Master Thesis, Department of Geosciences

Thermal regimes and horizontal surface velocities on Hellstugubreen and Storbreen, Jotunheimen, Southern Norway

Mathieu Tachon

UNIVERSITY OF OSLO
FACULTY OF MATHEMATICS AND NATURAL SCIENCES

Thermal regimes and horizontal surface velocities on Hellstugubreen and Storbreen, Jotunheimen, Southern Norway

Mathieu Tachon

Master Thesis in Geosciences

Discipline: Physical Geography, Hydrology and Geomatics

Department of Geosciences

Faculty of Mathematics and Natural Sciences

University of Oslo

October 1st 2015

Cover photo: Ground Penetrating Radar surveys on Storbreen in a sunny day, April 2014.

Abstract

Radio-Echo Sounding (RES) surveys are an effective way to map the thermal regime of glaciers. RES measurements at two different center frequencies were conducted on Hellstugubreen and Storbreen, two mountain glaciers located in Jotunheimen, Southern Norway. The ice thickness was investigated from measurements at a frequency of 10 MHz, from 2011 at Hellstugubreen and from 2005-2006 at Storbreen. In 2014, RES surveys at a frequency of 50 MHz was used to map the internal thermal layering of the glaciers. Ice temperature variations in the subsurface were also explored with shallow borehole measurements. The results revealed a polythermal regime for both glaciers, which are cold-based near the front and their margins, and with a cold surface layer underlain by temperate ice in their central parts. A maximum ice thickness of 177 m (±15 m) was recorded at Hellstugubreen. The bedrock was encountered at a maximum depth of 233 m (±15 m) at Storbreen (uncorrected for the surface lowering of the past 10 years). The depth of the *Cold-temperate Transition Surface* (CTS) generally increased with elevation, and reached a maximum depth of 90 m at Hellstugubreen and 55 m at Storbreen. By the end of the summer season, remaining cold ice was found in the subsurface of Hellstugubreen,whereas on Storbreen, the cold wave was completely eliminated at same depth levels.

Stake surveys based on accurate *Differential Global Navigation Satellite System* (DGNSS) georeferencing were carried out since September 2009 on Hellstugubreen and since September 2006 on Storbreen. The data available were exploited to gain an insight into the surface velocities of both glaciers. For the period 2013-2014, the surface velocities ranges from 0.5 m.yr^{-1} to 15.8 m.yr^{-1} at Hellstugubreen. Between 2010 and 2014, the measurements at Storbreen indicated surface velocities ranging from 2.5 m.yr^{-1} up to 16.2 m.yr^{-1}.

Acknowledgements

I would like to thank my supervisors Jon Ove Hagen and Liss Marie Andreassen for their support and their help during this investigation. I also truly appreciated the superb field work spent in their company. I thank UiO and NVE for the financial and logistical supports during the field work. I am also grateful to Thomas Schuler and Trond Eiken, for their valuable help provided during field work preparations and the field data acquisition. I wish to thank Thorben Dunsen for his technical help during the GPR data processing. I address my special thanks to Oda Jonette Røyset and Alex Walmsley for the nice team work on the field and for exchanging ideas on this investigation. I extend my special thanks to the students from the study desk 214 for their contributions to a nice working environment. Lastly I would like to thank my wonderful girlfriend and my son, for their continuous support during this work and to cheer me up during stressful periods.

Contents

List of Figures

List of Tables

Introduction

This thesis investigates the thermal regime of two glaciers located in the Jotunheimen mountains, Southern Norway. Most glaciers are considered to have a temperate thermal regime in mainland Norway (Andreassen et al., 2012). However, previous studies pointed out that it exists several exceptions to this general trend in the eastern part of Southern Norway (Urdahl, 2005; Ødegård et al., 2011; Sørdal, 2013). A large part of Jotunheimen area is located above the *Mountain Permafrost Altitude* (MPA) and the *Equilibrium Line Altitude* (ELA), and therefore offers a favourable climate to develop and sustain polythermal structures in glaciers (Etzelmüller and Hagen, 2005). On small glaciers characterized by relatively low accumulation rates, the snow cover do not suffice to impede the cold winter wave to penetrate deep into the ice (Björnsson et al., 1996). In the high-alpine environment of Jotunheimen, the summer temperatures are not always warm enough to eliminate this cold wave, which allows a cold surface layer to persist. In widespread permafrost areas, the glaciers thermal regime can also be affected from underneath, and transit from a temperate to a partly cold-based thermal regime (Björnsson et al., 1996; Hagen et al., 2003).

The thermal regime of glaciers is of great importance, as it affects both their hydrology and dynamics. Temperature variations in ice have direct effects on its physical properties and deformation rate. Moreover, hydrological processes present in temperate glaciers are not sustainable in cold ice. Temperate-based regime allows for basal sliding and leads to higher flow velocities. In temperate ice, the water from the summer surface melt can find its ways down to the bottom of glaciers through water channels, and lubricate the bed. This water input in the subglacial drainage system can result in significant increases of the surface velocity (Rabus and Echelmeyer, 1997; Copland et al., 2003). The presence of a cold surface layer in polythermal glaciers can limit or inhibit completely this process. The prior knowledge of the temperature distribution in glaciers is therefore essential for ice flow modelling.

The initial focus of this thesis was to get an insight into the ice thickness and the thermal regimes of Hellstugubreen and Storbreen, both suspected to be polythermal. This work, carried out in collaboration with the *Norwegian Water Resources and Energy Directorate* (NVE), was extended to horizontal surface velocity assessments. The specific objectives of this thesis are as follows :

1. Mapping and estimate at a regional scale the ice thickness and thermal regimes of Hellstugubreen and Storbreen, by using multi-frequency *Radio-Echo Sounding* (RES) measurements.

2. Investigating the ice temperature variations in the subsurface with shallow borehole temper-

ature measurements.

3. Estimating horizontal surface velocities from stake surveys, based on non-continuous *Differential Global Navigation Satellite System* (DGNSS) georeferencing.

This thesis is organised into three main parts. The first one includes a theoretical chapter on temperature distributions and classification of glaciers. A following section highlights the main processes that contribute to heat transfers in ice masses. A third section gives an overview of the effects of temperature on glacier dynamics. A last section sheds a light on basic RES principles and applications in glaciology. The second and last chapter is devoted to the description of the study area, with a short overview of the geographical setting and the data available from past measurements. The second part describes the field and data analysis methods. The last part presents the results from this work, with separate chapters for the ice thickness, the thermal regimes and the surface velocities assessments of the two glaciers of interest. Each result is followed by a discussion section. This thesis ends with a last chapter highlighting the main conclusions from this work.

Part § 1

Theoretical background and geographical setting

Chapter A

Theoretical background

1 Glaciers thermal regimes and dynamics

1.1 Temperature distribution and glacier classification

The temperature distribution in glaciers and ice-sheets define their thermal regimes. The temperature distribution in a glacier results from the combination of numerous processes and heat sources, which effects on ice temperature are more or less significant, at a local or glacier-wide scale. In addition, a number of these processes and heat sources depend on the on-site climatic conditions, and therefore on the geographical location of the ice masses. Cuffey and Paterson (2010) identified four types of temperature distribution in ice masses : the ice temperature can either be (i) below the melting point across the full ice thickness; (ii) at the melting point only at the ice/bed interface; (iii) at the pressure-melting point for a basal layer of a finite thickness; and (iv) at the pressure-melting point for the full ice thickness. The different temperature distributions enable to distinguish three types of glacier (Maohuan, 1990, 1999; Cuffey and Paterson, 2010) : cold or polar type glaciers, polythermal or sub-polar type glaciers, and temperate glaciers.

Cold/Polar glaciers

Cold or polar glaciers describe typically the glaciers for which the ice is below the melting point. If only the ice at the ice/bed interface reaches the pressure-melting point, the glacier can also be regarded as a polar type one (Maohuan, 1999). Polar glaciers usually stand at high altitude or lay in cold and dry regions of the Earth. As the ice is below the melting point, the small amount of surface melt water, produced during the warmest periods, never reaches the bed and refreezes almost instantly (Maohuan, 1999). The cold ice cannot sustain a subglacial hydrological network, and therefore the glacier is '*frozen*' to its bedrock. Thus, the velocity of cold glacier depends on a single component, which is the deformation rate of the ice (see section *Ice temperature and Glen's flow law*). Therefore, their velocity is often relatively low compared to the other types of glacier (Cuffey and Paterson, 2010). The surface velocity during summer is also not much different from the winter surface velocity (Maohuan, 1999), as cold glaciers are not affected by the development of a hydrological network that lubricate the base, and makes the ice less viscous by cryo-hydrological warming.

Polythermal/Sub-polar glaciers

Polythermal or sub-polar glaciers belong to type which has only a finite thickness of ice at the melting point. The parts of temperate ice are often at the base of the glacier, where higher basal pressures allow the ice to reach the pressure melting point (Cuffey and Paterson, 2010). In the accumulation area, the melt water produced early in the ablation season will refreeze (see section *Refreezing*) in the upper layers (i.e. snow and firn mainly), where temperature is below 0°C, and thus allowing an early warming of these layers (Hagen et al., 2003). In the ablation zone, melt water percolates down to the base of the glacier, through moulins, crevasses and fractures that propagate due to this melt water input. The melt water is then drained through a subglacial hydrologic network within the temperate ice, or directly contributes to the ground water flow, beneath the glacier. Hagen et al. (2003) and Gilbert et al. (2012) support the idea that no refreezing occurs in the ice in the ablation zone of polythermal glaciers. Thus, as the air temperature is most likely negative in sub-polar regions, the upper part of the ice remains cold all through the year.

Temperate glaciers

Temperate glaciers are at the pressure melting point throughout their entire mass. They form in regions at lower latitude than the types of glaciers mentioned above, and usually require a more maritime climate or a mountainous climate below a certain elevation threshold, where amounts of precipitation are larger and ablation-season temperatures are greater. Owing to the fact that most of the ice is at the melting point, subglacial hydrological systems can easily develop in the ice mass. These subglacial hydrological systems have two different regimes (Fountain and Walder, 1998). During the winter season, when the melt water production is low, it is a highly pressurized cavity-based system that drains melt water down to the glacier front (Nye, 1973). When summer comes and the melt water production increases, the subglacial hydrological network switch to a more efficient but less pressurized channel-based drainage system (Röthlisberger, 1972). The higher basal temperature gives them a different basal regime which is explained more deeply in section *Basal thermal regimes*.

1.2 Temperature profiles

So far, only one measurement method enable to obtain accurately the temperature profiles of glaciers and ice-sheets. This consists in drilling a borehole into the ice, and to set up thermistors inside the borehole, at multiple depths. The main inconvenience of this method is its limited spatial resolution, due to the time required to drill down to great depths, for studies of temperature distribution on ice-sheets for instance. Temperature profiles have been measured in different regions, representing various climates. For the polar and sub-polar climate, characterized by a dry air and mean annual temperatures below freezing (e.g. Antarctica, Greenland and glaciers in Arctic latitudes), boreholes measurements were performed at shallow depths and at depths greater than 2 300 meters with a vertical accuracy of ~2 meters for the deeper measurements, and with uncertainties ranging from ±0.01 to ±0.5° K after calibration of the thermistors (Paterson, 1968; Jania et al., 1996; Price et al., 2002; Rolandone et al., 2003; Phillips et al., 2013). Cold and polythermal glaciers have also been identified in regions with a more continental/alpine climate, such as in the Himalayas (Maohuan, 1990, 1999; Conway and Rasmussen, 2000), in the Alps (Haeberli and Funk, 1991) or even in southern Norway (Andreassen et al., 2012). A large number of studies were likewise carried out on glaciers subjected to a maritime climate, which can be defined as mild temperatures and important precipitations during winter, and with higher summer temperatures . The work of Andreassen et al. (2012) points out that most glaciers are temperate in these regions.

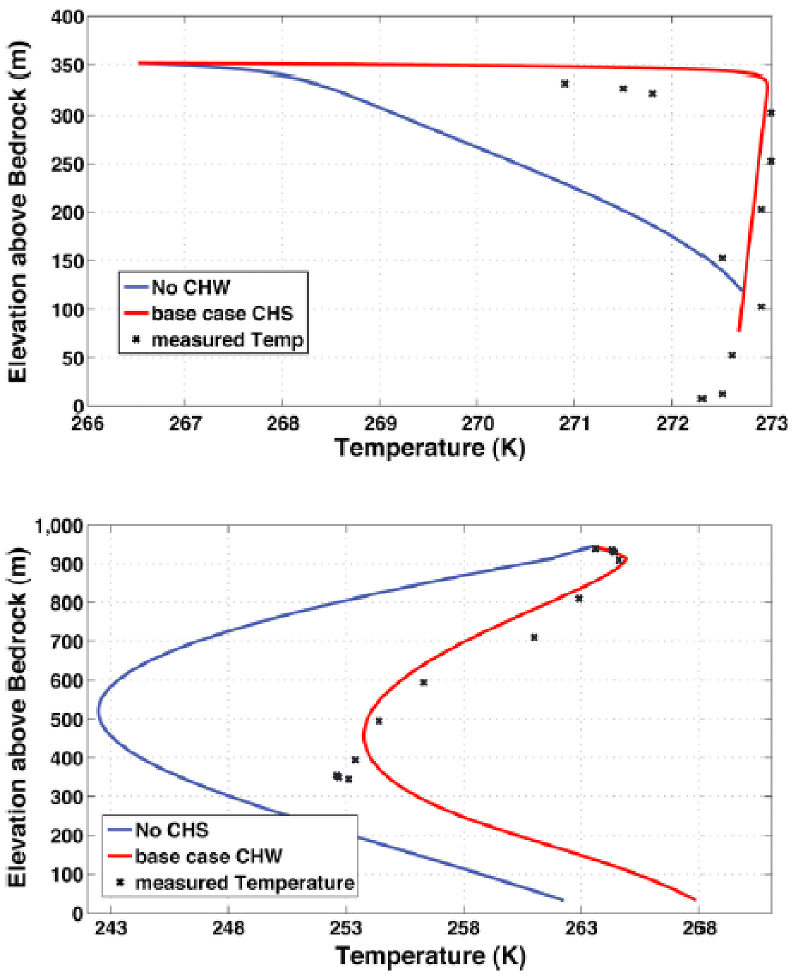


Figure § 1.1: Simulated temperature profiles superimposed with observations (dots) performed at two borehole sites, on Sermeq Avannarleq, West Greenland (Phillips et al., 2013). The blue curves are simulations without the effect of the cryo-hydrologic warming, and the red curves are simulations accounting for this effect.

Radio-echo soundings provide other means of measuring temperature profiles in ice masses. Radar signal attenuation is proportional to the depth penetration and the conductivity of the ice. Changes in dielectric properties of the ice are dependent of the ice temperature (Hughes, 2008), and such changes lead to polarisation and conductivity losses in the radar signal. The method using radar sounding relies on estimating these changes which depend on the ice temperature, in order to obtain the temperature itself. However, the ice conductivity variations can be the result of other factors, such as the impurities concentration (Hughes, 2008). In addition, water pockets and subglacial hydrological channels that are present in temperate ice prevent the radio waves to penetrate further. Radar sounding has therefore several limitations when it comes to measure temperature distributions in glaciers.

Finally, the last approach to estimate the temperature distribution of ice masses is by using models. For cold glaciers, a temperature profile can be estimated from a limited number of data (Robinson, 1984), such as : the ice geometry, the mean annual near-surface temperatures, the ice velocities and the geothermal heat influx. When it comes to temperate and polythermal glaciers, more data sets are required. Firstly, for both types, the surface velocity results from the combination of two components : the deformation rate of the ice (see sections *Ice deformation* and *Ice temperature and Glen's flow law*), and the basal sliding (Section *Basal thermal regimes*). The actual surface velocity fields can be measured accurately using several techniques, such as *Interferometric Synthetic Aperture Radar* (InSAR) or ground-based measurements (i.e. *Global Positioning System*) (Phillips et al., 2013). However, in ice flow models, the basal sliding is often the unknown parameter since no accurate method to measure its velocity is available at the present's day. The second ambiguity that derives from the temperate and the polythermal type is their ability to sustain a complex subglacial drainage system, which evolves seasonally. This subglacial hydrological network is likely to affect the temperature gradient throughout the ice thickness (Phillips et al., 2013) and therefore to change its flow properties. It is essential to account for the *Cryo-Hydrologic Warming* (CHW) (see section *Cryo-hydrologic warming*) to model the temperature distribution in polythermal and temperate glaciers. The effect of the inclusion of the CHW (*Cryo-Hydrologic Warming*) in a thermo-mechanical model is illustrated by the *Figure § 1.1.*

2 Processes affecting the temperature gradient

As mentioned in section *Temperature distribution and glacier classification*, the temperature in ice masses depends on numerous processes such as : the heat conduction, the heat convection, refreezing, the ice deformation, the sliding friction, or even the geothermal flux. The strength of these processes depends on the geographical location of the glaciers, or ice-sheets, and therefore it is related to a certain climate and to certain types of ice mass.

2.1 Heat conduction and mean annual surface temperature

The heat conduction is the heat energy transfer by microscopic diffusion. It is the process that can lead an ice mass to reach a steady-state temperature distribution, if no other heat sources interact with the ice mass and if the boundary conditions are unchanged (i.e. stable air surface temperature and constant geothermal influx). An ice mass is assumed reach the thermal steady-state if its temperature gradient is constant across the full ice thickness. In most cases, an ice mass shows large deviations from the thermal steady-state, as many processes contribute to heat transfers throughout the medium, and not only conduction. The heat conduction depends on the thermal conductivity of the medium. For ice, dry snow and firn, Van Dusen (1929) gave the following empirical formula to calculate the thermal conductivity :

$$k_T = 2.1 \times 10^{-2} + 4.2 \times 10^{-4} \rho + 2.2 \times 10^{-9} \rho^3 \tag*{(§ 1.1)}$$

where ρ is the density of the material. The heat conductivity (q) can be expressed as the amount of energy flowing across unit area, per unit of time, and it is proportional to the temperature gradient ($\partial T/\partial z$) :

$$q = -k_T \cdot \frac{\partial T}{\partial z} \tag*{(§ 1.2)}$$

where T is the temperature, and z is the distance, measured in the direction of the temperature variation. In order to obtain the temperature profile, z is taken as the difference in depth between the two boundaries of the medium. The minus sign in (§ 1.2) stands for the direction of the flux propagation towards lower temperatures. The heat conduction can be the dominant heat transfer across a stagnant ice mass in polar regions (Paterson, 1968; Rolandone et al., 2003). In summer, however, for most glacial areas that experience surface melt or receive precipitation as rain, the energy transfer in the upper layers from heat conduction is often negligible in comparison with that of the refreezing process(see section *Refreezing*), especially in the accumulation area (Maohuan, 1990; Ødegård et al., 1992; Gilbert et al., 2012).

2.2 Refreezing

Refreezing is the process that results from the melt water input, via percolation through snow, firn and ice, and which turns back to the solid phase, owing to the temperature below melting point of the environment. The energy (i.e. radiations, warm temperatures) consumed by the snow or ice surface for melt is then released as latent heat when the melt water refreezes. This release in energy warms up the medium where the process occurs. Refreezing can therefore be expressed as an amount of heat available per units of time and volume. The following equation enables to quantify refreezing of surface water in firn (Cuffey and Paterson, 2010) :

$$R = L \cdot w_s \cdot \rho_s \cdot \frac{m_f}{z_m} \tag{§ 1.3}$$

where L denotes the specific latent heat fusion, w_s stands for the vertical velocity of percolation at the surface, ρ_s is the density of the surface layer, m_f represents the melt fraction (fraction of annual firn layer, in weight units, formed by refreezing), and z_m is the maximum depth of percolation. Refreezing is one of the most efficient heat source in areas covered by a firn layer, and that experiences surface melt (Ødegård et al., 1992; Gilbert et al., 2012). Refreezing 1 g of water releases enough energy to warm up 160 g of snow or firn by 1°C (Cuffey and Paterson, 2010). The study of Maohuan (1990) show that the warming penetration depth can reach 30 meters. This partly explains the temperature distribution of certain sub-polar glaciers, with warmer ice in the accumulation than in the ablation zone, at lower altitude for the latter. During the ablation season, melt water percolates through the firn, then refreezes and warms up the firn. The temperature of the underlying ice increases by conduction. At the same period of the year, there is only little refreezing in the ablation area (Hagen et al., 2003), where the ice is not snow-covered and exposed directly to the air surface temperatures. During winter, the snow/firn cover that remains in the accumulation zone acts as an "*insulating blanket*", owing to its low thermal conductivity, and impedes the cooling of the underlying ice by the low air temperatures. The ice in the accumulation area is therefore at the melting point all through the year. However, in the ablation zone, the temperature of the uppermost layer fluctuates with air temperature, and underneath, a finite layer of ice remains below the melting point since the mean annual temperatures in the sub-polar regions are below 0°C.

2.3 Cryo-hydrologic warming

The *Cryo-Hydrologic Warming* (CHW) is the combined effects of refreezing and heat conduction and convection. It occurs when a surface melt water input flows through a *Cryo-Hydrologic System* (CHS) (Fountain and Walder, 1998), or is simply standing in crevasses and other conduits (Phillips et al., 2013). Because of its temperature equal to zero or even positive, melt water can significantly affect the temperature gradient of the ice. In response to a higher temperature gradient, the heat transfer by conduction and convection lead to an effective warming of the ice surrounding the CHS (*Cryo-Hydrologic System*), whose intensity is controlled by the density and the geometry of the CHS (Phillips et al., 2010). As the surface melt water need first to reach the CHS for the onset of the CHW, this combination of processes occurs mainly in the ablation zones of glaciers. Indeed, the snow cover that remains during summer in the accumulation area traps the melt water that percolates and refreezes in the snowpack, before reaching the ice. Thus, the surface melt water in the accumulation zone alters only the temperature gradient of the uppermost ice layer (see section *Refreezing*). The fluctuations of the snowline is therefore a controlling factor on the glacier area that may be affected by CHW.

2.4 Heat advection and ice velocity

The heat convection or heat advection is the process by which the internal temperature distribution of an ice body is altered due to a displacement of ice with different temperature. As such, cold ice flowing towards zones with relatively warm ice may increase the temperature gradient at some point, and the other way around for warm ice flowing towards cold ice areas. On a vertical profile, the temperature can be affected by convection occurring on both horizontal and vertical directions :

$$\frac{\partial T}{\partial t} = -w\frac{\partial T}{\partial z} - u\frac{\partial T}{\partial z} \tag{§ 1.4}$$

where t is the elapsed time between the start and the end of the calculation, and where w and u are the vertical velocity and the horizontal velocity respectively. Equation (§ 1.4) enables to calculate changes of temperature over time, at a particular point of the ice mass ($\partial T/\partial t$), and if convection would be the only heat transfer. For fast flowing ice streams (e.g. $>$ 1 km.yr^{-1} for some outlets of the Antarctica ice-sheet), ice convection is one of the main heat transfer component (Huybrechts and Oerlemans, 1988; Cuffey and Paterson, 2010; Pattyn, 2010). Conversely, in relatively stagnant ice masses, the convection term is near to zero in the heat transfer equation (Paterson, 1968).

2.5 Ice deformation

The ice deformation is another type of heat source. The energy produced by the ice deformation is proportional to the stress applied by the environment on the ice, and to the strain rates of the medium. Considering the ice as incompressible is a good approximation, and the energy released as heat by the ice deformation can then be expressed as follows (Cuffey and Paterson, 2010) :

$$d_{ice} = \dot{\epsilon}_{jk} \cdot \tau_{jk} \tag{§ 1.5}$$

where $\dot{\epsilon}_{jk}$ and τ_{jk} are the deviatoric strain rates and stresses respectively. Hence, the temperature gradient contribution of the ice deformation is usually more important at the interface ice/bed or in lateral shear margins of glaciers and ice-sheets.

2.6 Sliding frictions

The heat production that results from the friction between a flowing ice mass and its bedrock is a potential contributor to the ice temperature gradient variations. It leads to an increase in the basal layer temperature gradient and keeps the basal ice at pressure melting point in fast flowing zones of glaciers (Blatter, 1987; Pattyn, 2010). This heat source is generally more significant along the center line of glaciers, and decreases towards the margins (Robinson, 1984). The energy generated is equal to the product of the ice velocity and the resistive force :

$$f_b = u_b \cdot \tau_b \qquad (\S\ 1.6)$$

where u_b is the basal ice velocity, and τ_b is the shear stress of the ice against the bedrock. Cuffey and Paterson (2010) pointed out that for a thickness and a slope of the bed corresponding to a shear stress of 100 kPa, combined with a basal slip ranging from ~15 to 20 m.yr^{-1}, the heat released by basal sliding friction is of the order of a typical geothermal heat flux (see section *Geothermal heat flux*).

2.7 Geothermal heat flux

Most of the material in section *Geothermal heat flux* is based on the work of Sclater et al. (1980).

The geothermal heat flux affects the temperature gradient in ice masses from beneath. This flux results partly from the formation of the continental crust from the warm mantle of the astenosphere. The temperature of this newly formed lithosphere decreases gradually over time, which leads to spatial variations of the geothermal flux.

In addition to the initial warmth of the young lithosphere, other factors contribute to the total geothermal flux, at specific points of the Earth's surface. Orogenic events triggered by continental collision, or even continental stretching are potential heat producers. As for the ice deformation (see section *Ice deformation*), the deformation of the continental crust is an exothermic transformation. The stresses and strains arising from the lithosphere motions can be sources of significant amounts of energy.

Furthermore, radio-elements can also be heat-producing elements, when it comes to their decay into radio-genic compounds. Hence, the geothermal influx in ice masses depends likewise on the radio-elements content of the underlying lithosphere, especially the content in uranium (**U**), thorium (**Th**) and potassium (**K**).

The last major contributor to the geothermal flux is certainly volcanoes, especially in regions, such as Iceland, where both volcanoes and glaciers are close, or even in contact, to each others.

The non-radiogenic component of the geothermal flux reduces to a constant value of ~21-25 mW.m^{-2}, after a period ranging from 200 to 400 Ma following the lithosphere formation. The total geothermal flux would decrease to reach the constant value of ~42-50 mW.m^{-2} after 800 Ma. In their work, Sclater et al. also contend that most of the continental crust dates back to ~3 800 Ma. However, orogenic events or even erosion tend to modify continuously the age of the continental crust. Therefore, the continents have been divided into four age provinces. A mean geothermal flux of ∼77 mW.m^{-2} has been recorded for the youngest province (<250 Ma), and the averaged geothermal flux measured at the oldest (>1 700 Ma) neared ∼46 mW.m^{-2}.

3 Temperature and dynamics

3.1 Ice temperature and Glen's flow law

One the main contributions of temperature in the glaciers dynamics is its effect on the ice deformation rate. An increase in the ice temperature has as consequences to decrease the ice viscosity. The creep or shear strain rate of the ice is directly proportional to the viscosity of the material as illustrated by the Glen's flow law (Glen, 1955) :

$$\dot{\epsilon} = A \cdot \tau^n \qquad (\S\ 1.7)$$

where A is the creep factor dependant on temperature, τ is the dominant shear stress and n is an empirical creep exponent with a mean value of about 3. As an order of magnitude, a cooling of the ice from -10°C to -25°C increases its viscosity, and hence the shear strain rate, by a factor of 5 (Cuffey and Paterson, 2010). Changes in ice temperature have therefore direct effects on the flow velocity of glaciers.

3.2 Basal thermal regimes

Ice masses can be characterized by two different basal thermal regimes, which reflect if the basal ice is sliding over the bed or not. The basal regime has likewise a significant role in the subglacial hydrology of glaciers, and can determine whether the melt water will find its way to bed or not.

Cold-based glaciers have a well defined basal thermal regime. The basal ice of these glaciers is at temperature below the melting point, which prevents basal sliding. Cold-based regime can also be characterized by its inability to sustain subglacial hydrological system. If ablation-season temperatures are high enough to produce surface melt water, the water would refreeze before reaching the bed, while percolating in the overlaying firn, or in contact with the cold ice. Mountain and small valley glaciers in polar regions are typical glacier with a cold-based regime (Blatter, 1987; Maohuan, 1990; Haeberli and Funk, 1991; Maohuan, 1999; Lovell et al., 2015). Polythermal glaciers may partly be cold-based, usually close to the front and their margins (Björnsson et al., 1996; Jania et al., 1996). For land terminating glaciers, a cold-based regime usually leads to weak proglacial streams, generated by a relatively poor annual melt water input reaching the glacier front. The water feeding these streams may be a good approximation for the total surface run-off of the glacier.

Temperate glaciers have another thermal regime. Owing to the presence of temperate ice at the base of these glaciers, basal sliding is therefore possible. This component must be taken into account in ice flow models (Shannon et al., 2013). Increases in velocity during the melt season is often an indicator of temperate or polythermal basal ice (Rabus and Echelmeyer, 1997). Studies performed at Jakobshavn Isbrae using temperature borehole measurements, combined with surface velocity measurements, support this theory (Iken et al., 1993; Funk et al., 1994; Lüthi et al., 2002; Zwally et al., 2002). As opposed to cold ice, temperate or polythermal ice enable the development of a subglacial drainage system (see *Temperature distribution and glacier classification*). The sudden acceleration at Jakobshavn Isbrae is thought to result from enhanced basal lubrication, due to the melt water input produced during the ablation-season, together with an active basal sliding. The basal sliding can occur on account for the presence of polythermal ice at the lower part of the ice-sheet. Basal ice at the melting point can therefore affect significantly glaciers dynamics, and consequences of a temperate/polythermal basal regime can be observed even on high latitudes glaciers (Rabus and Echelmeyer, 1997; Copland et al., 2003). Other field observations referred to temperate/polythermal basal ice and its ability to shelter a subglacial drainage system (Fountain and Walder, 1998). Such assumptions were done when a supraglacial lake, which presumably drained to the base of the ice-sheet, triggered an uplift of the ice, followed thereafter by a flow acceleration (Das et al., 2008). Accelerated flows downstream of moulins in Greenland support also the idea of the basal lubrication process (van de Wal et al., 2008).

4 GPR principles and applications in glaciology

4.1 Overview of the Ground Penetrating Radar principles

Ground Penetrating Radar (GPR) is a geophysical investigation device widely used in Earth sciences since the 1960s. This remote sensing technology belong to the *Radio-Echo Sounding* (RES) systems. It consists of two separate antennas, one emitting an electromagnetic signal, and the other one receiving back the signal. The signal is sent by pulse with a known frequency, and is partly reflected by the inhomogeneities of the medium investigated. The reflections of the signal by these inhomogeneities produce various amplitudes of the signal return, for each pulse. A layer, commonly called *Internal Reflection Horizon* (IRH), can act as a reflector if its dielectric properties are different from the ones of the overlaying material. The depth of the signal propagation is also strongly dependent on the used frequency. A high frequency enables to determine the location of internal reflection horizons with a higher vertical resolution. However, as the electromagnetic waves are more quickly dissipated into heat with a high frequency, lower frequencies will enable a greater depth of investigation. The frequencies used for glaciological applications usually range from 50 to 1000 MHz (Plewes and Hubbard, 2001).

The radar signal propagation in a medium depends mainly on two electrical of this medium : the relative permittivity (relative to the permittivity in free air) and the conductivity, often expressed in mS.m^{-1}. The relative permittivity describes the ability of the material to store an electrical charge, and the conductivity describes the ability of the material to transmit an applied electrical charge (Plewes and Hubbard, 2001).

Finally, the GPR system records a two-way time return (travel of the signal before and after reflection), as well as an amplitude of the signal return. If one knows the velocity of propagation of the electromagnetic signal, the time return corresponding to each internal reflection horizon can be converted to depth. The amplitude of the signal return gives information about the characteristics of the reflecting layers.

4.2 GPR applications in glaciology

GPR has numerous application in glaciology. It has proven to be very useful for mass balance measurements. The traditional way to calculate the winter balance on small valley glaciers is to probe manually the snowpack at multiple locations to get an overview of the snow depth distribution, and to combine the measurements with snow density profiles. However, this method is time consuming and does not enable to cover large areas. The use of the GPR technology revolutionised mass balance measurements for its ability to map the snow depth distribution at a regional scale over a short time, and with a high spatial resolution. Kohler et al. (1997) mapped the depth of the last summer surface on glacier sections of several hundreds of meters, with a point measurement every 20 cm.

The accumulation rate over a glacier is variable both in time and space. A prior knowledge of the past accumulation rates is therefore essential in climatic archives and ice cores analysis. In order to obtain a reliable accumulation rates, measurements must be averaged over several years. Certain IRHs with a known date can be used to compute the mean accumulation rates, such as sulfate-rich layers marked by volcanic eruptions, or even layers showing a high content in radioelements (Pinglot et al., 2001). These IRHs can be detected on radargrams, and their depth calibrated from ice core sites. The GPR can then be used to map the depth of the IRHs over large distances in order to get an insight into the spatial variability of the accumulation rate for a given period. Palli et al. (2002) used the Chernobyl layer together with the 1963 bomb horizon to calculate the mean accumulation rate between 1963 and 1986 on Nordenskjöldbreen, a Svalbard glacier. The 11.4 km GPR profile was calibrated from four drilling sites.

In addition to its valuable use for mass balance measurements, GPRs can be used to record the depth of the bedrock IRH of ice masses. When coupled with the ice surface topography, the glaciers thickness enable to determine the bedrock topography. The ice thickness and the bedrock topography are boundary conditions for numerical modelling of the ice flow of glaciers, and therefore essential in glacier dynamics studies (Dowdeswell et al., 2004).

GPRs are useful devices to investigate the internal structures and thermal layering in glaciers. Borehole measurements can be used to assess the temperature distributions in glaciers. However, for great depths of investigations, these can be expensive and time-consuming operations. Multi-frequency GPR surveys enabled to map at a regional scale the thermal regimes of numerous polythermal glaciers (Björnsson et al., 1996; Jania et al., 1996; Moore et al., 1999; Pettersson et al., 2003).

Finally, the use of GPRs can shed a light on bedrock properties and basal conditions of ice masses, such as the roughness, the wetness of the ice-bed interface, the existence of basal crevasses, or even the presence of subglacial debris (Bamber, 1989; Plewes and Hubbard, 2001).

Chapter B

Study sites

1 Hellstugubreen

Hellstugubreen (61°34'N, 8°26'E) is valley glacier laying in the mountains of Jotunheimen (*Figure § 1.2a*). The glaciers has mostly north-facing slopes and has an area of 2.81 km^2. On the upper part, the ice divide separate Hellstugubreen and Vestre Memurubre glacier. Length measurements were conducted since 1901 and mass balance measurements were carried out annually since 1962 (Andreassen et al., 2012). In 2009, the glacier front elevation was 1494 m.a.s.l. and the uppermost part of the glacier at 2212 m.a.s.l. *Figure § 1.2a* shows the retreat since 1941, with the glacier outlines for different years. The glaciers outlines were derived from orthophotos. The map shows that a large ice patch was disconnected from the glacier between 1968 and 1980. The results from mass balance measurements indicates a predominance of the ablation area over the accumulation area, with the ELA fluctuating between 1840 m.a.s.l. and the maximum elevation of the glacier for the past 20 years. In 2010, the specific net balance was -1.34 m *water equivalent*, resulting in a volume loss of $3.89 \cdot 10^6$ m^{-3}. The surface topography of the glacier is known from *Light Detection And Ranging* (LiDAR) measurements conducted in 2009 by the mapping Norwegian company Blom Geomatics AS. The output data is available at a 5 m spatial resolution. The glacier ice thickness is known from GPR measurements conducted in 2011, with an measurement uncertainties of ±15 m (Andreassen et al., 2015). The first ice surface velocities were estimated by triangulation methods in the 1940s and 1960s (Pay, 2014). Accurate surface velocities can be derived from stake surveys, which are based on DGNSS georeferencing and available from September 2009.

2 Storbreen

Storbree (61°34'N, 8°8'E) is another mountain glacier situated in Jotunheimen (*Figure § 1.2b*). A map from 2009 estimate the glacier area to be 5.1 km^2 (Andreassen et al., 2011*b*). The minimum elevation is at 1400 m.a.s.l. and the maximum elevation at 2102 m.a.s.l. The glacier slopes are north-east oriented. Length measurements were carried out since 1902 at Storbreen (Andreassen et al., 2012). The map in *Figure § 1.2b* shows the glacier front positions at different times since 1940. The glacier outlines were also derived from orthophotos. As the glacier retreated, a nunatak

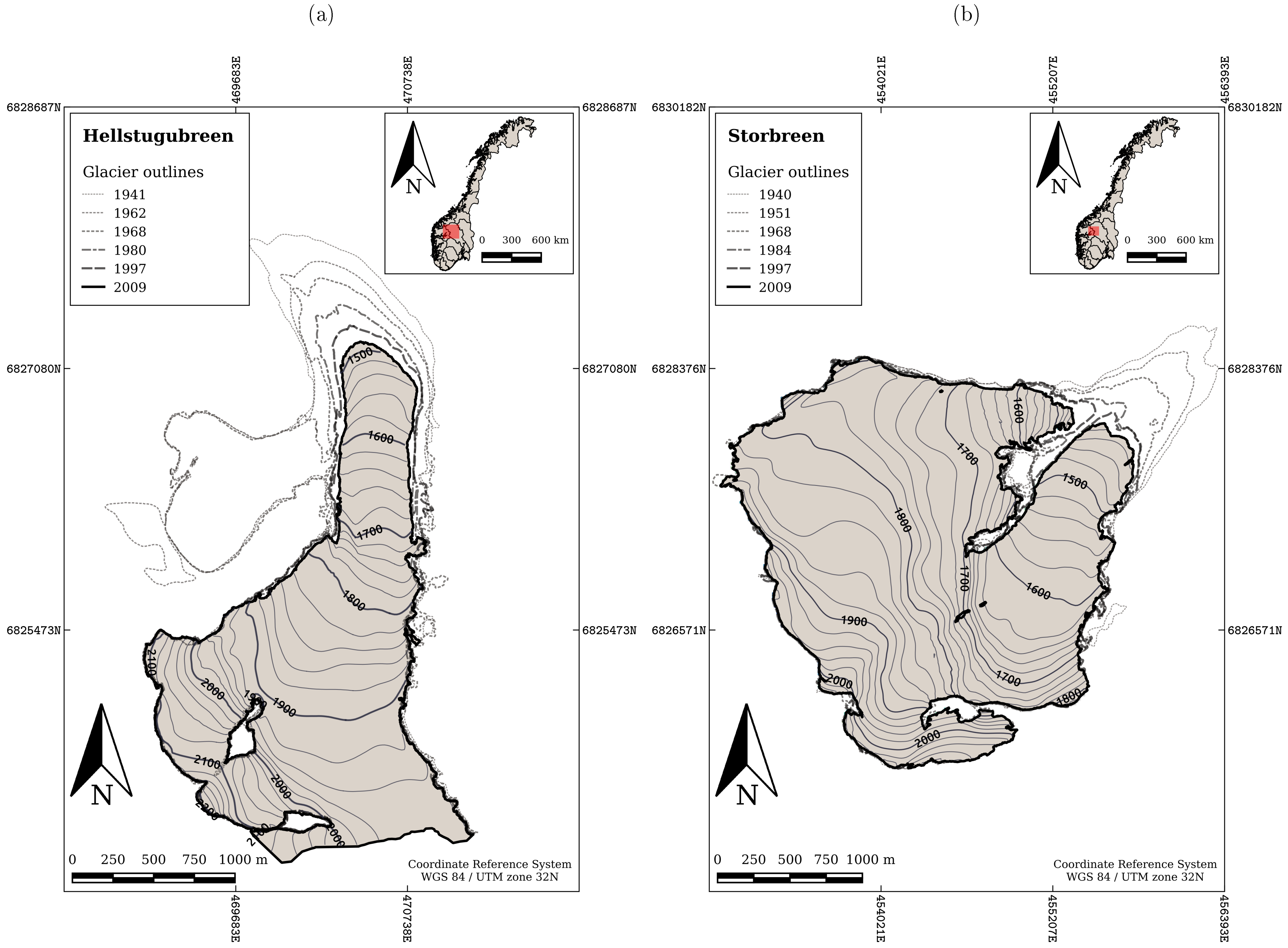


Figure § 1.2: The figure shows the study sites with Hellstugubreen in (a) and Storbreen in (b). In (a), the map shows the glacier retreat since 1941, with the fluctuations of the glacier outlines position. In (b), the glacier retreat from 1940 is shown. For both maps, the elevation contour lines were generated from the 2009 LiDAR data, and the outlines are derived from orthophotos (data : NVE).

separated the front into two glacier tongues. The glacier mass balance was measured annually since 1949 (Andreassen et al., 2011*b*). The ELA showed larger fluctuations than observed at Hellstugubreen, oscillating between 1650 ma.s.l. and the maximum elevation of the glacier during the past 20 years (Andreassen et al., 2011*b*). The work from Andreassen et al. (2011*b*) indicated a specific net balance of -1.76 m w.e. in 2010, resulting in a total mass loss of $9.07 \cdot 10^6$ m^{-3}. Regarding the surface topography, the same LiDAR data are available for Storbreen in 2009. The surface velocity was estimated from previous triangulation works carried out in the 1960s (Liestøl, 1967). Stake surveys with DGNSS referencing started in September 2006 at Storbreen. At Storbreen, the ice thickness is also know at points measurements, covering most of the elevation range of the glacier. The uncertainties of the measurements are also estimated to be $\pm$15 m (Andreassen et al., 2015).

Part § 2

Methods

Chapter A

Ice thickness

1 Field data acquisition

1.1 GPR antenna

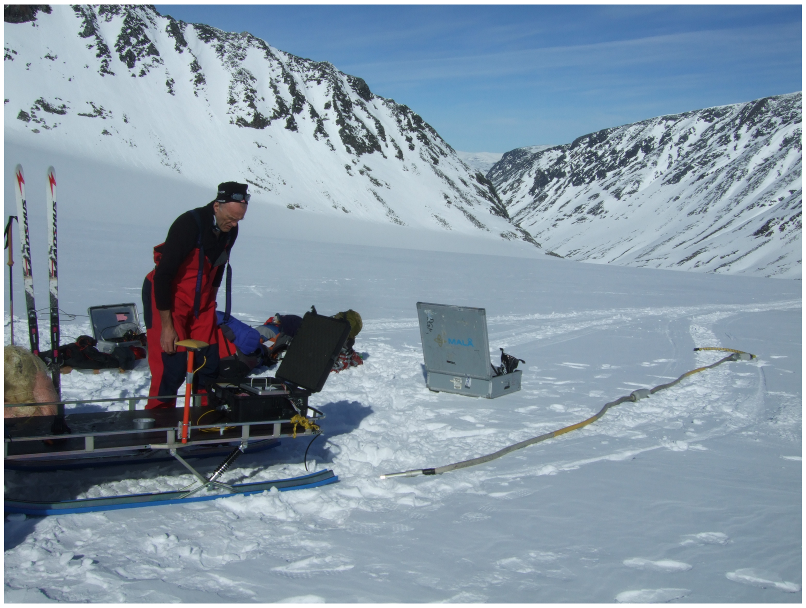


Figure § 2.1: GPR RTA 50 MHz antenna and DGNSS rover towed by snowmobile.

The ice thickness of both Hellstugubreen and Storbreen was obtained along GPR transects during the field work of April 2014. The thickness for both glaciers was measured earlier (see section **??**) using a 10 MHz antenna. As for the field work in April 2014, a 50 MHz MALÅ *Rough Terrain Antenna* (RTA) was chosen (*Figure § 2.1*), since the main objective was to observe a potential layering regarding the temperature distribution in the ice, and to identify the cold/temperate ice interface. As the choice of a frequency is a trade-off between vertical resolution and penetration depth of the signal (see section *GPR principles and applications in glaciology*), the higher frequency chosen for this field work resulted in an IRH from the bedrock not always visible on the GPR profiles.

1.2 GPR profiles

A set of six transects was obtained on Hellstugubreen (*Figure § 2.2a*). The antenna was dragged along the surface, towed by a snowmobile, and a separate sledge, a DGNSS rover was install to georeferencing the radar profiles (*Figure § 2.1*). The base station used as reference for the rover is located a few hundred meters away from the glacier front. The RES transects were obtained using a common-offset geometry with a distance of 4.2 meters separating the transmitter and receiver antennae. A sampling frequency of ∼510 MHz and a time window of ∼3110 ns were chosen, resulting in about 1900 samples per traces. While profiling, the snowmobile went at a constant speed of ∼1 m.s^{-1}, and the time interval between consecutive traces was 0.5 s. This gave about two records every meter along the profiles. The profiles distances range between ∼315 and ∼1430 meters. The elevation of the profiles spans from the glacier front (1490 m.a.s.l. in 2013) to ∼2080 m.a.s.l. The elevation range covered by the glacier obtained from the 2009 LiDAR data was 1484-2222 m.a.s.l., which makes a coverage of 80% of the total elevation range by the RES transects. No measurements were performed in the two upper cirques of the glacier, as both zones are heavily crevassed and therefore were inaccessible by snowmobile.

As for Storbreen, five profiles were obtained in the same manner, with a slightly different setup. The sampling frequency was set to ∼610 MHz, and a shorter time window of ∼2460 ns, as the thickness along the profiles to be mapped on Storbreen was expected to be generally smaller than the profiles obtained on Hellstugubreen. The number of samples per trace neared 1250. The RES profiles on Storbreen were between ∼150 and ∼1115 meters long (*Figure § 2.2b*), and the trace interval distance was about the same as for Hellstugubreen. The elevation of the profiles ranges from the glacier front (1438 m.a.s.l. in 2014) to ∼1630 m.a.s.l. Only the lower part of Storbreen was mapped during the field work of April 2014. The upper part was unapproachable with the snowmobile due to crevasses and steep topography.

2 Data analysis

2.1 Post-processing

Software and filtering

All the post-processing of the radargrams were effectuated in the 2D data-analysis module of the REFLEXW^{TM} Sandmeier software, version 7.5. The radargrams were first filtered horizontally by removing certain traces obtained while the snowmobile stopped, and which do not give additional information on depth variations of the IRHs. The start time of the records was then re-adjusted to remove the direct wave travel time. This delay is the time that the radar signal takes to travel from the transmitter antenna directly to the receiver antenna.

The next processing steps were to use 1D-filters on the radargrams, such as subtract-mean (dewow) and bandpass filters. The subtract-mean filter was used to remove some of the low frequency noises. This filter affects each traces of the profile independently. On each trace, it computes a running mean for each value and for a given time window. This mean is then subtracted from the center value of the time window. A time window of 20 ns was chosen, as one principal period of the radar signal is a suitable value (Sandmeier, 2014). The bandpass frequency filter aims to improves the signal-to-noise ratio by suppressing unwanted frequencies (noise) from the traces, that differ too much from the center frequency of the signal (50 MHz). A low-cut frequency and a high-cut frequency were defined, and outside the interval of these frequency, the frequency spectrum was set to zero.

Finally, a gain filter was used on the radargrams to improve the readable signal at depth. Indeed, as the signal penetrates deeper in the ice, the electromagnetic energy is dissipated into heat, which causes a loss in signal strength. It is to limit this effect that the energy decay filters come in handy. First a mean amplitude decay function is computed automatically by the software from all the traces. The traces are then corrected by dividing all sample values by the values of the decay function for the corresponding depths.

Time-depth conversion

A mean velocity of 168 m.μ s^{-1} was used to convert the two-way time return of the signal into depth. Neither a *Common Midpoint* (CMP) analysis, nor comparisons of radar and boreholes measurements were done on the field to determine the propagation velocity of the signal in the ice. The choice of the mean value was based on numerous previous studies (Glen and Paren, 1975; Murray et al., 1997; Pettersson et al., 2003; Navarro et al., 2005; Urbini et al., 2006). The constant velocity for the time-depth conversion assumes that the medium in which the signal propagates is homogeneous regarding its dielectric properties.

Digitizing and visualisation

Once the coordinates of the traces were defined based on the DGNSS measurements, the IRH matching to ice/bedrock interface was picked manually on the radargrams, every 2-10 meters. The points were then exported into pickfiles (*.pck) at the ASCII format, and imported into Quantum GIS for analysis.

Errors estimates in ice thickness

As mentioned previously, the use of a constant propagation velocity for the time-depth conversion relies on the homogeneity of the investigated medium, regarding its dielectric properties. This method therefore assume the absence of a snow/firn layer at the surface. Moreover, the radio-wave velocity is very sensitive to the water content as the relative permittivity of ice and water respectively differ by more than one order of magnitude (Moore et al., 1999; Pettersson et al., 2004; Navarro and Eisen, 2009). Since the water content may vary widely in space, with time and depth in polythermal glaciers (Jania et al., 1996; Murray et al., 2000; Pettersson et al., 2003; Bingham et al., 2005; Irvine-Fynn et al., 2011), the assumption of a constant propagation velocity may be inaccurate for Hellstugubreen and Storbreen (see *Temperature distribution and thermal regimes*). As an example, Benjumea et al. (2003) show that a change of 1% in the water content results in a variation by ∼3% of the propagation velocity of the signal.

As mentioned above, the calculation of the ice thickness depends both on the two-way time return and the propagation velocity of the signal. The error propagation of the ice thickness resulting from errors in the electromagnetic signal velocity and the two-way travel time can be estimated using the following equation (Navarro and Eisen, 2009) :

$$e_{IT} = \frac{1}{2}\sqrt{(\tau^2 e_v^2 + v^2 e_\tau^2)} \tag{§ 2.1}$$

where τ is the two-way time return, v is the propagation velocity, and e_τ and e_v are the error estimates from the two-way travel time and the propagation velocity respectively. Considering the previous years GPR data, the maximum ice thickness to be expected along the profiles on both Hellstugubreen and Storbreen is about 180 meters. With an error estimate of the two-way travel of half a principal period of the signal (10 ns), and an error of 2% for the propagation velocity, the equation (§ 2.1) gives an error of ∼3.70 m in ice thickness at the thickest zones of the glaciers.

The theoretical vertical resolution of GPR antennae is about a quarter of the wavelength ($^{\lambda}/_{4}$) of the propagating signal (Sheriff and Geldart, 1995; Jol, 2009). In practice, however, half of the wavelength ($^{\lambda}/_{2}$) is a more sensible estimation of the range resolution (Navarro and Eisen, 2009), and can therefore be calculated using the following formula :

$$r \approx \frac{\lambda}{2} \approx 0.5 \cdot \frac{v_p}{f_c} \tag{§ 2.2}$$

where λ is the wavelength of the electromagnetic signal, v_p is the propagation velocity in the medium, and f_c is the center frequency of the antenna. With a propagation velocity of ∼168 m.μ s^{-1} in ice and a center frequency equal to 50 MHz, the range resolution expected is ∼1.7 m.

Regarding the horizontal resolution, as no migration methods were performed on the radargrams from April 2014, its value is dependent on both the wavelength and the depth of the IRHs. The horizontal resolution of the non-migrated radar profiles is determined by the footprint of the radar beam, the also called *first Fresnel zone* (Navarro and Eisen, 2009). The radius of the *first Fresnel zone* can be calculated with the following formula (Robin et al., 1969) :

$$r_F = \sqrt{\frac{\lambda z}{2} + \frac{\lambda^2}{16}} \tag{§ 2.3}$$

where λ is the wavelength and z is the depth of the reflector. With $\lambda \simeq 3.4$ m and a maximum ice thickness of ∼180 m, the radius of the *first Fresnel zone* is about 17.4 m. This means that on the deepest zones of the glaciers, every reflector matching to the bedrock and visible on the radargrams resulted from the contribution of an area with a radius of ∼17.4 m. This may result in large uncertainties for the ice thickness measurements, especially where the bedrock topography is steep such as near the valley walls, at the glacier margins (Moran et al., 2000; Jol, 2009). The study from Moran et al. (2000) pointed out that performing a three-dimensional migration method on GPR data may improve the depth accuracy by 36%.

Further errors of the ice thickness may result from different sources. The digitization process can be subjective and leads to uncertainties in ice thickness along the radar profiles. By comparing same profiles digitized several months apart, Pettersson et al. (2003) noted differences of ±0.25 m for the ice thickness. A crossover analysis is a common method to assess the uncertainties coming from vertical accuracy and digitizing (Pettersson et al., 2011; Navarro et al., 2014; Andreassen et al., 2015). However, owing to a bedrock rarely visible on the radargrams from April 2014, there was not enough crossover points available after digitization to perform this analysis. Finally, between profiles, the interpolation of the ice thickness (see *Thickness data interpolation*) also results in uncertainties, which are larger at greater distance from the profiles. No performance analysis was done to test the accuracy of the interpolated values.

The uncertainties for the RES measurements at Hellstugubreen from 2011 and at Storbreen from 2005-2006 are both estimated to be ±15 m, at the point measurements (Andreassen et al., 2015). According to the above errors and uncertainties assessment, along with previous studies on the ice thickness mapping of sub-polar glaciers (Björnsson et al., 1996; Pettersson et al., 2003; Andreassen et al., 2015), the errors on the ice thickness is estimated to be ±25 m for the measurements from April 2014, at both Hellstugubreen and Storbreen.

2.2 RES profiles from 2011 at Hellstugubreen

The ice thickness at Hellstugubreen was for the most part determined from the RES profiles from 2011 owing to the dense spatial coverage of the profiles (e.g. Andreassen et al., 2015, *Figure § 3.1*), and as the ice/bedrock IRH was not much visible on the radargrams from 2014, due to the use of a higher frequency antenna for this year. The transects from 2011 were corrected for the melt from 2011 to 2014, and for the snowpack thickness from April 2014 (*Figure § 2.3*). The RES profiles from 2014 were used to compare with the results from 2011, and validate the correction methods.

Surface lowering derived GPS profiles and LiDAR data differentiation

During the field work in April 2014, the GPR profiles were georeferenced using the *Real Time Kinematics* DGNSS technique. The height of the GPR transects were therefore obtained while profiling. The height of the rover antenna attached on the sledge was subtracted from the height recorded by the rover antenna. The accuracy of the vertical coordinates is expected to be only a few centimetres, as the base station is located in the vicinity of the measurements.

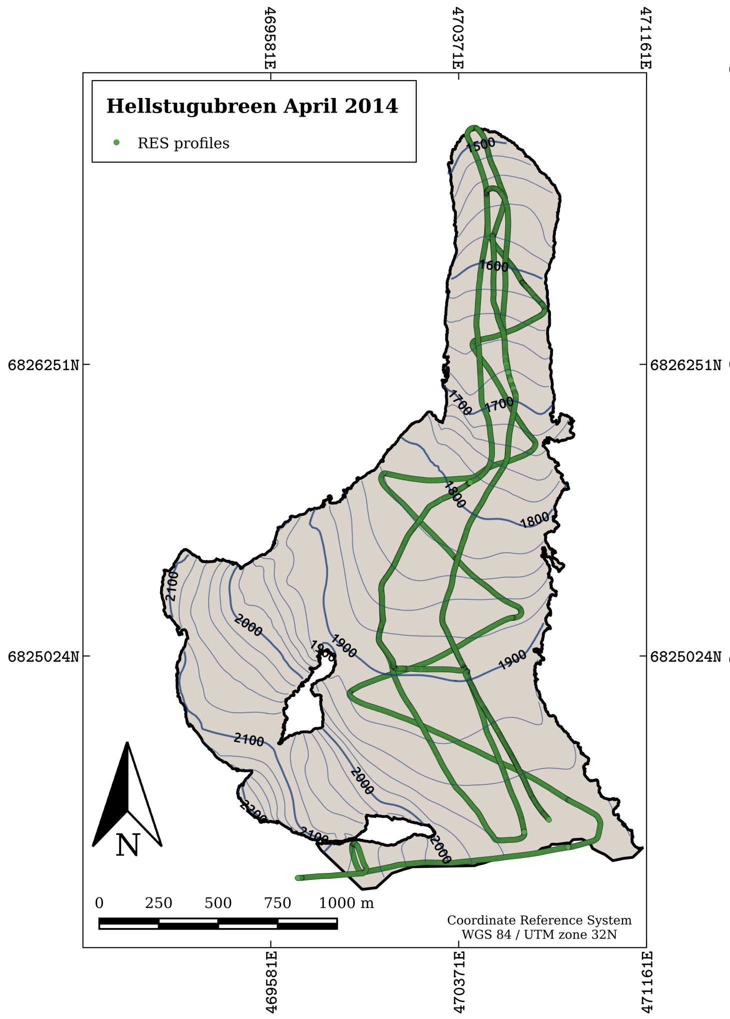
The thickness of the snowpack from April 2014 had values ranging from ∼145 to ∼475 centimetres. Those values were derived from manual probings. 167 manual snow probings were effectuated in the same period as the GPR measurements, at elevations spanning from ∼1550 m.a.s.l. to ∼2100 m.a.s.l. The snow depth values were then interpolated in ArcGIS software, developed by ESRI. The interpolation was done using the Ordinary Kriging algorithm of the Spatial Analyst toolbox. A spherical model was chosen to fit the empirical semivariogram, with 12 lags of 100 meters each. The result is the snow depth map presented in *Figure § 2.3*. The snow depth map was then used to correct the elevation of the RES profiles, in order to obtain the height of the ice surface. However, in the higher parts of the glacier, it is likely that the snow probings values represents the snow depth to the firn surface, as the probings were intended to measure the snowpack thickness of the winter 2013-2014. The ice/firn surface elevation along the profiles was then compared with the elevation of the 2009 LiDAR data at the same locations. The difference between both datasets gives an estimate of the surface lowering experienced by the glacier between 2009 and 2014.

The surface lowering values estimated along the profiles were also interpolated using the Ordinary Kriging algorithm. A spherical model was also used to fit the empirical semivariogram, with 15 lags of 100 meters each. The interpolation resulted in the surface lowering map for the period 2009-2014, as shown in *Appendix A.2*. The calculated surface lowering ranges from ∼15.8 meters near the front, to an increase of the surface elevation (accumulation) by about 3.1 meters (*Appendix A.1*). The area that experienced an increase in surface elevation is situated near the ice divide between Hellstugubreen and Vestre Memurubreen.

Corrections of the RES profiles

In order to use the GPR profiles from 2011 to generate an updated ice map for 2014, the surface lowering between these two years needed to be subtracted from the the GPR measurements. The comparison between the ice/firn surface elevation profiles from April 2014 and the laser scanning from 2009 gives an estimate of the surface lowering between 2009 and 2014. The surface lowering presented in *Appendix A.2* can therefore not be used directly to correct the GPR profiles. Continuous mass balance measurements for the period 2009-2014 were obtained at stakes H13, H26 and H44. The stakes H13, H26 and H44 were located in September 2014 at 1570, 1687 and 1890 m.a.s.l., respectively. The surface lowering between May 2011 and April 2014 represent ∼71.2% of the total surface lowering experienced by the glacier at stake H13, between September 2009 and April 2014. The same calculations at stakes H26 and H44 resulted in percentage values of ∼75.1 and 63.0, respectively. The average value of ∼69.8% was used as a multiplying factor to correct the surface lowering map. However, the use of a constant correcting factor over the entire map has its limitations : it assumes (i) the net mass balance at any points with same elevations on the glacier is identical; and (ii) the net mass balance changes are synchronized and proportionally the same at any elevation.The GPR profiles from 2011 were then corrected by subtracting from each records the value of the resulting map at the corresponding locations. To obtain better results, one should use mass balance measurements at more stakes and to use a correcting factor that varies spatially. However, continuous mass balance measurements for 2009-2014 were only available at these three stake positions.

(a)



(b)

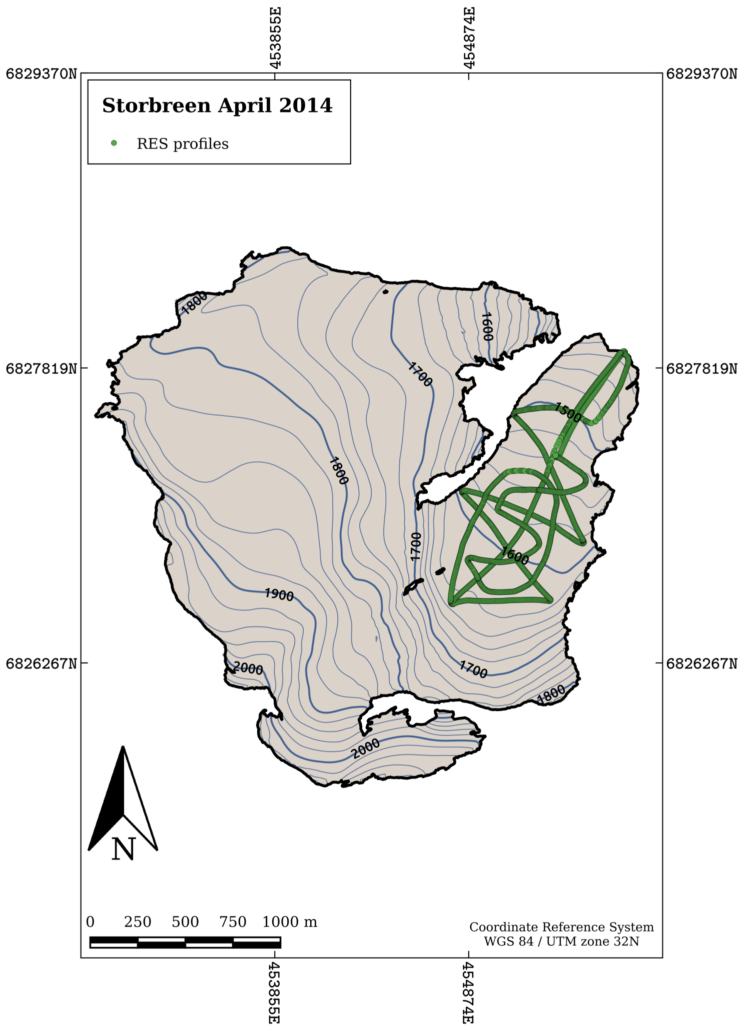


Figure § 2.2: RES profiles on Hellstugubreen (a) and Storbreen (b) covered in April 2014. The elevation contours and glacier outlines for both glaciers are derived from the 2009 laser scanning and orthophotos (data : NVE).

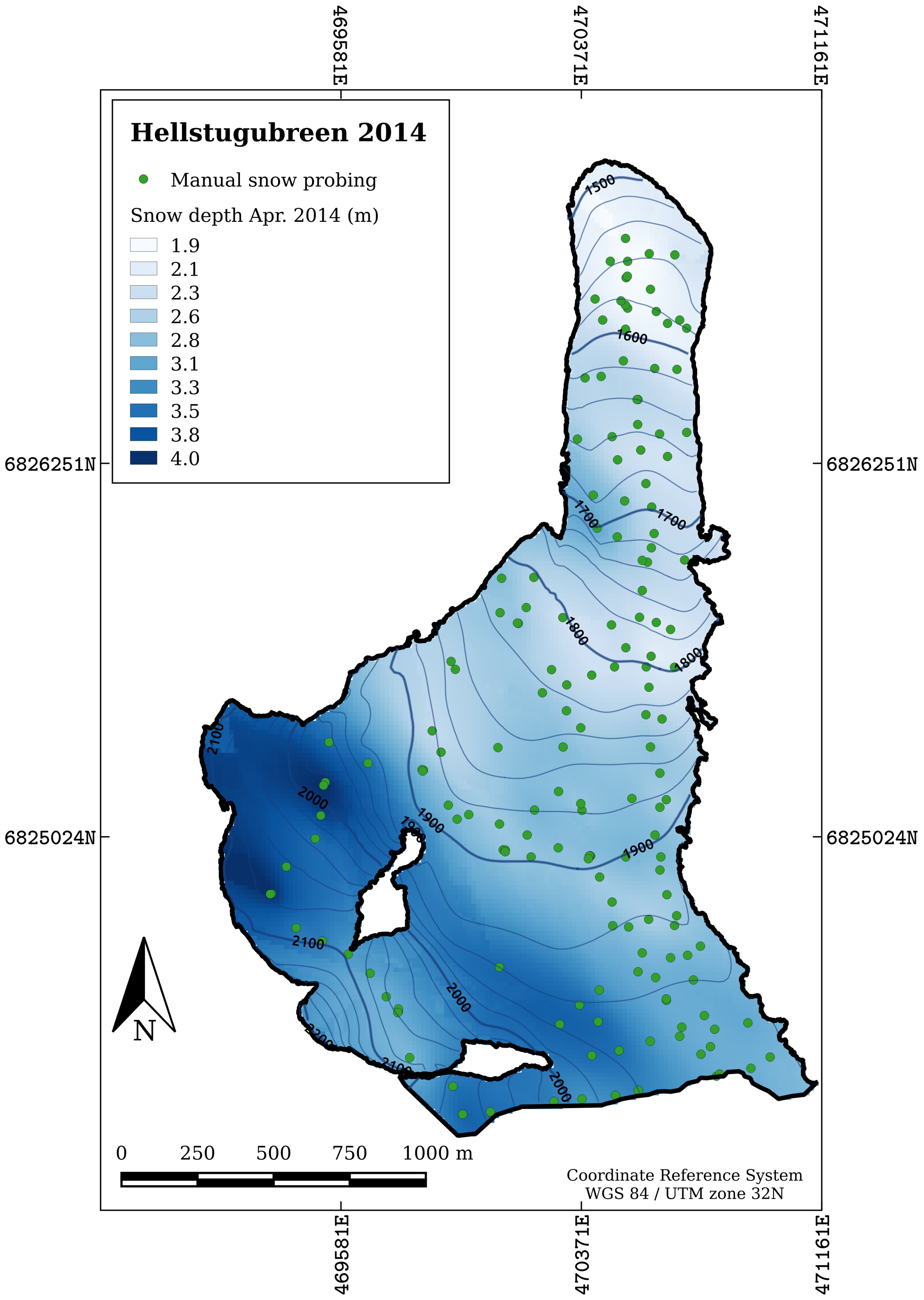


Figure § 2.3: Snow depth map at Hellstugubreen in April 2014, derived from manual snow probings. The probings were georeferenced with a hand-held GPS. The elevation contours and glacier outlines are derived from the 2009 laser scanning and orthophotos (data : NVE).

Thickness data interpolation

In order to create the ice thickness map of Hellstugubreen, the ice thickness was set to 0 along the glacier oultines from 2009, and along the glacier front georeferenced with DGNSS in September 2013. The corrected RES profiles and the glacier oultines were interpolated using Ordinary Kriging. A spherical model was used to fit the empirical semivariogram, with 15 lags of 100 meters. The resulting map is shown in *Figure § 3.1*, in *Ice thickness at Hellstugubreen*.

2.3 Ice thickness on Storbreen

Much scattering was observed on GPR profiles done on Storbreen during the field work of April April. This made difficult the manual picking of the IRH matching to the interface ice/bedrock. The interface was digitized at only 85 points. Moreover, the GPR measurements covered only the lower part of the glacier. It was therefore not possible to get an overview of the ice thickness of the whole glacier. Past RES measurements were performed in April 2005 and May 2006. The measurements cover also the upper parts of the glaciers. However, owing to technical problems during the field works, the ice thickness was only recorded at point locations, as opposed to continuous measurements along profiles. As such, the ice thickness was measured at about 130 points, over the period 2005-2006.

As the GPS profiles did cover only the lower parts of the glacier in April 2014, changes of the surface elevation could not be assessed for the whole glacier by comparing with the LiDAR data. In addition, the snow depth during the field work of April 2014 did not have a global coverage either. Therefore, no correction was applied on the RES measurements of 2005-2006, nor on the measurements from 2014. The ice thickness data for Storbreen is shown in *Figure § 3.4*, in *Ice thickness at Storbreen*.

Chapter B

Investigating the thermal regime and air temperature measurements

1 Hellstugubreen

1.1 Subsurface ice temperature profiles at Hellstugubreen

Figure § 2.4: Thermistors line and Hobo external temperature data logger mounted on stake H13.

The ice temperature in the subsurface was measured at two stake locations on Hellstugubreen. The first location was stake H13 (1570 m.a.s.l.), which is near the glacier front, and the second location was at stake H44 (1890 m.a.s.l.) (see *Figure § 2.7a*). One borehole was drilled at each site, using a steam drill. A depth approximating 14 meters was reached at both sites. One thermistor line was inserted in each borehole, allowing to to obtain the ice temperature at several depth levels, inferior to 13 meters. The thermistor lines were mounted on 8-conductor shielded cables. Seven NTC thermistors PR103J2 were installed on each line, at 2-meter intervals. Each thermistor was connected soldered to one conductor lead and the metallic shield, and was then protected by heat-shrink tubing. The thermistors have an accuracy of ±0.05°C within the temperature range measured. After the mounting of the thermistor, each of the sensors were calibrated for a temperature of 0°C. The factory-tested resistance value for this type of sensor is 32.65 kΩ, however after the mounting of the line, the resistance value observed at 0°C ranged from 32.4 to 32.7 kΩ. For the sensors that shown a shift between the observed value and the factory-tested value at 0°C, each point of the calibration curve (*Appendix C.1*) was corrected for the same difference. As such, the calibration curve was corrected for each sensor before converting the resistance value into a temperature value.

A weight was attached to the bottom of each line to ease the cable insertion, and the cable was then taped to the stake (*Figure § 2.4*). At stake H13 and at the time of setup, the lowermost and uppermost sensors were at a depth of ~12.6 m and ~0.6 m respectively. At stake H44, the lowermost and uppermost sensors were at a depth of ~12.1 m and ~0.1 m respectively. As the thermistors lines were mounted on shielded cables at Hellstugubreen, the ice temperature could only be obtained manually, by using a multimeter device. As such, ice temperature measurements have a low temporal resolution. At stake H13 the resistance of each thermistor was measured seven times between the 2nd of April 2014 and the 19th of September 2014. At stake H44 and for the same period, the resistance of each thermistor was measured eight times. From the resistance read with the multimeter device, the temperature value was directly obtained from the calibration curve. As the calibration curve was a set of points and not a continuous function, if the resistance value read on the multimeter fell between two known points, the temperature was approximated by a linear interpolation between both neighbouring points as shown in *Figure § 2.5*. This is not a bad approximation as the calibration curve is nearly linear within the temperature range of measurements (*Appendix C.1*). The length of the cable for the tape on the stake to the first sensor was known for both thermistors lines. The distance between sensors on the same line was also a known variable. Therefore, for each measurements, the length of the cable from the tape to the ice surface enabled to obtain the depth of the sensors at the time of measurements. This assume the thermistors lines remained straight and vertical in the ice, which goes against the presence of significant ice deformation in the subsurface.

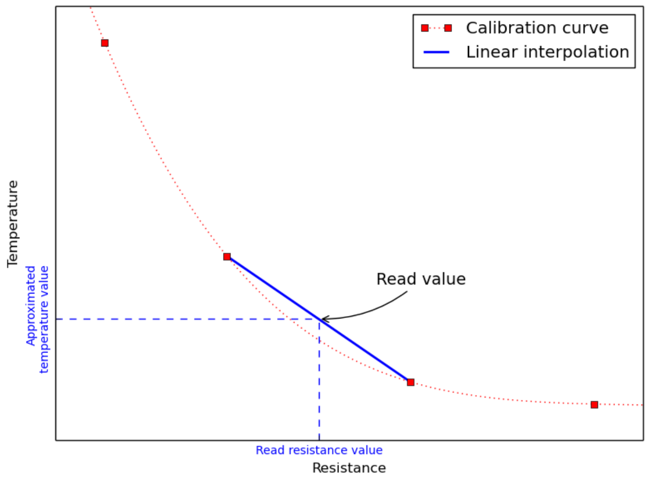


Figure § 2.5: Temperature approximation using a linear interpolation on the calibration curve of the NTC thermistors.

1.2 Air temperature

The air temperature was also measured at stakes H13 and H44. At each location, one HOBO Pro V2 2x External Temperature data logger was installed on the stake. Two wires were connected to the loggers, with a temperature sensor at the other wires end. On of the sensors was set as far down in the borehole as possible. At both stakes, the restricted length of the sensor-logger wires and the thickness of the snowpack at the time of setup did not give enough reach to the sensor for being installed in the ice. The other sensor connected to the data logger was inserted in a radiation shield and mounted on the stake (see *Figure § 2.4*). At the time of setup, the height of the radiation shield above the surface was 87 cm at stake H13 and 53 cm at stake H44. However, the height changed rapidly over the period of measurements with the fluctuations of the surface level, associated with melt processes and snow accumulation. From the 2nd of April to the 19th of September 2014, the data loggers recorded the air temperature at 30-minute intervals, with an accuracy ranging from ±0.2 to ±0.3°C and with a resolution inferior to 0.05°C (see *Appendix C.3*).

2 Storbreen

2.1 Subsurface ice temperature at Storbreen

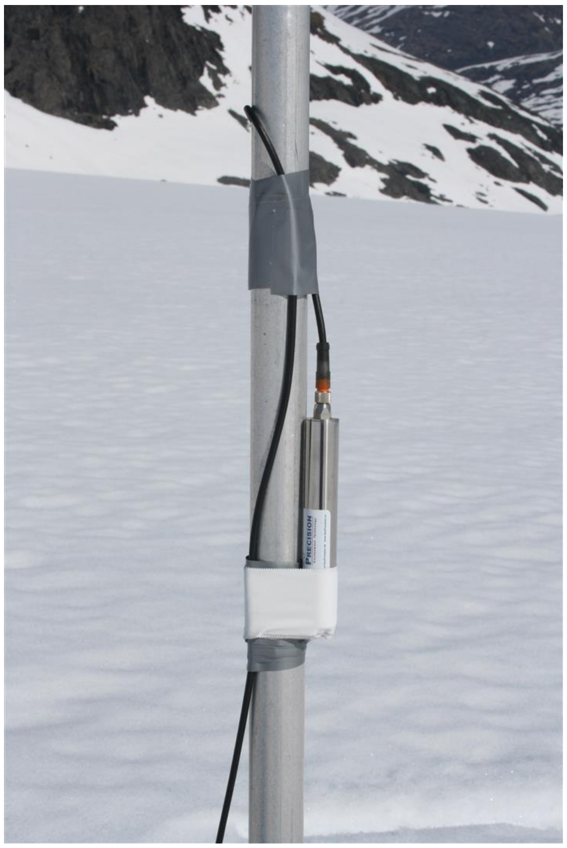


Figure § 2.6: Digital thermistors string and GeoPrecision M-Log5W data logger mounted on stake S2.

On Storbreen, the subsurface temperature was measured at only one location. The ice temperature was recorded using a digital thermistor string connected to a GeoPrecision M-Log5W data logger (*Figure § 2.6*). The advantage of a digital thermistor string is that it records automatically data at a predefined time-interval, as opposed to the manual temperature measurements on Hellstugubreen. The data could then be collected when needed. The thermistor string was mounted on stake S2 on the 3rd of April 2014, but the data logger was programmed and mounted only on the 20th of May. The stake was located at 1527 m.a.s.l. in September 2014, at 400 m from the glacier front (*Figure § 2.7b*). On the 18up of September, an attempt to move the data logger on a neighbouring newly drilled stake, the thermistor string was severed, and therefore no temperature data are available beyond this time. This resulted in about 4 months of ice temperature data, at 2-hour time intervals and for various depth levels. The digital thermistor string was ready mounted with ten sensors. These sensors measure temperature with an accuracy of ±0.25°C and with a resolution of 0.065°C. The uppermost sensor was at 3 meters from the logger, the second sensor was at 3 meters from the first one, the third sensor at 2 meters further away on the line, and all the remaining sensors were at 1-meter interval from each other. Owing a snowpack with a thickness of ~2.58 m at the time of setup, the uppermost sensor was not in the ice when the thermistor string was inserted in the borehole. At the end of the period of measurements and as melt processes occurred, a second sensor was out of the ice.

Modelling the depth changes of the sensors

As the ice temperature measurements at Storbreen had a much better time resolution than the measurements at Hellstugubreen, another method was chosen to update the depth of the sensors in the ice. The depths were updated every day for the whole period of measurements, using a *Positive Degree Day* (PDD) melt model (see *Appendix D.2*). The degree day melt factor was computed using the on-site daily mean air temperature (see *Air temperature*). The model defined *Control Periods* (CP), which are periods between two consecutive field observations during which the ice surface was not snow covered. For each CP (*Control Periods*), the PDD (*Positive Degree Day*) values were summed, and the total amount of ice melt computed from the field observations (stake readings). The melt factor was obtained by dividing the amount of melt by the sum of the PDD values. The melt factor happened to be slightly different between CPs. For this reason, each CPs kept its own melt factor in the model. For the periods defined between two field observations, where the ice surface was overlaid by snow for one or both observations, the melt factor used was the average of the factors computed for all CPs. The mean melt factor was computed giving a weight to each CP directly proportional to the number of days of the CPs. This was based on the assumption than the longer a CP is, the less likely to be error-prone the calculation of the melt factor is. For each day, the amount of melt could then be calculated by multiplying the daily mean air temperature by the melt factor. It is assumed that no melt occurred for the days with a negative mean air temperature. Finally, the depths of the sensors was then updated every day by subtracting the amount of melt from the previous day depths.

In the model it was likewise assumed that no ice melt occurred if the surface was snow covered. A special procedure was therefore used for the periods defined between two consecutive field observations, which one them was done when the ice surface was snow cover. If there was no ice ablation, the depth of the sensors did not need to be updated. If there was ice ablation, the depth of the sensors was updated every day, starting from the field observation where the ice surface was snow-free towards the one where the surface was snow covered, until reaching the total amount of melt for the period. When the total amount of ice melt is reached, the ice surface is considered to be snow covered, and therefore no ice melt occurs.

Estimating the ice surface temperature

In order to have a continuous ice temperature profile over the period of measurements, and starting from the surface to the lowermost sensor, the surface temperature needs to be known. As the thermistors string melted out, it happened that a sensor was exactly at the ice surface level, but this rather seldom and never lasting. To obtain information about the ice surface temperature, a temperature value was estimated from the other temperature values of the profile. For each temperature profile (every other hour), a second degree polynomial function was fitted to the data. The ice surface temperature was estimated by extrapolation of the polynomial function and reading the value for a depth of 0. A second degree polynomial was chosen to be able to represent diurnal temperature changes in the near surface, to a certain extent. Diurnal temperature changes are likely to happened if the ice surface is not snow covered or if there is only a thin snowpack (see *Appendix D.1*). No higher degree polynomials was used for the estimation method, as they are prone to much divergence outside the observation range due to *Runge's phenomenon*. Lastly, if the estimation method resulted in a positive ice surface temperature, this value was set to 0.The maximum positive value estimated was around 0.076°C.

2.2 Air temperature

At stake S2 on Storbreen, the air temperature was measured by the GeoPrecision data logger itself (*Figure § 2.6*). It this temperature dataset that was used to compute the PDD for the melt model in *Modelling the depth changes of the sensors.* The logger recorded air temperature from the 20th of May to the 18th of September 2014, at 12-hour intervals. The air temperature was measured at 4:00 in the morning and at 4:00 in the afternoon. An *Automatic Weather Station* (AWS) was also located in the glacier, about hundred meters away from stake S2. It measures air temperature at two different levels, as well as other climatic variables and snow surface parameters such as humidity, albedo, wind speed and direction, solar radiations... The measurements are effectuated every few minutes and are averaged every 30 minutes. The air temperature measurements from the AWS were not used in this work as there was no time for the necessary pre-processing of the data, such as corrections for radiative heating of the sensors.

3 Mapping the Cold-temperate transition surface with GPR

The last focus of the thesis regarding ice temperature was to map the internal layering and particularly the *Cold-temperate Transition Surface* (CTS) of both Hellstugubreen and Storbreen. It is common to use *Ground Penetrating Radar* to map the CTS (*Cold-temperate Transition Surface*) of polythermal glaciers (Björnsson et al., 1996; Jania et al., 1996; Moore et al., 1999; Pettersson et al., 2003). The principles of this method is based on the behaviour of the radio-wave that propagates in a medium that has inhomogeneities regarding dielectric properties. As mentioned in *GPR principles and applications in glaciology*, the lower the center frequency of the GPR antenna, the larger is the depth of investigation. For glaciological studies, the depth of investigation also depends on the conductivity and dielectric constant of the ice. The low conductivity of ice enables the electromagnetic signal to propagate without much attenuation, and GPRs are therefore suitable tools for great depths of investigations (Plewes and Hubbard, 2001). On temperate glaciers, the center frequency of the GPR antenna commonly used is ∼15 MHz or lower (Watts and England, 1976; Sætrang and Wold, 1986; Kennett et al., 1993; Navarro et al., 2005). Higher frequencies for temperate ice leads to strong scattering and signal attenuation, owing to water inclusions (Watts and England, 1976; Navarro et al., 2005). *Ultra High Frequencies* (UHF) are often use for cold ice to improve the vertical resolution of the RES measurements. The absence of liquid water in cold ice and its relatively homogeneity regarding its dielectric properties makes it transparent to the radar signal. On polythermal glaciers, the use of UHF (*Ultra High Frequencies*) allows to investigate the depth of the CTS, which is thought to be where much scattering occur, owing to the liquid water content in temperate ice. Using lower frequencies enables to see the ice/bedrock interface, otherwise masked by the scattering at the CTS with UHF (Björnsson et al., 1996; Moore et al., 1999; Pettersson et al., 2003).

On Hellstugubreen and Storbreen, the center frequency of the GPR antenna used to observe thermal layering in the ice was the same (50 MHz) as the one used for ice thickness measurements. The same processing steps were done on the radargrams as for the ice thickness, except for the digitizing. The assumption of a constant velocity of propagation results in less uncertainties than for the ice thickness measurements, as the medium overlaying the CTS is mostly cold ice. The uncertainties on the depth measurements of the CTS are expected to be less significant than for the ice thickness measurements. Indeed both vertical and horizontal accuracies depends on errors in the propagation velocity and depth of investigation (see equations (§ 2.1) and (§ 2.3)).

Figure § 2.7a and *Figure § 2.7b* show the GPR profiles on Hellstugubreen and Storbreen, along which the CTS was digitized.

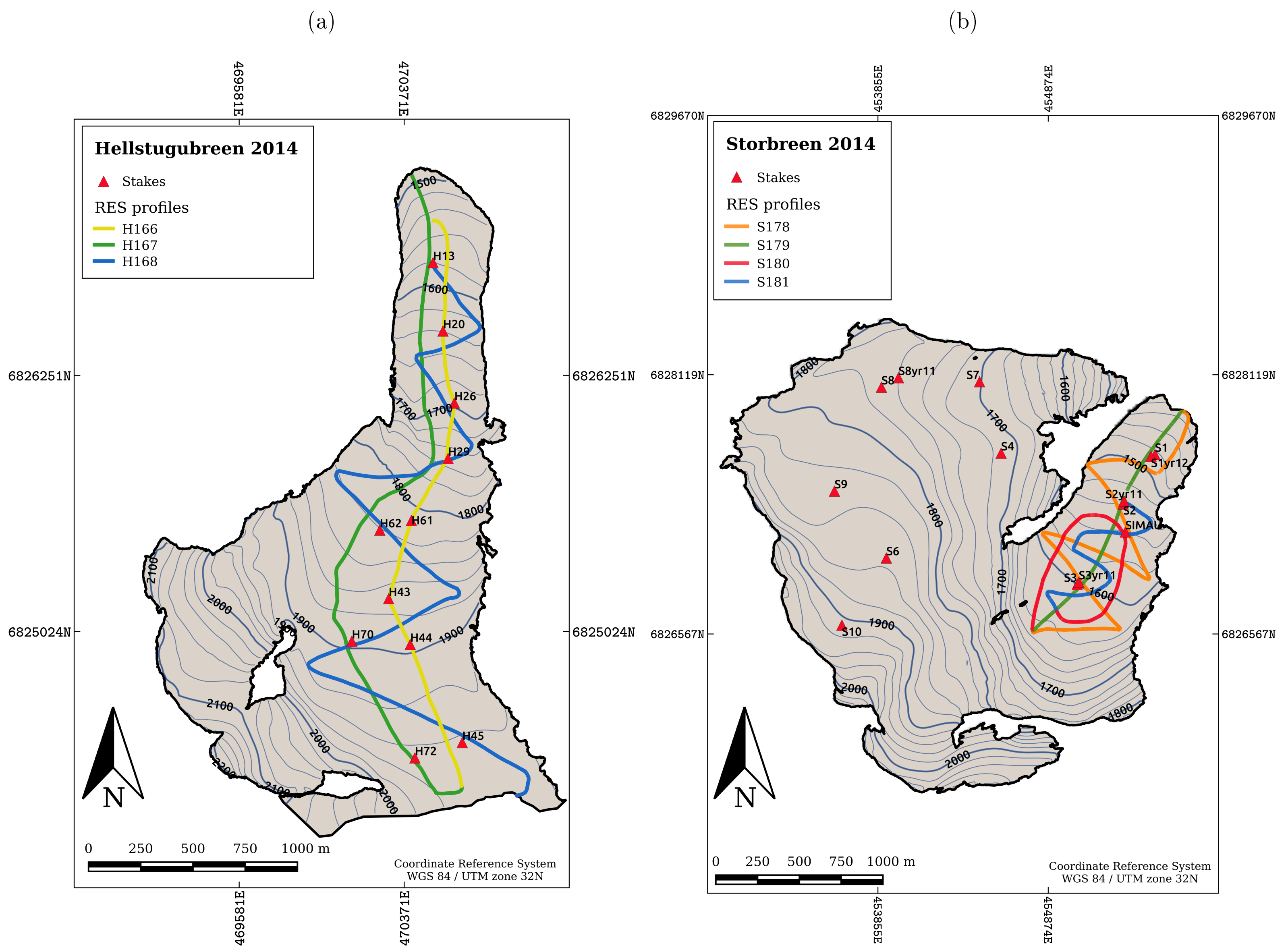


Figure § 2.7: RES profiles from 2014 where the CTS was digitized, at Hellstugubreen (a) and at Storbreen (b). The elevation contours and glacier outlines for both glaciers are derived from the 2009 laser scanning and orthophotos (data : NVE).

Chapter C

Ice flow velocity

The mapping of Hellstugubreen and Storbreen was also related to ice dynamics. As the datasets available for both glaciers were not all the same and of same quality, this chapter is also divided in two parts, one for each glacier.

1 Hellstugubreen

1.1 DGNSS measurements at stake positions

One of the techniques enabling to measure the surface velocity of glaciers is the repeated surveys of stakes drilled at the surface. On Hellstugubreen, the first stake surveys with accurate georeferencing from DGNSS started in September 2009. Horizontal velocity changes were already studied by comparing accurate data from the period 2009-2012, with triangulation measurements performed in the 1940s and 1960s (Pay, 2014). The quality of the data depends on the continuity and the density of the measurements. On Hellstugubreen, the stakes position was recorded at the beginning of almost melt season, usually during the first two weeks of May. The positions were measured a second time in September, at the end of melt season. A few years have also measurements in August. For each field work between 2009 and 2013, it was the locations of 5 to 13 stakes that were recorded. In 2014, the stake network density was improved, and 21 stakes were georeferenced on the 16th of September. To computed the surface velocity at stakes locations, the following formula was used :

$$V_{surface} = \frac{\sqrt{(x_{end} - x_{start})^2 + (y_{end} - y_{start})^2}}{\Delta t} \qquad (\S\ 2.4)$$

where x_{end} and x_{start} are the Easting coordinates of two consecutive measurements, y_{end} and y_{start} are the Northing coordinates, and Δt is the time lapse between the measurements. The Easting and Northing values were given in meters, and were *Universal Transverse Mercator* (UTM) coordinates, in the zone 32V. The Δt was computed in second, but the final velocity values were in m.yr^{-1}. It is also to be noted that the equation (§ 2.4) does not include any vertical component, as such the values calculated using this formula reflects the horizontal surface velocity. In addition, the equation does not account for changes in the surface velocity between consecutive measurements, this results in average values of the velocity between measurements. Lastly, the formula is based on the assumption of a horizontal linear displacement, which means that it does not account for flow divergence. In other words, when two different positions of a same stake and from two different times are compared, it is assumed that the stake displacement was effectuated along the linear and minimum distance between both positions. This assumption is not bad as the stakes surveyed were not close from the glacier margins, where divergence of the ice flow can be significant. Furthermore, the time interval between two consecutive measurements at each stake was relatively short, which limits the occurrence of large errors due to flow divergence for the calculation of the stakes displacement.

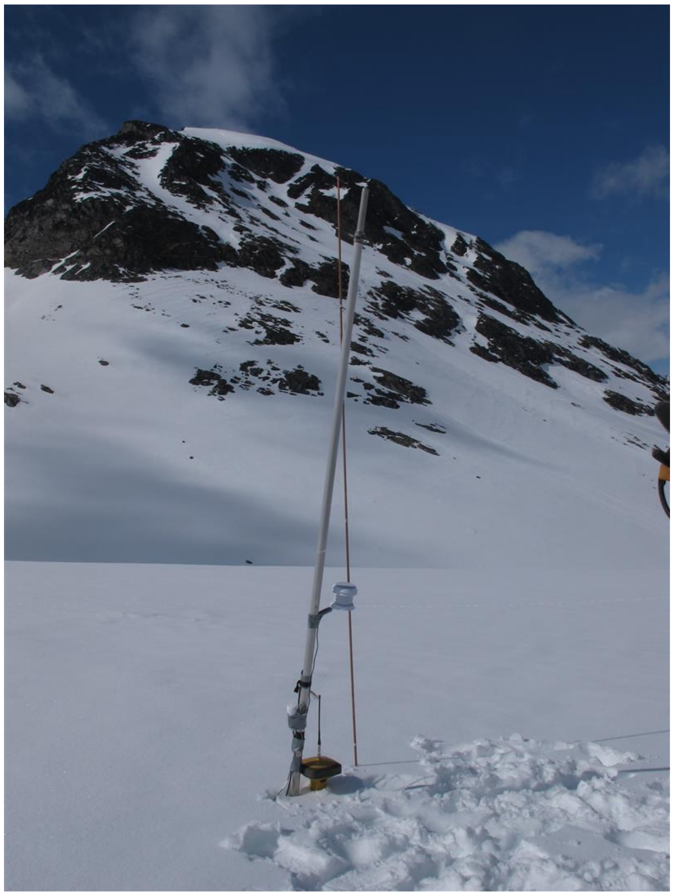


Figure § 2.8: Stake georeferencing on Hellstugubreen using DGNSS. The accuracy of the measurements may be altered by multipath effects.

To georeference the stakes positions, the rover antenna was either upstream, on top or downstream of the stake. The antenna was usually placed downstream as the stake was too high. In case of a downstream location, the distance from the antenna to the stake was subtracted from the minimum displacement (numerator term in equation (§ 2.4)). In case of an upstream location, the distance between the antenna and the stake was added to the minimum displacement. These corrections assume that the flow lines follow the surface topography gradient. For newly drilled stakes, the antenna was positioned in the hole before inserting the stake, or over the stake. As such, their positions did not require any correction. The scripts in *Appendix F.2* and *Appendix F.2* enable to enter the stakes coordinates, to apply the necessary corrections, and to compute the horizontal velocities for each stakes and between all consecutive measurements.

As the base station was located only a few hundred from the glacier front, the stakes positioning is expected to have an accuracy of a few centimetres only. However, the position of the rover antenna near the stake as shown in *Figure § 2.8* may results in georeferencing errors due to multipath effects (King and Watson, 2010; Nilsson, 2011). Indeed, the satellite signal received by the rover antenna may be reflected by the stake before or after reaching the DGNSS antenna. Thus, the accuracy of the stakes positioning may be affected by these effects. No multipath mitigation was done on the measurements.

1.2 Ice surface velocity interpolation

The observed surface velocity depends on numerous variables. The surface velocity can be estimated by the following formula (Cuffey and Paterson, 2010) :

$$u_s = u_b + \frac{2A}{n+1} \cdot \tau_b^n \cdot H \tag{§ 2.5}$$

where u_b is the basal sliding velocity, A is the creep flow parameter (see *Ice temperature and Glen's flow law*), n the empirical exponent with a mean value of 3, τ_b is the basal shear stress and H is the ice thickness. Cuffey and Paterson (2010) shows that it is a good approximation to assume that :

$$\begin{cases} \tau_b = f' \cdot \tau_d \\ \tau_d \approx \rho g H \alpha \end{cases} \tag{§ 2.6}$$

where τ_d is the driving stress component of the ice flow, f' refers to a number usually of order one, ρ is the ice density (~917 kg.m^{-3}), g is the gravitational acceleration constant (~9.81 m.s^{-1}), and α is the surface slope in radian.

In equation (§ 2.5), the basal sliding component is a variable difficult to measure, as the base of glaciers is usually not directly accessible. The remaining terms of the equation are either constants or calculable. Amongst the latter variables, the basal shear stress can be estimated as it require the knowledge of the ice thickness, which is available on Hellstugubreen from the corrected GPR profiles from 2011, and the surface topography which is easily calculated from the LiDAR data from 2009. The value of the creep flow parameter A was hard to estimate over the whole ice thickness, as it depends physical and chemical properties of the ice (Cuffey and Paterson, 2010), and this information being not available throughout the entire mass of the glacier. The interpolation of ice surface velocity data was performed using the autocorrelation velocity dataset itself, together with cross-correlations between the velocity data and the ice thickness on the one hand, and between the velocity and the surface slope on the other hand. In order to include the relationships between the surface velocity and these two parameters, the cokriging algorithm was used in ArcGis to interpolate the velocity data. For the autocorrelation of the velocity values at the stake location, a stable model was used to fit the empirical semivariogram, with 12 lags of each ~130 meters. Exponential models were used to fit the semivariograms resulting from the cross-correlation between the surface velocity and the ice thickness, and between the surface velocity and the surface slope. The ice thickness data used for the interpolation is shown in *Figure § 3.1*, in section *Ice thickness at Hellstugubreen*. Likewise, the surface slope map used in the cokriging algorithm is presented in *Appendix G*. As the density of the stake network was much better for

2013 and 2014, only the horizontal velocity values were interpolated. The interpolation resulted in an averaged surface velocity map for 2013-2014.

2 Storbreen

2.1 DGNSS measurements at stake positions

Repeated surveys of stakes were also performed on Storbreen. The first stakes positions were recorded with DGNSS in September 2006 (Andreassen et al., 2007), and stake measurements are available until the 18th September 2014. The stake network density is less than on Hellstugubreen, with 5 stakes georeferenced at minimum during one field work, and up to 14 stake positions recorded in 2014. Most of the stakes on Storbreen are georeferenced at least once a year. About half of the stakes of have their position recorded twice a year since 2012, with both measurements done between the beginning of August and the end of October.

The same method to estimate the horizontal surface velocities was used on Storbreen, by calculating the minimum linear surface displacement between consecutive field observations. As the stakes positions was mostly measured during the end of the summer or the beginning of the autumn, the equation (§ 2.4) gives an estimate of the annual mean horizontal velocity for each stake. The same corrections were also applied on the final velocity values, to account for the rover antenna offset position, relatively to the stakes.

As the stake network had a lower spatial resolution than on Hellstugubreen, no surface velocity map was generated for Storbreen. In addition, the density and amount of the ice thickness data on Storbreen was also less than on Hellstugubreen. The relationship between the ice thickness and the surface velocity would therefore have given poorer results using the cokriging algorithm when interpolating the velocity data.

2.2 Subsurface deformation rate

The subsurface deformation rate was the last element studied on Storbreen. The ice temperature in the subsurface measured at stake S2 and at a high temporal resolution enabled to account for its effects on the ice viscosity, and therefore on the deformation rate (see *Ice temperature and Glen's flow law*). The ice deformation rate changes with depth was estimated based on the assumption that the glacier deforms in simple shear, in the same way as a laminar flow. For a simple shear deformation, the z-component of the deformation velocity is 0 and the only deviatoric stress component is τ_{xz}. The creep relation is then as follows (Cuffey and Paterson, 2010) :

$$\frac{1}{2}\frac{du}{dz} = \frac{1}{2}\frac{\partial u}{\partial z} = A \cdot \tau_{xz}^{n} \qquad (\S\ 2.7)$$

where u is the x-component of the deformation rate and z is the depth axis.

If we assume that the ice density is constant throughout the thickness of the glacier and following the equation (§ 2.6), the shear stress component increases linearly with depth and therefore we have :

$$\tau_{xz} = \tau_b \left[\frac{z}{H}\right] \tag{§ 2.8}$$

where τ_b is the value of τ_{xz} at the bed and H the total ice thickness. By substitution of τ_{xz} in equation (§ 2.7) with equation (§ 2.8), we have :

$$\frac{du}{dz} = 2\,A\,\tau_b^n \left[\frac{z}{H}\right]^n \tag{§ 2.9}$$

In order to compute the creep flow parameter A, Cuffey and Paterson (2010) give the following formula that account for effects of temperature and hydrostatic pressure which lower the melting point of ice :

$$\begin{cases} A = A_* \cdot \exp\left(-\dfrac{Q_c}{R} \cdot \left[\dfrac{1}{T_h} - \dfrac{1}{T_*}\right]\right) & \\ T_h = T + 7 \times 10^{-8} P; & T_* = 263 + 7 \times 10^{-8} P; \\ Q_c = Q^- \text{ if } T_h < T_*; & Q_c = Q^+ \text{ if } T_h > T_*. \end{cases} \tag{§ 2.10}$$

where A_* is a constant, Q_c is the activation energy for creep, R is the universal gas constant (8.314 J.mol^{-1}.K^{-1}), T is the ice temperature in Kelvin and where P is the pressure in Pascal (positive in compression). The prefactor A_* is the value of the creep flow parameter A for a temperature of -10°C. The recommended value 3.5×10^{-25}Pa^{-3}.s^{-1} was used for further computations (Cuffey and Paterson, 2010). The activation energy for creep for warm ice (Q^+) is equal to 115 kJ.mol^{-1}, and is equal to 60 kJ.mol^{-1} for cold ice (Q^-). The value of the activation energy changes at about -10°C.

For each temperature record (every two hours), the subsurface deformation was calculated by cumulative trapezoidal integration of the equation (§ 2.9). A constant value of ~120 kPa was given to the shear stress component at the bed (τ_b). This value was calculated with an ice density equal to 917 kg.m^{-3}, an ice thickness of 85 meters at the stake location, estimated from the GPR measurements from 2005-2006 (see *Figure § 3.4*, in *Ice thickness at Storbreen*), and a surface slope of 9 degrees (*Appendix G*), derived from the LiDAR data from 2009. The empirical exponent n was given a fixed value of 3. In order to use the trapezoidal rule, hundred points equally spaced were generated for each profile, with depth values ranging from zero to the depth of the lowermost sensor of the thermistor string. The temperature at each point was obtained from the thermistors, for which the depth was known. In case the depth value of a point was different from that of a sensor, the temperature was obtained by linear interpolation of the values recorded at both closest sensors. This gave a temperature value measured or estimated every ∼10.7 cm at the beginning of the measurements period, on the 20th of May 2014. On the 18th September of the same year, as the depth of the lowermost sensor was less and that the same number of points was used to apply the trapezoidal rule, a temperature value was measured or estimated for every ∼8.2 cm. For every temperature profile, the creep flow parameter A was calculated at all hundred points using the equation (§ 2.10). While integrating cumulatively the equation (§ 2.9), A was replaced by the calculated values and z by the depth assigned to each point. Finally, it was assumed that no deformation occurred at the surface, as the shear stress component is there equal to 0 (see equation (§ 2.8)).

Part § 3

Results and discussions

Chapter A

Ice thickness

1 Ice thickness at Hellstugubreen

The ice thickness map resulting from the interpolation of the corrected GPR profiles from 2011 is shown in *Figure § 3.1.* The interpolated values of the ice thickness range from 0 to ~177 meters. The thickest part is located at the ice divide on the southern part, between Hellstugubreen and the larger glacier Memurubreen, not shown on the map. The ice thickness decreases gradually towards the front. A local depression of the thickness appears between the elevation contour lines 1820 and 1880. The buffer-like area (value equal to zero) between the glacier contour lines and the thickness values superior to zero results from the interpolation algorithm. The ordinary kriging used to produce the ice map interpolated the thickness values recorded along the RES profiles from 2011, as well as the values of the contour lines equal to zero. The density of points along the contour lines is too high which gives a weight too important to the border lines in the interpolation process. The width of the buffer area depends on the distance between the RES profiles and the contour lines. The greater this distance, the broader the buffer area is. Similarly to the field work from April 2014, the ice thickness was not measured in the two upper cirques in 2011, for the difficult access with snowmobile.

2 Comparison of the ice thickness map with the RES measurements from 2014

The ice thickness map produced for Hellstugubreen (*Figure § 3.1*) was compared with RES measurements from 2014. To every thickness record from the 2014 RES measurements was subtracted the thickness value estimated/measured from the 2011 RES data. The ice thickness map was created with a spatial resolution output of 15 m as interpolation parameter. As such, if the RES records from 2011 and 2014 were separated by less than 15 m, the 2014 RES values were directly compared with the nearest 2011 RES measurements, to limit averaging artifacts from the interpolation algorithm.

The ice thickness was digitized at 564 points on the radargrams from 2014, excluding the points located in the 0 m buffer area (*Figure § 3.1*). The combined uncertainties from the

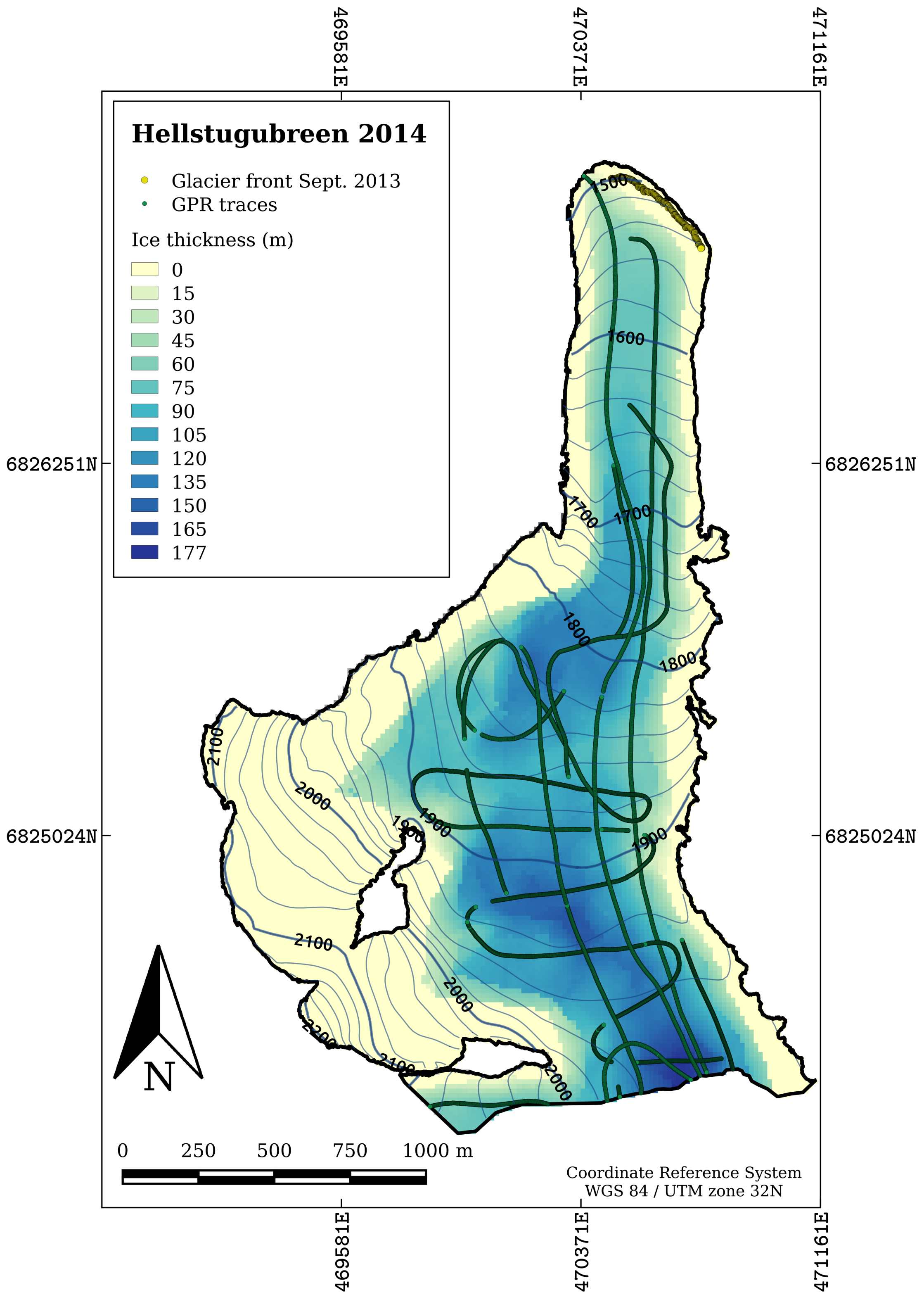


Figure § 3.1: Ice thickness map at Hellstugubreen for 2014, derived from GPR measurements conducted in 2011. The elevation contours and glacier outlines are derived from the 2009 laser scanning and orthophotos (data : NVE).

2011 RES (±15 m) and the 2014 RES (±25 m) records can explain ice thickness differences up to ±40 m for both years. 29 points (5.1%) showed a difference larger than the total uncertainty. Out of 29, 8 points overestimate the ice thickness from the 2014 RES, relatively to the ice thickness map. This make 21 points that underestimate the ice thickness relatively to the map. Accounting for all 564 point records, 435 points (77 %) present a negative difference value (relative underestimation of the measurements from 2014), 129 a positive value (relative overestimation of the measurements from 2014). The average of the absolute differences between RES records from 2011 and 2014 is 18 m.

3 Discussion

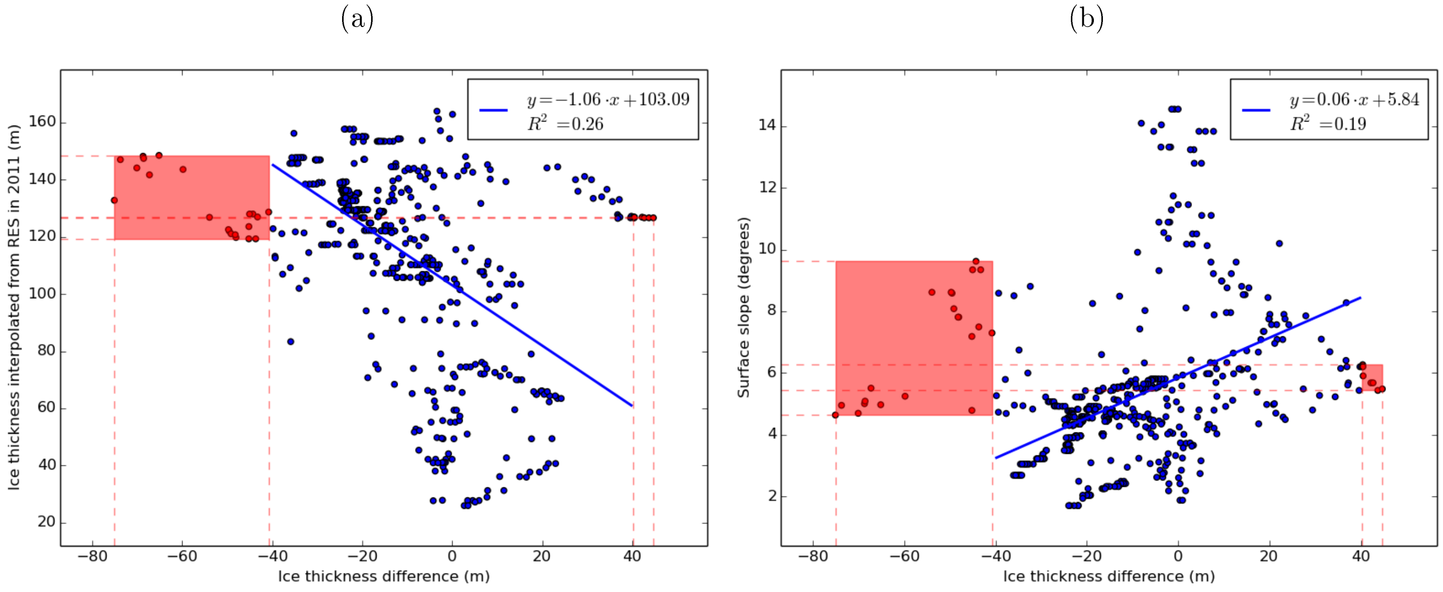


Figure § 3.2: Thickness differences from RES measurements done in 2011 and 2014 at Hellstugubreen. (a) shows the relationship between thickness differences and the ice thickness values from *Figure § 3.1*. (b) shows the relationship between thickness differences and the surface slope. In the red shaded zones are the points with thickness difference values that exceed the uncertainties expected from the RES measurements.

In order to explain the differences between the 2014 RES records and the ice thickness map, the relationships between the differences and four parameters was assessed. First, the thickness differences were compared with the interpolated thickness values from the 2011 RES measurements (*Figure § 3.2a*). A linear regression analysis shows that the higher the interpolated thickness values are, the more positive the thickness differences are. The relationship between both variables has however a low determination coefficient ($R^2 = 0.26$). The comparison between interpolated values and absolute difference values show that, generally, the greater the interpolated thickness values are, the larger the thickness differences are (*Appendix B*). It is also to be noted that all differences superior to the global uncertainty occur for an ice thickness larger than 120 m on the map.

The relationship between thickness differences and surface slope ($R^2 = 0.19$) shows that a steeper surface slope leads towards an overestimation of the 2014 RES records, relatively to the ice thickness map (*Figure § 3.2b*).

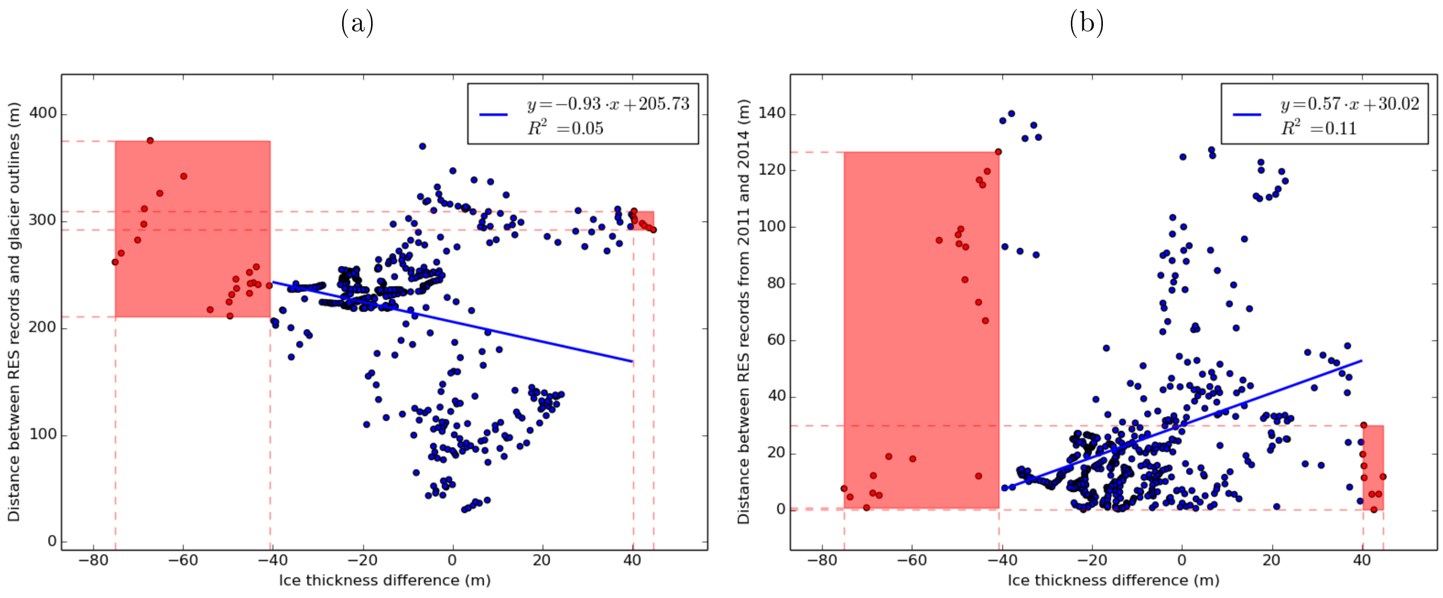


Figure § 3.3: Thickness differences from RES measurements done in 2011 and 2014 at Hellstugubreen. The distance between the 2014 RES records and the glacier oultines are plotted against the ice thickness differences in (a). (b) shows the relationship between thickness differences and the minimum distance between RES records from 2011 and 2014. In the red shaded zones are the points with thickness difference values that exceed the uncertainties expected from the RES measurements.

As mentioned earlier, the bedrock topography is usually steeper near the valley walls, at the glacier margins. For non-migrated radar profiles, the steep bedrock topography may lead to large ice thickness errors (Moran et al., 2000; Jol, 2009). Therefore, the relationship between thickness differences and the distance separating the glacier outlines and the 2014 RES measurements was also assessed (*Figure § 3.3a*). However, a simple linear regression shows a very low correlation between both variables ($R^2 = 0.05$). In 2014, the ice thickness was measured at closest $\sim$30 m from the glacier margin. The large errors expected with a steep bedrock topography may not occur at such distance. A direct comparison of the thickness differences and the bedrock slope did not give any correlation between both variables. The errors larger than the global data uncertainties happened to occur at a minimum of 200 m away from the glacier outline. This confirms that larger errors occur for a greater ice thickness, as the ice thickness values are more important away from the margins.

Finally, the distances between closest RES records from the years 2011 and 2014 could be another factor influencing the ice thickness differences. If a greater distance separating the measurements from both years lead to a large ice thickness difference, this would point out to the limits of the interpolation algorithm. However, the distance between RES records does not seem to be an explanatory factor for large ice thickness differences, as the relationship between both variable has a very low determination coefficient (*Figure § 3.3b*).

When interpolating the thickness data, a model was fitted interactively in ArcGIS to the empirical semivariogram. To further investigate errors resulting from the ordinary kriging interpolation, a cross-validation analysis or similar model validation technique should be used. The results from such validation technique would give a predictive accuracy of the model used, and therefore give a better insight on potential interpolation errors. Alternatively, the kriging algorithm could be compared with other geostatistical or deterministic methods, and look where the largest differences occur.

In order to improve the accuracy of the thickness measurements from April 2014, several options are available. First, a migration method should be used on the radargrams. For migrated radargrams, the horizontal resolution is no longer dependant on the depth and can be approximated by $\lambda/2$ (Welch et al., 1998). Instead of the initial horizontal resolution of ~35 m (see *Errors estimates in ice thickness*, equation (§ 2.3)), the horizontal resolution of the migrated radargrams would become ~3.4 m. A radio-wave velocity varying spatially would also be more appropriate for the time-depth conversion of the radar signal. As the propagation velocity depends on the ice water content (Benjumea et al., 2003), the hydro-thermal structure of Hellstugubreen may have large effects on the ice thickness measurements (see *Hellstugubreen*). To limit the effects of the spatial variations of the ice water content, one could perform local or regional Common-Midpoint measurements for a better estimation of the signal propagation velocity (Navarro et al., 2014).

4 Ice thickness at Storbreen

On Storbreen, as much scattering was observed on the radargrams from April 2014, the bedrock was almost not visible and therefore hard to digitize. *Figure § 3.4* presents the thickness data for Storbreen, obtained during the years 2005 and 2006. The ice thickness measurements were conducted at an elevation ranging from ~1450 to ~1890 m.a.s.l., corresponding to about 64% of the elevation range covered by the whole glacier. The maximum thickness recorded by the RES measurements was ~233 m, and was located on the upper parts of the glacier with a low topographic gradient. *Figure § 3.4* also shows the locations of the RES records from 2014 where the ice thickness was measured.

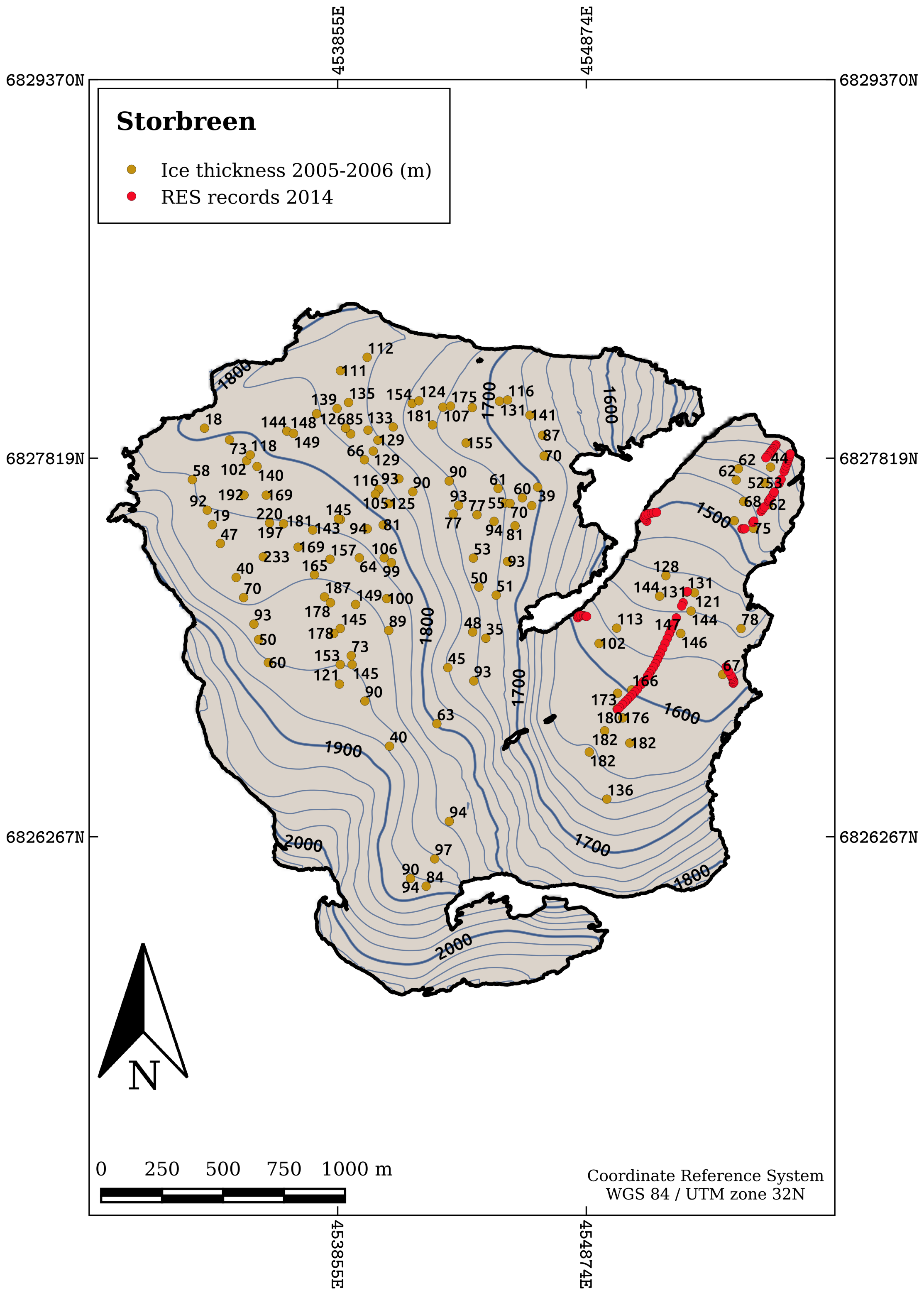


Figure § 3.4: Ice thickness at Storbreen, derived from GPR measurements performed April 2005 and May 2006. The red dots are the locations of the RES records from April 2014. The elevation contours and glacier outlines are derived from the 2009 laser scanning and orthophotos (data : NVE).

Chapter B

Temperature distribution and thermal regimes

1 Hellstugubreen

1.1 Subsurface temperature variations

On Hellstugubreen, the ice temperature variations in the subsurface was observed at stake H13 and H44. The measurement period started on the 2nd of April and ended on the 16th of September 2014. At the time of setup, the lowermost sensor was located at a depth slightly over 12 m, at both sites (see *Subsurface ice temperature profiles at Hellstugubreen*). Seven temperature profiles were obtained during the entire measurement period at stake H13, including the profile recorded right after the setup of the thermistor line. At stake H44, eight profiles were recorded over the same time period, also including the profile measured right after the thermistor line setup. *Figure § 3.5* show the ice temperature profiles measured at both stakes, with above the air temperature variations recorded by the HOBO data loggers every 30 minutes at both locations. The measured data used to plot the ice temperature profiles are presented in *Table § 3.1* and *Table § 3.1*.

In average over the the whole measurement period, the air temperature was 2.6°C warmer at stake H13 than at stake H44. With an elevation difference of 320 m between both stake locations, the mean temperature gradient is -0.81°C/100 m. Sudden temperature peaks appeared in the recorded data, which may result from measurement errors. The temperature high that occurred at stake H13 on the 10th of May at 3.30 pm is not seen at stake H44 and is certainly a measurement error from the HOBO data logger. The temperature reaches a peak of 20.5°C, while the temperature recorded 30 minutes earlier was 4.2°C, and the temperature recorded 30 minutes later was 3.0°C.

Regarding the ice temperature, the presence of cold ice was observed in the subsurface at both stake locations. The temperature measurements performed in April were affected by drilling disturbances, as the ice temperature measured later in May show lower temperatures at the same levels. The profile recorded at the time of setup also show higher temperature gradient, with a near zero temperature at the bottom of the borehole related to the presence of melt water, and

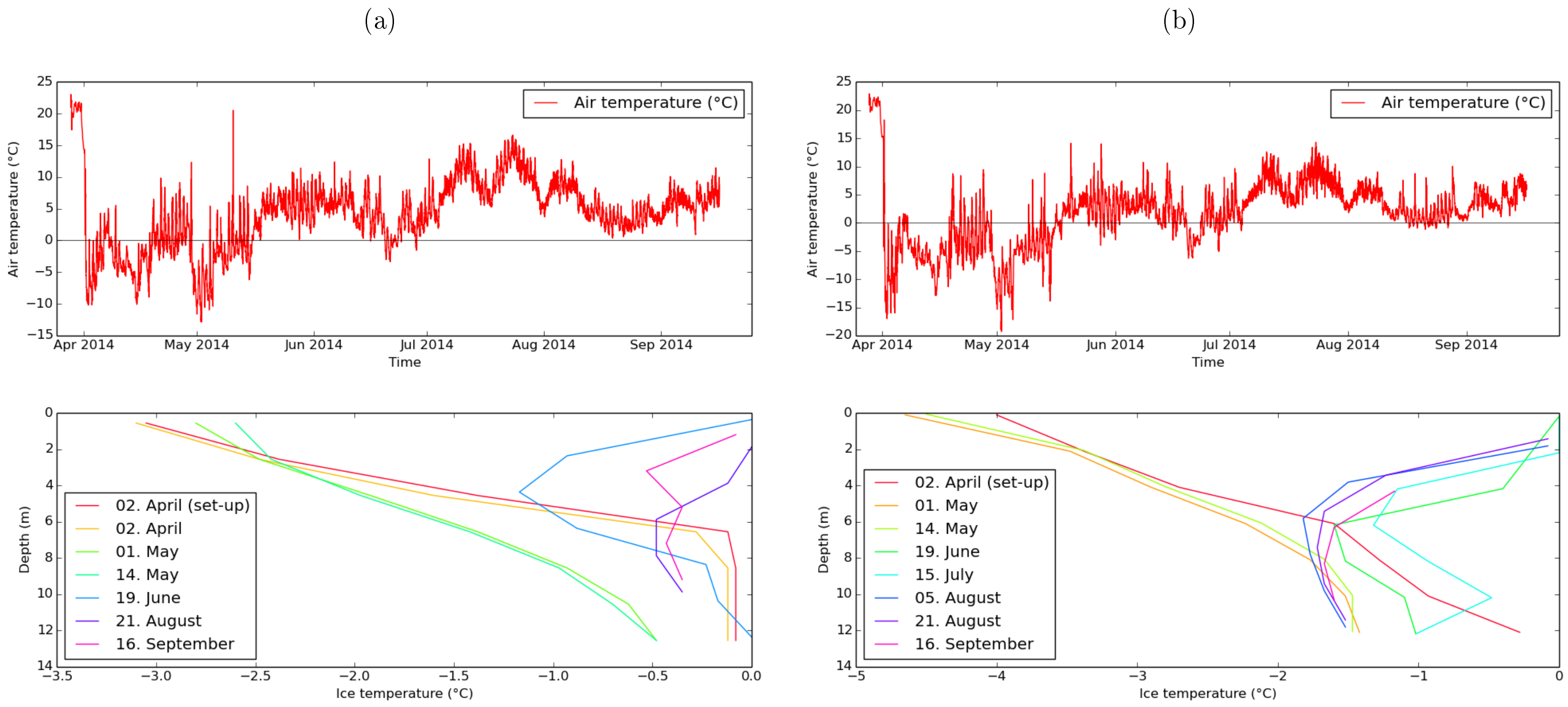


Figure § 3.5: Air temperature and ice temperature profiles at stake H13 (a) and H44 (b), Hellstugubreen 2014. Note that the scale of the ice temperature axis is different for both figures.

Table § 3.1: Results of temperature measurements at stake H13 (accuracy: ±0.05°C). Missing data or temperature values recorded above the surface are indicated with the × symbol.

2014-04-02 at 10:30		2014-04-02 at 16:15		2014-05-01		2014-05-14 at 13:00		2014-06-19 at 18:00		2014-08-21 at 13:00		2014-09-16 at 13:55	
Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)
0.56	-3.05	0.56	-3.10	0.56	-2.80	0.56	-2.60	0.37	0.00	-2.12	×	-2.80	×
2.56	-2.38	2.56	-2.48	2.56	-2.48	2.56	-2.42	2.37	-0.93	-0.12	×	-0.80	×
4.56	-1.38	4.56	-1.60	4.56	-1.92	4.56	-1.97	4.37	-1.17	1.88	0.00	1.20	-0.08
6.56	-0.12	6.56	-0.28	6.56	-1.38	6.56	-1.42	6.37	-0.88	3.88	-0.12	3.20	-0.53
8.56	-0.08	8.56	-0.12	8.56	-0.93	8.56	-0.97	8.37	-0.23	5.88	-0.48	5.20	-0.35
10.56	-0.08	10.56	-0.12	10.56	-0.62	10.56	-0.70	10.37	-0.17	7.88	-0.48	7.20	-0.43
12.56	-0.08	12.56	-0.12	12.56	-0.48	12.56	-0.48	12.37	0.00	9.88	-0.35	9.20	-0.35

Table § 3.2: Results of temperature measurements at stake H44 (accuracy: ±0.05°C). Missing data or temperature values recorded above the surface are indicated with the × symbol.

2014-04-02 at 15:00		2014-05-01		2014-05-14 at 09:45		2014-06-19 at 15:40		2014-07-15 at 17:30		2014-08-05 at 17:30		2014-08-21 at 10:00		2014-09-16 at 18:55	
Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)	Depth (m)	Temp. (°C)
0.20	-4.00	0.20	-4.65	0.20	-4.50	0.20	×	0.20	×	-0.18	×	-0.57	×	-1.68	×
2.20	-3.38	2.20	-3.48	2.20	-3.38	2.20	×	2.20	0.00	1.82	-0.08	1.43	-0.08	0.32	×
4.20	-2.70	4.20	-2.90	4.20	-2.80	4.20	-0.40	4.20	-1.15	3.82	-1.50	3.43	-1.23	2.32	×
6.20	-1.60	6.20	-2.23	6.20	-2.12	6.20	-1.60	6.20	-1.32	5.82	-1.82	5.43	-1.67	4.32	-1.17
8.20	-1.28	8.20	-1.77	8.20	-1.67	8.20	-1.52	8.20	-0.93	7.82	-1.77	7.43	-1.72	6.32	-1.60
10.20	-0.93	10.20	-1.52	10.20	-1.47	10.20	-1.10	10.20	-0.48	9.82	-1.67	9.43	-1.67	8.32	-1.67
12.20	-0.28	12.20	-1.42	12.20	-1.47	12.20	-1.02	12.20	-1.02	11.82	-1.52	11.43	-1.52	10.32	-1.60

with a colder temperature approximating the temperature at the bottom of the snowpack recorded by the uppermost sensors (-3.05°C at H13 and -4.00°C at H44). The temperature profile measured ∼6 hours after setup at stake H13 already starts stabilizing towards a lower temperature gradient through heat conduction.

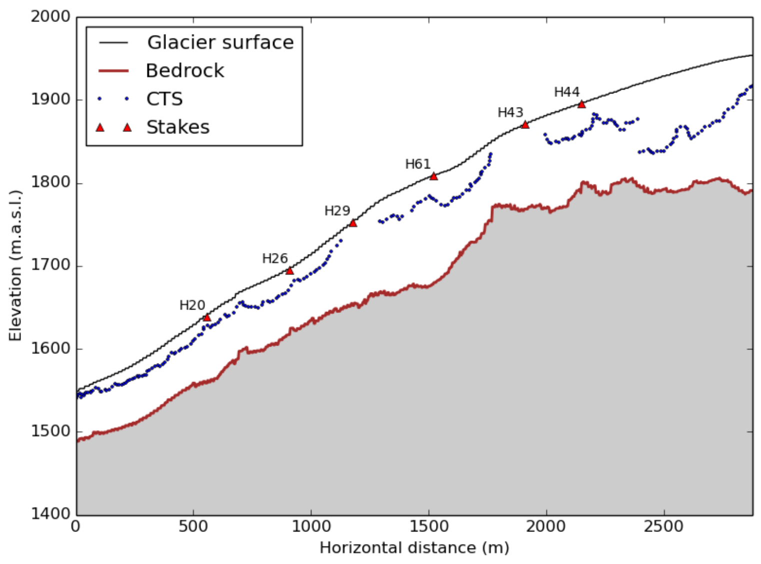
A temperature gradient inversion can observed at both stake locations from the 19th of June. The inversion is not seen on the 14th of May, which suggests that the snowpack covering the ice surface thinned considerably or disappeared completely between the 14th of May and the 19th of June. Field observations locate the snow line above the stake H13 at the end of June, and the snow line migrated above the stake H44 between the 17th of July and the 5th of August (Oda J. Røyset (pers. communication)). The insulating effect of a thick snowpack simultaneously stopped, and the ice temperature in the near surface were affected by warmer air temperatures. The inversion seems to occur around the sensor at a depth of ∼4.37 m at stake H13 (*Table § 3.1*), while occurring around the sensor located at ∼6.20 m at stake H44 (*Table § 3.2*). After the 19th of June the temperature gradient below the temperature inversion become less steep with time, while the temperature gradient above is affected by diurnal variations of the surface air temperature. At stake H13 at the end of the summer, the ice is almost temperate but remained cold from a depth between 1.20 and 3.20 m, accounting for the sensors accuracy (*Table § 3.1*). At stake H44, the cold winter wave is clearly not eliminated in the subsurface, with ice temperatures lower than -1.17 ± 0.05 °C from a depth superior to 4.32 m (*Table § 3.2*).

1.2 Internal layering and basal thermal regime at Hellstugubreen

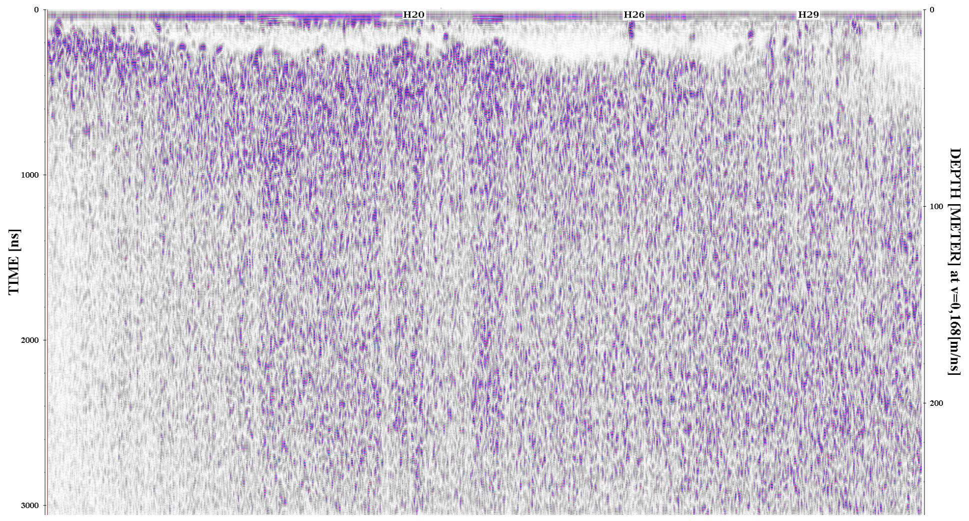
The thermal layering was mapped at Hellstugubreen using RES measurements at a center frequency of 50 MHz (*Mapping the Cold-temperate transition surface with GPR*). The digitization of the CTS along two profiles are presented in this section. The profile H166 (*Figure § 3.6*) was chosen as it followed approximately the center flow line of the glacier (*Figure § 2.7a*), where the ice is expected to be thickest. The second profile presented is H168 (*Figure § 3.7*), as it includes several transverse sections of the glacier, and therefore shows the thermal layering both close to the margins and close to the center line (*Figure § 2.7a*). The results for the third profile (H167) are shown in *Appendix E.1*. On these results, the glacier surface along the profile is derived from the 2009 laser scanning data. Likewise, the depth of the ice/bedrock interface is estimated from the difference between the glacier surface and the ice thickness map (*Figure § 3.1*).

Figure § 3.6a shows the digitization of the CTS from the radar measurements along H166. Hellstugubreen has a surface cold layer almost along the whole length of the profile H166. The cold surface layer seems to disappear at two different locations, though, in the neighbourhood of stakes H29 and H43. On the radargrams from *Figure § 3.6b* and *Figure § 3.6c*, much scattering of the radio-waves can be observed near the surface, at these two same locations. Large crevasses

(a) CTS mapping on profile H166



(b) H166 profile (first half)



(c) H166 profile (second half)

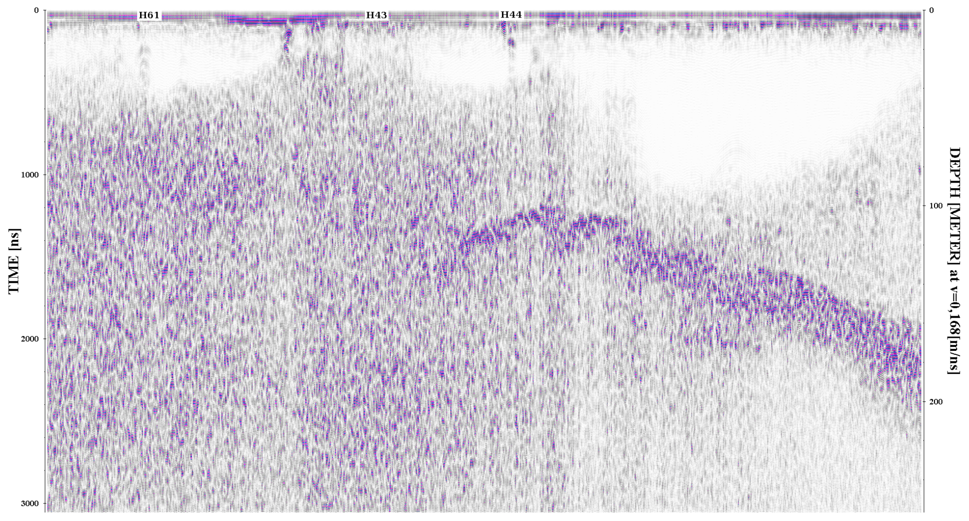
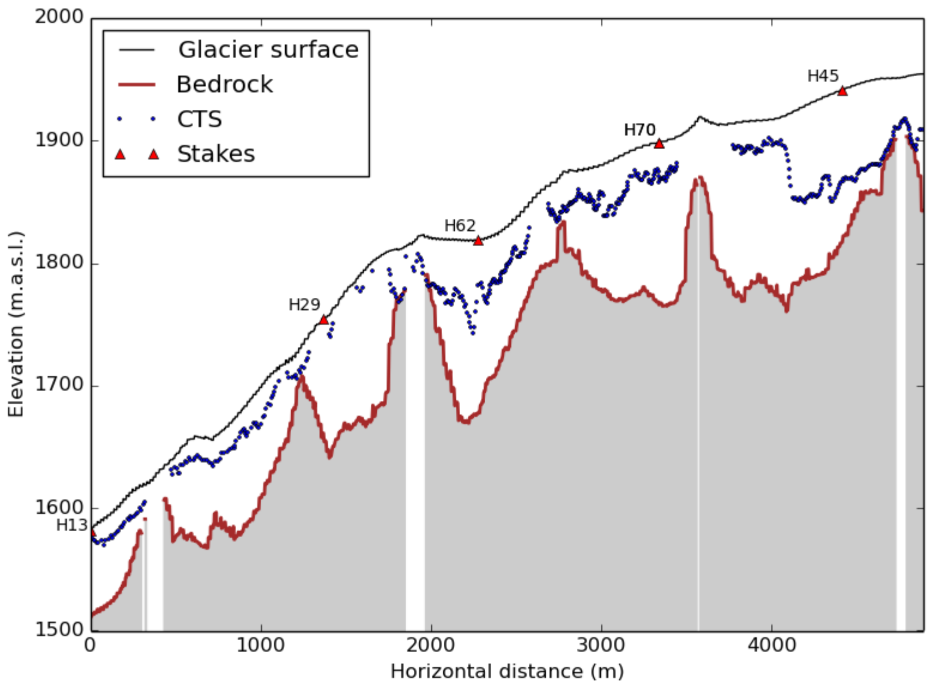
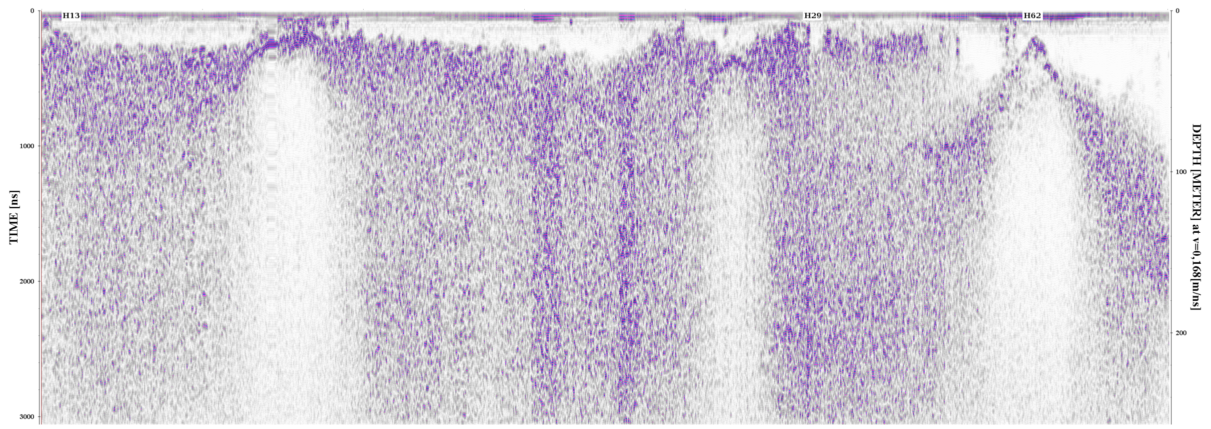


Figure § 3.6: RES on Hellstugubreen along the profile H166. On (a), the CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements (10 MHz) from 2011 (data : NVE). (b) and (c) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014). The distance along the profile shown in (b) and (c) increases from left to right.

(a) CTS mapping on profile H168



(b) H168 profile (first half)



(c) H168 profile (second half)

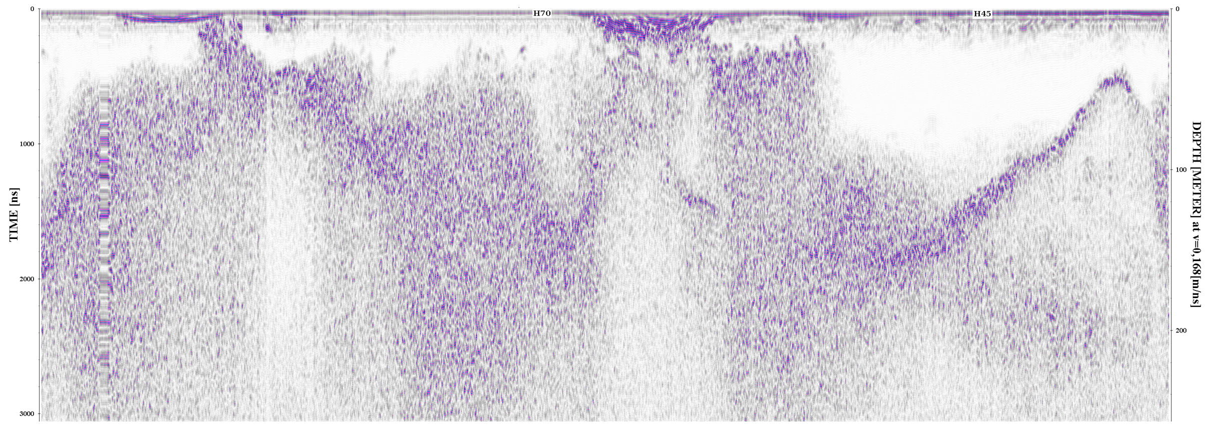


Figure § 3.7: RES on Hellstugubreen along the profile H168. On (a), the CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements (10 MHz) from 2011 (data : NVE). (b) and (c) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014). The distance along the profile shown in (b) and (c) increases from left to right.

can be seen during summer around stake H29, which could be the reason for the radio-waves scattering (Plewes and Hubbard, 2001). Overall, the thickness of the cold surface layer increases up glacier, with a sudden deepening of the CTS at ~1940 m.a.s.l., from a depth of ~40 m to ~90 m. The CTS is almost not visible in the lower parts of the profile H166, where the glacier is at the pressure-melting point almost throughout the whole ice thickness. Hellstugubreen has a temperate basal thermal regime along the entire profile H166.

Figure § 3.7a presents the results of the CTS digitizations of the RES measurements along the profile H168. The depth of the CTS shows greater variations as regards to the profile H166. The ice/bedrock interface is not always shown in the parts where the profile was close to the glacier margins. This results from the buffer-like area that appeared on the ice thickness map after interpolating the thickness values (see *Ice thickness at Hellstugubreen*). The depth of the bedrock was not estimated at these locations. Overall, the variations of the CTS depth follows the variations of the depth of the bedrock. The glacier seems to be cold-based close to the margins, and to have a temperate basal thermal regime where the ice is thicker. The temperate basal layer was encountered at about 80 m and 90 m at the stake H62 and H45 respectively. A sudden deepening of the CTS was also observed on the profile H168, at around 1965 m.a.s.l. The CTS seemed to disappear over ~300 m along the profile, between the stakes H70 and H45. The radar signal was completely reflected at the surface at this location (*Figure § 3.7c*).

2 Storbreen

2.1 Subsurface temperature variations

On Storbreen, the ice temperature variations were measured at stake S2 from the 21st of May to the 18th of September 2014 (*Figure § 2.7b*). The temperature was recorded at ten depth levels, every two hours, which give twelve temperature profile per day (see *Subsurface ice temperature at Storbreen*). The ice temperature profiles were plotted in *Figure § 3.11*, with above the air temperature recorded by the GeoPrecision data logger. The temperature profiles were interpolated in order to visualize ice temperature variations with time, and the effect of the air temperature on the subsurface ice temperature.

The ice melt occurring at the surface was modelled using a *Positive Degree-Day* (PDD) model, which *Degree-Day Factor* was computed from the *Control Periods* (CPs) (see *Modelling the depth changes of the sensors*). The first CP lasted from the 5th of August to 23rd August (18 days). The DDF computed for this period was 1.6 mm °C^{-1} d^{-1} or 1.47 mm °C^{-1} d^{-1} *water equivalent* (w.e.) with an ice density of 917 kg.m^{-3}. The second and last CP started on the 23rd of August and ended on the 18th of September 2014 (26 days). The DDF estimated for this period was 4.1 mm °C^{-1}.d^{-1}

or 3.76 mm w.e. °C^{-1} d^{-1}. The average DDF calculated from the CPs, and weighted by the number of days of each CP, was equal to 2.84 mm w.e. °C^{-1} d^{-1}. For the measurement periods other than CPs, a DDF equal to 2.84 mm w.e. °C^{-1} d^{-1} was therefore used to model the surface melt and update the depth of the sensors in the ice. Only one measurement period happened to be between field observations during which the ice surface was snow covered. This period started on the 21st of May, when the snowpack was 2.58 m thick, and ended on the 5th of August. The snowpack had completely melted away by the end of the period. Therefore, the surface melt was computed starting from the end of the period, in order to update the sensors depth, and using the mean DDF. Once the total amount of melt for the period was reached, the ice surface was assumed to be snow covered. For the first simulation, the disappearance of the snow cover was modelled on the 25th of May, four days after the start of the period. The melt of a snowpack 2.58 m thick in four days is very unlikely and could not be explained by the temperature only. The elimination of the snowpack through melt processes was modelled too early, owing to a mean DDF certainly too minor.

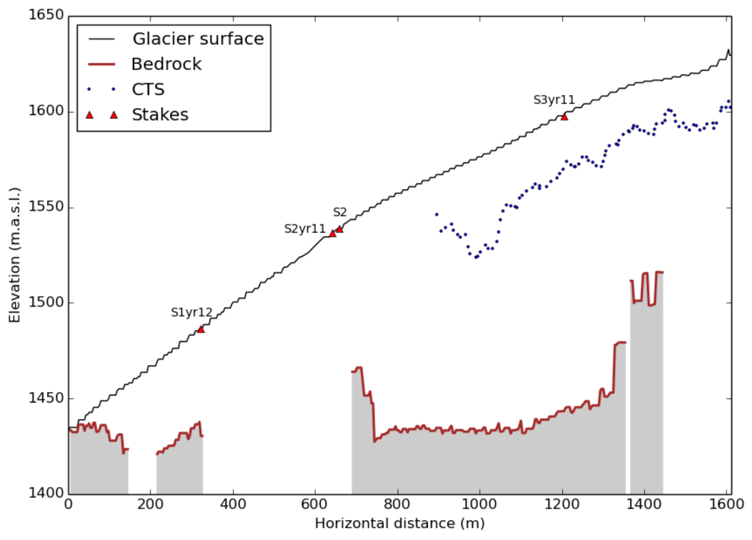
Regarding the ice temperature, the temperature gradient show an inversion from the beginning of the whole measurements period, at a depth of about 2 m. The measured/modelled ice surface temperature (see *Estimating the ice surface temperature*) is equal to -2.0°C on the 21st at midnight, and becomes temperate starting from the 6th of July, around 4 pm. This means that it took about 46 days for the ice to be temperate near the surface. This is inconsistent with mostly positive air temperatures and the absence of a snowpack over the major part of this period. The cold winter wave is completely eliminated in the subsurface on the 1st of August, with ice at the pressure-melting point along the entire profile.

2.2 Internal layering and basal thermal regime at Storbreen

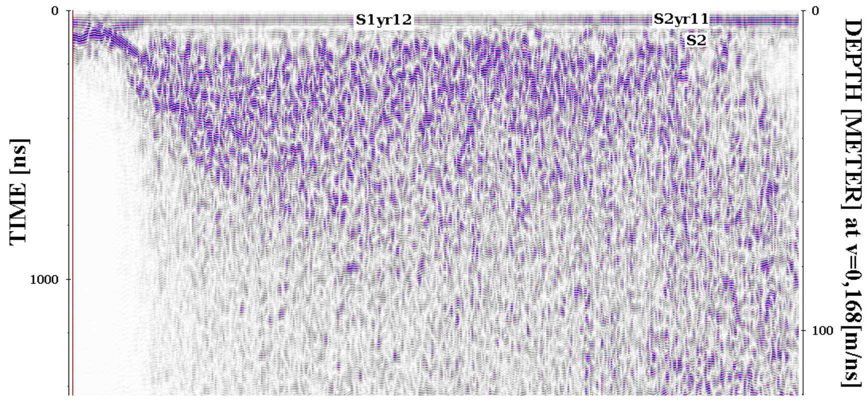
The internal thermal layering was also mapped on the lower parts of Storbreen, using a GPR antenna with a center frequency of 50 MHz (*Mapping the Cold-temperate transition surface with GPR*). In the same way as done for Hellstugubreen, two profiles along which the CTS was digitized are presented for Storbreen. The profile S179 (*Figure § 3.8*) was chosen for its medial location, as regards to the glacier margins (*Figure § 2.7b*). The second profile (S178, see *Figure § 3.9*) is also presented as it includes several traverse sections of the glacier (*Figure § 2.7b*), and therefore shows the relationship between the distance to the glacier margins and the CTS depths variations. The results of the CTS digitization for the profile S180 and S181 (*Figure § 3.8*) are shown in *Appendix E.2.*

On these results, the glacier surface is derived from the 2009 LiDAR data. As for the depth of the ice/bedrock interface, no ice thickness map was produced for Storbreen, as opposed to Hellstugubreen, and therefore a different method was used to estimated the depth of the bedrock

(a) CTS mapping on profile S179



(b) S179 profile (first half)



(c) S179 profile (second half)

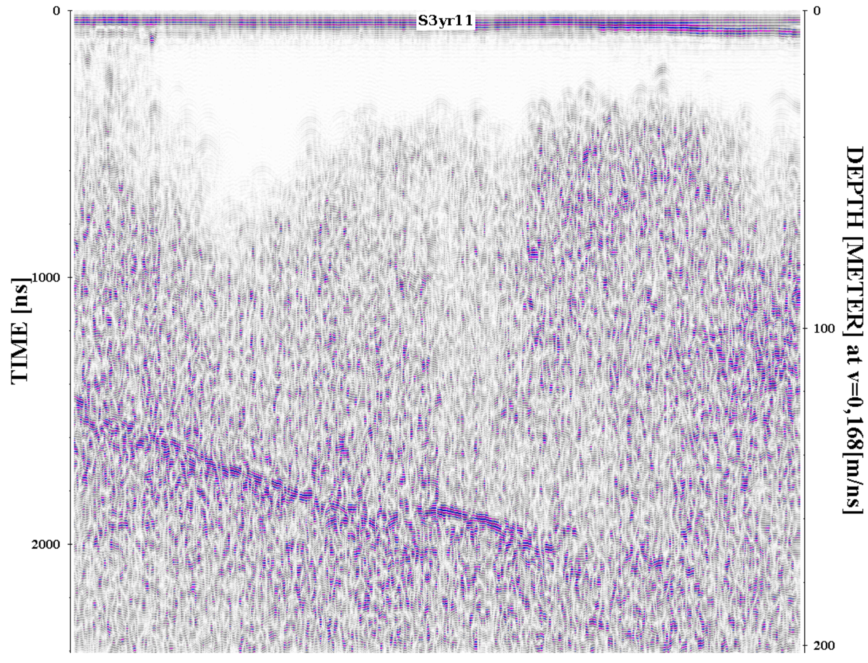
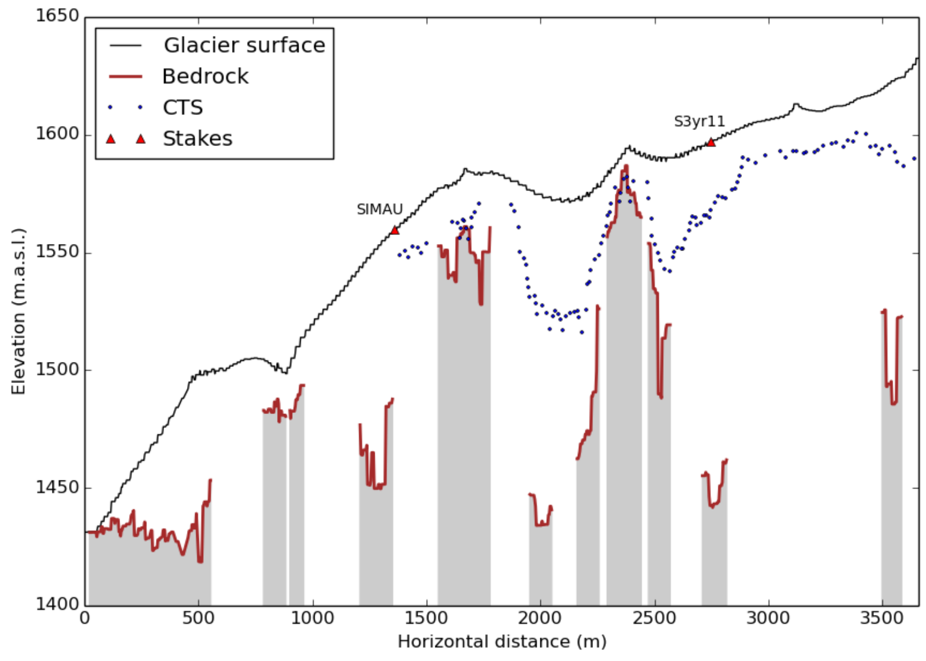
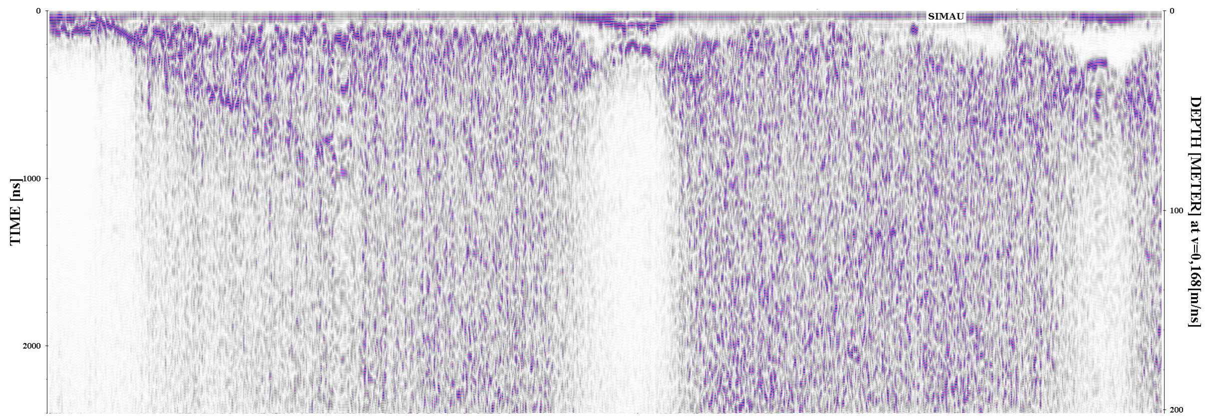


Figure § 3.8: RES on Storbreen along the profile S179. On (a), the CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements from 2005-2006 (10 MHz, data : NVE) and from 2014 (50 MHz). (b) and (c) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014). The distance along the profile shown in (b) and (c) increases from left to right.

(a) CTS mapping on profile S178



(b) S178 profile (first half)



(c) S178 profile (second half)

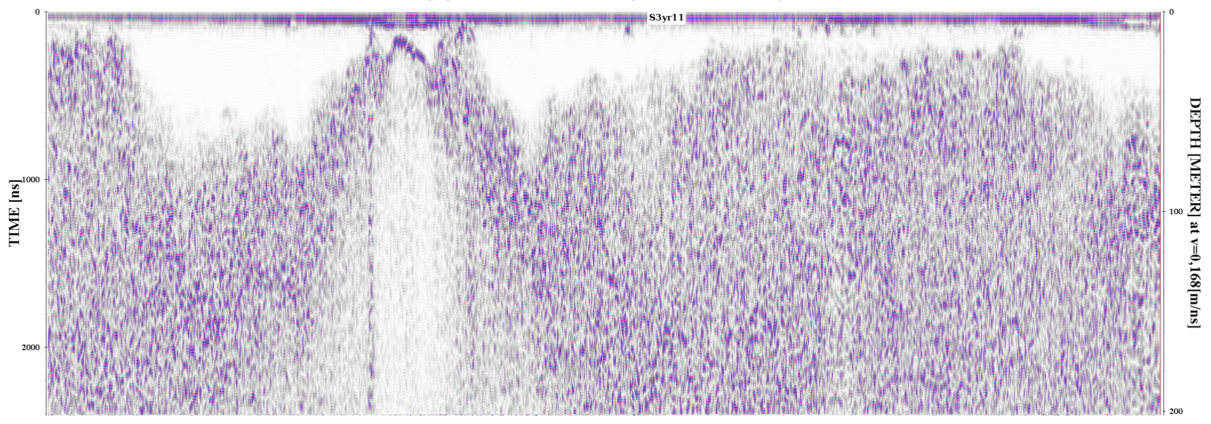


Figure § 3.9: RES on Storbreen along the profile S178. On (a), the CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements from 2005-2006 (10 MHz, data : NVE) and from 2014 (50 MHz). (b) and (c) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014). The distance along the profile shown in (b) and (c) increases from left to right.

along the profiles. The available thickness data (*Figure § 3.4*) were interpolated using a bicubic spline algorithm, with four control points for each spline. The glacier outlines, for which the depth was set to zero, were also used in the interpolation process. The interpolated output had a spatial resolution of 25 m. The ice thickness along the profiles was derived from this interpolation. However, as the amount of point measurements was scarce in the studied glacier area, only the interpolated values located at 50 m or less from a GPR record were considered. This 50 m proximity threshold was used to limit the occurrence of interpolation errors in the results. The GPR records from 2005-2006 were not corrected for the surface lowering experienced by the glacier between the measurements and the field work in April 2014 (see *Ice thickness on Storbreen*). As the thickness of the snowpack was unknown along the GPR profiles of 2014, while mapping the CTS, the depth of the digitized ice/bedrock horizon on the same profiles was not corrected for the snow thickness. The depth to the bedrock was then estimated in the same way as with Hellstugubreen, by calculating the difference between the glacier surface elevation and the assessed ice thickness values, where available. Large portions of the profiles were situated farther than 50 m away from the nearest GPR records. As such, the ice/bedrock interface is not represented in the results at several parts of the profiles (*Figure § 3.8*, *Figure § 3.9*).

Figure § 3.8a shows the digitization of the CTS along the profile S179. Storbreen has a temperate basal layer along the entire profile length. At an elevation lower than ∼1550 m.a.s.l., this temperate layer is almost as thick as the full ice thickness of the glacier. Downstream this point, the glacier seems to have a thin surface layer below the pressure-melting point. However, the CTS was not digitized on this part of the profile, as it could not be done accurately owing to subsurface structures and frequent signal scattering patterns (*Figure § 3.8b*). The subsurface ice temperature measurements performed at stake S2 confirm the presence of a thin cold surface layer (*Figure § 3.11*). The thickness of the cold layer increases abruptly up glacier between stake S2 and S3yr11 (∼1000 m from the glacier front), to a value nearing 50 meters. The cold surface layer becomes thinner again higher up along the profile, where the depth to the CTS oscillates around a value of 30 m.

Figure § 3.9a presents the results of the CTS digitization along the profile S178. Similarly to traverse sections at Hellstugubreen, the CTS level seems to sink with increasing ice thickness. However, this pattern is obvious only upstream the stake SIMAU (1555 m.a.s.l. in 2013). At lower elevation, the CTS was not digitized due to the same difficulties as encountered with the profile S179. The glacier is cold-based in the proximity of the margins. The cold surface layer reaches a thickness of ∼55 m at about 1570 m.a.s.l. (horizontal distance of ∼2180 m on *Figure § 3.9a*), and the CTS was detected at about 50 m depth at ∼1590 m.a.s.l., in the vicinity of stake S3yr11.

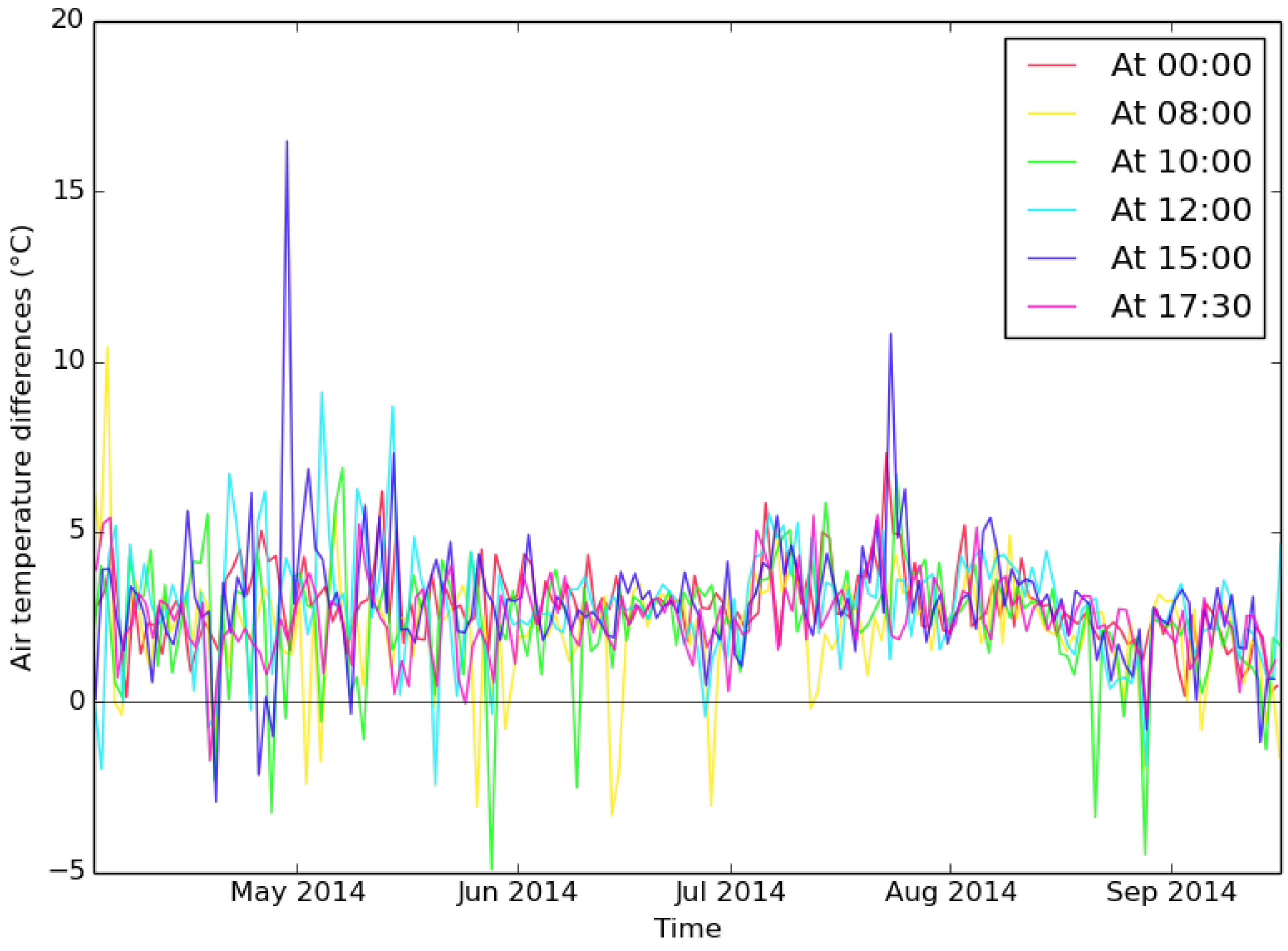


Figure § 3.10: Air temperature differences observed between the stake locations H13 and H44 on Hellstugubreen. The values plotted are the temperature records at H13 minus the temperature measured at H44, at different times of the day.

3 Discussion

At Hellstugubreen, the temperature profile measurements at stakes H13 and H44 both witness the existence of cold ice in the subsurface. H13 is clearly in the ablation area in the lower part of the glacier, whereas H44 is in the upper part of the glacier. The stake H13 was located under the ELA for the last 50 years (Andreassen et al., 2011*a*). The stake H44 was mostly under the ELA for the last 10 years, except in 2008 and 2012 where remaining snow was observed at the stake location, by the end of the summer (Andreassen et al., 2011*a*, Liss M. Andreassen (pers. communication), *Appendix A.1*). As such, the ice is rarely snow covered early in winter at these two stake locations, and is therefore not or poorly insulated from cold temperatures in this season. At the end of the summer, the cold winter wave is almost eliminated at stake H13, while the subsurface ice temperature at stake H44 remains well below the pressure-melting point. This difference is explained by the higher elevation of stake H44 resulting in lower yearly air temperatures. Air temperature measurements pointed out a mean difference of 2.6°C between both locations. However, the air temperature is not always warmer at stake H13. *Figure § 3.10* shows the air temperature differences at both stakes, and at different times of the day. The air temperature gradient is not constant in time over the glacier. During the morning, the upper parts of the glacier are exposed to the sun, while the steeper lower parts are still hidden from the sun. This resulted occasionally in warmer temperatures at stake H44 than at stake H13, owing to different intensities of the radiative heating of the air by direct solar radiations. In summer, as the solar elevation angle is larger, the shading effects from the surface topography is diminished in this high latitude area. Therefore, the air temperature was almost always colder at H44. The surface orientation and surface slope have an influence on the subsurface ice temperature, by affecting the time of the onset and of the end of the diurnal signal penetration in the ice. However, on a glacier, the elevation and the presence of a snow cover or not are more significant contributors to the ice thermal regime.

Regarding the temperature measurements at stake S2 on Storbreen, the results show that a cold ice surface layer remained from the last cold winter wave, down to a depth of ~6 m, until the month of July. After this month, the ice is temperate along the entire profile. The DDF calculated from the CPs is too small, as it model the melt of a 2.58 m thick snowpack in four days. For the calculation of the DDF, the model assumed that no precipitation events occur during the CPs. A snowfall event during a CP may affect significantly the estimation of the DDF, as new snow would increase the surface albedo and decrease the melt rate. If the glacier surface receives precipitation as rain, the relatively warm water would bring energy to the ice surface, available for melt. However, the heating from rain is often a minor contributor the energy balance of glaciers (Benn and Evans, 2010). The DDF is an empirical factor used in degree-day models, and has the purpose to represents parameters that affect the melting rate, other than temperature. These parameters (e.g. wind, radiations, precipitation...) are variable in time and space and are therefore difficult to represent with a constant coefficient. Furthermore, the DDF should be calculated over longer CPs. On Storbreen, Engelhardt (2014) estimated from all summer ablation measurements available a DDF equal to 5.3 mm w.e. $°C^{-1}$ d^{-1}. *Figure § 3.11* show the results using the same DDF.

Using the new DDF in the model, the 2.58 snowpack melted away after 45 days and the ice surface was snow-free starting from the 7th of July. This result is much more sensible than when using the computed DDF from the CPs. The snowpack had experienced considerable thinning in a month time, as the effects of the air temperature diurnal variations are visible in the ice, down to a depth of ~1.4 m on the 20th, 21st and 22nd of June (see *Appendix D.1*). With a

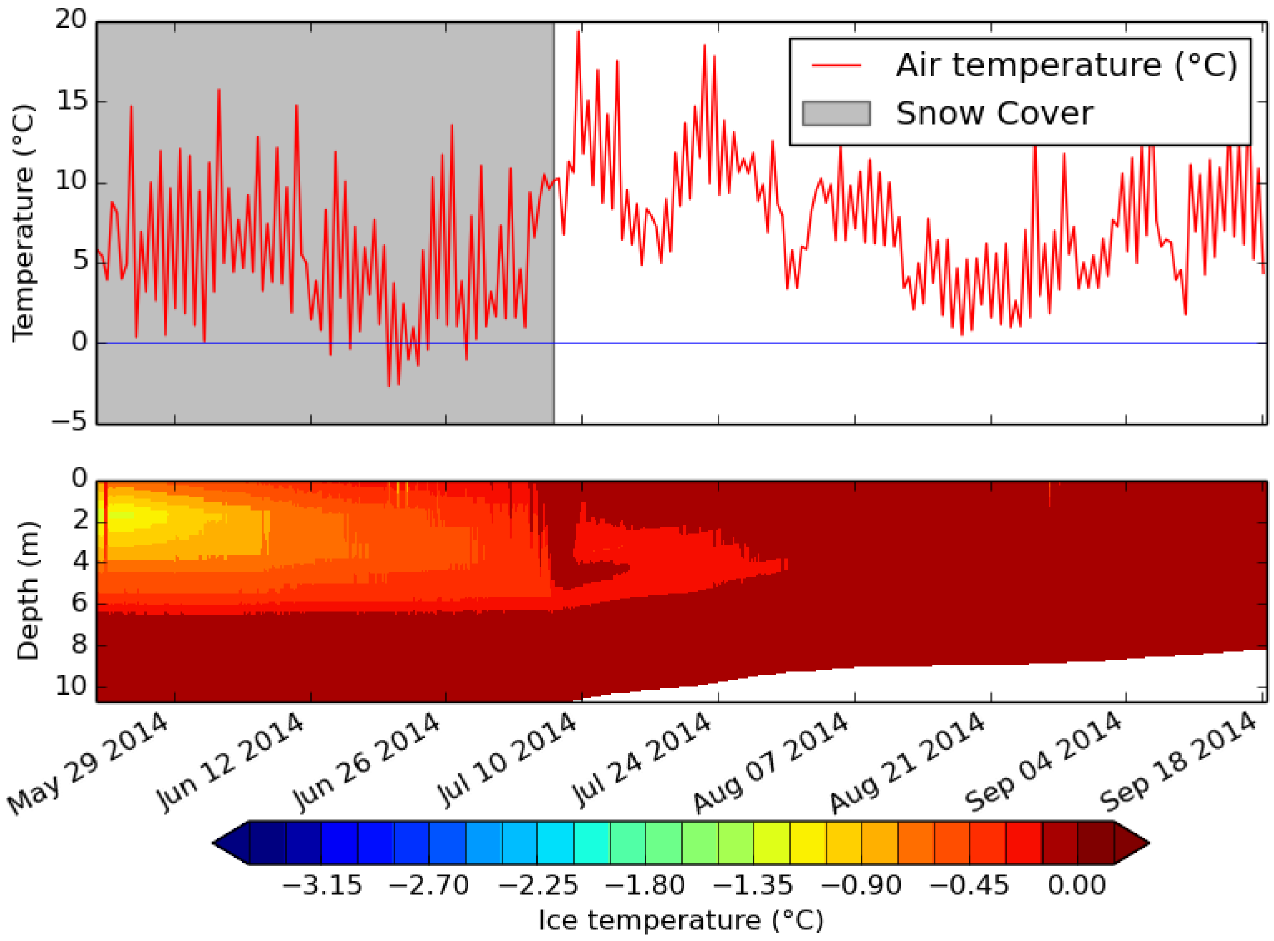


Figure § 3.11: Air and ice temperature at stake S2, Storbreen 2014. The depth of the sensors in ice are updated using the corrected DDF (5.3 mm w.e. °C^{-1} d^{-1}) from Engelhardt (2014).

DDF of 5.3 mm w.e. °C^{-1} d^{-1}, the disappearance of the snowpack is well synchronized with the transition towards a temperate thermal regime in the subsurface. The use of a DDF equal to 5.3 mm w.e. °C^{-1} d^{-1} gives more consistent results. This value should be adopted for further studies on Storbreen.

The RES measurements enabled to get an insight into the thermal regimes of Hellstugubreen and Storbreen at greater depths of investigations. The RES surveys were only conducted in the ablation area of the glaciers. The temperature distribution of both glaciers is typical of the one observed in the ablation area of polythermal glaciers (Björnsson et al., 1996; Hagen et al., 2003; Pettersson et al., 2003). On the radargrams from April 2014, the glaciers seems cold based at the front. This is confirmed from the borehole temperature measurements. The CTS was however not digitized in this locations, as surface structures and heavy signal scattering made difficult the digitizing process. At the end of the summer, the cold winter wave is not eliminated at

Hellstugubreen, with only the only the uppermost layer that is affected by diurnal temperature variations. At Storbreen, however, the cold wave seemed to be completely eliminated at stake S2 . Further up on the glacier, the cold surface layer reaches a thickness of ∼50 m at certain locations. The summer temperatures are there not sufficient to restore a temperate thermal regime.

Traverse sections show that both glaciers are cold-based near their margins, and that the CTS level deepens further away from the margins, as the ice thickness increases. Overall, the cold surface layer was thickest where the ice was thickest on both glacier. This results from the fact that the glaciers are thicker high up in the part of the ablation area mapped with GPR. At higher elevations where the mean temperatures are lower, the cold winter wave is more intense and penetrate deeper in the ice. This led to an increase of the CTS depth with increasing elevation observed on the results. The cold surface was at maximum 90 m at Hellstugubreen and 55 m thick at Storbreen along the the GPR profiles.

Except at the front, both glaciers have a temperate basal thermal regime beneath their central part. In the lowermost parts of the ablation area, the early winter temperatures, before the settlement of a thick insulating snowpack, do not allow the cold wave to penetrate down to the bedrock. At higher elevation where the winter temperatures penetrate deeper in the ice, the ice is too thick to allow the transition towards a cold-based regime. Moreover, at greater depths the pressure-melting point is depressed owing to the overburden pressure of the overlaying ice (Cuffey and Paterson, 2010).

As opposed to Svalbard and other polar latitude locations (Ahlmann, 1935; Schytt, 1964; Liestøl, 1988; Björnsson et al., 1996; Jania et al., 1996; Hagen et al., 2003), most glaciers in mainland Norway are considered to be temperate (Andreassen et al., 2012). However, in Southern Norway and above the lower limit of alpine permafrost, where the local climate is characterized by low winter temperatures and precipitations (Etzelmüller and Hagen, 2005), the presence of cold ice was observed in several glaciers. Borehole temperature measurements combined with GPR surveys conducted at Nedre Steindalsbre indicated ice temperatures below the pressure-melting point close the glacier front (Urdahl, 2005). Gråsubreen also located in Jotunheimen region has a thermal regime similar to the one observed at Hellstugubreen and Storbreen (Sørdal, 2013).

The *Internal Reflecting Horizons* (IRHs) observed on the GPR profiles may not show the actual CTS. A study on a Hansbreen polythermal glacier in southern Spitsbergen compared temperature profiles obtained from borehole measurements with IRHs (*Internal Reflecting Horizons*) obtained both from airborn *Ultra High Frequency* (UHF) radio-echo soundings and low frequency radio-echo soundings (Jania et al., 1996). Internal reflections observed from the radio-echo soundings occurred all at greater depth than the interface cold/temperate ice obtained from borehole measurements. Jania et al. (1996) explain these differences by a specific layering : A finite temperate ice layer with a low water content underlying the isotherm limit, which is underlain by temperate ice with a high water content. The temperate with lower water content is transparent to the radio-echo soundings. However, much scattering of the radar signal occurs when the water content increases. This increases the uncertainties of the CTS depth estimation from RES surveys. To map accurately the CTS positions, GPR measurements should be combined with borehole temperature measurements (Pettersson et al., 2004).

Chapter C

Ice flow velocity

1 Ice surface velocity at Hellstugubreen

At Hellstugubreen, the ice surface velocity data were interpolated, using the information from cross-correlations between the velocity data and the ice thickness on the one hand, and between the velocity and the surface slope on the other hand (see *Ice surface velocity interpolation*). The surface velocity map resulting from the cokriging algorithm is shown in *Figure § 3.12*. The same buffer-like area along the glacier outlines is present on this map, as the input thickness data used by the algorithm have a value of zero at this location. The estimated surface velocity values range from ~0.5 m.yr^{-1}, in the upper parts of the glacier close to stake H45, up to ~15.8 m.yr^{-1} near stake H29. The velocity values measured directly at stake locations are presented in *Table § 3.3*, together with the ice thickness and surface slope parameters estimated at each location. It must be noted that the surface velocity values are not all averaged from the same measurement period. The stake network density was improved in 2013-2014 on Hellstugubreen. Prior to 2013, less measurements were available, and the resulting values were not interpolated, as large distances between stake leads to larger uncertainties in the interpolated velocity values. As the surface velocity of a glacier is not constant in time, the stake surveys conducted before 2013 were not used to produce the surface velocity map.

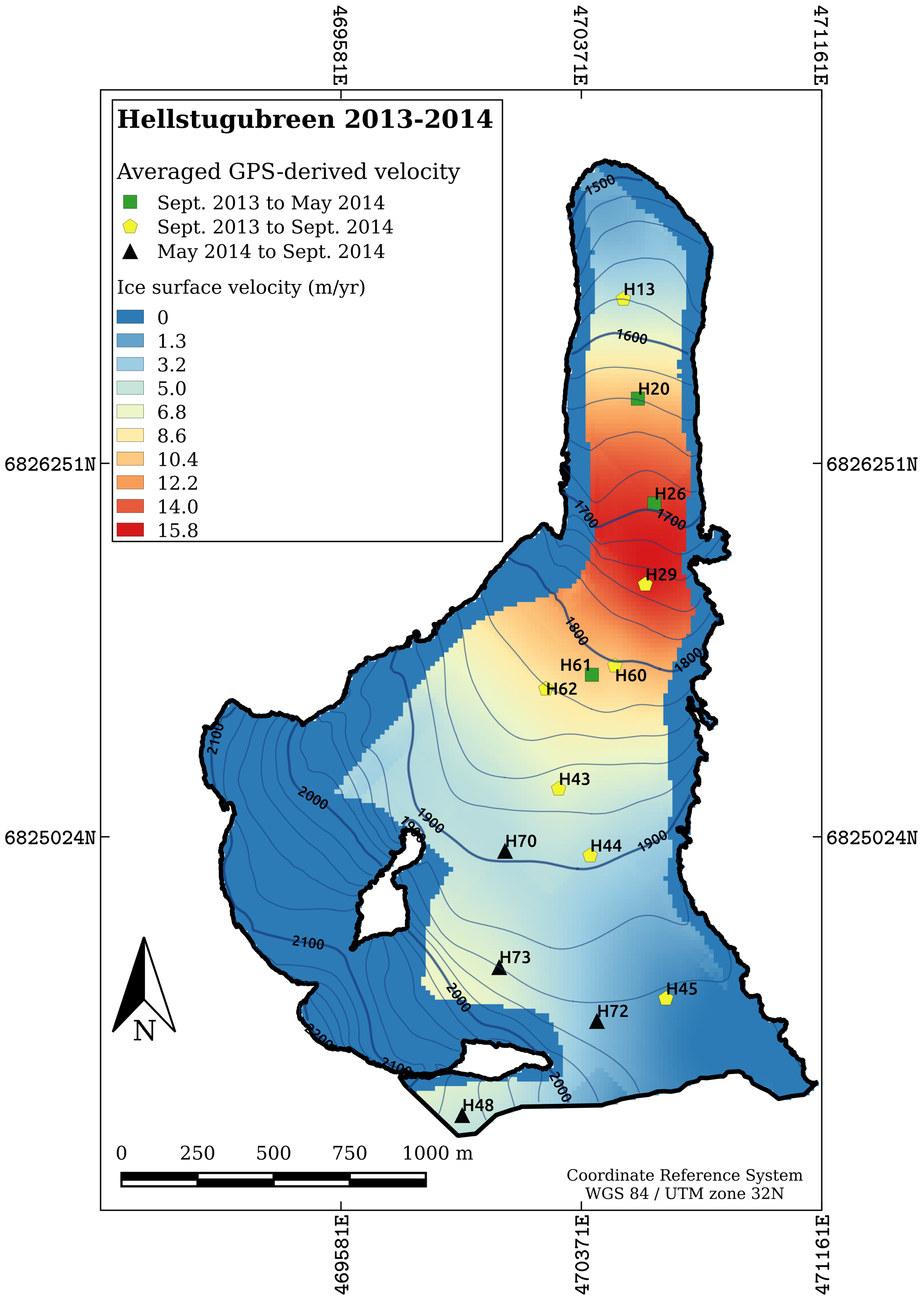


Figure § 3.12: Ice surface velocity map at Hellstugubreen for 2013-2014. The velocity values are derived from DGNSS measurements performed at different times, and are averaged between consecutive measurements. The elevation contours and glacier outlines are derived from the 2009 laser scanning and orthophotos (data : NVE).

Table § 3.3: Horizontal surface velocity values derived from repeated stake surveys between 2013 and 2014 at Hellstugubreen. The symbol × means that the stake position was recorded with DGNSS, and the symbol — indicates the absence of measurement at this date. The thickness (±25 m) at the stake locations are extracted from the ice thickness map, and the surface slope is derived from the 2009 LiDAR data (data : NVE).

Stake	Elevation in 2014 (m.a.s.l.)	DGNSS georeferencing			Ice thickness (m)	Surface slope (degrees)	Averaged surface velocity (m.yr^{-1})
		2013-09-10	2014-05-15	2014-09-16			
H13	1570	×	×	×	69	10.1	4.5
H20	1638	×	×	—	80	11.6	11.0
H26	1693	×	×	—	73	10.3	14.8
H29	1743	×	×	×	94	11.5	15.8
H43	1864	×	—	×	103	7.9	6.5
H44	1890	×	×	×	108	5.8	4.1
H45	1937	×	×	×	133	4.1	0.5
H48	2068	—	×	×	66	8.0	5.2
H60	1795	×	×	×	124	7.3	10.2
H61	1807	×	×	—	131	6.6	10.2
H62	1815	×	—	×	137	5.7	8.6
H70	1891	—	×	×	129	6.3	4.2
H72	1945	—	×	×	103	8.0	2.4
H73	1934	—	×	×	106	12.6	6.1

2 Ice surface velocity at Storbreen

At Storbreen, the ice surface velocity estimated at stake locations were not interpolated, as the stake network was less rich than at Hellstugubreen. However, the stake surveys were more continuous in time. *Figure § 3.13* shows the horizontal velocity variations in time at each stake location where velocity measurements are available. The stake locations are shown in *Figure § 2.7b*. A surface slope map of the glacier, derived from the 2009 laser scanning, is available in *Appendix G*, and the thickness at each stake can be estimated from *Figure § 3.4*, in section *Ice thickness at Storbreen*. The averaged surface velocities range from a value nearing 0 m.yr^{-1} at stake up to ~18.3 m.yr^{-1} at S1yr12. All surface velocity values are not averaged over the same measurement period and time of the year. As such, some may represent the yearly mean surface velocity, while others may give an estimate of the summer or winter surface velocities.

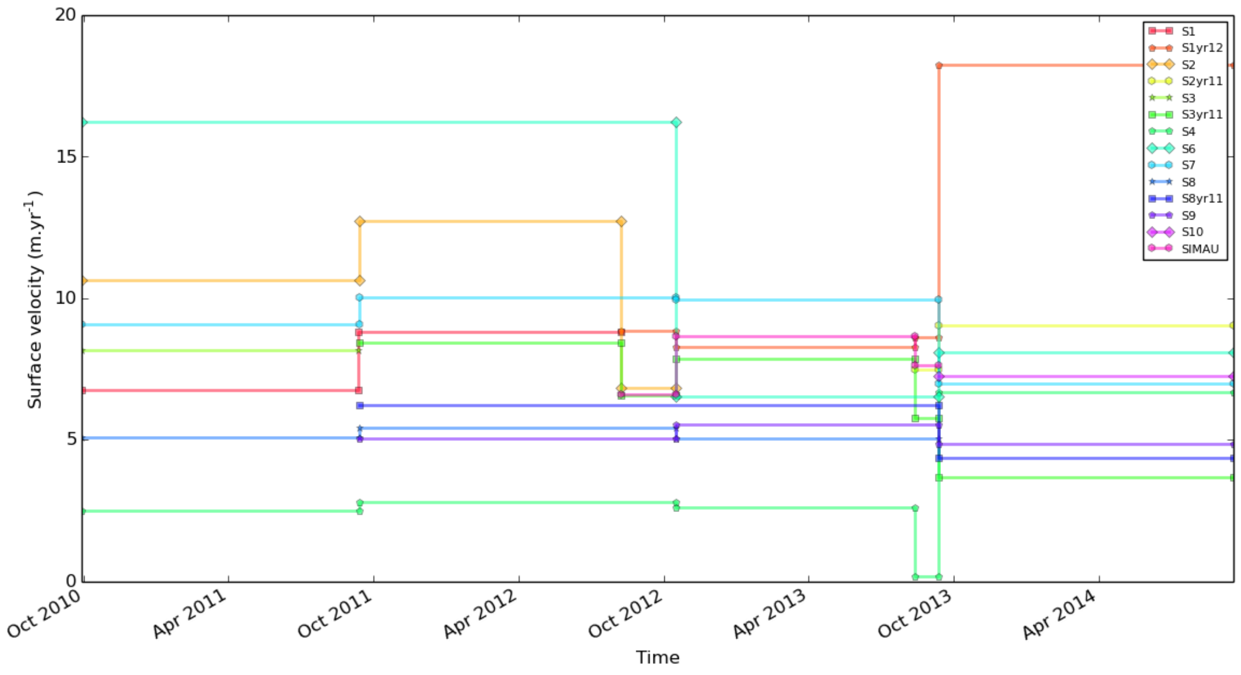


Figure § 3.13: Ice surface velocity at Storbreen at different stakes. The velocity values are derived from DGNSS measurements performed at different times, and are averaged between two consecutive measurements.

3 Subsurface deformation rate at stake S2

At stake S2 on Storbreen, the ice temperature was accurately measured in the subsurface. The high temporal resolution of the measurements allows to estimate the effects of temperature variations on the ice deformation rate. *Figure § 3.14* shows the result of the integration of the creep relation of ice (equation (§ 2.9)), accounting for changes of the temperature dependent creep flow parameter A. It was assumed that the ice has a constant density of 917 kg.m^{-3} for the calculations.

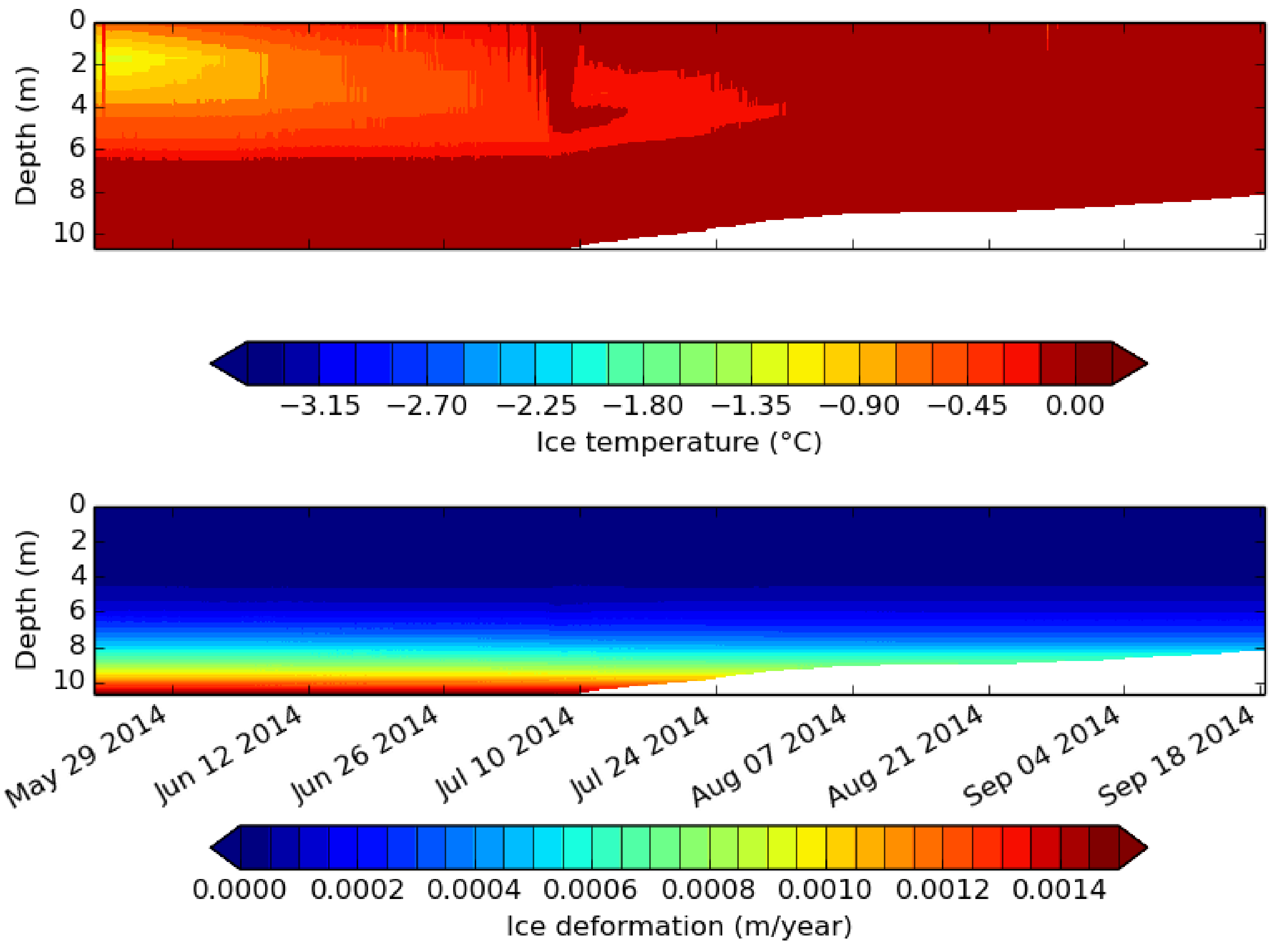


Figure § 3.14: Effects of ice temperature variations on the ice deformation rate in the subsurface at stake S2, Storbreen 2014. A DDF of 5.3 mm w.e. °C^{-1} d^{-1} was used to update the depth of the sensors as the surface melts.

The results show that the ice deformation rate increases from a value equal to zero at the surface to a value of ~1.5 mm.yr^{-1} at a depth of ~12 m, at the beginning of the measurement period. The increase of the deformation rate with depth is not linear. This results from the cubic relationship between the deformation rate and the shear stress component (equation (§ 2.7)). The effects of temperature variations are very minor on the deformation rate at this depth level.

4 Discussion

The output product from the ice surface velocities interpolation at Hellstugubreen indicates that higher velocities are found between stake H20 (1634 m.a.s.l.) and stake H60 (1795 m.a.s.l.). In summer when the ice surface is snow-free, this area appears to be heavily crevassed. Crevasses are known to form under relatively large strain-rates (Wu and Christensen, 1964; Vaughan, 1993; Campbell et al., 2013). Higher surface velocities and surface velocities increasing over a short distance are associated to larger strain-rates. The transverse crevasses formation in this zone of Hellstugubreen is therefore consistent with higher local surface velocities. The highest surface velocity measured (15.8 m.yr^{-1}) was at stake H29. The lowest velocity recorded (0.5 m.yr^{-1}) was at stake H45, in a relatively flat area close to the ice divide separating Hellstugubreen from Vestre Memurubreen.

Both on Hellstugubreen and on Storbreen, the measured surface velocities can be hard to compare and interpret, as they are not derived from continuous measurements. They are averaged velocities, estimated over different periods and times of the year. Some represents yearly velocities, other estimates summer or even winter surface velocities. To assess surface velocity changes over time, the stake surveys should be conducted at regular time intervals, several times every year if one wants to get an insight into seasonal variations of the ice flow.

No general conclusion on surface velocity changes can be drawn from the stake surveys on Storbreen. Ice flow accelerations and decelerations are not synchronized between all stake locations. The highest velocity measured at stake S1yr12 (~18.3 m.yr^{-1}) is likely to result from observational error, as previous velocities measured at the neighbouring stake S1 were all lower than ~10 m.yr^{-1}. Likewise, the velocity drop observed at stake S4 and minimum velocity estimated on Storbreen (~0.2 m.yr^{-1}) is most likely based on a measurement error. Indeed, this deceleration of the ice flow occurs at the end of the summer, between the 13th of August and the 12th of September, when velocities are generally higher than the mean annual velocity. The earlier estimated value of ~2.6 m.yr^{-1} is therefore more sensible for the surface velocity at this stake location. The surface velocity values averaged over short periods are very sensitive to errors of measurement.

Velocity measurements had already been conducted on Storbreen in 1960s (Liestøl, 1967). The surface velocity was estimated using a triangulation method. The velocity values estimated ranged from a few millimetre per day to ~21.5 m.yr^{-1}. The highest velocity measured was downstream stake S7 (see *Figure § 2.7b*) at ~1640 ma.s.l. An overview map from this work is shown in *Appendix F.1.* Storbreen had a that time a different geometry and its ice thickness must have been larger at the today's stake locations.

For the parts of Hellstugubreen and Storbreen mapped with radio-echo sounding, the glacier has mainly a temperate basal thermal regime, which allow for basal sliding (see *Basal thermal regimes*). Therefore, part of the ice flow velocity observed at the surface may result from the basal sliding component. The temperate basal thermal regime can sustain a subglacial hydrological network, which lubricates the bed and increases the ice flow. On polythermal glaciers, the melt water input in subglacial drainage pathways during the summer season may result in significant increases of the horizontal surface velocity (Rabus and Echelmeyer, 1997; Copland et al., 2003). In order to isolate the effects of this speed-up event on averaged velocities, a higher frequency of stake surveys becomes even more important.

The ice temperature variations in the subsurface do not affect significantly the surface velocity. The changes of the creep flow parameter resulting from temperature variations are not important under low shear stress conditions for the calculation of the total surface velocity. However, under larger shear stress conditions, the thermal regime of the ice becomes an import contributor to the total surface velocity. At stake H45, where the ice is estimated to be $\sim$133 m thick and the CTS is encountered at a depth of $\sim$90 m, the measured surface velocity was only 0.5 m.yr^{-1}. If the stake was located far in the accumulation instead, where an insulating snowpack impedes the cold winter wave to penetrate deeper than the subsurface layers, the ice would likely have a temperate thermal regime across the full ice thickness (Hagen et al., 2003). With a temperate thermal regime under such stress conditions, the measured surface velocity at stake H45 may have been significantly higher. The snow line was however located in the uppermost part of the glacier during the last past years (Andreassen et al., 2011*a*). On the long term, the fluctuations of the snow line may be an important contributor to ice flow velocity variations on polythermal glaciers.

Conclusions

The aim of this work was to gain an insight into the thermal regime of Hellstugubreen and Storbreen, two glaciers in Jotunheimen area thought to be polythermal. The thermal regime assessments were based on Radio-Echo Sounding surveys at two different center frequencies and on shallow borehole temperature measurements. The RES measurements enabled to map the ice thickness and the Cold-temperate Transition Surface on a regional scale. The horizontal surface velocities on both glaciers were also investigated. The velocities were estimated from stake surveys based on non-continuous DGNSS georeferencing. The conclusions from this work can be briefly summarized as follows :

- Generally, the ice thickness estimated from RES measurements conducted in 2014 at Hellstugubreen are consistent with the corrected RES records from 2011. Only 5.1% of the ice thickness differences observed between the measurements for both years show a value greater than the total measurement uncertainty, with a mean absolute difference of 18 meters. The largest differences between measurements occurred where the ice was thickest.

- The borehole temperature measurements at Hellstugubreen indicated the presence of ice below the pressure-melting point in the subsurface. At stake H44, by the end of the summer, the cold winter wave is not eliminated at the depths investigated. At stake H13, the ice almost transited towards a temperate regime, but remained cold at a depth of 3.2 m, accounting for the measurement uncertainties. On Storbreen, at stake S2, the borehole measurements also pointed out the existence of a thin cold surface layer. At this location, however, the ice became temperate along the entire profile from the start of August.

- The RES surveys confirmed that both Hellstugubreen and Storbreen have a polythermal regime. The RES measurements were conducted the ablation area of these glaciers. The glaciers seemed cold-based at the front and near their margins. Beneath their central parts, the glaciers have a temperate basal thermal regime. Generally, the thickness of the cold surface layer increases up glacier, and reaches a maximum value of 90 m at Hellstugubreen and 55 m at Storbreen.

- The surface velocities estimated on Hellstugubreen for 2013-2014 ranges from 0.5 m.yr^{-1}, at stake H45 (1937 m.a.s.l.) near the ice divide between Hellstugubreen and Vestre Memurubreen, to a maximum value of 15.8 m.yr^{-1} at stake H29 (1743 m.a.s.l.). The surface velocities estimated on Storbreen ranged from nearly to 0 m.yr^{-1} to 18.3 m.yr^{-1}. However, these values are both suspected to result from measurement errors. Sensible values range from ∼2.5 m.yr^{-1} at stake S4 (1708 m.a.s.l.) up to ∼16.2 m.yr^{-1} at stake S6 (1851 m.a.s.l.).

- The ice temperature variations in the subsurface do not lead to large deformation rate differences. The ice temperature at shallow depths is therefore not an important factor in the surface velocities modelling of glaciers. The use of simple Degree-Day models is an efficient way to update the depth of the sensors in shallow borehole temperature measurements. However, the Degree-Day Factor used in the model requires to be well calibrated, preferably estimated from measurements over a long time period.

Further processing can be done on the radargrams from 2014, in order to improve the accuracy of both the ice thickness measurements and the mapping of the CTS. The CTS was mapped only in the lower part of Storbreen. The area investigated could be extended to the larger upper parts. However this may not be possible with the use of snowmobile owing to the steep topography and presence of crevasses. It would likewise be interesting to get an overview of the ice thermal regime in the two upper cirques at Hellstugubreen. The higher elevations of the cirques, together with the shadow from the surrounding topography lead to the presence of a snowpack more resistant to the summer melt. The thermal regime of these zones may therefore be less affected by the cold winter temperatures. Regarding the surface velocities, more frequent measurements and a denser stake network would greatly improve the quality of the output products.

References

Ahlmann, H. W. (1935), 'Contribution to the physics of glaciers', *Geographical Journal* **86**(2), 97–113.

Andreassen, L. M., Elvehøy, H., Jackson, M., Kjøllmoen, B., Tvede, A. M., Laumann, T. and Giesen, R. H. (2007), Storbreen, *in* B. Kjøllmoen, ed., 'Glaciological investigations in Norway in 2006', Vol. 1, Norwegian Water Resources and Energy Directorate, Oslo, p. 99.

Andreassen, L. M., Elvehøy, H., Jackson, M., Kjøllmoen, B. and Giesen, R. H. (2011*a*), Hellstugubreen, *in* B. Kjøllmoen, ed., 'Glaciological investigations in Norway in 2010', Vol. 3, Norwegian Water Resources and Energy Directorate, Oslo, p. 89.

Andreassen, L. M., Elvehøy, H., Jackson, M., Kjøllmoen, B. and Giesen, R. H. (2011*b*), Storbreen, *in* B. Kjøllmoen, ed., 'Glaciological investigations in Norway in 2010', Vol. 3, Norwegian Water Resources and Energy Directorate, Oslo, p. 89.

Andreassen, L. M., Huss, M., Melvold, K., Elvehøy, H. and Winsvold, S. (2015), 'Ice thickness measurements and volume estimates for glaciers in Norway', *Journal of Glaciology* **61**(228), 763–775.

Andreassen, L. M., Winsvold, S. H., Paul, F. and Hausberg, J. E. (2012), *Inventory of Norwegian Glaciers*, Norwegian Water Resources and Energy Directorate, OSLO.

Bamber, J. L. (1989), 'Ice/bed interface and englacial properties of Svalbard ice masses deduced from airborne radio-echo sounding data', *Journal of Glaciology* **35**, 30–39.

Benjumea, B., Macheret, Y. Y., Navarro, F. J. and Teixidó, T. (2003), 'Estimation of water content in a temperate glacier from radar and seismic sounding data', *Annals of Glaciology* **37**(1), 317–324.
URL: *http://www.ingentaconnect.com/content/igsoc/agl/2003/00000037/00000001/art00049*

Benn, D. I. and Evans, D. J. A. (2010), *Glaciers and Glaciation*, Hodder Arnold Publication, 2nd edn, Hodder Education.

Bingham, R. G., Nienow, P. W., Sharp, M. J. and Boon, S. (2005), 'Subglacial drainage processes at a High Arctic polythermal valley glacier', *Journal of Glaciology* **51**(172), 15–24.

URL: `http://www.ingentaconnect.com/content/igsoc/jog/2005/00000051/00000172/art00002`

Björnsson, H., Gjessing, Y., Hamran, S.-E., Hagen, J. O., Liestøl, O., Pálsson, F. and Erlingsson, B. (1996), 'The thermal regime of sub-polar glaciers mapped by multi-frequency radio-echo sounding', *Journal of Glaciology* **42**(140), 23–32.

Blatter, H. (1987), 'On the thermal regime of an Arctic valley glacier: A study of White Glacier, Axel Helberg Island, N.W.T., Canada', *Journal of Glaciology* **33**(114), 200–211.

Campbell, S., Roy, S., Kreutz, K., Arcone, S., Osterberg, E. and Koons, P. (2013), 'Strain-rate estimates for crevasse formation at an alpine ice divide:Mount Hunter, Alaska', *Annals of Glaciology* **54**(63), 200–208.

Conway, H. and Rasmussen, L. A. (2000), Summer temperature profiles within supraglacial debris on khumbu glacier, nepal, *in* 'Debris Covered Glaciers', number 264, International Association of Hydrological Sciences, pp. 89–97.

Copland, L., Sharp, M. J. and Nienow, P. W. (2003), 'Links between short-term velocity variations and the subglacial hydrology of a predominantly cold polythermal glacier', *Journal of Glaciology* **49**(166), 337–348.

Cuffey, K. M. and Paterson, W. S. B. (2010), *The Physics of Glaciers*, 4th edn, Elsevier.

Das, S. B., Joughin, I., Behn, M. D., Howat, I. M., King, M. A., Lizarralde, D. and Bhatia, M. P. (2008), 'Fracture propagation to the base of the Greenland Ice Sheet during supraglacial lake drainage', *Science* **320**(5877), 778–781.

Dowdeswell, J., Benham, T., Gorman, M., Burgess, D. and Sharp, M. (2004), 'Form and flow of the Devon Island Ice Cap, Canadian Arctic', *Journal of Geophysical Research* **109**.

Engelhardt, M. (2014), Glacier mass-balance and discharge modeling, PhD thesis, Faculty of Mathematics and Natural Sciences, University of Oslo.
URL: `http://folk.uio.no/markusen/disputas/Avhandling_Markus.pdf`

Etzelmüller, B. and Hagen, J. O. (2005), 'Glacier-permafrost interaction in Arctic and alpine mountain environments with examples from southern Norway and Svalbard', *Geological Society, London, Special Publications* **242**(1), 11–27.

Fountain, A. G. and Walder, J. S. (1998), 'Water flow through temperate glaciers', *Review of Geophysics* **3**, 299–328.

Funk, M., Echelmeyer, K. A. and Iken, A. (1994), 'Mechanisms of fast flow in Jakobshavn Isbrae, West Greenland : Part ii. Measurements of temperature and water-level in deep boreholes', *Journal of Glaciology* **40**(136), 569–585.

Gilbert, A., Vincent, C., Wagnon, P., Thibert, E. and Rabatel, A. (2012), 'The influence of snow cover thickness on the thermal regime of Tête Rousse Glacier (Mont Blanc range, 3200 m a.s.l.): Consequences for outburst flood hazards and glacier response to climate change', *Journal of Geophysical Research: Earth Surface* **117**(F4), n/a–n/a. F04018.
URL: `http://dx.doi.org/10.1029/2011JF002258`

Glen, J. and Paren, J. (1975), 'The electrical properties of snow and ice', *Journal of Glaciology* **15**(73), 15–385.

Glen, J. W. (1955), The creep of polycrystalline ice, *in* 'Proceedings of the Royal Society of London', Vol. 228, pp. 519–538.

Haeberli, W. and Funk, M. (1991), 'Borehole temperatures at the Colle Gnifetti core-drilling site (Monte Rosa, Swiss Alps)', *Journal of Glaciology* **37**(125), 37–46.

Hagen, J. O., Kohler, J., Melvold, K. and J.-G., W. (2003), 'Glaciers in Svalbard : mass balance, runoff and freshwater flux', *Polar Research* **22**(2), 145–159.

Hughes, M. (2008), Determination of glacial-ice temperature profiles using radar and an antenna-gain estimation technique, Master's thesis, University of Kansas.

Huybrechts, P. and Oerlemans, J. (1988), 'Evolution of the East Antarctic ice sheet: a numerical study of thermo-mechanical response patterns with changing climate', *Annals of Glaciology* **11**, 52–59.

Iken, A., Echelmeyer, K. A., Harrison, W. D. and Funk, M. (1993), 'Mechanisms of fast flow in Jakobshavn Isbrae, West Greenland : Part i. Measurements of temperature and water-level in deep boreholes', *Journal of Glaciology* **39**(131), 15–25.

Irvine-Fynn, T. D. L., Hodson, A. J., Moorman, B. J., Vatne, G. and Hubbard, A. L. (2011), 'Polythermal glacier hydrology: A review', *Reviews of Geophysics* **49**(4), n/a–n/a.
URL: `http://dx.doi.org/10.1029/2010RG000350`

Jania, J., Mochnacki, D. and Gądek, B. (1996), 'The thermal structure of Hansbreen, a tidewater glacier in southern Spitsbergen, Svalbard', *Polar Research* **15**(1), 53–66.

Jol, H., ed. (2009), *Ground Penetrating Radar: Theory and Applications*, Elsevier Science, Amsterdam.

Kennett, M., Laumann, T. and Cecile, L. (1993), ‘Helicopter-borne radio-echo sounding of Svartisen, Norway’, *Annals of Glaciology* **17**, 23–26.

King, M. A. and Watson, C. S. (2010), ‘Long GPS coordinate time series: Multipath and geometry effects’, *Journal of Geophysical Research: Solid Earth* **115**(B4), n/a–n/a. B04403.
URL: `http://dx.doi.org/10.1029/2009JB006543`

Kohler, J., Moore, J., Kennett, M., Engeset, R. and Elvehøy, H. (1997), ‘Using ground-penetrating radar to image previous years’ summer surfaces for mass-balance measurements’, *Annals of Glaciology* **24**, 355–360.

Liestøl, O. (1967), ‘Storbreen glacier in Jotunheimen, Norway’, *Norsk Polarinstitutt Skrifter* **141**, 63p.

Liestøl, O. (1988), ‘The glaciers in the kongsfjorden area, spitsbergen’, *Norsk Geografisk Tidsskrift* **42**(4), 231–238.

Lovell, H., Fleming, E. J., Benn, D. I., Hubbard, B., Lukas, S. and Naegeli, K. (2015), ‘Former dynamic behaviour of a cold-based valley glacier on Svalbard revealed by basal ice and structural glaciology investigations’, *Journal of Glaciology* **61**(226), 309–328.
URL: `http://www.ingentaconnect.com/content/igsoc/jog/2015/00000061/00000226/art00011`

Lüthi, M., Funk, M., Iken, A., Gogineni, S. and Truffer, M. (2002), ‘Mechanisms of fast flow in Jakobshavn Isbrae, West Greenland : Part iii. Measurements of temperature and water-level in deep boreholes’, *Journal of Glaciology* **48**(162), 369–385.

Maohuan, H. (1990), ‘On the temperature distribution of glaciers in China’, *Journal of Glaciology* **36**(123), 210–216.

Maohuan, H. (1999), ‘Forty year’s study of glacier temperature distribution in China : Review and Suggestions’, *Journal of Glaciology and Geocryology* **21**(4), 310–317.

Moore, J. C., Pälli, A., Ludwig, F., Blatter, H., Jania, J., Gadek, B., Glowacki, P., Mochnacki, D. and Isaksson, E. (1999), ‘High-resolution hydrothermal structure of Hansbreen, Spitsbergen, mapped by ground-penetrating radar’, *Journal of Glaciology* **45**, 524–532.
URL: `http://www.igsoc.org:8080/journal/45/151/igs_journal_vol45_issue151_pg524-532.pdf`

Moran, M. L., Greenfield, R. J., Arcone, S. A. and Delaney, A. J. (2000), ‘Delineation of a complexly dipping temperate glacier bed using short-pulse radar arrays’, *Journal of Glaciology*

46(153), 274–286.
URL: *http://www.ingentaconnect.com/content/igsoc/jog/2000/00000046/00000153/art00012*

Murray, T., Gooch, D. L. and Stuart, G. W. (1997), 'Structures within the surge front at Bakaninbreen, Svalbard, using ground-penetrating radar', *Annals of Glaciology* **24**, 122–129.
URL: *http://www.igsoc.org:8080/annals/24/igs_annals_vol24_year1997_pg122-129.pdf*

Murray, T., Stuart, G. W., Fry, M., Gamble, N. H. and Crabtree, M. D. (2000), 'Englacial water distribution in a temperate glacier from surface and borehole radar velocity analysis', *Journal of Glaciology* **46**(154), 389–398.

Navarro, F. and Eisen, O. (2009), *Remote Sensing of Glaciers – Techniques for Topographic, Spatial and Thematic Mapping*, Taylor & Francis Group, London, chapter Ground-penetrating radar in glaciological applications, pp. 195–229.

Navarro, F. J., Macheret, Y. Y. and Benjumea, B. (2005), 'Application of radar and seismic methods for the investigation of temperate glaciers', *Journal of Applied Geophysics* **57**(3), 193 – 211.
URL: *http://www.sciencedirect.com/science/article/pii/S0926985104000989*

Navarro, F. J., Martín-Español, A., Lapazaran, J. J., Grabiec, M., Otero, J., Vasilenko, E. V. and Puczko, D. (2014), 'Ice Volume Estimates from Ground-Penetrating Radar Surveys, Wedel Jarlsberg Land Glaciers, Svalbard', *Arctic, Antarctic, and Alpine Research* **46**(2), 394–406.
URL: *http://www.bioone.org/doi/full/10.1657/1938-4246-46.2.394*

Nilsson, J. (2011), Multipath mitigation of carrier-phase GPS position estimates from the Helheim glacier: using new reduced sidereal filtering approach, Master's thesis, Chalmers University of Technology, Department of Earth and Space Sciences, Gothenburg, Sweden.
URL: *http://publications.lib.chalmers.se/records/fulltext/164985.pdf*

Nye, J. F. (1973), Water at the bed of a glacier, *in* 'Symposium on the Hydrology of Glaciers', Vol. 95, International Association of Hydrological Sciences, pp. 189–194.

Ødegård, R., Nesje, A., Isaksen, K. and Eiken., T. (2011), Perennial ice patch studies - preliminary results from a case study in Jotunheimen, southern Norway, *in* 'Geophysical Research Abstracts', Vol. 13. EGU2011-12027.

Ødegård, R. S., Hamran, S.-E., Bø, P. H., Etzelmuller, B., Vatne, G. and Sollid, J. L. (1992), 'Thermal regime of a valley glacier, erikbreen, northern spitsbergen', *Polar Research* **11**(2), 69–79.
URL: *http://dx.doi.org/10.1111/j.1751-8369.1992.tb00413.x*

Onset Computer Corporation (2014), 'HOBO Pro v2 Data Logger (U23-00x) User's Manual', http://www.onsetcomp.com/files/manual_pdfs/10694-N%20MAN-U23.pdf. Accessed: 2014-09-07.

Palli, A., Kohler, J., Isaksson, E., Moore, J., Pinglot, J., Pohjola, V. and Samuelsson, H. (2002), 'Spatial and temporal variability of snow accumulation using ground-penetrating radar and ice cores on a Svalbard glacier', *Journal of Glaciology* **48**(162), 417–424.

Paterson, W. S. B. (1968), A temperature profile through the Meighen ice cap, Arctic Canada, *in* 'ASH General Assembly of Bern, Commission of Snow and Ice 1967.', number 79, Association Internationale d'Hydrologie Scientifique, Gentbrugge, Belgium, pp. 440–449.

Pattyn, F. (2010), 'Antarctic subglacial conditions inferred from a hybrid ice sheet/ice stream model', *Earth and Planetary Science Letters* **295**(3–4), 451 – 461.
URL: *http://www.sciencedirect.com/science/article/pii/S0012821X10002712*

Pay, I. (2014), Changes in driving stresses and horizontal surface velocity on Hellstugubreen, Jotunheimen, Norway. An investigation of inter-decadal fluctuations., Master's thesis, Norges Teknisk-Naturvitenskapelige Universitet, Trondheim. Unpublished.

Pettersson, R., Christoffersen, P., Dowdeswell, J. A., Pohjola, V. A., Hubbard, A. and Strozzi, T. (2011), 'Ice thickness and basal conditions of vestfonna ice cap, eastern svalbard', *Geografiska Annaler: Series A, Physical Geography* **93**(4), 311–322.
URL: *http://dx.doi.org/10.1111/j.1468-0459.2011.00438.x*

Pettersson, R., Jansson, P. and Blatter, H. (2004), 'Spatial variability in water content at the cold-temperate transition surface of the polythermal Storglaciären, Sweden', *Journal of Geophysical Research* **109**(F02009). Part of urn:nbn:se:su:diva-161.

Pettersson, R., Jansson, P. and Holmlund, P. (2003), 'Cold surface layer thinning on Storglaciären, Sweden, observed by repeated ground penetrating radar surveys', *Journal of Geophysical Research: Earth Surface* **108**(F1), n/a–n/a. 6004.
URL: *http://dx.doi.org/10.1029/2003JF000024*

Phillips, T., Rajaram, H., Colgan, W., Steffen, K. and Abdalati, W. (2013), 'Evaluation of cryo-hydrologic warming as an explanation for increased ice velocities in the wet snow zone, sermeq avannarleq, west greenland', *Journal of Geophysical Research: Earth Surface* **118**, 1241–1256.

Phillips, T., Rajaram, H. and Steffen, K. (2010), 'Cryo-hydrologic warming: A potential mechanism for rapid thermal response of ice sheets', *Geophysical Research Letters* **37**(20), n/a–n/a. L20503.
URL: `http://dx.doi.org/10.1029/2010GL044397`

Pinglot, J., Hagen, J., Melvold, K., Eiken, T. and Vincent, C. (2001), 'A mean net accumulation pattern derived from radioactive layers and radar soundings on Austfonna, Nordaustlandet, Svalbard', *Journal of Glaciology* **47**(159), 555–566.

Plewes, L. and Hubbard, B. (2001), 'A review of the use of radio-echo sounding in glaciology', *Progress in Physical Geography* **25**(2), 203–236.

Price, P. B., Nagornov, O. V., Bay, R., Chirkin, D., He, Y., Miocinovic, P., Richards, A., Woschnagg, K., Koci, B. and Zagorodnov, V. (2002), Temperature profile for glacial ice at the South Pole: Implications for life in a nearby subglacial lake, *in* 'Proceedings- National Academy of Sciences USA', Vol. 99, National Academy of Sciences, pp. 7844–7847.

Rabus, B. T. and Echelmeyer, K. A. (1997), 'The flow of polythermal glacier: McCall Glacier, Alaska', *Journal of Glaciology* **43**(145), 552–536.

Robin, G. D. Q., Evans, S. and Bailey, J. T. (1969), 'Interpretation of Radio Echo Sounding in Polar Ice Sheets', *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* **265**(1166), 437–505.
URL: `http://www.jstor.org/stable/73767`

Robinson, P. H. (1984), 'Ice dynamics and thermal regime of taylor glacier, South Victoria land, Antarctica', *Journal of Glaciology* **30**(105), 153–160.

Rolandone, R., Mareschal, J.-C. and Jaupart, C. (2003), 'Temperatures at the base of the Laurentide Ice Sheet inferred from borehole temperature data', *Geophysical Research Letters* **30**(18), CRY 3.

Röthlisberger, H. (1972), 'Water pressure in intra- and subglacial channels', *Journal of Glaciology* **11**(62), 177–203.

Sætrang, A. C. and Wold, B. (1986), 'Results from the radio echo-sounding on parts of the Jostedalsbreen ice cap, Norway', *Annals of Glaciology* **8**, 156–158.

URL: *http://www.igsoc.org:8080/annals/8/igs_annals_vol08_year1985_pg156-158.pdf*

Sandmeier, K. J. (2014), Reflexw 7.5 manual, Technical report, Sandmeier scientific software, Zipser Strasse 1, 76227 Karlsruhe, Germany.
URL: *http://www.sandmeier-geo.de/Download/reflexw_manual_a4.pdf*

Schytt, V. (1964), 'Scientific Results of the Swedish glaciological expedition to Nordaustlandet, Spitsbergen 1957 and 1958', *Geografiska Annaler* **46**(3), 242–281.

Sclater, J. G., Jaupart, C. and Galson, D. (1980), 'The heat flow through oceanic and continental crust and the heat loss of the earth', *Reviews of Geophysics* **18**, 269–311.

Shannon, S. R., Payne, A. J., Bartholomew, I. D., Van Den Broeke, M. R., Edwards, T. L., Fettweis, X., Gagliardini, O., Gillet-Chaulet, F., Goelzer, H., Hoffman, M. J., Huybrechts, P., Mair, D. W. F., Nienow, P. W., Perego, M., Price, S. F., Smeets, C. J. P. P., Sole, A. J., Van De Wal, R. S. W. and Zwinger, T. (2013), Enhanced basal lubrication and the contribution of the Greenland ice sheet to future sea-level rise, *in* 'Proceedings- National Academy of Sciences USA', Vol. 110, National Academy of Sciences, pp. 14156–14161.

Sheriff, R. E. and Geldart, L. P. (1995), *Exploration Seismology*, 2nd edn, Cambridge University Press, Cambridge.

Sørdal, I. (2013), Kartlegging av temperaturtilhøva i Gråsubreen og Juvfonne, Master's thesis, University of Oslo.

Urbini, S., Cafarella, L., Zirizzotti, A., Bianchi, C., Tabacco, I. and Frezzotti, M. (2006), 'Location of a new ice core site at Talos Dome (East Antarctica)', *Annals of Geophysics* **49**(4-5), 1133 – 1138.
URL: *http://www.annalsofgeophysics.eu/index.php/annals/article/view/3104*

Urdahl, H. (2005), Temperaturregime og stabilitet med henblikk på isskred fra hengebreer- eksempel fra Steindalsnosi, Sognefjellet, Vest Norge, Master's thesis, University of Oslo.

van de Wal, R. S. W., Boot, W., van den Broeke, M. R., Smeets, C. J. P. P., Reijmer, C. H., Donker, J. J. A. and Oerlemans, J. (2008), 'Large and rapid melt-induced velocity changes in the ablation zone of the Greenland Ice Sheet', *Science* **321**(5885), 111–113.

Van Dusen, M. S. (1929), *International Critical Tables of Numerical Data, Physics, Chemistry and Technology*, Vol. 5, McGraw Hill, New York, chapter Thermal conductivity of non-metallic solids, pp. 216–217.

Vaughan, D. G. (1993), ‘Relating the occurrence of crevasses to surface strain rates’, *Journal of Glaciology* **39**(132), 255–266.

Watts, R. D. and England, A. W. (1976), ‘Radio-echo sounding of temperate glaciers: ice properties and sounder design criteria’, *Journal of Glaciology* **17**(75), 39–48.

Welch, B., Pfeffer, W., Harper, J. and Humphrey, N. (1998), ‘Mapping subglacial surfaces below temperate valley glaciers using 3-dimensional radio-echo sounding techniques’, *Journal of Glaciology* **44**(146), 164–170.

Wu, T. H. and Christensen, R. W. (1964), ‘Measurement of surface strain-rate on Taku Glacier, Alaska’, *Journal of Glaciology* **5**(39), 305–313.

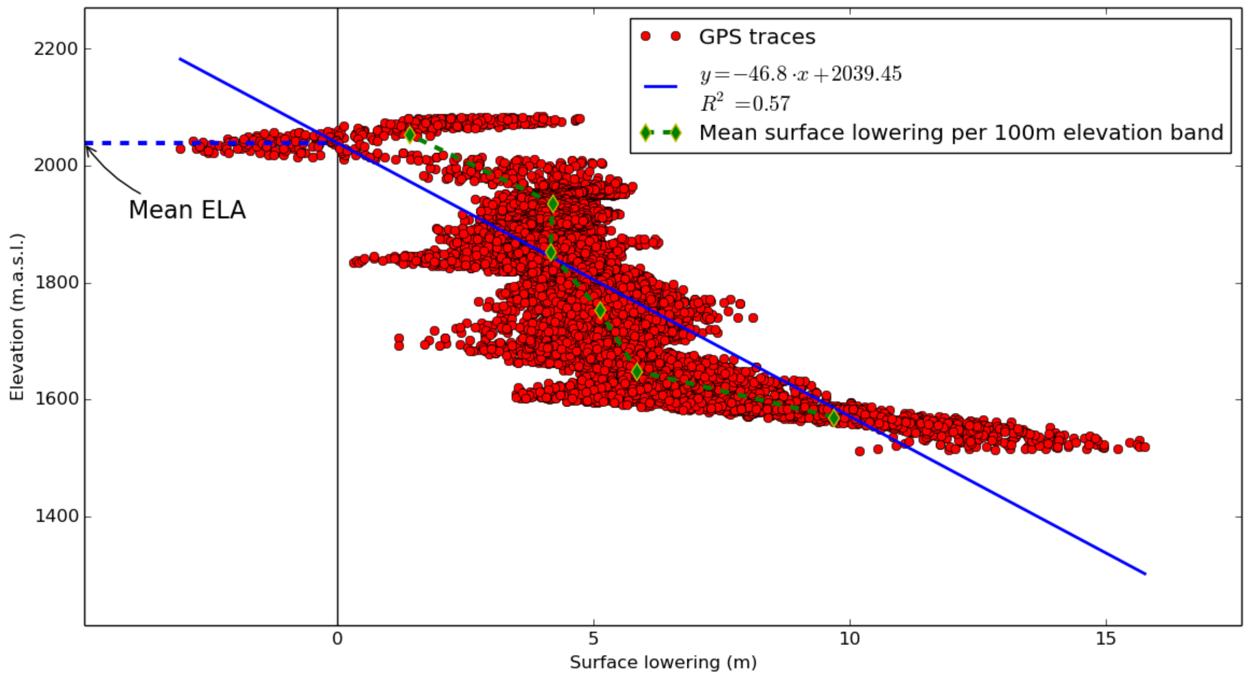
Zwally, H. J., Abdalati, W., Herring, T., Larson, K., Saba, J. and Steffen, K. (2002), ‘Surface melt-induced acceleration of Greenland ice-sheet flow’, *Science* **297**(5579), 218–222.

Appendices

Appendix A

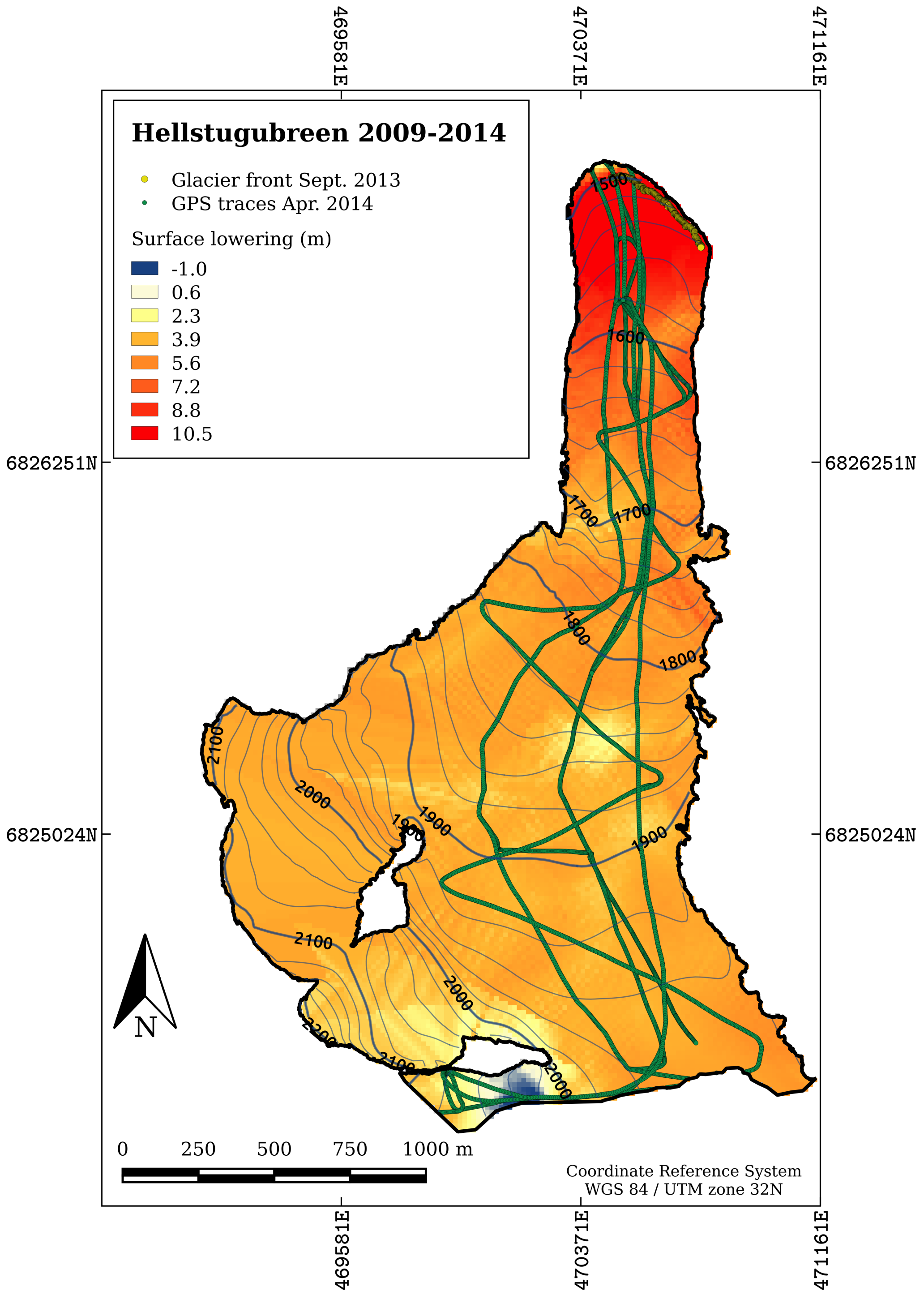
Surface lowering at Hellstugubreen

A.1 Surface lowering gradient



Surface lowering gradient at Hellstugubreen for the period 2009-2014, derived from GPS measurements and LiDAR data differentiation. The elevation on the y-axis is derived from the 2009 LiDAR data.

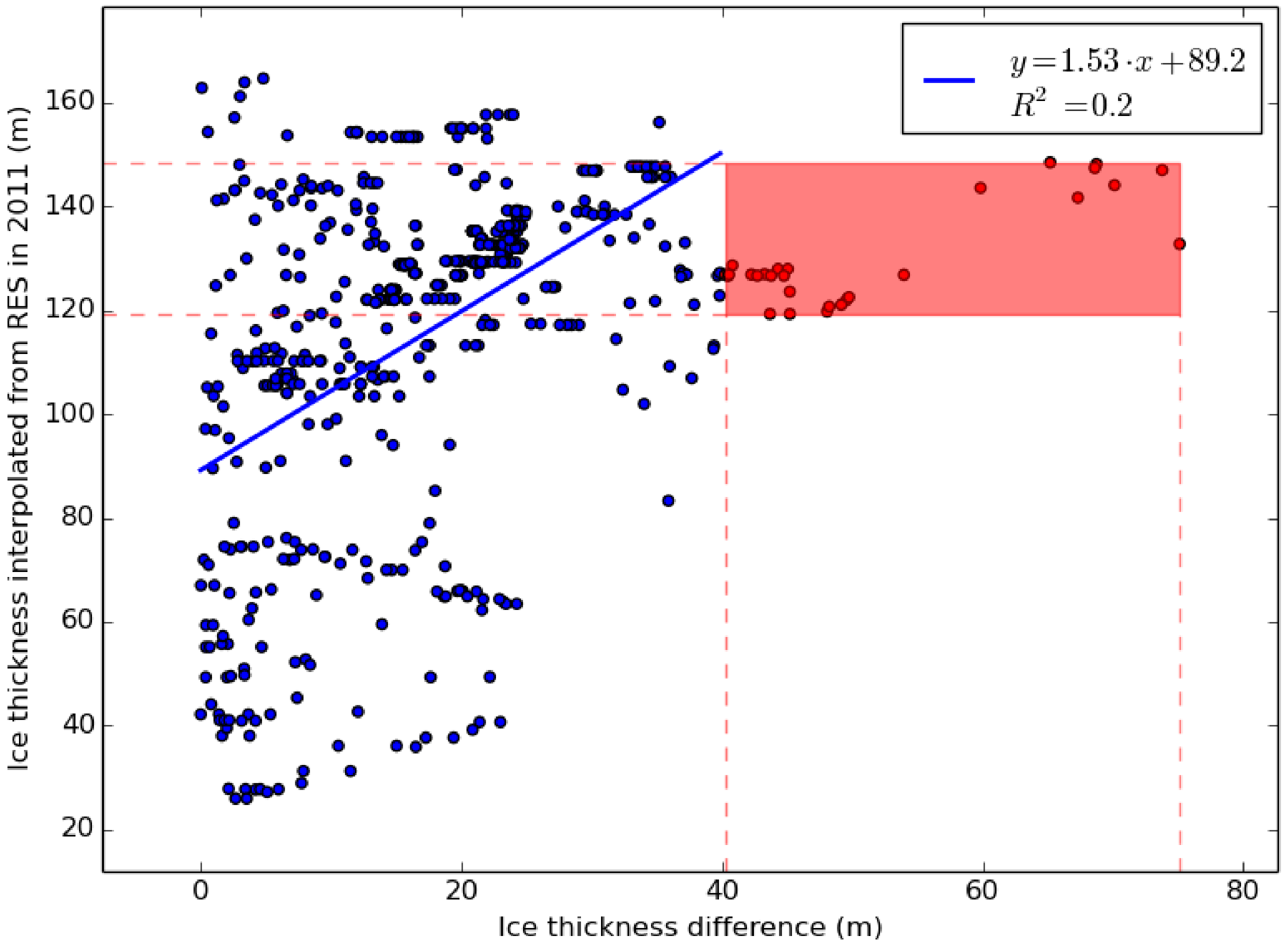
A.2 Surface lowering map



Surface lowering map at Hellstugubreen for the period 2009-2014, derived from GPS measurements and LiDAR data differentiation. The elevation contours and glacier outlines are derived from the 2009 laser scanning and orthophotos (data : NVE).

Appendix B

Ice thickness differences between RES measurements from 2011 and 2014, Hellstugubreen

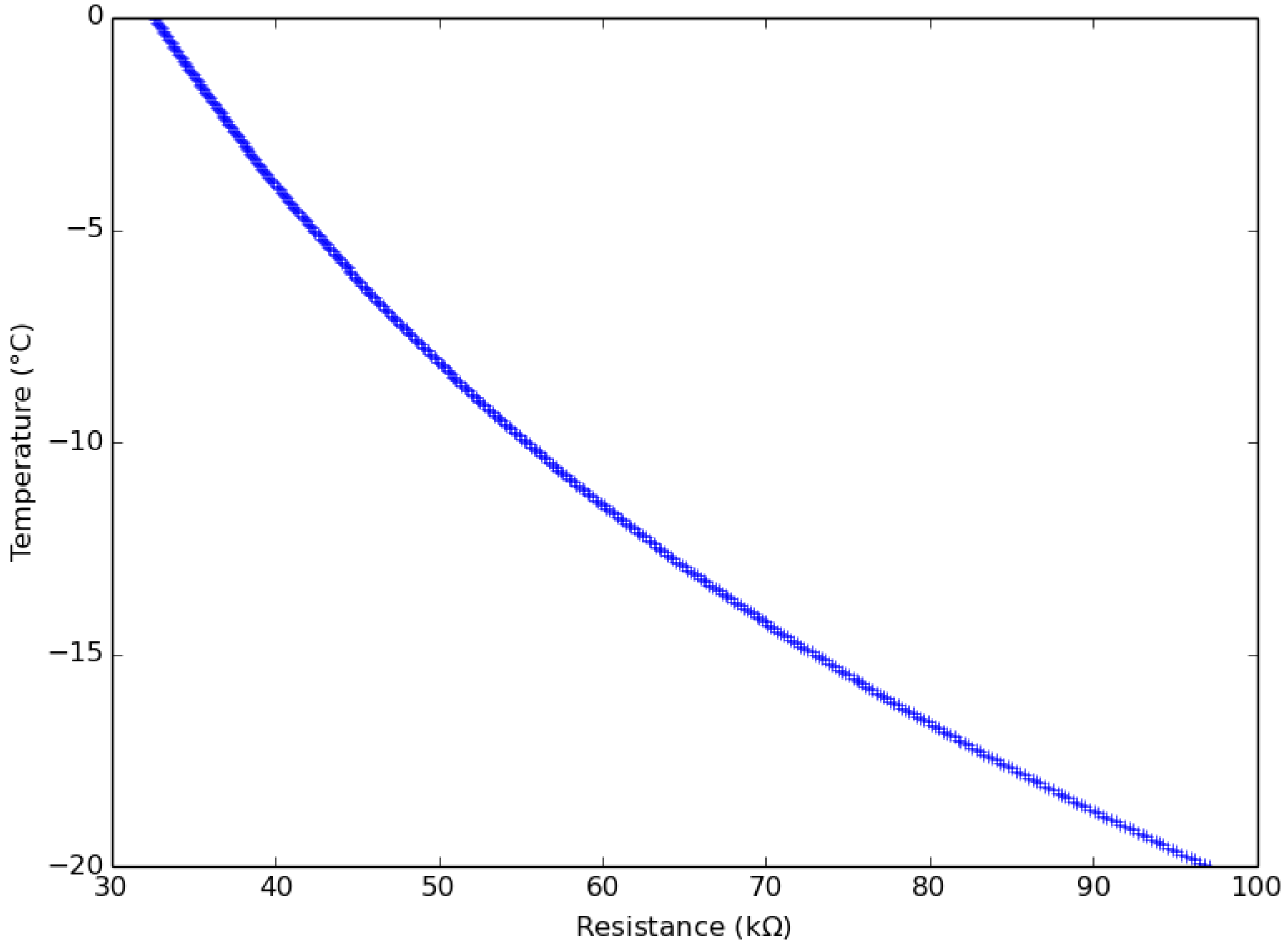


Ice thickness from 2011 measurements plotted against thickness differences observed between RES records from 2011 and 2014.

Appendix C

Ice and air temperature at Hellstugubreen

C.1 NTC thermistors calibration curve



Calibration curve for the thermistors PR103J2 for a temperature ranging between -20 and 0°C.

C.2 Python code

temperature.py

Listing 1: This program converts the resistance values from the thermistors into temperature values. It also formats the temperature data and allows to update the depth of the sensor.

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import os
import fileinput
import re
import pickle
import time
import datetime


import functionstemperature as ft


## Identification of the temperature string location
ID = 'H13'

## Indicate the date of the field measurements date =
## datetime.datetime(year,month(1-12),dom(1-31),hours(0-23),minutes(0-59))
date = datetime.datetime(2014,4,2,10,30)

## Number of sensors on the string
num = 7

## Defines the position of the sensors on the temperature string
## The position of the sensors are relative to the uppermost one
## Assign 'True' to equidistant if the sensors are equidistant,
## else assign 'False'
equidistant = True
loc = 2 # Value in meters that separates neighbouring sensors if
        #equidistant

if equidistant:
    dist = dict()
    key = 1
    pos = 0
    count = 0
    while count < num:
        dist[key] = pos
        key += 1
        pos += loc
        count += 1

## Dictionary containing positions of the sensors if not equidistant
## Key 1 is for the uppermost (closest to surface) sensor, key 2 the
## one below and so on... The position is relative to the sensor 1. If
## the sensor 2 is 3.5 meters away from sensor 1 on the line, the
## value 3.5 should be assigned to key 2 (dist[2])
else:
    dist = {1:0, 2:2}

## Update the depth of all sensors Choose one sensor that you want to
## update (its number on the line), and indicate its depth in meters
## Note : A negative value can be used to tell how far out of the
## borehole the sensor is
sensor = 1
sensor_depth = 2.23
depth = ft.get_depth(dist, sensor, sensor_depth)

## Read the original calibration curve
calibration = ft.read_calibrationc('calibration_curve')

## Correct the calibration curve for each sensor : indicate the
## resistance value for each sensor at 0 degree C
zero_degree = {1:32.7, 2:32.6, 3:32.6, 4:32.6, 5:32.6, 6:32.6, 7:32.6}

## Indicate resistance values measured by the sensors, manual input
## for the first time, in order to create the main class
res_values = {1:38.3, 2:36.9, 3:35.1, 4:32.9, 5:32.8, 6:32.8, 7:32.8}

## Convert the resistance values into temperature values
Ts = dict()
key = 1
count = 0
while count < num:
        new_calib = ft.sensor_calib(zero_degree[key], calibration)
        Ts[key] = ft.res_in_temp(res_values[key], new_calib)
        key += 1
        count += 1

## Main class
class TempProfile:

    """ A profile is defined by its ID (ID), the number of sensors
    (num), the distance between sensors (dist), their depth (depth),
    their temperature values (Ts) and the date of the temperature
    measurements (date). The calibration of the sensors is included in
    the last attribute (zero_degree), which represents the resistance
    values indicated by the sensors at 0 °C.
    """

    def __init__(self, ID, num, dist, depth, Ts, date, zero_degree):
        """ Create the object attributes
        Keyword Arguments:
        ID          --
        num         --
        dist        --
        depth       --
        Ts          --
        date        --
        zero_degree --
        """
        ## Creates data folder in current directory if it does not exists
        if not os.path.exists('data'):
            os.makedirs('data')

        ## Check that there is no thermistor string with the same ID
        if not os.path.exists('data/{}'.format(ID)):
            if not ID.isalnum() or len(ID) < 2:
                raise AttributeError('ID not valid! It should be '
                'alphanumeric and at least two caracters.')
            else:
                self._ID = ID
        else:
            raise AttributeError('A thermistor string has already this'
            ' ID, choose another ID.')

        ## Check that the number of sensors is superior to 0 and is an integer value
        try:
            num = int(num)
        except ValueError:
            print('The value entered must be an integer!')
        else:
            if num < 1 :
                raise AttributeError('There must be at least one sensor!')
            else:
                self._num = num

        ## Check that the attribute 'dist' has the right format
        if not isinstance(dist,dict):
            raise AttributeError('The attribute dist must be a '
                                 'dictionary!')
        for value in dist.values():
            try:
                value = float(value)
            except ValueError:
                print('The distances must either be integers of '
                      'floatting numbers!')
        count = 1
        for key in dist.keys():
            if key != count:
                raise AttributeError('The keys of the dictionary '
                '"dist" must be integers,\nstarting from 1 (uppermost'
                ' sensor), and incremented by 1 every next key.')
            count += 1
        self._dist = dist

        ## Check that the attribute 'depth' has the right format
        if not isinstance(depth,dict):
            raise AttributeError('The attribute depth must be a '
                                 'dictionary!')
        for value in depth.values():
            try:
                value = float(value)
            except ValueError:
                print('The depth values must either be integers of '
                      'floatting numbers!')
        count = 1
        for key in depth.keys():
            if key != count:
                raise AttributeError('The keys of the dictionary '
                '"depth" must be integers,\nstarting from 1 (uppermost'
                ' sensor), and incremented by 1 every next key.')
            count += 1
        self._depth = depth

        ## Check that the attribute 'Ts' has the right format
        if not isinstance(Ts,dict):
            raise AttributeError('The attribute Ts must be a '
                                 'dictionary!')
        for value in Ts.values():
            try:
                value = float(value)
            except ValueError:
                print('The temperature values must either be integers'
                ' of floatting numbers!')
        count = 1
        for key in Ts.keys():
            if key != count:
                raise AttributeError('The keys of the dictionary "Ts"'
                ' must be integers,\nstarting from 1 (uppermost sensor),'
                ' and incremented by 1 every next key.')
            count += 1
        self._Ts = Ts

        ## Check that the attribute 'date' has the right format
        if not isinstance(date,datetime.datetime):
            raise AttributeError('The date of temperature measurements'
            ' must be at the format datetime.datetime.\n'
            'e.g. date = datetime.datetime(year,month,dayofmonth[,'
            'hours[,minutes]])')
        self._date = date

        ## Check that the attribute 'zero_degree' has the right format
        if not isinstance(zero_degree,dict):
            raise AttributeError('The attribute zero_degree must be a'
            ' dictionary!')
        for value in zero_degree.values():
            try:
                value = float(value)
            except ValueError:
                print('The resistance values of the sensors at 0 °C'
                ' must either be integers or floatting numbers!')
        count = 1
        for key in Ts.keys():
            if key != count:
                raise AttributeError('The keys of the dictionary '
                '"zero_degree" must be integers,\nstarting from 1 '
                '(uppermost sensor), and incremented by 1 every next'
                ' key.')
            count += 1
        self._zero_degree = zero_degree

        ## Creates a specific folder for the data of the temperature string
        if not os.path.exists('data/{}'.format(ID)):
            os.chmod('data',0o777)
            os.makedirs('data/{}'.format(ID))

        ## Save the object in a file
        with open('data/{0}/{0}_object'.format(ID),'wb') as file_object:
            my_pickler = pickle.Pickler(file_object)
            my_pickler.dump(self)

        ## Write attributes in a text file
        self._headers = ('TempString: {} (depth in meters and temp in '
        '°C)\nTime\t\t').format(ID)
        count = 0
        while count < self._num:
            self._headers += '\tdepth,temp'
            count += 1

        with open('data/{0}/{0}.txt'.format(ID),'w') as file_txt:
            file_txt.write(self._headers)
            count = 0
            file_txt.write('\n{}'.format(str(self._date)))
            while count < self._num:
                index = count + 1
                file_txt.write('\t{},{}'.format(self._depth[index],
                                                self._Ts[index]))
                count += 1
            file_txt.write('\n')
        ## Protect the files and directories created from writing by
        ## changing permissions
        ft.protect(ID)

    ## Definition of properties for the attributes
    def ID():
        doc = """Property : Identification of the thermistor string"""
        def fget(self):
            print('The identification of this thermistor string is : {}'\
                  .format(self._ID))
            return self._ID
        def fset(self, value):
            print('The Identification of a thermistor string cannot'
                  ' be changed!')
        def fdel(self):
            print('You cannot delete the ID of a thermistor string!')
        return locals()

    ID = property(**ID())

    def num():
        doc = """Property : Number of sensors on the thermistor string"""
        def fget(self):
            print('The number of sensors on the thermistor string {}'
                  ' is {}'.format(self._ID,self._num))
            return self._num
        def fset(self, value):
            print('You cannot changed the number of sensors of the'
                  ' thermistor string!')
        def fdel(self):
            print('You cannot delete the number of sensors of the'
                  ' thermistor string!')
        return locals()

    num = property(**num())

    def dist():
        doc = """Property : Distance between sensors on the thermistor
        string (in meters)"""
        def fget(self):
            distances = dict(self._dist)
            for key,value in distances.items():
                distances[key] = str(distances[key]) + ' m'
            print("""The distance between sensors on the thermistor
            string is given in meters by the dictionary :
            {}

            Sensor 1 is the uppermost sensor on the line (closest to
            surface), the distance given to the other sensors is
            relative to sensor 1.""".format(distances))
            return self._dist
        def fset(self, value):
            print('You cannot change the distance between sensors on'
                  ' the line!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    dist = property(**dist())

    def depth():
        doc = """Property : Depth of the sensors on the thermistor
        string (in meters)"""
        def fget(self):
            depths = dict(self._depth)
            for key,value in depths.items():
                depths[key] = str(depths[key]) + ' m'
            print("""The depth of the sensors on the thermistor string
            is given in meters by the dictionary :
            {}

            Sensor 1 is the uppermost sensor on the line (closest to
            surface).""".format(depths))
            return self._depth
        def fset(self, value):
            print('The "depth" attribute cannot be modified by '
                  're-assignment!\nUse the class method'
                  ' update_depth() instead.')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    depth = property(**depth())

    def Ts():
        doc = """Property : Temperature measured by the sensors on the
        thermistor string (in degrees Celsius)"""
        def fget(self):
            temperatures = dict(self._Ts)
            for key,value in temperatures.items():
                temperatures[key] = str(temperatures[key]) + ' °C'
            print("""The temperature values measured by the sensors on
            the thermistor string are given in degrees Celsius by the
            dictionary :
            {}

            Sensor 1 is the uppermost sensor on the line (closest to
            surface).""".format(temperatures))
            return self._Ts
        def fset(self, value):
            print('The "Ts" attribute cannot be modified by '
                  're-assignment!\nUse the class method'
                  ' update_temperature() instead.')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

```

```python
    Ts = property(**Ts())

    def date():
        doc = """Property : Date of the temperature measurements"""
        def fget(self):
            print('The temperature measurements were performed at '
            'this date :\n{}'.format(self._date))
            return self._date
        def fset(self, value):
            print('The "date" attribut cannot be modified by '
            're-assignment!\nUse one of the two following class '
            'methods to update the temperature profile:\n'
            'update_depth()\n'
            'update_temperature()')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    date = property(**date())

    def zero_degree():
        doc = """Property : Calibration of the sensors"""
        def fget(self):
            calibration = dict(self._zero_degree)
            for key,value in calibration.items():
                calibration[key] = str(calibration[key]) + ' kiloOhms'
            print('The "zero_degree" attribute represents the sensor '
            'calibrations\n(resistance values in kiloOhms at 0 °C) :\n'
            '{}\n\n'
            'Sensor 1 is the uppermost sensor on the line (closest to '
            'surface).'.format(calibration))
            return self._zero_degree
        def fset(self, value):
            print('You cannot change the "zero_degree" attribute '
            '(sensor calibration) !')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    zero_degree = property(**zero_degree())


    def __repr__(self):
        """ Function called when entering the class object directly in
        the interpreter.

        It is meant to ease the debug. It lists the most important
        attributes of the object.
        """
        return ('Temperature string, object of the class "TempProfile"'
                '\n\n'
                'ID:\n{0}\n\n'
                'number of sensors:\n{1}\n\n'
                'depths of the sensors:\n{2}\n\n'
                'temperature measured lastly by the sensors:\n{3}\n\n'
                'date of the last field measurements:\n{4}\n\n'
                'calibration (resistance at 0°C):\n{5}\n\n'\
                .format(self._ID,self._num,self._depth,self._Ts,
                        self._date,self._zero_degree))


    @classmethod
    def strings_list(cls):
        """ This method lists the existing thermistor strings.
        """
        if not os.path.exists('data'):
            print('No thermistor string has been created yet.')
        else:
            existing_strings = [d for d in os.listdir('data/')
                                  if os.path.isdir('data/{}'.format(d))]
            existing_strings.sort()
            if len(existing_strings) > 0:
                print('There is/are {} existing thermistor string(s) :'\
                      .format(len(existing_strings)))
                for string in existing_strings:
                    print(string)
            else:
                print('No thermistor string has been created yet.')


    @classmethod
    def update_depth(cls):
        """This method enables to update the depth of the sensors in
        the ice. The depth must be given in meters (floating or
        integer value). The sensor 1 is the upppermost sensor (closest
        to the surface or the furthest out of the ice).
        """
        ## Update the depth
        if not os.path.exists('data'):
            raise NameError('No thermistor string has been created'
                            ' yet.\nThere is no possible update.')
        cls.strings_list()
        existing_strings = [d for d in os.listdir('data/')
                            if os.path.isdir('data/{}'.format(d))]
        ID = input('Which thermistor string do you want to update ?\n')
        if not ID in existing_strings:
            raise NameError('{} is not a valid name for any existing'
                            ' thermistor string!'.format(ID))
        ## Make editable the files of the thermistor string
        ft.unprotect(ID)

        with open('data/{0}/{0}_object'.format(ID),'rb') as file_object:
            my_unpickler = pickle.Unpickler(file_object)
            content = my_unpickler.load()

        count = 1
        list_sensor = list()
        while count <= content._num:
            if count == 1:
                print('Sensor 1 (uppermost sensor)')
            elif count == content._num:
                print('Sensor {} (lowermost sensor)'.format(content._num))
            else:
                print('Sensor {}'.format(count))
            list_sensor.append(count)
            count += 1
        sensor = input('Which sensor do you to update ? (number)\n')
        try:
            sensor = int(sensor)
        except ValueError:
            print('The sensor number is not an integer!')
        if not sensor in list_sensor:
            raise NameError('There is no sensor {}!'.format(sensor))
        sensor_depth = input('The sensor {0} had a depth of {1} m.\n'
                             'What depth do you want to give to the'
                             ' sensor {0} now?\nNote : A negative '
                             'value indicates how far out of the ice '
                             'the sensor is.\n'.format(sensor,
                                                        content._depth[sensor]))

        if re.match(r'\d+,\d+',sensor_depth):
            raise ValueError('The value entered must be an integer '
                             'or a floating number!\nFloatting '
                             'numbers must be written with a dot for '
                             'the decimal separator.')
        try:
            sensor_depth = float(sensor_depth)
        except ValueError:
            print('The value entered must be an integer or a '
                  'floatting number!')

        content._depth = ft.get_depth(content._dist,sensor,sensor_depth)

        ## Update the date of the field measurements
        ## Find the date for the last field measurements
        last_update = ft.last_update(ID)

        year = input('Last measurements date back to: {}\n'
                     'What is the date matching to the update?\n'
                     'Year : '.format(last_update))
        try:
            year = int(year)
        except ValueError:
            print('The year must be an integer value!')
        if not re.match(r'\d{4}',str(year)):
            raise ValueError('The year is not valid (4 digits)!\n'
                             'Example of valid year : 2014')
        month = input('Month (1 - 12): ')
        try:
            month = int(month)
        except ValueError:
            print('The month must be an integer value!')
        if month < 1 or month > 12:
            raise ValueError('The month must be a value between 1 and'
                             ' 12 included.')
        dom = input('Day of month (1 - 31) : ')
        try:
            dom = int(dom)
        except ValueError:
            print('The day of month must be an integer value (1-31)!')
        if dom < 1 or dom > 31:
            raise ValueError('The day of month must be a value between'
                             ' 1 and 31 included.')
        HM = str()
        while HM.lower() != 'y' and HM.lower() != 'n':
            HM = input('Do you also want to update the time (hours and'
                       ' minutes)? (y/n)\n')
        HM = HM.lower()
        if HM == 'y':
            hours = input('Hours (0-23) : ')
            try:
                hours = int(hours)
            except ValueError:
                print('The number of hours must be an integer value!')
            if hours < 0 or hours > 23:
                raise ValueError('The number of hours must be a value'
                                 ' between 0 and 23 included.')
            minutes = input('Minutes (0-59) : ')
            try:
                minutes = int(minutes)
            except ValueError:
                print('The number of minutes must be an integer value!')
            if minutes < 0 or minutes > 59:
                raise ValueError('The number of minutes must be a '
                                 'value between 0 and 59 included.')
            content._date = datetime.datetime(year,month,dom,
                                              hours,minutes)
            print('The depth of the sensors has been updated!')
        else:
            content._date = datetime.datetime(year,month,dom)
            print('The depth of the sensors has been updated!')
        ## Write in both the file.txt and the file_object

        ## file.txt
        replacement = False
        pattern = str(content._date)
        matched = re.compile(pattern).search
        with fileinput.input('data/{0}/{0}.txt'.format(ID),inplace=1) as file_txt:
            for line in file_txt:
                if not matched(line):
                    print(line, end='')
                elif matched(line):
                    content._Ts = ft.temp_at_T(line)
                    count = 0
                    line = '{}'.format(pattern)
                    while count < content._num:
                        index = count + 1
                        line += '\t{},{}'.format(content._depth[index],
                                                 content._Ts[index])
                        count += 1
                    print(line)
                    replacement = True

        if not replacement:
            with open('data/{0}/{0}.txt'.format(ID),'a') as file_txt:
                line = '{}'.format(pattern)
                count = 1
                while count <= content._num:
                    line += '\t{},{}'.format(content._depth[count],
                                             content._Ts[count])
                    count += 1
                line += '\n'
                file_txt.write(line)

        ## Sorts field measurements in the text file
        ft.sort_measurements(ID)

        ## file_object
        with open('data/{0}/{0}_object'.format(ID),'wb') as file_object:
            content._Ts = ft.last_temp(ID)
            content._depth = ft.last_depth(ID)
            my_pickler = pickle.Pickler(file_object)
            my_pickler.dump(content)


        ## Protect files and directory of the thermistor string from
        ## editing
        ft.protect(ID)


    @classmethod
    def update_temp(cls):
        """This method enables to update the temperature measured by
        the sensors in the ice. The temperature must be given in
        degrees Celsius (floatting or integer value). The sensor 1 is
        the upppermost sensor (closest to the surface or the furthest
        out of the ice).
        """
        ## Update the temperature
        if not os.path.exists('data'):
            raise NameError('No thermistor string has been created '
                            'yet.\nThere is no possible update.')
        cls.strings_list()
        existing_strings = [d for d in os.listdir('data/')
                            if os.path.isdir('data/{}'.format(d))]
        ID = input('Which thermistor string do you want to update ?\n')
        if not ID in existing_strings:
            raise NameError('{} is not a valid name for any existing'
                            ' thermistor string!'.format(ID))
        ## Make editable the files of the thermistor string
        ft.unprotect(ID)

        with open('data/{0}/{0}_object'.format(ID),'rb') as file_object:
            my_unpickler = pickle.Unpickler(file_object)
            content = my_unpickler.load()

        count = 1
        list_sensor = list()
        while count <= content._num:
            if count == 1:
                print('Sensor 1 (uppermost sensor)')
            elif count == content._num:
                print('Sensor {} (lowermost sensor)'.format(content._num))
            else:
                print('Sensor {}'.format(count))
            list_sensor.append(count)
            count += 1

        sensor = int()
        first_sensor = True
        temperatures = dict()
        while str(sensor).lower() != 'q':
            sensor = input("Which sensor do you want to update ? "
                           "(number)\nType 'q' to exit.\n")
            sensor = sensor.lower()
            if sensor == 'q':
                if first_sensor:
                    raise KeyboardInterrupt('No update was performed'
                                            ' for the temperature '
                                            'string {}.'.format(ID))
                else:
                    print('The temperature values measured by the '
                          'sensors have been updated!')
                    break
            try:
                sensor = int(sensor)
            except ValueError:
                print('The sensor number is not an integer, the update'
                      ' was cancelled!')
            if not sensor in list_sensor:
                raise NameError('There is no sensor {}, the update'
                                ' was cancelled!'.format(sensor))
            sensor_res = input('The sensor {0} indicated a temperature'
                               ' of {1} °C.\nWhat is the new resistance'
                               ' value measured by the sensor ? (kiloOhms)\n'\
                               .format(sensor,content._Ts[sensor]))

            if re.match(r'\d+,\d+',sensor_res):
                raise ValueError('The value entered must be an integer'
                                 ' or a floatting number!\nFloatting '
                                 'numbers must be written with a dot '
                                 'for the decimal separator.\nThe '
                                 'update was cancelled!')
            try:
                sensor_res = float(sensor_res)
            except ValueError:
                print('The value entered must be an integer or a '
                'floatting number!\nThe update was cancelled!')
            ## Compute new temperature using the calibration
            calibration = ft.read_calibrationc('calibration_curve')
            new_calib = ft.sensor_calib(zero_degree[sensor], calibration)
            temperatures[sensor] = ft.res_in_temp(sensor_res, new_calib)

            ## Update the date of the field measurements
            ## Find the date for the last field measurements
            while first_sensor:
                last_update = ft.last_update(ID)

                year = input('Last measurements date back to: {}\n'
                             'What is the date matching to the update?'
                             '\nYear : '.format(last_update))
                try:
                    year = int(year)
                except ValueError:
                    print('The year must be an integer value! The '
                          'update was cancelled!')
                if not re.match(r'\d{4}',str(year)):
                    raise ValueError('The year is not valid (4 digits)!'
                                     ' Example of valid year : 2014.\n'
                                     'The update was cancelled!')
                month = input('Month (1 - 12): ')
                try:
                    month = int(month)
                except ValueError:
                    print('The month must be an integer value! The '
                          'update was cancelled!')
                if month < 1 or month > 12:
                    raise ValueError('The month must be a value between'
                                     ' 1 and 12 included.\nThe '
                                     'update was cancelled!')
                dom = input('Day of month (1 - 31) : ')
                try:
                    dom = int(dom)
                except ValueError:
                    print('The day of month must be an integer value '
                          '(1-31)! The update was cancelled!')
                if dom < 1 or dom > 31:
                    raise ValueError('The day of month must be a value'
                                     ' between 1 and 31 included.\n'
                                     'The update was cancelled!')
                HM = str()
                while HM.lower() != 'y' and HM.lower() != 'n':
                    HM = input('Do you also want to update the time '
                               '(hours and minutes)? (y/n)\n')
                HM = HM.lower()
                if HM == 'y':
                    hours = input('Hours (0-23) : ')
                    try:
                        hours = int(hours)
                    except ValueError:
                        print('The number of hours must be an integer'
                              ' value! The update was cancelled!')
                    if hours < 0 or hours > 23:
                        raise ValueError('The number of hours must be'
                                         ' a value between 0 and 23 '
                                         'included.\nThe update was'
                                         ' cancelled!')
                    minutes = input('Minutes (0-59) : ')
                    try:
                        minutes = int(minutes)
                    except ValueError:
                        print('The number of minutes must be an '
                              'integer value! The update was cancelled!')
                    if minutes < 0 or minutes > 59:
                        raise ValueError('The number of minutes must'
                                         ' be a value between 0 and 59'
                                         ' included.\nThe update was'
                                         ' cancelled!')
                    content._date = datetime.datetime(year,month,
                                                      dom,hours,minutes)
                else:
                    content._date = datetime.datetime(year,month,dom)

                continue_update = input('Do you want to update the '
                                        'temperature of other\nsensors'
                                        ' for the same date ? (y/n)\n')
                continue_update = continue_update.lower()
                while continue_update != 'n' and continue_update != 'y':
                    continue_update = input('Do you want to update the'
                                            ' temperature of another\n'
                                            'sensor for the same date'
                                            ' ? (y/n)\n')
                first_sensor = False
                if continue_update == 'y':
                    sensor = int()
                else:
                    print('The temperature measured by the sensor has'
                          ' been updated!')
                    sensor = 'q'
                    break


        ## Write in both the file.txt and the file_object

        ## file.txt
        replacement = False
        pattern = str(content._date)
        matched = re.compile(pattern).search
        with fileinput.input('data/{0}/{0}.txt'.format(ID),inplace=1) as file_txt:
            for line in file_txt:
                if not matched(line):
                    print(line,end='')
                elif matched(line):
                    content._depth = ft.depth_at_T(line)
                    content._Ts = ft.temp_at_T(line)
                    for key,value in temperatures.items():
                        content._Ts[key] = value
                    count = 0
                    line = '{}'.format(pattern)
                    while count < content._num:
                        index = count + 1
                        line += '\t{},{}'.format(content._depth[index],
                                                 content._Ts[index])
                        count += 1
                    print(line)
                    replacement = True

        if not replacement:
            content._Ts = ft.last_temp(ID)
            for key,value in temperatures.items():
                content._Ts[key] = value
            with open('data/{0}/{0}.txt'.format(ID),'a') as file_txt:
                line = '{}'.format(pattern)
                count = 1
                while count <= content._num:
                    line += '\t{},{}'.format(content._depth[count],
                                             content._Ts[count])
                    count += 1
                line += '\n'
                file_txt.write(line)

        ## Sorts field measurements in the text file
        ft.sort_measurements(ID)

        ## file_object
        with open('data/{0}/{0}_object'.format(ID),'wb') as file_object:
            content._Ts = ft.last_temp(ID)
            content._depth = ft.last_depth(ID)
            my_pickler = pickle.Pickler(file_object)
            my_pickler.dump(content)

        ## Protect files and directory of the thermistor string from
        ## editing
        ft.protect(ID)
```

plottemperature.py

Listing 2: This file enables to plot the data formatted by the program `temperature.py`.

```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-


import re
import pickle
from itertools import repeat
import datetime as dt
import os


import numpy as np
from pandas import DataFrame, Series
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import cm
from scipy.interpolate import griddata


from temperature import TempProfile
import functionstemperature as ft


## Identification of the temperature string location
ID = 'H44'
## Presence of air temperature data from HOBO logger as csv file
Hobodata = True

## If in the same folder, look for the right csv file
if Hobodata:
    pattern = re.compile(r'{}[\w-]+\.csv'.format(ID))
    folder = [f for f in os.listdir() if os.path.isfile(f)]
    for f in folder:
        if pattern.match(f):
            filename = f
    # Which sensor was in the radiation shield (represents Air Temp)
    Ta = 'T2'

## Make editable the files of the thermistor string
ft.unprotect(ID)

## Open the ice temperature data file formatted by the program
## temperature.py
with open('data/{0}/{0}_object'.format(ID), 'rb') as file_object:
    my_unpickler = pickle.Unpickler(file_object)
    content = my_unpickler.load()


Sensor = list(range(1,content._num+1))
with open('data/{0}/{0}.txt'.format(ID), 'r') as file_txt:
    lines = file_txt.readlines()[2:]
    text = ''.join(lines)

## Creates a pattern to find dates in the txt file created by the
## program temperature.py
datePattern = re.compile(r'^\d{4}-\d{2}-\d{2}\s\d{2}:\d{2}:\d{2}',
                   re.MULTILINE)
dates = datePattern.findall(text)

## Assign a date to each sensor and for each field measurement
date = [x for item in dates for x in repeat(item, content._num)]
dateFormat = '%Y-%m-%d %H:%M:%S'
date = [dt.datetime.strptime(i,dateFormat)
        for i in date]
date = np.array(date)
Date = np.array(date)

## Prepare and clean the data for the later creation of a
## pandas.DataFrame and ease the data handling
Sensor = Series(list(range(1, content._num+1)) * len(dates))

depthPattern = re.compile(r'-?\d+\.?\d{1,2}(?=,)')
depths = depthPattern.findall(text)
depths = np.array(depths, dtype=np.float64)
depths = np.where(depths<0, np.nan, depths)
Depth = Series(depths)

tempPattern = re.compile(r'(?<=,)(nan|-?\d+\.\d{1,2})')
temps = tempPattern.findall(text)
temps = np.array(temps, dtype=np.float64)
cond1 = depths > 0
cond2 = temps > 0
temps = np.where(cond1 & cond2, 0, temps)
Temp = Series(temps)
```

```python

## Creates a pandas.DataFrame with the data
data = {'Date':Date, 'Sensor':Sensor, 'Depth':Depth, 'Temp':Temp,
        'Time':Date}
frame = DataFrame(data).dropna()
frame = frame.pivot('Time','Sensor').stack('Sensor')
frame.columns = pd.Index(frame.columns, name='Parameters')

## Import the HOBO data (csv file) into another DataFrame
frame2 = pd.read_table(filename, sep=',',
                       usecols=[1,2,3,4], parse_dates=1, header=1,
                       names=['Date','T1','T2','V']).dropna()

## Filtered data
# array of dates at the datetime format
DDTF = list()
for i in frame.Date.values:
    DDTF.append(dt.datetime.strptime(str(pd.to_datetime(i)),
                                     '%Y-%m-%d %H:%M:%S'))
DDTF = np.array(DDTF)
# create xaxis with dates for the whole period of measurements, for
# every hour
xaxis = pd.date_range(DDTF.min(), DDTF.max(),freq='H')

# array of dates at the datetime format, for the air temperature time
# series
DTa = list()
for i in frame2.Date.values:
    DTa.append(dt.datetime.strptime(i,'%m.%d.%y %I:%M:%S %p'))
DTa = np.array(DTa)

# array of the air temperature time series
Ta = frame2['{}'.format(Ta)].values

# array of Timestamps for interpolations with griddata function
TS = list()
for i in DDTF:
    TS.append(i.timestamp())
TS = np.array(TS)


## Plots the figure of the ice temperature and air temperature for the
## same period, with the ice temperature profiles interpolated over
## time
numrows = 300
numcolors = 15
cmap = plt.cm.get_cmap(name='jet',lut=numcolors)
xi = np.linspace(TS.min(), TS.max(), len(xaxis))
yi = np.linspace(frame.Depth.min(), frame.Depth.max(), numrows,
                 endpoint=True)
x, y, z = TS, frame.Depth.values, frame.Temp.values
zi = griddata((x, y), z, (xi[None,:], yi[:,None]), method='cubic')
fig = plt.figure(dpi=150)
ax2 = fig.add_subplot(2, 1, 2)
im = ax2.contourf(xaxis, yi, zi, numcolors, cmap=cmap, extend='both')
cs = ax2.contour(xaxis, yi, zi, numcolors, linewidths=.5, colors='k')
ax2.scatter(DDTF, y, 20, z, cmap=cmap)
ax2.set_xlabel('Time')
ax2.set_ylabel('Depth (m)')
period = ax2.get_xlim()
depth_lim = ax2.get_ylim()
ax2.set_ylim([0,depth_lim[1]])
ax2.invert_yaxis()
cbar = plt.colorbar(im,orientation='horizontal',ax=ax2, pad=0.25,
                    drawedges=True,shrink=0.8, extendfrac='auto')
cbar.set_label('Ice temperature (°C)')
ax1 = fig.add_subplot(2,1,1)
cond1 = DTa < (DDTF.min()-dt.timedelta(5))
cond2 = DTa > (DDTF.max()+dt.timedelta(5))
mask = np.where(cond1 & cond2, False, True)
DTa = DTa[mask]
Ta = Ta[mask]
ax1.plot(DTa,Ta,'r-',label='Air temperature (°C)')
ax1.axhline(color='k',linewidth=.5,label='_nolegend_')
ax1.set_xlim(period)
ax1.legend(loc='best')
ax1.set_xlabel('Time')
ax1.set_ylabel('Temperature (°C)')
fig.suptitle('Air and ice temperature at stake {}'.format(ID),
             fontsize=14)
fig.tight_layout()
plt.show()

## Plot only the ice temperature profiles (not interpolated), with one
## curve for each field measurement
fig2 = plt.figure(dpi=150)
number = len(dates)
cmap = plt.get_cmap('gist_rainbow')
colors = [cmap(i) for i in np.linspace(0, 1, number)]
datesLegend = dates.copy()
for indx, date in enumerate(datesLegend):
    datesLegend[indx] = dt.datetime.strptime(date, dateFormat)\
      .strftime('%d. %B')

for idx in np.arange(number):
    if idx == 0:
        plt.plot(frame.ix[dates[idx]].Temp, frame.ix[dates[idx]].Depth,
             color=colors[idx],
             label='{} (set-up)'.format(datesLegend[idx]))
    else:
        plt.plot(frame.ix[dates[idx]].Temp, frame.ix[dates[idx]].Depth,
                 color=colors[idx],
                 label='{}'.format(datesLegend[idx]))
ax = fig2.gca()
ax.legend(loc='best')
ax.invert_yaxis()
ax.set_xlabel('Temperature (°C)')
ax.set_ylabel('Depth (m)')
fig2.suptitle('Ice temperature at stake {} in 2014'.format(ID),
             fontsize=14)
plt.show()

## Plot the ice temperature profiles with one curve for each field
## measurements, with the air temperature time series above
fig3 = plt.figure(dpi=150)
ax2 = fig3.add_subplot(2, 1, 2)
for idx in np.arange(number):
    if idx == 0:
        plt.plot(frame.ix[dates[idx]].Temp, frame.ix[dates[idx]].Depth,
             color=colors[idx],
             label='{} (set-up)'.format(datesLegend[idx]))
    else:
        plt.plot(frame.ix[dates[idx]].Temp, frame.ix[dates[idx]].Depth,
                 color=colors[idx],
                 label='{}'.format(datesLegend[idx]))
ax2.set_xlabel('Ice temperature (°C)')
ax2.set_ylabel('Depth (m)')
ax2.legend(loc='best')
depth_lim = ax2.get_ylim()
ax2.set_ylim([0,depth_lim[1]])
ax2.invert_yaxis()
ax1 = fig3.add_subplot(2,1,1)
ax1.plot(DTa,Ta,'r-',label='Air temperature (°C)')
ax1.axhline(color='k',linewidth=.5,label='_nolegend_')
ax1.set_xlim(period)
ax1.legend(loc='best')
ax1.set_xlabel('Time')
ax1.set_ylabel('Air temperature (°C)')
fig3.suptitle('Air and ice temperature at stake {} in 2014'.format(ID),
              fontsize=14)
fig3.tight_layout()
plt.show()

## Protect the files and directories created from writing by changing
## permissions
ft.protect(ID)
```

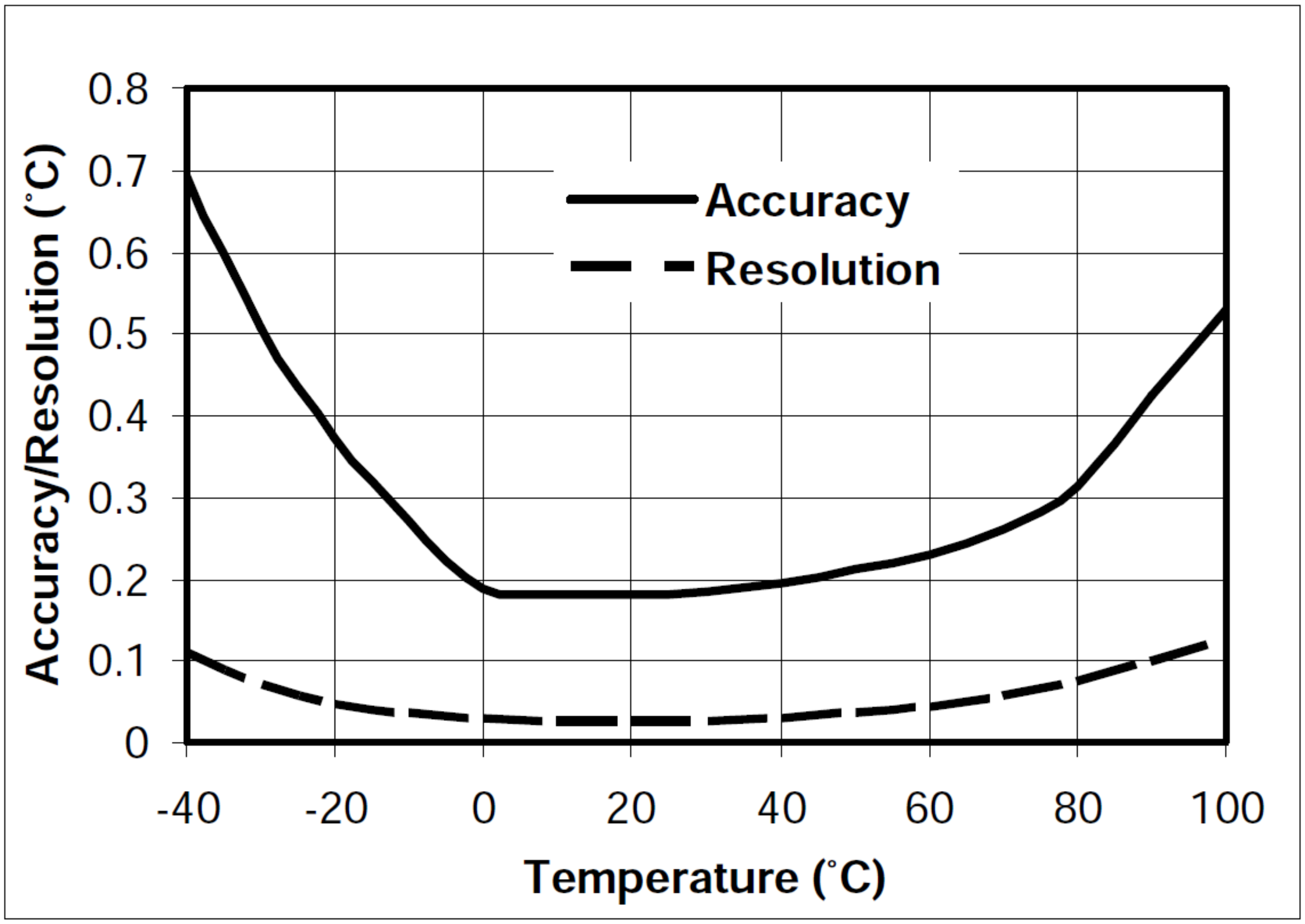
functionstemperature.py

Listing 3: This file contains the functions required to run the scripts `temperature.py` and `plottemperature.py`.

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import os
import csv
import re
import datetime
import time


import numpy as np


## Function that computes the depth of the sensors.
def get_depth(dist, sensor, sensor_depth):
    """ This function sets the depth of one of the sensors, and
    updates automatically the depth of the other sensors.

    Positional arguments:
    dist: dictionary that contains the positions of the sensors on the
    line.
    sensor:       the sensor that is used for the update.
    sensor_depth: the depth of the sensor.

    Note:

    A negative value for the parameter sensor_depth indicates how far
    out the sensor of interest is.
    """
    depths = dict()
    for key in dist.keys():
        if key != sensor:
            depths[key] = sensor_depth + (dist[key] - dist[sensor])
        else:
            depths[key] = sensor_depth
    for key,value in depths.items():
        depths[key] = float('{0:.2f}'.format(depths[key]))
    return depths


## Function that reads the calibration curve.
def read_calibrationc(filename, headerlines=3):
    """ This function reads the calibration curve.

    The calibration curve is used to convert the resistance values of
    the sensors into temperature values. It returns a list for which
    each item is a dictionary with a temperature value (key: 'Temp')
    matching to a resistance value in kiloOhms (key: 'Resistance').

    Positional argument:
    filename: path and filename of the calibration curve.

    Optional argument:
    headerlines: number of lines not interpreted by the function
    (default is 3).
    """
    data = list()
    headerlines += 1
    lines =  open(filename).readlines()[headerlines:]
    csvdictreader = csv.DictReader(lines, delimiter='\t')
    # Convert Ohms units into kiloOhms and rounds to first decimal
    for line in csvdictreader:
        line['Resistance'] = float(line['Resistance'])
        line['Resistance'] = float('{0:.1f}'.format(line['Resistance']/1000))
        line['Temp'] = float(line['Temp'])
        data.append(line)
    return data


## Function that corrects the calibration curve for each sensor, with
## a linear interpolation.
def sensor_calib(res, calibration_curve):
    """ This function corrects the calibration curve for temperature
    sensors.

    It returns a corrected calibration curve for temperature sensors,
    according to their resistance values at 0 degree Celsius (in kiloOhms).

    Positional arguments:
    res:               the resistance value at 0 degree Celsius.
    calibration_curve: the orignal calibration curve for this type of
    sensor.
    """
    new_calib = list(calibration_curve)
    match = [line for line in calibration_curve
             if line['Resistance'] == res]
    if len(match) == 1:
        for line in new_calib:
            line['Temp'] -= match[0]['Temp']
    else:
        average = float()
        for elt in match:
            average += elt['Temp']
        average = average/len(match)
        for line in new_calib:
            line['Temp'] -= average
    # Rounds temperature values to the second decimal.
    for line in new_calib:
        line['Temp'] = float('{0:.2f}'.format(line['Temp']))
    return new_calib


## Function that converts the resistance values into temperature
## values using the corrected calibration curves.
def res_in_temp(res, calibration_curve):
    """ This function converts resitance values into temperature
    values.

    It converts the resistance value of a sensor into a temperature
    value, using the corrected calibration curve of the sensor.

    Positional arguments:
    res:               resistance value in kiloOhms.
    calibration_curve: corrected calibration curve.
    """
    if str(res) == 'nan':
        temperature = np.nan
        return temperature
    else:
        match = [line for line in calibration_curve
                 if line['Resistance'] == res]
        while not match:
            increment = 0.1
            res1 = res + increment
            res2 = res - increment
            increment += 1
            match = [line for line in calibration_curve
                     if line['Resistance'] == res1
                     or line['Resistance'] == res2]
        if len(match) == 1:
            temperature = match[0]['Temp']
        else:
            temperature = float()
            for elt in match:
                temperature += elt['Temp']
            temperature = temperature/len(match)
            temperature = float('{0:.2f}'.format(temperature))
        return temperature


## Function that protect the files and directories created from
## writing by changing permissions.
def protect(ID):
    """ This function protects from writing the data of a thermistor
    string.

    It changes the permissions of the 'data' directory, the
    subdirectory and the files of the thermistor string.

    Positional argument:
    ID: identification of the thermistor string.
    """
    os.chmod('data/{0}/{0}.txt'.format(ID),0o444)
    os.chmod('data/{0}/{0}_object'.format(ID),0o444)
    os.chmod('data/{}'.format(ID),0o555)
    os.chmod('data',0o555)


## Function that changes the permissions on the files and directories
## of the temperature string to make them editable.
def unprotect(ID):
    """ This function makes editable the data of a thermistor string.

    It changes the permissions of the 'data' directory, the
    subdirectory and the files of the thermistor string.

    Positional argument:
    ID: identification of the thermistor string.
    """
    os.chmod('data',0o777)
    os.chmod('data/{}'.format(ID),0o777)
    os.chmod('data/{0}/{0}_object'.format(ID),0o666)
```

```python
    os.chmod('data/{0}/{0}.txt'.format(ID),0o666)


## Function that finds the date of the last field measurements.
def last_update(ID):
    """ This function returns the date of the last measurements of a
     thermistor string.

    It extracts the date which appears on the last line in the text
    file of the thermistor string.

    Positional argument:
    ID: identification of the thermistor string.
    """
    with open('data/{0}/{0}.txt'.format(ID),'r') as file_txt:
        last_line = file_txt.readlines()[-1]
        whole_date = re.findall(r'^\d{4}-\d{2}-\d{2}',last_line)
        lyear,lmonth,lday = whole_date[0].split('-')
        lyear, lmonth, lday = int(lyear), int(lmonth), int(lday)
        timestamp = datetime.datetime.timestamp(datetime.datetime(lyear,lmonth,lday))
        last_update = time.strftime('%A %d %B %Y',time.localtime(timestamp))
    return last_update


## Function that sorts the measurements in the text file.
def sort_measurements(ID,headerlines=2):
    """ This function sorts the lines in the text file of a thermistor
    string.

    It sorts the lines using the date of the field measurements. The
    most recent measurements are at the end of the file.

    Positional argument:
    ID: identification of the thermistor string.

    Optional argument:
    headerlines: number of lines not interpreted by the function
    (default is 2).
    """
    with open('data/{0}/{0}.txt'.format(ID),'r') as file_txt:
        content = file_txt.readlines()
        first_lines = content[:headerlines]
        lines = content[headerlines:]
        lines.sort()
        content = ''.join(first_lines+lines)

    with open('data/{0}/{0}.txt'.format(ID),'w') as file_txt:
        file_txt.write(content)


## Function that finds the depth values of the last field
## measurements.
def last_depth(ID):
    """ This function returns the depth of the sensors of a thermistor
     string at the time of the last field measurements.

    It extracts the depth values which appear on the last line in the
    text file of the thermistor string.

    Positional argument:
    ID: identification of the thermistor string.
    """
    pattern = re.compile(r'((\d|-|\.)+)(?=,)')
    depth = dict()
    with open('data/{0}/{0}.txt'.format(ID),'r') as file_txt:
        last_line = file_txt.readlines()[-1]
        depths = pattern.findall(last_line)
        for i,value in enumerate(depths):
            depth[i+1] = value[0]
    return depth


## Function that finds the temperature values of the last field
## measurements.
def last_temp(ID):
    """ This function returns the temperature values recorded by the
     sensors of a thermistor string at the time of the last field
     measurements.

    It extracts the temperature values which appear on the last line
    in the text file of the thermistor string.

    Positional argument:
    ID: identification of the thermistor string.
    """
    pattern = re.compile(r'(?<=,)((\d|-|\.)+|nan)')
    temp = dict()
    with open('data/{0}/{0}.txt'.format(ID),'r') as file_txt:
        last_line = file_txt.readlines()[-1]
        temperatures = pattern.findall(last_line)
        for i,value in enumerate(temperatures):
            temp[i+1] = value[0]
    return temp


## Function that returns the temperature values from field
## measurements at a given date.
def temp_at_T(line):
    """ This function returns the temperature values recorded by the
    sensors of a thermistor string, for a given date.

    It extracts the temperature values from the text file of the
    thermistor string.

    Positional arguments:
    line: line that matches to the date of the field measurements.
    """
    pattern = re.compile(r'(?<=,)((\d|-|\.)+|nan)')
    temp = dict()
    temperatures = pattern.findall(line)
    for i,value in enumerate(temperatures):
        temp[i+1] = value[0]
    return temp


## Function that returns the depth values from field measurements at a
## given date.
def depth_at_T(line):
    """ This function returns the depths of the sensors of a
    thermistor string, for a given date.

    It extracts the depth values from the text file of the thermistor
    string.

    Positional arguments:
    line: line that matches to the date of the field measurements.
    """
    pattern = re.compile(r'((\d|-|\.)+)(?=,)')
    depth = dict()
    depths = pattern.findall(line)
    for i,value in enumerate(depths):
        depth[i+1] = value[0]
    return depth
```

C.3 HOBO Pro V2 Accuracy and resolution

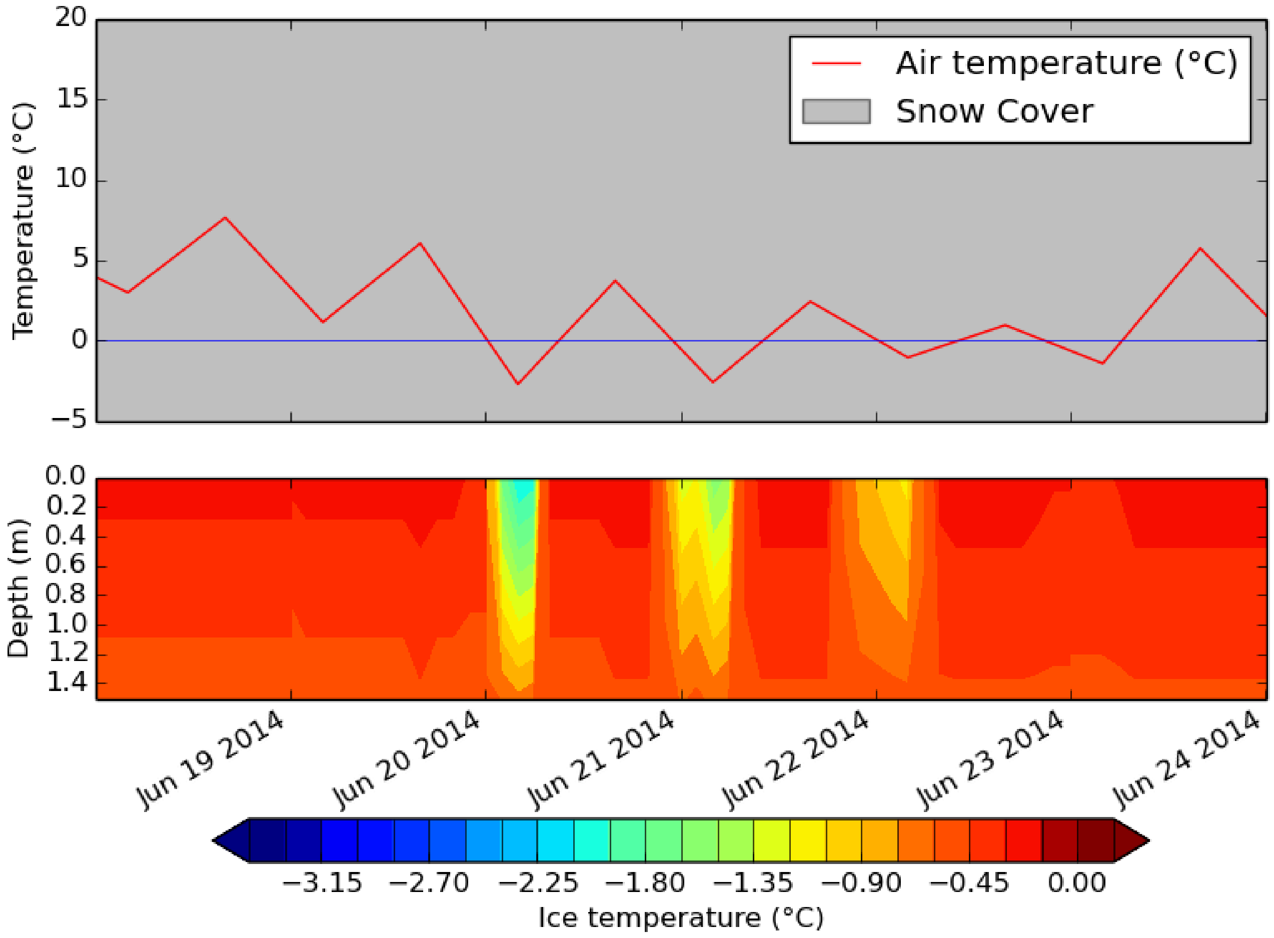


Accuracy and resolution of HOBO Pro V2 external temperature data logger (from Onset Computer Corporation, 2014).

Appendix D

Ice and air temperature at Storbreen

D.1 Penetration of the diurnal signal



Penetration of the air temperature diurnal signal in the subsurface of the glacier. The depth of the sensor was here updated using the corrected DDF. Note that the ice is then assumed to be snow covered by the model, and that the diurnal signal penetrates down to a depth of ∼1.4 m.

D.2 Python code

icetemperatureprofile.py

Listing 4: This program reads the temperature data obtained from the GeoPrecision M-Log5W data logger. It enables also to update the temperature profile with field observations, such as the depth of the sensors, the presence of a snowpack and its thickness if there is any.

```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-


import re
import pickle
import datetime as dt
import os
import fileinput
import argparse

import numpy as np
from pandas import DataFrame, Series
import pandas as pd

from functions import *


## Identification string of the thermistor line
ID = 'S2'

## Number of sensors on the string
num = 10

## Defines the position of the sensors on the temperature string
## The position of the sensors are relative to the uppermost one
## Assign 'True' to equidistant if the sensors are equidistant,
## else assign 'False'
equidistant = False
## Value in meters that separates neighbouring sensors if equidistant
loc = 2

if equidistant:
    dist = dict()
    key = 1
    pos = 0
    count = 0
    while count < num:
        dist[key] = pos
        key += 1
        pos += loc
        count += 1

## Dictionary containing positions of the sensors if not equidistant
## Key 1 is for the uppermost (closest to surface) sensor, key 2 the
## one below and so on... The position is relative to the sensor 1. If
## the sensor 2 is 3.5 meters away from sensor 1 on the line, the
## value 3.5 should be assigned to key 2 (dist[2])
dist = {1:0, 2:3, 3:5, 4:6, 5:7, 6:8, 7:9, 8:10, 9:11, 10:12}

## Update the depth of all sensors Choose one sensor that you want to
## update (its number on the line), and indicate its depth in meters
## Note : A negative value can be used to tell how far out of the
## borehole the sensor is
sensor = 1
sensor_depth = -1.33

depth = get_depth(dist, sensor, sensor_depth)

## Path to the data file exported from a geoprecision data logger
path = ('/home/mtac/Documents/Oslo/Years_2013-2015/'
        'MSc/MSc_Thesis/temperature/S2_17-09-14_12-20_txt.txt')
## Check that the path exists
if not os.path.exists(path):
    raise FileNotFoundError('The path to the data file does not exists!')

## Formatting the data file into a pandas.DataFrame
headerlines = 1

## Number of columns in the data file.
numcols = len(pd.read_table(path, sep=',', header=headerlines,
                            nrows=1).columns)

## Creates the name of the DataFrame columns
```

```python
names = ['T{}'.format(i) for i in range(1,num+1)]
names.insert(0,'Date')
names.append('Ta')

## Columns to use in the data file (only the temperate values and the
## date )
cols = list(range(1,num+2))
cols.append(numcols-1)

WholeFrame = pd.read_table(path, sep=',', usecols=cols, names=names,
                      header=1, na_values=['(NO SENSORS)',
                                           '(ERROR 107)']).dropna(how='all')

## Convert the string in the first column into a datetime array
dateformat = '%d.%m.%Y %H:%M:%S'
convert2datetime = lambda x: dt.datetime.strptime(x, dateformat)
WholeFrame.Date = WholeFrame['Date'].apply(convert2datetime)

## Convert temperature values to float
for i in np.arange(1, len(WholeFrame.columns)):
    WholeFrame[WholeFrame.icol(i).name] = WholeFrame.icol(i).astype(np.float64)

## First and last dates valid for calibration
first_date_calib = dt.datetime(2014, 1, 10)
last_date_calib = dt.datetime(2014, 4, 1)
cond1 = WholeFrame['Date'] > first_date_calib
cond2 = WholeFrame['Date'] < last_date_calib
CalibFrame = WholeFrame[cond1 & cond2].dropna()
CalibFrame['Offset'] = CalibFrame.ix[:,1:num+1].mean(axis=1)-CalibFrame['Ta']
OffsetTa = CalibFrame['Offset'].mean()

## Correct the air temperature offset
applyoffset = lambda x: x + OffsetTa
WholeFrame.Ta = WholeFrame['Ta'].apply(applyoffset)

## Start of the period of interest
start_year = 2014
start_month = 5
start_day = 21
## End of the period of interest
end_year = 2014
end_month = 9
end_day = 19

## Select only the peroid of interest
cond1 = WholeFrame.Date >= dt.datetime(start_year, start_month, start_day)
cond2 = WholeFrame.Date <= dt.datetime(end_year, end_month, end_day)
frame = WholeFrame[cond1 & cond2]

## Change the row numbers of the frame
frame.index = np.arange(1,len(frame)+1)


## Main class
class Profile:
    """ This class creates a temperature profile evolving with time.

    It represents the temperature variations with time in ice snow. It
    requires data obtained from a geoprecision datalogger. A
    temperature profile is define by its ID (ID), the number of
    sensors on the thermistor line (num), the distance of the sensors
    (dist) relative to the uppermost sensor (closest to surface), and
    their depth (depth) at the set up of the line. To create a new
    instance of this class, one must pass a pandas.DataFrame (frame)
    that contains : the date of the measurements in a first column,
    the temperature values recorded in separate columns for the
    different sensors, and in a last column, the air temperature if
    available for the same period. If the air temperature is
    available, the parameter 'Ta' must be True (default), else
    False. If the temperature profile is performed in the snowpack, or
    that the temperature variations are measured in ice which is snow
    covered at the date of setup of the line pass True to 'snow', else
    False (default). If 'snow' is True, the keyword argument thickness
    is the thickness of the snowpack in meters.
    """

    def __init__(self, ID, num, dist, depth, frame, Ta=True,
                 snow=False, thickness=0):
        """ Create the attributes of a new instance of the class Profile.


        Positional Arguments:

        ID    -- identification of the thermistor string
        num   -- num of sensor on the line
        dist  -- distance of the sensors relative to the uppermost one
        (dict with sensors numbers as keys)
        depth -- depths of each sensor (dict with sensors numbers as
        keys)
        frame -- pandas.DataFrame containing the data (see also class
        Profile)
        Keyword Arguments:

        Ta        -- existing air temperature time series (default:
        True)
        snow      -- existing snowpack at last field observations
        (default: False)
        thickness -- thickness in meters of the snowpack
        """
        ## Creates data folder in current directory if it does not exists
        if not os.path.exists('data'):
            os.makedirs('data')

        ## Check that there is no thermistor string with the same ID
        if not os.path.exists('data/{}'.format(ID)):
            if not ID.isalnum() or len(ID) < 2:
                raise TypeError('ID not valid! It should be '
                'alphanumeric and at least two caracters.')
            else:
                self._ID = ID
        else:
            raise TypeError('A thermistor string has already this'
            ' ID, choose another ID.')

        ## Check that the number of sensors is superior to 0 and is an integer value
        try:
            num = int(num)
        except ValueError:
            raise ValueError('The value entered must be an integer!')
        else:
            if num < 1 :
                raise TypeError('There must be at least one sensor!')
            else:
                self._num = num

        ## Check that the attribute 'dist' has the right format
        if not isinstance(dist,dict):
            raise TypeError('The attribute dist must be a '
                             'dictionary!')
        for value in dist.values():
            try:
                value = float(value)
            except ValueError:
                raise ValueError('The distances must either be '
                                 'integers of floatting numbers!')
        count = 1
        for key in dist.keys():
            if key != count:
                raise TypeError('The keys of the dictionary '
                '"dist" must be integers,\nstarting from 1 (uppermost'
                ' sensor), and incremented by 1 every next key.')
            count += 1
        self._dist = dist

        ## Check that the attribute 'depth' has the right format
        if not isinstance(depth,dict):
            raise AttributeError('The attribute depth must be a '
                                 'dictionary!')
        for value in depth.values():
            try:
                value = float(value)
            except ValueError:
                raise ValueError('The depth values must either be '
                                 'integers of floatting numbers!')
        count = 1
        for key in depth.keys():
            if key != count:
                raise AttributeError('The keys of the dictionary '
                '"depth" must be integers,\nstarting from 1 (uppermost'
                ' sensor), and incremented by 1 every next key.')
            count += 1
        self._depth = depth

        ## Check that Ta is a boolean
        if not isinstance(Ta,bool):
            raise TypeError('The attribute Ta must be an instance'
                           ' of the bool class!')
        self._Ta = Ta

        ## Check that the frame has the right format
        if not isinstance(frame,DataFrame):
            raise TypeError('The attribute frame must be an '
                              'instance of the class '
                              'pandas.DataFrame!')
        if not Ta:
            if not len(frame.columns) == num + 1:
                raise TypeError('The frame must contains only dates '
                                'and ice/snow temperature values!')
        else:
            if not len(frame.columns) == num + 2:
                raise TypeError('The frame must contains only dates '
                                'and temperature values!')
        try:
            frame.icol(np.arange(1,len(frame.columns))
                          ).values.astype(np.float64)
            frame.icol(0).astype('datetime64[ns]')
        except TypeError:
            raise TypeError("""The Dates of the measurements must be
        of the dtype datetime64[ns], and the
        temperature values must be intergers
        or float numbers!""")
        self._frame = frame

        ## Registers the date of the start and of the time series, and
        ## the date matching the latest updates.
        self._start = self._frame.icol(0).irow(0).to_datetime()
        self._end = self._frame.icol(0).irow(-1).to_datetime()
        self._last_update = self._start

        ## Check the format and set the snowpack attributes
        if not isinstance(snow,bool):
            raise TypeError('The snow attribute must be a boolean!')
        self._snow = snow
        try:
            float(thickness)
        except ValueError:
            raise ValueError('The thickness attribute must be an '
                             'integer or a floatting point number!')
        self._thickness = thickness

        ## Creates a specific folder for the data of the temperature string
        if not os.path.exists('data/{}'.format(ID)):
            os.chmod('data',0o777)
            os.makedirs('data/{}'.format(ID))

        ## Save the object in a file
        with open('data/{0}/{0}_object'.format(ID),'wb') as file_object:
            my_pickler = pickle.Pickler(file_object)
            my_pickler.dump(self)

        ## Write the formatted frame in a csv file
        self._frame.to_csv('data/{0}/{0}.csv'.format(ID), sep=',')

        ## Write updates in a text file
        self._headers = ('Profile: {} (depth (D*) and thickness in meters)'
                         '\nDate'.format(ID))
        count = 0
        while count < self._num:
            self._headers += ',D{}'.format(count+1)
            count += 1
        self._headers += ',Snow,Thickness'
        first_row = '\n{}'.format(self._start)
        for i in self._depth:
            first_row += ',{}'.format(self._depth[i])

        with open('data/{0}/updates.txt'.format(ID),'w') as file_txt:
            file_txt.write(self._headers)
            file_txt.write(first_row)
            file_txt.write(',{0},{1}\n'.format(snow,thickness))
        ## Protect the files and directories created from writing by
        ## changing permissions
        protect(ID)

    ## Definition of properties for the attributes
    def ID():
        doc=""" Property : Identification of the thermistor string."""
        def fget(self):
            print('The identification of this thermistor string is : {}.'\
                  .format(self._ID))
            return self._ID
        def fset(self, value):
            print('The Identification of a thermistor string cannot'
                  ' be changed!')
        def fdel(self):
            print('You cannot delete the ID of a thermistor string!')
        return locals()

    ID = property(**ID())

    def num():
        doc=""" Property : Number of sensors on the thermistor string."""
        def fget(self):
            print('The number of sensors on the thermistor string {}'
                  ' is {}.'.format(self._ID,self._num))
            return self._num
        def fset(self, value):
            print('You cannot changed the number of sensors of the'
                  ' thermistor string!')
        def fdel(self):
            print('You cannot delete the number of sensors of the'
                  ' thermistor string!')
        return locals()

    num = property(**num())

    def dist():
        doc=""" Property : Distance (meters) between sensors on the line."""
        def fget(self):
            distances = dict(self._dist)
            for key,value in distances.items():
                distances[key] = str(distances[key]) + ' m'
            print("""The distance between sensors on the thermistor
            string is given in meters by the dictionary :
            {}

            Sensor 1 is the uppermost sensor on the line (closest to
            surface), the distance given to the other sensors is
            relative to sensor 1.""".format(distances))
            return self._dist
        def fset(self, value):
            print('You cannot change the distance between sensors on'
                  ' the line!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    dist = property(**dist())

    def depth():
        doc=""" Property : Depth (meters) of the sensors on the line."""
        def fget(self):
            depths = dict(self._depth)
            for key,value in depths.items():
                depths[key] = str(depths[key]) + ' m'
            print("""The depth of the sensors on the thermistor string
            is given in meters by the dictionary :
            {}

            Sensor 1 is the uppermost sensor on the line (closest to
            surface).""".format(depths))
            return self._depth
        def fset(self, value):
            print('The "depth" attribute cannot be modified by '
                  're-assignment!\nUse the class method'
                  ' update() instead.')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    depth = property(**depth())

    def frame():
        doc=""" Property : DataFrame containing the main data of the Profile."""
        def fget(self):
            print("""DataFrame containing the dates of the measurements
            and the temperature values recorded by the sensors :
            """)
            return self._frame
        def fset(self, value):
            print('You cannot change this attribute by assignment!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    frame = property(**frame())

    def Ta():
        doc=""" Property : Boolean value. True if the air temperature
        is in the DataFrame 'frame', False if not."""
        def fget(self):
            if self._Ta:
                print('The DataFrame contains the air temperature time'
                      'series.')
            else:
                print('The DataFrame does not contain the air '
                      'temperature time series.')
            return self._Ta
        def fset(self, value):
            print('You cannot change this attribute by assignment!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    Ta = property(**Ta())

    def start():
        doc=""" Property : Date of the start of the time series."""
        def fget(self):
            print('The time series obtained from the thermistor line'
                  ' start in:\n{}'\
                  .format(self._start.strftime('%Y-%m-%d %H:%M:%S')))
            return self._start
        def fset(self, value):
            print('This attribute cannot be changed by assignment!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    start = property(**start())

    def end():
        doc=""" Property : Date of the end of the time series."""
        def fget(self):
            print('The time series obtained from the thermistor line'
                  ' end in:\n{}'\
                  .format(self._end.strftime('%Y-%m-%d %H:%M:%S')))
            return self._end
        def fset(self, value):
            print('This attribute cannot be changed by assignment!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    end = property(**end())

    def date_last_update():
        doc=""" Property : Date matching to the updates."""
        def fget(self):
            print('The date matching to the latest update is:\n{}'\
                  .format(self._last_update.strftime('%Y-%m-%d %H:%M:%S')))
            return self._last_update
        def fset(self, value):
            print('You cannot change this attribute by assignment!')
        def fdel(self):
            print('You cannot delete this attribbute!')
        return locals()

    date_last_update = property(**date_last_update())

    def snow():
        doc=""" Property : Bool value that tells if there is a snowpack."""
        def fget(self):
            if self._snow:
                print('There was snow at the last field measurements.')
            else:
                print('There was no snow at the last field measurements.')
            return self._snow
        def fset(self, value):
            print('You cannot change this attribute by assignment!')
        def fdel(self):
            print('You cannot delete this attrribute!')
        return locals()

    snow = property(**snow())

    def thickness():
        doc=""" Property : thickness (m) of the snowpack if any."""
        def fget(self):
            if not self._snow:
                print('There was no snowpack at the last field '
                      'measurements.x')
            else:
                print('The thickness of the snowpack during the last '
                      'field measurements was {} m'\
                      .format(self._thickness))
            return self._thickness
        def fset(self, value):
            print('You cannot change this attribute by assignment!')
        def fdel(self):
            print('You cannot delete this attribute!')
        return locals()

    thickness = property(**thickness())

    def __repr__(self):
        """ Function called when entering the class object directly in
        the interpreter.

        It is meant to ease the debug. It lists the most important
        attributes of the object.
        """
        line = str()
        for i in range(1,60):
            line += '-'
        with open('data/{0}/{0}_object'.format(self._ID),
                  'rb') as file_object:
            my_unpickler = pickle.Unpickler(file_object)
            content = my_unpickler.load()
        self._depth = content._depth
        self._snow = content._snow
        self._last_update = content._last_update
        depths = str()
        for i in range(1,self._num+1):
            depths += 'Sensor {0}: {1} m\n'.format(i, self._depth[i])
        return ('Temperature string, instance of the class "Profile"'
                '\n{0}\n\n'
                'ID:\n{1}\n\n'
                'Start of the time series:\n{2}\n\n'
                'End of the time series:\n{3}\n\n'
                'Number of sensors:\n{4}\n\n'
                'Latest update:\n{5}\n\n'
                'Depths of the sensors at the lastest field '
                'observations:\n{6}\n'
                'Existing snowpack at the latest field observations:\n'
                '{7}\n\n'
                'Existing air temperature time series:\n{8}\n'\
                .format(line, self._ID, self._start, self._end,
                        self._num, self._last_update, depths,
                        self._snow, self._Ta))

    def delete(self):
        """ Function called to delete the data of the object.

        This funtions deletes the object and all data files and
        directories related to the object.
        """
        condition = str()
        while condition.lower() != 'y' and condition.lower() != 'n':
            condition = input('Are you sure to delete all the data '
                              'files related to this object ? (y/n)\n')
        if condition.lower() == 'y':
            unprotect(self._ID)
            os.remove('data/{0}/{0}.csv'.format(self._ID))
            os.remove('data/{0}/updates.txt'.format(self._ID))
            os.remove('data/{0}/{0}_object'.format(self._ID))
            os.removedirs('data/{}'.format(self._ID))
            if os.path.exists('data'):
                os.chmod('data',0o555)

    @classmethod
    def strings_list(cls):
        """ This method lists the existing thermistor strings.
        """
        if not os.path.exists('data'):
            print('No thermistor string has been created yet.')
        else:
            existing_strings = [d for d in os.listdir('data/')
                                if os.path.isdir('data/{}'.format(d))]
            existing_strings.sort()
            if len(existing_strings) > 0:
                print('There is/are {} existing thermistor string(s) :'\
                      .format(len(existing_strings)))
                for string in existing_strings:
                    print(string)
            else:
                print('No thermistor string has been created yet.')

    @classmethod
    def update(cls, ID=None):
        """Updates the field observations (depth of sensors, snowpack...).

        This method enables to update the depth of the sensors in the
        ice. The depth must be given in meters (floatting point number
        or integer value). The sensor 1 is the upppermost sensor
        (closest to the surface or the furthest out of the ice). It
        also updates the field observations required for the use of
        the temperature.plot module.


        Keyword Argument:

        ID: identification of the thermistor string to update.
        """
        ## Update the depth
        if not os.path.exists('data'):
            raise NameError('No thermistor string has been created'
                            ' yet.\nThere is no possible update.')
        existing_strings = [d for d in os.listdir('data/')
                            if os.path.isdir('data/{}'.format(d))]
        if not ID:
            cls.strings_list()
            ID = input('Which thermistor string do you want to update ?\n')
        else:
            ID = str(ID)
        if not ID in existing_strings:
            raise NameError('{} is not a valid name for any existing'
                            ' thermistor string!'.format(ID))
        ## Make editable the files of the thermistor string
        unprotect(ID)

        with open('data/{0}/{0}_object'.format(ID),'rb') as file_object:
            my_unpickler = pickle.Unpickler(file_object)
            content = my_unpickler.load()

        count = 1
        list_sensor = list()
        while count <= content._num:
            if count == 1:
                print('Sensor 1 (uppermost sensor)')
            elif count == content._num:
                print('Sensor {} (lowermost sensor)'.format(content._num))
            else:
                print('Sensor {}'.format(count))
            list_sensor.append(count)
            count += 1
        sensor = input('Which sensor do you to update ? (number)\n')
        try:
            sensor = int(sensor)
        except ValueError:
            raise ValueError('The sensor number is not an integer!')
        if not sensor in list_sensor:
            raise NameError('There is no sensor {}!'.format(sensor))
        sensor_depth = input('The sensor {0} had lastly a depth of {1}'
                             ' m.'
                             '\nWhat depth do you want to give to the'
                             ' sensor {0} now?\nNote : A negative '
                             'value indicates how far out of the ice '
                             'the sensor is.\n'.format(sensor,
                                                      content._depth[sensor]))

        if re.match(r'\d+,\d+',sensor_depth):
            raise ValueError('The value entered must be an integer '
                             'or a floatting number!\nFloatting '
                             'numbers must be written with a dot for '
                             'the decimal separator.')
        try:
            sensor_depth = float(sensor_depth)
        except ValueError:
            raise ValueError('The value entered must be an integer or a '
                  'floatting number!')

        content._depth = get_depth(content._dist,sensor,sensor_depth)

        ## Update the date of the field measurements
        year = input('Last measurements date back to: {}\n'
                     'What is the date matching to the update?\n'
                     'Year : '.format(content._last_update))
        try:
            year = int(year)
        except ValueError:
            raise ValueError('The year must be an integer value!')
        if not re.match(r'\d{4}',str(year)):
            raise ValueError('The year is not valid (4 digits)!\n'
                             'Example of valid year : 2014')
        month = input('Month (1 - 12): ')
        try:
            month = int(month)
        except ValueError:
            raise ValueError('The month must be an integer value!')
        if month < 1 or month > 12:
            raise ValueError('The month must be a value between 1 and'
                             ' 12 included.')
        dom = input('Day of month (1 - 31) : ')
        try:
            dom = int(dom)
        except ValueError:
            raise ValueError('The day of month must be an integer value (1-31)!')
        if dom < 1 or dom > 31:
            raise ValueError('The day of month must be a value between'
                             ' 1 and 31 included.')
        HM = str()
        while HM.lower() != 'y' and HM.lower() != 'n':
            HM = input('Do you also want to update the time (hours and'
                       ' minutes)? (y/n)\n')
        HM = HM.lower()
        if HM == 'y':
            hours = input('Hours (0-23) : ')
            try:
                hours = int(hours)
            except ValueError:
                raise ValueError('The number of hours must be an integer value!')
            if hours < 0 or hours > 23:
                raise ValueError('The number of hours must be a value'
                                 ' between 0 and 23 included.')
            minutes = input('Minutes (0-59) : ')
            try:
                minutes = int(minutes)
            except ValueError:
                raise ValueError('The number of minutes must be an integer value!')
            if minutes < 0 or minutes > 59:
                raise ValueError('The number of minutes must be a '
                                 'value between 0 and 59 included.')
            content._last_update = dt.datetime(year,month,dom,
                                               hours,minutes)
        else:
            content._last_update = dt.datetime(year,month,dom)

        ## Update field observation about the snow pack
        snow = str()
        while snow.lower() != 'y' and snow.lower() != 'n':
            snow = input('Was there a snowpack at the location of the'
                         ' thermistor line at that time ? (y/n)\n')
        if snow.lower() == 'y':
            content._snow = True
        else:
            content._snow = False
            content._thickness = 0
            print('The profile {} has been updated!'.format(ID))
        if content._snow:
            thickness = str()
            while not isinstance(thickness,float):
                thickness = input('What was the thickness of the '
                                  'snowpack in meters ?\n')
                try:
                    thickness = float(thickness)
                except ValueError:
                    print('The thickness must be an integer or a '
                          'floatting point number!')
                if isinstance(thickness,float):
                    if not thickness > 0:
                        print('The thickness must be a value greater'
                              ' than 0!')
                        thickness = str(thickness)
            content._thickness = thickness
            print('The profile {} has been updated!'.format(ID))

        ## Write both in the text file and the object file

        ## Text file
        replacement = False
        pattern = str(content._last_update)
        matched = re.compile(pattern).search
        with fileinput.input('data/{0}/updates.txt'.format(ID),inplace=1) as file_txt:
            for line in file_txt:
                if not matched(line):
                    print(line, end='')
                elif matched(line):
                    count = 0
                    line = '{}'.format(pattern)
                    while count < content._num:
                        index = count + 1
                        line += ',{}'.format(content._depth[index])
                        count += 1
                    line += ',{0},{1}'.format(content._snow,
                                              content._thickness)
                    print(line)
                    replacement = True

        if not replacement:
            with open('data/{0}/updates.txt'.format(ID),'a') as file_txt:
                line = '{}'.format(pattern)
                count = 1
                while count <= content._num:
                    line += ',{}'.format(content._depth[count])
                    count += 1
                line += ',{0},{1}'.format(content._snow,
                                          content._thickness)
                line += '\n'
                file_txt.write(line)

        ## Sorts field observations in the text file
        sort_observations(ID)

        ## Write in the file object with the new attribute values.
        with open('data/{0}/{0}_object'.format(ID),'wb') as file_object:
            content._snow, content._thickness = last_snowpack(ID)
            content._last_update = last_update(ID)
            content._depth = last_depth(ID)
            my_pickler = pickle.Pickler(file_object)
            my_pickler.dump(content)

        ## Protect files and directory of the thermistor string from
        ## editing
        protect(ID)

## Function that loads instances of the class Profile created in
## previous sessions. Usefull to load Profile objects into the current
## name space.
def get_strings(*IDs):
    """ Loads former instances of the class Profile.

    This function loads instances of the class Profile created in
    previous sessions. The IDs of the thermistor lines will be the new
    reference to the objects in the current session.

    Note: it will override the variables in the current namespace with
    the same name as an ID of a former 'Profile' instance.


    Optional Arguments:

    IDs: list of identifications of thermistor strings.
    """
    existing_strings = [d for d in os.listdir('data/')
                        if os.path.isdir('data/{}'.format(d))]
    for i in IDs:
        if i not in existing_strings:
            raise FileNotFoundError('{} is not a valid ID!'.format(i))
        with open('data/{0}/{0}_object'.format(i),'rb') as file_object:
            my_unpickler = pickle.Unpickler(file_object)
            globals()[i] = my_unpickler.load()

## Function that loads all instances of the class Profile created in
## previous sessions. Usefull to load Profile objects into the current
## namespace.
def get_all():
    """ Loads all instances of the class Profile into the current session.

    This function loads all instances of the class Profile created in
    previous session. The IDs of these instances will be the
    references to the objects in the current session.

    Note: it will override the variables in the current namespace with
    the same name as an ID of a former 'Profile' instance.
    """
    existing_strings = [d for d in os.listdir('data/')
                        if os.path.isdir('data/{}'.format(d))]
    for i in existing_strings:
        with open('data/{0}/{0}_object'.format(i),'rb') as file_object:
            my_unpickler = pickle.Unpickler(file_object)
            globals()[i] = my_unpickler.load()

## Define optional arguments when running the script from the terminal
parser =  argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument('-u', '--update', action='store_true',
                    help=('update an existing thermistor string with '
                          'new field observations'))
group.add_argument('-U', '--updateString', metavar='ID', nargs=1,
                    help=('update the thermistor string matching to '
                          'the ID with new field observations'))
group.add_argument('-l', '--list', action='store_true',
                    help=('list the existing thermistor strings'))
group.add_argument('-g', '--getall', action='store_true',
                   help=('loads all instances of the class Profile '
                         'created during previous sessions into the '
                         'current session (the IDs will be the '
                         'references to the objects in the current '
                         'namespace)'))
group.add_argument('-G', '--get', metavar='IDs', nargs='+',
                    help=('loads instances of the class Profile '
                          ' created during previous sessions into the '
                          'current session (the IDs will be the '
                          'references to the objects in the current '
                          'namespace)'))

args = parser.parse_args()
if args.update:
    Profile.update()
if args.updateString:
    Profile.update(ID=args.updateString[0])
if args.list:
    Profile.strings_list()
if args.get:
    get_strings(*args.get)
if args.getall:
    get_all()
```

tempplot.py

Listing 5: This program loads the temperature data formatted by the script `icetemperatureprofile.py`. It also updates the depth of the sensors using a positive degree day model and plots the temperature data recorded by the GeoPrecision M-Log5W data logger. Finally, it plots the ice deformation in the subsurface to analyse the effect of temperature changes on the ice dynamics.

```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-


import pickle
import datetime as dt
from itertools import repeat


import pandas as pd
from pandas import DataFrame, Series
import matplotlib.pyplot as plt
import matplotlib as mpl
import numpy as np
import h5py
from scipy.integrate import cumtrapz


from icetemperatureprofile import Profile
from functions import *


## Identification string of the thermistor line
ID = 'S2'

with open('data/{0}/{0}_object'.format(ID),'rb') as file_object:
    my_unpickler = pickle.Unpickler(file_object)
    data = my_unpickler.load()

## Loads the data as DataFrame instance
DFrame = data.frame

## Reads the updates text file : field observations which tells if the
## ice was snow covered and what is the depth of the sensors with
## accuracy of field measurements.
UFrame = pd.read_table('data/{0}/updates.txt'.format(ID), sep=',', header=1)

## Convert the string in the first column into a datetime array.
dateformat = '%Y-%m-%d %H:%M:%S'
convert2datetime = lambda x: dt.datetime.strptime(x, dateformat)
UFrame.Date = UFrame['Date'].apply(convert2datetime)


TaFrame = DFrame[['Date','Ta']].dropna()

## Extracts the "control periods" from the dataset. A control period
## is a period between two consecutive field observations where the
## ice was not snow covered. The variable "control_periods" is a list
## of tuples which group the start date, the end date and the amount
## of ice melt for each "control period". The amount of melt is
## computed using the depth differences of the uppermost sensor of the
## thermistor string.
control_periods = sorted([(UFrame.irow(-i-1)['Date'],
                           UFrame.irow(-i)['Date'],
                           UFrame.irow(-i)['D1']-UFrame.irow(-i-1)['D1'])
                           for i in range(1,len(UFrame))
                           if ~(UFrame.irow(-i)['Snow'])
                           and ~(UFrame.irow(-i-1)['Snow'])])
sec2day = 24 * 60 * 60
convert2meltrate = lambda x: x[2]/((x[1]-x[0]).total_seconds()/sec2day)
CP = list(control_periods)

## Computes melt rate [m/d] for the control periods
melt_rate_perDay = [(i[0],i[1],convert2meltrate(i)) for i in CP] # [m/d]

datetime2date = lambda x: dt.datetime(x.year, x.month, x.day)

## Creates a DataFrame with temperature data only of the whole
## dataset. The mean temperature is computed for each day of the whole
## period of measurements.
PDDFrame = TaFrame.copy()
PDDFrame.Date = PDDFrame['Date'].apply(datetime2date)
mean_PD_perDay = PDDFrame['Ta'].groupby(PDDFrame['Date']).mean()

## Sets to NaN the mean daily temperature values that are negatives,
## to leave only the "positive degree day" values.
```

```
mean_PD_perDay[mean_PD_perDay <= 0] = np.nan
mean_PD_perDay = mean_PD_perDay.dropna()

## Computes the melt rate coefficients for each "control periods"
## (mean amount of melt per positive degree).
melt_rate_PDD = list()
for i,value in enumerate(control_periods):
    cond1 = mean_PD_perDay.index > control_periods[i][0]
    cond2 = mean_PD_perDay.index <= control_periods[i][1]
    tot_PD = mean_PD_perDay[cond1 & cond2].sum()
    melt_rate_PDD.append((control_periods[i][0], control_periods[i][1],
                          control_periods[i][2]/tot_PD))
## Total number of days for all the "control periods".
num_tot_days = np.sum([i[1] - i[0] for i in melt_rate_PDD])
## Computes a mean melt rate (amount of melt in meters, per positive
## degree), considering all "control periods".
mean_melt_rate_PDD = np.sum([((i[1] - i[0]) / num_tot_days) * i[2]
                             for i in melt_rate_PDD])
mean_PD_perDay = DataFrame(mean_PD_perDay, columns=['PDD'])
mean_PD_perDay.reset_index(inplace=True)
UFrame.Date = UFrame['Date'].apply(datetime2date)

## Creates a DataFrame (WorkFrame) with field observations and a
## positive degree day columns matching the dates of continuous
## measurements.
WorkFrame = pd.merge(UFrame, mean_PD_perDay, on='Date', how='outer')
WorkFrame = WorkFrame.sort(columns='Date')

## Sorted list of tuples which are made up with the start date, end
## date and the amount of melt for each period (Interp_periods)
## between two consecutive field observations.
Interp_periods = sorted([(UFrame.irow(-i-1)['Date'],
                          UFrame.irow(-i)['Date'],
                          UFrame.irow(-i)['D1']-UFrame.irow(-i-1)['D1'])
                          for i in range(1,len(UFrame))])

## Makes a list which elements tell the method of interpolation to be
## used, depending on if the ice was snow covered for the field
## observations at the beginning or at the end of each
## "Interp_period", and if there was an ice accumulation or ice
## ablation for the same "Interp_periods".
Interp_method = list()

for i,value in enumerate(Interp_periods):
    cond_snow_start = UFrame['Snow'][UFrame['Date']==Interp_periods[i][0]].values
    cond_snow_end = UFrame['Snow'][UFrame['Date']==Interp_periods[i][1]].values
    cond_melt = Interp_periods[i][2] < 0
    cond_acc = Interp_periods[i][2] > 0
    if cond_melt:
        if not cond_snow_start and not cond_snow_end:
            Interp_method.append('IIM')
        elif not cond_snow_start and cond_snow_end:
            Interp_method.append('ISM')
        elif cond_snow_start and not cond_snow_end:
            Interp_method.append('SIM')
        else:
            Interp_method.append('SSM')
    else:
        if not cond_snow_start and not cond_snow_end:
            Interp_method.append('IIA')
        elif not cond_snow_start and cond_snow_end:
            Interp_method.append('ISA')
        elif cond_snow_start and not cond_snow_end:
            Interp_method.append('SIA')
        else:
            Interp_method.append('SSA')

starts = [Interp_periods[i][0] for i,val in enumerate(Interp_periods)]
Interp_data = {'Date':starts, 'Interpolation':Interp_method}
InterpFrame = DataFrame(Interp_data)

## Adds to the WorkFrame a column with the interpolation method to be
## used.
WorkFrame = pd.merge(InterpFrame, WorkFrame, on='Date', how='outer')
WorkFrame = WorkFrame.sort(columns='Date')
WorkFrame.Interpolation = WorkFrame['Interpolation'].fillna(method='ffill')

temporal_res = (WorkFrame.irow(-1).Date -
                WorkFrame.irow(1).Date)/len(WorkFrame)

## For each period to interpolate (Interp_period), if it matches to a
## control_period, the melt coefficient of this control period will be
## used to estimate the ice melt over the period. If the Interp_period
## is not a control period, the mean melt rate will be used. If the
## ice melt is calculated over a control period, the melt rate is used
## from the beginning to the end of the period, weighed by the
## positive degree day values. For this dataset, the only type of
## period else than a control period matches to the interpolation type
## "SIM". This means that there was ice ablation, and that the ice was
## snow covered at the start date, but that there was no snowpack at
## the end date. For this case, the mean melt rate was applied from
```

```python
## the end of the period, weighed by the positive degree day values,
## until the total amount of melt for the period was reached. After
## that this total amount of is reached, it is assumed that no ice
## melt is happening, as the ice is snow covered and that the snow
## should melt first.
melt_rate_PDD = np.array(melt_rate_PDD)
snow_covered = list()
for val in Interp_periods:
    start = val[0]
    end = val[1]
    WorkFrame = WorkFrame.sort(columns='Date',ascending=True)
    WorkFrame.index = np.arange(0,len(WorkFrame))
    if val in CP:    ## if the interpolation method is of type IIM or
                     ## that there was ice accumulation between the
                     ## consecutive field measurements
        cond_start = WorkFrame['Date'] >= start
        cond_end = WorkFrame['Date'] < end
        slice_period = WorkFrame[cond_start & cond_end].copy()
        index_array = np.array(slice_period.index)[[0,-1]]
        for i in melt_rate_PDD:
            if start == i[0]:
                melt_rate = i[2]
        slice_period.index = np.arange(0,len(slice_period))
        values = slice_period.values[1:,2:-3]
        known_depths = slice_period.values[0,2:-3]
        PDD_cum = slice_period.values[1:,-1:].cumsum(axis=0)
        values[:] = known_depths + (melt_rate * PDD_cum)
        slice_period.ix[1:,2:-3] = values
        WorkFrame.ix[index_array[0]:index_array[1]] = slice_period
        WorkFrame = WorkFrame.sort(columns='Date',ascending=True)
        WorkFrame.index = np.arange(0,len(WorkFrame))
    else:
        for i,value in enumerate(Interp_data['Date']):
            if val[0] == value:
                interpolation = Interp_data['Interpolation'][i]
        tot_melt = val[2]
        if interpolation == 'SIM':
            cond_start = WorkFrame['Date'] > start
            cond_end = WorkFrame['Date'] <= end
            depths_start = WorkFrame[WorkFrame.Date == start].values[0,2:-3]
            slice_period = WorkFrame[cond_start & cond_end].copy()
            index_array = np.array(slice_period.index)[[0,-1]]
            slice_period = slice_period.sort(columns='Date',
                                             ascending=False)
            slice_period.index = np.arange(0,len(slice_period))
            values = slice_period.values[1:,2:-3]
            known_depths = slice_period.values[0,2:-3]
            PDD_cum = slice_period.values[1:,-1:].cumsum(axis=0)
            values[:] = known_depths - (mean_melt_rate_PDD * PDD_cum)
            values = values.astype(np.float64)
            depths_start = depths_start.astype(np.float64)
            snow_cover = slice_period.values[1:,-3]
            snow_cover = snow_cover.astype(np.bool)
            for v,dst in np.nditer([values, depths_start],
                                   flags=['external_loop'],
                                   op_flags=[['readwrite'],['readonly']],
                                   order='C'):
                if v[0] > dst[0]:
                    v[...] = dst
            for ds,d,sc in np.nditer([depths_start[0],values[:,0],snow_cover],
                                      flags=['external_loop'],
                                      op_flags=[['readonly'],['readonly'],
                                                ['readwrite']],
                                                order='F'):
                    sc[...] = (ds == d)
            slice_period.ix[1:,2:-3] = values
            slice_period.ix[1:,-3] = snow_cover
            slice_period = slice_period.sort(columns='Date',
                                             ascending=True)
            WorkFrame.ix[index_array[0]:index_array[1]] = slice_period
            WorkFrame = WorkFrame.sort(columns='Date',ascending=True)
            WorkFrame.index = np.arange(0,len(WorkFrame))
            sc_start = WorkFrame[['Date']][WorkFrame.Snow == True].values[0]
            sc_end = WorkFrame[['Date']][WorkFrame.Snow == True].values[-1]
            snow_covered.append((sc_start,sc_end))
        elif interpolation == 'ISM':
            # algorithm to define here for this type of interpolation
            continue
        elif interpolation == 'SSM':
            # algorithm to define here for this type of interpolation
            continue

## Merge the DataFrame of the depth of the sensors, with the one of
## the temperature measured by the sensors in one single DataFrame.
FinalFrame = pd.merge(WorkFrame, DFrame, on='Date', how='outer')
FinalFrame = FinalFrame.sort(columns='Date')

## Change the row numbers of the frame
FinalFrame.index = np.arange(0,len(FinalFrame))

## Fill the missing values in the FinalFrame, assuming that the depth
## difference for each sensor within a single day is neglectable.
FinalFrame.ix[:,1:3+data.num] = FinalFrame.ix[:,1:3+data.num]\
  .fillna(method='ffill')
FinalFrame.ix[:,-(data.num+2):-1] = FinalFrame.ix[:,-(data.num+2):-1]\
  .fillna(method='ffill')
FinalFrame.ix[:,-(data.num+2):-1] = FinalFrame.ix[:,-(data.num+2):-1]\
  .fillna(method='bfill')
list_cols = list(np.arange(0,data.num+2))
list_cols.pop(1)

## Depth values of the sensors for the whole period of measurments.
DepthFrame = FinalFrame[list_cols]
list_cols = list(np.arange(data.num+5,len(FinalFrame.columns)))
list_cols.insert(0,0)
list_cols.pop()

## Temperature values of the sensors for the whole period of
## measurements.
TempFrame = FinalFrame[list_cols]

## Merge the temperature and depth values into one single DataFrame,
## and set to 0 all the temperature values that are positive (sensors
## out of the ice measuring air temperature, or slightly positive
## values due to sensor accuracy).
PlotFrame = pd.merge(TempFrame, DepthFrame, on='Date', how='outer')
Temp = PlotFrame.ix[:,1:1+data.num].values
Temp[Temp > 0] = 0
Depth = PlotFrame.ix[:,1+data.num:].values

## Estimate the temperature of the ice at the surface. For each time
## the temperature is measured by the sensors, a second degree
## polynomial is fitted to the data (temperature against depth), and
## the temperture at the surface is extrapolated from the data, by
## reading the temperature value at a depth of 0 for the polynomial
## function. A second degree polynomial was chosen to be able to
## represent temperature diurnal variations to a certain point. No
## higher degree was chosen to avoid to much divergence of the
## polynomial fit.
T0 = np.empty((len(PlotFrame)))

for dep,temp,t0 in np.nditer([Depth,Temp,T0[:,None]], flags=['external_loop',
                                                              'reduce_ok'],
                             op_flags=[['readonly'],['readonly'],
                                       ['readwrite']], order='C'):
    z=np.polyfit(dep[dep>0],temp[dep>0],2)
    p = np.poly1d(z)
    t0[...] = p(0)
## The temperature values slightly positive are set to zero.
T0[T0>0] = 0
Temp = np.concatenate([T0[:,None],Temp],axis=1)
Depth = np.concatenate([np.zeros((len(Depth),1)),Depth],axis=1)

## Plot filled contours of the temperature in the ice, with the depth
## of the sensors updated with field observations and melt
## estimates. An upper subplot shows the mean daily air temperature
## for the whole period. The shaded areas are the periods when the ice
## is snow covered according to the melt model.
depthvalues = Depth
tempvalues = Temp
tempvalues[tempvalues>0] = 0
dateaxis = FinalFrame.Date.values
dt64todatetime = np.vectorize(lambda x: pd.to_datetime(x).to_datetime())
dateaxis = dt64todatetime(dateaxis)
datematrix = [x for item in dateaxis for x in repeat(item, data.num+1)]
datematrix = np.array(datematrix)
date2num = np.vectorize(lambda x: mpl.dates.date2num(x))
datematrix = date2num(datematrix)
datematrix = datematrix.reshape(datematrix.shape[0]/(data.num+1),data.num+1)
DTa = TaFrame.Date.values
DTa = dt64todatetime(DTa)
snow_covered = dt64todatetime(snow_covered)
numcolors = 30
cmap = plt.cm.get_cmap(name='jet',lut=numcolors)
fig = plt.figure(dpi=150)
ax1 = fig.add_subplot(2,1,1)
ax1.xaxis_date()
ax1.plot(DTa,TaFrame.Ta.values,'r-',
         label='Air temperature (°C)')
for i,val in enumerate(snow_covered):
    if i == 0:
        ax1.axvspan(val[0],val[1],facecolor='0.5', alpha=0.5,label='Snow Cover')
    else:
        ax1.axvspan(val[0],val[1],facecolor='0.5', alpha=0.5)
ax1.axhline(color='b',linewidth=.5,label='_nolegend_')
ax1.legend(loc='best')
ax1.set_ylabel('Temperature (°C)')
ax2 = fig.add_subplot(2, 1, 2, sharex=ax1)
plt.gcf().autofmt_xdate()
im = ax2.contourf(datematrix, depthvalues, tempvalues,
                numcolors,cmap=cmap,extend='both')
contour_levels = 0.5,1
```

```python
cs = ax2.contour(datematrix, depthvalues, tempvalues,
                contour_levels, linewidths=2, colors='k',hold='on')
ax2.set_ylabel('Depth (m)')
depth_lim = ax2.get_ylim()
ax2.set_ylim([0,depth_lim[1]])
ax2.invert_yaxis()
cbar = plt.colorbar(im,orientation='horizontal',ax=ax2, pad=0.25,
                    drawedges=True,shrink=0.8, extendfrac='auto')
cbar.set_label('Ice temperature (°C)')
fig.suptitle('Air and ice temperature at stake {}'.format(ID),
             fontsize=14)
fig.tight_layout()
plt.show()


## Computes ice deformation in the subsurface using the ice
## temperature data.

## Glacier and stake ID for velocity measurements stored in the
## data.hdf5 generated by the surfacevelocity.py and
## computevelocity.py scripts.
glacier = 'Storbreen'
stake = 'S2yr11'

## Activation energy for creep.
Qpos = 115000 # J/mol (if the ice is warmer than -10 degree C)
Qneg = 60000 # J/mol (if the ice is colder than -10 degree C)

## Universal gas constant.
R = 8.314 # J/mol/K

## Ice thickness at the stake.
H = 85 # m

## Surface slope in degrees.
surf_slope = 9
alpha = np.pi/180*surf_slope # rad

## Ice density.
rho = 917 # kg/m3

## Factor for conversion (number of seconds in a year).
sec2year = 365*24*60*60

## Pre-factor to compute the value of the creep parameter A (A at
## -10°C).
A0 = 3.5*10**-25 # 1/Pa3/s

## Gravitational acceleration constant.
g = 9.81 # m/s2

## zero Celsius degree in Kelvin.
zero = 273.15 # K

## Constant parameter of the relationship between stress and strain.
n = 3

## Approximation of the shear stress at the bed.
Tb = rho*g*H*alpha # Pa

## Number of points where the velocity will be estimated for each
## profile.
nb_points = 100

it = np.nditer([depthvalues,tempvalues], flags=['external_loop'],
               op_flags=[['readonly'],['readonly']],
               order='C')
num = tempvalues.shape[1] # Number of sensors
first_loop = True
for dep,temp in it:
    temperature = temp[dep>=0]
    depth = dep[dep>=0].astype(np.float128)
    step = np.max(depth)/nb_points
    for i in np.arange(len(depth)-1):
        ## Temperature gradient between neighbouring sensors on the
        ## thermistor string.
        a = (temperature[i+1]-temperature[i])/(depth[i+1]-depth[i])
        ## Temperature (in Kelvin) of the upper sensor of the
        ## interval, or surface temperature estimated if iterator 'i'
        ## is equal to zero.
        b = temperature[i] + zero
        ## Depth values for velocity estimation.
        if i==0:
            z = np.arange(depth[i], depth[i+1], step)
        else:
            z = np.arange(last+step, depth[i+1], step)
            z = np.insert(z, 0 , depth[i])
        last = z[-1] # last value of that depth interval
        z = np.append(z,depth[i+1])
        P = rho * g * z
        T_h = a*z + b + 7 * 10**-8 * P
```

```
            T_star = 263 + 7 * 10**-8 * P
            condition = T_h > T_star
            Q = np.where(condition, Qpos, Qneg)
            A = A0*np.exp((-Q/R)*((1/(T_h))-1/(T_star)))
            dvdz = 2*A*(Tb**n)*((z/H)**n)
            ## Deformation velocity between 2 sensors on the profile,
            ## approximated by the trapezoidale rule.
            Vz = cumtrapz(dvdz, z, initial=0)
            ## Convert from meters per second into meters per year.
            Vz *= sec2year
            if i==0:
                profile = Vz
                depth_velocity = z
            else:
                Vz += profile[-1]
                profile = np.delete(profile, -1)
                Vz = np.delete(Vz,0)
                profile = np.concatenate([profile,Vz])
                depth_velocity = np.delete(depth_velocity, -1)
                z = np.delete(z, 0)
                depth_velocity = np.concatenate([depth_velocity, z])
        if first_loop:
            ProfilesMatrix = profile
            DepthsMatrix = depth_velocity
            first_loop = False
        else:
            ProfilesMatrix = np.vstack((ProfilesMatrix, profile))
            DepthsMatrix = np.vstack((DepthsMatrix, depth_velocity))

    ## Creates a new datematrix for the velocity estimation points
    dateaxis = FinalFrame.Date.values
    dt64todatetime = np.vectorize(lambda x: pd.to_datetime(x).to_datetime())
    dateaxis = dt64todatetime(dateaxis)
    datematrix2 = [x for item in dateaxis for x in repeat(item, nb_points+1)]
    datematrix2 = np.array(datematrix2)
    date2num = np.vectorize(lambda x: mpl.dates.date2num(x))
    datematrix2 = date2num(datematrix2)
    datematrix2 = datematrix2.reshape(datematrix2.shape[0]/(nb_points+1),
                                      nb_points+1)

    ## Plot temperature and subsurface ice deformation in one figure
    fig = plt.figure(dpi=150)
    ax1 = fig.add_subplot(2,1,1)
    ax1.xaxis_date()
    im1 = ax1.contourf(datematrix, depthvalues, tempvalues,
                    numcolors,cmap=cmap,extend='both')
    contour_levels = 0.5,1
    cs = ax1.contour(datematrix, depthvalues, tempvalues,
                    contour_levels, linewidths=2, colors='k',hold='on')
    ax1.set_ylabel('Depth (m)')
    ax2 = fig.add_subplot(2, 1, 2, sharex=ax1, sharey=ax1)
    plt.gcf().autofmt_xdate()
    cbar = plt.colorbar(im1, orientation='horizontal', ax=ax1, pad=0.25,
                        drawedges=True,shrink=0.8, extendfrac='auto')
    cbar.set_label('Ice temperature (°C)')
    im2 = ax2.contourf(datematrix2, DepthsMatrix, ProfilesMatrix, numcolors,
                       cmap=cmap, extend='both')
    ax2.set_ylabel('Depth (m)')
    depth_lim = ax2.get_ylim()
    ax2.set_ylim([0,depth_lim[1]])
    ax2.invert_yaxis()
    cbar2 = plt.colorbar(im2, orientation='horizontal',ax=ax2, pad=0.25,
                        drawedges=True,shrink=0.8, extendfrac='auto')
    cbar2.set_label('Ice deformation (m/year)')
    fig.suptitle('Ice temperature and ice deformation at stake {}'.format(ID),
                 fontsize=14)
    fig.tight_layout()
    plt.show()
```

functions.py

Listing 6: This file contains the functions required to run the scripts `icetemperatureprofile.py` and `tempplot.py`.

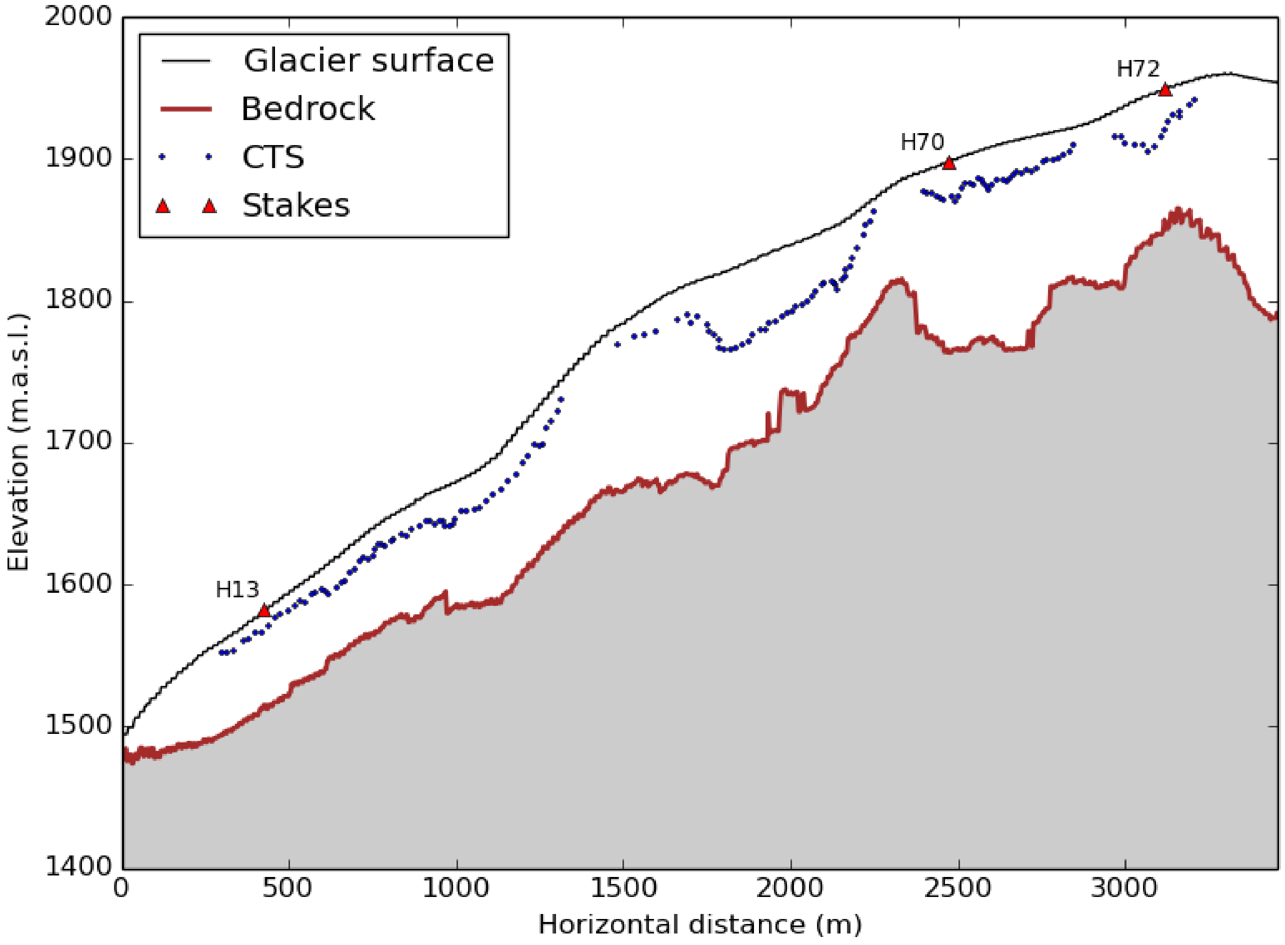
```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-

import os
import datetime as dt
import re
import pickle


## Function that computes the depth of the sensors.
def get_depth(dist, sensor, sensor_depth):
    """ Updates the depth of the sensor on the thermistor line.

    The function sets the depth of one of the sensors, and updates
    automatically the depth of the other sensors using the distances
    between them. It returns a dict object with the sensor numbers as
    keys, and the depths as values.


    Positional arguments:

    dist: dictionary that contains the positions of the sensors on the
    line.
    sensor:       the sensor that is used for the update.
    sensor_depth: the depth of the sensor.


    Note:

    A negative value for the parameter sensor_depth indicates how far
    out the sensor of interest is.
    """
    depths = dict()
    for key in dist.keys():
        if key != sensor:
            depths[key] = sensor_depth + (dist[key] - dist[sensor])
        else:
            depths[key] = sensor_depth
    for key,value in depths.items():
        depths[key] = float('{0:.2f}'.format(depths[key]))
    return depths


## Function that protect the files and directories created from
## writing by changing permissions.
def protect(ID):
    """ Protects from writing the data of a thermistor string.

    It changes the permissions of the 'data' directory, the
    subdirectory and the files of the thermistor string.


    Positional argument:

    ID: identification of the thermistor string.
    """
    os.chmod('data/{0}/{0}.csv'.format(ID),0o444)
    os.chmod('data/{0}/updates.txt'.format(ID),0o444)
    os.chmod('data/{0}/{0}_object'.format(ID),0o444)
    os.chmod('data/{}'.format(ID),0o555)
    os.chmod('data',0o555)


## Function that changes the permissions on the files and directories
## of the temperature string to make them editable.
def unprotect(ID):
    """ This function makes editable the data of a thermistor string.

    It changes the permissions of the 'data' directory, the
    subdirectory and the files of the thermistor string.


    Positional argument:

    ID: identification of the thermistor string.
    """
    os.chmod('data',0o777)
    os.chmod('data/{}'.format(ID),0o777)
    os.chmod('data/{0}/{0}_object'.format(ID),0o666)
    os.chmod('data/{0}/updates.txt'.format(ID),0o666)
    os.chmod('data/{0}/{0}.csv'.format(ID),0o666)

```

```python

## Function that finds the date of the last field measurements.
def last_update(ID):
    """ Returns the date of the last field observations.

    It extracts the date which appears on the last line in the text
    file of the thermistor string.


    Positional argument:

    ID: identification of the thermistor string.
    """
    with open('data/{0}/updates.txt'.format(ID),'r') as file_txt:
        last_line = file_txt.readlines()[-1]
        whole_date = re.findall(r'^\d{4}-\d{2}-\d{2}',last_line)
        lyear,lmonth,lday = whole_date[0].split('-')
        lyear, lmonth, lday = int(lyear), int(lmonth), int(lday)
        last_update = dt.datetime(lyear,lmonth,lday).strftime('%A %d %B %Y')
    return last_update


## Function that sorts the field observations in the text file.
def sort_observations(ID,headerlines=2):
    """ Sorts the lines in the updates.txt file of a thermistor string.

    It sorts the lines using the date of the field observations. The
    most recent observations are at the end of the file.


    Positional argument:

    ID: identification of the thermistor string.


    Keyword argument:

    headerlines: number of lines not interpreted by the function
    (default is 2).
    """
    with open('data/{0}/updates.txt'.format(ID),'r') as file_txt:
        content = file_txt.readlines()
        first_lines = content[:headerlines]
        lines = content[headerlines:]
        lines.sort()
        content = ''.join(first_lines+lines)

    with open('data/{0}/updates.txt'.format(ID),'w') as file_txt:
        file_txt.write(content)


## Function that returns the values of the keyword arguments snow and
## thickness for the latest field observations.
def last_snowpack(ID):
    """ This function returns the values of 'snow' and 'thickness'.

    It returns the values of the keyword aguments 'snow' and
    'thickness' of an instance of the class Profile, for the latest
    field observation.  It extracts these values from the last line of
    the text file (updates.txt), for the corresponding thermistor
    string.


    Positional argument:

    ID: identification of the thermistor string.
    """
    pattern = re.compile(r'(?P<snow>\w{4,5}),(?P<thickness>(\d|\.)+)$')
    with open('data/{0}/updates.txt'.format(ID),'r') as file_txt:
        last_line = file_txt.readlines()[-1]
        match = pattern.search(last_line)
    return match.group('snow'),match.group('thickness')


## Function that returns the depth values of the sensors for the
## latest field observations.
def last_depth(ID):
    """ This function returns the depth values of the sensors.

    It returns the values of the depth values of the sensors of an
    instance of the class Profile, for the latest field observation.
    It extracts these values from the last line of the text file
    (updates.txt), for the corresponding thermistor string.


    Positional arguments:

    ID: identification of the thermistor string.
    """
    pattern=re.compile(r'(?<=\d,)((\d|-|\.)+)')
    depth = dict()
    with open('data/{0}/updates.txt'.format(ID),'r') as file_txt:
        last_line = file_txt.readlines()[-1]
        depths = pattern.findall(last_line)
        for i,value in enumerate(depths):
            depth[i+1] = float(value[0])
    return depth
```

Appendix E

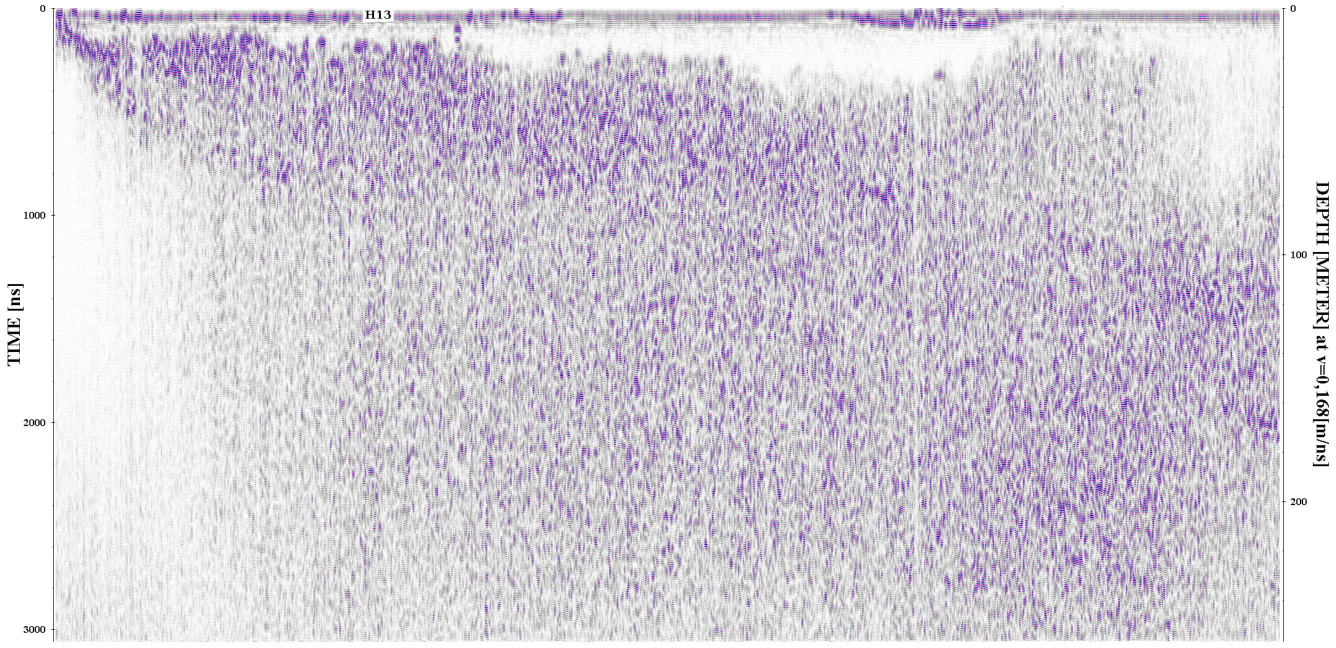
Mapping of the Cold-temperate Transition Surface

E.1 Hellstugubreen

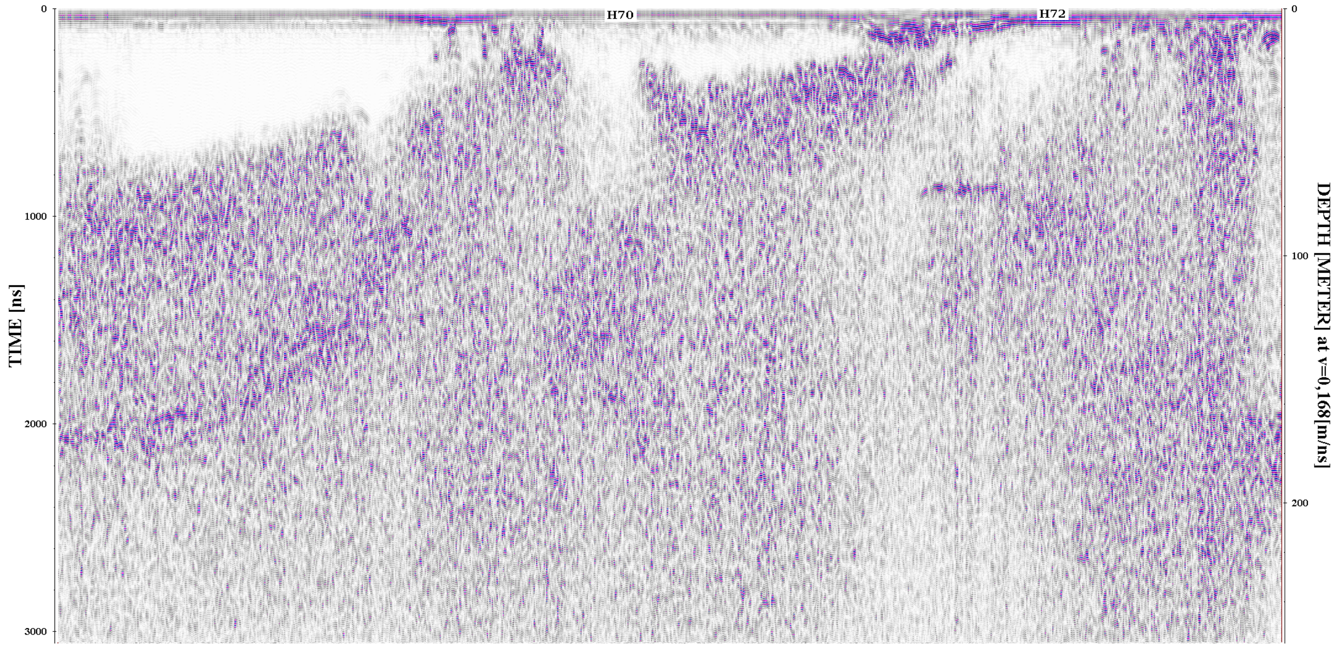


CTS mapping on profile H167. The CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements (10 MHz) from 2011 (data : NVE). The CTS was not digitized near the glacier front, as this could not be done accurately owing to subsurface structures and frequent signal scattering patterns.

(a) H167 profile (first half)

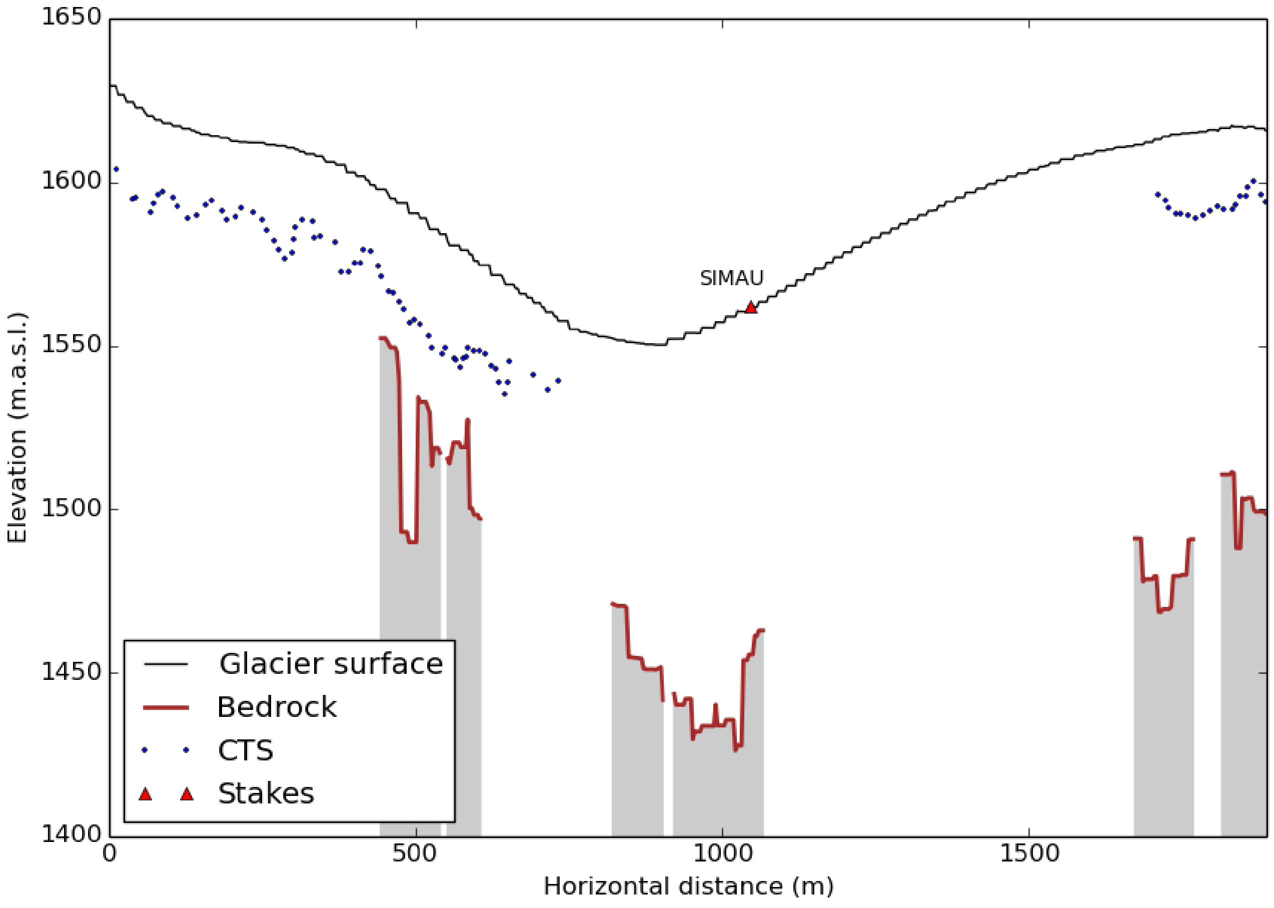


(b) H167 profile (second half)



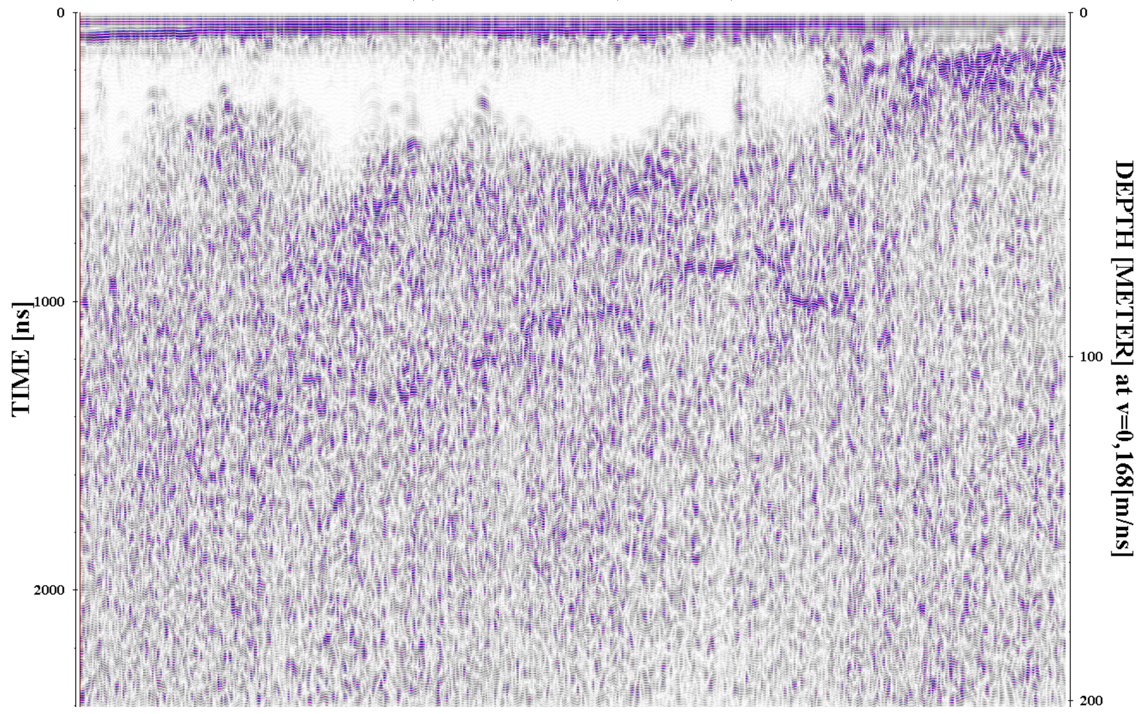
(a) and (b) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014) along the profile H167. The distance along the profile shown in (a) and (b) increases from left to right.

E.2 Storbreen

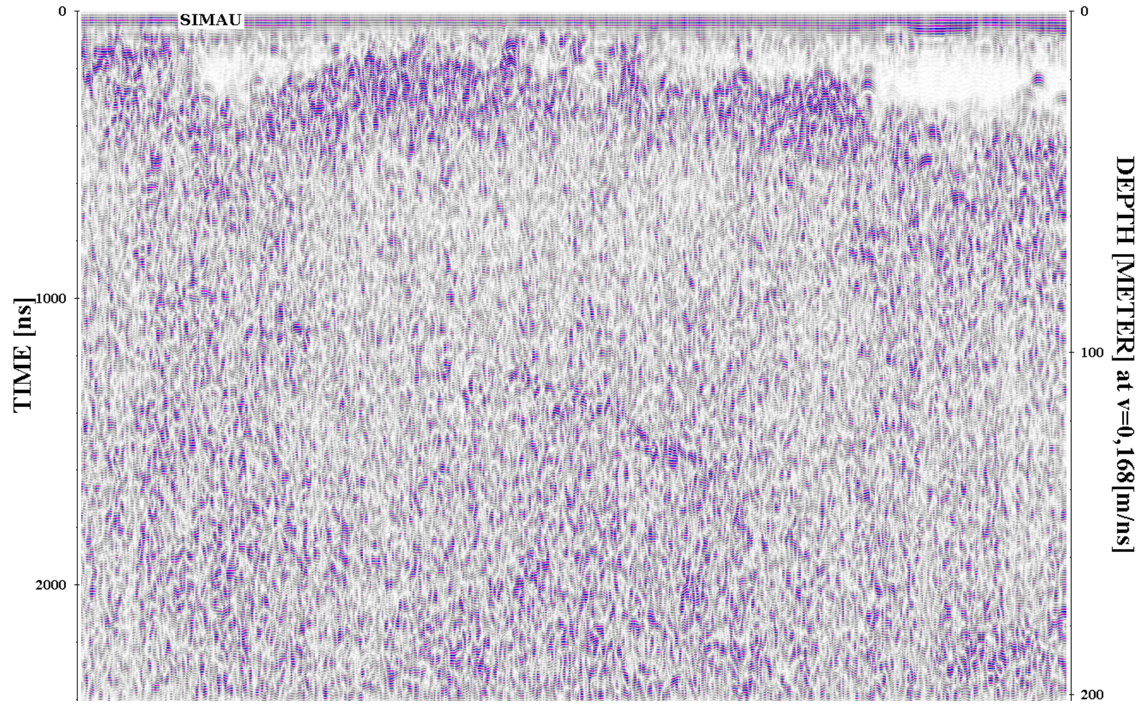


CTS mapping on profile S180. The CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements from 2005-2006 (10 MHz, data : NVE) and from 2014 (50 MHz).

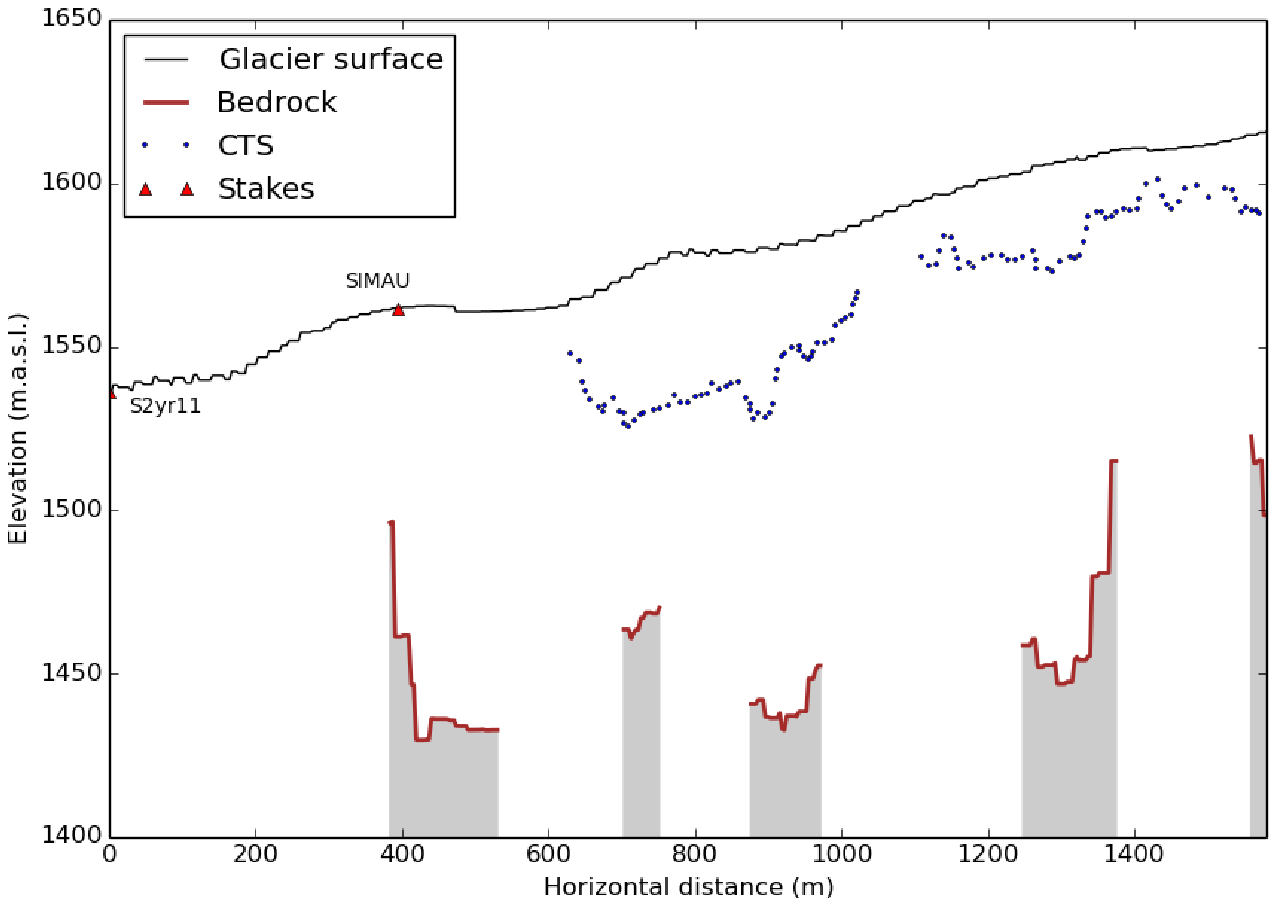
(a) S180 profile (first half)



(b) S180 profile (second half)

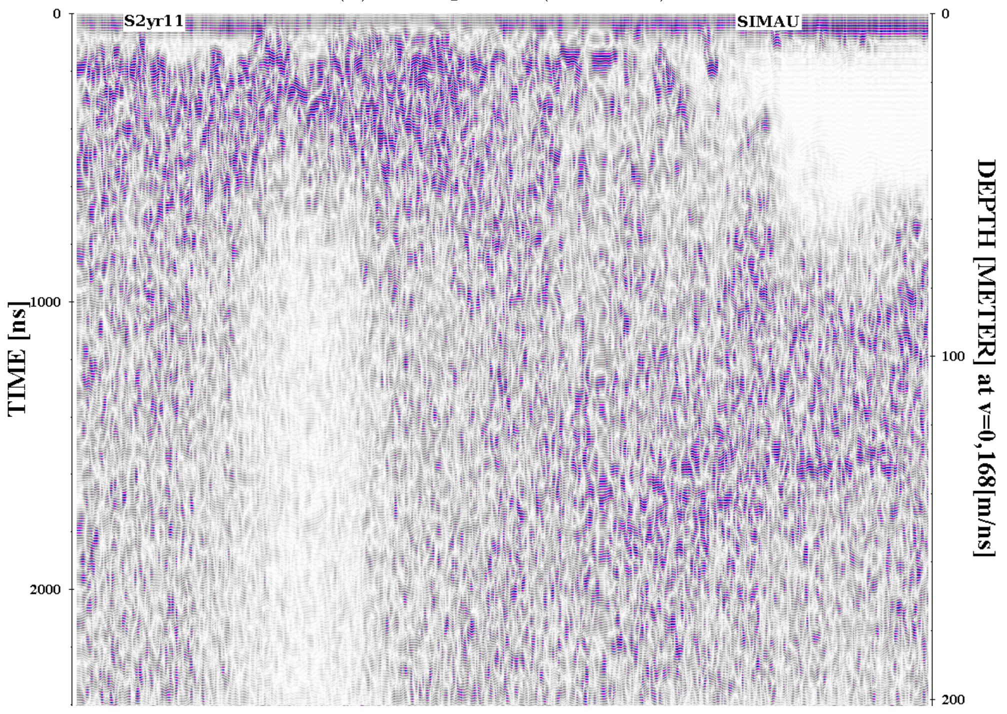


(a) and (b) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014) along the profile S180. The distance along the profile shown in (a) and (b) increases from left to right.

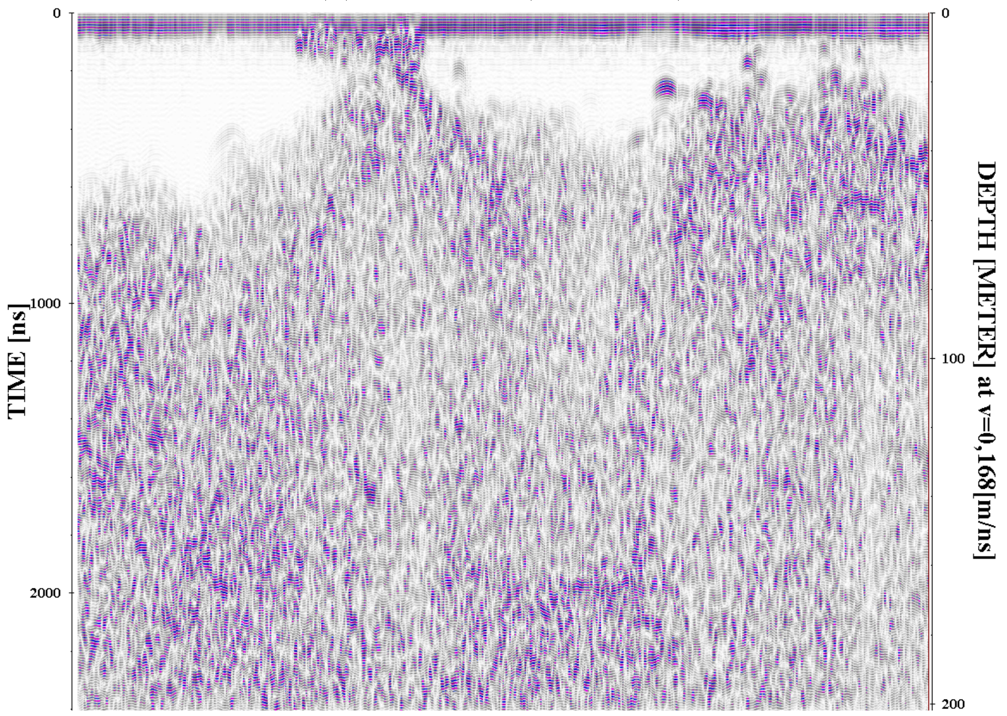


CTS mapping on profile S181. The CTS was digitized on radargrams from 2014 (50 MHz), the glacier surface is derived from LiDAR data (data : NVE, 2009), and the ice/bedrock interface is derived from RES measurements from 2005-2006 (10 MHz, data : NVE) and from 2014 (50 MHz). The CTS was not digitized in proximity of stake S2yr11, as this could not be done accurately owing to subsurface structures and frequent signal scattering patterns.

(a) S181 profile (first half)



(b) S181 profile (second half)

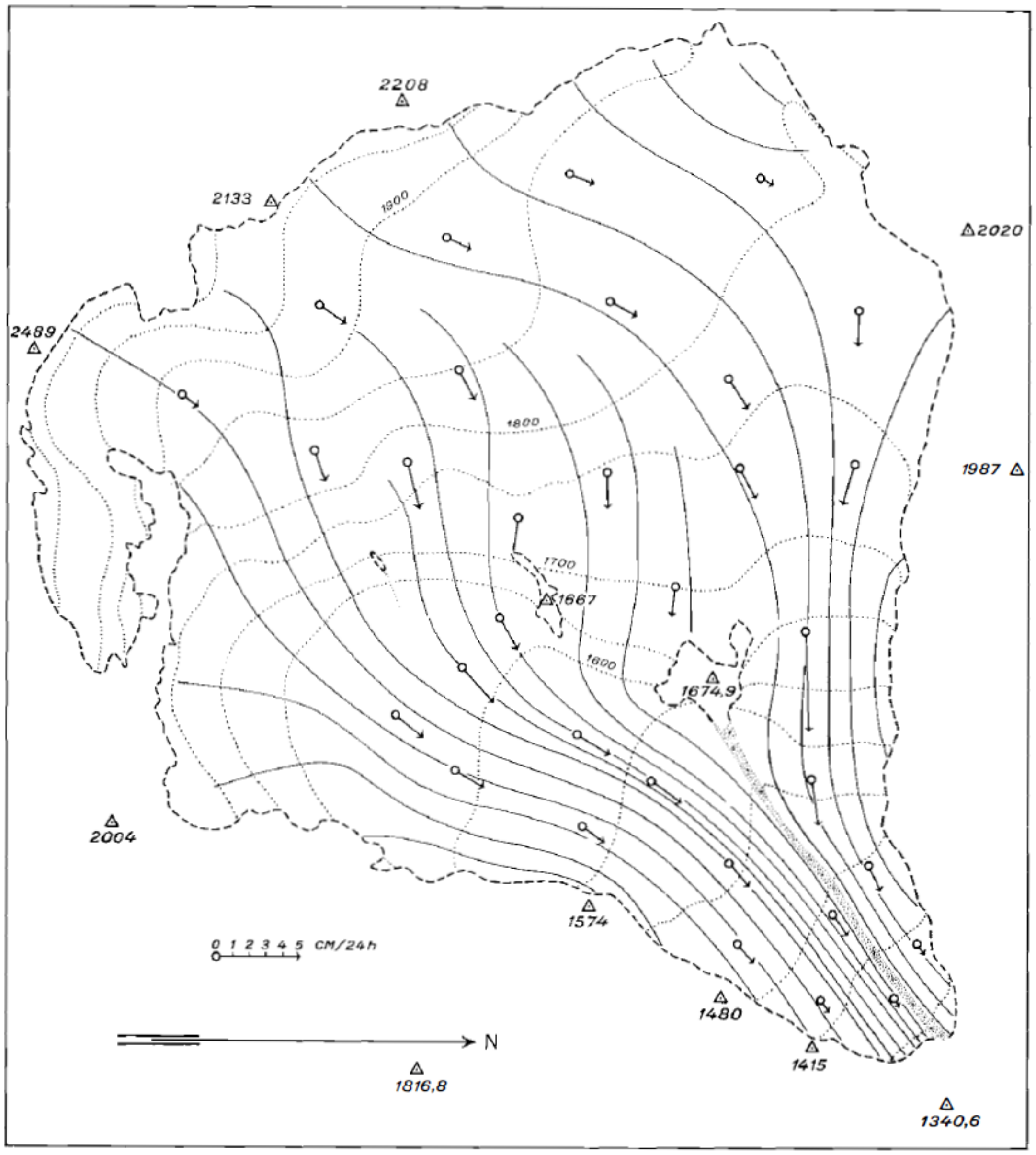


(a) and (b) are intensity-modulated plots of internal reflections of the 50 MHz GPR antenna (2014) along the profile S181. The distance along the profile shown in (a) and (b) increases from left to right.

Appendix F

Ice surface velocity

F.1 Surface velocities on Storbreen in the 1960s



Surface velocity map of Storbreen, resulting from triangulation measurements conducted in the 1960s (Liestøl, 1967).

F.2 Python code

surfacevelocity.py

Listing 7: This program enables to save the positions of the stakes, for a given glacier, and from different field measurements. The positions of the stakes are saved in a data.hdf5 file.

```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-


import datetime as dt
import os
import re
import argparse


import numpy as np
import h5py


## Identification of the stake location
ID = 'H60yr2009'

## Glacier name
glacier = 'Hellstugubreen'

## Start date of the velocity measurements
date_start = dt.datetime(2010,5,7)

## Northing and Easting coordinates of the stake at start date. UTM
## coordinates system, in meters.
northing = 6825519.2056
easting = 470624.9577

## Main class
class StakeVelocity:

    dtype = [('timestamp', np.float64), ('northing', np.float64),
             ('easting', np.float64), ('loc', np.int8),
             ('distance', np.float32)]
    def __init__(self, glacier, ID, northing, easting, date_start):

        """ Create the attributes of an instance of the class StakeVelocity.

        Positional Arguments:
        glacier    -- name of the glacier
        ID         -- identification of the stake location
        northing   -- northing coordinate at the start date
        easting    -- easting coordinate at the start date
        date_start -- start date
        """
        ## Check is there is already a data.hdf5 file created in the
        ## same folder
        if not os.path.exists('data.hdf5'):
            if not isinstance(glacier,str):
                raise TypeError('The name of the glacier should be '
                                'an instance of the class str!')
            if not ID.isalnum() or len(ID) < 2:
                raise TypeError('ID not valid! It should be alphanumeric'
                                ' and at least two caracters.')
            else:
                self._glacier = glacier
                self._ID = ID
        else:
            f = h5py.File('data.hdf5','r')
            if f.get('StakePositions/{0}/{1}'.format(glacier,ID)):
                f.close()
                raise TypeError('A stake location has already this ID, '
                                'choose another ID.')
            else:
                f.close()
                if not isinstance(glacier,str):
                    raise TypeError('The name of the glacier should be '
                                    'an instance of the class str!')
                if not ID.isalnum() or len(ID) < 2:
                    raise TypeError('ID not valid! It should be alphanumeric'
                                    ' and at least two caracters.')
                else:
                    self._glacier = glacier
                    self._ID = ID

        ## Check that 'northing' and 'easting' have the right format
        if not isinstance(northing,int) and not isinstance(northing,float):
            raise TypeError('The attribute northing must either be an'
                            ' integer or a floatting point number!')
        if not isinstance(easting,int) and not isinstance(easting,float):
            raise TypeError('The attribute easting must either be an'
                            ' integer or a floatting point number!')
        self._northing = northing
        self._easting = easting

        ## Check that 'date_start' has the right format
        if not isinstance(date_start,dt.datetime):
            raise TypeError('The attribute date_start must be an '
                            'instance of the class datetime.datetime!')
        self._date_start = date_start

        ## Correct the position of the dGNSS antenna if needed
        ## (downstream or upstream offset corrections available).
        correction = str()
        while correction != 'y' and correction != 'n':
            correction = input('Is there any correction to apply to '
                               'the position coordinates of the stake'
                               ' ? (y/n)\n')
            correction = correction.lower()
        loc = str()
        dist = str()
        if correction == 'y':
            while loc != 'downstream' and loc != 'upstream':
                loc = input('Was the GNSS antenna "upstream" or '
                            '"downstream" the stake during the field '
                            'measurement ?\n')
                loc = loc.lower()
            while not isinstance(dist,np.float32):
                dist = input('What was the distance (in meters) '
                             'between the antenna and the stake ?\n')
                try:
                    dist = np.float32(dist)
                except ValueError:
                    print('The distance must be an integer or a '
                          'floatting point number!')
                if isinstance(dist,np.float32):
                    if not dist > 0:
                        dist= str()
                        print('The distance must be a positive value!')
            if loc == 'downstream':
                loc = -1
            else:
                loc = 1
        else:
            loc,dist = 0,0

        ## Save object and updates in a .hdf5 file
        with h5py.File('data.hdf5','a') as f:
            if not f.get('/StakePositions/{0}/{1}'.format(glacier,ID)):
                dset = f.create_dataset('/StakePositions/{0}/{1}/data'\
                                        .format(glacier,ID), (1,),
                                        dtype=StakeVelocity.dtype,
                                        maxshape=(None,), compression=9,
                                        shuffle=True, fletcher32=True)
                data = np.array([(date_start.timestamp(), northing, easting,
                                  loc, dist)], dtype=StakeVelocity.dtype)
                dset[...] = data
                dset.attrs['Description'] = ("'timestamp' : Unix time. "
                                             "'northing' and 'easting' : "
                                             "UTM coordinate system, "
                                             "zone 32V. 'loc' : location"
                                             " of the GNSS antenna "
                                             "relatively to the stake,"
                                             " during the field "
                                             "measurements (0: no offset,"
                                             " -1: downstream, 1: "
                                             "upstream). 'distance' : "
                                             "distance GNSS antenna to"
                                             " stake (meters).")

    ## If a stake is already saved in the .hdf5 file, this class
    ## method updates the position of stake at a different date.
    @classmethod
    def update(cls, glacier=None, ID=None ):
        """ Updates the new position of a stake.
        """
        f = h5py.File('data.hdf5','r+')
        glaciers_list = list()
        for g in f['StakePositions'].keys():
            glaciers_list.append(g)
            glaciers_list.sort()
            string = '\n'.join(glaciers_list)
        if not glacier:
            print('There is/are {0} existing glacier(s) on this data '
                  'file :\n{1}\n'.format(len(glaciers_list),string))
            glacier = input('Which glacier do you want to update data'
                            ' from ?\n')
        if not glacier in glaciers_list:
            raise NameError('{} is not a valid name for any existing '
                            'glacier on this data file!'.format(glacier))
        if not ID:
            ID = input('Which stake do you want to update the position'
                       ' ?\n')
        if not ID in f['StakePositions/{}'.format(glacier)]:
            raise NameError('{} is not a valid name for any existing '
                            'stake!'.format(ID))
        f.close()
        northing = str()
        while not isinstance(northing,np.float64):
            northing = input('What is the UTM northing values ?\n')
            try:
                northing = np.float64(northing)
            except ValueError:
                print('You must enter an integer or a floatting point'
                      ' number!')
        easting = str()
        while not isinstance(easting,np.float64):
            easting = input('What is the UTM easting values ?\n')
            try:
                easting = np.float64(easting)
            except ValueError:
                print('You must enter an integer or a floatting point'
                      ' number!')

        ## Correct the position of the dGNSS antenna if needed
        correction = str()
        while correction != 'y' and correction != 'n':
            correction = input('Is there any correction to apply to '
                               'the position coordinates of the stake'
                               ' ? (y/n)\n')
            correction = correction.lower()
        loc = str()
        dist = str()
        if correction == 'y':
            while loc != 'downstream' and loc != 'upstream':
                loc = input('Was the GNSS antenna "upstream" or '
                            '"downstream" the stake during the field '
                            'measurement ?\n')
                loc = loc.lower()
            while not isinstance(dist,np.float64):
                dist = input('What was the distance (in meters) '
                             'between the antenna and the stake ?\n')
                try:
                    dist = np.float64(dist)
                except ValueError:
                    print('The distance must be an integer or a '
                          'floatting point number!')
                if isinstance(dist,np.float64):
                    if not dist > 0:
                        dist= str()
                        print('The distance must be a positive value!')
            if loc == 'downstream':
                loc = -1
            else:
                loc = 1
        else:
            loc,dist = 0,0

        ## Update the date of the field measurements
        year = input('What is the date matching to the update?\n'
                     'Year : ')
        try:
            year = int(year)
        except ValueError:
            raise ValueError('The year must be an integer value!')
        if not re.match(r'\d{4}',str(year)):
            raise ValueError('The year is not valid (4 digits)!\n'
                             'Example of valid year : 2014')
        month = input('Month (1 - 12): ')
        try:
            month = int(month)
        except ValueError:
            raise ValueError('The month must be an integer value!')
        if month < 1 or month > 12:
            raise ValueError('The month must be a value between 1 and'
                             ' 12 included.')
        dom = input('Day of month (1 - 31) : ')
        try:
            dom = int(dom)
        except ValueError:
            raise ValueError('The day of month must be an integer value (1-31)!')
        if dom < 1 or dom > 31:
            raise ValueError('The day of month must be a value between'
                             ' 1 and 31 included.')
        HM = str()
        while HM.lower() != 'y' and HM.lower() != 'n':
            HM = input('Do you also want to update the time (hours and'
                       ' minutes)? (y/n)\n')
```

```python
        HM = HM.lower()
        if HM == 'y':
            hours = input('Hours (0-23) : ')
            try:
                hours = int(hours)
            except ValueError:
                raise ValueError('The number of hours must be an integer value!')
            if hours < 0 or hours > 23:
                raise ValueError('The number of hours must be a value'
                                 ' between 0 and 23 included.')
            minutes = input('Minutes (0-59) : ')
            try:
                minutes = int(minutes)
            except ValueError:
                raise ValueError('The number of minutes must be an integer value!')
            if minutes < 0 or minutes > 59:
                raise ValueError('The number of minutes must be a '
                                 'value between 0 and 59 included.')
            timestamp_update = dt.datetime(year,month,dom,
                                           hours, minutes).timestamp()
        else:
            timestamp_update = dt.datetime(year, month, dom).timestamp()

        ## Write the updates in the data.hdf5 file
        with h5py.File('data.hdf5','r+') as f:
            dset = f['StakePositions/{0}/{1}/data'.format(glacier,ID)]
            if timestamp_update in dset['timestamp']:
                for i in dset[...]:
                    if timestamp_update == i['timestamp']:
                        dset[i] = np.array([(timestamp_update, northing,
                                             easting, loc, dist)],
                                             dtype=StakeVelocity.dtype)
            else:
                dset.resize((dset.shape[0]+1,))
                dset[-1] = np.array([(timestamp_update,northing,easting,
                                      loc, dist)],
                                      dtype=StakeVelocity.dtype)
                arr = dset[...]
                arr = arr[arr['timestamp'].argsort()] # sort by date
                dset[...] = arr

## Define optional arguments when running the file from the terminal
parser =  argparse.ArgumentParser()
group = parser.add_mutually_exclusive_group()
group.add_argument('-u', '--update', action='store_true',
                    help=('update a stake location with new field '
                          'observations'))
group.add_argument('-U', '--updateStake', metavar=('glacier','ID'),
                   nargs=2, help=('update the stake location matching '
                                  'to the glacier and the ID with new '
                                  'field observations'))
args = parser.parse_args()
if args.update:
    StakeVelocity.update()
if args.updateStake:
    StakeVelocity.update(glacier=args.updateStake[0],
                         ID=args.updateStake[1])
```

computevelocity.py

Listing 8: This script computes the velocities from the field observations written in the data.hdf5 file.

```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-


import datetime as dt


import numpy as np
import h5py


## List of glaciers name to compute the velocity for. Data from these
## glaciers (stake positions) need to be saved in the data.hdf5 file
## for the velocities to be computed. If the list is empty, the
## velocities will be computed for each glacier found in the data.hdf5
## file.
glaciers =list() # string elements (e.g. 'Hellstugubreen')

## Creates a compound dtype for the velocity datasets
dtype = [('velocity', np.float64), ('start', np.float64),
         ('end', np.float64), ('northing', np.float64),
         ('easting', np.float64)]

## Open the data.hdf5 file, compute and save the velocity values
with h5py.File('data.hdf5','r+', driver='core') as f:
    if glaciers:
        for glacier in glaciers:
            folder = f['StakePositions/{}'.format(glacier)]
            for stake in folder.keys():
                data = f['StakePositions/{0}/{1}/data'.format(glacier,
                                                              stake)][...]
                if f.get('Velocity/{0}/{1}/data'.format(glacier, stake)):
                    del f['Velocity/{0}/{1}/data'.format(glacier, stake)]
                vel_dset = f.create_dataset('Velocity/{0}/{1}/data'\
                                            .format(glacier, stake), (0,),
                                            dtype=dtype, maxshape=(None,),
                                            compression=9, shuffle=True,
                                            fletcher32=True)
                for i,v in enumerate(data[1:]):
                    dt = v['timestamp'] - data[i]['timestamp']
                    dn = v['northing'] - data[i]['northing']
                    de = v['easting'] - data[i]['easting']
                    tot_offset = (v['loc'] * v['distance']) + \
                      (data[i]['loc'] * data[i]['distance'])
                    velocity = ((np.sqrt(dn**2 + de**2) +
                                 tot_offset) / dt) #m/s
                    start = data[i]['timestamp'] #timestamp
                    end = v['timestamp'] #timestamp
                    arr = np.array([(velocity, start, end, v['northing'],
                                    v['easting'])], dtype=dtype)
                    vel_dset.resize((vel_dset.shape[0]+1,))
                    vel_dset[-1] = arr
    else:
        for glacier in f['StakePositions'].keys():
            for stake in f['StakePositions/{}'.format(glacier)].keys():
                data = f['StakePositions/{0}/{1}/data'\
                         .format(glacier, stake)][...]
                if f.get('Velocity/{0}/{1}/data'.format(glacier, stake)):
                    del f['Velocity/{0}/{1}/data'.format(glacier, stake)]
                vel_dset = f.create_dataset('Velocity/{0}/{1}/data'\
                                            .format(glacier, stake), (0,),
                                            dtype=dtype, maxshape=(None,),
                                            compression=9, shuffle=True,
                                            fletcher32=True)
                for i,v in enumerate(data[1:]):
                    dt = v['timestamp'] - data[i]['timestamp']
                    dn = v['northing'] - data[i]['northing']
                    de = v['easting'] - data[i]['easting']
                    tot_offset = (v['loc'] * v['distance']) + \
                      (data[i]['loc'] * data[i]['distance'])
                    velocity = ((np.sqrt(dn**2 + de**2) +
                                 tot_offset) / dt) #m/s
                    start = data[i]['timestamp'] #timestamp
                    end = v['timestamp'] #timestamp
                    arr = np.array([(velocity, start, end, v['northing'],
                                    v['easting'])], dtype=dtype)
                    vel_dset.resize((vel_dset.shape[0]+1,))
                    vel_dset[-1] = arr
```

plotsurfacevelocity.py

Listing 9: This script writes a csv file with the averaged velocities of the stakes, for a given glacier, and for a time period defined between a 'start date' and an 'end date'. It also plots (2D) all the averaged velocities for all stakes of the given glacier that are written in the data.hdf5 file.

```python
#!/usr/bin/python3.4
# -*- coding: utf-8 -*-


import datetime as dt
import calendar
import re
from itertools import repeat


import numpy as np
import pandas as pd
from pandas import DataFrame, Series
import h5py
import matplotlib.pyplot as plt
from matplotlib.ticker import ScalarFormatter


## Name of the glacier in the data.hdf5 file.
Glacier = 'Storbreen'

## Factor to use to convert m/s into m/year.
sec2year = 365*24*60*60

## Find all the velocity measurements for this glacier in the
## data.hdf5 file.
with h5py.File('data.hdf5','r+', driver='core') as f:
     folder = f['Velocity/{}'.format(Glacier)]
     velocities = dict()
     for stake in folder.keys():
         velocities['{}'.format(stake)] = folder['{}/data'\
                                                 .format(stake)][...]
velocities_formatted = dict()
for stake,data in velocities.items():
    velocities_formatted['{}'.format(stake)] = [(i[0]*sec2year,
                                                 dt.datetime.fromtimestamp(i[1]),
                                                 dt.datetime.fromtimestamp(i[2]),
                                                 i[3],
                                                 i[4])
                                                 for i in data]

## Creates a pandas.Dataframe where to store temporarily tha data read
## from the hdf5 file.
names = ['Velocity', 'Start', 'End', 'Northing', 'Easting']
WorkFrame = DataFrame(columns=names)
for stake,data in velocities_formatted.items():
    for i in data:
        WorkFrame = WorkFrame.append(DataFrame([i], index=[stake],
                                               columns=names))
WorkFrame.reset_index(inplace=True)
WorkFrame = WorkFrame.rename(columns={'index':'Stake'})
dt64todatetime = lambda x: pd.to_datetime(x).to_datetime()

## Calculate the number of days over which each velocity value was
## averaged.
numberDays = [(dt64todatetime(WorkFrame.ix[i, 'End']) -
               dt64todatetime(WorkFrame.ix[i, 'Start'])).days
              for i in np.arange(len(WorkFrame))]
WorkFrame['numberDays'] = numberDays

## Choose a start and end date for the computed velocity values.
start_year = 2012
end_year = 2014
start_month = 8
end_month = 10
start_date = dt.datetime(start_year, start_month, 1)
end_day = calendar.monthrange(end_year, end_month)[1]
end_date = dt.datetime(end_year, end_month, end_day)

cond_start = WorkFrame.Start >= start_date
cond_end = WorkFrame.End <= end_date
WorkFrame = WorkFrame[cond_start & cond_end]

## Computes the total number of days used for each stake, between the
## first field measurements after the start_date, and the last field
## measurements before the end_date.
tot_days = WorkFrame['numberDays'].groupby(WorkFrame.Stake).sum()
tot_days = DataFrame(tot_days,columns=['totDays'])
```

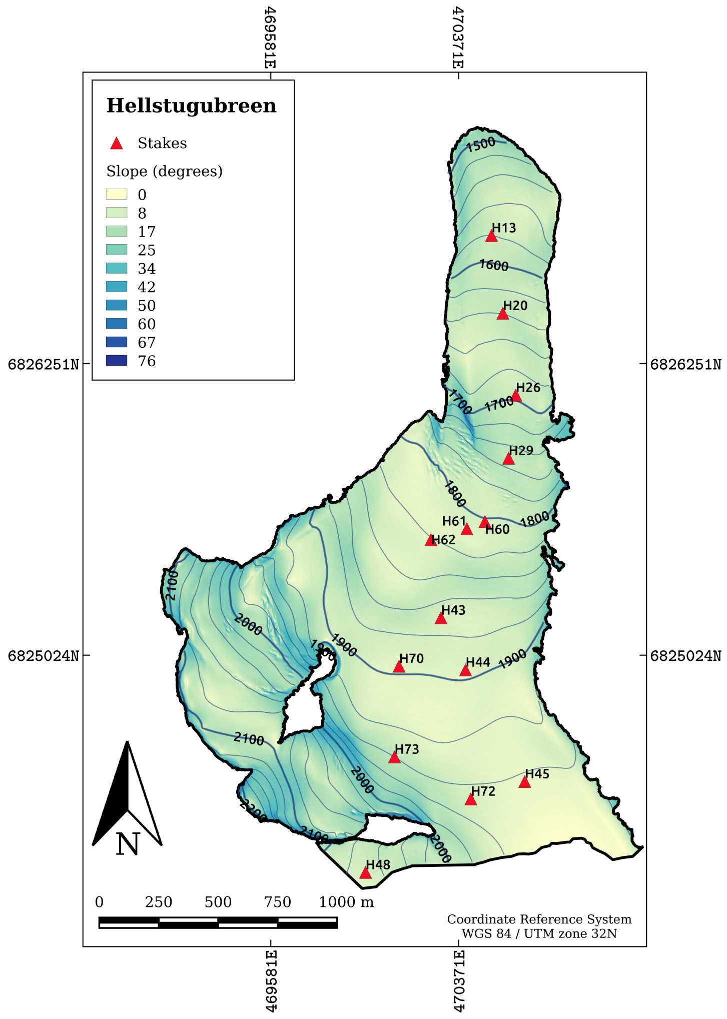
```
tot_days.reset_index(inplace=True)
WorkFrame = pd.merge(WorkFrame, tot_days, on='Stake', how='outer')

## Computes the weighted velocities for each stake and for each period
## between two consecutive field measurements. The weight used is the
## ratio of the number of days between two consecutive measurements,
## and the total number of days for the stake between start and end
## dates. The weight is then multiplied to each displacement distance
## observed between consecutive measurements.
WorkFrame['weightedVelocity'] = WorkFrame.Velocity * \
  (WorkFrame.numberDays.values/WorkFrame.totDays.values)

## The weighted velocities are summed for the period between start and
## end dates, to obtain the mean velocity for this period, and for
## each stake.
tot_vel = WorkFrame['weightedVelocity'].groupby(WorkFrame.Stake).sum()
tot_vel = DataFrame(tot_vel, columns=['totVelocity'])
tot_vel.reset_index(inplace=True)
WorkFrame = pd.merge(WorkFrame, tot_vel, on='Stake', how='outer')
stakes = np.unique(WorkFrame.Stake.values)

## Creates a new pandas.DataFrame with updated velocity values for
## each stake.
names = ['Stake', 'totVelocity', 'Start', 'End', 'Northing', 'Easting']
FinalFrame = DataFrame(columns=names)
for stake in stakes:
    stakeFrame = WorkFrame[WorkFrame.Stake == stake]
    start = stakeFrame.Start.argmin()
    end = stakeFrame.End.argmax()
    stakeSeries = stakeFrame.ix[start,['Stake','Start','totVelocity']]\
      .append(stakeFrame.ix[end,['End','Northing','Easting']])
    FinalFrame = FinalFrame.append(stakeSeries, ignore_index=True)

## Save the DataFrame including velocity values in a csv file.
dateformat = '%Y-%m-%d'
start_date = dt.datetime.strftime(start_date, dateformat)
end_date = dt.datetime.strftime(end_date, dateformat)
FinalFrame.to_csv('{}Velocity{}to{}.csv'.format(Glacier, start_date, end_date),
                  sep=',')

## Change the order of the elements in the dictionary of velocities,
## so that they are sorted according to an alphanumerically.
def sorted_nicely(it):
    """ Sorts the given iterable in the way that is expected.


    Positional argument:
    it: the iterable (stake) to be sorted.
    """
    convert = lambda text: int(text) if text.isdigit() else text
    alphanum_key = lambda key: [convert(c) for c in re.split('([0-9]+)', key)]
    return sorted(it, key = alphanum_key)
velocities = dict()
for k,v in velocities_formatted.items():
    if v:
        velocities['{}'.format(k)] = v
keys = list(velocities.keys())
keys = sorted_nicely(keys)

## Creates a customized color map to ease the differentiation of the
## stakes curves on the figure.
number = len(keys)
cmap = plt.get_cmap('gist_rainbow')
colors = [cmap(i) for i in np.linspace(0, 1, number)]
markerTypes = ['s', 'p', 'D','h', '*']
markers = list()
iterator = 0
while len(markers) < number:
    markers.append(markerTypes[iterator])
    if iterator == (len(markerTypes) - 1):
        iterator = 0
    else:
        iterator += 1
style = {stake: (color,marker) for stake,color,marker in \
         zip(keys, colors, markers)}

## Plots the averaged velocity values for each stake, which have field
## observations saved in the data.hdf5 file.
fig = plt.figure(dpi=150)
for k in keys:
    xaxis = list()
    yaxis = list()
    for date in velocities['{}'.format(k)]:
        xaxis.append(date[1])
        xaxis.append(date[2])
        yaxis.append(date[0])
    yaxis = [x for item in yaxis for x in repeat(item, 2)]
    plt.plot(xaxis, yaxis, label='{}'.format(k), ls='solid',
             alpha=0.5, color=style['{}'.format(k)][0], lw=2,
             marker=style['{}'.format(k)][1], markersize=5)
ax = fig.gca()
```

```
plt.yscale('log')
ax.yaxis.set_major_formatter(ScalarFormatter())
plt.gcf().autofmt_xdate()
plt.legend(loc='best', prop={'size':8})
plt.xlabel('Time')
plt.ylabel('Surface velocity ($\mathregular{m.yr^{-1}}$)')
fig.tight_layout()
plt.show()
```

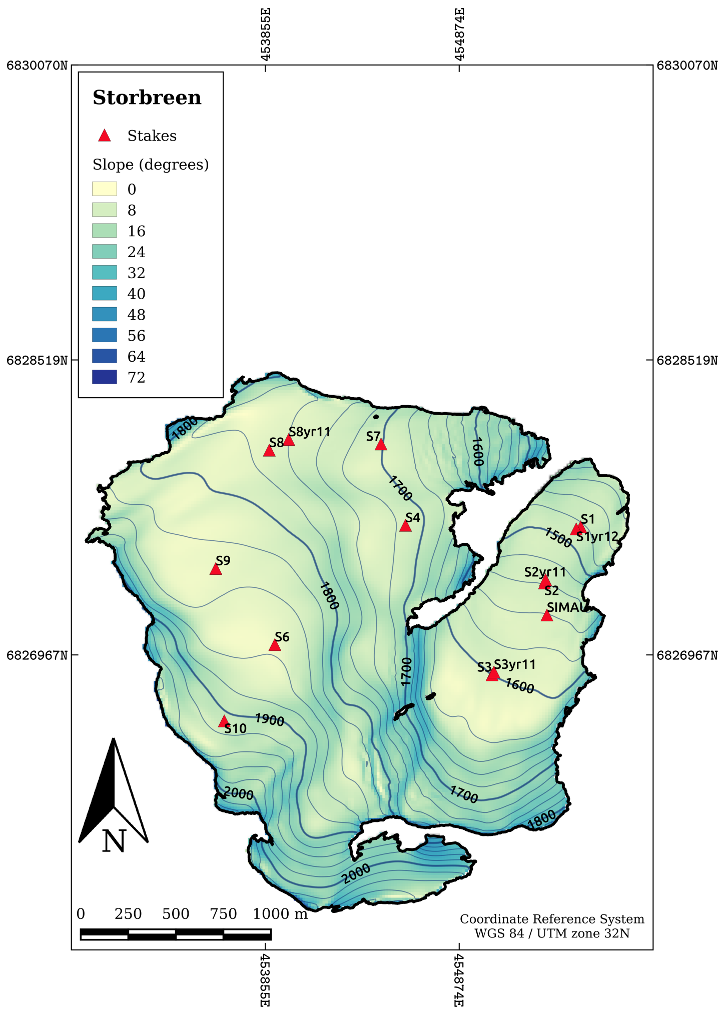
Appendix G

Surface slope at Hellstugubreen and Storbreen

(a)



(b)



Surface slope at Hellstugubreen (a) and Storbreen (b). The surface slope and the elevation contour lines are generated from the 2009 laser scanning data, and the glacier outlines are derived from orthophotos (data : NVE).