.initialiseIMS();
1050                 tempOrPrecipImported=true;
1051              }
1052              ps = Integer.parseInt(shiftPamt.getText());

1053              dispR = climateControl.displayPrecip(0,ps,hiLoFlag);
1054              if("shiftP".equals(arg))
1055              {
1056                 if(ps<50)
1057                    ps+=5;
1058                 else if(ps==55)
1059                    ps=-50;
1060              }
1061              updateInfo(dispR,"Annual Precip: "+(ps+100)+"%");
1062              shiftPamt.setText(Integer.toString(ps));
1063              precipB.setBackground(Color.GREEN);
1064              setMonthComboBox(precipCB,0);
1065              precipCB.setBackground(Color.YELLOW);
1066           }
1067           catch(Exception e2)
1068           {
1069              messageOutput.append("\n"+"Precip raster preparation failed"
1070           }
1071        }
1072        //Precip display
1073        else if(source.equals(precipCB))
1074        {
1075           try
1076           {
1077              mn=precipCB.getSelectedIndex();
1078              ps = Integer.parseInt(shiftPamt.getText());

1079              dispR = climateControl.displayPrecip(mn,ps,hiLoFlag);
1080              if((mn>=1)&&(mn<=12))
1081              {
1082                 messageOutput.append("\n"+"Displaying Precipitation for
    "+months[mn]);
1083                 updateInfo(dispR,"Precip: "+(ps+100)+"%, "+months[mn]);
1084              }
1085              else
```

```
1086                {
1087                   messageOutput.append("\n"+"Displaying Annual Precip");
1088                   updateInfo(dispR,"Annual PPT: "+(ps+100)+"%");
1089                }
1090                if(mn==12)
1091                   mn=0;
1092                precipCB.setSelectedIndex(mn);
1093             }
1094             catch(Exception e2)
1095             {
1096                messageOutput.append("\n"+"Display Precip failed");
1097             }
1098          }
1099
1100          //PDD display
1101          else if (("snow".equals(arg))|| ("veg".equals(arg)))
1102          {
1103             messageOutput.append("\n"+"Calculating PDD sum");
1104             rasters = new Vector();
1105             try
1106             {
1107                if("snow".equals(arg))
1108                   snowVegFlag=true;
1109                else
1110                   snowVegFlag =false;
1111                pddVeg = Float.parseFloat(pddVegLim.getText());
1112                pddK = Float.parseFloat(pddSnowIce.getText());
1113    //sunShed factor
1114                ssf=Float.parseFloat(fSShed.getText());
1115                dispR = climateControl.pddManager(pddK,snowVegFlag,pddVeg,ss
1116                Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1117                tRaster.setScaleType("iceland_topo4");
1118                tRaster.setSea(true);
1119                overlay(tRaster,dispR,1);
1120                if(snowVegFlag==false)
1121                {
1122                   area = dispR.calculateArea(0,true);
1123                   messageOutput.append("\n"+"PDD area for "+pddVeg+" = "+ar
      sq km. T: "+ts);
1124                   dispR = climateControl.pddManager(pddK,snowVegFlag,7.5f,s
1125                   dispR.setScaleType("2nd PDD");
1126                   overlayAnother(dispR,1);
1127                }
1128                updateInfo(dispR,"PDD - annual");
1129                if("snow".equals(arg))
1130                   pddGoB.setBackground(Color.GREEN);
1131                else
1132                   pddVegB.setBackground(Color.GREEN);
1133                pddCB.setBackground(Color.YELLOW);
1134             }
1135             catch(Exception e2)
1136             {
1137                messageOutput.append("\n"+"PDD manager failed");
1138             }
1139          }
1140
1141          else if(source.equals(pddCB))
1142          {
1143             rasters = new Vector();
1144             try
1145             {
1146                mn=pddCB.getSelectedIndex();
1147                dispR = climateControl.dispPdd(mn,snowVegFlag);
1148                Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1149                tRaster.setScaleType("iceland_topo4");
1150                tRaster.setSea(true);
1151                overlay(tRaster,dispR,1);
1152                if((mn>=1)&&(mn<=12))
1153                {
```



```
1154                   messageOutput.append("\n"+"Displaying PDD for "+months[mn
1155                   updateInfo(dispR,"PDD - "+months[mn]);
1156                }
1157                else
1158                {
1159                   messageOutput.append("\n"+"Displaying Annual PDD");
1160                   updateInfo(dispR,"PDD - annual");
1161                }
1162                if(mn==12)
1163                   mn=0;
1164                pddCB.setSelectedIndex(mn);
1165             }
1166             catch(Exception e2)
1167             {
1168                messageOutput.append("\n"+"Display failed");
1169             }
1170          }
1171
1172          else if ("Growing Season length".equals(arg))
1173          {
1174             messageOutput.append("\n"+"Calculating growing season...");
1175             try
1176             {
1177                pddVeg = Float.parseFloat(pddVegLim.getText());
1178                dispR = climateControl.findGS(pddVeg);
1179                Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1180                tRaster.setScaleType("iceland_topo4");
1181                tRaster.setSea(true);
1182                overlay(tRaster,dispR,1);
1183                updateInfo(dispR,"Growing Season");
1184                growSCalcB.setBackground(Color.GREEN);
1185                growSStartB.setBackground(Color.GREEN);
1186                growSEndB.setBackground(Color.GREEN);
1187                growSMeanTB.setBackground(Color.GREEN);
1188             }
1189             catch(Exception e2)
1190             {
1191                messageOutput.append("\n"+"Snowfall calculation failed");
1192             }
1193          }
1194
1195          else if("Start of GS".equals(arg))
1196          {
1197             try
1198             {
1199                dispR = climateControl.displayGS(true);
1200                Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1201                tRaster.setScaleType("iceland_topo4");
1202                tRaster.setSea(true);
1203                overlay(tRaster,dispR,1);
1204                updateInfo(dispR,"Growing Season Start");
1205             }
1206             catch(Exception e2)
1207             {
1208                messageOutput.append("\n"+"GS calculation failed");
1209             }
1210          }
1211
1212          else if("End of GS".equals(arg))
1213          {
1214             try
1215             {
1216                dispR = climateControl.displayGS(false);
1217                Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1218                tRaster.setScaleType("iceland_topo4");
1219                tRaster.setSea(true);
1220                overlay(tRaster,dispR,1);
1221                updateInfo(dispR,"Growing Season End");
```

```
1222            }
1223            catch(Exception e2)
1224            {
1225               messageOutput.append("\n"+"GS calculation failed");
1226            }
1227         }
1228         else if("Mean T of GS".equals(arg))
1229         {
1230            try
1231            {
1232               dispR = climateControl.displayGSmeanT();
1233               updateInfo(dispR,"Growing Season Mean T");
1234            }
1235            catch(Exception e2)
1236            {
1237               messageOutput.append("\n"+"GS calculation failed");
1238            }
1239         }
1240
1241         else if ("Snowcover".equals(arg))
1242         {
1243            try
1244            {
1245               mn=Integer.parseInt(snowcMonth.getText());
1246               lt=Float.parseFloat(limT.getText());
1247   //aspect factors
1248               aa=Integer.parseInt(aspectA.getText());
1249               ai = Float.parseFloat(aspectI.getText());
1250               for(int sc=1;sc<37;sc++)
1251               {
1252                  dispR = climateControl.findSnowcover(mn,lt,aa,ai);
1253                  Raster vegR = climateControl.dispPdd(mn,false);
1254                  vegR.setScaleType("pdd_veg_month");
1255                  Raster tRaster = new
     Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1256                  tRaster.setScaleType("iceland_topo4");
1257                  tRaster.setSea(true);
1258                  overlay(tRaster,vegR,1);
1259                  overlayAnother(dispR,1);
1260                  messageOutput.append("\n"+"Displaying Snowcover for
     "+months[mn]);
1261                  updateInfo(dispR,"Snowcover - "+months[mn]);
1262                  if((mn<12)&&(mn!=0))
1263                     mn+=1;
1264                  else if(mn==12)
1265                     mn=1;
1266                  snowcMonth.setText(Integer.toString(mn));
1267               }
1268               snowCCalcB.setBackground(Color.BLUE);
1269            }
1270            catch(Exception e2)
1271            {
1272               messageOutput.append("\n"+"Snowcover failed");
1273            }
1274         }
1275
1276         else if ("Reset.".equals(arg))
1277         {
1278            try
1279            {
1280               climateControl.resetSnowcover();
1281               snowcMonth.setText(Integer.toString(9));
1282               snowCCalcB.setBackground(Color.GRAY);
1283            }
1284            catch(Exception e2)
1285            {
1286               messageOutput.append("\n"+"Snowcover reset failed");
1287            }
1288         }
1289
1290         else if (("Sun Visibility".equals(arg))||(source.equals(sunsCB)))
```



```
1291         {
1292            messageOutput.append("\n"+"Calculating sunShed...");
1293            try
1294            {
1295               ssm=sunsCB.getSelectedIndex();
1296               ssr=Integer.parseInt(sSRadius.getText());
1297               dispR = climateControl.findSunShed(ssm,ssr,hiLoFlag);
1298   //          sSRadius.setText(""+climateControl.getSunShedRadius());
1299               if((ssm<13)&&(ssm>0))
1300                  updateInfo(dispR,"Sun Visibility: "+months[ssm]);
1301               else if(ssm==0)
1302                  updateInfo(dispR,"Sun Visibility: Annual");
1303               if(ssm==12)
1304                  ssm=0;
1305               sunShedB.setBackground(Color.GREEN);
1306               sunsCB.setSelectedIndex(ssm);
1307               sunsCB.setBackground(Color.YELLOW);
1308            }
1309            catch(Exception e2)
1310            {
1311               messageOutput.append("\n"+"SunShed calculation failed");
1312            }
1313         }
1314
1315         else if ("Snowfall".equals(arg))
1316         {
1317            messageOutput.append("\n"+"Calculating snowfall...");
1318            try
1319            {
1320               lt=Float.parseFloat(limT.getText());
1321               pddK = Float.parseFloat(pddSnowIce.getText());
1322   //aspect factors
1323               aa=Integer.parseInt(aspectA.getText());
1324               ai = Float.parseFloat(aspectI.getText());
1325               dispR = climateControl.findSnowfall(lt,pddK,aa,ai);
1326               updateInfo(dispR,"Total Snowfall");
1327               snowCalcB.setBackground(Color.GREEN);
1328            }
1329            catch(Exception e2)
1330            {
1331               messageOutput.append("\n"+"Snowfall calculation failed");
1332            }
1333         }
1334
1335         else if ("Mass Balance".equals(arg))
1336         {
1337            messageOutput.append("\n"+"Calculating mass balance...");
1338            try
1339            {
1340               dispR = climateControl.findMassBalance();
1341               Raster tRaster = new
     Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1342               tRaster.setScaleType("iceland_topo");
1343               tRaster.setSea(true);
1344               overlay(tRaster,dispR,1);
1345               updateInfo(dispR,"Mass Balance");
1346               massBalB.setBackground(Color.GREEN);
1347            }
1348            catch(Exception e2)
1349            {
1350               messageOutput.append("\n"+"Mass balance calculation failed")
1351            }
1352
1353         }
1354
1355         else if ("ELA Offset".equals(arg))
1356         {
1357            messageOutput.append("\n"+"***OFFSET HILOFLAG: "+hiLoFlag);
1358            messageOutput.append("\n"+"Calculating ELA offset from surface
     temp...");
1359            try
```

```
1360            {
1361               lt= Float.parseFloat(limT.getText());
1362               pddK = Float.parseFloat(pddSnowIce.getText());
1363   //aspect factors
1364               aa=Integer.parseInt(aspectA.getText());
1365               ai = Float.parseFloat(aspectI.getText());

1366   //sunShed factors
1367               ssf=Float.parseFloat(fSShed.getText());
1368               ssp=Float.parseFloat(pddSShed.getText());
1369               dispR = climateControl.elaOffset(lt,pddK,aa,ai,ssf);
1370               updateInfo(dispR,"ELA Offset");
1371               elaOffCalcB.setBackground(Color.GREEN);
1372            }
1373            catch(Exception e2)
1374            {
1375               messageOutput.append("\n"+"ELA Offset calculation failed");
1376            }
1377         }
1378
1379         else if ("ELA Height".equals(arg))
1380         {
1381            messageOutput.append("\n"+"***HILOFLAG: "+hiLoFlag);
1382            messageOutput.append("\n"+"Calculating ELA from offset, DEM and
      surface temp...");
1383            try
1384            {
1385               fr=Integer.parseInt(focalField.getText());
1386               dispR = climateControl.elaHeight(fr,false,hiLoFlag);
1387               updateInfo(dispR,"ELA height");
1388               elaCalcB.setBackground(Color.GREEN);
1389            }
1390            catch(Exception e2)
1391            {
1392               messageOutput.append("\n"+"ELA calculation failed");
1393            }
1394         }
1395
1396         else if (("Topo-ELA".equals(arg))||("shift:".equals(arg)))
1397         {
1398            messageOutput.append("\n"+"Calculating topography-ELA differenc
      map...");
1399            try
1400            {
1401               sa =Integer.parseInt(shiftamt.getText());
1402               dispR = climateControl.findTopoELA(sa);
1403               if("shift:".equals(arg))
1404               {
1405                  if(sa<500)
1406                  sa+=50;
1407                  else if(sa>=500)
1408                  sa=-500;
1409                  shiftamt.setText(Integer.toString(sa));
1410               }
1411               updateInfo(dispR,"Topo-ELA");
1412               ela_TopoB.setBackground(Color.GREEN);
1413            }
1414            catch(Exception e2)
1415            {
1416               messageOutput.append("\n"+"topography-ELA calculation failed
1417            }
1418
1419         }
1420
1421         else if ("Lapse Rate".equals(arg))
1422         {
1423            messageOutput.append("\n"+"Calculating temperature lapse rate..
1424            try
1425            {
1426               if(tempOrPrecipImported==false)
1427               {
```



```
1428                  climateControl.initialiseIMS();
1429                  tempOrPrecipImported=true;
1430               }
1431               fr=Integer.parseInt(focalField.getText());
1432               dispR = climateControl.getLapseRate(fr,hiLoFlag,true);
1433               updateInfo(dispR,"T Lapse Rate");
1434               lapseB.setBackground(Color.GREEN);
1435            }
1436            catch(Exception e2)
1437            {
1438               messageOutput.append("\n"+"Display lapse rate failed");
1439            }
1440         }
1441         else if ("P Lapse".equals(arg))
1442         {
1443            messageOutput.append("\n"+"Calculating precip lapse rate...");
1444            try
1445            {
1446               if(tempOrPrecipImported==false)
1447               {
1448                  climateControl.initialiseIMS();
1449                  tempOrPrecipImported=true;
1450               }
1451               fr=Integer.parseInt(focalField.getText());
1452               dispR = climateControl.getLapseRate(fr,hiLoFlag,false);
1453               updateInfo(dispR,"P Lapse Rate");
1454               pLapseB.setBackground(Color.GREEN);
1455            }
1456            catch(Exception e2)
1457            {
1458               messageOutput.append("\n"+"Display lapse rate failed");
1459            }
1460         }
1461         else if ("AIG".equals(arg))
1462         {
1463            messageOutput.append("\n"+"Calculating AIG...");
1464            try
1465            {
1466               if(tempOrPrecipImported==false)
1467               {
1468                  climateControl.initialiseIMS();
1469                  tempOrPrecipImported=true;
1470               }
1471
1472               fr=Integer.parseInt(focalField.getText());
1473               dispR = climateControl.calculateAIG(fr,hiLoFlag);
1474               updateInfo(dispR,"Alt of Inst Glac");
1475               pLapseB.setBackground(Color.GREEN);
1476            }
1477            catch(Exception e2)
1478            {
1479               messageOutput.append("\n"+"Display lapse rate failed");
1480            }
1481         }
1482         else if ("GBS".equals(arg))
1483         {
1484            messageOutput.append("\n"+"Calculating GBS...");
1485            try
1486            {
1487               if(tempOrPrecipImported==false)
1488               {
1489                  climateControl.initialiseIMS();
1490                  tempOrPrecipImported=true;
1491               }
1492
1493               fr=Integer.parseInt(focalField.getText());
1494               dispR = climateControl.calculateGBS(fr,hiLoFlag);
1495               updateInfo(dispR,"Glacier Build Sens");
1496               pLapseB.setBackground(Color.GREEN);
1497            }
1498            catch(Exception e2)
```

```
1499            {
1500               messageOutput.append("\n"+"Display lapse rate failed");
1501            }
1502         }
1503         else if ("LOW".equals(arg))
1504            setHigh();
1505         else if ("HIGH".equals(arg))
1506            setLow();
1507
1508         else if ("Auto Control".equals(arg))
1509         {
1510            messageOutput.append("\n"+"Automated control started");
1511            try
1512            {
1513               start();
1514               String autoFile =
root_dir+"config_files/"+(String)autoFileInput.getText()+".csv";
1515               System.out.println(autoFile);
1516               String[] files;
1517               if("autoMulti".equals((String)autoFileInput.getText()))
1518               {
1519                  Vector autoFiles = SimpleIO.getLines(autoFile);
1520                  files = new String[autoFiles.size()];
1521                  for(int s=0;s<autoFiles.size();s++)
1522
files[s]=(String)SimpleIO.getLineElement(autoFiles,s,0,",");
1523               }
1524               else
1525               {
1526                  files = new String[1];
1527                  files[0]=(String)autoFileInput.getText();
1528               }
1529               for(int s=0;s<files.length;s++)
1530               {
1531                  autoFile = root_dir+"config_files/"+files[s]+".csv";
1532                  Vector autoOutput = new Vector();
1533                  Vector autoControls = SimpleIO.getLines(autoFile);
1534
if(((String)SimpleIO.getLineElement(autoControls,0,0,",")).compareTo("no
)==0)
1535                  {
1536                     System.out.println("area vals");
1537                     getAreaValues();
1538                     climateControl = new
ClimateControl(this,hiLoFlag,messageOutput);
1539                  }
1540                  else
if(((String)SimpleIO.getLineElement(autoControls,0,0,",")).compareTo("pr
t")==0)
1541                  {
1542                     String presetName =
(String)SimpleIO.getLineElement(autoControls,0,1,",");
1543                     System.out.println("preset vals: "+presetName);
1544                     presetReset(presetName);
1545
if("high".equals((String)SimpleIO.getLineElement(autoControls,0,2,",")))
1546                        setHigh();
1547                     else
1548                        setLow();
1549                  }
1550                  else
1551                  {
1552                     System.out.println("file vals");
1553                     dirName =
(String)SimpleIO.getLineElement(autoControls,0,0,",");
1554
xll=Double.parseDouble((String)SimpleIO.getLineElement(autoControls,0,2,
));
1555
yll=Double.parseDouble((String)SimpleIO.getLineElement(autoControls,0,3,
));
```



```
1556
xur=Double.parseDouble((String)SimpleIO.getLineElement(autoControls,0,4,
));
1557
yur=Double.parseDouble((String)SimpleIO.getLineElement(autoControls,0,5,
));
1558                     setAreaValues();
1559
if("high".equals((String)SimpleIO.getLineElement(autoControls,0,1,",")))
1560                        setHigh();
1561                     else
1562                        setLow();
1563                  }
1564                  System.gc();
1565                  climateControl.initialiseIMS();
1566                  dispR = climateControl.displayArea(false);
1567                  updateInfo(dispR,arg);
1568                  int lineIndex=1;
1569                  while(lineIndex<autoControls.size())
1570                  {
1571 //                  System.out.println("XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
"+rt.totalMemory()+" "+rt.maxMemory()+" "+rt.freeMemory());
1572 //                  SimpleIO.readKeyboard();
1573                     String keyText =
(String)SimpleIO.getLineElement(autoControls,lineIndex,0,",");
1574                     if("set".equals(keyText))
1575                     {
1576                        messageOutput.append("\n"+"Extracting temperature
data...");
1577                        ts =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
1578                        dispR = climateControl.dispTemp(0,ts,hiLoFlag);
1579                        shiftTamt.setText(Float.toString(ts));
1580                        updateInfo(dispR,"Temp");
1581                        ps =
Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
1582                        dispR = climateControl.displayPrecip(0,ps,hiLoFlag)
1583                        shiftPamt.setText(Float.toString(ps));
1584                        updateInfo(dispR,"Precip");
1585
1586                     }
1587                     if("topo".equals(keyText))
1588                     {
1589                        dispR = climateControl.displayArea(false);
1590                        updateInfo(dispR,dirName);
1591                     }
1592                     if("images".equals(keyText))
1593                     {
1594                        imageScaling =
(double)Float.parseFloat((String)SimpleIO.getLineElement(autoControls,li
ndex,1,","));
1595                        saveFileFlag=true;
1596                     }
1597                     if("noimages".equals(keyText))
1598                     {
1599                        saveFileFlag=false;
1600                     }
1601
1602                     if("edits".equals(keyText))
1603                     {
1604                        manageEditsLayer(true);
1605                     }
1606                     if("rivers".equals(keyText))
1607                     {
1608                        showRiversAndLakes=true;
1609
drainMin=Integer.parseInt((String)SimpleIO.getLineElement(autoControls,l
Index,1,","));
1610                        dChanged=true;
```

```
1611                        Raster tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1612                        tRaster.setScaleType("iceland_topo");
1613                        tRaster.setSea(true);
1614                        updateInfo(tRaster,"Topo & Drainage");
1615                    }
1616                    if("basins".equals(keyText))
1617                    {
1618                        dispR = climateControl.displayArea(false);
1619
dispR.setParameters(root_dir+dirName,"topo",dirName+"_topo","iceland_top
true);
1620                        basMin =
Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
1621                        basMax =
Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
1622                        dispR =
getDrainageBasins(drainMin,false,basMin,basMax);
1623                        Raster tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1624                        tRaster.setScaleType("iceland_topo4");
1625                        tRaster.setSea(true);
1626                        overlay(tRaster,dispR,1);
1627                        updateInfo(dispR,dirName);
1628                    }
1629                    if("icebounds".equals(keyText))
1630                    {  showIceMargins=true;showIceArea=false;}
1631                    if("icearea".equals(keyText))
1632                    {  showIceArea=true;showIceMargins=false;}
1633                    if("sandar".equals(keyText))
1634                        showSandar=true;
1635                    if("nosandar".equals(keyText))
1636                        showSandar=false;
1637                    if("nofeatures".equals(keyText))
1638                    {
showIceArea=false;showIceMargins=false;showSandar=false;showRiversAndLak
false;}
1639                    if("trimtoice".equals(keyText))
1640                        trimToIce=true;
1641                    if("notrimtoice".equals(keyText))
1642                        trimToIce=false;
1643                    if("trimtosandar".equals(keyText))
1644                        trimToSandar=true;
1645                    if("notrimtosandar".equals(keyText))
1646                        trimToSandar=false;
1647                    if("largedrainage".equals(keyText))
1648                        largerDrainage=true;
1649                    if("normaldrainage".equals(keyText))
1650                        largerDrainage=false;
1651                    if("b&w".equals(keyText))
1652                        blackAndWhite=true;
1653                    if("colour".equals(keyText))
1654                        blackAndWhite=false;
1655                    if("setT".equals(keyText))
1656                    {
1657                        ts =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
1658                        messageOutput.append("\n"+"Extracting temperature
data..."+ts);
1659                        shiftTamt.setText(Float.toString(ts));
1660                        dispR = climateControl.dispTemp(0,ts,hiLoFlag);
1661                        updateInfo(dispR,"Temp");
1662                        pack();
1663                        setVisible(true);
1664                    }
1665                    if("setP".equals(keyText))
1666                    {
1667                        ps =
```



```
1667  Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
      ,"));
1668                        dispR = climateControl.displayPrecip(0,ps,hiLoFlag)
1669                        shiftPamt.setText(Float.toString(ps));
1670                        updateInfo(dispR,"Precip");
1671                        pack();
1672                        setVisible(true);
1673                    }
1674                    else if("kset".equals(keyText))
1675                    {
1676                        kc =
      climateControl.setKVals(Integer.parseInt((String)SimpleIO.getLineElement
      toControls,lineIndex,1,",")));
1677                        tkChoice.select(kc-1);
1678                        kSet=true;
1679                    }
1680                    else if("sunvis".equals(keyText))
1681                    {
1682                        dispR =
      climateControl.findSunShed(0,Integer.parseInt((String)SimpleIO.getLineEl
      nt(autoControls,lineIndex,1,",")),hiLoFlag);
1683                        updateInfo(dispR,"sunvis");
1684                        pack();
1685                        setVisible(true);
1686                    }
1687                    else if("pddsnow".equals(keyText))
1688                    {
1689                        pddK =
      Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
      ,"));
1690                        ssf =
      Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
      ,"));
1691                        dispR = climateControl.pddManager(pddK,true,0,ssf);
1692                        Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1693                        tRaster.setScaleType("iceland_topo4");
1694                        tRaster.setSea(true);
1695                        overlay(tRaster,dispR,1);
1696                        updateInfo(dispR,"snow Pdd");
1697                        pack();
1698                        setVisible(true);
1699                    }
1700                    else if("pddveg".equals(keyText))
1701                    {
1702                        pddVeg =
      Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
      ,"));
1703                        dispR = climateControl.pddManager(0,false,pddVeg,0)
1704                        Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1705                        tRaster.setScaleType("iceland_topo3");
1706                        tRaster.setSea(true);
1707                        overlay(tRaster,dispR,1);
1708                        updateInfo(dispR,"Veg PDD");
1709                        pack();
1710                        setVisible(true);
1711                        area = dispR.calculateArea(0,true);
1712                        messageOutput.append("\n"+"PDD area for "+pddVeg+"
      "+area+" sq km. T: "+ts);
1713                        area = dispR.calculateValueArea(0,true);
1714                        messageOutput.append("\n"+"Area without veg for
      "+pddVeg+" = "+area+" sq km. T: "+ts);
1715                    }
1716                    else if("multipddveg".equals(keyText))
1717                    {
1718                        pddVeg =
      Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
      ,"));
1719                        dispR = climateControl.pddManager(0,false,pddVeg,0)
1720                        Raster tRaster = new
```

```
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
                        tRaster.setScaleType("iceland_topo3");
                        tRaster.setSea(true);
                        overlay(tRaster,dispR,1);
                        float pddVeg2 =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        dispR = climateControl.pddManager(0,false,pddVeg2,0
                        dispR.setScaleType("2nd PDD");
                        overlayAnother(dispR,1);
                        updateInfo(dispR,"Veg PDD");
                    }
                    else if("snowfall".equals(keyText))
                    {
                        lt =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        pddK =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
            //aspect factors
                        aa =
Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        ai =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        dispR = climateControl.findSnowfall(lt,pddK,aa,ai);
                        updateInfo(dispR,"Total Snowfall");
                    }
                    else if ("massb".equals(keyText))
                    {
                        dispR = climateControl.findMassBalance();
                        Raster tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
                        tRaster.setScaleType("iceland_topo4");
                        tRaster.setSea(true);
                        overlay(tRaster,dispR,1);
                        updateInfo(dispR,"Mass Balance");

if(SimpleIO.checkFile(root_dir+dirName+"/"+dirName+"_basin_100_2000.asc"
                        {
                            Raster db = new
Raster(root_dir+dirName+"/"+dirName+"_basin_100_2000.asc");

SimpleIO.writeComparisonVals(root_dir+dirName+"/compare_mb_topo.csv",db,
ispR,tRaster,700);
                        }
                    }
                    else if ("veg&massb".equals(keyText))
                    {
                        pddVeg =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        dispR = climateControl.pddManager(0,false,pddVeg,0)
                        Raster tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
                        tRaster.setScaleType("iceland_topo3");
                        tRaster.setSea(true);
                        overlay(tRaster,dispR,1);
                        float pddVeg2 =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        dispR = climateControl.pddManager(0,false,pddVeg2,0
                        dispR.setScaleType("2nd PDD");
                        overlayAnother(dispR,1);
                        dispR = climateControl.findMassBalance();
                        dispR.setScaleType("massb");
                        overlayAnother(dispR,1);
                        updateInfo(dispR,"Veg PDD & mass balance");
                    }
```



```
                    else if ("grows".equals(keyText))
                    {
                        pddVeg =
Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        dispR = climateControl.findGS(pddVeg);
                        Raster tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
                        tRaster.setScaleType("iceland_topo3");
                        tRaster.setSea(true);
                        overlay(tRaster,dispR,1);
                        updateInfo(dispR,"Growing Season length");
                        dispR = climateControl.displayGS(true);
                        tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
                        tRaster.setScaleType("iceland_topo3");
                        tRaster.setSea(true);
                        overlay(tRaster,dispR,1);
                        updateInfo(dispR,"Growing Season Start");
                        dispR = climateControl.displayGS(false);
                        tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
                        tRaster.setScaleType("iceland_topo3");
                        tRaster.setSea(true);
                        overlay(tRaster,dispR,1);
                        updateInfo(dispR,"Growing Season End");
                    }
                    if("lapse".equals(keyText))
                    {

fr=Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineInd
1,","));
                        dispR = climateControl.getLapseRate(fr,hiLoFlag,tru
                        updateInfo(dispR,"T Lapse Rate");
                    }
                    if("plapse".equals(keyText))
                    {

fr=Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineInd
1,","));
                        dispR = climateControl.getLapseRate(fr,hiLoFlag,fal
                        updateInfo(dispR,"P Lapse Rate");
                    }
                    if("glimmer".equals(keyText))
                    {

fr=Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineInd
1,","));
                        dispR = climateControl.getGlimmerData(fr,hiLoFlag);
                        updateInfo(dispR,"Lapse Rate");
                    }
                    else if("hold".equals(keyText))
                    {
                        SimpleIO.readKeyboard();
                    }
                    else if("sensitivityseq".equals(keyText))
                    {
                        System.out.println("sensitivity started...");
                        Raster sensR;
                        dispR = new Raster(dispR,0);
                        pddVeg =
Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
,"));
                        float oldT = 0;
                        for(float tChange=-1.5f;tChange<=1.5f;tChange+=0.75
                        {
                            messageOutput.append("\n"+"Extracting temperatur
data..."+tChange);
                            shiftTamt.setText(Float.toString(tChange));
                            climateControl.dispTemp(0,tChange,hiLoFlag);
                            sensR = climateControl.pddManager(0,false,pddVeg
```

```
1828                        updateInfo(sensR,"PDD");
1829                        int cols = sensR.getXcells();
1830                        int rows = sensR.getYcells();
1831                        for(int i=0;i<cols;i++)
1832                           for(int j=0;j<rows;j++)
1833                           {
1834                              if(sensR.getValue(i,j)==noData)
1835                                 dispR.changeValue(i,j,noData);
1836                              else if(sensR.getValue(i,j)>0)
1837                                 dispR.addValue(i,j,1);
1838                              if((tChange==1.5f)&&(dispR.getValue(i,j)>0
1839                                 dispR.addValue(i,j,1);
1840                           }
1841                        oldT = tChange;
1842                     }
1843
      dispR.setParameters(root_dir+dirName,"Sensitivity",dirName+"_sensitivity
      sensitivity",false);
1844
      SimpleIO.rasterToAscii(root_dir+dirName,dirName+"_sensitivity",dispR);
1845                     Raster tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1846                     tRaster.setScaleType("iceland_topo3");
1847                     tRaster.setSea(true);
1848                     overlay(tRaster,dispR,1);
1849                     updateInfo(dispR,"Landscape Sensitivity");
1850
1851                  }
1852                  else if("climateseq".equals(keyText))
1853                  {
1854                     String
      climateFile=(String)SimpleIO.getLineElement(autoControls,lineIndex,1,","
1855
      pddK=Float.parseFloat((String)SimpleIO.getLineElement(autoControls,lineI
      x,2,","));
1856                     int basinsIn =
      Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lineIndex,
      ,","));
1857                     if(basinsIn==1)
1858                     {
1859
      basMin=Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lin
      dex,4,","));
1860
      basMax=Integer.parseInt((String)SimpleIO.getLineElement(autoControls,lin
      dex,5,","));
1861                     }
1862                     climateFile =
      root_dir+"input_files/"+climateFile+".csv";
1863                     Vector climateData = SimpleIO.getLines(climateFile)
1864                     Raster rAndL,db;
1865                     if(showRiversAndLakes==true)
1866                        rAndL = getDrainageRaster(drainMin,false,true);
1867                     else
1868                        rAndL = new Raster(dispR,0);
1869                     if(basinsIn==1)
1870                        db =
      getDrainageBasins(drainMin,false,basMin,basMax);
1871                     else
1872                        db = new Raster(dispR,0);
1873                     climateControl.climateSeqInitialise(rAndL,db);
1874                     Raster tRaster;
1875                     String numPrefix = "";
1876                     for(int
      iteration=0;iteration<climateData.size();iteration++)
1877                     {
1878                        dispR =
      climateControl.climateSeq(iteration,climateData,pddK);
1879                        if(iteration<10)numPrefix = "000";
1880                        else if(iteration<100)numPrefix = "00";
1881                        else if(iteration<1000)numPrefix = "0";
```



```
1882                        else numPrefix = "";
1883
      dispR.setParameters(root_dir+dirName,"Climate_step_"+numPrefix+iteration
      rName+"_clim_seq_"+iteration,"climseq",false);
1884                        tRaster = new
      Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
1885
1886                        tRaster.setScaleType("iceland_topo3");
1887                        tRaster.setSea(true);
1888                        overlay(tRaster,dispR,1);
1889                        updateInfo(dispR,"Lapse Rate");
1890                     }
1891                     climateControl.exportClimateOutputData();
1892                  }
1893
1894                  lineIndex++;
1895               }
1896            }
1897         }
1898         catch(Exception e2)
1899         {
1900            messageOutput.append("\n"+"Automate controls failed");
1901         }
1902      }
1903      else if ("Reset".equals(arg))
1904         resetModel(defDirName,xllD,yllD,xurD,yurD,3);
1905
1906      else if("Allow Raster Changes".equals(arg))
1907      {
1908         if(allowChanges==true)
1909         {
1910            allowC.setForeground(Color.RED);
1911            allowChanges=false;
1912            messageOutput.append("\n"+"Edit: "+allowChanges);
1913         }
1914         else
      if((allowChanges==false)&&("iceland_topo".equals(dispR.getScaleType())))
1915         {
1916            allowC.setForeground(Color.GREEN);
1917            allowChanges=true;
1918            editPoints=false;
1919            editLines=true;
1920            editAreas=false;
1921            messageOutput.append("\n"+"Edit: "+allowChanges);
1922         }
1923         clickValue = Float.parseFloat(clickVal.getText());
1924      }
1925      else if("Points".equals(arg))
1926      {
1927         editPoints=true;
1928         editLines=false;
1929         editAreas=false;
1930      }
1931      else if("Lines".equals(arg))
1932      {
1933         editPoints=false;
1934         editLines=true;
1935         editAreas=false;
1936      }
1937      else if("Areas".equals(arg))
1938      {
1939         editPoints=false;
1940         editLines=false;
1941         editAreas=true;
1942      }
1943      else if ("Save Edits to Topo".equals(arg))
1944      {
1945         try
1946         {
1947            int xOffset,yOffset;
1948            Raster fullR;
```

```
1949                if(hiLoFlag==true)
1950                   fullR = new Raster(defRoot_dir+"input_files/icedem_250.as
1951                else
1952                   fullR = new Raster(defRoot_dir+"input_files/icedem_cp.asc
1953                xOffset =
(int)((dispR.getXll()-fullR.getXll())/dispR.getCellSize());
1954                yOffset =
(int)((dispR.getYll()-fullR.getYll())/dispR.getCellSize());
1955                System.out.println(xOffset+" "+yOffset+" "+fullR.getXcells()
"+fullR.getYcells()+" "+dispR.getXcells()+" "+dispR.getYcells());
1956                for(int i=xOffset;i<xOffset+dispR.getXcells();i++)
1957                   for(int j=yOffset;j<yOffset+dispR.getYcells();j++)
1958                   {
1959
if((i>=0)&&(j>=0)&&(i<fullR.getXcells())&&(j<fullR.getYcells())&&(dispR.
Value(i-xOffset,j-yOffset)!=noData))
1960
fullR.changeValue(i,j,dispR.getValue(i-xOffset,j-yOffset));
1961                   }
1962                fullR.setParameters(defRoot_dir+dirName,"Saved
Topography",dirName+"_topo","iceland_topo",true);
1963                updateInfo(dispR,"edited Raster?");
1964                if(hiLoFlag==true)
1965
SimpleIO.rasterToAscii("C:/gisdata/input_files","icedem_250",fullR);
1966                else
1967                   SimpleIO.rasterToAscii("C:/gisdata","icedem_cp",fullR);
1968             }
1969             catch(Exception e)
1970             {
1971                messageOutput.append("\n"+"Save edited raster failed");
1972             }
1973          }
1974          else if ("Save Changes to Edits Layer".equals(arg))
1975          {
1976             try
1977             {
1978
SimpleIO.rasterToAscii(root_dir+dirName,dirName+"_edits",editsR);
1979                messageOutput.append("\n"+"Saved edits layer");
1980
1981                int xOffset,yOffset;
1982                Raster fullR;
1983                if(hiLoFlag==true)
1984                {
1985
if(SimpleIO.checkFile(defRoot_dir+"config_files/overall_edits_250.asc")=
lse)
1986                      fullR = new
Raster(defRoot_dir+"config_files/icedem_250.asc",0);
1987                   else
1988                      fullR = new
Raster(defRoot_dir+"config_files/overall_edits_250.asc",200,"N");
1989                }
1990                else
1991                {
1992
if(SimpleIO.checkFile(defRoot_dir+"config_files/overall_edits_cp.asc")==
se)
1993                      fullR = new
Raster(defRoot_dir+"config_files/icedem_cp.asc",0);
1994                   else
1995                      fullR = new
Raster(defRoot_dir+"config_files/overall_edits_cp.asc");
1996                }
1997                xOffset =
(int)((editsR.getXll()-fullR.getXll())/editsR.getCellSize());
1998                yOffset =
(int)((editsR.getYll()-fullR.getYll())/editsR.getCellSize());
1999                System.out.println(xOffset+" "+yOffset+" "+fullR.getXcells()
"+fullR.getYcells()+" "+editsR.getXcells()+" "+editsR.getYcells());
```



```
2000                for(int i=xOffset;i<xOffset+editsR.getXcells();i++)
2001                   for(int j=yOffset;j<yOffset+editsR.getYcells();j++)
2002                   {
2003
if((i>=0)&&(j>=0)&&(i<fullR.getXcells())&&(j<fullR.getYcells())&&(editsR
tValue(i-xOffset,j-yOffset)!=noData))
2004
fullR.changeValue(i,j,editsR.getValue(i-xOffset,j-yOffset));
2005                   }
2006                fullR.setParameters(defRoot_dir+dirName,"Saved
Topography",dirName+"_topo","iceland_topo",true);
2007                updateInfo(dispR,"edited Raster?");
2008                if(hiLoFlag==true)
2009
SimpleIO.rasterToCompAscii("C:/gisdata/config_files","overall_edits_250"
llR);
2010                else
2011
SimpleIO.rasterToAscii("C:/gisdata/config_files","overall_edits_cp",full
             }
2012             catch(Exception e)
2013             {
2014                messageOutput.append("\n"+"Save edited raster failed");
2015             }
2016          }
2017          else if ("Clear Edits Layer".equals(arg))
2018          {
2019             try
2020             {
2021                editsR = new Raster(dispR,0);
2022                manageEditsLayer(true);
2023                updateInfo(dispR,"Edits layer cleared?");
2024             }
2025             catch(Exception e)
2026             {
2027                messageOutput.append("\n"+"Save edited raster failed");
2028             }
2029          }
2030          else if ("Revert to original DEM".equals(arg))
2031          {
2032             try
2033             {
2034                Raster fullR;
2035                Rectangle footP = dispR.getFootprint();
2036                if(hiLoFlag==true)
2037                   fullR = new
Raster(defRoot_dir+"input_files/icedem_250_nosandar.asc",dispR.getYll(),
pR.getYur(),"none");
2038                else
2039                   fullR = new
Raster(defRoot_dir+"input_files/icedem_cp_nosandar.asc",dispR.getYll(),d
R.getYur(),"none");
2040                dispR = fullR.gridclip(footP);
2041                System.out.println("ready to update");
2042                dispR.setParameters(defRoot_dir+dirName,"Reverted
Topography",dirName+"_topo","iceland_topo",true);
2043                updateInfo(dispR,"reverted Raster?");
2044             }
2045             catch(Exception e)
2046             {
2047                messageOutput.append("\n"+"Revert edited raster failed");
2048             }
2049          }
2050          else if ("Reset saves to original DEM".equals(arg))
2051          {
2052             try
2053             {
2054                int xOffset,yOffset;
2055                Raster fullR,revertR;
2056                Rectangle footP = dispR.getFootprint();
2057                if(hiLoFlag==true)
```

```
2058                  fullR = new
Raster(defRoot_dir+"input_files/icedem_250_nosandar.asc",dispR.getYll(),
pR.getYur(),"none");
2059               else
2060                  fullR = new
Raster(defRoot_dir+"input_files/icedem_cp_nosandar.asc",dispR.getYll(),d
R.getYur(),"none");
2061               revertR = fullR.gridclip(footP);
2062               xOffset =
(int)((revertR.getXll()-fullR.getXll())/revertR.getCellSize());
2063               yOffset =
(int)((revertR.getYll()-fullR.getYll())/revertR.getCellSize());
2064               System.out.println(xOffset+" "+yOffset+" "+fullR.getXcells()
"+fullR.getYcells()+" "+revertR.getXcells()+" "+revertR.getYcells());
2065               for(int i=xOffset;i<xOffset+revertR.getXcells();i++)
2066                  for(int j=yOffset;j<yOffset+revertR.getYcells();j++)
2067                  {
2068
if((i>=0)&&(j>=0)&&(i<fullR.getXcells())&&(j<fullR.getYcells()))
2069
fullR.changeValue(i,j,revertR.getValue(i-xOffset,j-yOffset));
2070                     if(i==xOffset)
2071                        System.out.println("updated original raster "+j);
2072                  }
2073               if(hiLoFlag==true)
2074
SimpleIO.rasterToAscii("C:/gisdata/input_files","icedem_250",fullR);
2075               else
2076
SimpleIO.rasterToAscii("C:/gisdata/input_files","icedem_cp",fullR);
2077               dispR.setParameters(defRoot_dir+dirName,"Reverted
Topography",dirName+"_topo","iceland_topo",true);
2078               updateInfo(dispR,"reverted Raster");
2079            }
2080            catch(Exception e)
2081            {
2082               messageOutput.append("\n"+"Revert edited raster failed");
2083            }
2084         }
2085         else if ("Show Edits Layer".equals(arg))
2086         {
2087            if(editsLayer==true)
2088               manageEditsLayer(false);
2089            else
2090               manageEditsLayer(true);
2091         }
2092         else if ("Viking Farms Model".equals(arg))
2093         {
2094            try
2095            {
2096               vikingFarmFlag = true;
2097               pInfo.remove(pInfoIcelandModel);
2098               vfr = new Viking_Farms_Run(this,modelPanel,dispR);
2099               pInfoIcelandModel =
vfr.getVikingPanel(infoPWidth,infoPHeight-selectDispHeight-8);
2100               pInfo.add(pInfoIcelandModel);
2101               pack();
2102               setVisible(true);
2103            }
2104            catch(Exception e)
2105            {
2106               messageOutput.append("\n"+"Viking farms display failed");
2107            }
2108         }
2109         else if ("Iceland Climate Model".equals(arg))
2110         {
2111            try
2112            {
2113               vikingFarmFlag = false;
2114               pInfo.remove(pInfoIcelandModel);
2115               vfr = new Viking_Farms_Run(this,modelPanel,dispR);
```



```
2116               pInfoIcelandModel = buildClimateControls();
2117               pInfo.add(pInfoIcelandModel);
2118               pack();
2119               setVisible(true);
2120            }
2121            catch(Exception e)
2122            {
2123               messageOutput.append("\n"+"Viking farms display failed");
2124            }
2125         }
2126      }
2127
2128      public void manageEditsLayer(boolean controlEdits)
2129      {
2130         try
2131         {
2132            editsLayer=controlEdits;
2133            if(controlEdits==true)
2134            {
2135   //          displayPoints = SimpleIO.readPoints("testPoints_myvatn");
2136               if(editsR==null)
2137               {
2138                  if(hiLoFlag==true)
2139                  {
2140                     editsR = new
Raster(root_dir+"config_files/overall_edits_250.asc",getYll(),getYur(),"
2141                     editsR = editsR.gridclip(new Rectangle(new
Point(xll,yll),new Point(xur,yur)));
2142                  }
2143                  else
2144                  {
2145                     editsR = new
Raster(root_dir+"config_files/overall_edits_cp.asc",getYll(),getYur(),"n
");
2146                     editsR = editsR.gridclip(new Rectangle(new
Point(xll,yll),new Point(xur,yur)));
2147                  }
2148               }
2149               messageOutput.append("\n"+"Edits layer visible");
2150
editsR.setParameters(root_dir+dirName,"Edits",dirName+"_edits","edits",f
e);
2151            }
2152            else
2153            {
2154               messageOutput.append("\n"+"Edits layer not visible");
2155            }
2156         }
2157         catch(Exception e)
2158         {
2159            messageOutput.append("\n"+"Show edits layer failed");
2160         }
2161      }
2162      public void editRaster(float editX,float editY)
2163      {
2164         try
2165         {
2166            if(allowChanges==true)
2167            {
2168               manageEditsLayer(true);
2169               double cellSize = dispR.getCellSize();
2170               double xllCorner = dispR.getXll();
2171               double yllCorner = dispR.getYll();
2172               int x = (int)((editX-xllCorner)/cellSize);
2173               int y = (int)((editY-yllCorner)/cellSize);
2174               editsR.changeValue(x,y,clickValue);
2175               messageOutput.append("\n"+"Edit made?: "+x+","+y+":
"+clickValue);
2176               updateInfo(dispR,"edited Raster?");
2177            }
2178         }
```

```
2179         catch(Exception e)
2180         {
2181            messageOutput.append("\n"+"Edit failed...");
2182         }
2183      }
2184
2185      public void editRaster(Rectangle area)
2186      {
2187         try
2188         {
2189            if(allowChanges==true)
2190            {
2191               manageEditsLayer(true);
2192               double cellSize = dispR.getCellSize();
2193               double xllCorner = dispR.getXll();
2194               double yllCorner = dispR.getYll();
2195               int x1 = (int)((area.getXll()-xllCorner)/cellSize);
2196               int y1 = (int)((area.getYll()-yllCorner)/cellSize);
2197               int x2 = (int)((area.getXur()-xllCorner)/cellSize);
2198               int y2 = (int)((area.getYur()-yllCorner)/cellSize);
2199               for(int i=x1;i<=x2;i++)
2200                  for(int j=y1;j<=y2;j++)
2201                     editsR.changeValue(i,j,clickValue);
2202               messageOutput.append("\n"+"Edit made from: "+x1+","+y1);
2203               messageOutput.append("\n"+"Edit made to: "+x2+","+y2+" using
"+clickValue);
2204               updateInfo(dispR,"edited Raster?");
2205            }
2206         }
2207         catch(Exception e)
2208         {
2209            messageOutput.append("\n"+"Edit failed...");
2210         }
2211      }
2212
2213      public void editRaster(Polygon area,boolean line)
2214      {
2215         try
2216         {
2217            manageEditsLayer(true);
2218            Point point1,point2;
2219            Vector v = area.getPolygon();
2220            if(allowChanges==true)
2221            {
2222               double cellSize = dispR.getCellSize();
2223               double xllCorner = dispR.getXll();
2224               double yllCorner = dispR.getYll();
2225               point1 = (Point)v.elementAt(0);
2226               for(int lp=0;lp<v.size();lp++)
2227               {
2228                  point2 = (Point)v.elementAt(lp);
2229
editsR.changeValue((int)((point1.getX()-xllCorner)/cellSize),(int)((poin
getY()-yllCorner)/cellSize),clickValue);
2230               //
editsR.drawLine(point1.getX(),point1.getY(),point2.getX(),point2.getY(),
0);
2231                  messageOutput.append("\n"+"line started at
"+point1.getX()+","+point1.getY()+": "+clickValue);
2232                  point1 = point2;
2233               }
2234               updateInfo(dispR,"edited Raster?");
2235            }
2236         }
2237         catch(Exception e)
2238         {
2239            messageOutput.append("\n"+"Edit failed...");
2240         }
2241      }
2242
2243      public DrainageRaster getDrainage(int dMin,boolean showLines,boolean
```



```
2243   largerDrainage)
2244      {
2245         System.out.println("New Drainage?: "+newDrainage);
2246         DrainageRaster dr=new DrainageRaster();
2247         try
2248         {
2249            System.out.println("larger drainage: "+largerDrainage);
2250            if(newDrainage==true)
2251            {
2252               getAreaValues();
2253               Raster tRaster = new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
2254
tRaster.setParameters(root_dir+dirName,"topo",dirName+"_topo","iceland_t
",true);
2255               dr = new DrainageRaster(tRaster,editsR,true,largerDrainage);
2256               dr.setParameters(root_dir+dirName,dirName+"_topo");
2257               Raster drainageR = dr.drainageDensity();
2258
SimpleIO.rasterToAscii(root_dir+dirName,dirName+"_drainage",drainageR);
2259               newDrainage=false;
2260            }
2261            else
2262            {
2263               Raster drainageR = new
Raster(root_dir+dirName+"/"+dirName+"_drainage.asc");
2264               Raster parentR = new
Raster(root_dir+dirName+"/"+dirName+"_parent.asc");
2265               dr = new DrainageRaster(drainageR,parentR,false,largerDraina
2266            }
2267         }
2268         catch(Exception e2)
2269         {
2270            System.out.println("DrainageRaster generation failed");
2271         }
2272         return dr;
2273      }
2274
2275      public Raster getDrainageRaster(int dMin,boolean showLines,boolean
largerDrainage)
2276      {
2277         Raster drainageR;
2278         try
2279         {
2280            DrainageRaster dr = getDrainage(dMin,showLines,largerDrainage);
2281            drainageR = dr.getDrainageArray();
2282            for(int i=1;i<drainageR.getXcells()-1;i++)
2283               for(int j=1;j<drainageR.getYcells()-1;j++)
2284               {
2285                  if(drainageR.getValue(i,j)>9999)
2286                     drainageR.changeValue(i,j,9999);
2287                  else if(drainageR.getValue(i,j)<dMin)
2288                     drainageR.changeValue(i,j,0);
2289               }
2290            drainageR.addLakes("lakeSeeds",new
Raster(root_dir+dirName+"/"+dirName+"_topo.asc"),editsR);
2291            if(showLines==true)
2292            {
2293               displayLines.addElement(dr.getDrainageLines());
2294               drainagePath=true;
2295            }
2296            dChanged=false;
2297         }
2298         catch(Exception e2)
2299         {
2300            System.out.println("Drainage raster construction failed");
2301            drainageR = new Raster(dispR,0);
2302         }
2303
drainageR.setParameters(root_dir+dirName,"drainage",dirName+"_drainage",
ainage",false);
```

```
2304         return drainageR;
2305      }
2306
2307      public Raster getDrainageBasins(int dMin,boolean showLines,int bMin,i
      bMax)
2308      {
2309         Raster basinR = new Raster(dispR,0);
2310         messageOutput.append("\n"+"generating basins...");
2311         try
2312         {
2313            DrainageRaster dr = getDrainage(dMin,showLines,false);
2314            dr.setParameters(root_dir+dirName,dirName+"_topo");
2315            dr.report();
2316            basinR = dr.getBasins(bMin,bMax);
2317            basinR.report();
2318
      basinR.setParameters(root_dir+dirName,"basins",dirName+"_basins","thiess
      ",false);
2319         }
2320         catch(Exception e2)
2321         {
2322            messageOutput.append("\n"+"Fill failed.");
2323         }
2324         return basinR;
2325      }
2326
2327
2328      public void editRaster(Polygon area)
2329      {
2330         try
2331         {
2332            manageEditsLayer(true);
2333            Point p;
2334            Vector v = area.getPolygon();
2335            if(allowChanges==true)
2336            {
2337               double cellSize = dispR.getCellSize();
2338               double xllCorner = dispR.getXll();
2339               double yllCorner = dispR.getYll();
2340               int x1 = (int)((area.getminX()-xllCorner)/cellSize);
2341               int y1 = (int)((area.getminY()-yllCorner)/cellSize);
2342               int x2 = (int)((area.getmaxX()-xllCorner)/cellSize);
2343               int y2 = (int)((area.getmaxY()-yllCorner)/cellSize);
2344               System.out.println(x1+","+y1+" "+x2+","+y2+"    - changeVal:
      "+clickValue);
2345               for(int i=x1;i<=x2;i++)
2346                  for(int j=y1;j<=y2;j++)
2347                  {
2348                     p = new
      Point((((double)i+0.5)*cellSize)+xllCorner,(((double)j+0.5)*cellSize)+yl
      rner);
2349                     p.displayValues();
2350                     if(area.pointIn(p))
2351                     {
2352                        editsR.changeValue(i,j,clickValue);
2353                        messageOutput.append("\n"+"Polygon edit made at
      "+i+","+j+": "+clickValue);
2354                     }
2355                  }
2356               messageOutput.append("\n"+"Polygon edit made");
2357               updateInfo(dispR,"edited Raster?");
2358            }
2359         }
2360         catch(Exception e)
2361         {
2362            messageOutput.append("\n"+"Edit failed...");
2363         }
2364      }
2365
2366      public void setHigh()
2367      {
```



```
2368         hiLoFlag=true;
2369         lowB.setText("HIGH");
2370         defDEM = "icedem_250.asc";
2371         dem = defDEM;
2372         getAreaValues();
2373         if((dirName.substring(0,3)).compareTo("hi_")!=0)
2374            dirName = "hi_"+getSubDir();
2375         resetModel(dirName,xll,yll,xur,yur,6);
2376         System.out.println("setHigh: "+hiLoFlag);
2377      }
2378      public void setLow()
2379      {
2380         hiLoFlag=false;
2381         lowB.setText("LOW");
2382         defDEM = "icedem_cp.asc";
2383         dem = defDEM;
2384         getAreaValues();
2385         dirName = getSubDir();
2386         if((dirName.substring(0,3)).compareTo("hi_")==0)
2387            dirName = dirName.substring(3);
2388         resetModel(dirName,xll,yll,xur,yur,6);
2389         System.out.println("setHigh: "+hiLoFlag);
2390      }
2391
2392      public void resetModel(String dName,double xllD,double yllD,double
      xurD,double yurD,int kc)
2393      {
2394         try
2395         {
2396            System.gc();
2397            messageOutput.append("\n"+"new Iceland model running!");
2398            if(vikingFarmFlag==true)
2399            {
2400               pInfo.remove(pInfoIcelandModel);
2401               pInfo.add(buildClimateControls());
2402               vikingFarmFlag=false;
2403            }
2404
2405            //reset the panel to default values
2406            if((hiLoFlag==true)&&((dName.substring(0,3)).compareTo("hi_")!=
2407               tdirinput.setText("hi_"+dName);
2408            else
2409               tdirinput.setText(dName);
2410            tfileinput.setText(defDEM);
2411            xllInput.setText(Double.toString(xllD));
2412            yllInput.setText(Double.toString(yllD));
2413            xurInput.setText(Double.toString(xurD));
2414            yurInput.setText(Double.toString(yurD));
2415            tkChoice.select(kc-1);
2416            pddSnowIce.setText(Float.toString(0.0065f));
2417            pddVegLim.setText(Integer.toString(4));
2418            setMonthComboBox(tempCB,0);
2419            setMonthComboBox(precipCB,0);
2420            setMonthComboBox(pddCB,0);
2421            setMonthComboBox(sunsCB,0);
2422            aspectA.setText(Integer.toString(135));
2423            aspectI.setText(Integer.toString(0));
2424            shiftamt.setText(Integer.toString(0));
2425            shiftTamt.setText(Integer.toString(0));
2426            shiftPamt.setText(Integer.toString(0));
2427            limT.setText(Integer.toString(1));
2428            focalField.setText(Integer.toString(15));
2429            saveCB.setSelected(false);
2430            kSet=false;
2431            precipIndex=false;
2432            pddIndex=false;
2433            sIndex=false;
2434
2435            snowcMonth.setText(Integer.toString(9));
2436            lowB.setBackground(Color.ORANGE);
2437            snowCCalcB.setBackground(Color.GRAY);
```

```
2438            tempB.setBackground(Color.ORANGE.darker());
2439            setKB.setBackground(Color.RED);
2440            precipB.setBackground(Color.ORANGE.darker());
2441            sunShedB.setBackground(Color.ORANGE.darker());
2442            pddGoB.setBackground(Color.RED);
2443            pddVegB.setBackground(Color.RED);
2444            growSCalcB.setBackground(Color.RED);
2445            growSStartB.setBackground(Color.RED);
2446            growSEndB.setBackground(Color.RED);
2447            growSMeanTB.setBackground(Color.RED);
2448            snowCalcB.setBackground(Color.RED);
2449            massBalB.setBackground(Color.RED);
2450            elaOffCalcB.setBackground(Color.RED);
2451            elaCalcB.setBackground(Color.RED);
2452            ela_TopoB.setBackground(Color.RED);
2453            lapseB.setBackground(Color.RED);
2454            getAreaValues();
2455            newDrainage=true;
2456            hideAllOverlays();
2457    //      hiLoFlag=false;
2458            displayLines = new Vector();
2459            displayPoints = new Vector();
2460            climateControl = new ClimateControl(this,hiLoFlag,messageOutput
2461            tempOrPrecipImported=false;
2462            blackAndWhite=false;
2463            dispR = climateControl.displayArea(false);
2464            noData = dispR.getNoData();
2465            if(hiLoFlag==true)
2466            {
2467               editsR = new
    Raster(root_dir+"config_files/overall_edits_250.asc",getYll(),getYur(),"
2468               editsR = editsR.gridclip(new Rectangle(new Point(xll,yll),ne
    Point(xur,yur)));
2469            }
2470            else
2471            {
2472               editsR = new
    Raster(root_dir+"config_files/overall_edits_cp.asc",getYll(),getYur(),"N
2473               editsR = editsR.gridclip(new Rectangle(new Point(xll,yll),ne
    Point(xur,yur)));
2474            }
2475            messageOutput.append("\n"+"reset?");
2476
2477            modelPanel.redrawGraphics();
2478            updateInfo(dispR,dName);
2479         }
2480         catch(Exception e2)
2481         {
2482            messageOutput.append("\n"+"reset failed");
2483         }
2484      }
2485
2486      public void setMonthComboBox(JComboBox jcb,int index)
2487      {
2488         jcb.removeActionListener(this);
2489         jcb.setSelectedIndex(index);
2490         jcb.setBackground(Color.lightGray);
2491         jcb.addActionListener(this);
2492      }
2493
2494      public void resetButtonColour(int index)
2495      {
2496         if(index==1)//temp
2497         {
2498            lapseB.setBackground(Color.ORANGE.darker());
2499            pddGoB.setBackground(Color.ORANGE.darker());
2500            pddVegB.setBackground(Color.ORANGE.darker());
2501            setKB.setBackground(Color.ORANGE.darker());
2502            growSCalcB.setBackground(Color.RED);
2503            snowCalcB.setBackground(Color.RED);
2504            massBalB.setBackground(Color.RED);
```



```
2505            elaOffCalcB.setBackground(Color.RED);
2506            elaCalcB.setBackground(Color.RED);
2507            ela_TopoB.setBackground(Color.RED);
2508            growSStartB.setBackground(Color.RED);
2509            growSEndB.setBackground(Color.RED);
2510            growSMeanTB.setBackground(Color.RED);
2511         }
2512         if(index==2)//temp K
2513         {
2514            kSet=true;
2515            snowCalcB.setBackground(Color.RED);
2516            massBalB.setBackground(Color.RED);
2517            elaOffCalcB.setBackground(Color.RED);
2518            elaCalcB.setBackground(Color.RED);
2519            ela_TopoB.setBackground(Color.RED);
2520            growSCalcB.setBackground(Color.ORANGE.darker());
2521            growSStartB.setBackground(Color.RED);
2522            growSEndB.setBackground(Color.RED);
2523            growSMeanTB.setBackground(Color.RED);
2524            if(precipIndex==true)
2525               snowCalcB.setBackground(Color.ORANGE.darker());
2526            else
2527               snowCalcB.setBackground(Color.RED);
2528         }
2529         if(index==0)//precip
2530         {
2531            precipIndex=true;
2532            sIndex=false;
2533            if(kSet==true)
2534               snowCalcB.setBackground(Color.ORANGE.darker());
2535            else
2536               snowCalcB.setBackground(Color.RED);
2537            massBalB.setBackground(Color.RED);
2538            elaOffCalcB.setBackground(Color.RED);
2539            elaCalcB.setBackground(Color.RED);
2540            ela_TopoB.setBackground(Color.RED);
2541         }
2542         if (index==3)//PDD Veg
2543            growSCalcB.setBackground(Color.ORANGE.darker());
2544         if (index==4)//Growing season calc
2545         {
2546            growSStartB.setBackground(Color.ORANGE.darker());
2547            growSEndB.setBackground(Color.ORANGE.darker());
2548            growSMeanTB.setBackground(Color.ORANGE.darker());
2549         }
2550         if (index==5)//PDD Snow
2551         {
2552            if(sIndex==true)
2553            {
2554               massBalB.setBackground(Color.ORANGE.darker());
2555               elaOffCalcB.setBackground(Color.ORANGE.darker());
2556            }
2557            else
2558            {
2559               massBalB.setBackground(Color.RED);
2560               elaOffCalcB.setBackground(Color.RED);
2561            }
2562            elaCalcB.setBackground(Color.RED);
2563            ela_TopoB.setBackground(Color.RED);
2564            pddIndex=true;
2565         }
2566         if(index==6)//snowfall
2567         {
2568            sIndex=true;
2569            if(pddIndex==true)
2570            {
2571               massBalB.setBackground(Color.ORANGE.darker());
2572               elaOffCalcB.setBackground(Color.ORANGE.darker());
2573            }
2574            else
2575            {
```

```
2576                massBalB.setBackground(Color.RED);
2577                elaOffCalcB.setBackground(Color.RED);
2578             }
2579             elaCalcB.setBackground(Color.RED);
2580             ela_TopoB.setBackground(Color.RED);
2581          }
2582          if(index==7)//ELA offset
2583          {
2584             elaCalcB.setBackground(Color.ORANGE.darker());
2585             ela_TopoB.setBackground(Color.RED);
2586          }
2587          if(index==8)//ELA Height
2588             ela_TopoB.setBackground(Color.ORANGE.darker());
2589          if(index==9)
2590             sunShedB.setBackground(Color.GREEN);
2591       }
2592
2593       public void updateInfo(Raster dispR,String name) throws Exception
2594       {
2595          Raster sandarR=new Raster();
2596          if((showSandar==true)||(trimToSandar==true))
2597          {
2598             Raster tRaster = new
   Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
2599             sandarR = editsR.getIceCells("sandarSeeds",tRaster,20,15,false)
2600             sandarR.setScaleType("sandar");
2601             if(multiRasterFlag==false)
2602                overlay(dispR,sandarR,0);
2603             else
2604                overlayAnother(sandarR,0);
2605          }
2606          Raster iceR,iceR2;
2607          iceR2=new Raster();
2608          iceR=new Raster();
2609          if((showIceMargins==true)||(showIceArea==true)||(trimToIce==true))
2610          {
2611             Raster tRaster = new
   Raster(root_dir+dirName+"/"+dirName+"_topo.asc");
2612             iceR2 = editsR.getIceCells("iceSeeds",tRaster,20,20,true);
2613             if(showIceArea==true)
2614                iceR = iceR2;
2615             else if(showIceMargins==true)
2616                iceR = editsR.getIceEdges("iceSeeds",tRaster,20,20,true);
2617             iceR.setScaleType("ice");
2618          }
2619          if(showRiversAndLakes==true)
2620          {
2621             Raster dRaster;
2622             dRaster =
   getDrainageRaster(drainMin,showDrainageVectors,largerDrainage);

2623             if(multiRasterFlag==false)
2624                overlay(dispR,dRaster,0);
2625             else
2626                overlayAnother(dRaster,0);
2627          }
2628          if((showIceMargins==true)||(showIceArea==true))
2629          {
2630             if(multiRasterFlag==false)
2631                overlay(dispR,iceR,0);
2632             else
2633                overlayAnother(iceR,0);
2634          }
2635          if((name=="Mass Balance")||(name=="basins")||(name=="Veg PDD & mas
   balance"))
2636          {
2637             int layerCount=1;
2638             if(showSandar==true)
2639                layerCount++;
2640             if(showRiversAndLakes==true)
2641                layerCount++;
```



```
2642             if((showIceMargins==true)||(showIceArea==true))
2643                layerCount++;
2644             Raster mbR = (Raster)rasters.elementAt(rasters.size()-layerCoun
2645             rasters.removeElementAt(rasters.size()-layerCount);
2646             upperLayer=rasters.size()-1;
2647             overlayAnother(mbR,1);
2648          }
2649          if((trimToSandar==true)||(trimToIce==true))
2650          {
2651             if(multiRasterFlag==true)
2652                for(int tr=1;tr<rasters.size();tr++)
2653                {
2654                   Raster tRast=(Raster)rasters.elementAt(tr);
2655
   if((tRast.getScaleType()!="ice")&&(tRast.getScaleType()!="sandar")&&(tRa
   getScaleType()!="massb"))
2656                   {
2657
   if((trimToSandar==true)&&(tRast.getScaleType()!="drainage"))
2658                      tRast.removeSelectedCells(sandarR);
2659                   if(trimToIce==true)
2660                      tRast.removeSelectedCells(iceR2);
2661                   rasters.removeElementAt(tr);
2662                   rasters.insertElementAt(tRast,tr);
2663                }
2664                if(blackAndWhite==true)
2665                {
2666
   if((tRast.getScaleType()=="pdd_veg")||(tRast.getScaleType()=="2nd PDD"))
2667                   {
2668                      float[] cVals = new float[1];
2669                      cVals[0]=100;
2670                      displayLines.addElement(tRast.contours(cVals));
2671                   }
2672                   if(tRast.getScaleType()=="gs_length")
2673                   {
2674                      float[] cVals = new float[12];
2675                      for(int v=0;v<12;v++)
2676                         cVals[v]=v*30;
2677                      displayLines.addElement(tRast.contours(cVals));
2678                   }
2679                   if(tRast.getScaleType()=="gs_date")
2680                   {
2681                      float[] cVals = new float[11];
2682                      cVals[0]=32;cVals[1]=60;cVals[2]=91;cVals[3]=121;
2683                      cVals[4]=152;cVals[5]=182;cVals[6]=213;cVals[7]=244
2684                      cVals[8]=274;cVals[9]=305;cVals[10]=335;
2685                      displayLines.addElement(tRast.contours(cVals));
2686                   }
2687                }
2688             }
2689             else
2690                dispR.removeSelectedCells(iceR2);
2691          }
2692
2693          if((editsLayer==true)&&(multiRasterFlag==false))
2694             overlay(dispR,editsR,0);
2695          else if(editsLayer==true)
2696             overlayAnother(editsR,0);
2697          int classes;
2698          if(dispR.getScaleType()=="thiessens")
2699             classes = (int)dispR.getMaxVal(noData,noData);
2700          else
2701             classes=0;
2702          RasterScalePicker rsp = new
   RasterScalePicker(dispR.getScaleType(),dispR.getNoData(),classes,blackAn
   ite);
2703          //update display text
2704          dataTitle.setText(getRootDir()+getSubDir());
2705          dataName.setText(name);
2706          dataCells.setText((String)Integer.toString(dispR.getXcells())+" x
```

```
2707                  +(String)Integer.toString(dispR.getYcells()));
2708
      dataNumCells.setText((String)Integer.toString(dispR.getXcells()*dispR.ge
      ells()));
2709        float horExtent =
      ((dispR.getXcells()*(float)dispR.getCellSize())/1000);
2710        float verExtent =
      ((dispR.getYcells()*(float)dispR.getCellSize())/1000);
2711        dataKm.setText(horExtent+"km x "+verExtent+"km");
2712        dataCellsize.setText(Float.toString((float)dispR.getCellSize())+"m
2713        dataReg.setText(Float.toString((float)dispR.getXll())+","+
2714                           Float.toString((float)dispR.getYll()));
2715  //    updateDisplayLines();
2716        updateDisplayPoints();
2717
2718        clickValue = Float.parseFloat(clickVal.getText());
2719        //update raster panel
2720  //    p0.remove(rastPanel);
2721        if(multiRasterFlag==false)
2722        {
2723           rastPanel = new
      RasterPanelVik(this,dispR,displayLines,displayPoints,rsp,
2724
      statusField,statusValueField,clickValue,allowChanges,saveFileFlag);
2725           paintedRaster = dispR;
2726        }
2727        else
2728        {
2729           rastPanel = new
      RasterPanelVik(this,rasters,displayLines,displayPoints,rsp,
2730
      statusField,statusValueField,clickValue,allowChanges,saveFileFlag);
2731           multiRasterFlag=false;
2732           paintedRaster = (Raster)rasters.elementAt(rasters.size()-1);
2733        }
2734        rastPanel.addGraphicsListener(modelPanel);
2735        fileSaved=false;
2736        modelPanel.redrawGraphics();
2737
2738  //    rastPanel.setPreferredSize(new
      Dimension(rasterPWidth,infoPHeight-144));
2739  //    rastPanel.setBorder(BorderFactory.createEtchedBorder());
2740  //    p0.add(rastPanel,BorderLayout.NORTH);
2741  //    savePanel(name,rastPanel.getBImage());
2742
2743     //update gradient panel
2744        rastInfoPanel.remove(gradPanel);
2745        gradPanel = new GradientPanel(rsp,saveFileFlag);
2746        gradPanel.setBorder(BorderFactory.createEtchedBorder());
2747        rastInfoPanel.add(gradPanel,0);
2748
2749        pack();
2750        setVisible(true);
2751
2752     //update gradient info panel
2753        rastScaleP.remove(oldMinRsc);
2754        oldMinRsc = new JLabel(Float.toString(rsp.getMinScale()));
2755        rastScaleP.add(oldMinRsc,0);
2756        rastScaleP.remove(oldMaxRsc);
2757        oldMaxRsc = new JLabel("      "+Float.toString(rsp.getMaxScale()))
2758        rastScaleP.add(oldMaxRsc,2);
2759        displayLines = new Vector();
2760        climateControl.resetDisplayLines();
2761        System.out.println("update finished");
2762     }
2763
2764     public boolean areaValuesChanged()
2765     {
2766        boolean changed = false;
```



```
2767        if((this.xll != Double.parseDouble(xllInput.getText()))||
2768        (this.yll != Double.parseDouble(yllInput.getText()))||
2769        (this.xur != Double.parseDouble(xurInput.getText()))||
2770        (this.yur != Double.parseDouble(yurInput.getText()))||
2771        (dem.compareTo((String)tfileinput.getText())!=0))
2772  //    ||(dirName.compareTo((String)tdirinput.getText())!=0))
2773           changed =true;
2774        messageOutput.append("\n"+"Area Changed?: "+changed);
2775        return changed;
2776     }
2777
2778     public void getAreaValues()
2779     {
2780        xll=getXll();
2781        yll=getYll();
2782        xur=getXur();
2783        yur=getYur();
2784        root_dir = getRootDir();
2785        dirName = getSubDir();
2786        dem = getDEMname();
2787     }
2788
2789     public void setAreaValues(Point newLL,Point newUR)
2790     {
2791        xllInput.setText(Double.toString(newLL.getX()));
2792        yllInput.setText(Double.toString(newLL.getY()));
2793        xurInput.setText(Double.toString(newUR.getX()));
2794        yurInput.setText(Double.toString(newUR.getY()));
2795     }
2796
2797     public void setAreaValues()
2798     {
2799        xllInput.setText(Double.toString(xll));
2800        yllInput.setText(Double.toString(yll));
2801        xurInput.setText(Double.toString(xur));
2802        yurInput.setText(Double.toString(yur));
2803        tdirinput.setText(dirName);
2804        tfileinput.setText(dem);
2805     }
2806
2807     public void updateDisplayLines()
2808     {
2809        if(vikingFarmFlag==true)
2810           displayLines = vfr.getDisplayLines();
2811        else
2812           displayLines = climateControl.getDisplayLines();
2813     }
2814
2815     public void updateDisplayPoints()
2816     {
2817        try
2818        {
2819           if(vikingFarmFlag==true)
2820              displayPoints = vfr.getDisplayPoints();
2821           else
2822              displayPoints =
      SimpleIO.readPoints(root_dir+"config_files/farms.csv");
2823        }
2824        catch (Exception e)
2825        {
2826           messageOutput.append("\n"+"Farm sites not added");
2827        }
2828     }
2829
2830     //upperL defines the 'uppermost' layer in the Raster (which will be u
      for enquiries)
2831     //1 means the layer will be the top enquire layer,0 that the layer wi
      not be the 'top' layer
2832     public void overlay(Raster lowerRaster, Raster upperRaster,int upperL
      throws Exception
2833     {
```

```
2834        rasters = new Vector();
2835        rasters.addElement(lowerRaster);
2836        rasters.addElement(upperRaster);
2837        upperLayer = upperL;
2838        System.out.println("new upper:   "+upperLayer);
2839        multiRasterFlag=true;
2840     }
2841
2842     public void overlayAnother(Raster upperRaster,int upperL) throws
      Exception
2843     {
2844        rasters.addElement(upperRaster);
2845        upperLayer +=upperL;
2846        System.out.println("upper updated: "+upperLayer);
2847     }
2848
2849     public void presetReset(String arg)
2850     {
2851        if ("All Iceland".equals(arg))
2852           resetModel("all_iceland",227000,291000,774000,698000,7);
2853        else if ("Breida".equals(arg))
2854           resetModel("breida",590000,375000,660000,453000,7);
2855        else if ("East Fjords".equals(arg))
2856           resetModel("eastfjords",680000,430000,765000,580000,7);
2857        else if ("W Vatnajokull".equals(arg))
2858           resetModel("w_vatnajokull",530000,375000,600000,500000,7);
2859        else if ("E Vatnajokull".equals(arg))
2860           resetModel("e_vatnajokull",640000,415000,710000,490000,7);
2861        else if ("SE Vatnajokull".equals(arg))
2862           resetModel("se_vatnajokull",590000,365000,710000,455000,7);
2863        else if ("Hofsjokull".equals(arg))
2864           resetModel("hofsjokull",680000,450000,715000,480000,4);
2865        else if ("Hofsjokull-m".equals(arg))
2866           resetModel("hofs-m",680000,450000,710000,480000,4);
2867        else if ("Myrdalsjokull".equals(arg))
2868           resetModel("myrdalsjokull",442000,319000,522000,399000,7);
2869        else if ("Myvatn".equals(arg))
2870           resetModel("myvatn",560000,545000,605000,615000,5);
2871        else if ("Myvatn & S".equals(arg))
2872           resetModel("myvatn_s",550000,455000,620000,615000,5);
2873        else if ("NW Fjords".equals(arg))
2874           resetModel("nwfjords",240000,530000,400000,670000,5);
2875        else if ("Oraefi".equals(arg))
2876           resetModel("oraefi",590000,370000,650000,430000,5);
2877        else if ("Reykholtsdalur".equals(arg))
2878           resetModel("reykholtsdalur",335000,430000,435000,500000,6);
2879        else if ("Snaefellsnes".equals(arg))
2880           resetModel("snaefellsnes",260000,477000,315000,503000,3);
2881        else if ("Svarfadardalur".equals(arg))
2882           resetModel("svardalur",497000,573000,530000,610000,2);
2883        else if ("Thorsmork".equals(arg))
2884           resetModel("thorsmork",440000,320000,495000,375000,5);
2885        else if ("Thorsmork2".equals(arg))
2886           resetModel("thorsmork2",467000,347000,495000,375000,5);
2887        else if ("Trollaskagi".equals(arg))
2888           resetModel("trollaskagi",485000,540000,545000,615000,4);
2889        else if ("Vatnsdalur".equals(arg))
2890           resetModel("vatnsdalur2",503000,586500,513000,596500,4);
2891        else if ("testarea".equals(arg))
2892           resetModel("testarea",455000,347000,458000,349000,2);
2893     }
2894
2895     public int getUpperLayer()
2896     {
2897        return upperLayer;
2898     }
2899
2900     public double getXll()
2901     {
2902        return Double.parseDouble(xllInput.getText());
2903     }
```



```
2904
2905     public double getYll()
2906     {
2907        return Double.parseDouble(yllInput.getText());
2908     }
2909
2910     public double getXur()
2911     {
2912        return Double.parseDouble(xurInput.getText());
2913     }
2914
2915     public double getYur()
2916     {
2917        return Double.parseDouble(yurInput.getText());
2918     }
2919
2920     public boolean getHighlightCheck()
2921     {
2922        return this.highlightCheck;
2923     }
2924
2925     public String getDEMname()
2926     {
2927        return (String)tfileinput.getText();
2928     }
2929
2930     public String getSubDir()
2931     {
2932        return (String)tdirinput.getText();
2933     }
2934     public String getRootDir()
2935     {
2936        return this.root_dir;
2937     }
2938
2939     public Raster getDisplayRaster()
2940     {
2941        return this.paintedRaster;
2942     }
2943
2944     public void setSunShedRadius(int ssr)
2945     {
2946        sSRadius.setText(""+ssr);
2947     }
2948
2949     public boolean getBlackAndWhite()
2950     {
2951        return blackAndWhite;
2952     }
2953
2954     private void closeDown()
2955     {
2956        int response = JOptionPane.showConfirmDialog(this,"End this Icelan
      Climate Model?");
2957        if (response == JOptionPane.YES_OPTION)
2958           System.exit(0);    // Exit program.
2959        if ((response == JOptionPane.NO_OPTION)||(response ==
      JOptionPane.CANCEL_OPTION))
2960           return;
2961     }
2962
2963     public afc.spatial.Point getGeoTrans(double x,double y)
2964     {
2965        Dimension panelSize = modelPanel.getSize();
2966        int numRows = dispR.getYcells();
2967        int numCols = dispR.getXcells();
2968        double cellSize = dispR.getCellSize();
2969        double panelWidth = panelSize.width;
2970        double panelHeight = panelSize.height;
2971        double mapAspectRatio = (double)numRows/(double)numCols;
2972        double panelAspectRatio = panelHeight/panelWidth;
```

```
2973        double scaling =
    Math.min(panelWidth/(numCols*cellSize),panelHeight/(numRows*cellSize));
2974        double hCornerOffset,wCornerOffset;
2975        if(mapAspectRatio>panelAspectRatio)//lower aspect ratio=wider map
2976        {
2977           hCornerOffset=0;
2978           wCornerOffset=(panelWidth-(panelHeight/mapAspectRatio))/2d;
2979        }
2980        else
2981        {
2982           wCornerOffset=0;
2983           hCornerOffset=(panelHeight-(panelWidth*mapAspectRatio))/2d;
2984        }
2985        x+=(-BORDER_SIZE-wCornerOffset);
2986        y+=(-panelHeight-BORDER_SIZE+hCornerOffset);
2987        x=(x/scaling);
2988        y=(-y/scaling);
2989        x+=dispR.getXll();
2990        y+=dispR.getYll();
2991
2992        return new afc.spatial.Point(x,y);
2993     }
2994
2995     public afc.spatial.Point getWindowTrans(double x,double y)
2996     {
2997        Dimension panelSize = modelPanel.getSize();
2998        int numRows = dispR.getYcells();
2999        int numCols = dispR.getXcells();
3000        double cellSize = dispR.getCellSize();
3001        double panelWidth = numCols*imageScaling;
3002        double panelHeight = numRows*imageScaling;
3003        double mapAspectRatio = (double)numRows/(double)numCols;
3004        double panelAspectRatio = panelHeight/panelWidth;
3005        double scaling =
    Math.min(panelWidth/(numCols*cellSize),panelHeight/(numRows*cellSize));
3006        double hCornerOffset;
3007        if(mapAspectRatio>panelAspectRatio)//lower aspect ratio=wider map
3008           hCornerOffset=0;
3009        else
3010           hCornerOffset=(panelHeight-(panelWidth*mapAspectRatio))/2d;
3011        int biHeight = (int)(panelHeight-(2*hCornerOffset));
3012        x-=dispR.getXll();
3013        y-=dispR.getYll();
3014        x=(x*scaling);
3015        y=(-y*scaling);
3016        x+=(BORDER_SIZE);
3017        y+=(biHeight+BORDER_SIZE);
3018        return new afc.spatial.Point(x,y);
3019     }
3020
3021     public Vector pathFromAbsCoords(Vector polyCoord)
3022     {
3023        System.out.println("Path from Coords? "+polyCoord.size());
3024        int k=0;
3025        int[] gpcIndex = new int[1];
3026        afc.spatial.Point p,p2;
3027        Vector paths = new Vector();
3028        GeneralPath gpcoords = new GeneralPath();
3029        try
3030        {
3031           gpcIndex = (int[]) polyCoord.elementAt(polyCoord.size()-1);
3032        }
3033        catch (Exception e4)
3034        {
3035           System.out.println("cannot turn Vector into a GeneralPath");
3036           return paths;
3037        }
3038        double xllCorner = editsR.getXll();
3039        double yllCorner = editsR.getYll();
3040        double cellSize = editsR.getCellSize();
3041        while(k<polyCoord.size()-1)
```



```
3042        {
3043           p2 = (afc.spatial.Point)polyCoord.elementAt(k);
3044     //    p2.displayValues();
3045           p = getWindowTrans(p2.getX(),p2.getY());
3046     //
    if((drainagePath==false)||((drainagePath==true)&&((editsR.getValue((int)
    2.getX()-xllCorner)/cellSize),(int)((p2.getY()-yllCorner)/cellSize))>126
3047     //    {
3048              if(gpcIndex[k]==1)//line start point
3049              {
3050                 if(k>0)
3051                    paths.addElement(gpcoords);
3052     //
    if((drainagePath==true)&&(editsR.getValue((int)((p2.getX()-xllCorner)/ce
    ize),(int)((p2.getY()-yllCorner)/cellSize))>126))
3053                 if(drainagePath==true)
3054                    paths.addElement(new Color(0,0,255));
3055     //             else if(drainagePath==true)
3056     //                paths.addElement(new Color(255,255,0,0));
3057                 else
3058                    paths.addElement(new Color(255,255,255));
3059                 gpcoords = new GeneralPath();
3060                 gpcoords.moveTo((float)p.getX(),(float)p.getY());
3061              }
3062              else if(gpcIndex[k]==2)//draw a line to here
3063                 gpcoords.lineTo((float)p.getX(),(float)p.getY());
3064              else if(gpcIndex[k]==3) //close the shape
3065              {
3066                 gpcoords.lineTo((float)p.getX(),(float)p.getY());
3067                 gpcoords.closePath();
3068              }
3069     //       }
3070           k++;
3071        }
3072        paths.addElement(gpcoords);
3073        System.out.println("converted path");
3074        return paths;
3075     }
3076
3077     public void hideAllOverlays()
3078     {
3079        editsLayer=false;
3080        drainagePath=false;
3081        showRiversAndLakes=false;
3082        showIceMargins=false;
3083        showIceArea=false;
3084        showSandar=false;
3085        trimToIce=false;
3086        trimToSandar=false;
3087        showDrainageVectors=false;
3088        largerDrainage=false;
3089     }
3090
3091     public static void main(String args[]) throws Exception
3092      {
3093          Ice_model_Run imp = new Ice_model_Run();
3094          imp.start();
3095      }
3096
3097     //-----------nested class-----------
3098     private class WinMonitor extends WindowAdapter
3099     {
3100        public void windowClosing(WindowEvent event)
3101        {
3102           closeDown();
3103        }
3104     }
3105
3106     private class OptionMonitor extends WindowAdapter
3107     {
3108        public void windowClosing(WindowEvent event)
```

```
3109        {
3110           int response = JOptionPane.showConfirmDialog(null,"Apply
    Options?");
3111           if (response == JOptionPane.YES_OPTION)
3112           {
3113              optionResults = new int[optionObjects.length];
3114              for(int oi=0;oi<optionObjects.length;oi++)
3115                 optionResults[oi] =
    Integer.parseInt(((JTextField)optionObjects[oi]).getText());
3116              if(optionIndex==1)
3117              {
3118                 if(optionResults[0]!=drainMin)
3119                    dChanged=true;
3120
    drainMin=optionResults[0];basMin=optionResults[1];basMax=optionResults[2
3121              }
3122           }
3123           if ((response == JOptionPane.NO_OPTION)||(response ==
    JOptionPane.CANCEL_OPTION))
3124              return;
3125        }
3126     }
3127
3128     private class ModelPanel extends JPanel implements GraphicsListener
3129     {
3130        private double hCornerOffset,wCornerOffset;
3131
3132        public void paintComponent(Graphics g)
3133        {
3134           Vector images = rastPanel.paint(g);
3135           //Draw each of the drawable graphics (rasters, points or lines)
    provided there are vectors to draw
3136     //    super.paintComponent(g);   //paints the background
3137           Graphics2D g2 = (Graphics2D) g; // Typecast graphics as graphic
3138
    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE
    TIALIAS_ON);
3139           Dimension panelSize = getSize();
3140           g2.setPaint(new Color(80,80,80));
3141           g2.fillRect(0,0,getWidth(),getHeight());
3142
3143     //set parameters to scale the image with the right aspect ratio insid
    the panel
3144           int numRows = dispR.getYcells();
3145           int numCols = dispR.getXcells();
3146           double panelWidth = panelSize.width;
3147           double panelHeight = panelSize.height;
3148           double mapAspectRatio = (double)numRows/(double)numCols;
3149           double panelAspectRatio = panelHeight/panelWidth;
3150     //        imageScaling = Math.min(2048/numCols,1536/numRows);
3151           if(mapAspectRatio>panelAspectRatio)//lower aspect ratio=wider m
3152           {
3153              hCornerOffset=0;
3154              wCornerOffset=(panelWidth-(panelHeight/mapAspectRatio))/2d;
3155           }
3156           else
3157           {
3158              wCornerOffset=0;
3159              hCornerOffset=(panelHeight-(panelWidth*mapAspectRatio))/2d;
3160           }
3161           bi = (BufferedImage)
    createImage((int)(numCols*imageScaling),(int)(numRows*imageScaling));
3162           Graphics2D big = bi.createGraphics();
3163           big.setRenderingHints(g2.getRenderingHints());
3164           //Scale and draw the images inside the panel
3165           int i=0;
3166           while (i<images.size())
3167           {
3168              MemoryImageSource ms = (MemoryImageSource)images.elementAt(i
3169              Image rasterImage = createImage(ms);
```



```
3170
    big.drawImage(rasterImage,0,(int)(numRows*imageScaling),(int)(numCols*im
    Scaling),(int)(-numRows*imageScaling),this);
3171              i++;
3172           }
3173
3174           //if there are lines to plot
3175           if((displayLines!=null)&&(displayLines.size()>0))
3176           {
3177              i=0;
3178              System.out.println("drawing lines?");
3179              GeneralPath path;
3180              int counti=0;
3181              while (i<displayLines.size())
3182              {
3183                 stroke = new BasicStroke((float)(imageScaling/5));
3184                 int colourCode = (i*(int)(255/displayLines.size()));
3185                 Color fillC = new Color(180,colourCode,colourCode,255);
3186                 System.out.println("path: "+displayLines.size());
3187                 Vector paths =
    pathFromAbsCoords((Vector)displayLines.elementAt(i));
3188                 System.out.println("path no: "+counti+"   "+paths.size())
3189                 counti++;
3190                 for(int gpp=0;gpp<paths.size();gpp++)
3191                 {
3192                    border_colour = (Color)paths.elementAt(gpp++);
3193                    path = (GeneralPath)paths.elementAt(gpp);
3194                    big.setPaint(fillC);
3195                    big.fill(path);
3196                    big.setPaint(border_colour);
3197                    big.setStroke(stroke);
3198                    big.draw(path);
3199                 }
3200                 i++;
3201              }
3202           }
3203
3204           //if there are points to plot
3205           if((displayPoints!=null)&&(displayPoints.size()>0))
3206           {
3207              i=0;
3208              int x,y;
3209              point_colour = new Color(255,255,0);
3210              border_colour = new Color(0,0,255);
3211              stroke = new BasicStroke((float)(imageScaling/5));
3212              afc.spatial.Point pt = new afc.spatial.Point();
3213              Point2D.Double ptd = new Point2D.Double();
3214              Point2D.Double ptd2 = new Point2D.Double();
3215              while (i<displayPoints.size())
3216              {
3217                 pt = (afc.spatial.Point)displayPoints.elementAt(i);
3218                 pt = getWindowTrans(pt.getX(),pt.getY());
3219                 x=(int) pt.getX();
3220                 y=(int) pt.getY();
3221     /*          big.setPaint(point_colour);
3222                 big.fillOval(x-3,y-3,6,6);
3223                 big.setStroke(stroke);
3224                 big.drawOval(x-3,y-3,6,6);*/
3225                 big.setPaint(new Color(0,0,0));
3226                 big.setStroke(stroke);
3227
3228                 if(hiLoFlag==true)
3229                 {
3230                    int crossWidth = (int)(imageScaling*0.9f);
3231                    int crossHeight = (int)(imageScaling*0.6f);
3232
    big.drawLine(x-crossWidth,y-crossHeight,x+crossWidth,y+crossHeight);
3233
    big.drawLine(x-crossWidth,y+crossHeight,x+crossWidth,y-crossHeight);
3234              }
```

```
3235    /*              else
3236                    {
3237                       int crossWidth = (int)(imageScaling/2f);
3238                       int crossHeight = (int)(imageScaling/3.5f);
3239
        big.drawLine(x-crossWidth,y-crossHeight,x+crossWidth,y+crossHeight);
3240
        big.drawLine(x-crossWidth,y+crossHeight,x+crossWidth,y-crossHeight);
3241                    }*/
3242                    i++;
3243                 }
3244              }
3245              try
3246              {
3247                 if((saveFileFlag)&&(fileSaved==false))
3248                 {
3249                    fileSaved=true;
3250                    int n=0;
3251                    boolean check=false;
3252                    while(check==false)
3253                    {
3254
        if(SimpleIO.checkFile(dispR.getDirectory()+"/"+dispR.getName()+n+".JPG")
        alse)
3255                          check=true;
3256                       n++;
3257                    }
3258                    if(dispR.getScaleType()=="climseq")
3259                    {
3260
        SimpleIO.writeJPEG(dispR.getDirectory(),dispR.getName(),bi);
3261                       System.out.println("JPEG image created:
        "+dispR.getDirectory()+"/"+dispR.getName()+".jpg");
3262                    }
3263                    else
3264                    {
3265
        SimpleIO.writeJPEG(dispR.getDirectory(),dispR.getName()+(n-1),bi);
3266                       System.out.println("JPEG image created:
        "+dispR.getDirectory()+dispR.getName()+(n-1)+".jpg");
3267                    }
3268                 }
3269              }
3270              catch(Exception outE)
3271              {
3272                 System.out.println("JPEG image creation failed");
3273              }
3274        //    System.out.println("g2: "+(int)wCornerOffset+"
        "+(int)hCornerOffset+" "+(int)(panelWidth-(wCornerOffset*2))+"
        "+(int)(panelHeight-(hCornerOffset*2)));
3275
        g2.drawImage(bi,(int)wCornerOffset,(int)hCornerOffset,(int)(panelWidth-(
        rnerOffset*2)),(int)(panelHeight-(hCornerOffset*2)),this);
3276          }
3277
3278          public void initialise()
3279          {
3280             setPreferredSize(new Dimension(rasterPWidth,infoPHeight-144));
3281             setBackground(new Color(128,128,128));
3282             setCursor(new Cursor(Cursor.CROSSHAIR_CURSOR));
3283             addMouseMotionListener(new MouseMoveMonitor());
3284             addMouseListener(new MouseClickMonitor());
3285          }
3286
3287          // Immediately update graphics that need updating.
3288          public void redrawGraphics()
3289          {
3290             Dimension d = getSize();
3291             setBackground(new Color(80,80,80));
3292
        paintImmediately((int)wCornerOffset,(int)hCornerOffset,(int)(d.width-(wC
```



```
3292    erOffset*2)),(int)(d.height-(hCornerOffset*2)));
3293          }
3294       }
3295
3296       private class MouseClickMonitor extends MouseInputAdapter
3297       {
3298          public void mouseClicked(MouseEvent e)
3299          {
3300             start = getGeoTrans(e.getX(),e.getY());
3301             float x = (float)start.getX();
3302             float y = (float)start.getY();
3303             statusField.setText("Dragging: "+x+","+y);
3304             editRaster(x,y);
3305          }
3306
3307          public void mousePressed(MouseEvent e)
3308          {
3309             mouseCoords = new Vector();
3310             mPressed=true;
3311             start = getGeoTrans(e.getX(),e.getY());
3312             float x = (float)start.getX();
3313             float y = (float)start.getY();
3314             statusField.setText("Dragging: "+x+","+y);
3315             mouseCoords.addElement(new afc.spatial.Point(x,y));
3316          }
3317          public void mouseReleased(MouseEvent e)
3318          {
3319             mPressed=false;
3320    //         end = getGeoTrans(e.getX(),e.getY());
3321    //         float x = (float)e.getX();
3322    //         float y = (float)e.getY();
3323    //         afc.spatial.Point p = getGeoTrans(e.getX(),e.getY());
3324    //           output.setText("Dragging End: "+x+","+y);
3325    //           mouseCoords.addElement(new afc.spatial.Point(x,y));
3326    //           imr.editRaster(new afc.spatial.Rectangle(start,end));
3327             for(int i=0;i<mouseCoords.size();i++)
3328                ((afc.spatial.Point)mouseCoords.elementAt(i)).displayValues(
3329             editRaster(new afc.spatial.Polygon(mouseCoords),true);
3330             dragNo=0;
3331          }
3332       }
3333
3334       private class MouseMoveMonitor extends MouseMotionAdapter
3335       {
3336          public void mouseMoved(MouseEvent e)
3337          {
3338             afc.spatial.Point p = getGeoTrans(e.getX(),e.getY());
3339    //         p=getWindowTrans(p.getX(),p.getY());
3340             float x = (float)p.getX();
3341             float y = (float)p.getY();
3342             statusField.setText("Cursor: "+x+","+y);
3343             if(mPressed==true)
3344                mouseCoords.addElement(new afc.spatial.Point(x,y));
3345             statusValueField.setText("value: "+(dispR.getValue(x,y,true)));
3346          }
3347
3348          public void mouseDragged(MouseEvent e)
3349          {
3350             if(dragNo==0)
3351                mouseCoords = new Vector();
3352             mPressed=true;
3353    //         start = getGeoTrans(e.getX(),e.getY());
3354             afc.spatial.Point p = getGeoTrans(e.getX(),e.getY());
3355             float x = (float)p.getX();
3356             float y = (float)p.getY();
3357             statusField.setText("Dragging cursor: "+x+","+y);
3358             mouseCoords.addElement(p);
3359             statusValueField.setText("value: "+(dispR.getValue(x,y,true)));
3360             dragNo++;
3361          }
3362       }
```

```
3363
3364    }   // end class Ice_Model_Run
```

```
1   /*Umbrella class to control the climate model.  It takes inputs from
2    *Ice_Model_Run which creates the application panel and returns rasters and
3    *Vectors for drawing. © Andy Casely 2005.  Last updated 7th April. */
4
5   package afc.spatial;
6
7   import java.util.*;
8   import java.math.*;
9   import javax.swing.*;
10  import java.awt.geom.*;
11  import afc.utilities.*;
12  import afc.gui.*;
13
14  public class ClimateControl
15  {
16     private String[] monthNames = new String[13];
17     private int rows,cols,noData,ssr;
18     private float lim = 0.01f;
19     private float shiftT,shiftP;
20     private double xll,yll,xur,yur,cellSize;
21     private JTextArea messageOutput;
22     private Point ll,ur;
23     //boolean switches to check whether relevant previous parts of the model
24     //have been initialised prior to running a subsequent part
25     private boolean
   elaInit,elaHeightInit,lapseInitCheck,pddInit,snowfallInit,
26            tempInit,kInit,precipInit,goInitCheck,massBalanceInit,topoELAInit,
27            failedDrawFlag,kNewInit,gsStartInit,gsEndInit,gsInit,viewshedInit;

28     //Strings representing file and folder names
29     private String dem_name,root_dir,folder_dir,folder;
30     //selectFootprint is the footprint of the selected area.  maxFootprint
31     //is that of the whole Iceland raster
32     private Rectangle selectFootprint,icelandFP;
33     //lines and polygons to draw are stored as GeneralPaths in a Vector
34     private transient GeneralPath coords;
35     public Vector polyCoord;
36
37     //topoRaster stores the clipped topography to use as a template for
   drawing
38     //other Rasters. paintedRaster is the Raster to return as an object for
   drawing
39     private Raster topoRaster,snowcover;
40     //main model objects are and stored here
41     private Ice_model_Run ir;
42     private PrecipModel pm;
43     private PddSum pd;
44     private ELAFinderComplex efc;
45
46     //climate Seq instance variables - only initialised whe running a climate
   change model
47     private Vector[] climateOutputData;
48     private Vector climateOutputSummary;
49     private boolean[][] grassCover,birchCover,riversAndLakes;
50     private int[][] drainageBasins,combinedCover;
51     private int[] dBAreas;
52     private float
   snowPdd,vegPdd,oldTOffset,snowProb,snowfallf,tVal,absTOffset,dieOffRate,col
   onisationRateGrass,colonisationRateBirch;
53     private float[] totalSnow,snowArea,grassArea,birchArea;
54     private Raster vegcover4,vegcover7;
55     double ranVal;
56     Random ran = new Random();
57     int oldShiftP,pVal,nDB;
58
59  //--------Constructor-----------
60     public ClimateControl(Ice_model_Run ir,boolean hiLoFlag,JTextArea
   messageOutput) throws Exception
61     {
62        this.ir=ir;
63        this.messageOutput = messageOutput;
```



```
64  //     Raster initRaster;
65        monthNames = MonthsAndDays.getMonths();
66        polyCoord = new Vector();
67        String boundFile = "C:/gisdata/config_files/default_bounds.csv";
68        Vector defcoords = SimpleIO.getLines(boundFile);
69        double
   xll=Double.parseDouble((String)SimpleIO.getLineElement(defcoords,0,1,","))
70        double
   yll=Double.parseDouble((String)SimpleIO.getLineElement(defcoords,0,2,","))
71        double
   xur=Double.parseDouble((String)SimpleIO.getLineElement(defcoords,0,3,","))
72        double
   yur=Double.parseDouble((String)SimpleIO.getLineElement(defcoords,0,4,","))
73        icelandFP = new Rectangle(new Point(xll,yll),new Point(xur,yur));
74
75        shiftT=0;
76        shiftP=0;
77        goInitCheck = false;
78        tempInit = false;
79        precipInit = false;
80        viewshedInit=false;
81        pddInit = false;
82        kInit = false;
83        kNewInit= false;
84        gsInit= false;
85        gsStartInit= false;
86        gsEndInit= false;
87        snowfallInit = false;
88        massBalanceInit = false;
89        elaInit = false;
90        elaHeightInit = false;
91        topoELAInit=false;
92        lapseInitCheck = false;
93        failedDrawFlag=false;
94     }
95     //test constructor
96     public ClimateControl() throws Exception
97     {
98        topoRaster = new Raster("C:/gisdata/input_files/icedem_cp.asc");
99     }
100
101    //Area is selected, so save selected area's topography
102    //and initialise the rest of the model
103    public void initialiseIMS() throws Exception
104    {
105       messageOutput.append("\n"+"initialising model with input values...")
106       this.getAreaValues();
107       goInitCheck=true;
108       for(int i=50;i>0;i--)
109
   if(SimpleIO.checkFile(folder+"/"+folder_dir+"_vs_jan_"+i+".asc")==true)
110          {
111             viewshedInit=true;
112             ir.resetButtonColour(9);
113             ir.setSunShedRadius(i);
114             i=0;
115          }
116       Raster maxRaster = new
   Raster(root_dir+"input_files/"+ir.getDEMname(),yll,yur,"none");
117       SimpleIO.newDirectory(root_dir,folder_dir);
118       if(selectFootprint.contains(icelandFP)==false)
119       {
120          topoRaster=maxRaster.gridclip(selectFootprint);
121          SimpleIO.rasterToAscii(folder,folder_dir+"_topo",topoRaster);
122       }
123       else
124       {
125          topoRaster = maxRaster;
126          SimpleIO.rasterToAscii(folder,folder_dir+"_topo",topoRaster);

127       }
```

```
128
129         Raster snowcover = new Raster(topoRaster,0);
130         polyCoord = new Vector();
131
132         pm = new PrecipModel(this);
133         pd = new PddSum(this,pm);
134         efc = new ELAFinderComplex(this,pm,pd);
135         rows = topoRaster.getYcells();
136         cols = topoRaster.getXcells();
137         noData = topoRaster.getNoData();
138         cellSize = topoRaster.getCellSize();
139         pm.setVals(topoRaster);
140         pm.setFolder(folder);
141         pm.setFilePrefix(folder_dir);
142         messageOutput.append("\n"+"initialising complete.");
143      }
144
145   //---------Methods called by the GUI, usually by button presses----------
146   //most of these return rasters after running parts of the model
147
148      /*controls area selection by the GUI and displys either the topography
149       *the area or all of Iceland with a rectangular highlight area.  Checks
         if
150       *the selected area lies in Iceland.*/
151      public Raster displayArea(boolean highlight) throws Exception
152      {
153         polyCoord = new Vector();
154         getAreaValues();
155         Raster maxRaster;
156         if(highlight==true)
157            maxRaster = new Raster(root_dir+"input_files/icedem_cp.asc");
158         else
159            maxRaster = new
      Raster(root_dir+"input_files/"+ir.getDEMname(),yll,yur,"none");
160         System.out.println("**** "+ir.getDEMname());
161         maxRaster.setParameters(folder,"Iceland
      topography",folder_dir+"_topo","iceland_topo3",true);
162         topoRaster = maxRaster;
163         if(highlight==false)
164         {
165            if(icelandFP.contains(selectFootprint))
166            {
167               goInitCheck=true;
168               topoRaster = maxRaster.gridclip(selectFootprint);
169
      topoRaster.setParameters(folder,"Topography",folder_dir+"_topo","iceland_t
      po",true);
170   //
      topoRaster.setParameters(folder,"Topography",folder_dir+"_topo","elev_mass
      ",true);
171   /*          Raster elaR = new Raster(folder+"/"+folder_dir+"_ela_map.asc")
172               int elaCount=0;
173               int elaTotal=0;
174               for(int i=0;i<topoRaster.getXcells();i++)
175                  for(int j=0;j<topoRaster.getYcells();j++)
176                  if(elaR.getValue(i,j)!=-9999)
177                  {
178                     elaTotal+=elaR.getValue(i,j);
179                     elaCount++;
180                  }
181               elaTotal = elaTotal/elaCount;
182               System.out.println("ELA AVERAGE: "+elaTotal);
183               for(int i=0;i<topoRaster.getXcells();i++)
184                  for(int j=0;j<topoRaster.getYcells();j++)
185                  {
186
      if((topoRaster.getValue(i,j)!=0)&&(topoRaster.getValue(i,j)!=-9999))
187                     topoRaster.changeValue(i,j,1100);
188                  else
189                     topoRaster.changeValue(i,j,-9999);
190                  }
```



```
191               topoRaster.setParameters(folder,"Topography with ELA
      colours",folder_dir+"_topo","ela",false);
192               topoRaster.reclass(1000,200);*/
193            }
194            else
195            {
196               selectFootprint = selectFootprint.minEnclRect(icelandFP);
197
      ir.setAreaValues(selectFootprint.getLL(),selectFootprint.getUR());
198               topoRaster = maxRaster.gridclip(selectFootprint);
199
      topoRaster.setParameters(folder,"Topography",folder_dir+"_topo","iceland_t
      po",true);
200               messageOutput.append("\n"+"Clip footprint outside raster area
      MER displayed.");
201            }
202         }
203         else if(highlight==true)
204         {
205            if(!icelandFP.contains(selectFootprint))
206            {
207               selectFootprint = selectFootprint.minEnclRect(icelandFP);
208
      ir.setAreaValues(selectFootprint.getLL(),selectFootprint.getUR());
209               messageOutput.append("\n"+"Clip footprint outside raster area
      MER displayed.");
210            }
211            Vector polyRect = selectFootprint.getRectPointsForDrawing();
212            polyCoord.addElement(polyRect);
213            highlight=false;
214         }
215         return topoRaster;
216      }
217
218      //import and/or save temperature data files for the selected footprint

219      public Raster dispTemp(int monthNo,float shiftT,boolean hiLoFlag) throw
      Exception
220      {
221         Raster tFile;
222         if(((tempInit==false)||(shiftT!=this.shiftT))&&(goInitCheck==true))
223         {
224            this.shiftT=shiftT;
225            pm.createTempFiles(topoRaster,selectFootprint,shiftT,hiLoFlag);
226         }
227         if(goInitCheck==true)
228         {
229            newTParams(shiftT);
230            if(monthNo>0)
231            {
232               tFile = new
      Raster(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"0.asc");
233               tFile.setParameters(folder,"Temp
      "+monthNames[monthNo]+(int)(shiftT*10),folder_dir+"_"+monthNames[monthNo]+
      0","temp",false);
234            }
235            else
236            //returns the annual temp file (0 entered)
237            {
238               tFile = new Raster(folder+"/"+folder_dir+"_Annual_T0.asc");
239               tFile.setParameters(folder,"Annual Temp
      "+(int)(shiftT*10),folder_dir+"_Annual_T0","temp",false);
240            }
241            for(int i=0;i<tFile.getXcells();i++)
242               for(int j=0;j<tFile.getYcells();j++)
243                  if(tFile.getValue(i,j)!=noData)
244                     tFile.changeValue(i,j,tFile.getValue(i,j)+shiftT);
245         }
246         else
247         {
248            tFile = new Raster();
```

```
249            messageOutput.append("\n"+"model not started");
250         }
251         return tFile;
252      }
253
254      public void newTParams(float shiftT)
255      {
256         if((this.shiftT!=shiftT)||(tempInit==false))
257         {
258            tempInit=true;
259            massBalanceInit=false;
260            snowfallInit=false;
261            elaInit = false;
262            elaHeightInit = false;
263            gsInit=false;
264            pddInit=false;
265            ir.resetButtonColour(1);
266         }
267      }
268
269      public int setKVals(int kc) throws Exception
270      {
271         if((tempInit==true)&&(goInitCheck==true))
272         {
273            try
274            {
275               kc = pm.setTempKFactors(kc);
276               kInit=true;
277               ir.resetButtonColour(2);
278            }
279            catch(Exception e)
280            {
281               messageOutput.append("\n"+"No K values previously calculated.
   Calculating values...");
282               messageOutput.append("\n"+"K coarsening factor = "+kc);
283               pm.fitComplexSine(kc);
284               pm.writeAmpAndMedian(topoRaster);
285               pm.writeKasciis(topoRaster,kc);
286               kInit=true;
287               massBalanceInit=false;
288               snowfallInit=false;
289               elaInit = false;
290               elaHeightInit = false;
291               gsInit=false;
292               kInit=true;
293               ir.resetButtonColour(2);
294            }
295         }
296         else
297            messageOutput.append("\n"+"Temperature not imported. Get
   temperature first");
298         return kc;
299      }
300
301      //import and save precipitation data files for the selected footprint

302      public Raster displayPrecip(int monthNo,int shiftP,boolean hiLoFlag)
   throws Exception
303      {
304         Raster pFile;
305         if(((precipInit==false)||(shiftP!=this.shiftP))&&(goInitCheck==true))
306         {
307            this.shiftP=shiftP;
308            pm.importPrecipData(topoRaster,shiftP,hiLoFlag);
309         }
310         if(goInitCheck==true)
311         {
312            newPParams(shiftP);
313            if(monthNo>0)
314            {
315               pFile = new
```



```
315   Raster(folder+"/"+folder_dir+"_ppt_"+monthNames[monthNo]+".asc");
316               pFile.setParameters(folder,"Precip
   "+monthNames[monthNo],folder_dir+"_ppt_"+monthNames[monthNo],"precip_month
   ,false);
317            }
318            else
319            //returns the annual temp file (0 entered)
320            {
321               pFile = new Raster(folder+"/"+folder_dir+"_ppt_ann.asc");
322               pFile.setParameters(folder,"Annual
   Precipitation",folder_dir+"_ppt_ann","precip",false);
323            }
324            for(int i=0;i<pFile.getXcells();i++)
325               for(int j=0;j<pFile.getYcells();j++)
326                  if(pFile.getValue(i,j)!=noData)
327
   pFile.changeValue(i,j,(1+((float)shiftP/100))*pFile.getValue(i,j));
328            pFile.reclass(3000,500);
329         }
330         else
331         {
332            pFile = new Raster();
333            messageOutput.append("\n"+"model not started");
334         }
335         return pFile;
336      }
337
338      public void newPParams(int shiftP)
339      {
340         if((this.shiftP!=shiftP)||(precipInit==false))
341         {
342            precipInit=true;
343            massBalanceInit=false;
344            snowfallInit=false;
345            elaInit = false;
346            elaHeightInit = false;
347            ir.resetButtonColour(0);
348         }
349      }
350
351      public Raster getGlimmerData(int fr,boolean hiLoFlag)
352      {
353         Raster seaLevelTemp;
354         try
355         {
356            Raster tempRaster = pm.getGlimmerParams(topoRaster);
357            messageOutput.append("\n"+"Saved Glimmer temp & precip data for
   "+folder_dir+".");
358            getLapseRate(fr,hiLoFlag,true);
359            Raster lR = new Raster(folder+"/"+folder_dir+"_lapse_rate.asc");
360
   SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_tempL
   pse",lR);
361
362            seaLevelTemp = new Raster(topoRaster,noData);
363            for(int i=0;i<cols;i++)
364               for(int j=0;j<rows;j++)
365
   if((topoRaster.getValue(i,j)!=0)&&(topoRaster.getValue(i,j)!=noData))
366                  seaLevelTemp.changeValue(i,j,tempRaster.getValue(i,j)-
367                     (lR.getValue(i,j)*topoRaster.getValue(i,j)));
368
   SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_tempS
   aLevel",seaLevelTemp);
369         }
370         catch(Exception e)
371         {
372            messageOutput.append("\n"+"Could not save Glimmer data...");
373            seaLevelTemp = new Raster();
374         }
375         seaLevelTemp.setParameters(folder+"/"+folder_dir+"_glimmer","Annual
```

```
375  (sea level)",folder_dir+"_tempSeaLevel","temp",false);
376        return seaLevelTemp;
377     }
378
379     /*PDD requires monthly temp data to run. The empirical equation used
380      *to generate a statistical spread of daily values was calculated in
     Excel
381      * and is already coded
382      *PDD for snow includes a coding of influence of direct sunlight*/
383     public Raster pddManager(float snowIce,boolean snowVeg,float vegLim,flo
     ssf) throws Exception
384     {
385  //    System.out.println("::::::::::::: "+snowIce+" "+snowVeg+" "+vegLim+"
     "+ssf);
386        Raster pddMonth,pddFile;
387        float val;
388        if((tempInit==true)&&(goInitCheck==true)&&(viewshedInit==true))
389        {
390           pddFile=pd.createPddFiles(topoRaster,snowIce,snowVeg,vegLim,ssf);
391           if(snowVeg==true)
392           {
393              pddFile.setParameters(folder,"Annual Snow
     PDD",folder_dir+"_Annual_snow_PDD","pdd_ann",false);
394              ir.resetButtonColour(5);
395              massBalanceInit=false;
396              elaInit = false;
397              elaHeightInit = false;
398           }
399           else
400           {
401              pddFile.setParameters(folder,"Annual Veg
     PDD",folder_dir+"_Annual_veg_PDD","pdd_veg",false);
402              ir.resetButtonColour(3);
403              gsInit=false;
404           }
405           pddInit=true;
406           if(snowVeg==true)
407
     SimpleIO.writeComparisonVals(folder+"/compare_ablation_topo.csv",null,0,pd
     File,topoRaster,0);
408        }
409        else
410        {
411           messageOutput.append("\n"+"Temperature not yet calculated or mode
     not initialised. PDD not calculated.");
412           pddFile = drawAnX();
413        }
414        return pddFile;
415     }
416
417     public Raster dispPdd(int monthNo,boolean snowVeg) throws Exception
418     {
419        Raster pddFile;
420        if((pddInit==true)&&(goInitCheck==true))
421        {
422           if(snowVeg==true)
423           {
424              if((monthNo>0)&&(monthNo<=12))
425              {
426                 pddFile = new
     Raster(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"_snow_PDD.asc");
427                 pddFile.setParameters(folder,"Snow PDD
     "+monthNames[monthNo],folder_dir+"_"+monthNames[monthNo]+"_snow_PDD","pdd_
     onth",false);
428              }
429              else
430              //returns the annual pdd file (0 entered)
431              {
432                 pddFile = new
     Raster(folder+"/"+folder_dir+"_Annual_snow_PDD.asc");
433                 pddFile.setParameters(folder,"Annual Snow
```



```
433  PDD",folder_dir+"_Annual_snow_PDD","pdd_ann",false);
434              }
435           }
436           else
437           {
438              if((monthNo>0)&&(monthNo<=12))
439              {
440                 pddFile = new
     Raster(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"_veg_PDD.asc");
441                 pddFile.setParameters(folder,"Veg PDD
     "+monthNames[monthNo],folder_dir+"_"+monthNames[monthNo]+"_veg_PDD","pdd_v
     eg_month",false);
442              }
443              else
444              //returns the annual pdd file (0 entered)
445              {
446                 pddFile = new
     Raster(folder+"/"+folder_dir+"_Annual_veg_PDD.asc");
447                 pddFile.setParameters(folder,"Annual Veg
     PDD",folder_dir+"_Annual_veg_PDD","pdd_veg",false);
448              }
449           }
450        }
451        else
452        {
453           messageOutput.append("\n"+"Temperature or PDD not yet calculated.
     Original topography painted instead.");
454           pddFile = drawAnX();
455        }
456        return pddFile;
457     }
458
459     public Raster findGS(float pddVegVal) throws Exception
460     {
461        Raster gsFile;
462        if((kInit==true)&&(goInitCheck==true))
463        {
464           gsStartInit=false;
465           gsEndInit = false;
466           ir.resetButtonColour(4);
467           pm.findGrowingSeason(pddVegVal,lim);
468           pm.writeGSAsciis(topoRaster);
469           gsFile = new Raster(folder+"/"+folder_dir+"_GS_length.asc");
470           gsFile.setScaleType("");
471           gsFile.setParameters(folder,"GS
     Length",folder_dir+"_GS_length","gs_length",false);
472           gsInit=true;
473        }
474        else
475        {
476           messageOutput.append("\n"+"Temperature K not calculated yet.
     Original topography painted instead.");
477           gsFile = drawAnX();
478        }
479        return gsFile;
480     }
481
482     public Raster displayGS(boolean startEnd) throws Exception
483     {
484        Raster gsFile;
485        if((gsInit==true)&&(goInitCheck==true))
486        {
487           gsStartInit=true;
488           gsEndInit = true;
489           if(startEnd==true)
490           {
491              gsFile = new Raster(folder+"/"+folder_dir+"_GS_start.asc");
492              gsFile.setParameters(folder,"GS
     Start",folder_dir+"_GS_start","gs_date",false);
493           }
494           else
```

```
495             {
496                gsFile = new Raster(folder+"/"+folder_dir+"_GS_end.asc");
497                gsFile.setParameters(folder,"GS
    End",folder_dir+"_GS_end","gs_date",false);
498             }
499          }
500          else
501          {
502             messageOutput.append("\n"+"Growing season not yet calculated.
    Original topography painted instead.");
503             gsFile = drawAnX();
504          }
505          return gsFile;
506       }
507
508       public Raster displayGSmeanT() throws Exception
509       {
510          Raster gsFile;
511          if((gsInit==true)&&(goInitCheck==true))
512          {
513             gsFile = new Raster(folder+"/"+folder_dir+"_GS_temp.asc");
514             gsFile.setParameters(folder,"GS
    Temp",folder_dir+"_GS_temp","temp",false);
515          }
516          else
517          {
518             messageOutput.append("\n"+"Growing season not yet calculated.
    Original topography painted instead.");
519             gsFile = drawAnX();
520          }
521          return gsFile;
522       }
523
524       public void resetVegcover() throws Exception
525       {
526          vegcover4 = new Raster(topoRaster,0);
527          vegcover7 = new Raster(topoRaster,0);
528       }
529
530       public void resetSnowcover() throws Exception
531       {
532          snowcover = new Raster(topoRaster,0);
533          SimpleIO.rasterToAscii(folder,folder_dir+"_snowcover",snowcover);
534       }
535
536       public Raster findSnowcover(int monthNo,float tLim,int aa,float ai)
    throws Exception
537       {
538          float snowProb,aspectFactor,gradientFactor,snowfallf;
539          if(SimpleIO.checkFile(folder+"/"+folder_dir+"_snowcover.asc")==false
540             resetSnowcover();
541          else
542             snowcover = new
    Raster(root_dir+folder_dir+"/"+folder_dir+"_snowcover.asc");
543          snowcover.report();
544          Raster pddFile = new
    Raster(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"_snow_PDD.asc");
545          Raster precipFile = new
    Raster(folder+"/"+folder_dir+"_ppt_"+monthNames[monthNo]+".asc");
546          for(int i=0;i<pddFile.getXcells();i++)
547             for(int j=0;j<pddFile.getYcells();j++)
548                if(pddFile.getValue(i,j)!=noData)
549                {
550                   aspectFactor = (float)topoRaster.getAspect(i,j,false)-aa;
551                   if(aspectFactor<-180)
552                      aspectFactor+=360;
553                   aspectFactor = (float)(Math.abs(aspectFactor)/90)-1;

554                   if(aspectFactor>1)
555                      aspectFactor=0;
556                   aspectFactor = aspectFactor/2;
```



```
557                gradientFactor = (float)topoRaster.getGradient(i,j,false)/4

558
559                snowcover.addValue(i,j,-pddFile.getValue(i,j));
560                snowProb =
    1-pd.probabilityCalc(pm.getTValue(monthNo-1,i,j),tLim,noData);
561                snowfallf = (precipFile.getValue(i,j)*snowProb);
562                snowfallf = snowfallf*(1-(aspectFactor*gradientFactor*ai));
563                if(snowfallf<0)
564                   snowfallf=0;
565                snowcover.addValue(i,j,snowfallf);
566                if(snowcover.getValue(i,j)<0)
567                   snowcover.changeValue(i,j,0);
568                if((i==1)&&(j==1))
569                   System.out.println("AAAA snow: "+monthNo+"
    "+-pddFile.getValue(i,j)+"   "+snowfallf);
570
571             }
572          SimpleIO.rasterToAscii(folder,folder_dir+"_snowcover",snowcover);
573          snowcover.setParameters(folder,"Snowcover
    "+monthNames[monthNo],folder_dir+"_snowcover","snowcover",false);
574          return snowcover;
575       }
576
577       public void tempPrecipShift(float tVal,int pVal) throws Exception
578       {
579          pm.tempPrecipShift(tVal,(int)shiftP,pVal);
580          shiftP = pVal;
581       }
582
583       public void sensitivityInitialise() throws Exception
584       {
585          pm.calculateMonthlyP();
586       }
587
588       public void climateSeqInitialise(Raster rAndL,Raster db) throws Excepti
589       {
590          colonisationRateGrass = 0.02f;
591          colonisationRateBirch = 0.02f;
592          dieOffRate = 0.33f;
593          grassCover = new boolean[cols][rows];
594          birchCover = new boolean[cols][rows];
595          riversAndLakes = new boolean[cols][rows];
596          drainageBasins = new int[cols][rows];
597          combinedCover = new int[cols][rows];
598          oldTOffset=0;
599          oldShiftP=0;
600          pm.calculateMonthlyP();
601          snowcover = new Raster(topoRaster,0);
602          vegcover4 = new Raster(topoRaster,0);
603          vegcover7 = new Raster(topoRaster,0);
604          absTOffset=0;
605          nDB = 0;
606          for(int i=0;i<cols;i++)
607             for(int j=0;j<rows;j++)
608             {
609                if((int)db.getValue(i,j)>nDB)
610                   nDB = (int)db.getValue(i,j);
611                if(rAndL.getValue(i,j)>0)
612                   riversAndLakes[i][j]=true;
613                drainageBasins[i][j]=(int)db.getValue(i,j);
614             }
615          nDB+=1;
616          dBAreas = new int[nDB];
617          for(int i=0;i<cols;i++)
618             for(int j=0;j<rows;j++)
619                dBAreas[(int)db.getValue(i,j)]+=1;
620          climateOutputData = new Vector[nDB];
621          climateOutputSummary = new Vector();
622          Vector dataRow = new Vector();
623          dataRow.addElement("Timestep (years)");
```

```
624         dataRow.addElement("Temp Anomaly (C)");
625         dataRow.addElement("Precip Anomaly (%)");
626         dataRow.addElement("Birch Area (sq km)");
627         dataRow.addElement("Grass Area (sq km)");
628         dataRow.addElement("Unvegetated Area (sq km)");
629         dataRow.addElement("Snow Area (sq km)");
630         dataRow.addElement("Snow Volume");
631         for(int v=0;v<nDB;v++)
632         {
633            climateOutputData[v] = new Vector();
634            climateOutputData[v].addElement(dataRow);
635         }
636         climateOutputSummary.addElement(dataRow);
637         totalSnow = new float[nDB];
638         snowArea = new float[nDB];
639         grassArea = new float[nDB];
640         birchArea = new float[nDB];
641      }
642
643      public Raster climateSeq(int iteration,Vector climateData,float pddK)
    throws Exception
644      {
645         Vector dataRow = new Vector();
646         for(int v=0;v<nDB;v++)
647         {
648            totalSnow[v]=0;
649            snowArea[v]=0;
650            grassArea[v]=0;
651            birchArea[v]=0;
652         }
653         float cellArea = (float)((cellSize*cellSize)/1000000f);
654   //    Raster outSnow = new Raster(topoRaster,0);
655   //    Raster outAbl = new Raster(topoRaster,0);
656
657         if(iteration==0)
658         {
659            combinedCover = new int[cols][rows];
660            tVal =
    Float.parseFloat((String)SimpleIO.getLineElement(climateData,0,0,","));
661            pVal =
    Integer.parseInt((String)SimpleIO.getLineElement(climateData,0,1,","));
662            pm.tempPrecipShift(tVal,oldShiftP,pVal);
663            vegcover4 = pddManager(pddK,false,4f,0f);
664   //       vegcover4.printIntVals();
665            vegcover7 = pddManager(pddK,false,7.5f,0f);
666            for(int i=0;i<cols;i++)
667               for(int j=0;j<rows;j++)
668               {
669                  if(vegcover4.getValue(i,j)>0)
670                  {
671                     combinedCover[i][j] = -1;
672                     grassCover[i][j]=true;
673                     grassArea[drainageBasins[i][j]]+=cellArea;
674                  }
675                  if(vegcover7.getValue(i,j)>0)
676                  {
677                     combinedCover[i][j] = -2;
678                     birchCover[i][j]=true;
679                     birchArea[drainageBasins[i][j]]+=cellArea;
680                     grassArea[drainageBasins[i][j]]-=cellArea;
681                  }
682
683               }
684         oldTOffset=tVal;
685         oldShiftP=pVal;
686         }
687         else
688         {
689            combinedCover = new int[cols][rows];
690            tVal =
    Float.parseFloat((String)SimpleIO.getLineElement(climateData,iteration,0,"
```



```
690   "));
691            tVal = tVal - oldTOffset;
692            pVal =
    Integer.parseInt((String)SimpleIO.getLineElement(climateData,iteration,1,"
    "));
693            pm.tempPrecipShift(tVal,oldShiftP,pVal);
694   //       Raster vegT = pm.dispArrayTemp(topoRaster,0);
695   //       vegT.printRasterData();
696   //       SimpleIO.readKeyboard();
697   float massbal = 0;
698            float pVals[][] = new float[cols][rows];
699            for (int monthNo=1;monthNo<13;monthNo++)
700               for(int i=0;i<cols;i++)
701                  for(int j=0;j<rows;j++)
702                     if(pm.getTValue(1,i,j)!=noData)
703                     {
704                        float tx = topoRaster.getValue(i,j);
705                        float snowAdjust  =
    (float)((0.0000004*(tx*tx*tx))-(0.0009*(tx*tx))+(0.4708*tx)-74.698);
706                        if(snowAdjust>250)
707                           snowAdjust=250;
708                        snowcover.addValue(i,j,snowAdjust/12f);
709
710                        snowProb =
    1-pd.probabilityCalc(pm.getTValue(monthNo-1,i,j),1,noData);
711                        snowfallf = (pm.getPValue(monthNo-1,i,j)*snowProb);
712                        if(snowfallf<0)
713                           snowfallf=0;
714                        pVals[i][j] = (int)pm.getTValue(monthNo-1,i,j);
715                        snowcover.addValue(i,j,snowfallf);
716                        snowPdd =
    pd.pddCalc(monthNo,pm.getTValue(monthNo-1,i,j),pddK,(float)noData,true);
717                        snowcover.addValue(i,j,-snowPdd);
718                        if(snowcover.getValue(i,j)<0)
719                           snowcover.changeValue(i,j,0);
720   //                     outSnow.addValue(i,j,snowfallf);
721   //                     outAbl.addValue(i,j,-snowPdd);
722                        if((i==10)&&(j==12))
723                        {
724                           System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    snow: "+-snowPdd+"    "+snowfallf+"    "+snowAdjust/12f);
725                           massbal+=snowfallf;
726                           massbal-=snowPdd;
727                        }
728
729                        if((monthNo==6)&&(snowcover.getValue(i,j)>0))
730                           combinedCover[i][j] = (int)snowcover.getValue(i,j)
731                        if(monthNo==1)
732                        {
733                           vegcover4.changeValue(i,j,0);
734                           vegcover7.changeValue(i,j,0);
735                        }
736   //                     if(snowcover.getValue(i,j)<=0)
737   //                     {
738                           vegcover4.addValue(i,j,
739
    pd.pddCalc(monthNo,pm.getTValue(monthNo-1,i,j),pddK,(float)noData,false));
740                           vegcover7.addValue(i,j,
741
    pd.pddCalc(monthNo,pm.getTValue(monthNo-1,i,j),pddK,(float)noData,false));
742   //                     }
743                     }
744   //       SimpleIO.rasterToAscii(folder,folder_dir+"_outSnow",outSnow);
745   //       SimpleIO.rasterToAscii(folder,folder_dir+"_outAbl",outAbl);
746            System.out.println("%%%%%%%%%%%%%%%%%%%%%%%%%%%%% mass: "+massbal)
747            for(int i=0;i<cols;i++)
748               for(int j=0;j<rows;j++)
749                  if(pm.getTValue(1,i,j)!=noData)
750                  {
751                     //set colonisation rate proportional to amount of
    surrounding veg
```

```
752                     colonisationRateGrass = 0.01f;
753                     colonisationRateBirch = 0.01f;
754                     for(int ii=i-1;ii<i+2;ii++)
755                        for(int jj=j-1;jj<j+2;jj++)
756
    if((ii>=0)&&(jj>=0)&&(ii<cols)&&(jj<rows)&&((ii!=i)||(jj!=j)))
757                           {
758                              if(grassCover[ii][jj])
759                                 colonisationRateGrass += 0.01f;
760                              if(birchCover[ii][jj])
761                                 colonisationRateBirch += 0.01f;
762                           }
763                     //remove river and lake cells from coverages
764                     if(riversAndLakes[i][j]==true)
765                     {
766                        grassCover[i][j]=false;
767                        birchCover[i][j]=false;
768                        combinedCover[i][j] = -3;
769                     }
770                     //no veg in areas of permanent snowcover (covered by sno
    in Sept)
771                     else if(combinedCover[i][j]!=0)
772                     {
773                        grassCover[i][j]=false;
774                        birchCover[i][j]=false;
775
    totalSnow[drainageBasins[i][j]]+=(combinedCover[i][j]/1000)*cellArea;
776                        snowArea[drainageBasins[i][j]]+=cellArea;
777                     }
778                     else
779                     {
780
    if((grassCover[i][j]==true)&&(vegcover4.getValue(i,j)<600)&&(ran.nextDoubl
    ()<=dieOffRate))
781                        {
782     //                    vegcover4.changeValue(i,j,0);
783                        grassCover[i][j]=false;
784                        }
785                        else
786                        {
787
    if((grassCover[i][j]==true)||((grassCover[i][j]==false)&&(vegcover4.getVal
    e(i,j)>=600)&&(ran.nextDouble()<=colonisationRateGrass)))
788                           {
789                              combinedCover[i][j] = -1;
790                              grassCover[i][j]=true;
791                              grassArea[drainageBasins[i][j]]+=cellArea;
792                           }
793                        }
794
795
    if((birchCover[i][j]==true)&&(vegcover7.getValue(i,j)<1125)&&(ran.nextDoub
    e()<dieOffRate))
796                        {
797     //                    vegcover7.changeValue(i,j,0);
798                        birchCover[i][j]=false;
799                        }
800                        else
801                        {
802
    if((birchCover[i][j]==true)||((birchCover[i][j]==false)&&(vegcover7.getVal
    e(i,j)>=1125)&&(ran.nextDouble()<=colonisationRateBirch)))
803                           {
804                              combinedCover[i][j] = -2;
805                              birchCover[i][j]=true;
806                              birchArea[drainageBasins[i][j]]+=cellArea;
807                              grassArea[drainageBasins[i][j]]-=cellArea;
808                           }
809                        }
810                     }
811               }
```



```
812
813               oldTOffset =
    Float.parseFloat((String)SimpleIO.getLineElement(climateData,iteration,0,"
    "));
814               oldShiftP = pVal;
815               System.out.println(iteration+" "+oldTOffset+" "+tVal);
816     //        snowcover.printVals(pVals);
817               System.out.println();
818            }
819            float[] summaryVals = new float[5];
820            for(int v=0;v<nDB;v++)
821            {
822               dataRow = new Vector();
823               dataRow.addElement(new Integer(iteration));
824               dataRow.addElement(new
    Float(Float.parseFloat((String)SimpleIO.getLineElement(climateData,iterati
    n,0,",",))));
825               dataRow.addElement(new
    Float(Float.parseFloat((String)SimpleIO.getLineElement(climateData,iterati
    n,1,",",))));
826               dataRow.addElement(new Float(birchArea[v]));
827               summaryVals[0] += birchArea[v];
828               dataRow.addElement(new Float(grassArea[v]));
829               summaryVals[1] += grassArea[v];
830               dataRow.addElement(new
    Float((cellArea*dBAreas[v])-(birchArea[v]+grassArea[v]+snowArea[v])));
831               dataRow.addElement(new Float(snowArea[v]));
832               summaryVals[2] += snowArea[v];
833               dataRow.addElement(new Float(totalSnow[v]));
834               summaryVals[3] += totalSnow[v];
835               climateOutputData[v].addElement(dataRow);
836            }
837            dataRow = new Vector();
838            dataRow.addElement(new Integer(iteration));
839            dataRow.addElement(new
    Float(Float.parseFloat((String)SimpleIO.getLineElement(climateData,iterati
    n,0,",",))));
840            dataRow.addElement(new
    Float(Float.parseFloat((String)SimpleIO.getLineElement(climateData,iterati
    n,1,",",))));
841            dataRow.addElement(new Float(summaryVals[0]));
842            dataRow.addElement(new Float(summaryVals[1]));
843            dataRow.addElement(new
    Float((cellArea*cols*rows)-(summaryVals[0]+summaryVals[1]+summaryVals[2]))
    ;
844            dataRow.addElement(new Float(summaryVals[2]));
845            dataRow.addElement(new Float(summaryVals[3]));
846            climateOutputSummary.addElement(dataRow);
847            return new Raster(snowcover,combinedCover);
848         }
849
850         public void exportClimateOutputData() throws Exception
851         {
852            if(nDB==0)
853
    SimpleIO.writeCsvArray(root_dir+folder_dir+"/"+folder_dir+"_data.csv",clim
    teOutputData[0]);
854            else
855            {
856               for(int v=0;v<nDB;v++)
857
    SimpleIO.writeCsvArray(root_dir+folder_dir+"/"+folder_dir+"_data_db"+v+".c
    v",climateOutputData[v]);
858
    SimpleIO.writeCsvArray(root_dir+folder_dir+"/"+folder_dir+"_data.csv",clim
    teOutputSummary);
859            }
860         }
861
862         public Raster findSunShed(int ssm,int ssr,boolean hiLoFlag) throws
    Exception
```

```
863         {
864            int i=0;
865            topoRaster = new Raster(folder+"/"+folder_dir+"_topo.asc");
866            ViewshedCalculator vsc = new
       ViewshedCalculator(topoRaster,ssr,24,hiLoFlag);
867            Raster viewshed;
868            for(i=50;i>0;i--)
869               if((SimpleIO.checkFile(folder+"/"+folder_dir+"_vs_jan_"+i+".asc
870                  &&(i>=ssr))
871               {
872                  ssr=i;
873                  messageOutput.append("\n"+"importing sunShed with a radius o
       "+i+".");
874                  i=0;
875               }
876            ir.setSunShedRadius(ssr);
877            this.ssr=ssr;
878            if((ssm>=1)&&(ssm<13))
879            {
880
       if(SimpleIO.checkFile(folder+"/"+folder_dir+"_vs_"+monthNames[ssm]+"_"+s
       ".asc")==false)
881               {
882                  vsc.calculateViewshed();
883                  vsc.storeViewshedMonths(folder,folder_dir);
884               }
885               viewshed = new
       Raster(folder+"/"+folder_dir+"_vs_"+monthNames[ssm]+"_"+ssr+".asc");
886               viewshed.setParameters(folder,"Viewshed score
       "+monthNames[ssm],folder_dir+"_vs_"+monthNames[ssm]+"_"+ssr,"viewshed_mo
       ",false);
887            }
888            else
889            {
890
       if(SimpleIO.checkFile(folder+"/"+folder_dir+"_vs_ann_"+ssr+".asc")==fals
891               {
892                  vsc.calculateViewshed();
893                  vsc.storeViewshedMonths(folder,folder_dir);
894               }
895               viewshed = new Raster(folder+"/"+folder_dir+"_vs_ann_"+ssr+".as
896               viewshed.setParameters(folder,"Annual Viewshed
       score",folder_dir+"_vs_ann_"+ssr,"viewshed_ann",false);
897            }
898            for(i=ssr-1;i>0;i--)
899               for(int m=0;m<13;m++)
900
       if(SimpleIO.checkFile(folder+"/"+folder_dir+"_vs_"+monthNames[m]+"_"+i+"
       c"))
901
       SimpleIO.deleteAscii(folder,folder_dir+"_vs_"+monthNames[m]+"_"+i);

902            viewshedInit=true;
903            return viewshed;
904         }
905
906         public Raster findSnowfall(float lt,float snowIceK,int aspectAngle,fl
       aspectInfluence) throws Exception
907         {
908            Raster snowFile;
909            if((precipInit==true)&&(kInit==true)&&(goInitCheck==true))
910            {
911
912
       pm.findComplexSnowfall(lt,lim,snowIceK,aspectAngle,aspectInfluence,topoR
       er);
913               topoRaster = new Raster(folder+"/"+folder_dir+"_topo.asc");
914               pm.writeSnowAsciis(topoRaster);
915               snowFile = new Raster(folder+"/"+folder_dir+"_snowfall.asc");
916               snowFile.setParameters(folder,"Annual
       Snowfall",folder_dir+"_snowfall","snow",false);
```



```
917               massBalanceInit=false;
918               elaInit = false;
919               elaHeightInit = false;
920               ir.resetButtonColour(6);
921               snowfallInit=true;
922
       SimpleIO.writeComparisonVals(folder+"/compare_snowfall_topo.csv",null,0,
       wFile,topoRaster,0);
923            }
924            else
925            {
926               messageOutput.append("\n"+"PDD or precipitation not yet calcula
       Original topography painted instead.");
927               snowFile = drawAnX();
928            }
929            return snowFile;
930         }
931
932         public Raster findMassBalance() throws Exception
933         {
934            Raster massBalance;
935            if((pddInit==true)&&(snowfallInit==true)&&(goInitCheck==true))
936            {
937               elaInit = false;
938               elaHeightInit = false;
939               Raster snow = new Raster(folder+"/"+folder_dir+"_snowfall.asc")
940               Raster ablation = new
       Raster(folder+"/"+folder_dir+"_Annual_snow_PDD.asc");
941               massBalance = new Raster(snow,noData);
942               for(int i=0;i<snow.getXcells();i++)
943                  for(int j=0;j<snow.getYcells();j++)
944                     if(snow.getValue(i,j)!=noData)
945                     {
946
       massBalance.changeValue(i,j,(snow.getValue(i,j)-ablation.getValue(i,j)))
947                        if(massBalance.getValue(i,j)<-15000)
948                           massBalance.changeValue(i,j,-15000);
949                     }
950               massBalance.setParameters(folder,"Mass
       Balance",folder_dir+"_mass_balance","massb",false);
951
       SimpleIO.rasterToAscii(folder,folder_dir+"_mass_balance",massBalance);
952
       SimpleIO.writeComparisonVals(folder+"/compare_mb_topo_all.csv",null,0,ma
       alance,topoRaster,0);
953               massBalanceInit=true;
954            }
955            else
956            {
957               messageOutput.append("\n"+"PDD sum or Snowfall not yet
       calculated");
958               massBalance = drawAnX();
959            }
960            return massBalance;
961         }
962
963         public Raster elaOffset(float lt,float snowIceK,int aspectAngle,float
       aspectInfluence,float ssf) throws Exception
964         {
965            Raster elaR;
966            if((pddInit==true)&&(snowfallInit==true)&&(goInitCheck==true))
967            {
968               elaInit=true;
969               ir.resetButtonColour(7);
970               elaHeightInit=false;
971
       efc.configure(cols,rows,noData,aspectAngle,aspectInfluence,ssf,ssr);
972               efc.calcELAoffset(lt,lim,snowIceK,topoRaster);
973               elaR = efc.writeELAOffsetAsciis();
974               elaR.setParameters(folder,"ELA
       Offset",folder_dir+"_ela_offset","offset",false);
```

```
975         }
976         else
977         {
978            elaR = drawAnX();
979            messageOutput.append("\n"+"PDD sum or Snowfall not yet calculat
    Topography painted instead");
980         }
981         return elaR;
982      }
983
984      public Raster elaHeight(int fr,boolean lapseNotChanged,boolean hiLoFl
    throws Exception
985      {
986         messageOutput.append("\n"+"***HILOFLAG: "+hiLoFlag);
987         Raster elaH;
988         if((elaInit==true)&&(lapseNotChanged==false)&&(goInitCheck==true))
989         {
990            elaH = efc.calculateELA(fr,hiLoFlag,icelandFP);
991            elaH.setParameters(folder,"ELA
    Height",folder_dir+"_ela_map","ela",false);
992            elaH.reclass(1000,200);
993            ir.resetButtonColour(8);
994            elaHeightInit=true;
995         }
996         else if((elaInit==true)&&(lapseNotChanged==true)&&(goInitCheck==tr
997         {
998            elaH = efc.calculateELAexistingLapse();
999            elaH.setParameters(folder,"ELA
    Height",folder_dir+"_ela_map","ela",false);
1000           elaH.reclass(1000,200);
1001           ir.resetButtonColour(8);
1002           elaHeightInit=true;
1003        }
1004        else
1005        {
1006           messageOutput.append("\n"+"ELA offset not yet calculated.");
1007           elaH = drawAnX();
1008        }
1009        return elaH;
1010     }
1011
1012     public Raster getLapseRate(int fr,boolean hiLoFlag,boolean tOrP) thro
    Exception
1013     {
1014        Raster lapseR;
1015        if(goInitCheck==true)
1016        {
1017           efc.configure(cols,rows,noData,0,0,0,ssr);
1018           lapseR =
    efc.calculateLapseRate(fr,topoRaster,hiLoFlag,icelandFP,tOrP);
1019           if(tOrP==true)
1020              lapseR.setParameters(folder,"T Lapse
    Rate",folder_dir+"_Tlapse","lapse",false);
1021           else
1022              lapseR.setParameters(folder,"P Lapse
    Rate",folder_dir+"_Plapse","lapse",false);
1023        }
1024        else
1025        {
1026           messageOutput.append("\n"+"Model not started. Press GO to
    continue.");
1027           lapseR = drawAnX();
1028        }
1029        return lapseR;
1030     }
1031
1032     public Raster findTopoELA(int shiftVal) throws Exception
1033     {
1034        Raster topoELA;
1035        if(elaHeightInit==true)
1036        {
```



```
1037           Raster elaR = new Raster(folder+"/"+folder_dir+"_ela_map.asc");
1038           Raster topoR = new Raster(folder+"/"+folder_dir+"_topo.asc");
1039           topoELA = new Raster(elaR,noData);
1040           for(int i=0;i<topoR.getXcells();i++)
1041              for(int j=0;j<topoR.getYcells();j++)
1042                 if(elaR.getValue(i,j)!=noData)
1043
    topoELA.changeValue(i,j,(topoR.getValue(i,j)+shiftVal-elaR.getValue(i,j)
1044           topoELA.setParameters(folder,"Topo-ELA
    difference",folder_dir+"_ela_topo_diff","topo-ela",false);
1045           SimpleIO.rasterToAscii(folder,folder_dir+"_ela_topo_diff",topoE
1046           topoELAInit=true;
1047        }
1048        else
1049        {
1050           messageOutput.append("\n"+"ELA height not yet calculated");
1051           topoELA = drawAnX();
1052        }
1053        return topoELA;
1054     }
1055
1056     public Raster calculateAIG(int fr,boolean hiLoFlag) throws Exception
1057     {
1058        Raster tLapse,pLapse,tSelect,pSelect,aigR,annualP;
1059        aigR = new Raster(topoRaster,noData);
1060        if((tempInit==true)&&(precipInit==true))
1061        {
1062           annualP = new Raster(folder+"/"+folder_dir+"_ppt_ann.asc");
1063           tSelect = new Raster(topoRaster,pm.getSummerT());
1064           pSelect = new Raster(topoRaster,pm.getWinterP());
1065           efc.configure(cols,rows,noData,0,0,0,ssr);
1066           tLapse =
    efc.calculateLapseRate(fr,topoRaster,hiLoFlag,icelandFP,true);
1067           tLapse.setParameters(folder,"T Lapse
    Rate",folder_dir+"_Tlapse","lapse",false);
1068           pLapse =
    efc.calculateLapseRate(fr,topoRaster,hiLoFlag,icelandFP,false);
1069           pLapse.setParameters(folder,"P Lapse
    Rate",folder_dir+"_Plapse","lapse",false);
1070           float ln915 = (float)Math.log(0.915);
1071           float aig1,aig2;
1072           for(int i=0;i<cols;i++)
1073              for(int j=0;j<rows;j++)
1074              {
1075                 //% precip lapse rate per 100m
1076
    pLapse.changeValue(i,j,(pLapse.multiplyValue(i,j,100)/annualP.getValue(i
    ));
1077                 //precip in m
1078                 pSelect.multiplyValue(i,j,0.001f);
1079                 //t lapse as positive units & per 100m
1080                 tLapse.multiplyValue(i,j,-100);
1081                 aig1 =
    ln915+(0.339f*tSelect.getValue(i,j))-(float)Math.log(pSelect.getValue(i,
1082                 aig2 =
    (float)Math.log(1+pLapse.getValue(i,j))+(0.339f*tLapse.getValue(i,j));
1083                 if(aig1/aig2>0)
1084
    aigR.changeValue(i,j,topoRaster.getValue(i,j)+((aig1/aig2)*100));
1085                 else
1086                    aigR.changeValue(i,j,3000);
1087 //               System.out.println(i+","+j+"  "+tSelect.getValue(i,j)+"
    "+tLapse.getValue(i,j)+"  "+pSelect.getValue(i,j)+"
    "+pLapse.getValue(i,j)+"  "+aigR.getValue(i,j));
1088              }
1089        }
1090        return aigR;
1091     }
1092
1093     public Raster calculateGBS(int fr,boolean hiLoFlag) throws Exception
```

```
1094      {
1095         Raster tLapse,pLapse,tSelect,pSelect,gbsR,annualP;
1096         gbsR = new Raster(topoRaster,noData);
1097         if((tempInit==true)&&(precipInit==true))
1098         {
1099            annualP = new Raster(folder+"/"+folder_dir+"_ppt_ann.asc");
1100            tSelect = new Raster(topoRaster,pm.getSummerT());
1101            pSelect = new Raster(topoRaster,pm.getWinterP());
1102            efc.configure(cols,rows,noData,0,0,0,ssr);
1103            tLapse =
      efc.calculateLapseRate(fr,topoRaster,hiLoFlag,icelandFP,true);
1104            tLapse.setParameters(folder,"T Lapse
      Rate",folder_dir+"_Tlapse","lapse",false);
1105            pLapse =
      efc.calculateLapseRate(fr,topoRaster,hiLoFlag,icelandFP,false);
1106            pLapse.setParameters(folder,"P Lapse
      Rate",folder_dir+"_Plapse","lapse",false);
1107            float gbs0,gbs1,gbs2;
1108            for(int i=0;i<cols;i++)
1109               for(int j=0;j<rows;j++)
1110               {
1111                  //% precip lapse rate per 100m
1112
      pLapse.changeValue(i,j,(pLapse.multiplyValue(i,j,100)/annualP.getValue(i
      ));
1113                  //precip in m
1114                  pSelect.multiplyValue(i,j,0.001f);
1115                  //adjust to sea level precip
1116
      pSelect.multiplyValue(i,j,(float)Math.pow((1+pLapse.getValue(i,j)),(1/(t
      Raster.getValue(i,j)/100))));
1117                  //t lapse as positive units & per 100m
1118                  tLapse.multiplyValue(i,j,-100);
1119
      tSelect.addValue(i,j,tLapse.getValue(i,j)*(topoRaster.getValue(i,j)/100)
1120                  gbs0 =
      (float)Math.log(((pSelect.getValue(i,j)*Math.pow((1+pLapse.getValue(i,j)
      topoRaster.getValue(i,j)/100)))/0.915)/0.339);
1121                  gbs1 =
      (tSelect.getValue(i,j)-(tLapse.getValue(i,j)*(topoRaster.getValue(i,j)/1
      ))-gbs0;
1122                  gbs2 = gbs1*(100/tLapse.getValue(i,j));
1123                  if(gbs2<0)
1124                     gbsR.changeValue(i,j,noData);
1125                  else
1126                     gbsR.changeValue(i,j,gbs2+topoRaster.getValue(i,j));
1127   //               System.out.println(i+","+j+"  "+tSelect.getValue(i,j)+"
      "+tLapse.getValue(i,j)+"  "+pSelect.getValue(i,j)+"
      "+pLapse.getValue(i,j)+"  "+aigR.getValue(i,j));
1128               }
1129         }
1130         return gbsR;
1131      }
1132
1133   //---------Other methods----------
1134
1135      //gathers and stores input geographic data
1136      public void getAreaValues() throws Exception
1137      {
1138         xll=ir.getXll();
1139         yll=ir.getYll();
1140         xur=ir.getXur();
1141         yur=ir.getYur();
1142         ll=new Point(xll,yll);
1143         ur=new Point(xur,yur);
1144         selectFootprint = new Rectangle(ll,ur);
1145         root_dir = ir.getRootDir();
1146         folder_dir = ir.getSubDir();
1147         folder = root_dir + folder_dir;
1148         dem_name = ir.getDEMname();
1149      }
```



```
1150
1151      //path indexes: 0 is a point, 1 is the start of a line, 2 is on a lin
1152      //3 is the end of a line, 4 is the end of a line with a path closure
1153      public Raster drawAnX()
1154      {
1155         Raster defaultR = topoRaster;
1156         polyCoord = new Vector();
1157         Vector polyX = new Vector();
1158         int[] coordIndex = new int[4];
1159         polyX.addElement(new
      Point(selectFootprint.getXll(),selectFootprint.getYll()));
1160         coordIndex[0]=1;
1161         polyX.addElement(new
      Point(selectFootprint.getXur(),selectFootprint.getYur()));
1162         coordIndex[1]=2;
1163         polyX.addElement(new
      Point(selectFootprint.getXur(),selectFootprint.getYll()));
1164         coordIndex[2]=1;
1165         polyX.addElement(new
      Point(selectFootprint.getXll(),selectFootprint.getYur()));
      polyCoord.addElement(coords);
1166         coordIndex[3]=2;
1167         polyX.addElement(coordIndex);
1168         polyCoord.addElement(polyX);
1169         defaultR.setParameters(folder,"Unsuccessful
      Raster",folder_dir+"_topo","iceland_topo2",false);
1170         failedDrawFlag=true;
1171         return defaultR;
1172      }
1173
1174      public boolean getFailedDrawFlag()
1175      {
1176         return failedDrawFlag;
1177      }
1178
1179      public void setFailedDrawFlag()
1180      {
1181         this.failedDrawFlag=false;
1182      }
1183
1184      public void restoreTopoRaster()
1185      {
1186         try{
1187         polyCoord=new Vector();
1188   //    topoRaster = maxRaster.gridclip(selectFootprint);
1189         }
1190         catch (Exception e2)
1191         {
1192            messageOutput.append("\n"+"Clip footprint outside raster area -
      restore failed.");
1193         }
1194      }
1195
1196      //---------acessor methods
1197
1198      public String[] getTempStrings()
1199      {
1200         String[] strings = new String[13];
1201         for(int i=0;i<12;i++)
1202            strings[i] = monthNames[i+1]+"_t";
1203         strings[12] = "Annual_T";
1204         return strings;
1205      }
1206
1207      public Raster getTopoRaster()
1208      {
1209         return topoRaster;
1210      }
1211      public String getFolder()
1212      {
1213         return folder;
```

```
1214      }
1215
1216      public String getSubDir()
1217      {
1218         return folder_dir;
1219      }
1220
1221      public String getRootDir()
1222      {
1223         return root_dir;
1224      }
1225
1226      public Vector getDisplayLines()
1227      {
1228         return this.polyCoord;
1229      }
1230
1231      public void resetDisplayLines()
1232      {
1233         this.polyCoord = new Vector();
1234      }
1235
1236      public int getSunShedRadius()
1237      {
1238         return this.ssr;
1239      }
1240   }
```

```
1    /*degree-day calculation class for Iceland incorporating intra-month
2     *variability, created by AFC 6/10/04.  Last modified on 7th April 2005*/
3    package afc.spatial;
4
5    //uses AFC's Raster and SimpleIO classes and the Java Vector class
6    import afc.utilities.*;
7    import java.util.*;
8
9    public class PddSum
10   {
11      private Raster meanR,pddR;
12      private String[] monthNames = new String[13];
13      private String folder,folder_dir;
14      private int[] days = new int[13];
15      private float[] zTable = new float[41];
16      private float[] pTable = new float[41];
17      private float[][] mean,max,min,pVals,pdd;
18      private int noData,nHor,nVer;
19
20      private PrecipModel pm;
21      private ClimateControl cmc;
22
23      private float kFactor;
24      private boolean snowVeg;
25
26      public PddSum(ClimateControl cmc,PrecipModel pm) throws Exception
27      {
28         //constructor - define month info and extract p-value table
29         monthNames = MonthsAndDays.getMonths();
30         days = MonthsAndDays.getDays();
31         snowVeg=true;
32         this.folder = cmc.getFolder();
33         this.folder_dir = cmc.getSubDir();
34         this.kFactor = 0.007f;
35         this.pVals = readPvals();
36         this.pm=pm;
37      }
38
39      //simple constructor
40      public PddSum() throws Exception
41      {
42         //constructor - define month info and extract p-value table
43         SimpleIO SimpleIO = new SimpleIO();
44         MonthsAndDays mad = new MonthsAndDays();
45         monthNames = mad.getMonths();
46         days = mad.getDays();
47         kFactor = 0.007f;
48         snowVeg=true;
49         this.pVals = this.readPvals();
50      }
51
52      public Raster createPddFiles(Raster topoRaster,float snowIce,boolean
    snowVeg,
53         float vegLim,float ssf) throws Exception
54      {
55         System.out.println(snowIce+" "+snowVeg+" "+vegLim+" "+ssf);
56         Raster pddSnowMonth,pddVegMonth,sunShedMonth;
57         float val;
58         int vsRes=0;
59         float ssp = ssf/2f;
60         this.snowVeg=snowVeg;
61         Raster pddSnowFile = new Raster(topoRaster,0);
62         Raster pddSnowSummer,pddSnowWinter;
63         Raster pddVegFile = new Raster(topoRaster,0);
64         if(snowVeg==true)
65         {
66            pddSnowSummer = new Raster(topoRaster,0);
67            pddSnowWinter = new Raster(topoRaster,0);
68         }
69         else
70         {
```



```
71            pddSnowSummer = new Raster();
72            pddSnowWinter = new Raster();
73         }
74         this.kFactor = snowIce;
75         noData = topoRaster.getNoData();
76         for(int vsr = 50;vsr>0;vsr--)
77            if(SimpleIO.checkFile(folder+"/"+folder_dir+"_vs_jan_"+vsr+".asc"
78            {
79               vsRes=vsr;
80               vsr=0;
81            }
82         System.out.println(". ssp: "+ssp+". ssf: "+ssf+". ssr: "+vsRes);
83         for(int monthNo=1;monthNo<13;monthNo++)
84         {
85            pddSnowMonth = new Raster(topoRaster,0);
86            pddVegMonth = new Raster(topoRaster,0);
87            if(snowVeg==true)
88               sunShedMonth = new
    Raster(folder+"/"+folder_dir+"_vs_"+monthNames[monthNo]+"_"+vsRes+".asc");
89            else
90               sunShedMonth = new Raster();
91            for(int i=0;i<topoRaster.getXcells();i++)
92               for(int j=0;j<topoRaster.getYcells();j++)
93               {
94                  val =
    pddCalc(monthNo,pm.getTValue(monthNo-1,i,j),snowIce,topoRaster.getNoData()
    nowVeg);
95                  if (pm.getIgnoreVal(i,j)==false)
96                  {
97                     if(snowVeg==true)
98                     {
99    //the sunShedMonth value ranges from 0 to 0.13 depending on time of year a
    aspect
100   //low values= less sun.
101                       val = val*(1-(ssp/12)+(sunShedMonth.getValue(i,j)*ssf
102                       if(val<0)
103                          val=0;
104                       if((monthNo>=5)&&(monthNo<=9))
105                          pddSnowSummer.addValue(i,j,val);
106                       else
107                          pddSnowWinter.addValue(i,j,val);
108                       pddSnowFile.addValue(i,j,val);
109                       pddSnowMonth.changeValue(i,j,val);
110                    }
111                    else
112                    {
113                       pddVegFile.addValue(i,j,val);
114                       if(pm.getTValue(monthNo-1,i,j)>vegLim)
115                          pddVegMonth.changeValue(i,j,val);
116                    }
117                 }
118                 else
119                 {
120                    pddSnowMonth.changeValue(i,j,topoRaster.getNoData());
121                    pddVegMonth.changeValue(i,j,topoRaster.getNoData());
122                    if(monthNo==12)
123                    {
124                       pddSnowFile.changeValue(i,j,topoRaster.getNoData());
125                       pddVegFile.changeValue(i,j,topoRaster.getNoData());
126                       if(snowVeg==true)
127                       {
128
    pddSnowSummer.changeValue(i,j,topoRaster.getNoData());
129
    pddSnowWinter.changeValue(i,j,topoRaster.getNoData());
130                       }
131                    }
132                 }
133              }
134           if(snowVeg==true)
135
```

```
135   SimpleIO.rasterToAscii(folder,folder_dir+"_"+monthNames[monthNo]+"_snow_PD
      ,pddSnowMonth);
136             else
137
      SimpleIO.rasterToAscii(folder,folder_dir+"_"+monthNames[monthNo]+"_veg_PDD
      pddVegMonth);
138          }
139
140          if(snowVeg==true)
141          {
142
      SimpleIO.rasterToAscii(folder,folder_dir+"_Annual_snow_PDD",pddSnowFile);
143             pddSnowFile.setParameters(folder,"Annual Snow
      PDD",folder_dir+"_Annual_snow_PDD","pdd_ann",false);
144
      SimpleIO.rasterToAscii(folder,folder_dir+"_Summer_snow_PDD",pddSnowSummer)
145             pddSnowFile.setParameters(folder,"Summer Snow
      PDD",folder_dir+"_Summer_snow_PDD","pdd_ann",false);
146
      SimpleIO.rasterToAscii(folder,folder_dir+"_Winter_snow_PDD",pddSnowWinter)
147             pddSnowFile.setParameters(folder,"Winter Snow
      PDD",folder_dir+"_Winter_snow_PDD","pdd_ann",false);
148             return pddSnowFile;
149          }
150          else
151          {
152             for(int i=0;i<topoRaster.getXcells();i++)
153                for(int j=0;j<topoRaster.getYcells();j++)
154                {
155
      if((pm.getIgnoreVal(i,j)==false)&&(pddVegFile.getValue(i,j)<(vegLim*150)))
156                      pddVegFile.changeValue(i,j,0);
157                else if(pm.getIgnoreVal(i,j)==true)
158                   pddVegFile.changeValue(i,j,-9999);
159                }
160
      SimpleIO.rasterToAscii(folder,folder_dir+"_Annual_veg_PDD",pddVegFile);
161             pddVegFile.setParameters(folder,"Annual Veg
      PDD",folder_dir+"_Annual_veg_PDD","pdd_veg_ann",false);
162             return pddVegFile;
163          }
164       }
165
166       public float getTotalPdd(float meanVal,float stdev,float snowIceK)
167       {
168          float totalPdd = 0;
169
170          for(int i=1;i<13;i++)
171             totalPdd+=pddCalc(i,meanVal,snowIceK,noData,true);
172          return totalPdd;
173       }
174
175       public float stdevCalc3(float meanVal)
176       {
177          return (float)(3.3057*Math.exp(-0.0577*meanVal));
178       }
179
180       //meanVal is the average temp for 1 month (eg May's average)
181       //pddCalc generates the PDD for 1 cell for 1 month
182       public float pddCalc(int monthNo,float meanVal,float kFactor,float
      noData,boolean snowVeg)
183       {
184          int j,hundredthInt,zInt;
185          float hundredths,tenths,pddTotal,stdev;
186          Float zFloat,hFloat;
187          boolean negative=false;
188
189          //only do probability calculation on cells that have data values
190          if(meanVal!=noData)
191          {
192       //empirical calculation of stdev generated from daily data
```



```
193             stdev = stdevCalc3(meanVal);
194       //System.out.println("stdev: "+stdev);
195
196       //loop through T's of -20 to +20 deg, calculating probabilities
197             for(j=-20;j<21;j++)
198             {
199                negative=false;
200                zTable[j+20] = (meanVal-(float)j)/stdev;
201
202          /* Z values need to be positive to calculate p values, but we
203           * need to remember that the value was originally negative */
204                if(zTable[j+20]<0)
205                {
206                   zTable[j+20] = -zTable[j+20];
207                   negative=true;
208                }
209          /* the p-value table runs up to z values of 4
210           *(4 s.d., so a v. low prob.) - restrict calculation to z<4 */
211                if(zTable[j+20]<4)
212                {
213          //calculate appropriate value to extract from p-table
214                   zTable[j+20] = zTable[j+20]*10;
215                   zFloat = new Float(zTable[j+20]);
216                   zInt = zFloat.intValue();
217                   zTable[j+20] = zTable[j+20]/10;
218                   tenths = (float)zInt/10;
219                   hundredths = zTable[j+20]-tenths;
220                   hFloat = new Float((float) hundredths*100);
221                   hundredthInt = hFloat.intValue()+1;
222          /* extract probability value, adjust sign and convert to
223           * 'probability of T greater than...' */
224                   if(negative==true)
225                      pTable[j+20] = 0.5f- pVals[hundredthInt][zInt];
226                   else
227                      pTable[j+20] = 0.5f+ pVals[hundredthInt][zInt];
228                }
229                //set p as 0 or 1 for z values >4, depending on +ve or -ve
230                else if(zTable[j+20]>=4)
231                {
232                   if(negative==true)
233                      pTable[j+20]=0;
234                   else
235                      pTable[j+20]=1;
236                }
237             }
238             //calculate probabilities between each degree (eg between 3 and 4
      deg)
239             for (j=-20;j<20;j++)
240             {
241                pTable[j+20]-=pTable[j+21];
242             }
243       //do pdd sum for one month
244             pddTotal=0;
245       /*for each degree category (eg 3 to 4 take a 'middle' value of 0.5 (ie
      3.5)
246        *and calculate pdd sum
247        *= probability * no of days in month * temperature * k conversion
      factor*/
248             for(j=0;j<20;j++)
249             {
250                //ie for ablation, with a k factor, & the total in mm rather t
      m
251                if(snowVeg==true)
252                {
253                   if(pTable[j+20]>0.01)
254
      pddTotal+=((pTable[j+20]*(float)days[monthNo]*((float)j+0.5f))*kFactor)*10
255                }
256                //for vegetation - a 'raw' pdd count
257                else
258                {
```

```
259                 if(pTable[j+20]>0.01)
260
    pddTotal+=(pTable[j+20]*(float)days[monthNo]*((float)(j+0.5f)));
261     //System.out.println(pTable[j+20]+" "+(float)days[monthNo]+"
    "+((float)j+0.5f));
262                 }
263             }
264         }
265         //if cell has a 'nodata' value, set pdd to noData
266         else
267             pddTotal = noData;
268         return pddTotal;
269     }
270
271   /*simplified version of pddCalc - simply outputs sthe probability of a
272    *temperature being above a selected value (lowerLimit)*/
273     public float probabilityCalc(float meanVal,float lowerLimit,float noDat
274     {
275         int j,hundredthInt,zInt;
276         float hundredths,tenths,probTotal,stdev;
277         Float zFloat,hFloat;
278         boolean negative=false;
279
280         if(meanVal!=noData)
281         {
282             stdev = stdevCalc3(meanVal);
283             for(j=-20;j<21;j++)
284             {
285                 negative=false;
286                 zTable[j+20] = (meanVal-(float)j)/stdev;
287                 if(zTable[j+20]<0)
288                 {
289                     zTable[j+20] = -zTable[j+20];
290                     negative=true;
291                 }
292                 if(zTable[j+20]<4)
293                 {
294                     zTable[j+20] = zTable[j+20]*10;
295                     zFloat = new Float(zTable[j+20]);
296                     zInt = zFloat.intValue();
297                     zTable[j+20] = zTable[j+20]/10;
298                     tenths = (float)zInt/10;
299                     hundredths = zTable[j+20]-tenths;
300                     hFloat = new Float((float) hundredths*100);
301                     hundredthInt = hFloat.intValue()+1;
302                     if(negative==true)
303                         pTable[j+20] = 0.5f- pVals[hundredthInt][zInt];
304                     else
305                         pTable[j+20] = 0.5f+ pVals[hundredthInt][zInt];
306                 }
307                 else if(zTable[j+20]>=4)
308                 {
309                     if(negative==true)
310                         pTable[j+20]=0;
311                     else
312                         pTable[j+20]=1;
313                 }
314             }
315             for (j=-20;j<20;j++)
316             {
317                 pTable[j+20]-=pTable[j+21];
318             }
319             probTotal=0;
320             for(j=0;j<20;j++)
321             {
322                 if((pTable[j+20]>0.01)&&(j>lowerLimit))
323                     probTotal+=pTable[j+20];
324             }
325         }
326         else
327             probTotal = noData;
```



```
328         return probTotal;
329     }
330
331     public float[][] readPvals() throws Exception
332     {
333         Vector pData =
    SimpleIO.getLines("C:/gisdata/config_files/p_values.txt");
334         Vector line;
335         String s;
336         float value;
337         float[][] pVals = new float[11][41];
338         // Loop through all the data, getting each line from the data file
339         // and parsing it
340         for (int j=1; j < 42; j++)
341         {
342             s = (String) pData.elementAt(j);
343             line = SimpleIO.parseLine(s," ");
344             for(int i=0; i < line.size(); i++)
345             {
346                 value = (float) Float.parseFloat((String) line.elementAt(i));
347                 pVals[i][j-1] = value;
348             }
349         }
350         return pVals;
351     }
352
353     public float[][] getPVals(){
354         return this.pVals;
355     }
356
357     public void setKFactor(float snowIce){
358         this.kFactor = snowIce;
359     }
360
361
362     public static void main(String args[]) throws Exception
363     {
364 // pddCalc(int monthNo,float meanVal,float kFactor,float noData,boolean
    snowVeg,int vegLim)
365         PddSum ps = new PddSum(new ClimateControl(),new PrecipModel());
366         System.out.println("PDD: "+ps.pddCalc(1,4,0.007f,-9999,false));
367     }
368 }
```

```
1   /* class to calculate mass balance and growing season.  Inputs are monthly
2    *temperature maps, annual precipitation map and precipitation range and
3    *alpha parameters(for a sine curve model of precipitation).
4    *Last modified 7th April 2005 */
5
6   package afc.spatial;
7
8   import java.util.*;
9   import java.math.*;
10  import afc.utilities.*;
11  import afc.spatial.*;
12
13  public class PrecipModel
14  {
15     //rasters and arrays required for precipitation data
16     private Raster rangeR,alphaR,pptR,meanTR,averageTR;
17     private float[][]
    alpha,range,ppt,snowfall,rt1,rt2,rt1GS,rt2GS,growSeason,
18              meanGSt;
19     //rasters and arrays required to import and process tempeature data
20     private boolean[][] ignore;
21     private float[][] averageT,maxVals,minVals,meanT;
22     private float[][] totalResid,kOne,kTwo,kThree,kFour,tAmp,tMed;
23     private float[][] bestPos,bestAmp,totalWinResid,bestWinPos,bestWinAmp;
24     //precipData only constructed when running climate change model
25     private float[][][] data,precipData;
26     private float
    amp1,amp2,lim,limitingT,aspectFactor,gradientFactor,aspectInfluence;
27     private int noData,nHor,nVer;
28     private double cellSize,yll,yur;
29     private ClimateControl cmc;
30     private String[] monthNames = new String[13];
31     private int[] days = new int[13];
32     private String folder,folder_dir,root_dir;
33     private boolean customFlag,customAnnFlag;
34
35     private int resolutionFactor = 1;
36
37     //----------constructor----------
38
39     //empty constructor, for use in ELAFinderComplex and Ice_model_Setup
40     public PrecipModel(ClimateControl cmc) throws Exception
41     {
42        this.cmc = cmc;
43        monthNames = MonthsAndDays.getMonths();
44        days = MonthsAndDays.getDays();
45        this.folder = cmc.getFolder();
46        this.folder_dir = cmc.getSubDir();
47        this.root_dir = cmc.getRootDir();
48        customFlag=true;
49        customAnnFlag=true;
50     }
51
52     public PrecipModel()
53     {
54     }
55
56     //---------methods---------
57
58     public void setVals(Raster topoRaster) throws Exception
59     {
60        nHor=topoRaster.getXcells();
61        nVer=topoRaster.getYcells();
62        yll = topoRaster.getYll();
63        yur = topoRaster.getYur();
64        this.noData=topoRaster.getNoData();
65        this.cellSize=topoRaster.getCellSize();
66
67        //set relevant arrays - temp
68        data = new float[12][nHor][nVer];
69        ignore = new boolean[nHor][nVer];
```



```
70        maxVals = new float[nHor][nVer];
71        minVals = new float[nHor][nVer];
72        totalResid = new float[nHor][nVer];
73
74  //    meanT = new float[nHor][nVer];
75        //set relevant arrays - precip
76        ppt = new float[nHor][nVer];
77        alpha = new float[nHor][nVer];
78        range = new float[nHor][nVer];
79
80        //set relevant arrays - snowfall
81        kOne = new float[nHor][nVer];
82        kTwo = new float[nHor][nVer];
83        kThree = new float[nHor][nVer];
84        kFour = new float[nHor][nVer];
85        tAmp = new float[nHor][nVer];
86        tMed = new float[nHor][nVer];
87        rt1 = new float[nHor][nVer];
88        rt2 = new float[nHor][nVer];
89        snowfall = new float[nHor][nVer];
90        rt1GS = new float[nHor][nVer];
91        rt2GS = new float[nHor][nVer];
92        growSeason = new float[nHor][nVer];
93        meanGSt = new float[nHor][nVer];
94
95        for (int i=0;i<nHor;i++)
96           for (int j=0;j<nVer;j++)
97           {
98              maxVals[i][j]=-Float.MAX_VALUE;
99              minVals[i][j]=Float.MAX_VALUE;
100             if(topoRaster.getValue(i,j)==0)
101             {
102                ignore[i][j]=true;
103                tAmp[i][j] = noData;
104                tMed[i][j] = noData;
105                rt1[i][j] = noData;
106                rt2[i][j] = noData;
107                snowfall[i][j] = noData;
108                rt1GS[i][j] = noData;
109                rt2GS[i][j] = noData;
110                growSeason[i][j] = noData;
111                meanGSt[i][j] = noData;
112             }
113             else
114                ignore[i][j] = false;
115          }
116    }
117
118    public Raster dispArrayTemp(Raster topoRaster,int monthNo)
119    {
120       Raster tFile = new Raster(topoRaster,0);
121       for(int i=0;i<nHor;i++)
122          for(int j=0;j<nVer;j++)
123          {
124             if(monthNo==0)
125             {
126                float totalTemp=0;
127                for(int m=0;m<12;m++)
128                   totalTemp+=data[m][i][j];
129                totalTemp/=12;
130                tFile.changeValue(i,j,totalTemp);
131             }
132             else
133             {
134                tFile.changeValue(i,j,data[monthNo-1][i][j]);
135             }
136          }
137       return tFile;
138    }
139
140    //extract and save monthly temperature files
```

```
141     public Raster createTempFiles(Raster topoRaster,Rectangle
    selectFootprint,float shiftT,boolean hiLoFlag) throws Exception
142     {
143        Raster tFile;
144        Raster tFileAnn = new Raster(topoRaster,0);
145        boolean changedTvals = false;
146        float val;
147        for (int i=0;i<nHor;i++)
148           for (int j=0;j<nVer;j++)
149           {
150              maxVals[i][j]=-Float.MAX_VALUE;
151              minVals[i][j]=Float.MAX_VALUE;
152           }
153        for (int monthNo=1;monthNo<13;monthNo++)
154        {
155           /*import temperature data for an offset of 0 if it doesn't exis
    or if
156            *the selected area is custom(not a preset)*/
157
    if((SimpleIO.checkFile(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"0..
    c")==false)
158
    ||((customFlag==true)&&(folder_dir.compareTo("custom_area")==0)))
159           {
160              if(hiLoFlag==false)
161                 tFile = new
    Raster(root_dir+"input_files/t_"+monthNames[monthNo]+"_cp.asc",yll,yur,"
    ne");
162              else
163                 tFile = new
    Raster(root_dir+"input_files/"+monthNames[monthNo]+"_m100.asc",yll,yur,"
    );
164              tFile=tFile.gridclip(selectFootprint);
165           }
166           else
167           {
168              tFile = new
    Raster(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"0.asc");
169           }
170        //total the mean annual temperature, including the ofset
171           Regression reg;
172           changedTvals = false;
173           for(int i=0;i<tFile.getXcells();i++)
174              for(int j=0;j<tFile.getYcells();j++)
175                 if(ignore[i][j]==false)
176                 {
177                    val = tFile.getValue(i,j);
178        /*put the temp data into the PrecipModel 3-D temp array with the
    temperature
179         *offset*/
180                    if(val==noData)
181                    {
182                       if(topoRaster.getValue(i,j)>0)
183                       {
184                          changedTvals = true;
185                          float[] demSample = topoRaster.boxValues(i,j,50,
186                          float[] tSample = tFile.boxValues(i,j,50,noData)
187                          reg=new Regression(demSample,tSample,noData);
188
    val=(reg.getGradient()*topoRaster.getValue(i,j))+reg.getIntercept();
189                          tFile.changeValue(i,j,val);
190                          data[monthNo-1][i][j]=val+shiftT;
191                          tFileAnn.addValue(i,j,val/12f);
192                       }
193                       else
194                       {
195                          data[monthNo-1][i][j]=noData;
196                          if(monthNo==1)
197                             tFileAnn.changeValue(i,j,noData);
198                       }
199                    }
```



```
200                    else
201                    {
202        //the 'shift T' goes into the data[][][] array but not saved to th
    files
203                       data[monthNo-1][i][j]=val+shiftT;
204                       tFileAnn.addValue(i,j,val/12f);
205                    }
206        //find max and min values
207                    if(data[monthNo-1][i][j]>maxVals[i][j])
208                       maxVals[i][j]=data[monthNo-1][i][j];
209                    if(data[monthNo-1][i][j]<minVals[i][j])
210                       minVals[i][j]=data[monthNo-1][i][j];
211        //last month, so all temp data in - set amp and median values valu
212                    if(monthNo==12)
213                       if(data[11][i][j]!=noData)
214                       {
215                          tAmp[i][j] = (maxVals[i][j] - minVals[i][j])/2;
216                          tMed[i][j] = maxVals[i][j] - tAmp[i][j];
217                       }
218                 }
219        System.out.println(tFile.getMaxVal(noData,noData)+" --MAXMIN--
    "+tFile.getMinValNotNoData(noData,noData));
220        // saving monthly temp: curently only saves t at an offset of 0
221
    if((SimpleIO.checkFile(folder+"/"+folder_dir+"_"+monthNames[monthNo]+"0..
    c")==false)
222              ||(changedTvals =
    true)||((customFlag==true)&&(folder_dir.compareTo("custom_area")==0)))
223           {
224
    SimpleIO.rasterToAscii(folder,folder_dir+"_"+monthNames[monthNo]+"0",tFi
    );
225              if(monthNo==12)
226                 customFlag=false;
227           }
228        }
229        //saving annual temp at an offset of 0
230
    if((SimpleIO.checkFile(folder+"/"+folder_dir+"_Annual_T0.asc")==false)
231           ||(changedTvals =
    true)||(customAnnFlag==true)&&(folder_dir.compareTo("custom_area")==0))
232           {
233
    SimpleIO.rasterToAscii(folder,folder_dir+"_Annual_T0",tFileAnn);
234              customAnnFlag=false;
235           }
236     //    fitSine(topoRaster);
237           tFileAnn.setParameters(folder,"Annual
    Temperature"+shiftT,folder_dir+"_Annual_T0","temp_ann",false);
238           return tFileAnn;
239        }
240
241        //a climate change method.
242        //shiftT needs to be the shift from year1 to year2, ie if +3 to -2,
    shift is -5
243        //requires precipData to have been set up.
244        public void tempPrecipShift(float shiftT,int oldShiftP,int newShiftP)
    throws Exception
245        {
246           for (int monthNo=1;monthNo<13;monthNo++)
247              for(int i=0;i<nHor;i++)
248                 for(int j=0;j<nVer;j++)
249                    if(ignore[i][j]==false)
250                    {
251                       data[monthNo-1][i][j]=data[monthNo-1][i][j]+shiftT;
252                       precipData[monthNo-1][i][j] =
    precipData[monthNo-1][i][j]/(1+((float)oldShiftP/100));
253                       precipData[monthNo-1][i][j] =
    precipData[monthNo-1][i][j]*(1+((float)newShiftP/100));
254                    }
255        }
```

```
256
257      public void writeAmpAndMedian(Raster topoRaster) throws Exception
258      {
259         Raster tempR = new Raster(topoRaster,kOne);
260         tempR.replaceRaster(tAmp);
261         SimpleIO.rasterToAscii(folder,folder_dir+"_tAmp",tempR);
262         tempR.replaceRaster(tMed);
263         SimpleIO.rasterToAscii(folder,folder_dir+"_tMed",tempR);
264      }
265
266      public Raster importPrecipData(Raster topoRaster,int shiftP,boolean
   hiLoFlag) throws Exception
267      {
268         Rectangle footprint = topoRaster.getFootprint();
269         System.out.println("getting ppt rasters");
270   //    if((SimpleIO.checkFile(folder+"/"+folder_dir+"_ppt_ann.asc")==fals
271   //       ||((customFlag==true)&&(folder_dir.compareTo("custom_area")==0)
272   //    {
273            rangeR = new
   Raster("C:/gisdata/input_files/p_rng_pcnt2.asc",yll,yur,"none");
274            rangeR=rangeR.gridclip(footprint);
275            alphaR = new
   Raster("C:/gisdata/input_files/ppt_alpha_cp.asc",yll,yur,"none");
276            alphaR=alphaR.gridclip(footprint);
277            pptR = new
   Raster("C:/gisdata/input_files/pptmap_cp.asc",yll,yur,"none");
278            pptR=pptR.gridclip(footprint);
279            if(hiLoFlag==true)
280            {
281               rangeR.increaseRes(250);
282               alphaR.increaseRes(250);
283               pptR.increaseRes(250);
284            }
285   /*    }
286         else
287         {
288            rangeR = new Raster(folder+"/"+folder_dir+"_ppt_range.asc");
289            alphaR = new Raster(folder+"/"+folder_dir+"_ppt_alpha.asc");
290            pptR = new Raster(folder+"/"+folder_dir+"_ppt_ann.asc");
291         }*/
292
293         for (int i=0;i<nHor;i++)
294            for(int j=0;j<nVer;j++)
295               if(ignore[i][j]==true)
296                  pptR.changeValue(i,j,noData);
297         SimpleIO.rasterToAscii(folder,folder_dir+"_ppt_ann",pptR);
298         SimpleIO.rasterToAscii(folder,folder_dir+"_ppt_range",rangeR);
299         SimpleIO.rasterToAscii(folder,folder_dir+"_ppt_alpha",alphaR);
300
301         //range is percentage value of total precipitation, so convert to
   fraction,
302         //multiply by ppt value and divide by 2 to convert to amplitude.
303         //precipitation values are annual averages derived from a map, so
304         //divide by 12 to be comparable to weather station values.
305         System.out.println("******ppt shift: "+shiftP+" "+nHor+" "+nVer);
306         for (int i=0;i<nHor;i++)
307            for(int j=0;j<nVer;j++)
308               if(ignore[i][j]==false)
309               {
310                  //convert from annual to monthly values and add offset %
311                  ppt[i][j] = pptR.multiplyValue(i,j,(1/12f));
312                  ppt[i][j] = (1+((float)shiftP/100))*ppt[i][j];
313                  //convert % range to mm amplitude and calculate sine offs
314                  range[i][j] =
   rangeR.multiplyValue(i,j,(pptR.getValue(i,j))/200f);
315                  alpha[i][j] = alphaR.getValue(i,j);
316               }
317               else
318               {
319                  range[i][j] = noData;
```



```
320                  alpha[i][j] = noData;
321                  ppt[i][j] = noData;
322               }
323         calculateMonthlyP(topoRaster);
324         pptR = new Raster(folder+"/"+folder_dir+"_ppt_ann.asc");
325         pptR.setParameters(folder,"Annual
   Precipitation",folder_dir+"_ppt_ann","precip",false);
326         return pptR;
327      }
328
329      public float[][] getSummerT()
330      {
331         float[][] tSelect = new float[nHor][nVer];
332         for (int i=0;i<nHor;i++)
333            for(int j=0;j<nVer;j++)
334               if(ignore[i][j]==false)
335               {
336                  for(int m=4;m<9;m++)
337                     tSelect[i][j]+=data[m][i][j];
338                  tSelect[i][j] = tSelect[i][j]/5f;
339               }
340               else
341                  tSelect[i][j]=noData;
342         return tSelect;
343      }
344
345      public float[][] getWinterP() throws Exception
346      {
347         float[][] pSelect = new float[nHor][nVer];
348         for(int m=1;m<13;m++)
349            if((m<=4)||(m>=10))
350            {
351               Raster precipMR = new
   Raster(folder+"/"+folder_dir+"_ppt_"+monthNames[m]+".asc");
352               for (int i=0;i<nHor;i++)
353                  for(int j=0;j<nVer;j++)
354                  {
355                     if(ignore[i][j]==false)
356                        pSelect[i][j]+=precipMR.getValue(i,j);
357                     else if(m==1)
358                        pSelect[i][j]=noData;
359                  }
360            }
361         return pSelect;
362      }
363
364      public Raster getGlimmerParams(Raster topoRaster) throws Exception
365      {
366         float[][] maxP = new float[nHor][nVer];
367         float[][] minP = new float[nHor][nVer];
368         float[][] meanP = new float[nHor][nVer];
369         float[][] rangeP = new float[nHor][nVer];
370         float[][] rangeT = new float[nHor][nVer];
371         Raster precipMR = new Raster();
372         for(int m=1;m<13;m++)
373         {
374            precipMR = new
   Raster(folder+"/"+folder_dir+"_ppt_"+monthNames[m]+".asc");
375            for (int i=0;i<nHor;i++)
376               for(int j=0;j<nVer;j++)
377               {
378                  if(m==1)
379                  {
380                     maxP[i][j] = precipMR.getValue(i,j);
381                     minP[i][j] = precipMR.getValue(i,j);
382                  }
383                  else
384                  {
385                     if(precipMR.getValue(i,j)>maxP[i][j])
386                        maxP[i][j] = precipMR.getValue(i,j);
387                     if(precipMR.getValue(i,j)<minP[i][j])
```

```
388                     minP[i][j] = precipMR.getValue(i,j);
389                  }
390               }
391   //       precipMR.printIntVals();
392         }
393         for (int i=0;i<nHor;i++)
394            for(int j=0;j<nVer;j++)
395            {
396               meanP[i][j] = ppt[i][j]*12;
397               rangeP[i][j] = range[i][j]*2;
398               rangeT[i][j] = tAmp[i][j]*2;
399            }
400         SimpleIO.newDirectory(folder+"/",folder_dir+"_glimmer");
401
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_top
      ,topoRaster);
402         precipMR.replaceRaster(meanP);
403
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_pre
      pAnn",precipMR);
404         precipMR.replaceRaster(rangeP);
405
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_pre
      pRange",precipMR);
406         precipMR.replaceRaster(maxP);
407
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_pre
      pMonthMax",precipMR);
408         precipMR.replaceRaster(minP);
409
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_pre
      pMonthMin",precipMR);
410         precipMR.replaceRaster(alpha);
411
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_pre
      pAlpha",precipMR);
412         precipMR.replaceRaster(rangeT);
413
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_tem
      ange",precipMR);
414         precipMR.replaceRaster(maxVals);
415
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_tem
      onthMax",precipMR);
416         precipMR.replaceRaster(minVals);
417
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_tem
      onthMin",precipMR);
418         precipMR = new Raster(folder+"/"+folder_dir+"_Annual_T0.asc");
419
      SimpleIO.rasterToAscii(folder+"/"+folder_dir+"_glimmer",folder_dir+"_tem
      eanAnn",precipMR);
420         return precipMR;
421      }
422
423      public void calculateMonthlyP(Raster topoRaster) throws Exception
424      {
425         double root1,root2,integral1,integral2;
426         int totaldays=0;
427         Raster precipMR;
428         for(int m=1;m<13;m++)
429         {
430            precipMR = new Raster(topoRaster,noData);
431            for (int i=0;i<nHor;i++)
432               for(int j=0;j<nVer;j++)
433                  if(ignore[i][j]==false)
434                  {
435                     root1 = totaldays*(Math.PI/182.5);
436                     root2 = (totaldays+days[m])*(Math.PI/182.5);
437                     integral2 =
      (ppt[i][j]*root2)-(range[i][j]*Math.cos(root2+alpha[i][j]));
```



```
438                     integral1 =
      (ppt[i][j]*root1)-(range[i][j]*Math.cos(root1+alpha[i][j]));
439   //we're working in monthly, not daily precip so convert the integral fro
      days to months.
440
      precipMR.changeValue(i,j,(float)((integral2-integral1)*(6d/Math.PI)));
441                  }
442            totaldays+=days[m];
443
      SimpleIO.rasterToAscii(folder,folder_dir+"_ppt_"+monthNames[m],precipMR)
444         }
445      }
446
447      //generates a 3D array with monthly precipitation values in it
448      public void calculateMonthlyP() throws Exception
449      {
450         double root1,root2,integral1,integral2;
451         int totaldays=0;
452         precipData = new float[12][nHor][nVer];
453         for(int m=1;m<13;m++)
454         {
455            for (int i=0;i<nHor;i++)
456               for(int j=0;j<nVer;j++)
457               {
458                  if(ignore[i][j]==false)
459                  {
460                     root1 = totaldays*(Math.PI/182.5);
461                     root2 = (totaldays+days[m])*(Math.PI/182.5);
462                     integral2 =
      (ppt[i][j]*root2)-(range[i][j]*Math.cos(root2+alpha[i][j]));
463                     integral1 =
      (ppt[i][j]*root1)-(range[i][j]*Math.cos(root1+alpha[i][j]));
464                     precipData[m-1][i][j] =
      (float)((integral2-integral1)*(6d/Math.PI));
465                  }
466                  else
467                  {
468                     precipData[m-1][i][j] = noData;
469                  }
470               }
471            totaldays+=days[m];
472         }
473      }
474
475      //series of loops using four variables to fit a more complex curve to
      the data
476      //importTempData must have been run in order to save the rasters
477      public void fitComplexSine(int kc) throws Exception
478      {
479         int i,j,l;
480         float k1,k2,k3,k4,residual,a,b,c,y;
481
482         //loop through the raster
483         for (j=0;j<nVer;j++)
484         {
485            System.out.println("data analysis on row "+(j+1)+" of "+nVer);
486            for (i=0;i<nHor;i++)
487            {
488               //set residual tally to be high initially
489               totalResid[i][j]=Float.MAX_VALUE;
490               //only evaluate cells with values other than 'nodata'
491               if(ignore[i][j]==false)
492               {
493   //             System.out.println("Amp&Median: "+tAmp[i][j]+"
      "+tMed[i][j]);
494                  /*loop through 100 possible values for amplitude and
495                   *360 possible values for alpha
496                   ***could use other function to improve and reduce
      processing!*/
497                  for(k1=0.13f;k1<0.92f;k1+=(0.026f*kc))
498                     for(k2=1.57f;k2<3.77f;k2+=(0.0625f*kc))
```

```
499                         for(k3=0.1f;k3<0.7f;k3+=(0.03f*kc))
500                            for(k4=0;k4<0.6;k4+=(0.05*kc))
501                         {
502                            //determine annual residuals for one combination of
    amp and alpha
503                            residual = residualCompCalc(k1,k2,k3,k4,i,j,false);
504
505                            //set equation values if the residual is smallest
506                            if(residual<totalResid[i][j])
507                            {
508                               totalResid[i][j] = residual;
509                               //gives the 'alpha' position in radians
510                               kOne[i][j] = k1;
511                               kTwo[i][j] = k2;
512                               kThree[i][j] = k3;
513                               kFour[i][j] = k4;

514                            }
515                         }//end amp and k loops and 'if ignore' statement
516                   }
517                   else
518                   {
519                      totalResid[i][j] = noData;
520                      kOne[i][j] = noData;
521                      kTwo[i][j] = noData;
522                      kThree[i][j] = noData;
523                      kFour[i][j] = noData;
524                   }
525                }//end nHor loop
526   //             System.out.println("K Vals - col "+j+": "+kOne[i][j]+"
    "+kTwo[i][j]+" "+kThree[i][j]+" "+kFour[i][j]+" ***** "+totalResid[i][j]
527             }//end nVer loop
528             float med,newResid;
529             //adjust median value to improve the equation fit
530             for (j=0;j<nVer;j++)
531                for (i=0;i<nHor;i++)
532                {
533                   tMed[i][j]-=0.05;
534                   med = tMed[i][j];
535                   residual =
    residualCompCalc(kOne[i][j],kTwo[i][j],kThree[i][j],kFour[i][j],i,j,true
536                   for(int m=-5;m<=30;m++)
537                   {
538                      tMed[i][j] += 0.01f;
539                      newResid =
    residualCompCalc(kOne[i][j],kTwo[i][j],kThree[i][j],kFour[i][j],i,j,true
540                      if(Math.abs(newResid)<Math.abs(residual))
541                      {
542                         totalResid[i][j] =
    residualCompCalc(kOne[i][j],kTwo[i][j],kThree[i][j],kFour[i][j],i,j,fals
543                         residual = newResid;
544                         med = tMed[i][j];
545                      }
546                   }
547                   tMed[i][j]=med;
548                }
549             //delete other files with coarser K resolution
550             for(i=kc+1;i<10;i++)
551                if(SimpleIO.checkFile(folder+"/"+folder_dir+"_kOne"+i+".asc"))
552                {
553                   System.out.println("delete entered");
554                   SimpleIO.deleteAscii(folder,folder_dir+"_kOne"+i);
555                   SimpleIO.deleteAscii(folder,folder_dir+"_kTwo"+i);
556                   SimpleIO.deleteAscii(folder,folder_dir+"_kThree"+i);
557                   SimpleIO.deleteAscii(folder,folder_dir+"_kFour"+i);
558                }
559          }//end method fitComplexSine
560
561          //calculation of reidual from a give amplitude, alpha and k values
562          public float residualCompCalc(float k1,float k2,float k3,float k4,int
    i,int j,boolean sumResids)
```



```
563          {
564             int l;
565             float a,b,c,y,residual;
566             residual=0;
567             for(l=1;l<13;l++)
568             {
569                //correction from month number to a value out of 2*pi
570                //ie january = pi/12; december = 23pi/12
571                //values are taken for the middle of the month
572                a = ((((float)l*2f)-1f)*(float)Math.PI)/12f;//the x value
573                c = k3*(float)Math.sin(a+k2);
574                b = (a-k1)+c;
575                y= tMed[i][j]-((tAmp[i][j]+k4)*(float)Math.cos(b));
576
577                if(sumResids==false)
578                   residual += Math.abs(y-data[l-1][i][j]);
579                else
580                   residual += y-data[l-1][i][j];
581             }
582             return residual;
583          }
584
585          public void findComplexSnowfall(float limitingT,float lim,float
    snowIceK,
586                float aspectAngle,float aspectInfluence,Raster topoRaster) thro
    Exception
587          {
588             this.aspectInfluence = aspectInfluence;
589             for (int j=0;j<nVer;j++)
590                for (int i=0;i<nHor;i++)
591                   if(ignore[i][j]==false)
592                   {
593
594                      aspectFactor =
    (float)topoRaster.getAspect(i,j,false)-aspectAngle;
595                      if(aspectFactor<-180)
596                         aspectFactor+=360;
597                      aspectFactor = (float)(Math.abs(aspectFactor)/90)-1;

598                      if(aspectFactor>1)
599                         aspectFactor=0;
600                      aspectFactor = aspectFactor/2;
601                      gradientFactor =
    (float)topoRaster.getGradient(i,j,false)/45;
602
    getSnowfallCplx(i,j,limitingT,lim,snowIceK,false,aspectFactor,gradientFa
    or);
603                   }
604          }
605
606          public void setAspectInfluence(float aspectInfluence)
607          {
608             this.aspectInfluence = aspectInfluence;
609          }
610
611          public double determineRoot(int monthNo,double root1,double
    root2,boolean maxMin)
612          {
613             double rootLim,resultRoot;
614             if(maxMin==true)
615                rootLim = (Math.PI/6)*monthNo;
616             else
617                rootLim = (Math.PI/6)*(monthNo-1);
618             if(root2>(2*Math.PI))
619
    if((maxMin&&(root1>=rootLim))||((maxMin==false)&&(root1>=rootLim+(Math.P
    6))))
620                   rootLim+=(2*Math.PI);
621             resultRoot = rootLim;
622             if(maxMin==true)
623             {
```

```
624            if((root2<rootLim)&&(root2>rootLim-(Math.PI/6)))
625               resultRoot = root2;
626         }
627         else
628         {
629            if((root1>rootLim)&&(root1<rootLim+(Math.PI/6)))
630               resultRoot = root1;
631         }
632         return resultRoot;
633      }
634      //limitingT is the temp threshold below which it snows
635      //lim defines the precision of the equation solution (roots 1 and 2)
636      //getSnowfallCplx can be used for totals (elaCheck=false) or snowfall
   calculation
637      public float getSnowfallCplx(int i,int j,float limitingT,float lim,fl
   snowIceK,boolean elaCheck,float aspectFactor,float gradientFactor) throw
   Exception
638      {
639         float snowfallf=0;
640         double
   derivRoot1,principalAngle,cosA,angle2,root1,root2,testRoot,integral1,int
   ral2;
641
642         if(ignore[i][j]==false)
643         {
644            cosA = (tMed[i][j]-limitingT)/(tAmp[i][j]+kFour[i][j]);
645            if(cosA <= -1) //ie snow all year round
646            {
647               root1=0;
648               root2=2*Math.PI;
649               integral2 =
   (ppt[i][j]*root2)-(range[i][j]*Math.cos(root2+alpha[i][j]));
650               integral1 =
   (ppt[i][j]*root1)-(range[i][j]*Math.cos(root1+alpha[i][j]));
651               //store in 'snowfall' raster, and correct back from radians
   month numbers
652               snowfallf = (float)((integral2-integral1)*(6d/Math.PI));
653   //          if(elaCheck==false)
654   //             System.out.println(i+" "+j+" ALL SNOW "+snowfallf);
655            }
656            else if(cosA >= 1) //ie no snow at all
657            {
658               snowfallf = 0f;
659               root1=noData;
660               root2=noData;
661   //          if(elaCheck==false)
662   //             System.out.println(i+" "+j+"NO SNOW ");
663            }
664   //a negative cosA indicates roots lying in the summer part of the curve
665            else //other cases, with a spell of no snow in the summer
666            {
667               principalAngle = Math.acos(cosA);
668               angle2 = (2*Math.PI) - principalAngle;
669               root1 = findXVal(i,j,(float)principalAngle,lim);
670               root2 = findXVal(i,j,(float)angle2,lim);
671               derivRoot1 = complexDerivative(i,j,root1);
672               if(root1<0)
673                  root1+=(2*Math.PI);
674               if(root2<0)
675                  root2+=(2*Math.PI);
676               if((derivRoot1>0)&&(root1<root2))
677               {
678                  root1+=(2*Math.PI);
679                  testRoot = root2;
680                  root2 = root1;
681                  root1 = testRoot;
682               }
683               else if((derivRoot1<0)&&(root1>root2))
684               {
685                  root2+=(2*Math.PI);
686               }
```



```
687               integral2 =
   (ppt[i][j]*root2)-(range[i][j]*Math.cos(root2+alpha[i][j]));
688               integral1 =
   (ppt[i][j]*root1)-(range[i][j]*Math.cos(root1+alpha[i][j]));
689               snowfallf = (float)((integral2-integral1)*(6d/Math.PI));
690            }
691      //determine aspect (more snow downwind, less upwind)
692      /*val is multiplied by 1 + aspectFactor (-0.5...0.5) * gradient
   (0...2)*/
693            snowfallf =
   snowfallf*(1-(aspectFactor*gradientFactor*aspectInfluence));
694            if(snowfallf<0)
695               snowfallf=0;
696
697            if(elaCheck==false)
698            {
699               rt1[i][j] = (float)root1;
700               rt2[i][j] = (float)root2;
701               snowfall[i][j] = snowfallf;
702   //          if(i==0)
703   //          System.out.println("Snowfall - col "+j+": ***"+snowfall[i][j]+"
   rt1:"+rt1[i][j]+" rt2:"+rt2[i][j]);
704            }
705         }//end if ignore..
706         return snowfallf;
707      }
708
709      public double complexDerivative(int i,int j,double x)
710      {
711         double fOfX = (x - kOne[i][j]) +
   (kThree[i][j]*Math.sin(x+kTwo[i][j]));
712         double dydX = (tAmp[i][j]+kFour[i][j])*Math.sin(fOfX);
713         double dXdx = 1 + (kThree[i][j]*Math.cos(x+kTwo[i][j]));
714         return dydX*dXdx;
715      }
716
717      //used to empirically solve X in the complex k equation for the roots
718      //to a resolution defined by lim.
719      //X = PAngle + k1 - k3sin(X + k2)
720      public float findXVal(int i,int j,float pAngle,float lim) throws
   Exception
721      {
722         float lowX,highX,testX,lowXcalc,highXcalc,testXcalc;
723         // the root x lies within a range defined by (principal angle + k1
   +/- k3
724         lowX = pAngle + kOne[i][j] - kThree[i][j];
725         highX = pAngle + kOne[i][j] + kThree[i][j];
726         //this is used to calculate a value to be compared to cosA
727         //(the principal angle pAngle)
728         lowXcalc =
   (lowX-kOne[i][j])+(kThree[i][j]*(float)Math.sin(lowX+kTwo[i][j]));
729         highXcalc =
   (highX-kOne[i][j])+(kThree[i][j]*(float)Math.sin(highX+kTwo[i][j]));
730
731         //define the limits of the equation
732
   while((Math.abs(highXcalc-pAngle)>lim)&&(Math.abs(lowXcalc-pAngle)>lim))
733         {
734            testX = (lowX+highX)/2;
735            testXcalc =
   (testX-kOne[i][j])+(kThree[i][j]*(float)Math.sin(testX+kTwo[i][j]));
736            if(testXcalc>pAngle)
737               highX = testX;
738            else
739               lowX = testX;
740            lowXcalc =
   (lowX-kOne[i][j])+(kThree[i][j]*(float)Math.sin(lowX+kTwo[i][j]));
741            highXcalc =
   (highX-kOne[i][j])+(kThree[i][j]*(float)Math.sin(highX+kTwo[i][j]));
742         }
743         if (Math.abs(highXcalc-pAngle)<=lim)
```

```
744             return highX;
745          else
746             return lowX;
747       }
748
749       //limitingT is the temp threshold above which the vegetation grows
750       //lim defines the precision of the equation solution (roots 1 and 2)
751       public float getGrowingSeason(int i,int j,float limitingT,float lim)
      throws Exception
752       {
753          float growS=0;
754          double
      derivRoot1,principalAngle,cosA,angle2,root1,root2,testRoot,integral1,int
      ral2;
755
756          if(ignore[i][j]==false)
757          {
758             cosA = (tMed[i][j]-limitingT)/(tAmp[i][j]+kFour[i][j]);
759             if(cosA <= -1) //ie no growing season at all
760             {
761                growS = 0f;
762                root1=noData;
763                root2=noData;
764    //          System.out.println(i+" "+j+" NO GROWING SEASON ");
765             }
766             else if(cosA >= 1) //ie growing season all year round
767             {
768                growS = 365f;
769                root1=0;
770                root2=2*Math.PI;
771    //          System.out.println(i+" "+j+" ALL YEAR GROWING SEASON "+growS
772             }
773    /*other cases, with some spells of no growth in the winter - negative
      values of
774     *cosA imply a shorter growing season*/
775             else
776             {
777                principalAngle = Math.acos(cosA);
778                angle2 = (2*Math.PI) - principalAngle;
779                //spring
780                root1 = findXVal(i,j,(float)principalAngle,lim);
781                //autumn
782                root2 = findXVal(i,j,(float)angle2,lim);
783
784                derivRoot1 = complexDerivative(i,j,root1);
785                if(root1<0)
786                   root1+=(2*Math.PI);
787                if(root2<0)
788                   root2+=(2*Math.PI);
789                if((derivRoot1<0)&&(root2>root1))
790                {
791                   root1+=(2*Math.PI);
792                   testRoot = root2;
793                   root2 = root1;
794                   root1 = testRoot;
795                }
796                else if((derivRoot1>0)&&(root1>root2))
797                {
798                   root2+=(2*Math.PI);
799                }
800                //convert radians into days (not degrees)
801                growS = (float)((root2-root1)*(182.5/Math.PI));
802                root1 = root1*(182.5/Math.PI);
803                if(root2>2*Math.PI)
804                   root2 = (root2-(2*Math.PI))*(182.5/Math.PI);
805                else
806                   root2 = root2*(182.5/Math.PI);
807             }
808             //growing season dates root1 = start date; root2 = end date
809             rt1GS[i][j] = (float)root1;
810             rt2GS[i][j] = (float)root2;
```



```
811             growSeason[i][j] = (float)growS;
812          }//end if ignore..
813          return growS;
814       }
815
816       public void findGrowingSeason(float limitingT,float lim) throws
      Exception
817       {
818          float growS;
819          for (int j=0;j<nVer;j++)
820             for (int i=0;i<nHor;i++)
821                if(ignore[i][j]==false)
822                {
823                   growS = getGrowingSeason(i,j,limitingT,lim);
824                   if(growS>0)
825                      meanGSt[i][j] = growingSeasonMeanT(i,j);
826                }
827       }
828
829       public float growingSeasonMeanT(int i,int j)
830       {
831          float totalT=0;
832          float temp;
833          int days=0;
834          float startRads = (float)(rt1GS[i][j]*(Math.PI/182.5));
835          float endRads = (float)(rt2GS[i][j]*(Math.PI/182.5));
836          for(float d=startRads;d<endRads;d+=(float)(Math.PI/182.5))
837          {
838             days++;
839             temp =
      tMed[i][j]-(float)((tAmp[i][j]+kFour[i][j])*Math.cos(d-kOne[i][j]+(kThre
      i][j]*Math.sin(d+kTwo[i][j])));
840             totalT+=temp;
841          }
842          return (totalT/days);
843       }
844
845       public boolean getIgnoreVal(int i,int j)
846       {
847          return ignore[i][j];
848       }
849
850       public int setTempKFactors(int maxK) throws Exception
851       {
852          int i;
853          int cNum=10;
854          for(i=1;i<10;i++)
855             if((SimpleIO.checkFile(folder+"/"+folder_dir+"_kOne"+i+".asc"))
856                &&(i<=maxK))
857             {
858                cNum=i;
859                System.out.println("importing K values with a coarsening fac
      of "+i+".");
860                i=10;
861             }
862          meanTR = new Raster(folder+"/"+folder_dir+"_kOne"+cNum+".asc");
863          kOne = meanTR.getRaster();
864          meanTR = new Raster(folder+"/"+folder_dir+"_kTwo"+cNum+".asc");
865          kTwo = meanTR.getRaster();
866          meanTR = new Raster(folder+"/"+folder_dir+"_kThree"+cNum+".asc");
867          kThree = meanTR.getRaster();
868          meanTR = new Raster(folder+"/"+folder_dir+"_kFour"+cNum+".asc");
869          kFour = meanTR.getRaster();
870          return cNum;
871       }
872
873       public void writeKasciis(Raster topoRaster,float kc) throws Exception
874       {
875          //saves memory by reusing the same Raster while writing files
876          Raster pptR = new Raster(topoRaster,kOne);
877          SimpleIO.rasterToAscii(folder,folder_dir+"_kOne"+(int)kc,pptR);
```

```
878          pptR.replaceRaster(kTwo);
879          SimpleIO.rasterToAscii(folder,folder_dir+"_kTwo"+(int)kc,pptR);
880          pptR.replaceRaster(kThree);
881          SimpleIO.rasterToAscii(folder,folder_dir+"_kThree"+(int)kc,pptR)
882          pptR.replaceRaster(kFour);
883          SimpleIO.rasterToAscii(folder,folder_dir+"_kFour"+(int)kc,pptR);
884       }
885
886       public void writeSnowAsciis(Raster topoRaster) throws Exception
887       {
888          //saves memory by reusing the same Raster while writing files
889          Raster pptR = new Raster(topoRaster,totalResid);
890          SimpleIO.rasterToAscii(folder,folder_dir+"_temp_residuals",pptR)
891          pptR.replaceRaster(rt1);
892          SimpleIO.rasterToAscii(folder,folder_dir+"_root1",pptR);
893          pptR.replaceRaster(rt2);
894          SimpleIO.rasterToAscii(folder,folder_dir+"_root2",pptR);
895          pptR.replaceRaster(ignore);
896          SimpleIO.rasterToAscii(folder,folder_dir+"_ignore_vals",pptR);
897          pptR.replaceRaster(snowfall);
898          SimpleIO.rasterToAscii(folder,folder_dir+"_snowfall",pptR);
899       }
900
901       public void writeGSAsciis(Raster topoRaster) throws Exception
902       {
903          //saves memory by reusing the same Raster while writing files
904          Raster pptR = new Raster(topoRaster,totalResid);
905          SimpleIO.rasterToAscii(folder,folder_dir+"_temp_residuals",pptR)
906          pptR.replaceRaster(rt1GS);
907          SimpleIO.rasterToAscii(folder,folder_dir+"_GS_start",pptR);
908          pptR.replaceRaster(rt2GS);
909          SimpleIO.rasterToAscii(folder,folder_dir+"_GS_end",pptR);
910          pptR.replaceRaster(ignore);
911          SimpleIO.rasterToAscii(folder,folder_dir+"_ignore_vals",pptR);
912          pptR.replaceRaster(growSeason);
913          SimpleIO.rasterToAscii(folder,folder_dir+"_GS_length",pptR);
914          pptR.replaceRaster(meanGSt);
915          SimpleIO.rasterToAscii(folder,folder_dir+"_GS_temp",pptR);
916       }
917
918          //limitingT is the temp threshold below which it snows
919       //lim defines the precision of the equation solution (roots 1 and 2
920       //getSnowfallCplx can be used for totals (elaCheck=false) or snowfa
     calculation
921       public float getSnowfallCplxSummer(int i,int j,float limitingT,floa
     lim,float snowIceK,boolean elaCheck,float aspectFactor,float
     gradientFactor) throws Exception
922       {
923          float snowfallf=0;
924          double
     derivRoot1,principalAngle,cosA,angle2,root1,root2,testRoot,integral1,i
     ral2;
925
926          if(ignore[i][j]==false)
927          {
928             cosA = (tMed[i][j]-limitingT)/(tAmp[i][j]+kFour[i][j]);
929             if(cosA <= -1) //ie snow all year round
930             {
931                root1=Math.PI*0.666f;
932                root2=1.5f*Math.PI;
933                integral2 =
     (ppt[i][j]*root2)-(range[i][j]*Math.cos(root2+alpha[i][j]));
934                integral1 =
     (ppt[i][j]*root1)-(range[i][j]*Math.cos(root1+alpha[i][j]));
935                //store in 'snowfall' raster, and correct back from radian
     month numbers
936                snowfallf = (float)((integral2-integral1)*(6d/Math.PI));
937    //          if(elaCheck==false)
938    //             System.out.println(i+" "+j+" ALL SNOW "+snowfallf);
939             }
940             else if(cosA >= 1) //ie no snow at all
```



```
941          {
942             snowfallf = 0f;
943             root1=noData;
944             root2=noData;
945    //       if(elaCheck==false)
946    //          System.out.println(i+" "+j+"NO SNOW ");
947          }
948    //a negative cosA indicates roots lying in the summer part of the curv
949          else //other cases, with a spell of no snow in the summer
950          {
951             principalAngle = Math.acos(cosA);
952             angle2 = (2*Math.PI) - principalAngle;
953             System.out.println(i+" "+j+" "+principalAngle+" "+lim);
954             root1 = findXVal(i,j,(float)principalAngle,lim);
955             System.out.println(i+" "+j+" "+principalAngle+" "+lim);
956             root2 = findXVal(i,j,(float)angle2,lim);
957             System.out.println(i+" "+j+" "+principalAngle+" "+lim);
958             derivRoot1 = complexDerivative(i,j,root1);
959             if(root1<0)
960                root1+=(2*Math.PI);
961             if(root2<0)
962                root2+=(2*Math.PI);
963             if((derivRoot1>0)&&(root1<root2))
964             {
965                root1+=(2*Math.PI);
966                testRoot = root2;
967                root2 = root1;
968                root1 = testRoot;
969             }
970             else if((derivRoot1<0)&&(root1>root2))
971             {
972                root2+=(2*Math.PI);
973             }
974             if((root1>(Math.PI*0.666f))&&(root1<2*Math.PI))
975             {
976                testRoot = (Math.PI*0.666f);
977                integral2 =
     (ppt[i][j]*root2)-(range[i][j]*Math.cos(root2+alpha[i][j]));
978                integral1 =
     (ppt[i][j]*root1)-(range[i][j]*Math.cos(root1+alpha[i][j]));
979                snowfallf = (float)((integral2-integral1)*(6d/Math.PI))
980             }
981          }
982       //determine aspect (more snow downwind, less upwind)
983       /*val is multiplied by 1 + aspectFactor (-0.5...0.5) * gradient
     (0...2)*/
984          snowfallf =
     snowfallf*(1-(aspectFactor*gradientFactor*aspectInfluence));
985          if(snowfallf<0)
986             snowfallf=0;
987
988          if(elaCheck==false)
989          {
990             rt1[i][j] = (float)root1;
991             rt2[i][j] = (float)root2;
992             snowfall[i][j] = snowfallf;
993    //       if(i==0)
994    //       System.out.println("Snowfall - col "+j+": ***"+snowfall[i][j]
     rt1:"+rt1[i][j]+" rt2:"+rt2[i][j]);
995          }
996       }//end if ignore..
997       return snowfallf;
998    }
999
1000
1001
1002
1003    public void setFolder(String f)
1004    {
1005       this.folder = f;
1006    }
```

```
1007
1008        public void setFilePrefix(String fp)
1009        {
1010           this.folder_dir = fp;
1011        }
1012
1013        public float getTValue(int m,int x,int y)
1014        {
1015           return this.data[m][x][y];
1016        }
1017
1018        public float getPValue(int m,int x,int y)
1019        {
1020           return this.precipData[m][x][y];
1021        }
1022
1023        public float getK1Value(int x,int y)
1024        {
1025           return this.kOne[x][y];
1026        }
1027
1028        public float getK2Value(int x,int y)
1029        {
1030           return this.kTwo[x][y];
1031        }
1032
1033        public float getK3Value(int x,int y)
1034        {
1035        return this.kThree[x][y];
1036        }
1037
1038        public float getK4Value(int x,int y)
1039        {
1040           return this.kFour[x][y];
1041        }
1042
1043        public float getTAmpValue(int x,int y)
1044        {
1045           return this.tAmp[x][y];
1046        }
1047
1048        public float getTMedValue(int x,int y)
1049        {
1050           return this.tMed[x][y];
1051        }
1052
1053        //empty constructor, for use in testing classes
1054        public PrecipModel(String folder,String folder_dir) throws Exceptio
1055        {
1056           monthNames = MonthsAndDays.getMonths();
1057           this.folder = folder;
1058           this.folder_dir = folder_dir;
1059        }
1060
1061     /* public void fitSine(Raster topoRaster) throws Exception
1062        {
1063           int i,j,l;
1064           float amp,k,residual;
1065           float[][] totalResid = new float[nHor][nVer];
1066           float[][] bestPos = new float[nHor][nVer];
1067           float[][] bestAmp = new float[nHor][nVer];
1068
1069           //loop through the raster
1070           for (j=0;j<nVer;j++)
1071           {
1072              System.out.println("data analysis on row "+j+" of "+nVer);
1073              for (i=0;i<nHor;i++)
1074              {
1075                 //set residual tally to be high initially
1076                 totalResid[i][j]=Float.MAX_VALUE;
1077
```



```
1078              //only evaluate cells with values other than 'nodata'
1079              if(ignore[i][j]==false)
1080              {
1081                 meanT[i][j]=0;
1082                 for(l=1;l<13;l++)
1083                    meanT[i][j]+=data[l-1][i][j];
1084                 meanT[i][j] = meanT[i][j]/12;
1085                 //*loop through 100 possible values for amplitude and
1086                 // *360 possible values for alpha
1087                 //***could use other function to improve and reduce
     processing
1088                 for(k=0;k<360;k++)
1089                    for(amp=0;amp<(tAmp[i][j]*2);amp+=(tAmp[i][j]/50))
1090                    {
1091                       //determine annual residuals for one combination
     amp and alpha
1092                       residual = residualCalc(amp,k,i,j);
1093                       //set equation values if the residual is smallest
1094                       if(residual<totalResid[i][j])
1095                       {
1096                          totalResid[i][j] = (float)residual;
1097                          //gives the 'alpha' position in radians
1098                          bestPos[i][j] = (float)(Math.PI/180)*((float)k
1099                          bestAmp[i][j] = amp;
1100                       }
1101                    }//end amp and k loops and 'if ignore' statement
1102              }
1103           }//end nHor loop
1104        }//end nVer loop
1105        Raster pptR = new Raster(topoRaster,totalResid);
1106        SimpleIO.rasterToAscii(folder,folder_dir+"_sineresids",pptR);
1107        pptR.replaceRaster(bestAmp);
1108        SimpleIO.rasterToAscii(folder,folder_dir+"_sineamp",pptR);
1109        pptR.replaceRaster(bestPos);
1110        SimpleIO.rasterToAscii(folder,folder_dir+"_sinepos",pptR);
1111     }//end method fitSine
1112
1113     public float residualCalc(float amp,float k,int i,int j)
1114     {
1115        int l;
1116        float a,y,residual;
1117        residual=0;
1118        for(l=1;l<13;l++)
1119        {
1120           //correction from month number to a value out of 2*pi
1121           //ie january = pi/12; december = 23pi/12
1122           //values are taken for the middle of the month
1123           a = ((((float)l*2f)-1f)*(float)Math.PI)/12f;
1124           a += (Math.PI/180f)*k;
1125           if(a >= 2f*Math.PI)
1126              a -= 2f*Math.PI;
1127           y= (amp*(float)Math.sin(a))+meanT[i][j];
1128
1129           residual += Math.abs(y-data[l-1][i][j]);
1130        }
1131
1132        return residual;
1133     }
1134  */
1135     public static void main(String args[]) throws Exception
1136     {
1137        PrecipModel pm = new PrecipModel();
1138        int monthNo = 1;
1139        double root1 = 2;
1140        double root2 = 5;
1141        boolean maxMin = false;
1142        double t = 1000;
1143        double snowAdjust  =
     (float)((0.0000004*(t*t*t))-(0.0009*(t*t))+(0.4708*t)-74.698);
1144
1145        double result = pm.determineRoot(monthNo,root1,root2,maxMin);
```

```
1146            System.out.println(snowAdjust);
1147    //      PrecipModel pm = new
        PrecipModel(true,"temp_avg","temp_summer_amp","temp_summer_alpha","tem
        nter_amp","temp_winter_alpha");
1148    //      PrecipModel pm = new PrecipModel("C:/gisdata/","testfolder");
1149    //      Raster testR = new Raster("C:/gisdata/icedem_cp.asc");
1150    //      pm.importPrecipData(testR,0,false);
1151    //      pm.fitComplexSine();
1152    //      pm.setTempKFactors(8);
1153    //      pm.findComplexSnowfall(1,0.01f,0.007f);
1154        }
1155
1156    }
```

```
1   /*class to calculate ELA, based on given temperature and precipitation data
2    *last updated on 7th April 2005*/
3   package afc.spatial;
4
5   import java.util.*;
6   import java.math.*;
7   import afc.utilities.*;
8
9
10  public class ELAFinderComplex
11  {
12     private Raster meanTR,demR,lapseRsqR,lapseRateR,elaMapR;
13     private float[][] elaOffset,elaMap,lapseRSq,lapseRate,screwup;
14     private int nHor,nVer,noData,aspectAngle,ssr;
15     private float aspectInfluence,ssp,ssf;
16     private PddSum pd;
17     private PrecipModel pm;
18     private ClimateControl cmc;
19     private String folder,filePrefix,root_dir;
20     private boolean testing;
21
22     public ELAFinderComplex(ClimateControl cmc,PrecipModel pmod,PddSum pdds)
   throws Exception
23     {
24        this.pm=pmod;
25        this.pd=pdds;
26        this.cmc = cmc;
27        this.folder = cmc.getFolder();
28        this.filePrefix = cmc.getSubDir();
29        this.root_dir = cmc.getRootDir();
30        testing=false;
31     }
32
33     //constructor for testing
34     public ELAFinderComplex(ClimateControl cmc,float snowLim) throws Exceptio
35     {
36        testing=true;
37        pd = new PddSum();
38        pm = new PrecipModel("C:/gisdata","");
39        folder = "C:/gisdata";
40        filePrefix = "testfolder";
41        configure(0,0,0,0,0,0,0);
42        float lim=0.0001f;
43        float k = 0.007f;
44  //    calcELAoffset(snowLim,lim,k,0,0);
45  //    calculateELA(15);
46        writeRasters();
47     }
48
49     public void configure(int nHor,int nVer,int noData,int aspectAngle,float
   aspectInfluence,float ssf,int ssr) throws Exception
50     {
51        this.nHor = nHor;
52        this.nVer = nVer;
53        this.noData = noData;
54        this.ssf = ssf;
55        this.ssp = 1-(ssf/2);
56        this.ssr=ssr;
57        this.aspectInfluence = aspectInfluence;
58        this.aspectAngle = aspectAngle;
59        if(testing==true)
60        {
61           meanTR =  cmc.getTopoRaster();
62           this.nHor = meanTR.getXcells();
63           this.nVer = meanTR.getYcells();
64           this.noData = meanTR.getNoData();
65  //       pm.importTempData();
66           pm.importPrecipData(meanTR,0,false);
67        }
68        elaOffset = new float[nHor][nVer];
69        elaMap = new float[nHor][nVer];
```



```
70        lapseRSq = new float[nHor][nVer];
71        lapseRate = new float[nHor][nVer];
72     }
73
74     //calculates the ELA offset in each cell (deg C) by matching (iterative
75     //the total ablation from the DDM (PddSum) and accumulation (PrecipMode
76     //by varying the temperature input to each.
77     //elaOffset is an array of these temperature offset values
78     public void calcELAoffset(float limitingT,float lim,float snowIceK,Rast
   topoRaster) throws Exception
79     {
80        System.out.println("calcELAoffset limT: "+limitingT);
81        int i,j,m;
82        int screwupCount=0;
83        screwup = new float[nHor][nVer];
84        float tOffset=0;
85        boolean offsetSign=true;
86        boolean switched=false;
87        boolean firstSwitch=false;
88        float totalPdd,snowValue,tOffsetFactor,aspectFactor,gradientFactor;
89        pm.setAspectInfluence(aspectInfluence);
90        Raster sunShedYear = new
   Raster(folder+"/"+filePrefix+"_vs_ann_"+ssr+".asc");
91        if(testing==true)
92           pm.setTempKFactors(8);
93
94        System.out.println(". ssp: "+ssp+". ssf: "+ssf+". ssr: "+ssr);
95
96        for (i=0;i<nHor;i++)
97        {
98           for (j=0;j<nVer;j++)
99           {
100             if(pm.getIgnoreVal(i,j)==false)
101             {
102                aspectFactor =
   (float)topoRaster.getAspect(i,j,false)-aspectAngle;
103                if(aspectFactor<-180)
104                   aspectFactor+=360;
105                aspectFactor = (float)(Math.abs(aspectFactor)/90)-1;

106                if(aspectFactor>1)
107                   aspectFactor=0;
108                aspectFactor = aspectFactor/2;
109                gradientFactor = (float)topoRaster.getGradient(i,j,false)/4

110
111                totalPdd = 0;
112                tOffset=0;
113                for(m=1;m<13;m++)
114
   totalPdd+=pd.pddCalc(m,pm.getTValue(m-1,i,j)+tOffset,snowIceK,noData,true)
115                totalPdd = totalPdd*(ssp+(sunShedYear.getValue(i,j)*ssf));
116                if(totalPdd<0)
117                   totalPdd=0;
118                snowValue =
   pm.getSnowfallCplx(i,j,limitingT-tOffset,lim,snowIceK,true,aspectFactor,gr
   ientFactor);
119             //Geometric loop to minimise the difference between snowfall and ablati
120             //using tOffset.  Has a problem - Occasionally doesn't resolve itself
121                tOffsetFactor=4;
122                int tries=0;
123                while
   ((Math.abs(snowValue-totalPdd)>10)&&(Math.abs(tOffsetFactor)>0.0001)&&(tri
   <500))
124                {
125                   tries++;
126                   switched=false;
127                   if(snowValue>=totalPdd)
128                   {
129                      tOffsetFactor = Math.abs(tOffsetFactor);
130                      if(offsetSign==false)
```

```
131                         {
132                            switched=true;
133                            firstSwitch=true;
134                         }
135                         offsetSign=true;
136                      }
137                      if(snowValue<totalPdd)
138                      {
139                         tOffsetFactor = -(Math.abs(tOffsetFactor));
140                         if(offsetSign==true)
141                         {
142                            switched=true;
143                            firstSwitch=true;
144                         }
145                         offsetSign=false;
146                      }
147                      if(switched==true)
148                         tOffsetFactor= tOffsetFactor*0.5f;
149                      tOffset+=tOffsetFactor;
150                      totalPdd = 0;
151                      for(m=1;m<13;m++)
152
    totalPdd+=pd.pddCalc(m,pm.getTValue(m-1,i,j)+tOffset,snowIceK,noData,true)
153                      totalPdd = totalPdd*(ssp+(sunShedYear.getValue(i,j)*ssf)
154                      if(totalPdd<0)
155                         totalPdd=0;
156   //       System.out.println(i+" "+j+" snow?");
157                      snowValue =
    pm.getSnowfallCplx(i,j,limitingT-tOffset,lim,snowIceK,true,aspectFactor,gr
    ientFactor);
158   //          System.out.println(i+" "+j+" snow:"+snowValue+" pdd:"+totalPdd
    offset:"+tOffset+" offsetF:"+tOffsetFactor+" sign:"+offsetSign+"
    swtch:"+switched+" ** "+screwupCount);
159   //          SimpleIO.readKeyboard();
160                   }
161                   elaOffset[i][j] = tOffset;
162                   if(Math.abs(tOffsetFactor)<0.0001)
163                   {
164                      screwup[i][j] = 1;
165                      screwupCount+=1;
166                      System.out.println("********** SCREWUP AT: "+i+" "+j);
167                   }
168                }
169                else if (pm.getIgnoreVal(i,j)==true)
170                {
171                   elaOffset[i][j] = noData;
172                   screwup[i][j] = noData;
173                }
174   //          System.out.println("ELA Offset Row: "+j);
175             }
176             System.out.println("ELA Offset Col: "+i);
177          }
178          System.out.println("********* "+screwupCount+ " screwed up
    calculations! *********");
179       }
180
181       public Raster writeELAOffsetAsciis() throws Exception
182       {
183          //saves memory by reusing the same Raster while writing files
184          Raster pptR = new Raster(cmc.getTopoRaster(),screwup);
185          SimpleIO.rasterToIntAscii(folder,filePrefix+"_screwup",pptR);
186          pptR.replaceRaster(elaOffset);
187          SimpleIO.rasterToAscii(folder,filePrefix+"_ela_offset",pptR);
188          return pptR;
189       }
190
191       //creates a raster of ELA elevation, based on using a DEM and average
192       //temperature to calculate a 'local' (based on focal radius size in
193       //cells of the DEM) temperature lapse rate, which is also calculated.
194       //This is used with the ELA offset value in deg C from the calcELAoffse
195       //method to give the ELA height
```



```
196       public Raster calculateELA(int fr,boolean hiLoFlag,Rectangle icelandFP)
    throws Exception
197       {
198          Raster elaOffR,elaMapR,lapseRsqR,lapseRateR,lapseRateInterceptR;
199          elaOffR = new Raster(folder+"/"+filePrefix+"_ela_offset.asc");
200          double cellSize = elaOffR.getCellSize();
201          Rectangle selectFootprint = elaOffR.getFootprint();
202          Point selectLL = selectFootprint.getLL();
203          Point selectUR = selectFootprint.getUR();
204          selectLL.translate((-fr*cellSize),(-fr*cellSize));
205          selectUR.translate((fr*cellSize),(fr*cellSize));
206          Rectangle selectFP = new Rectangle(selectLL,selectUR);
207          selectFP = selectFP.minEnclRect(icelandFP);
208          double yll = selectFP.getYll();
209          double yur = selectFP.getYur();
210          if(hiLoFlag==true)
211             demR = new
    Raster(root_dir+"input_files/icedem_250.asc",yll,yur,"none");
212          else
213             demR = new
    Raster(root_dir+"input_files/icedem_cp.asc",yll,yur,"none");
214          demR=demR.gridclip(selectFP);
215          if(hiLoFlag==true)
216             meanTR = new
    Raster(root_dir+"input_files/ann_m100.asc",yll,yur,"N");
217          else
218             meanTR = new
    Raster(root_dir+"input_files/t_ann.asc",yll,yur,"none");
219          meanTR=meanTR.gridclip(selectFP);
220
221          float[] demSample,avgTsample;
222          float height;
223          Regression reg;
224
225          //get the minimum enclosing rectangle of a selected area and the DEM
226          //using the select footprint expanded by the focal radius amount
227          //ensuring the area doesn't cross the DEM border
228          elaMapR = new Raster(elaOffR,elaOffR.getNoData());
229          lapseRsqR = new Raster(elaOffR,elaOffR.getNoData());
230          lapseRateR = new Raster(elaOffR,elaOffR.getNoData());
231          lapseRateInterceptR = new Raster(elaOffR,elaOffR.getNoData());
232          int iStart =
    (int)((selectFootprint.getXll()-selectFP.getXll())/demR.getCellSize());
233          int jStart =
    (int)((selectFootprint.getYll()-selectFP.getYll())/demR.getCellSize());
234          for(int i=iStart;i<iStart+elaOffR.getXcells();i++)
235             for(int j=jStart;j<jStart+elaOffR.getYcells();j++)
236                if (elaOffR.getValue(i-iStart,j-jStart)!=elaOffR.getNoData())
237                {
238                   demSample = demR.boxValues(i,j,fr,0);
239                   avgTsample = meanTR.boxValues(i,j,fr,noData);
240                   reg=new Regression(avgTsample,demSample,noData);
241                   height =
    reg.getIntercept()+(reg.getGradient()*(meanTR.getValue(i,j)+elaOffR.getVal
    (i-iStart,j-jStart)));
242                   elaMapR.changeValue(i-iStart,j-jStart,height);
243   //                System.out.println("ela: "+elaMap[i-iStart][j-jStart]);
244                   lapseRsqR.changeValue(i-iStart,j-jStart,reg.getRSquared());
245
    lapseRateR.changeValue(i-iStart,j-jStart,1f/reg.getGradient());
246
    lapseRateInterceptR.changeValue(i-iStart,j-jStart,reg.getIntercept());
247                }
248          SimpleIO.rasterToAscii(folder,filePrefix+"_lapse_rsq",lapseRsqR);
249          SimpleIO.rasterToAscii(folder,filePrefix+"_lapse_rate",lapseRateR);
250
    SimpleIO.rasterToAscii(folder,filePrefix+"_lapse_intercept",lapseRateInter
    ptR);
251          SimpleIO.rasterToAscii(folder,filePrefix+"_ela_map",elaMapR);
252          return elaMapR;
253       }
```

```
254
255     public Raster calculateELAexistingLapse() throws Exception
256     {
257        float height;
258        Raster lapseRateR = new
Raster(folder+"/"+filePrefix+"_lapse_rate.asc");
259        Raster elaOffR = new Raster(folder+"/"+filePrefix+"_ela_offset.asc")
260        Raster lapseRateInterceptR = new
Raster(folder+"/"+filePrefix+"_lapse_intercept.asc");
261        meanTR = new Raster(folder+"/"+filePrefix+"_t_ann.asc");
262        for(int i=0;i<elaOffR.getXcells();i++)
263           for(int j=0;j<elaOffR.getYcells();j++)
264              if
((elaOffR.getValue(i,j)!=elaOffR.getNoData())&&(lapseRateR.getValue(i,j)!=
265              {
266                 height =
lapseRateInterceptR.getValue(i,j)+(1/(lapseRateR.getValue(i,j))*(meanTR.ge
alue(i,j)+elaOffR.getValue(i,j)));
267                 elaMapR.changeValue(i,j,height);
268              }
269        SimpleIO.rasterToAscii(folder,filePrefix+"_ela_map",elaMapR);
270        return elaMapR;
271     }
272     //version of calculateELA designed as a standalone just to calculate an
273     //return the lapse rate for a given focal radius, fr
274     public Raster calculateLapseRate(int fr,Raster topoR,boolean
hiLoFlag,Rectangle icelandFP,boolean tOrP) throws Exception
275     {
276        Raster demR,meanTR,lapseRsqR,lapseRateR,lapseRateInterceptR;
277        float[] demSample,avgTsample;
278        float coeffB;
279        Regression reg;
280        noData = topoR.getNoData();
281        double cellSize = topoR.getCellSize();
282        lapseRate = new float[nHor][nVer];
283
284        Rectangle selectFootprint = topoR.getFootprint();
285        Point selectLL = selectFootprint.getLL();
286        Point selectUR = selectFootprint.getUR();
287        selectLL.translate((-fr*cellSize),(-fr*cellSize));
288        selectUR.translate((fr*cellSize),(fr*cellSize));
289        Rectangle selectFP = new Rectangle(selectLL,selectUR);
290        selectFP = selectFP.minEnclRect(icelandFP);
291
292        double yll = selectFP.getYll();
293        double yur = selectFP.getYur();
294        if(hiLoFlag==true)
295           demR = new
Raster(root_dir+"input_files/icedem_250.asc",yll,yur,"none");
296        else
297           demR = new
Raster(root_dir+"input_files/icedem_cp.asc",yll,yur,"none");
298        demR=demR.gridclip(selectFP);
299        if(tOrP==true)
300        {
301           if(hiLoFlag==true)
302              meanTR = new
Raster(root_dir+"input_files/ann_m100.asc",yll,yur,"N");
303           else
304              meanTR = new
Raster(root_dir+"input_files/t_ann.asc",yll,yur,"none");
305           meanTR=meanTR.gridclip(selectFP);
306        }
307        else
308        {
309           meanTR = new
Raster(root_dir+"input_files/pptmap_cp.asc",yll,yur,"none");
310           meanTR=meanTR.gridclip(selectFP);
311           if(hiLoFlag==true)
312              meanTR.increaseRes(250);
313        }
```



```
314        lapseRsqR = new Raster(topoR,topoR.getNoData());
315        lapseRateR = new Raster(topoR,topoR.getNoData());
316        lapseRateInterceptR = new Raster(topoR,topoR.getNoData());
317        int iStart =
(int)((selectFootprint.getXll()-selectFP.getXll())/demR.getCellSize());
318        int jStart =
(int)((selectFootprint.getYll()-selectFP.getYll())/demR.getCellSize());
319        for(int i=iStart;i<iStart+topoR.getXcells();i++)
320           for(int j=jStart;j<jStart+topoR.getYcells();j++)
321           {
322              if((demR.getValue(i,j)>0)&&(meanTR.getValue(i,j)!=noData))
323              {
324                 demSample = demR.boxValues(i,j,50,0);
325                 avgTsample = meanTR.boxValues(i,j,50,noData);
326                 reg=new Regression(demSample,avgTsample,noData);
327                 float
val=(reg.getGradient()*demR.getValue(i,j))+reg.getIntercept();
328                 meanTR.changeValue(i,j,val);
329              }
330              if(demR.getValue(i,j)>0)
331              {
332                 demSample = demR.boxValues(i,j,fr,0);
333                 avgTsample = meanTR.boxValues(i,j,fr,noData);
334                 reg=new Regression(avgTsample,demSample,noData);
335                 lapseRsqR.changeValue(i-iStart,j-jStart,reg.getRSquared());
336
lapseRateR.changeValue(i-iStart,j-jStart,1f/reg.getGradient());
337
lapseRateInterceptR.changeValue(i-iStart,j-jStart,reg.getIntercept());
338              }
339           }
340        SimpleIO.rasterToAscii(folder,filePrefix+"_lapse_rsq",lapseRsqR);
341
SimpleIO.rasterToAscii(folder,filePrefix+"_lapse_intercept",lapseRateInter
ptR);
342        SimpleIO.rasterToAscii(folder,filePrefix+"_lapse_rate",lapseRateR);
343        lapseRateR.setParameters(folder,"Lapse
Rate",filePrefix+"_lapse_rate","lapse",false);
344        return lapseRateR;
345     }
346
347     public void writeRasters() throws Exception
348     {
349
SimpleIO.writeRaster("elaOffsetCplx",meanTR,this.elaOffset,nHor,nVer,meanT
getCellSize());
350
SimpleIO.writeRaster("screwup",meanTR,this.screwup,nHor,nVer,meanTR.getCel
ize());
351  //
SimpleIO.writeRaster("regressDEM",meanTR,meanTR.getRaster(),meanTR.getXcel
(),meanTR.getYcells(),meanTR.getCellSize());
352  //
SimpleIO.writeRaster("regressTemp",averageTR,averageTR.getRaster(),average
.getXcells(),averageTR.getYcells(),averageTR.getCellSize());
353
SimpleIO.writeRaster("ela_map_cplx_test",meanTR,this.elaMap,nHor,nVer,mean
.getCellSize());
354
SimpleIO.writeRaster("ela_rsq_cplx_test",meanTR,this.lapseRSq,nHor,nVer,me
TR.getCellSize());
355
SimpleIO.writeRaster("lapse_rate_cplx_test",meanTR,this.lapseRate,nHor,nVe
meanTR.getCellSize());
356     }
357
358     public static void main(String args[]) throws Exception
359     {
360        ELAFinderComplex ef = new ELAFinderComplex(new ClimateControl(),1);
361
362  /*    ELAFinderComplex ef = new ELAFinderComplex(1);
```

```
363         Raster demR = new Raster("C:/gisdata/icedem_cp.asc");
364         PrecipModel pml = new PrecipModel();
365         float[][] demArray =
    pml.reduceRes(demR.getRaster(),demR.getXcells(),demR.getYcells(),2,1);
366         demR.replaceRaster(demArray,demR.getCellSizeD()*2d,nHor,nVer);
367         Raster meanTR = new Raster("C:/gisdata/t_ann_cp.asc");
368         ef.calcELA(15,demR,meanTR);*/
369       }
370   }
```

```
1   /* A raster class initially designed by RSP for software engineering class
2   Reads in data from ARC grid ascii files using methods readHeader and
    readData
3   modified and updated by AFC, 29/4/04-7/4/05.  Provides raster manipulation
    methods for the
4   Iceland model*/
5   package afc.spatial;
6
7   import java.io.*;
8   import java.util.*;
9   import afc.utilities.*;
10
11  public class Raster
12  {
13
14     public float [][] raster;
15     public int nCols,nRows,noData,boxSize;
16     public double xurCorner,yurCorner,xllCorner,yllCorner,cellSize;
17
18     public Point ll,ur;
19     public Rectangle footprint;
20     public String scaleType,name,directory,asciiName;
21     public boolean sea;
22
23     //--------------- Constructors ------------------
24     // Loads raster data into a floating array from the given file
25     public Raster(String fileName) throws Exception
26     {
27        loadRaster(fileName);
28        setDefaults();
29     }
30
31     //constructs an entire large raster
32     public Raster(String fileName,int numRowsToReadAtOnce,String noDataKey)
    throws Exception
33     {
34        loadRaster(fileName,numRowsToReadAtOnce,noDataKey);
35        setDefaults();
36     }
37
38     //constructs a raster from a piece (in rows) of a file
39     public Raster(String fileName,double lowY,double highY,String noDataKey)
    throws Exception
40     {
41        loadRaster(fileName,lowY,highY,noDataKey);
42        setDefaults();
43     }
44
45     //creates a blank raster
46     public Raster()
47     {
48        createRaster(0,0,0,0,0,-9999);
49        this.raster = new float[0][0];
50     }
51
52     //Creates a new raster based on given values. Has uniform initial values
53     public Raster(double xllCorner,double yllCorner,
54                   int nCols,int nRows,double cellSize,
55                   int noData,float initValue)
56     {
57        createRaster(xllCorner,yllCorner,nCols,nRows,cellSize,noData);
58        for(int i=0;i<nCols;i++)
59           for(int j=0;j<nRows;j++)
60              this.raster[i][j] = initValue;
61     }
62
63     //Creates a new raster based on a footprint (Rectangle) and a cellsize
64     //Has uniform initial values
65     public Raster(Rectangle footP,double cellSize,int noData,float initValue)
66     {
67        nCols = (int)((footP.getXur()-footP.getXll())/cellSize);
```



```
68        nRows = (int)((footP.getYur()-footP.getYll())/cellSize);
69        this.noData = -9999;
70
    createRaster(footP.getXll(),footP.getYll(),nCols,nRows,cellSize,noData);
71        for(int i=0;i<nCols;i++)
72           for(int j=0;j<nRows;j++)
73              this.raster[i][j] = initValue;
74     }
75
76     //creates a new raster with uniform initial values based on another
    Raster's dimensions
77     public Raster(Raster r2,int initValue)
78     {
79
    createRaster(r2.getXll(),r2.getYll(),r2.getXcells(),r2.getYcells(),r2.getC
    llSize(),r2.getNoData());
80        for(int i=0;i<nCols;i++)
81           for(int j=0;j<nRows;j++)
82              this.raster[i][j] = (float)initValue;
83        this.sea = r2.getSea();
84        scaleType = r2.getScaleType();
85     }
86
87     //creates a new raster with uniform initial values based on another
    Raster's dimensions
88     public Raster(String fileName,int initValue) throws Exception
89     {
90        System.out.println(fileName);
91        Vector data = SimpleIO.readLinesLargeFile(fileName,0,6);
92        this.readHeader(data);
93        for(int i=0;i<nCols;i++)
94           for(int j=0;j<nRows;j++)
95              this.raster[i][j] = (float)initValue;
96     }
97
98     //creates a new raster from another Raster
99     public Raster(Raster r2)
100    {
101
    createRaster(r2.getXll(),r2.getYll(),r2.getXcells(),r2.getYcells(),r2.getC
    llSize(),r2.getNoData());
102       for(int i=0;i<nCols;i++)
103          for(int j=0;j<nRows;j++)
104             this.raster[i][j] = r2.getValue(i,j);
105       this.sea = r2.getSea();
106       scaleType = r2.getScaleType();
107       setDirectory(r2.getDirectory());
108       setAsciiName(r2.getAsciiName());
109       setName(r2.getName());
110    }
111
112
113    //creates a new raster based on another Raster's dimensions
114    //and a different array (which should normally be the same dimensions a
    the Raster)
115    public Raster(Raster r2,float[][] array)
116    {
117
    createRaster(r2.getXll(),r2.getYll(),r2.getXcells(),r2.getYcells(),r2.getC
    llSize(),r2.getNoData());
118       this.raster = new float[nCols][nRows];
119       this.raster = array;
120       this.sea = r2.getSea();
121       scaleType = r2.getScaleType();
122    }
123
124    public Raster(Raster r2,int[][] array)
125    {
126
    createRaster(r2.getXll(),r2.getYll(),r2.getXcells(),r2.getYcells(),r2.getC
    llSize(),r2.getNoData());
```

```
127         this.raster = new float[nCols][nRows];
128         for(int i=0;i<nCols;i++)
129            for(int j=0;j<nRows;j++)
130               this.raster[i][j] = array[i][j];
131         this.sea = r2.getSea();
132         scaleType = r2.getScaleType();
133      }
134
135
136      //constructs an abstract Raster with a slope and random internal
    variability
137      public Raster(double xllCorner,double yllCorner,int nCols,int nRows,
138                    double gradient, double variability) throws Exception
139      {
140         Random ran = new Random();
141
142         int i,j,d;
143         double dbl;
144         double c = 127-(gradient*(nRows/2));
145         createRaster(xllCorner,yllCorner,nCols,nRows,1,-9999);
146         for(i=0;i<nRows;i++)
147            for(j=0;j<nCols;j++)
148            {
149               dbl = ran.nextDouble();
150               d = (int)(254-(50*dbl));
151               d = (int)(variability*dbl);
152               d += (int)((i*gradient)+c);
153               if (d>255)
154               {
155                  d-= (int)((i*gradient)+c-255);
156                  if (d>255)
157                     d= 255;
158               }
159               if (d<0)
160               {
161                  d+= (int)((i*gradient)+c);
162                  if (d<0)
163                     d= 0;
164               }
165               this.raster[j][i] = (float)d;
166            }
167         this.report();
168      }
169
170      //creates a Raster with a series of random hills set within it (using
    PointsetRandom)
171      public Raster(int xllCorner,int yllCorner,int nCols,int nRows,
172            int maxHeight,int minHeight,int nPoints,boolean hillCheck) throws
    Exception
173      {
174         double posHeight,ptDistance,pixelValue;
175         double maxDistFromPt = Math.sqrt((nCols*nCols)+(nRows*nRows));
176
177         createRaster(xllCorner,yllCorner,nCols,nRows,1,-9999);
178
179         PointsetRandom pSet = new PointsetRandom(nPoints,this);
180         pSet.createRandomPoints();
181         pSet.checkMinDist((int)(nCols/40));
182         Vector hills = pSet.getPoints();
183         for (int i=0;i<nCols;i++)
184            for (int j=0;j<nRows;j++)
185            {
186               Point rasterP = new
    Point(xllCorner+(i*cellSize),yllCorner+(j*cellSize));
187               Point nearPt = rasterP.nearestPoint(hills,hills.size());
188               ptDistance =  rasterP.distanceFrom(nearPt);
189               //pixelValue is a relative height of 0 to 1 based on distances
    and a hill stretching factor (8)
190               pixelValue =
    ((Math.cos(Math.toRadians((ptDistance/(maxDistFromPt/8))*180))+1)/2)*maxHe
    ght;
```



```
191               pixelValue = minHeight+((maxHeight-minHeight)*pixelValue);
192            //defines everything beyond (maxDistFromPt/8) as minHeight
193               if ((ptDistance/(maxDistFromPt/8))>1)
194                  pixelValue = minHeight;
195               this.raster[i][j] = (float)pixelValue;
196            }
197      }
198
199      //assigns values in a new raster but doesn't define the array values;
200      private void createRaster(double xllCorner,double yllCorner,int nCols,i
    nRows,double cellSize,int noData)
201      {
202         this.cellSize = cellSize;
203         this.xllCorner=xllCorner;
204         this.yllCorner=yllCorner;
205         this.nCols = nCols;
206         this.nRows = nRows;
207         this.noData = noData;
208         this.xurCorner = xllCorner+(this.nCols*this.cellSize);
209         this.yurCorner = yllCorner+(this.nRows*this.cellSize);
210         this.ll=new Point(xllCorner,yllCorner);
211         this.ur=new Point(xurCorner,yurCorner);
212         this.footprint = new Rectangle(ll,ur);
213         this.raster = new float[nCols][nRows];
214         setDefaults();
215      }
216
217      // Method which loads a raster
218      private void loadRaster(String fileName) throws Exception{
219         System.out.println(fileName);
220         // Get all the data in our file
221         Vector data = SimpleIO.getLines(fileName);
222         this.readHeader(data);
223         this.readData(data);
224         this.report();
225      }
226
227      // Method which loads a large raster
228      private void loadRaster(String fileName,int numRowsToReadAtOnce,String
    noDataKey) throws Exception
229      {
230         int addR=6;//initial values for header reading
231         int r=0;
232         System.out.println(fileName);
233
234         // Get header data and parse it to get the number of rows in the fil
235         Vector data = SimpleIO.readLinesLargeFile(fileName,r,addR);
236         this.readHeader(data);
237         r=6;
238         while(addR<this.nRows+6)
239         {
240            if(r+numRowsToReadAtOnce>=this.nRows+6)
241               addR=this.nRows+6;
242            else
243               addR=r+numRowsToReadAtOnce;
244            System.out.println("Getting rows in large raster from "+r+" to
    "+addR);
245            data = SimpleIO.readLinesLargeFile(fileName,r,addR);
246            this.readData(data,r,addR,noDataKey);
247            r=r+numRowsToReadAtOnce;
248         }
249         this.report();
250      }
251
252      // Method which loads a piece of a large raster
253      //lowY and highY are in raster units (eg metres)
254      private void loadRaster(String fileName,double lowY,double highY,String
    noDataKey) throws Exception
255      {
256         int addR=6;//initial values for header reading
257         int r=0;
```

```
258         System.out.println(fileName);
259
260         // Get header data and parse it to get the number of rows in the fil
261         Vector data = SimpleIO.readLinesLargeFile(fileName,r,addR);
262         this.readHeader(data);
263   //    System.out.println(nCols+" "+nRows);
264         //first row of the file to get
265   //    System.out.println(yllCorner+" "+lowY+" "+highY);
266         int lowYRow = nRows-(int)((float)(highY-yllCorner)/cellSize);
267         //row after the last row to get
268         int highYRow = nRows-(int)((float)(lowY-yllCorner)/cellSize);
269   //    System.out.println(lowYRow+" "+highYRow);
270         if(highYRow>=this.nRows)
271            addR=this.nRows+6;
272         else
273            addR=highYRow+6;
274         if(lowYRow<0)
275            lowYRow=0;
276         r=lowYRow+6;
277
278         this.yllCorner = lowY;
279         this.nRows = addR-r;
280         this.yurCorner = yllCorner+(this.nRows*this.cellSize);
281   //    System.out.println("here?");
282         this.ll=new Point(xllCorner,yllCorner);
283         this.ur=new Point(xurCorner,yurCorner);
284   //    ll.displayValues();
285   //    ur.displayValues();
286   //    System.out.println(nCols+" "+nRows);
287         this.footprint = new Rectangle(ll,ur);
288         this.raster = new float[this.nCols][this.nRows];
289   //    report();
290         System.out.println("Getting rows in large raster from "+r+" to
      "+addR);
291         data = SimpleIO.readLinesLargeFile(fileName,r,addR);
292         this.readData(data,6,nRows+6,noDataKey);
293         this.report();
294      }
295
296
297      // Method to display raster properties
298      public void report()
299      {
300         System.out.println("The current array is " + this.nCols + " x " +
      this.nRows + ".");
301         System.out.print("Corners are: (" + this.xllCorner + "," +
      this.yllCorner + ") & (");
302         System.out.println(this.xurCorner + "," + this.yurCorner + ").");
303         System.out.println("The cellsize is " + this.cellSize);
304         System.out.println("Min and max vals not noData:
      "+getMinValNotNoData(noData,noData)+","+getMaxVal(noData,noData));
305      }
306
307      // Read the header of an arc DEM
308      private void readHeader(Vector data) throws Exception{
309         /*The header is of the following form (GIS Ascii format)
310         NCOLS 401
311         NROWS 401
312         XLLCORNER 200000
313         YLLCORNER 760000
314         CELLSIZE 50
315         NODATA_value -9999
316         */
317         String s = (String) data.elementAt(0);
318         Vector line = SimpleIO.parseLine(s," ");
319         // Get the second element, i.e the number
320         this.nCols = Integer.parseInt((String) line.elementAt(1));
321
322
323         s = (String) data.elementAt(1);
324         line = SimpleIO.parseLine(s," ");
```



```
325         this.nRows = Integer.parseInt((String) line.elementAt(1));
326
327         s = (String) data.elementAt(2);
328         line = SimpleIO.parseLine(s," ");
329         this.xllCorner= (double) Double.parseDouble((String)
      line.elementAt(1));
330
331         s = (String) data.elementAt(3);
332         line = SimpleIO.parseLine(s," ");
333         this.yllCorner = (double) Double.parseDouble((String)
      line.elementAt(1));
334
335         s = (String) data.elementAt(4);
336         line = SimpleIO.parseLine(s," ");
337         this.cellSize = (double) Double.parseDouble((String)
      line.elementAt(1));
338
339         s = (String) data.elementAt(5);
340         line = SimpleIO.parseLine(s," ");
341         this.noData = (int) Double.parseDouble((String) line.elementAt(1));
342
343         // Set array parameters based on initial data
344         this.raster = new float[this.nCols][this.nRows];
345         this.xurCorner = xllCorner+(this.nCols*this.cellSize);
346         this.yurCorner = yllCorner+(this.nRows*this.cellSize);
347         this.ll=new Point(xllCorner,yllCorner);
348         this.ur=new Point(xurCorner,yurCorner);
349         this.footprint = new Rectangle(ll,ur);
350      }
351
352      private void readData(Vector data)throws Exception
353      {
354         // First 6 rows of data (0-5) are header
355         int startRow = 6;
356         Vector line;
357         String s;
358         float value;
359         // Loop through all the data, getting each line from the data file
360         // and parsing it
361         for (int j=0; j < nRows; j++){
362            s = (String) data.elementAt(startRow+j);
363            line = SimpleIO.parseLine(s," ");
364            for(int i=0; i < nCols; i++){
365               value = (float) Double.parseDouble((String) line.elementAt(i))
366               //System.out.println(i + " " +j);
367               raster[i][nRows-1-j] = value;
368               //System.out.println(value);
369            }
370         }
371
372      }
373
374      private void readData(Vector data,int startRow,int endRow,String
      noDataKey)throws Exception{
375
376         Vector line;
377         String s;
378         float value;
379         endRow-=6;
380         startRow-=6;
381         int numRows = endRow-startRow;
382         // Loop through all the data, getting each line from the data file
383         // and parsing it
384         for (int j=0; j < numRows; j++)
385         {
386            s = (String) data.elementAt(j);
387   //       System.out.println("Parsing Row: "+(j+startRow));
388            line = SimpleIO.parseLine(s," ");
389            for(int i=0; i < nCols; i++)
390            {
391               if(noDataKey.compareTo((String)line.elementAt(i))==0)
```

```
392                 value = noData;
393               else
394                 value = (float) Double.parseDouble((String)
    line.elementAt(i));
395               raster[i][(nRows-startRow)-1-j] = value;
396   //            System.out.println((startRow+j)+"
    "+((nRows-endRow)+(numRows-1-j)));
397             }
398   //        System.out.println();
399         }
400       }
401
402   // This gets the aspect of single cell (i,j) from -180 to 180(N-E-S-W-N)
403       public double getAspect(int i, int j,boolean absolute){
404         double aspect;
405         if(absolute==true)
406         {
407           i=(int)absToRelX(i);
408           j=(int)absToRelY(j);
409         }
410         if ((i == 0) || (j == 0) || (i == this.nCols-1) || (j ==
    this.nRows-1))
411           return this.noData;
412
413         aspect =
    Math.toDegrees(Math.atan2(eastWestGradient(i,j),southNorthGradient(i,j)))
414
415         return aspect;
416       }
417
418   // Calculates the gradient of single cell (i,j) in degrees
419       public double getGradient(int i, int j,boolean absolute){
420         double gradient;
421         double tanGradient;
422         if(absolute==true)
423         {
424           i=(int)absToRelX(i);
425           j=(int)absToRelY(j);
426         }
427
428         if ((i == 0) || (j == 0) || (i == this.nCols-1) || (j ==
    this.nRows-1))
429           return this.noData;
430
431         tanGradient = Math.sqrt(
432                   (double)
    (this.southNorthGradient(i,j)*this.southNorthGradient(i,j)) +
433                   (double)
    (this.eastWestGradient(i,j)*this.eastWestGradient(i,j)));
434
435
436         gradient = Math.toDegrees(Math.atan(tanGradient));
437         return gradient;
438       }
439
440       // Gradients calculated using pp190 of Burrough
441       private float eastWestGradient(int i, int j){
442         // First we check that i and j are not on the boundaries of our grid
443         float gradient;
444         gradient =0f;
445
446
447         gradient = this.raster[i+1][j+1] + this.raster[i+1][j-1] +
448                    2f * this.raster[i+1][j];
449
450         gradient = gradient -
451                   (this.raster[i-1][j+1] + this.raster[i-1][j-1] +
452                    2f * this.raster[i-1][j]);
453
454         gradient = gradient / (8f * (float)this.cellSize);
455
```



```
456         // Simpler treatment of finite differences to calculate
457         // the gradient
458         /*
459         gradient = (this.raster[i-1][j] + this.raster[i+1][j])/
460           (2f * this.cellSize);
461         */
462         return gradient;
463
464       }
465
466
467       private float southNorthGradient(int i, int j){
468         float gradient;
469         gradient =0f;
470
471         gradient = this.raster[i+1][j+1] + this.raster[i-1][j+1] +
472                    2f * this.raster[i][j+1];
473
474         gradient = gradient -
475                   (this.raster[i+1][j-1] + this.raster[i-1][j-1] +
476                    2f * this.raster[i][j-1]);
477
478         gradient = gradient / (8f * (float)cellSize);
479
480         // Simpler treatment of finite differences to calculate
481         // the gradient
482         /*
483         gradient = (this.raster[i][j-1] + this.raster[i][j+1])/
484           (2f * this.cellSize);
485         */
486
487         return gradient;
488
489       }
490
491       public float boxMean(int x,int y,boolean absolute)
492       {
493         int i,j,boxRad;
494         float total,average;
495
496         if(absolute==true)
497         {
498           x=(int)absToRelX(x);
499           y=(int)absToRelY(y);
500         }
501
502         boxRad = (this.boxSize-1)/2;
503         total = 0;
504         for (i=0; i<this.boxSize;i++)
505           for(j=0;j<this.boxSize;j++)
506             total += (float) this.raster[i+(x-boxRad)][j+(y-boxRad)];
507         average = total / ((this.boxSize)*(this.boxSize));
508
509         return average;
510       }
511
512       public float boxEdgeMean(int x,int y,boolean absolute)
513       {
514         int i,j,boxRad,counter;
515         float total,average;
516
517         if(absolute==true)
518         {
519           x=(int)absToRelX(x);
520           y=(int)absToRelY(y);
521         }
522
523         boxRad = (this.boxSize-1)/2;
524         total = 0;
525         counter = 0;
526         for (i= -boxRad; i<boxRad;i++)
```

```
527             for(j= -boxRad;j<boxRad;j++)
528                if ((x+i>=0) && (x+i<this.nCols) &&
529                   (y+j>=0) && (y+j<this.nRows))
530                {
531                   counter+=1;
532                   total += (float) this.raster[i+x][j+y];
533                }
534          average = total / counter;
535
536          return average;
537       }
538
539       public float[][] boxFocalMean(int boxSize, boolean overwrite,boolean
      toTheEdge)
540       {
541          int boxRad,i,j;
542          float rasterMean[][] = new float[this.nCols][this.nRows];
543          double mean;
544          this.boxSize = boxSize;
545
546          if ((this.boxSize%2==1)&&(this.boxSize>=3))
547          {
548             boxRad = (this.boxSize-1)/2;
549             for (i=0;i<(this.nCols);i++)
550                for (j=0;j<(this.nRows);j++)
551                {
552                   if (toTheEdge ==false)
553                   {
554
555                      if (i>(boxRad) && i<(this.nCols-(boxRad+1)) &&
556                               j>(boxRad) && (j<(this.nRows-(boxRad+1))))
557                         rasterMean[i][j] = this.boxMean(i,j,false);
558                      else
559                         rasterMean[i][j] = this.noData;
560                   }
561                   else if (toTheEdge ==true)
562                      rasterMean[i][j] = this.boxEdgeMean(i,j,false);
563                }
564          }
565          if (overwrite ==true)
566             this.raster = rasterMean;
567          return rasterMean;
568       }
569
570       public float[] boxValues(int i,int j,int boxSize,int ignoreValue)
571       {
572          int x,y;
573          int counter=0;
574          float[] values = new float[1];
575          if(boxSize>0)
576          {
577             values = new float[((boxSize*2)+1)*((boxSize*2)+1)];
578             for(x=0;x<(((boxSize*2)+1)*((boxSize*2)+1));x++)
579                values[x]=this.noData;
580             for (x=i-boxSize;x<=i+boxSize;x++)
581                for (y=j-boxSize;y<=j+boxSize;y++)
582                {
583                   if ((x>=0)&&(x<this.nCols)&&(y>=0)&&(y<this.nRows))
584
      if((this.raster[x][y]!=this.noData)&&(this.raster[x][y]!=ignoreValue))
585                   {
586                      values[counter] = this.raster[x][y];
587
588                   }
589                counter+=1;
590                }
591          }
592          return values;
593       }
594
595       public float[][] getBoxSubRaster(int i,int j,int boxSize)
```



```
596       {
597          int x,y;
598          float[][] boxArray = new float[(boxSize*2)+1][(boxSize*2)+1];
599          if(boxSize>0)
600          {
601             for (x=i-boxSize;x<=i+boxSize;x++)
602                for (y=j-boxSize;y<=j+boxSize;y++)
603                {
604                   if ((x>=0)&&(x<this.nCols)&&(y>=0)&&(y<this.nRows))
605                      boxArray[x+boxSize-i][y+boxSize-j] = this.raster[x][y];
606                   else
607                      boxArray[x+boxSize-i][y+boxSize-j] = noData;
608                }
609          }
610          return boxArray;
611       }
612
613
614       public void invertRaster()
615       {
616          int i,j;
617          for (i=0;i<this.nRows;i++)
618             for (j=0;j<this.nCols;j++)
619                this.raster[i][j] = 255- this.raster[i][j];
620       }
621
622       public void drawRoundPoint(Point p,int radius)
623       {
624          int x,y,i,j;
625          x = (int)p.getX();
626          y = (int)p.getY();
627          x=(int)absToRelX(x);
628          y=(int)absToRelY(y);
629
630          Point testPoint;
631          int[][] point = new int[radius*2][radius*2];
632          for (i=x-radius;i<x+radius;i++)
633             for (j=y-radius;j<y+radius;j++)
634             {
635                testPoint = new Point(i,j);
636                if(testPoint.distanceFrom(p)<=radius)
637                   if
      ((i>this.xllCorner)&&(i<(this.xllCorner+nCols))&&((j>this.yllCorner)&&(j<(
      his.yllCorner+nRows))))
638                   {
639                      if (this.raster[i][j]<128)
640                         this.raster[i][j] = 255;
641                      else
642                         this.raster[i][j] = 0;
643                   }
644             }
645       }
646
647       public void drawCircle(Point p,double radius,double lineWidth)
648       {
649          int x,y,i,j;
650          Vector lineData = new Vector();
651          x = (int)p.getX();
652          y = (int)p.getY();
653          x=(int)absToRelX(x);
654          y=(int)absToRelY(y);
655          Point testPoint = new Point();
656
657          for (i=(int)xllCorner;i<((int)xllCorner+this.nCols);i++)
658          {
659             for (j=(int)yllCorner;j<((int)yllCorner+this.nRows);j++)
660             {
661                testPoint = new Point(i,j);
662                if((testPoint.distanceFrom(p)<=(radius+(lineWidth/2)))
663                      &&(testPoint.distanceFrom(p)>=(radius-(lineWidth/2))))
```

```
664                     this.raster[i-(int)xllCorner][j-(int)yllCorner] = 1;
665             else if (testPoint.distanceFrom(p)<(radius-(lineWidth/2)))
666                     this.raster[i-(int)xllCorner][j-(int)yllCorner] = 255
667          }
668       }
669    }
670
671    public void drawTpCircle(Point p,double radius,double lineWidth)
672    {
673       int x,y,i,j;
674       Vector lineData = new Vector();
675       x = (int)p.getX();
676       y = (int)p.getY();
677       x=(int)absToRelX(x);
678       y=(int)absToRelY(y);
679       Point testPoint = new Point();
680
681       for (i=(int)xllCorner;i<((int)xllCorner+this.nCols);i++)
682       {
683          for (j=(int)yllCorner;j<((int)yllCorner+this.nRows);j++)
684          {
685             testPoint = new Point(i,j);
686             if((testPoint.distanceFrom(p)<=(radius+(lineWidth/2)))
687                   &&(testPoint.distanceFrom(p)>=(radius-(lineWidth/2))))
688                this.raster[i-(int)xllCorner][j-(int)yllCorner] = 1;
689          }
690       }
691    }
692
693    public void drawRaster(Raster subRast)
694    {
695       int i,j;
696       int noColour = 255;
697       int subXll=(int)subRast.getXll()-(int)xllCorner;
698       int subYll=(int)subRast.getYll()-(int)yllCorner;
699
700       for (i=subXll;i<subXll+subRast.getWidth();i++)
701          for (j=subYll;j<subYll+subRast.getHeight();j++)
702          {
703                if ((i>0)&&(i<nCols)&&((j>0)&&(j<nRows)))
704                {
705                   if((subRast.getValue(i-subXll,j-subYll,false)!=noColour)
706                      && (subRast.getValue(i-subXll,j-subYll,false)!=245))
707                         this.raster[i][j] =
    subRast.getValue(i-subXll,j-subYll,false);
708                }
709          }
710    }
711
712    //for use with VikingThiessen boundary rasters
713    public void highlightRaster(Raster subRast,int highlightAmt)
714    {
715       System.out.println("highlight reached here#1");
716       if (highlightAmt<0)
717          highlightAmt=0;
718       else if (highlightAmt>255)
719          highlightAmt=255;
720       int i,j;
721
722       System.out.println("highlight reached here#2");
723       for
    (i=(int)subRast.getXll();i<(int)subRast.getXll()+subRast.getWidth();i++)
724          for
    (j=(int)subRast.getYll();j<(int)subRast.getYll()+subRast.getHeight();j++)
725          {
726                //check the area is on the main raster (this one)
727                if
    ((i>this.xllCorner)&&(i<(this.xllCorner+nCols))&&((j>this.yllCorner)&&(j<(
    his.yllCorner+nRows))))
728                {
729                   //pick out the pixels drawing the circle
```



```
730                if
    (subRast.getValue(i-(int)subRast.getXll(),j-(int)subRast.getYll(),false)!=
    45)
731                {
732                   this.raster[i-(int)xllCorner][j-(int)yllCorner] =
    highlightAmt;
733                   if
    (this.raster[i-(int)xllCorner][j-(int)yllCorner]>=255)
734                      this.raster[i-(int)xllCorner][j-(int)yllCorner]=25
735                }
736             /*highlight all other pixels by increasing or decreasing
    their value by a set amount
737                //apart from those that have been assigned 245, which
    won't be highlighted
738                else
    if(subRast.getValue(i-subRast.getXll(),j-subRast.getYll())==245)
739                {
740                   if (this.raster[i-this.xllCorner][j-this.yllCorner]
    <128)
741                   this.raster[i-this.xllCorner][j-this.yllCorner] -=
    highlightAmt;
742                   if (this.raster[i-this.xllCorner][j-this.yllCorner] <
743                      this.raster[i-this.xllCorner][j-this.yllCorner] =0
744                   else if
    (this.raster[i-this.xllCorner][j-this.yllCorner] >=128)
745                   this.raster[i-this.xllCorner][j-this.yllCorner] -=
    highlightAmt;
746                }*/
747             }
748          }
749    }
750
751    public void drawLine(double sX,double sY,double eX,double eY,int
    lineWeight)
752    {
753       int oX,oY,rW,rH,i,j,startX,startY,endX,endY;
754       int noColour = 255;
755       Vector lineData = new Vector();
756       startX = (int)sX;
757       startY = (int)sY;
758       endX = (int)eX;
759       endY = (int)eY;
760       LineRaster lr = new
    LineRaster(startX,startY,endX,endY,lineWeight,cellSize);
761       int[][] lineR = lr.extractRaster();
762       oX = lr.getOriginX();
763       oX += lr.getX1();
764       oY = lr.getOriginY();
765       oY += lr.getY1();
766       rW = lr.getLineRasterWidth();
767       rH = lr.getLineRasterHeight();
768       System.out.println("array: "+oX+" "+rW+" -- "+oY+" "+rH);
769       System.out.println("xll: "+xllCorner+" "+yllCorner);
770       for (i=oX;i<oX+rW;i++)
771          for(j = oY;j<oY+rH;j++)
772             if (lineR[i-oX][j-oY] != noColour)
773                if
    ((i>this.xllCorner)&&(i<(this.xllCorner+this.nCols))&&((j>this.yllCorner)&
    (j<(this.yllCorner+this.nRows))))
774                   this.raster[i-(int)xllCorner][j-(int)yllCorner] =
    lineR[i-oX][j-oY];
775    }
776
777    public void drawTrimmedLine(double sX,double sY,double eX,double eY,int
    lineWeight)
778    {
779       int oX,oY,rW,rH,i,j,startX,startY,endX,endY;
780       int noColour = 255;
781       Vector lineData = new Vector();
782       startX = (int)sX;
783       startY = (int)sY;
```

```
784         endX = (int)eX;
785         endY = (int)eY;
786         LineRaster lr = new
      LineRaster(startX,startY,endX,endY,lineWeight,cellSize);
787         lr.trimLineRaster();
788         int[][] lineR = lr.extractRaster();
789         oX = lr.getOriginX()+lineWeight;
790         oX += lr.getX1();
791         oY = lr.getOriginY()+lineWeight;
792         oY += lr.getY1();
793
794         rW = lr.getLineRasterWidth();
795         rH = lr.getLineRasterHeight();
796
797         for (i=oX;i<oX+rW;i++)
798            for(j = oY;j<oY+rH;j++)
799               if (lineR[i-oX][j-oY] != noColour)
800                  this.raster[i][j] = lineR[i-oX][j-oY];
801
802      }
803
804      public void printRasterData()
805      {
806         for(int j=0;j<this.nRows;j++)
807         {
808            for(int i=0;i<this.nCols;i++)
809               System.out.print(this.raster[i][this.nRows-j-1]+" ");
810            System.out.println();
811         }
812      }
813
814      public void printVals(float[][] array)
815      {
816         for(int j=0;j<this.nRows;j++)
817         {
818            for(int i=0;i<this.nCols;i++)
819               System.out.print(array[i][j]+" ");
820            System.out.println();
821         }
822      }
823      public Vector generateHeader()
824      {
825         Vector v2 = new Vector();
826         Vector v=new Vector();
827         v.addElement(new String("NCOLS "+this.nCols));
828         v2.addElement(v);
829         v=new Vector();
830         v.addElement(new String("NROWS "+this.nRows));
831         v2.addElement(v);
832         v=new Vector();
833         v.addElement(new String("XLLCORNER "+this.xllCorner));
834         v2.addElement(v);
835         v=new Vector();
836         v.addElement(new String("YLLCORNER "+this.yllCorner));
837         v2.addElement(v);
838         v=new Vector();
839         v.addElement(new String("CELLSIZE "+this.cellSize));
840         v2.addElement(v);
841         v=new Vector();
842         v.addElement(new String("NODATA_value "+this.noData));
843         v2.addElement(v);
844         return v2;
845      }
846
847      public Vector generateHeader(int nCol,int nRow,double cellSizeb)
848      {
849         Vector v2 = new Vector();
850         Vector v=new Vector();
851         v.addElement(new String("NCOLS"));
852         v.addElement(new Integer(nCol));
853         v2.addElement(v);
```



```
854         v=new Vector();
855         v.addElement(new String("NROWS"));
856         v.addElement(new Integer(nRow));
857         v2.addElement(v);
858         v=new Vector();
859         v.addElement(new String("XLLCORNER"));
860         v.addElement(new Double(this.xllCorner));
861         v2.addElement(v);
862         v=new Vector();
863         v.addElement(new String("YLLCORNER"));
864         v.addElement(new Double(this.yllCorner));
865         v2.addElement(v);
866         v=new Vector();
867         v.addElement(new String("CELLSIZE"));
868         v.addElement(new Double(cellSizeb));
869         v2.addElement(v);
870         v=new Vector();
871         v.addElement(new String("NODATA_value"));
872         v.addElement(new Integer(this.noData));
873         v2.addElement(v);
874         return v2;
875      }
876
877
878      // Methods to return the values of the instance variables
879      public float[][] getRaster(){
880         return this.raster;
881      }
882
883      public double getXll(){
884         return this.xllCorner;
885      }
886      public double getYll(){
887         return this.yllCorner;
888      }
889
890      public double getXur(){
891         return this.xurCorner;
892      }
893      public double getYur(){
894         return this.yurCorner;
895      }
896
897      public double getCellSize(){
898         return this.cellSize;
899      }
900
901      public int getNoData(){
902         return this.noData;
903      }
904
905      public float getValue(int i,int j)
906      {
907         return this.raster[i][j];
908      }
909
910      public float getValue(Point p,boolean absolute)
911      {
912         int x,y;
913         if(absolute==true)
914         {
915            x = (int)((p.getX()-xllCorner)/cellSize);
916            y = (int)((p.getY()-yllCorner)/cellSize);
917            if(this.footprint.contains(p.getX(),p.getY()))
918               return this.raster[x][y];
919            else
920               return noData;
921         }
922         else
923         {
924            x = (int)p.getX();
```

```
925             y = (int)p.getY();
926             if((x>=0)&&(y>=0)&&(x<nCols)&&(y<nRows))
927                return this.raster[x][y];
928             else
929                return noData;
930          }
931       }
932
933       public float getAbsValue(double x,double y)
934       {
935          if(this.footprint.contains(x,y))
936             return
       this.raster[(int)((x-xllCorner)/cellSize)][(int)((y-yllCorner)/cellSize)]

937          else
938             return noData;
939       }
940
941       public float getValue(float i,float j,boolean absolute)
942       {
943          if(this.footprint.contains((double)i,(double)j))
944          {
945             int x=(int)i;
946             int y=(int)j;
947             if (absolute==true)
948             {
949                x=(int)absToRelX(i);
950                y=(int)absToRelY(j);
951             }
952             return this.raster[x][y];
953          }
954          else
955             return noData;
956       }
957
958       public void changeValue(int i,int j,float value)
959       {
960          this.raster[i][j] = value;
961       }
962
963       public void addValue(int i,int j,float value)
964       {
965          this.raster[i][j] += value;
966       }
967
968       public float multiplyValue(int i,int j,float value)
969       {
970          this.raster[i][j] = this.raster[i][j]*value;
971          return this.raster[i][j];
972       }
973
974
975       public double getWidth(){
976             return this.nCols * this.cellSize;
977       }
978
979       public double getHeight(){
980             return this.nRows * this.cellSize;
981       }
982
983       public int getXcells(){
984             return this.nCols;
985       }
986
987       public int getYcells(){
988             return this.nRows;
989       }
990
991       public Point getLL(){
992             return this.ll;
993       }
```



```
994
995       public Point getUR(){
996             return this.ur;
997       }
998
999       public Rectangle getFootprint(){
1000            return this.footprint;
1001      }
1002
1003      public boolean getSea(){
1004         return this.sea;
1005      }
1006      public void setSea(boolean set){
1007         this.sea = set;
1008      }
1009
1010      public String getScaleType(){
1011         return this.scaleType;
1012      }
1013
1014      public void setScaleType(String newType){
1015         this.scaleType=newType;
1016      }
1017
1018
1019      //for replacing rasters of the same size
1020      public void replaceRaster(float[][] newArray)
1021      {
1022         this.raster = newArray;
1023      }
1024
1025      //for replacing rasters of the same size
1026      public void replaceRaster(int[][] newArray)
1027      {
1028         for(int i=0;i<nCols;i++)
1029            for(int j=0;j<nRows;j++)
1030               this.raster[i][j] = newArray[i][j];
1031      }
1032      public void replaceRaster(float[][] newArray,int newCols,int
       newRows,double newCellSize)
1033      {
1034         this.raster = newArray;
1035         this.nCols = newCols;
1036         this.nRows = newRows;
1037         this.cellSize = newCellSize;
1038      }
1039
1040      //use with care! for replacing xll and yll points of the raster
1041      public void moveOrigin(double x,double y)
1042      {
1043         this.xllCorner = x;
1044         this.yllCorner=y;
1045         this.xurCorner = this.xllCorner+(nCols*cellSize);
1046         this.yurCorner = this.yllCorner+(nRows*cellSize);
1047         this.ll=new Point(xllCorner,yllCorner);
1048         this.ur=new Point(xurCorner,yurCorner);
1049         this.footprint = new Rectangle(ll,ur);
1050      }
1051
1052
1053      public void replaceRaster(boolean[][] newArray)
1054      {
1055         this.raster = new float[nCols][nRows];
1056         for (int i=0;i<nCols;i++)
1057            for(int j=0;j<nRows;j++)
1058            {
1059               if(newArray[i][j]==true)
1060                  raster[i][j]=1f;
1061               else
1062                  raster[i][j]=0f;
1063            }
```

```
1064      }
1065
1066      public void replaceRaster(Raster newR)
1067      {
1068         this.nCols = newR.getXcells();
1069         this.nRows = newR.getYcells();
1070         this.cellSize = newR.getCellSize();
1071         this.raster = newR.getRaster();
1072         this.xllCorner = newR.getXll();
1073         this.yllCorner = newR.getYll();
1074         this.xurCorner = newR.getXll()+(newR.getXcells()*newR.getCellSize(
1075         this.yurCorner = newR.getYll()+(newR.getYcells()*newR.getCellSize(
1076         this.ll=new Point(newR.getXll(),newR.getYll());
1077         this.ur=new Point(xurCorner,yurCorner);
1078         this.footprint = new Rectangle(ll,ur);
1079      }
1080      //simple resolution reduction method. Only works for integer reductio
1081      public void reduceRes(int resFactor)
1082      {
1083         if(resFactor!=1)
1084         {
1085            int tally;
1086            float val;
1087            int newXcells = (int)((float)nCols/resFactor);
1088            int newYcells = (int)((float)nRows/resFactor);
1089            double newCellSize = this.cellSize*resFactor;
1090            float[][] newArray = new float[newXcells][newYcells];
1091            for (int i=0;i<newXcells;i++)
1092               for (int j=0;j<newYcells;j++)
1093               {
1094                  tally = 0;
1095                  for(int p=0;p<resFactor;p++)
1096                     for(int q=0;q<resFactor;q++)
1097                     {
1098                        val = getValue((resFactor*i)+p,(resFactor*j)+q);
1099   //                     if(val!=noData)
1100                        if((val==20)||((val>0)&&(newArray[i][j]!=20)))
1101                        {
1102                           newArray[i][j]=val;
1103                           tally++;
1104                        }
1105                     }
1106                  //only put a new cell in
1107   //               if(tally!=0)
1108   //                  newArray[i][j]=20;
1109   //                  newArray[i][j] = newArray[i][j]/tally;
1110   //               else
1111                     if(tally==0)
1112                        newArray[i][j] = 0;
1113   //                  newArray[i][j] = noData;
1114               }
1115         replaceRaster(newArray,newXcells,newYcells,newCellSize);
1116         }
1117      }
1118
1119      public void increaseRes(double newCellSize)
1120      {
1121         int smallerX,smallerY;
1122         int resFactor = Math.round((float)(cellSize/newCellSize));
1123   /*    if(resFactor<(float)(cellSize/newCellSize))
1124         {
1125            resFactor +=1;
1126   //       System.out.println("** Cell Size Changed ** New size:
      "+newCellSize);
1127         }
1128   //    System.out.println("new res: "+resFactor);
1129   */    if(resFactor!=1)
1130         {
1131            int remj,remi;
1132            float val,remainderX,remainderY;
1133            int newXcells = (int)((float)nCols*resFactor);
```



```
1134            int newYcells = (int)((float)nRows*resFactor);
1135            float[][] newArray = new float[newXcells][newYcells];
1136            //currently very crude - is the same as the overlying old cell
1137            for (int i=0;i<newXcells;i++)
1138            {
1139               for (int j=0;j<newYcells;j++)
1140               {
1141                  smallerX = (int)(i-(i%resFactor))/resFactor;
1142                  smallerY = (int)(j-(j%resFactor))/resFactor;
1143   //               System.out.println((j%resFactor)+" "+i+" "+j+" rem:
      "+smallerX+" "+smallerY);
1144                  newArray[i][j] = this.raster[smallerX][smallerY];
1145               }
1146            }
1147         replaceRaster(newArray,newXcells,newYcells,newCellSize);
1148         }
1149      }
1150
1151      public Point cellCentrePos(int i,int j)
1152      {
1153         double x = xllCorner+(i*cellSize)+(cellSize/2);
1154         double y = yllCorner+(j*cellSize)+(cellSize/2);
1155         return new Point(x,y);
1156      }
1157
1158      public void bilinear(double x,double y)
1159      {
1160
1161      }
1162
1163      public float getMaxVal(float ignore1,float ignore2)
1164      {
1165         float testVal;
1166         int p=0;
1167         int q=0;
1168         float max = -Float.MAX_VALUE;
1169
1170         for(int i=0;i<this.nCols;i++)
1171            for(int j=0;j<this.nRows;j++)
1172            {
1173               testVal = this.raster[i][j];
1174               if((testVal>max)&&(testVal!=ignore1)&&(testVal!=ignore2))
1175                  max=testVal;
1176            }
1177         return max;
1178      }
1179
1180      public float getMinVal()
1181      {
1182         float testVal;
1183         float min = Float.MAX_VALUE;
1184         for(int i=0;i<this.nCols;i++)
1185            for(int j=0;j<this.nRows;j++)
1186            {
1187               testVal = this.raster[i][j];
1188               if(testVal<min)
1189                  min=testVal;
1190            }
1191         return min;
1192      }
1193
1194      public float getMinValNotNoData(float ignore1,float ignore2)
1195      {
1196         float testVal;
1197         float min = Float.MAX_VALUE;
1198         for(int i=0;i<this.nCols;i++)
1199            for(int j=0;j<this.nRows;j++)
1200            {
1201               testVal = this.raster[i][j];
1202
      if((testVal<min)&&(testVal!=noData)&&(testVal!=ignore1)&&(testVal!=ignor
```

```
1203                min=testVal;
1204             }
1205          return min;
1206       }
1207
1208       public Raster scaleValues(float minScale,float maxScale,float
      ignore1,float ignore2)
1209       {
1210          int i,j;
1211          float maxVal = getMaxVal(ignore1,ignore2);
1212          if(maxScale == noData)
1213             maxScale=maxVal;
1214          float minVal = getMinValNotNoData(ignore1,ignore2);
1215          Raster scaledRaster = new Raster(this,0);
1216          System.out.println(minScale+" -scale- "+maxScale+" "+minVal+" -val
      "+maxVal);
1217          if((minScale>minVal)||(maxScale<maxVal))
1218          {
1219             System.out.println("VALUES OUT OF SCALE RANGE: SCALE ADJUSTED")
1220             if(minScale>minVal)
1221                minScale=minVal;
1222             if(maxScale<maxVal)
1223                maxScale=maxVal;
1224          }
1225          for(i=0;i<this.nCols;i++)
1226             for(j=0;j<this.nRows;j++)
1227             {
1228                if(this.raster[i][j]==noData)
1229                   scaledRaster.changeValue(i,j,noData);
1230                else
      if(((sea==true)||(scaleType=="drainage"))&&(this.raster[i][j]==0))
1231                   scaledRaster.changeValue(i,j,0);
1232                else
      if((this.raster[i][j]!=ignore1)&&(this.raster[i][j]!=ignore2))
1233                {
1234                   if((sea==true)||(scaleType=="drainage"))
1235
      scaledRaster.changeValue(i,j,(((this.raster[i][j]-minScale)/(maxScale-mi
      ale))*254)+1);
1236                   else
1237
      scaledRaster.changeValue(i,j,(((this.raster[i][j]-minScale)/(maxScale-mi
      ale))*254));
1238                }
1239                else
1240                   scaledRaster.changeValue(i,j,this.raster[i][j]);
1241             }
1242          return scaledRaster;
1243       }
1244
1245       public void reclass(float centrePoint,float classSize)
1246       {
1247          int offsetVal;
1248    //    maxOffset =
      Math.max(Math.abs(centrePoint-getMaxVal()),Math.abs(centrePoint-getMinVa
      tNoData()));
1249    //    maxOffset = (int)(maxOffset/classSize)+1;//number of classes awy f
      the centrepoint
1250          for(int i=0;i<this.nCols;i++)
1251             for(int j=0;j<this.nRows;j++)
1252
      if(((sea==false)&&(this.raster[i][j]!=noData))||((sea==true)&&(this.rast
      i[j]!=0)&&(this.raster[i][j]!=noData)))
1253                {
1254                   offsetVal =
      (int)Math.floor(Math.abs(this.raster[i][j]-centrePoint)/classSize);
1255                   if(this.raster[i][j]<centrePoint)
1256                      raster[i][j] =
      centrePoint-((offsetVal*classSize)+(classSize/2));
1257                   else
1258                      raster[i][j] =
```



```
1258    centrePoint+((offsetVal*classSize)+(classSize/2));
1259    //             System.out.println("centreP: "+centrePoint+" offset:
      "+offsetVal+" classSize: "+classSize);
1260                }
1261       }
1262
1263       private int absToRelX(double x)
1264       {
1265          return (int)((x-this.xllCorner)/this.cellSize);
1266       }
1267
1268       private int absToRelY(double y)
1269       {
1270          return (int)((y-this.yllCorner)/this.cellSize);
1271       }
1272
1273       private double relToAbsX(int x)
1274       {
1275          return (x*this.cellSize)+this.xllCorner;
1276       }
1277
1278       private double relToAbsY(int y)
1279       {
1280          return (y*this.cellSize)+this.yllCorner;
1281       }
1282
1283       public Raster gridclip(Rectangle clipFP) throws Exception
1284       {
1285          float[][] clippedarray=new float[1][1];
1286          int xMin,xMax,yMin,yMax,newRows,newCols;
1287
1288          //check if the selected clip footprint is within the raster area
1289          //if it's not, simply return the original raster
1290          if(this.footprint.contains(clipFP)==false)
1291          {
1292             System.out.println("Clip footprint outside raster area = MER
      footprint used.");
1293    //       SimpleIO.readKeyboard("Press return to continue");
1294          }
1295          clipFP = this.footprint.minEnclRect(clipFP);
1296          //determine array minima and maxima from input coordinates
1297
1298          xMin = (int)((clipFP.getXll()-xllCorner)/cellSize);
1299          xMax = (int)(nCols-((xurCorner-clipFP.getXur())/cellSize));
1300
1301          yMin = (int)((clipFP.getYll()-yllCorner)/cellSize);
1302          yMax = (int)(nRows-((yurCorner-clipFP.getYur())/cellSize));
1303          System.out.println("clipping grid to - LL & UR: ("+xMin+","+yMin+"
      ("+xMax+","+yMax+")");
1304          newCols = xMax-xMin;
1305          newRows = yMax-yMin;
1306          clippedarray = new float[newCols][newRows];
1307          for (int i=0;i<newCols;i++)
1308             for(int j=0;j<newRows;j++)
1309                clippedarray[i][j] = raster[xMin+i][yMin+j];
1310          Raster newRaster = new Raster(clipFP,cellSize,noData,0);
1311          newRaster.replaceRaster(clippedarray);
1312    //
      this.replaceRaster(clippedarray,cellSize,newCols,newRows,clipFP.getXll()
      ipFP.getYll());
1313    //    newRaster.report();
1314          System.out.println("gridclip finished");
1315          return newRaster;
1316       }
1317
1318       public int[][] getIntArray()
1319       {
1320          int[][] intArr = new int[nCols][nRows];
1321          System.out.println("int array: "+nCols+" "+nRows);
1322          for (int i=0;i<nCols;i++)
1323             for(int j=0;j<nRows;j++)
```

```
1324                intArr[i][j] = (int)raster[i][j];
1325          return intArr;
1326       }
1327
1328       public void setDefaults()
1329       {
1330          scaleType = "none";
1331          if(getMinValNotNoData(noData,noData)==0)
1332             sea=true;
1333          else
1334             sea=false;
1335          directory = "C:/gisdata/";
1336          name = "trialRaster";
1337          asciiName = "trialRaster.asc";
1338       }
1339
1340       public void setParameters(String newDir,String newName,String
     newAsciiName,String newScaleType, boolean newSea)
1341       {
1342          directory = newDir;
1343          name = newName;
1344          asciiName = newAsciiName+".asc";
1345          scaleType = newScaleType;
1346          sea = newSea;
1347       }
1348
1349       public void setDirectory(String newDir)
1350       {
1351          directory = newDir;
1352       }
1353       public void setName(String newName)
1354       {
1355          name = newName;
1356       }
1357
1358       public String getDirectory()
1359       {
1360          return directory;
1361       }
1362
1363       public String getName()
1364       {
1365          return name;
1366       }
1367
1368       public void setAsciiName(String newAName)
1369       {
1370          asciiName = newAName;
1371       }
1372       public String getAsciiName()
1373       {
1374          return asciiName;
1375       }
1376
1377       public Raster costSurface(Point p,float costLimit,float gradientFacto
     throws Exception
1378       {
1379          Raster drainageR;
1380
     if(SimpleIO.checkFile(directory+"/"+asciiName.substring(0,asciiName.leng
     )-9)+"_filled_topo.asc")==true)
1381             drainageR = new
     Raster(directory+"/"+asciiName.substring(0,asciiName.length()-9)+"_fille
     opo.asc");
1382          float testValue,heightDiff;
1383          int a1,a2,b1,b2;
1384
1385          double pX = p.getX();
1386          double pY = p.getY();
1387
1388          int initX = (int)((pX-xllCorner)/cellSize);
```



```
1389          int initY = (int)((pY-yllCorner)/cellSize);
1390          int startX = (int)((pX-xllCorner)/cellSize);
1391          int startY = (int)((pY-yllCorner)/cellSize);
1392          int[][] checkArray = new int[nCols][nRows];
1393          //checkArray: 0=not checked;1=active;2=cleared;3=blocked
1394          checkArray[startX][startY]=1;
1395          float[][] costArray = new float[nCols][nRows];
1396          costArray[startX][startY] = -1;
1397    //    Vector checkVector = new Vector();
1398          boolean checkFlag=true;
1399          boolean initFlag=false;
1400
1401          //rule out the noData cells
1402          for(int x=0;x<nCols;x++)
1403             for(int y=0;y<nRows;y++)
1404                if(getValue(x,y)==noData)
1405                   checkArray[x][y]=5;
1406
1407          //check the first point
1408          for(int x=startX-1;x<=startX+1;x++)
1409             for(int y=startY-1;y<=startY+1;y++)
1410                if
     ((x>=1)&&(x<nCols-1)&&(y>=1)&&(y<nRows-1)&&(getValue(x,y)!=0))
1411                   if(((x!=startX)||(y!=startY))&&(checkArray[x][y]<4))
1412                   {
1413                      if(getValue(x,y)>=getValue(startX,startY))
1414                         testValue =
     1+((float)Math.tan(Math.toRadians(getGradient(x,y,false)))*gradientFacto
1415                      else
1416                         testValue = 1;
1417                      if(getValue(x,y)>=getValue(initX,initY))
1418                         heightDiff = getValue(x,y)-getValue(initX,initY);
1419                      else
1420                         heightDiff = 0;
1421                      costArray[x][y] =
     heightDiff+testValue*(float)p.distanceFrom(new
     Point(xllCorner+(cellSize*x)+(cellSize/2),yllCorner+(cellSize*y)+(cellSi
     2)));
1422                      if(costArray[x][y]>=costLimit)
1423                         checkArray[x][y]=3;
1424                      else
1425                      {
1426    //                   checkVector.addElement(new Point(x,y));
1427                         checkArray[x][y]=1;
1428                      }
1429                   }
1430          checkArray[startX][startY]=4;
1431          System.out.println("POINT: "+initX+" "+initY+" Val:
     "+getValue(initX,initY));
1432
1433          while(checkFlag==true)
1434          {
1435             startX=-1;
1436             startY=-1;
1437    //       int cx,cy;
1438             //loop to determine the nearest cell to the origin point for th
     calculation
1439
1440                float minCost= costLimit;
1441
1442                for(int i=1;i<nCols-1;i++)
1443                   for(int j=1;j<nRows-1;j++)
1444    //             for(int i=0;i<checkVector.size();i++)
1445    //             {
1446    //                cx=(int)((Point)checkVector.elementAt(i)).getX();
1447    //                cy=(int)((Point)checkVector.elementAt(i)).getY();
1448    //                if((costArray[cx][cy]<minCost)&&(costArray[cx][cy]>0))
1449
     if((checkArray[i][j]==1)&&(costArray[i][j]<minCost)&&(costArray[i][j]>0)
1450                      {
1451                         minCost = costArray[i][j];
```

```
1452                        startX=i;
1453                        startY=j;
1454                        checkFlag=true;
1455                     }
1456  //          }
1457           if(startX==-1)
1458              checkFlag=false;
1459
1460           if(checkFlag==true)
1461           {
1462              Point p2 = new Point(xllCorner+(cellSize*startX)+(cellSize/2
1463
      yllCorner+(cellSize*startY)+(cellSize/2));
1464              for(int x=startX-1;x<=startX+1;x++)
1465                 for(int y=startY-1;y<=startY+1;y++)
1466                    if
      ((x>=1)&&(x<nCols-1)&&(y>=1)&&(y<nRows-1)&&(getValue(x,y)!=0))
1467                       if(((x!=startX)||(y!=startY))&&(checkArray[x][y]<4)
1468                       {
1469                          //calculate the additional cost to move to that
1470                          //for 'downhill' cells, the value is the distanc
      otherwise multiply by 1+gradient
1471                          if(getValue(x,y)>=getValue(startX,startY))
1472                             testValue =
      1+((float)Math.tan(Math.toRadians(getGradient(x,y,false)))*gradientFacto
1473                          else
1474                             testValue = 1;
1475                          if(getValue(x,y)>getValue(initX,initY))
1476                             heightDiff = getValue(x,y)-getValue(initX,ini
1477                          else
1478                             heightDiff = 0;
1479                          testValue = costArray[startX][startY]+heightDiff
1480                             (testValue*(float)p2.distanceFrom(new
      Point(xllCorner+(cellSize*x)+(cellSize/2),yllCorner+(cellSize*y)+(cellSi
      2))));
1481
      if((testValue<costArray[x][y])&&(checkArray[x][y]==2)
1482                          &&(costArray[x][y]<costLimit))
1483                          {
1484  //                         checkVector.addElement(new Point(x,y));
1485                             checkArray[x][y]=1;
1486                          }
1487
      if((testValue<costArray[x][y])||(costArray[x][y]==0))
1488                          costArray[x][y] = testValue;
1489  //             System.out.println(x+" "+y+" "+getValue(x,y)+" dist:
      "+p2.distanceFrom(new Point(cellCentreX,cellCentreY))+" grad: "+testValu
1490
      if((costArray[x][y]>=costLimit)&&(checkArray[x][y]!=1))
1491                          checkArray[x][y]=3;
1492                       else if(checkArray[x][y]!=2)
1493                       {
1494  //                      checkVector.addElement(new Point(x,y));
1495                          checkArray[x][y]=1;
1496                       }
1497                    }
1498  //          for(int v=0;v<checkVector.size();v++)
1499  //             if(((int)((Point)checkVector.elementAt(v)).getX()==startX
1500  //                &&((int)((Point)checkVector.elementAt(v)).getY()==star
1501  //                   checkVector.removeElementAt(v);
1502              checkArray[startX][startY]=2;
1503           }
1504        }//end of while loop
1505  //    SimpleIO.readKeyboard();
1506  /*       for(int s=0;s<nRows;s++)
1507           {
1508              for(int t=0;t<nCols;t++)
1509                 System.out.print(checkArray[t][s]+" ");
1510              System.out.println();
1511           }*/
```



```
1512        for(int s=0;s<nRows;s++)
1513           for(int t=0;t<nCols;t++)
1514           {
1515              if(checkArray[t][s]==5)
1516                 costArray[t][s]=noData;
1517              else if(costArray[t][s]>costLimit)
1518                 costArray[t][s]=0;
1519           }
1520        return new Raster(this,costArray);
1521     }
1522
1523     public Rectangle nonZeroFootprint()
1524     {
1525        int minX = nCols;
1526        int maxX = 0;
1527        int minY = nRows;
1528        int maxY = 0;
1529        for(int i=0;i<this.nCols;i++)
1530           for(int j=0;j<this.nRows;j++)
1531           {
1532              if((getValue(i,j)!=0)&&(getValue(i,j)!=noData))
1533              {
1534                 if(i<minX)
1535                    minX=i;
1536                 if(j<minY)
1537                    minY=j;
1538                 if(i>maxX)
1539                    maxX=i;
1540                 if(j>maxY)
1541                    maxY=j;
1542              }
1543           }
1544        Point zll = new
      Point(xllCorner+(minX*cellSize),yllCorner+(minY*cellSize));
1545        Point zur = new
      Point(xllCorner+((maxX+1)*cellSize),yllCorner+((maxY+1)*cellSize));
1546        System.out.println("nonZeroFootprint params:");
1547        zll.displayValues();
1548        zur.displayValues();
1549        return new Rectangle(zll,zur);
1550     }
1551
1552     public Vector getSameHeightContCells(int startX,int startY)
1553     {
1554        Vector contigVector=new Vector();
1555        Vector newCellsList = new Vector();
1556        Point cell1,cell2;
1557        newCellsList.addElement(new Point(startX,startY));
1558        contigVector.addElement(new Point(startX,startY));
1559        while(newCellsList.size()>0)
1560        {
1561           cell1 = (Point)newCellsList.elementAt(0);
1562           int cvx = (int)cell1.getX();
1563           int cvy = (int)cell1.getY();
1564           for(int x=cvx-1;x<=cvx+1;x++)
1565              for(int y=cvy-1;y<=cvy+1;y++)
1566                 if
      (((x!=cvx)||(y!=cvy))&&(x>0)&&(x<nCols-1)&&(y>0)&&(y<nRows-1))
1567                 {
1568                    //check it's not one of the same points
1569                    boolean cvCheck=false;
1570                    for (int cv=0;cv<contigVector.size();cv++)
1571                    {
1572                       cell2 =(Point)contigVector.elementAt(cv);
1573                       if(((int)cell2.getX()==x)&&((int)cell2.getY()==y))
1574                          cvCheck=true;
1575                    }
1576                    if(cvCheck==false)
1577                    {
1578                       if(getValue(x,y)==getValue(cvx,cvy))
1579                       {
```

```
                        contigVector.addElement(new Point(x,y));
                        newCellsList.addElement(new Point(x,y));
                     }
                  }
               }
            newCellsList.removeElementAt(0);
         }
         return contigVector;
      }

      public Vector searchToBoundary(int startX,int startY,float
boundaryVal1,float boundaryVal2,boolean[][] check)
      {
         Vector contigVector=new Vector();
         Vector newCellsList = new Vector();
         Point cell1,cell2;
         contigVector.addElement(new Point(startX,startY));
         if((startX>=0)&&(startX<nCols)&&(startY>=0)&&(startY<nRows)

&&(getValue(startX,startY)!=boundaryVal1)&&(getValue(startX,startY)!=bou
ryVal2))
         {
            check[startX][startY]=true;
            newCellsList.addElement(new Point(startX,startY));
         }
         while(newCellsList.size()>0)
         {
            cell1 = (Point)newCellsList.elementAt(0);
            int cvx = (int)cell1.getX();
            int cvy = (int)cell1.getY();
            for(int x=cvx-1;x<=cvx+1;x++)
               for(int y=cvy-1;y<=cvy+1;y++)
                  if
(((x!=cvx)||(y!=cvy))&&(x>=0)&&(x<nCols)&&(y>=0)&&(y<nRows))
                  {
                     if(check[x][y]==false)
                     {

if((getValue(x,y)!=boundaryVal1)&&(getValue(x,y)!=boundaryVal2))
                           newCellsList.addElement(new Point(x,y));
                        contigVector.addElement(new Point(x,y));
                        check[x][y]=true;
                     }
                  }
            newCellsList.removeElementAt(0);
//          System.out.println("cells: "+newCellsList.size());
         }
         return contigVector;
      }

      //returns perimeter cells. If plusMinusHeight=1, returns all cells
      //plusMinusHeight=0: returns same elevation cells,
      //plusMinusHeight=-1: returns lower elevation cells
      //plusMinusHeight=-2: returns lower and same elevation cells
      public Vector getPerimeterCells(Vector contigVector,int plusMinusHeig
      {
         Vector perimeter = new Vector();
         boolean[][] perimCheck = new boolean[nCols][nRows];
         for (int cv=0;cv<contigVector.size();cv++)
         {
            int cvx = (int)((Point)contigVector.elementAt(cv)).getX();
            int cvy = (int)((Point)contigVector.elementAt(cv)).getY();
            perimCheck[cvx][cvy]=true;
         }
         float mainHeight = getValue((Point)contigVector.elementAt(0),false
         for (int cv=0;cv<contigVector.size();cv++)
         {
            int cvx = (int)((Point)contigVector.elementAt(cv)).getX();
            int cvy = (int)((Point)contigVector.elementAt(cv)).getY();
            for(int x=cvx-1;x<=cvx+1;x++)
               for(int y=cvy-1;y<=cvy+1;y++)
```



```
                  if
(((x!=cvx)||(y!=cvy))&&(x>=0)&&(x<nCols)&&(y>=0)&&(y<nRows))
                  {
                     //check it's not one of the same points
                     if(perimCheck[x][y]==false)

if(((plusMinusHeight==-1)&&(getValue(x,y)<mainHeight))||

((plusMinusHeight==-2)&&(getValue(x,y)<=mainHeight))||

((plusMinusHeight==0)&&(getValue(x,y)==mainHeight))||
                           (plusMinusHeight>0))
                        {
                           perimeter.addElement(new Point(x,y));
                           perimCheck[x][y]=true;
                        }
                  }
         }
         return perimeter;
      }

      public Point getLowJoiningCell(Vector contigVector,boolean
includeHorizontal)
      {
         Point lowP = new Point();
         float minSurroundHeight = Float.MAX_VALUE;
         for (int cv=0;cv<contigVector.size();cv++)
         {
            int cvx = (int)((Point)contigVector.elementAt(cv)).getX();
            int cvy = (int)((Point)contigVector.elementAt(cv)).getY();
            for(int x=cvx-1;x<=cvx+1;x++)
               for(int y=cvy-1;y<=cvy+1;y++)
                  if((x!=cvx)||(y!=cvy))
                  {
                     if(includeHorizontal==false)
                     {
            //only include horizontal cells if they are on the edge

if(((getValue(x,y)!=getValue(cvx,cvy))&&(getValue(x,y)<minSurroundHeight

||((getValue(x,y)==getValue(cvx,cvy))&&(getValue(x,y)<minSurroundHeight)
                           &&((x==0)||(y==0)||(x==nCols-1)||(y==nRows-1))))
                        {
                           lowP = new Point(x,y);
                           minSurroundHeight=getValue(x,y);
                        }
                     }
                     else
                     {
                        boolean usedCell=false;
                        for(int a=0;a<contigVector.size();a++)

if((((Point)contigVector.elementAt(a)).getX()==x)&&(((Point)contigVector
ementAt(a)).getY()==y))
                              usedCell=true;

if((getValue(x,y)<minSurroundHeight)&&(usedCell==false))
                        {
                           lowP = new Point(x,y);
                           minSurroundHeight=getValue(x,y);
                        }
                     }
                  }
         }
         return lowP;
      }

      public Point getLowJoiningCell(int loopIndex,int[][] parentX,int[][]
parentY,Vector pathPts)
      {
         Point lowP = new Point();
```

```
1706        int cvx = (int)((Point)pathPts.elementAt(0)).getX();
1707        int cvy = (int)((Point)pathPts.elementAt(0)).getY();
1708        float minSurroundHeight = getValue(cvx,cvy);
1709        if(loopIndex>=pathPts.size())
1710           loopIndex=pathPts.size()-1;
1711        for (int cv=0;cv<=loopIndex;cv++)
1712        {
1713           cvx = (int)((Point)pathPts.elementAt(cv)).getX();
1714           cvy = (int)((Point)pathPts.elementAt(cv)).getY();
1715           for(int x=cvx-1;x<=cvx+1;x++)
1716              for(int y=cvy-1;y<=cvy+1;y++)
1717                 if(((x!=cvx)||(y!=cvy))&&(getValue(x,y)<=minSurroundHeigh
1718                 {
1719                    boolean usedCell=false;
1720                    for(int a=0;a<pathPts.size();a++)
1721                    {
1722                       int px = (int)((Point)pathPts.elementAt(a)).getX();
1723                       int py = (int)((Point)pathPts.elementAt(a)).getY();
1724                       if((x==px)&&(y==py))
1725                          usedCell=true;
1726                       if((parentX[x][y]==px)&&(parentY[x][y]==py))
1727                          usedCell=true;
1728                    }
1729                    if(usedCell==false)
1730                    {
1731                       lowP = new Point(x,y);
1732                       minSurroundHeight=getValue(x,y);
1733                    }
1734                 }
1735        }
1736        return lowP;
1737     }
1738
1739     public Point getLowJoiningCell(Point testP,boolean includeHorizontal)
1740     {
1741        Point lowP=new Point(-9,-9);
1742        float minSurroundHeight = Float.MAX_VALUE;
1743        int cvx = (int)testP.getX();
1744        int cvy = (int)testP.getY();
1745        for(int x=cvx-1;x<=cvx+1;x++)
1746           for(int y=cvy-1;y<=cvy+1;y++)
1747              if((x!=cvx)||(y!=cvy))
1748              {
1749                 if(includeHorizontal==false)
1750                 {
1751        //only include horizontal cells if they are on the edge
1752
      if(((getValue(x,y)!=getValue(cvx,cvy))&&(getValue(x,y)<minSurroundHeight
1753
      ||((getValue(x,y)==getValue(cvx,cvy))&&(getValue(x,y)<minSurroundHeight)
1754                       &&((x==0)||(y==0)||(x==nCols-1)||(y==nRows-1))))
1755                    {
1756                       lowP = new Point(x,y);
1757                       minSurroundHeight=getValue(x,y);
1758                    }
1759                 }
1760                 else
1761                 {
1762                    if(getValue(x,y)<minSurroundHeight)
1763                    {
1764                       lowP = new Point(x,y);
1765                       minSurroundHeight=getValue(x,y);
1766                    }
1767                 }
1768              }
1769        return lowP;
1770     }
1771
1772     public void printIntVals()
1773     {
1774        int val;
```



```
1775        for(int j=this.nRows-1;j>=0;j--)
1776        {
1777           for(int i=0;i<this.nCols;i++)
1778           {
1779              val = (int)getValue(i,j);
1780              if(val<10)
1781                 System.out.print(val+"   ");
1782              else if(val<100)
1783                 System.out.print(val+"  ");
1784              else if(val<1000)
1785                 System.out.print(val+" ");
1786              else
1787                 System.out.print(val);
1788           }
1789           System.out.println();
1790        }
1791     }
1792
1793     public void printIntVals(int[][] array,int value)
1794     {
1795        int val;
1796           for(int j=this.nRows-1;j>=0;j--)
1797           {
1798              for(int i=0;i<this.nCols;i++)
1799              {
1800                 if(array[i][j] == value)
1801                 {
1802                    val = (int)getValue(i,j);
1803                    if(val<10)
1804                       System.out.print(val+"   ");
1805                    else if(val<100)
1806                       System.out.print(val+"  ");
1807                    else if(val<1000)
1808                       System.out.print(val+" ");
1809                    else
1810                       System.out.print(val);
1811                 }
1812                 else
1813                    System.out.print("    ");
1814              }
1815              System.out.println();
1816           }
1817     }
1818
1819     public void printIntVals(int[][] array)
1820     {
1821        for(int j=this.nRows-1;j>=0;j--)
1822        {
1823           for(int i=0;i<this.nCols;i++)
1824           {
1825                 if(array[i][j]<10)
1826                    System.out.print(array[i][j]+"   ");
1827                 else if(array[i][j]<100)
1828                    System.out.print(array[i][j]+"  ");
1829                 else if(array[i][j]<1000)
1830                    System.out.print(array[i][j]+" ");
1831                 else
1832                    System.out.print(array[i][j]);
1833           }
1834           System.out.println();
1835        }
1836     }
1837
1838     public void printIntVals(boolean[][] booleanArray,boolean trueOrFalse
1839     {
1840        int val;
1841        if(trueOrFalse==true)
1842           for(int j=this.nRows-1;j>=0;j--)
1843           {
1844              for(int i=0;i<this.nCols;i++)
1845              {
```

```
1846                if(booleanArray[i][j] == true)
1847                {
1848                   val = (int)getValue(i,j);
1849                   if(val<10)
1850                      System.out.print(val+"   ");
1851                   else if(val<100)
1852                      System.out.print(val+"  ");
1853                   else if(val<1000)
1854                      System.out.print(val+" ");
1855                   else
1856                      System.out.print(val);
1857                }
1858                else
1859                   System.out.print("    ");
1860             }
1861             System.out.println();
1862          }
1863       else
1864          for(int j=this.nRows-1;j>=0;j--)
1865          {
1866             for(int i=0;i<this.nCols;i++)
1867             {
1868                if(booleanArray[i][j] == false)
1869                {
1870                   val = (int)getValue(i,j);
1871                   if(val<10)
1872                      System.out.print(val+"   ");
1873                   else if(val<100)
1874                      System.out.print(val+"  ");
1875                   else if(val<1000)
1876                      System.out.print(val+" ");
1877                   else
1878                      System.out.print(val);
1879                }
1880                else
1881                   System.out.print("    ");
1882             }
1883             System.out.println();
1884          }
1885    }
1886
1887    public void printBooleanVals(boolean[][] booleanArray,boolean
      trueOrFalse)
1888    {
1889       int val;
1890       if(trueOrFalse==true)
1891          for(int j=this.nRows-1;j>=0;j--)
1892          {
1893             for(int i=0;i<this.nCols;i++)
1894             {
1895                if(booleanArray[i][j] == true)
1896                   System.out.print("T");
1897                else
1898                   System.out.print(" ");
1899             }
1900             System.out.println();
1901          }
1902       else
1903          for(int j=this.nRows-1;j>=0;j--)
1904          {
1905             for(int i=0;i<this.nCols;i++)
1906             {
1907                if(booleanArray[i][j] == false)
1908                   System.out.print("F");
1909                else
1910                   System.out.print(" ");
1911             }
1912             System.out.println();
1913          }
1914    }
1915
```



```
1916    public double calculateArea(int ignoreVal,boolean kmOrM)
1917    {
1918       double totalArea = 0;
1919       double cellArea = (cellSize*cellSize);
1920       if(kmOrM==true)
1921          cellArea = cellArea/1000000;
1922       for(int i=0;i<this.nCols;i++)
1923          for(int j=0;j<this.nRows;j++)
1924             if((getValue(i,j)!=noData)&&(getValue(i,j)!=ignoreVal))
1925                totalArea+=cellArea;
1926       return totalArea;
1927    }
1928
1929    public double calculateArea(int minVal,int maxVal,boolean kmOrM)
1930    {
1931       double totalArea = 0;
1932       double cellArea = (cellSize*cellSize);
1933       if(kmOrM==true)
1934          cellArea = cellArea/1000000;
1935       for(int i=0;i<this.nCols;i++)
1936          for(int j=0;j<this.nRows;j++)
1937
      if((getValue(i,j)!=noData)&&(getValue(i,j)>=minVal)&&(getValue(i,j)<=max
      ))
1938                totalArea+=cellArea;
1939       return totalArea;
1940    }
1941
1942    public double calculateArea(float[][] indicatorArray,float
      indicatorVal1,float indicatorVal2,boolean kmOrM)
1943    {
1944       double totalArea = 0;
1945       double cellArea = (cellSize*cellSize);
1946       if(kmOrM==true)
1947          cellArea = cellArea/1000000;
1948       for(int i=0;i<this.nCols;i++)
1949          for(int j=0;j<this.nRows;j++)
1950
      if((indicatorArray[i][j]==indicatorVal1)||(indicatorArray[i][j]==indicat
      al2))
1951                totalArea+=cellArea;
1952       return totalArea;
1953    }
1954
1955    public double calculateValueArea(int value,boolean kmOrM)
1956    {
1957       double totalArea = 0;
1958       double cellArea = (cellSize*cellSize);
1959       if(kmOrM==true)
1960          cellArea = cellArea/1000000;
1961       for(int i=0;i<this.nCols;i++)
1962          for(int j=0;j<this.nRows;j++)
1963             if(getValue(i,j)==value)
1964                totalArea+=cellArea;
1965       return totalArea;
1966    }
1967
1968    public double sumCellValues(boolean[][] indicatorArray)
1969    {
1970       double totalSum = 0;
1971       for(int i=0;i<this.nCols;i++)
1972          for(int j=0;j<this.nRows;j++)
1973             if(indicatorArray[i][j]==true)
1974                totalSum+=getValue(i,j);
1975       return totalSum;
1976    }
1977
1978    public void addLakes(String lakeSeedFile,Raster topoR,Raster editsR)
      throws Exception
1979    {
1980       lakeSeedFile = "C:/gisdata/config_files/"+lakeSeedFile+".csv";
```

```
1981        System.out.println(lakeSeedFile);
1982        Vector lakeSeedLines = SimpleIO.getLines(lakeSeedFile);
1983        int i=0;
1984        int xCell,yCell;
1985        while(i<lakeSeedLines.size())
1986        {
1987           double x =
    Double.parseDouble((String)SimpleIO.getLineElement(lakeSeedLines,i,0,","
1988           double y =
    Double.parseDouble((String)SimpleIO.getLineElement(lakeSeedLines,i,1,","
1989           xCell = (int)((x-xllCorner)/cellSize);
1990           yCell = (int)((y-yllCorner)/cellSize);
1991           if((xCell>=0)&&(xCell<nCols)&&(yCell>=0)&&(yCell<nRows))
1992           {
1993              String lakeName =
    (String)SimpleIO.getLineElement(lakeSeedLines,i,2,",");
1994              System.out.println("lake: "+lakeName);
1995              Vector lakeCells = new Vector();
1996              if("jokulsarlon".equals(lakeName))
1997                 lakeCells = editsR.searchToBoundary(xCell,yCell,20,15,new
    boolean[nCols][nRows]);
1998              else
1999                 lakeCells = topoR.getSameHeightContCells(xCell,yCell);
2000              for(int ls=0;ls<lakeCells.size();ls++)
2001              {
2002                 xCell = (int)(((Point)lakeCells.elementAt(ls)).getX());
2003                 yCell = (int)(((Point)lakeCells.elementAt(ls)).getY());
2004                 if((xCell>=0)&&(xCell<nCols)&&(yCell>=0)&&(yCell<nRows))
2005                    changeValue(xCell,yCell,-1);
2006              }
2007           }
2008           i++;
2009        }
2010
2011     }
2012
2013     public Raster getIceCells(String iceSeedFile,Raster topoR,float
    boundary1,float boundary2,boolean iceOrSandur) throws Exception
2014     {
2015        Raster iceR = new Raster(this,0);
2016        boolean[][] check = new boolean[nCols][nRows];
2017        for(int i=0;i<this.nCols;i++)
2018           for(int j=0;j<this.nRows;j++)
2019              if(topoR.getValue(i,j)==0)
2020                 check[i][j]=true;
2021        iceSeedFile = "C:/gisdata/config_files/"+iceSeedFile+".csv";
2022        Vector iceSeedLines = SimpleIO.getLines(iceSeedFile);
2023        int i=0;
2024        int xCell,yCell;
2025        while(i<iceSeedLines.size())
2026        {
2027           SimpleIO.getLineElement(iceSeedLines,i,0,",");
2028           double x =
    Double.parseDouble((String)SimpleIO.getLineElement(iceSeedLines,i,0,",")
2029           double y =
    Double.parseDouble((String)SimpleIO.getLineElement(iceSeedLines,i,1,",")
2030           xCell = (int)((x-xllCorner)/cellSize);
2031           yCell = (int)((y-yllCorner)/cellSize);
2032           if((xCell>=0)&&(xCell<nCols)&&(yCell>=0)&&(yCell<nRows))
2033           {
2034              String glacierName =
    (String)SimpleIO.getLineElement(iceSeedLines,i,2,",");
2035              System.out.println("glacier: "+glacierName);
2036              Vector iceCells =
    searchToBoundary(xCell,yCell,boundary1,boundary2,check);
2037              for(int is=0;is<iceCells.size();is++)
2038              {
2039                 xCell = (int)(((Point)iceCells.elementAt(is)).getX());
2040                 yCell = (int)(((Point)iceCells.elementAt(is)).getY());
2041                 if((xCell>=0)&&(xCell<nCols)&&(yCell>=0)&&(yCell<nRows))
2042                 {
```



```
2043                    if(iceOrSandur==true)
2044                       iceR.changeValue(xCell,yCell,220-(100*
    ((float)(Math.abs(topoR.getAspect(xCell,yCell,false))/45)-1)*((float)top
    getGradient(xCell,yCell,false)/75)));
2045                    else
2046                       iceR.changeValue(xCell,yCell,15+(((xCell%2)+(yCell%2))%2));
2047                    if(iceR.getValue(xCell,yCell)>255)
2048                       iceR.changeValue(xCell,yCell,255);
2049                 }
2050              }
2051           }
2052           i++;
2053        }
2054        for(i=0;i<this.nCols;i++)
2055           for(int j=0;j<this.nRows;j++)
2056              if((getValue(i,j)==boundary1)||(getValue(i,j)==boundary2))
2057              {
2058                 if(iceOrSandur==true)
2059                    iceR.changeValue(i,j,220-(100*
    ((float)(Math.abs(topoR.getAspect(i,j,false))/45)-1)*((float)topoR.getGr
    ent(i,j,false)/75)));
2060                 else
2061                    iceR.changeValue(i,j,15+(((i%2)+(j%2))%2));
2062                 if(iceR.getValue(i,j)>255)
2063                    iceR.changeValue(i,j,255);
2064              }
2065        //correcting for edge effects with ice cells
2066        if(iceOrSandur==true)
2067        {
2068           for(i=0;i<nCols;i++)
2069           {
2070              if(iceR.getValue(i,0)==255)
2071              {
2072                 float count=0;
2073                 int counter=0;
2074                 for(int c=i-1;c<i+2;c++)
2075                    if((c>=0)&&(c<nCols)&&(iceR.getValue(c,1)!=0)&&(iceR.getValue(c,1)!=255)
2076                    {
2077                       count+=iceR.getValue(c,1);
2078                       counter++;
2079                    }
2080                 if(counter>0)
2081                    iceR.changeValue(i,0,(float)(count/counter));
2082                 else
2083                    iceR.changeValue(i,0,0);
2084              }
2085              if(iceR.getValue(i,nRows-1)==255)
2086              {
2087                 float count=0;
2088                 int counter=0;
2089                 for(int c=i-1;c<i+2;c++)
2090                    if((c>=0)&&(c<nCols)&&(iceR.getValue(c,nRows-2)!=0)&&(iceR.getValue(c,nR
    -2)!=255))
2091                    {
2092                       count+=iceR.getValue(c,nRows-2);
2093                       counter++;
2094                    }
2095                 if(counter>0)
2096                    iceR.changeValue(i,nRows-1,(float)(count/counter));
2097                 else
2098                    iceR.changeValue(i,nRows-1,0);
2099              }
2100           }
2101           for(int j=1;j<nRows-1;j++)
2102           {
2103              if(iceR.getValue(0,j)==255)
2104              {
2105                 float count=0;
```

```
2106                {
2107                   float count=0;
2108                   int counter=0;
2109                   for(int c=j-1;c<j+2;c++)
2110                      if((iceR.getValue(1,c)!=0)&&(iceR.getValue(1,c)!=255))
2111                      {
2112                         count+=iceR.getValue(1,c);
2113                         counter++;
2114                      }
2115                   if(counter>0)
2116                      iceR.changeValue(0,j,(float)(count/counter));
2117                   else
2118                      iceR.changeValue(0,j,0);
2119                }
2120                if(iceR.getValue(nCols-1,j)==255)
2121                {
2122                   float count=0;
2123                   int counter=0;
2124                   for(int c=j-1;c<j+2;c++)
2125
     if((iceR.getValue(nCols-2,c)!=0)&&(iceR.getValue(nCols-2,c)!=255))
2126                      {
2127                         count+=iceR.getValue(nCols-2,c);
2128                         counter++;
2129                      }
2130                   if(counter>0)
2131                      iceR.changeValue(nCols-1,j,(float)(count/counter));
2132                   else
2133                      iceR.changeValue(nCols-1,j,0);
2134                }
2135             }
2136          }
2137          return iceR;
2138       }
2139
2140       //uses the iceR raster generated by getIceCells to produce the ice
     boundaries
2141       public Raster getIceEdges(String iceSeedFile,Raster topoR,float
     boundary1,float boundary2,boolean iceOrSandur) throws Exception
2142       {
2143          Raster
     iceCells=getIceCells(iceSeedFile,topoR,boundary1,boundary2,iceOrSandur);
2144          Raster iceEdges=new Raster(this,0);
2145          for(int i=0;i<nCols;i++)
2146             for(int j=0;j<nRows;j++)
2147             {
2148                if(iceCells.getValue(i,j)>0)
2149                   for(int x=i-1;x<=i+1;x++)
2150                      for(int y=j-1;y<=j+1;y++)
2151                         if((x>=0)&&(x<nCols)&&(y>=0)&&(y<nRows))
2152                            if(((x!=i)||(y!=j))&&(iceCells.getValue(x,y)==0)
2153                            {
2154                               iceEdges.changeValue(i,j,10);
2155                               x=i+2;y=j+2;
2156                            }
2157             }
2158          return iceEdges;
2159       }
2160
2161       public void removeSelectedCells(Raster selectR)
2162       {
2163          for(int i=0;i<this.nCols;i++)
2164             for(int j=0;j<this.nRows;j++)
2165                if(selectR.getValue(i,j)>0)
2166                   changeValue(i,j,noData);
2167       }
2168
2169       public Vector contours(float[] boundVals)
2170       {
2171          Vector contourV = new Vector();
2172          Point childPt,parentPt;
```



```
2173          Vector cIndex = new Vector();
2174          int coordCount=0;
2175          boolean boundCrossed;
2176          for(int i=0;i<nCols-1;i++)
2177             for(int j=0;j<nRows-1;j++)
2178             {
2179                if((getValue(i,j)!=noData)&&(getValue(i+1,j)!=noData))
2180                   for(int b=0;b<boundVals.length;b++)
2181
     if(((getValue(i,j)<boundVals[b])&&(getValue(i+1,j)>=boundVals[b]))
2182
     ||((getValue(i,j)>=boundVals[b])&&(getValue(i+1,j)<boundVals[b])))
2183                   {
2184                      b=boundVals.length;
2185                      childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+(j*cellSize));
2186                      parentPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+1)*cellSize));
2187                      contourV.addElement(childPt);
2188                      cIndex.addElement(new Integer(1));
2189                      contourV.addElement(parentPt);
2190                      cIndex.addElement(new Integer(2));
2191                   }
2192                if((getValue(i,j)!=noData)&&(getValue(i,j+1)!=noData))
2193                   for(int b=0;b<boundVals.length;b++)
2194
     if(((getValue(i,j)<boundVals[b])&&(getValue(i,j+1)>=boundVals[b]))
2195
     ||((getValue(i,j)>=boundVals[b])&&(getValue(i,j+1)<boundVals[b])))
2196                   {
2197                      b=boundVals.length;
2198                      childPt = new
     Point(xllCorner+(i*cellSize),yllCorner+((j+1)*cellSize));
2199                      parentPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+1)*cellSize));
2200                      contourV.addElement(childPt);
2201                      cIndex.addElement(new Integer(1));
2202                      contourV.addElement(parentPt);
2203                      cIndex.addElement(new Integer(2));
2204                   }
2205             }
2206          int[] coordIndex=new int[cIndex.size()];
2207          for(int i=0;i<cIndex.size();i++)
2208             coordIndex[i] = ((Integer)cIndex.elementAt(i)).intValue();
2209          contourV.addElement(coordIndex);
2210          return contourV;
2211       }
2212
2213       public static void main(String args[]) throws Exception
2214       {
2215          Raster test = new
     Raster("C:/gisdata/config_files/overall_edits_250.asc");
2216          test.reduceRes(4);
2217       //
     SimpleIO.rasterToAscii("C:/gisdata/config_files","overall_edits_cp",test
2218
2219
2220       //    Raster test2 = new Raster("C:/gisdata/overall_edits_cp.asc");
2221       //    test2.replaceRaster(test.getRaster());
2222       //    Raster test = new Raster("C:/gisdata/testfolder/testfolder_topo.as
2223       //    Raster test = new
     Raster("C:/gisdata/hi_myvatn_s/hi_myvatn_s_topo.asc");
2224       //    Raster test = new Raster("C:/gisdata/hofsjokull/hofsjokull_topo.as
2225       //    Raster test2 = new Raster(test,-9999);
2226       //    for (int i=0;i<test.getXcells();i++)
2227       //       for(int j=0;j<test.getYcells();j++)
2228       //          test2.changeValue(i,j,(float)test.getAspect(i,j,false));
2229       //    test2.printRasterData();
2230       //    Raster test = new Raster("C:/gisdata/jan_m.asc",300);
2231       //    test.increaseRes(250);
2232       //    test.report();
```

```
2233  //    test.printRasterData();
2234  //    Raster test = new Raster("C:/gisdata/testascii.asc");
2235  //    test.printRasterData();
2236  //    System.out.println();
2237
2238  //    test = new Raster("C:/gisdata/testascii.asc",291000,293000,"none")
2239  //    test.printRasterData();
2240  //    Point p = new Point(612500,519400);
2241  //    Raster cost = test.costSurface(p,40000,200);
2242  //    test.fillHollows();
2243  //    test.drainageDensity(100);
2244
2245     }
2246  }
2247
2248  /*
2249   *spare code - Contours...
2250   *
2251   *    for(int i=1;i<nCols-1;i++)
2252          for(int j=1;j<nRows-1;j++)
2253          {
2254             if((getValue(i,j)!=noData)&&(getValue(i+1,j)!=noData))
2255                for(int b=0;b<boundVals.length;b++)
2256                {
2257
     if((getValue(i,j)<boundVals[b])&&(getValue(i+1,j)>=boundVals[b]))
2258                   {
2259                      if(getValue(i+1,j+1)<boundVals[b])
2260                      {
2261                         childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2262                         parentPt = new
     Point(xllCorner+((i+1.5)*cellSize),yllCorner+((j+1)*cellSize));
2263                      }
2264                      else if(getValue(i,j+1)<boundVals[b])
2265                      {
2266                         childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2267                         parentPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+1.5)*cellSize));
2268                      }
2269                      else
2270                      {
2271                         childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2272                         parentPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2273                      }
2274                      contourV.addElement(childPt);
2275                      cIndex.addElement(new Integer(1));
2276                      contourV.addElement(parentPt);
2277                      cIndex.addElement(new Integer(2));
2278
2279                      if(getValue(i+1,j-1)<boundVals[b])
2280                      {
2281                         childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2282                         parentPt = new
     Point(xllCorner+((i+1.5)*cellSize),yllCorner+(j*cellSize));
2283                      }
2284                      else if(getValue(i,j-1)<boundVals[b])
2285                      {
2286                         childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2287                         parentPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j-0.5)*cellSize));
2288                      }
2289                      else
2290                      {
2291                         childPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
```



```
2292                         parentPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+(j*cellSize));
2293                      }
2294                      contourV.addElement(childPt);
2295                      cIndex.addElement(new Integer(1));
2296                      contourV.addElement(parentPt);
2297                      cIndex.addElement(new Integer(2));
2298                      b=boundVals.length;
2299                   }
2300                }
2301             if((getValue(i,j)!=noData)&&(getValue(i,j+1)!=noData))
2302                for(int b=0;b<boundVals.length;b++)
2303                {
2304
     if((getValue(i,j)<boundVals[b])&&(getValue(i,j+1)>=boundVals[b]))
2305                   {
2306                      if(getValue(i+1,j)<boundVals[b])
2307                      {
2308                         childPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2309                         parentPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2310                      }
2311                      else if(getValue(i+1,j+1)<boundVals[b])
2312                      {
2313                         childPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2314                         parentPt = new
     Point(xllCorner+((i+1.5)*cellSize),yllCorner+((j+1)*cellSize));
2315                      }
2316                      else
2317                      {
2318                         childPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2319                         parentPt = new
     Point(xllCorner+((i+1)*cellSize),yllCorner+((j+1.5)*cellSize));
2320                      }
2321                      contourV.addElement(childPt);
2322                      cIndex.addElement(new Integer(1));
2323                      contourV.addElement(parentPt);
2324                      cIndex.addElement(new Integer(2));
2325
2326                      if(getValue(i-1,j)<boundVals[b])
2327                      {
2328                         childPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2329                         parentPt = new
     Point(xllCorner+(i*cellSize),yllCorner+((j+0.5)*cellSize));
2330                      }
2331                      else if(getValue(i-1,j+1)<boundVals[b])
2332                      {
2333                         childPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2334                         parentPt = new
     Point(xllCorner+((i-0.5)*cellSize),yllCorner+((j+1)*cellSize));
2335                      }
2336                      else
2337                      {
2338                         childPt = new
     Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2339                         parentPt = new
     Point(xllCorner+(i*cellSize),yllCorner+((j+1.5)*cellSize));
2340                      }
2341                      contourV.addElement(childPt);
2342                      cIndex.addElement(new Integer(1));
2343                      contourV.addElement(parentPt);
2344                      cIndex.addElement(new Integer(2));
2345                      b=boundVals.length;
2346                   }
2347                }
2348             if((getValue(i,j)!=noData)&&(getValue(i+1,j)!=noData))
```

```
2349                    for(int b=0;b<boundVals.length;b++)
2350                    {
2351
    if((getValue(i,j)>=boundVals[b])&&(getValue(i+1,j)<boundVals[b]))
2352                        {
2353                            if(getValue(i,j+1)<boundVals[b])
2354                            {
2355                                childPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2356                                parentPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2357                                contourV.addElement(childPt);
2358                                cIndex.addElement(new Integer(1));
2359                                contourV.addElement(parentPt);
2360                                cIndex.addElement(new Integer(2));
2361                            }
2362                            else if(getValue(i+1,j+1)<boundVals[b])
2363                            {
2364                                childPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2365                                parentPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+1.5)*cellSize));
2366                                contourV.addElement(childPt);
2367                                cIndex.addElement(new Integer(1));
2368                                contourV.addElement(parentPt);
2369                                cIndex.addElement(new Integer(2));
2370                            }
2371
2372                            if(getValue(i+1,j-1)<boundVals[b])
2373                            {
2374                                childPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2375                                parentPt = new
    Point(xllCorner+((i+1.5)*cellSize),yllCorner+(j*cellSize));
2376                            }
2377                            else if(getValue(i,j-1)<boundVals[b])
2378                            {
2379                                childPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2380                                parentPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j-0.5)*cellSize));
2381                            }
2382                            else
2383                            {
2384                                childPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2385                                parentPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+(j*cellSize));
2386                            }
2387                            contourV.addElement(childPt);
2388                            cIndex.addElement(new Integer(1));
2389                            contourV.addElement(parentPt);
2390                            cIndex.addElement(new Integer(2));
2391                            b=boundVals.length;
2392                        }
2393                    }
2394                if((getValue(i,j)!=noData)&&(getValue(i,j+1)!=noData))
2395                    for(int b=0;b<boundVals.length;b++)
2396                    {
2397
    if((getValue(i,j)<boundVals[b])&&(getValue(i,j+1)>=boundVals[b]))
2398                        {
2399                            if(getValue(i+1,j)<boundVals[b])
2400                            {
2401                                childPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2402                                parentPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+0.5)*cellSize));
2403                            }
2404                            else if(getValue(i+1,j+1)<boundVals[b])
2405                            {
```



```
2406                                childPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2407                                parentPt = new
    Point(xllCorner+((i+1.5)*cellSize),yllCorner+((j+1)*cellSize));
2408                            }
2409                            else
2410                            {
2411                                childPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2412                                parentPt = new
    Point(xllCorner+((i+1)*cellSize),yllCorner+((j+1.5)*cellSize));
2413                            }
2414                            contourV.addElement(childPt);
2415                            cIndex.addElement(new Integer(1));
2416                            contourV.addElement(parentPt);
2417                            cIndex.addElement(new Integer(2));
2418
2419                            if(getValue(i-1,j)<boundVals[b])
2420                            {
2421                                childPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2422                                parentPt = new
    Point(xllCorner+(i*cellSize),yllCorner+((j+0.5)*cellSize));
2423                            }
2424                            else if(getValue(i-1,j+1)<boundVals[b])
2425                            {
2426                                childPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2427                                parentPt = new
    Point(xllCorner+((i-0.5)*cellSize),yllCorner+((j+1)*cellSize));
2428                            }
2429                            else
2430                            {
2431                                childPt = new
    Point(xllCorner+((i+0.5)*cellSize),yllCorner+((j+1)*cellSize));
2432                                parentPt = new
    Point(xllCorner+(i*cellSize),yllCorner+((j+1.5)*cellSize));
2433                            }
2434                            contourV.addElement(childPt);
2435                            cIndex.addElement(new Integer(1));
2436                            contourV.addElement(parentPt);
2437                            cIndex.addElement(new Integer(2));
2438                            b=boundVals.length;
2439                        }
2440                    }
2441    */
```

```
1    /*Class for selection of scale and colour scheme for display of a Raster,
2     *created 31/3/05, modified 7th April 2005*/
3    package afc.utilities;
4
5    import java.awt.*;
6    import java.util.*;
7
8    public class RasterScalePicker
9    {
10      private String scaleType;
11      private int noData,classes;
12      private int[] classRed,classGreen,classBlue;
13      private float minScale;
14      private float maxScale;
15      private float centrePoint;
16      private boolean blackAndWhite;
17
18
19      public RasterScalePicker(String scaleType,int noData,int classes,boolea
    blackAndWhite)
20      {
21         this.blackAndWhite =blackAndWhite;
22         this.scaleType= scaleType;
23         this.noData = noData;
24         this.classes = classes;
25         selectScaleType(scaleType);
26         if(scaleType=="thiessens")
27         {
28            Random ran = new Random();
29            classRed = new int[classes+1];
30            classGreen = new int[classes+1];
31            classBlue = new int[classes+1];
32            for(int i=0;i<=classes;i++)
33            {
34               double d1=-1;
35               double d2=-1;
36               double d3=-1;
37               while((d1<0)||(d2<0)||(d3<0))
38               {
39                  d1 = ran.nextDouble()*254;
40                  d2 = ran.nextDouble()*254;
41                  d3 = ran.nextDouble()*254;
42               }
43               classRed[i] = (int)d1;
44               classGreen[i] = (int)d2;
45               classBlue[i] = (int)d3;
46   //          System.out.println("class: "+i+".
    "+classRed[i]+","+classGreen[i]+","+classBlue[i]);
47            }
48   //       System.out.println("created classes");
49         }
50      }
51
52      public Color calculateColour(int greyLevel)
53      {
54         if(greyLevel==noData)
55            return new Color(0,0,0);
56   //       return new Color(150,0,100);
57         else
58         {
59            if((scaleType=="iceland_topo")&&(blackAndWhite==false))
60            {
61               if(greyLevel==-1)
62                  return new Color(100,100,100);
63               if(greyLevel==0)
64                  return new Color(0,0,200);
65               else if(greyLevel<85)//up to 0-170,255-128,0
66                  return new Color(greyLevel*2,255-greyLevel,0);
67               else if(greyLevel<170)//170,170,0-255
68                  return new Color(170,170,(greyLevel-85)*3);
69               else//170-255,170-255,255
```



```
70                   return new Color(greyLevel,greyLevel,255);
71            }
72            else if(scaleType=="iceland_topo2")//greyed-out topo
73               return new
    Color((int)((float)greyLevel/3)+100,(int)((float)greyLevel/3)+100,(int)(
    oat)greyLevel/3)+100);
74
75            else if(scaleType=="iceland_topo3")//v dark topo for overlays
76            {
77               if(greyLevel==0)
78                  return new Color(0,0,0);
79               else
80                  return new
    Color((int)(30+(float)greyLevel/2.3f),(int)(30+(float)greyLevel/2.3f),(i
    (30+(float)greyLevel/2.3f));
81            }
82
83            else
    if((scaleType=="iceland_topo4")||((scaleType=="iceland_topo")&&(blackAndW
    te==true)))//darkened topo for overlays
84            {
85               if(greyLevel==0)
86                  return new Color(0,0,0);
87               else
88                  return new
    Color((int)(90+(float)greyLevel/2.3f),(int)(90+(float)greyLevel/2.3f),(i
    (90+(float)greyLevel/2.3f));
89            }
90
91            else
    if((blackAndWhite==false)&&((scaleType=="temp")||(scaleType=="temp_ann")
92            ||(scaleType=="viewshed_month")||(scaleType=="viewshed_ann")))
93            {
94               if(greyLevel<128)
95                  return new Color(greyLevel*2,greyLevel*2,255);
96               else
97                  return new Color(255,(255-greyLevel)*2,(255-greyLevel)*2)
98            }
99            else if((blackAndWhite==false)&&(scaleType=="overlay"))
100           {
101              if(greyLevel<128)
102                 return new Color(50+greyLevel,255,50+greyLevel);
103  //            else if(greyLevel<132)
104  //               return new Color(100,0,0);
105              else
106                 return new Color(315-greyLevel,315-greyLevel,255);
107           }
108
109           else if((blackAndWhite==false)&&(scaleType=="ela"))
110           {
111              if(greyLevel<128)
112                 return new Color(greyLevel*2,greyLevel*2,170);
113  //             return new
    Color(greyLevel*2,170+(int)(greyLevel*0.666f),(int)(greyLevel*1.5));
114              else
115  //                return new
    Color(255-((greyLevel-128)*2),255-((greyLevel-128)*2),170);
116                return new
    Color(255-((greyLevel-128)*2),255-(int)((greyLevel-128)*0.666f),170-(int
    greyLevel-128)*1.333));
117           }
118
119           else
    if((blackAndWhite==false)&&((scaleType=="snow")||(scaleType=="precip")||
    aleType=="precip_month")))
120           {
121              if(greyLevel==0)
122                 return new Color(0,0,0,0);
123              else if(greyLevel<85)
124                 return new Color(170+greyLevel,170+greyLevel,greyLevel*3)
125              else
```

```
126                 return new
    Color(255-(int)((greyLevel-85)*1.5),255-(int)((greyLevel-85)*1.5),255);
127             }
128
129             else if(scaleType=="snowcover")
130             {
131                 if(greyLevel==0)
132                     return new Color(0,0,0,0);
133                 else
134                     return new Color(255-greyLevel,255-greyLevel,255);
135             }
136
137             else if(scaleType=="climseq")
138             {
139                 if(greyLevel==0)
140                     return new Color(0,0,0,0);
141                 else if(greyLevel==-1)
142                     return new Color(0,255,0);
143                 else if(greyLevel==-2)
144                     return new Color(0,128,0);
145                 else if(greyLevel==-3)
146                     return new Color(0,0,255);
147                 else
148                     return new Color(255-greyLevel,255-greyLevel,255);
149             }
150
151             else
    if((blackAndWhite==false)&&((scaleType=="pdd_month")||(scaleType=="pdd_a
    )))
152             {
153                 if(greyLevel==0)
154                     return new Color(0,0,0,0);
155                 else
156                     return new Color(greyLevel,0,0);
157             }
158             else if(scaleType=="pdd_veg")
159             {
160                 if(greyLevel==0)
161                     return new Color(0,0,0,0);
162                 else
163                 {
164                     if(blackAndWhite==true)
165                         return new
    Color(220-(int)(greyLevel/4),220-(int)(greyLevel/4),220-(int)(greyLevel/
166                     else
167                         return new
    Color(255-greyLevel,255-(int)(greyLevel/3),255-greyLevel);
168                 }
169
170             }
171             else if(scaleType=="2nd PDD")
172             {
173                 if(greyLevel==0)
174                     return new Color(0,0,0,0);
175                 else
176                 {
177                     if(blackAndWhite==true)
178                         return new
    Color(190-(int)(greyLevel/4),190-(int)(greyLevel/4),190-(int)(greyLevel/
179                     else
180                         return new
    Color(255-greyLevel,255-(int)(greyLevel/2),255-(int)(greyLevel/3));
181                 }
182             }
183             else if(scaleType=="pdd_veg_month")
184             {
185                 if(greyLevel==0)
186                     return new Color(0,0,0,0);
187                 else
188                     return new
    Color(255-greyLevel,255-(int)(greyLevel/3),255-greyLevel);
```



```
189             }
190             else if((blackAndWhite==true)&&(scaleType=="gs_length"))
191             {
192                 if(greyLevel==0)
193                     return new Color(0,0,0,0);
194                 else
195                     return new
    Color(85+(int)(greyLevel/1.5f),85+(int)(greyLevel/1.5f),85+(int)(greyLev
    1.5f));
196             }
197             else if((blackAndWhite==false)&&(scaleType=="gs_length"))
198             {
199                 if(greyLevel==0)
200                     return new Color(0,0,0,0);
201                 else if(greyLevel<85)
202                     return new Color(255,greyLevel*3,0);
203                 else if(greyLevel<170)
204                     return new Color(255-((greyLevel-85)*3),255,0);
205                 else
206                     return new
    Color(0,255-((greyLevel-170)*3),(greyLevel-170)*3);
207             }
208             else if((blackAndWhite==false)&&(scaleType=="gs_date"))
209             {
210                 if(greyLevel==0)
211                     return new Color(0,0,0,0);
212                 else if(greyLevel<21)
213                     return new Color(0,0,100);
214                 else if(greyLevel<41)
215                     return new Color(0,0,170);
216                 else if(greyLevel<62)
217                     return new Color(0,0,255);
218                 else if(greyLevel<83)
219                     return new Color(0,255,255);
220                 else if(greyLevel<105)
221                     return new Color(0,255,0);
222                 else if(greyLevel<126)
223                     return new Color(0,128,0);
224                 else if(greyLevel<148)
225                     return new Color(170,128,0);
226                 else if(greyLevel<170)
227                     return new Color(170,128,255);
228                 else if(greyLevel<191)
229                     return new Color(255,255,255);
230                 else if(greyLevel<213)
231                     return new Color(255,255,0);
232                 else if(greyLevel<234)
233                     return new Color(255,170,0);
234                 else
235                     return new Color(255,0,0);
236             }
237
238             else if((blackAndWhite==true)&&(scaleType=="gs_date"))
239             {
240                 if(greyLevel==0)
241                     return new Color(0,0,0,0);
242                 else if(greyLevel<21)
243                     return new Color(20,20,20);
244                 else if(greyLevel<41)
245                     return new Color(40,40,40);
246                 else if(greyLevel<62)
247                     return new Color(60,60,60);
248                 else if(greyLevel<83)
249                     return new Color(80,80,80);
250                 else if(greyLevel<105)
251                     return new Color(100,100,100);
252                 else if(greyLevel<126)
253                     return new Color(120,120,120);
254                 else if(greyLevel<148)
255                     return new Color(140,140,140);
256                 else if(greyLevel<170)
```

```
257                 return new Color(160,160,160);
258              else if(greyLevel<191)
259                 return new Color(180,180,180);
260              else if(greyLevel<213)
261                 return new Color(200,200,200);
262              else if(greyLevel<234)
263                 return new Color(220,220,220);
264              else
265                 return new Color(240,240,240);
266           }
267
268           else if((blackAndWhite==false)&&(scaleType=="gs_temp"))
269           {
270              if(greyLevel==0)
271                 return new Color(0,0,0,0);
272              if(greyLevel<128)
273                 return new Color(greyLevel*2,greyLevel*2,255);
274              else
275                 return new Color(255,(255-greyLevel)*2,(255-greyLevel)*2)
276           }
277
278           else if((blackAndWhite==false)&&(scaleType=="massb"))
279           {
280              if(greyLevel<170)
281                 return new Color(0,0,0,0);
282              else
283                 return new
    Color((int)((float)(255-greyLevel)*3f),(int)((float)(255-greyLevel)*3f),
    );
284           }
285
286           else if((blackAndWhite==true)&&(scaleType=="massb"))
287           {
288              if(greyLevel<170)
289                 return new Color(0,0,0,0);
290              else
291                 return new
    Color((int)((float)(297-greyLevel)*2f),(int)((float)(297-greyLevel)*2f),
    t)((float)(297-greyLevel)*2f));
292           }
293
294           else if((blackAndWhite==false)&&(scaleType=="elev_massb"))
295           {
296              if(greyLevel==-1)
297                 return new Color(100,100,100);
298              if(greyLevel==0)
299                 return new Color(0,0,200);
300              else if(greyLevel<85)//up to 0-170,255-128,0
301                 return new Color(greyLevel*2,255-greyLevel,0);
302              else if(greyLevel<134)//170,170,0-255
303                 return new Color(170,170,(greyLevel-85)*3);
304              else
305                 return new
    Color((int)((float)(255-greyLevel)*2f),(int)((float)(255-greyLevel)*2f),
    );
306           }
307
308           if((blackAndWhite==false)&&(scaleType=="topo-ela"))
309           {
310              if(greyLevel>=170)//up to 0-170,255-128,0
311                 return new Color((255-greyLevel)*2,greyLevel,0);
312              else if(greyLevel>=85)//170,170,0-255
313                 return new Color(170,170,-(greyLevel-170)*3);
314              else//170-255,170-255,255
315                 return new Color(255-greyLevel,255-greyLevel,255);
316           }
317
318           else if((blackAndWhite==false)&&(scaleType=="cost"))
319           {
320              if(greyLevel==-1)
321                 return new Color(255,0,0);
```



```
322              else if(greyLevel==0)
323                 return new Color(0,0,0,0);
324              else
325                 return new
    Color(255-(int)(greyLevel/2),255-(int)(greyLevel/2),0,130);
326           }
327           else if((blackAndWhite==false)&&(scaleType=="thiessens"))
328           {
329  //            System.out.println("##"+classes+"## "+greyLevel+":
    "+classRed[greyLevel]+" "+classGreen[greyLevel]+" "+classBlue[greyLevel]
330              if(greyLevel==0)
331                 return new Color(0,0,170,0);
332              else if(greyLevel<classes)
333                 return new
    Color(classRed[greyLevel],classGreen[greyLevel],classBlue[greyLevel],90)
334              else
335                 return new Color(180,180,180);
336  //            return new Color(greyLevel,greyLevel,greyLevel);
337           }
338           else if((blackAndWhite==false)&&(scaleType=="drainage"))
339           {
340              if(greyLevel==0)
341                 return new Color(0,0,0,0);
342              else if(greyLevel==-1)
343                 return new Color(0,0,255);
344              else
345                 return new Color(0,0,85+(int)((float)greyLevel*0.666f));
346           }
347           else if((blackAndWhite==true)&&(scaleType=="drainage"))
348           {
349              if(greyLevel==0)
350                 return new Color(0,0,0,0);
351              else if(greyLevel==-1)
352                 return new Color(50,50,50);
353              else
354                 return new
    Color(50+(int)((float)greyLevel*0.1f),50+(int)((float)greyLevel*0.1f),50
    nt)((float)greyLevel*0.1f));
355           }
356           else if(scaleType=="edits")
357           {
358              if(greyLevel==-1)
359                 return new Color(100,100,100);
360              else if((greyLevel==0)||(greyLevel==20))
361                 return new Color(0,0,0,0);
362              else if(greyLevel==4)
363                 return new Color(0,0,255);
364              else if(greyLevel==14)
365                 return new Color(255,255,0);
366              else if(greyLevel>127)//170-255,170-255,255
367                 return new Color(0,0,85+(greyLevel-127));
368              else
369                 return new Color(255-greyLevel,0,0);
370           }
371           else if((blackAndWhite==true)&&(scaleType=="ice"))
372           {
373              if(greyLevel<5)
374                 return new Color(0,0,0,0);
375              else if(greyLevel==9)
376                 return new Color(0,0,0);
377              else
378                 return new Color(255,255,255);
379           }
380           else if((blackAndWhite==false)&&(scaleType=="ice"))
381           {
382              if(greyLevel<5)
383                 return new Color(0,0,0,0);
384              else if(greyLevel==9)
385                 return new Color(0,0,0);
386              else
387                 return new Color(greyLevel,greyLevel,greyLevel);
```

```
388             }
389             else if(scaleType=="sandar")
390             {
391                if(greyLevel<5)
392                   return new Color(0,0,0,0);
393                else if(greyLevel==14)
394                   return new Color(110,110,110);
395                else if(greyLevel==15)
396                   return new Color(120,120,120);
397                else if(greyLevel==16)
398                   return new Color(130,130,130);
399                else
400                   return new Color(greyLevel,greyLevel,greyLevel);
401             }
402             else if(scaleType=="sensitivity")
403             {
404                if(greyLevel==0)
405                   return new Color(0,0,0,0);
406                else if(greyLevel<5)
407                {
408                   greyLevel = (int)((greyLevel-1)*42.5);
409                   if(greyLevel<85)
410                      return new Color(255,greyLevel*3,0);
411                   else if(greyLevel<170)
412                      return new Color(255-((greyLevel-85)*3),255,0);
413                   else
414                      return new
    Color(0,255-((greyLevel-170)*3),(greyLevel-170)*3);
415                }
416                else if(greyLevel==5)
417                   return new Color(0,85,170);
418                else
419                  return new Color(0,0,255);
420             }
421             else
422             {
423                if((greyLevel==0)&!((scaleType=="temp")||(scaleType=="temp_a
424                   ||(scaleType=="viewshed_month")||(scaleType=="viewshed_an
425                   ||(scaleType=="offset")||(scaleType=="ela")))
426                   return new Color(0,0,0,0);
427                else
428                   return new Color(greyLevel,greyLevel,greyLevel);
429             }
430          }
431       }
432
433       public void selectScaleType(String scaleType)
434       {
435    /*    if(scaleType=="iceland_topo")
436          {
437             minScale = 0;
438             centrePoint=50;
439             maxScale = 100;
440          }*/
441
    if((scaleType=="iceland_topo")||(scaleType=="iceland_topo2")||(scaleType
    iceland_topo3")||(scaleType=="iceland_topo4")||(scaleType=="elev_massb")
442          {
443             minScale = 0;
444             centrePoint=1054;
445             maxScale = 2109;
446          }
447          else if((scaleType=="topo-ela"))
448          {
449             minScale = -1940;
450             centrePoint=0;
451             maxScale = 1940;
452          }
453
454          else if(scaleType=="ela")
455          {
```



```
456             minScale = 600;
457             centrePoint=1400;
458             maxScale = 2200;
459          }
460          else if(scaleType=="temp")
461          {
462             minScale = -15;
463             centrePoint=0;
464             maxScale = 15;
465          }
466          else if((scaleType=="temp_ann")||(scaleType=="gs_temp"))
467          {
468             minScale = -11;
469             centrePoint=0;
470             maxScale = 11;
471          }
472          else if(scaleType=="offset")
473          {
474             minScale = -21;
475             centrePoint=0;
476             maxScale = 21;
477          }
478
479          else if((scaleType=="snow")||(scaleType=="snowcover")
480             ||(scaleType=="precip"))
481          {
482             minScale = 0;
483             centrePoint=0;
484             maxScale = -9999;
485          }
486          else if(scaleType=="climseq")
487          {
488             minScale = 0;
489             centrePoint=0;
490             maxScale = 7000;
491          }
492          else if(scaleType=="precip_month")
493          {
494             minScale = 0;
495             centrePoint=0;
496             maxScale = 900;
497          }
498
499          else if(scaleType=="pdd_month")
500          {
501             minScale = 0;
502             centrePoint=1250;
503             maxScale = 2500;
504          }
505          else if(scaleType=="pdd_ann")
506          {
507             minScale = 0;
508             centrePoint=7500;
509             maxScale = 15000;
510          }
511          else if((scaleType=="pdd_veg")||(scaleType=="2nd PDD"))
512          {
513             minScale = 0;
514             centrePoint=0;
515             maxScale = 2100;
516          }
517          else if(scaleType=="pdd_veg_month")
518          {
519             minScale = 0;
520             centrePoint=0;
521             maxScale = 400;
522          }
523          else if(scaleType=="gs_length")
524          {
525             minScale = 0;
526             centrePoint=182;
```

```
527             maxScale = 365;
528         }
529         else if(scaleType=="gs_date")
530         {
531             minScale = 0;
532             centrePoint=182;
533             maxScale = 365;
534         }
535         else if(scaleType=="viewshed_month")
536         {
537             minScale = 0;
538             centrePoint=0.065f;
539             maxScale = 0.13f;
540         }
541         else if(scaleType=="viewshed_ann")
542         {
543             minScale = 0;
544             centrePoint=0.5f;
545             maxScale = 1;
546         }
547         else if(scaleType=="lapse")
548         {
549             minScale = -0.009f;
550             centrePoint=-0.0075f;
551             maxScale = -0.006f;
552         }
553         else if((scaleType=="massb")||(scaleType=="overlay"))
554         {
555             minScale = -15000;
556             centrePoint=0;
557             maxScale = 7500;
558         }
559         else if(scaleType=="drainage")
560         {
561             minScale = 0;
562             centrePoint=5000;
563             maxScale = 9999;
564         }
565         else if(scaleType=="edits")
566         {
567             minScale = 0;
568             centrePoint=127;
569             maxScale = 255;
570         }
571         else if((scaleType=="ice")||(scaleType=="sandar"))
572         {
573             minScale = 0;
574             centrePoint=128;
575             maxScale = 255;
576         }
577         else
578         {
579             minScale = 0;
580             centrePoint=127;
581             maxScale = 255;
582         }
583     }
584
585     public float getMinScale()
586     {
587         return this.minScale;
588     }
589
590     public float getMaxScale()
591     {
592         return this.maxScale;
593     }
594
595     public String getScaleType()
596     {
597         return this.scaleType;
```



```
598     }
599
600     public float getCentrePoint()
601     {
602         return this.centrePoint;
603     }
604
605     public void setMinScale(float min)
606     {
607         this.minScale = min;
608     }
609
610     public void setMaxScale(float max)
611     {
612         this.maxScale = max;
613     }
614 }
```

```
1   /*Display class for drawing a Raster onto a JPanel. Based on Jo Wood's
    version,
2    *but modified and extended by AFC, last updated 7th April 2005*/
3   package afc.gui;
4
5   import javax.swing.*;
6   import javax.swing.text.*;
7   import java.awt.image.BufferedImage;
8   import javax.swing.event.MouseInputAdapter;
9   import java.awt.image.MemoryImageSource;
10  import java.awt.event.*;
11  import java.awt.geom.*;
12  import java.awt.*;
13
14  import java.util.*;
15  import java.math.*;
16  import afc.spatial.*;
17  import afc.utilities.*;
18
19  //panel that draws a raster map
20
21  public class RasterPanelVik implements Drawable
22  {
23     //------instance variables
24     private GraphicsListener graphicsListener;
25     private Image rasterImage,rasterImageGrad;
26     private BufferedImage bi;
27     private JTextComponent output,valueOutput;
28     private Raster rast;
29     private Vector vectCoords,images,points,mouseCoords;      // Vector map
   to draw.
30     private int noData,type,numRows,numCols,rastNo,numRasters;
31     private double xllCorner,yllCorner,cellSize;
32     private float minScale,maxScale,oldMin,oldMax,clickChangeValue;
33
34     private double panelWidth,panelHeight,mapAspectRatio,panelAspectRatio,
35           scaling,hCornerOffset,wCornerOffset,imageScaling;
36     private boolean saveFile,allowChanges,multiRasters;
37     private RasterScalePicker rsp;
38     private Ice_model_Run imr;
39     private String scaleType;
40     private Toolkit tk;
41
42
43     //---------constructor
44
45     //creates a panel and draws the given raster onto it
46
47     public RasterPanelVik(Ice_model_Run imr,Dimension d)
48     {
49        super();
50        this.imr=imr;
51  //    this.setPreferredSize(d);
52  //    this.setBackground(new Color(128,128,128));
53        numCols=1;
54        numRows=1;
55        cellSize=1;
56        images = new Vector();
57     }
58
59     public RasterPanelVik(Raster rast,Vector vectCoords,Vector
   points,RasterScalePicker rsp)
60     {
61        super();  //calls JPanel's constructor
62        this.rast = rast;
63        this.vectCoords = vectCoords;
64        this.points = points;
65        for (int f=0;f<points.size();f++)
66           ((afc.spatial.Point)points.elementAt(f)).displayValues();
67  //    initialise();
68              images = new Vector();
```



```
69        setRasterMap(rast,rsp); //store raster map
70  //    repaint();
71     }
72
73     public RasterPanelVik(Raster rast,Vector pts,RasterScalePicker
   rsp,boolean arePoints)
74     {
75        super();  //calls JPanel's constructor
76        this.rast = rast;
77        this.points = pts;
78        for (int f=0;f<points.size();f++)
79           ((afc.spatial.Point)points.elementAt(f)).displayValues();
80  //    initialise();
81              images = new Vector();
82        setRasterMap(rast,rsp); //store raster map
83  //    repaint();
84     }
85
86     public RasterPanelVik(Ice_model_Run imr,Raster rast,Vector
   vectCoords,Vector points,RasterScalePicker rsp,JTextComponent output,
87                           JTextComponent valueOutput,float
   clickChangeValue,boolean allowChanges,boolean saveFile)
88     {
89        super();  //calls JPanel's constructor
90        this.imr = imr;
91        rastNo=0;
92        multiRasters=false;
93        this.rast = rast;
94        this.output = output;
95        this.valueOutput = valueOutput;
96        this.rsp = rsp;
97  //    initialise();
98              images = new Vector();
99        this.vectCoords = vectCoords;
100       this.points = points;
101       this.saveFile = saveFile;
102       this.allowChanges = allowChanges;
103       this.clickChangeValue = clickChangeValue;
104       setRasterMap(rast,rsp); //store raster map
105 //    Dimension panelSize = getSize();
106 //    paintImmediately(0,0,panelSize.width,panelSize.height);
107    }
108 //       this.output = new JTextComponent("cursor");
109 //       output.setPreferredSize(new
   Dimension((infoPWidth-4)-pCAPWidth,pCAPHeight));
110
111 //       this.add(output, BorderLayout.SOUTH)
112
113    public RasterPanelVik(Raster rast,RasterScalePicker rsp)
114    {
115       super();  //calls JPanel's constructor
116       this.rast = rast;
117 //    initialise();
118             images = new Vector();
119       setRasterMap(rast,rsp); //store raster map
120 //    repaint();
121    }
122
123    public RasterPanelVik(Raster rast)
124    {
125       super();  //calls JPanel's constructor
126       this.rast = rast;
127 //    initialise();
128             images = new Vector();
129 //    setRasterMap(rast,rsp); //store raster map
130 //    repaint();
131    }
132
133    public RasterPanelVik(Ice_model_Run imr,Vector rasters,Vector
   vectCoords,Vector points,RasterScalePicker rsp,JTextComponent output,
134                          JTextComponent valueOutput,float
```

```
clickChangeValue,boolean allowChanges,boolean saveFile)
   {
      super();  //calls JPanel's constructor
      RasterScalePicker rspSelect;
      multiRasters=true;
      numRasters = rasters.size();
      this.imr = imr;
      this.output = output;
      this.valueOutput = valueOutput;
      this.saveFile = saveFile;
      this.allowChanges = allowChanges;
      this.clickChangeValue = clickChangeValue;
//    initialise();
            images = new Vector();
      int i=0;
      int classes;
      int upperL = imr.getUpperLayer();
      while(i<rasters.size())
      {
         rastNo=i;
         System.out.println("upperL: "+upperL);
         Raster rast1 = (Raster)rasters.elementAt(i);
         if(rast1.getScaleType()=="thiessens")
            classes = (int)rast1.getMaxVal(noData,noData);
         else
            classes=0;
         if(i!=upperL)
            rspSelect = new
RasterScalePicker(rast1.getScaleType(),rast1.getNoData(),classes,imr.get
ckAndWhite());
         else
            rspSelect = rsp;

         setRasterMap(rast1,rspSelect);//store raster map
         if(i==upperL)
            this.rast = rast1;

         i++;
      }
   }

   //---------methods

   public Vector paint(Graphics g)
   {
      return images;
   }

   public void setRasterMap(Raster rast,RasterScalePicker rsp)
   {
      System.out.println(rast.getScaleType());
      minScale = rsp.getMinScale();
      maxScale = rsp.getMaxScale();

      Color colour;
      this.numRows = rast.getYcells();
      this.numCols = rast.getXcells();
      this.xllCorner = (int)rast.getXll();
      this.yllCorner = (int)rast.getYll();
      this.cellSize = rast.getCellSize();
      this.noData=rast.getNoData();
      Raster scaledRaster;

if((rast.getScaleType()=="iceland_topo")||(rast.getScaleType()=="edits")
         ||(rast.getScaleType()=="cost")||(rast.getScaleType()=="drainag
      {
         this.oldMax = rast.getMaxVal(-1,-1);
         this.oldMin = rast.getMinValNotNoData(-1,-1);
         scaledRaster = rast.scaleValues(minScale,maxScale,-1,-1);
      }
      else if(rast.getScaleType()=="climseq")
```



```
      {
         scaledRaster = rast.scaleValues(minScale,maxScale,-1,-2);
         this.oldMax = rast.getMaxVal(1,2);
         this.oldMin = rast.getMinValNotNoData(1,2);
      }

      else if(rast.getScaleType()!="thiessens")
      {
         scaledRaster = rast.scaleValues(minScale,maxScale,noData,noData
         this.oldMax = rast.getMaxVal(noData,noData);
         this.oldMin = rast.getMinValNotNoData(noData,noData);
      }
      else
      {
         scaledRaster = rast;
         this.oldMax = rast.getMaxVal(noData,noData);
         this.oldMin = rast.getMinValNotNoData(noData,noData);
      }
      //Transfer raster values into a pixel array
      int[] pixels = new int[this.numRows*this.numCols];
      int pixelCount = 0;

   // rast.printRasterData();
      for (int row=0;row<this.numRows;row++)
         for(int col=0;col<this.numCols;col++)
         {
            int greyLevel = (int)scaledRaster.getValue(col,row);
            if((greyLevel==noData)&&(multiRasters==true)&&(rastNo>=0))
               colour = new Color(0,0,0,0);
            else
               colour = rsp.calculateColour(greyLevel);
            pixels[pixelCount++] = colour.getRGB();
         }
      //create a drawable image out of the pixel array
      //the class Image (rasterImage) expects a 1-D array of the image
      MemoryImageSource ms = new
java.awt.image.MemoryImageSource(numCols,numRows,pixels,0,numCols);
      images.addElement(ms);

   }

   public Raster getRaster()
   {
      return this.rast;
   }

   public int getRows()
   {
      return this.numRows;
   }

   public int getCols()
   {
      return this.numCols;
   }

   public double getXll()
   {
      return this.xllCorner;
   }
   public double getYll()
   {
      return this.yllCorner;
   }
   public double getCellSize()
   {
      return this.cellSize;
   }
   public float getRasterValue(float x,float y)
   {
      return this.rast.getValue(x,y,true);
```

```
272      }
273
274      public void addGraphicsListener(GraphicsListener graphicsListener)
275      {
276         this.graphicsListener = graphicsListener;
277      }
278   }
279
```