THE UNIVERSITY OF WESTERN ONTARIO DEPARTMENT OF CIVIL AND ENVIRONMENTAL ENGINEERING

Water Resources Research Report

Dynamic Feedback Coupling of Continuous Hydrologic and Socio-Economic Model Components of The Upper Thames River Basin CFCAS Project: Assessment of Water Resources Risk and Vulnerability to Changing Climatic Conditions

By:
Predrag Prodanovic
and
Slobodan P. Simonovic

Dynamic Feedback Coupling of Continuous Hydrologic and Socio-Economic Model Components of the Upper Thames River Basin

by

Predrag Prodanović and Slobodan P. Simonović

Email: {pprodano, simonovic}@uwo.ca

Faculty of Engineering Science
The University of Western Ontario
London, Ontario, Canada

February 16, 2007

Abstract

The main contribution of this work consists of formulating a novel simulation framework used in analysis of climate change impact assessment. The model developed consists of a continuous hydrologic component coupled (via feedback) to a socio-economic component developed using system dynamics. The hydrologic component of the model responds to changes in socio-economic conditions (such as changing economic, demographic and land use patterns), while socio-economic conditions are continually influenced by hydrologic quantities (such as available ground water recharge, flow and precipitation). As the two components are connected via feedback, each dynamically influences, and is influenced by, the other thereby mimicking such interactions in the real world. The combined model represents a comprehensive integrated water resources management tool developed to test climatic impact and change of both hydrologic and socio-economic conditions in the area.

The model is developed for the Upper Thames River basin, located in southwestern Ontario, Canada. The study area encompasses a number of growing urban and rural communities, with the largest community being the City of London. The entire basin is approximately 3,500 km^2, with an approximate population of 420,000 (of which 350,000 live in the City of London). Agricultural land occupies approximately 80% of basin's land area, while forest cover and urban land take up about 10% each. Hydrology of the basin is quantified with a continuous hydrologic model component, and incudes detailed modules describing snow accumulation and melt; losses; transformation of surface excess to river runoff; representation of baseflow; as well hydrologic river routing methods. The socio-economic characteristics are expressed with a component describing dynamics of urban and rural population; business and housing, as well as detailed land and water use patterns. The overall (or combined)

model couples two components, and is thus capable of testing a wide range of socio-economic policies and management strategies (like changes in demographics, housing, jobs, land and water use practices), as well as able to produce detailed hydrologic output (like frequency of floods/drought, timing and regularity of flows) typically used for impact analysis and/or engineering design.

Simulation of the model is performed for three different climate scenarios (no change, increased precipitation, and increased temperature) obtained from an external weather generator (a tool used to simulate alternate regional climate characteristics based on historical information as well as latest knowledge of global climate models). The climate scenarios are coupled with a range of socio-economic scenarios where different management strategies are explored (such as proceeding with the belief that regional water resources are infinite, implementing a strict water conservation policy, a combination of water conservation with limiting land development, as well as implementing a switch from ground to surface water use basin wide).

The main findings revealed by simulation of different scenarios include the following: (i) climate change has the potential to significantly alter flooding characteristics of the region by increasing risk levels (and its corresponding frequency of occurrence) of extreme conditions; (ii) frequency of extremes of drought conditions are likely to remain at their current levels and, (iii) the most significant regional socio-economic factor is availability of water, shown as a limiting agent to growth of population and regional economy. Recommendations are suggested to area's water resources professionals that urge them to consider revising existing management guidelines in light of knowledge of altered hydrologic and socio-economic conditions in the basin as a result of climatic change.

Keywords: System dynamics modelling, continuous hydrologic modelling, climate change impact assessment, hydrologic extremes modelling, socio-economic modelling.

Acknowledgements

Contents

List of Tables

List of Figures

1   Introduction

Despite significant progress of climate research during the past two or so decades, the field is still thriving today. Perhaps the driving factors for such intensity of research are the implications of changed climatic conditions—higher global average temperatures, changes in precipitation and land use patterns, and in some areas, increases in both dry and wet spells. These physical changes allow for the possibility that extreme events (such as floods, droughts, heat waves, snow and ice storms, etc) could occur with higher frequency in any given year.

The physical changes also imply changes in natural ecosystems and socio-economic activities. For example, changed climatic conditions could shift the sustainability of Canada's natural resources, such as water, air, land, forests, fish and wildlife. This is because these systems can not adapt as quickly as the climate is expected to change (Flannery, 2006; McBean, 2006). The implications of climate change on the socio-economic systems are also great. The threats of adequate supply of drinking water, energy and other necessary services in light of changing hydro-climatic conditions are real, and need to be addressed.

There is no shortage of publications that urge society to act in light of this knowledge; after all, the consequences of doing nothing are severe. Some of the most popular books on the subject even go to the extreme to emphasize this point. Flannery (2006), in his widely acclaimed book warns us that "if humans pursue a business-as-usual course for the first half of this century, ... the collapse of civilization due to climate change becomes inevitable"(p. 209). Kolbert (2006) does a similar thing by quoting a climate expert who warns us that changes in climate caused civilizations to collapse in the past; it is foreseeable that if we continue our present course of action, the same can happen to us. "The thing they [the civilizations that collapsed] couldn't

prepare for was the same thing that we won't prepare for, because in their case they didn't know about it and because in our case the political system can't listen to it", says her expert (Kolbert, 2006, p. 115).

Even though not specifically about climate change, another book deserves a special mention: *The Limits To Growth* by Meadows et al. (1972, 1992, 2004). Back in the early 1970's it tried to warn us of the dangers of continuing growth trends in world population, industrialization, pollution, food production and resource depletion; it too speculated collapse of global environmental and socio-economic systems in the business-as-usual scenario within the next century. In publishing the third edition of their popular book, the authors do not alter their original conclusions; they do, however, show that humanity's ecological footprint (our burden on the planet) has surpassed Earth's carrying capacity of that burden. Furthermore, one of the authors when asked how he thinks the world will act in light of this, says that:

> actions will ultimately be taken to avoid the worst possibilities for global collapse... [and] expects that the world will eventually choose a relatively sustainable future, but only after a severe global crisis force belated action. And the results secured after long delay will be much less attractive than those that could have been attained through earlier action. Many of the planets ecological treasures will be destroyed in the process; many attractive political and economic actions will be lost; there will be great and persisting inequalities, increasing militarization of society and widespread conflict (Meadows et al., 2004, p. xvi).

These are quite remarkable and powerful statements, and before we dismiss them as fear mongering, and decide to do nothing (perhaps because it is too difficult, or for economic or other reasons[1]) we should at least engage in trying to learn as much

---

[1]As an interesting side note, in April of 2006 there emerged two open letters to the Canadian Prime Minister Steven Harper regarding climate change—one urging for immediate action (McBean, 2006), while another presenting more of a sceptical viewpoint (Clark, 2006). It is also interesting that

as we can about it. After all, the greatest leverage to change is learning.

1.1 The Problem of Climate Change

The climate is one of our most complex problems, and one that is (and has been) studied extensively by many over the years. One of the least controversial conclusions within the scientific community is that Earth's climate is changing as a result of human activities (IPCC, 2001). Two activities dominate all discussions on the topic—burning of fossil fuels and changing the land use patterns.

Since the start of the industrial revolution some 150 years ago, the emissions of carbon dioxide (CO_2) into the atmosphere sky-rocketed as a result of increasing human demand for fossil energy. Even though the anthropogenic emissions are small when compared to the natural emissions from the biosphere, they are large enough to disrupt the fine balance of CO_2 in the atmosphere (Bronstert, 2004, p. 567). Further aggravating this fine balance are the increasing human demand for land and its resources.

One version of the problem of climate change is illustrated with a simple causal loop diagram[2] in Figure 1. This example is by no means exhaustive, but contains enough detail to illustrate our point. We start by assuming that in order to support the world's growing population and its desire for an ever-increasing standard of living, the level of capital in the world needs to increase (this means building more homes, hospitals, buildings, factories, power plants, etc); the more capital there is, the more

---

the former letter was signed by most of the country's senior climate experts, while the latter consisted of few junior professor signors and industry consultants standing to profit from lax government policies on climate change abatement.

[2]Causal loop diagrams are tools used for mapping a system structure with the purpose of capturing the dynamic hypothesis of the problem. They consist of a number of variables linked via feedback loops. The feedback loops are labelled as either reinforcing (R) or balancing (B), and their polarity labels are either *positive* (meaning change in one variable implies change in its corresponding variable in the *same* direction) or *negative* (change in the *opposite* direction). The feedback loops are fundamental building blocks of all systems. More on this is said in Section 2.4.
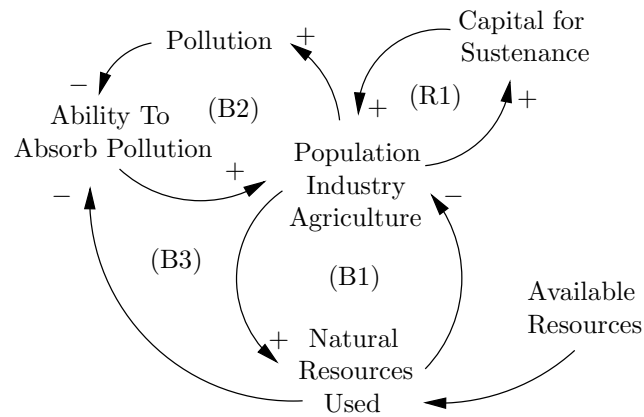
Figure 1: Causal loop of the problem of climate change

additional growth can take place (reinforcing loop R1). However, as population, industrial and agricultural activity grow, so must the use of natural resources (i.e., water, land, oil, minerals, etc); in the process forests get cleared, oil and minerals get extracted from the ground, agricultural lands become urbanized, water quality deteriorates and less of it is available, etc. But this process can not go on forever, as the Earth only has a finite amount of resources. Once people start running out of resources, the planet will not be able to support further growth (loop B1)—this is based on the assumption that our economies are based on exploitation of natural resources, which they currently are.

Additional trouble arises when we realize that we are stressing the planet with more and more pollution (more people plus more industry equals more pollution), while at the same time heavily depleting its natural resources. The higher pollution the lower is the planet's ability to absorb it. Of course, the less absorptive capacity the planet has, the less people and industry it can support (loops B2 and B3). For example, forests are one of nature's ways of regulating the carbon balance; if we keep

clearing them to make room for people, industry, and agriculture, less and less will be available to regulate the climate. Thus, if we keep doing more of what we have been doing, eventually we will not be able to sustain it.

The balancing feedback loops (i.e., loops B1-B3) are planet's self-regulating processes built-in to respond to changes (in our case, anthropogenic engines of growth). In the language of systems (Senge, 1990), the above example is a classic case of a limits to growth archetype—consisting of multiple limits (i.e., resources and pollution absorptive capacity) in combination with an engine of growth. It is a known fact that if a system like this is allowed to evolve uninterrupted, eventual collapse is *absolutely* unavoidable. Two possible strategies for dealing with a system structure like this (and thus avoiding collapse) are either to stretch out or extend the limiting factors (in our case either impossible or unlikely) or to reduce the effect of the engine of growth (possible and desired).

1.2 Climate Change Research

Since the world's climate is so complex and broad in scope, it is understandable that there would be such a wide range of research activity surrounding it. In Canada, there are a number of agencies (government and private) that are committed to producing high quality research and communicating the acquired knowledge to the public and private sectors. However, as the changed climate is expected to impose the greatest stress on the water resources sector, many of the reports have concentrated on this topic. Although six years old, the report by Bruce et al. (2000) still provides the most comprehensive top-down treatment of the subject for Canada.

Currently, one of the best ways to study the effects of climate change is to use global climate models. These models are the current state of the art in climate science,

and are the best available tools. Their aim is to describe the functioning of the world's climate system through the use of various equations from fluid mechanics, chemistry, biology as well as other sciences. More specifically, all global climate models discretise the planet and its atmosphere into a large number of three dimensional cells—these can be thought of as a large number of checker boards stacked on top of each other (Kolbert, 2006, p. 100)—to which relevant equations are applied.

In general, there are two different types of equations that are used in all global climate models—those describing fundamental governing physical laws, and those that are termed empirical (observed phenomena that are only partially understood). The former are representations of fundamental equations of motion, laws of thermodynamics, conservation of mass and energy, etc, and are well known; the latter, however, are those phenomena that are observed, but for which sound theory does not yet exist (i.e., small scale processes such as land use that can influence large scale processes such as the global climate). For most studies that are concerned with the response of small scale river basins to a changed climatic signal, the global models are inappropriate because they still have temporal and spacial scales that are incompatible with those of a river basin. One way around this is to still use the global input, but scale it appropriately for the basin in question.

Traditional[3] way of studying the impacts of climatic change in river basins involve scaling down the outputs from global climate models (temporally and spatially), and then using them as inputs to hydrologic models (see Figure 2), from which user and location specific impacts are derived. A number of studies have implemented such methodologies, and thus estimated the impacts of climatic change (Coulibaly and Dibike, 2004; Palmer et al., 2004). However, a number of uncertainties are inherent

---

[3]Some material of this section originally appeared in a introductory section in a paper by Prodanovic and Simonovic (2006b).
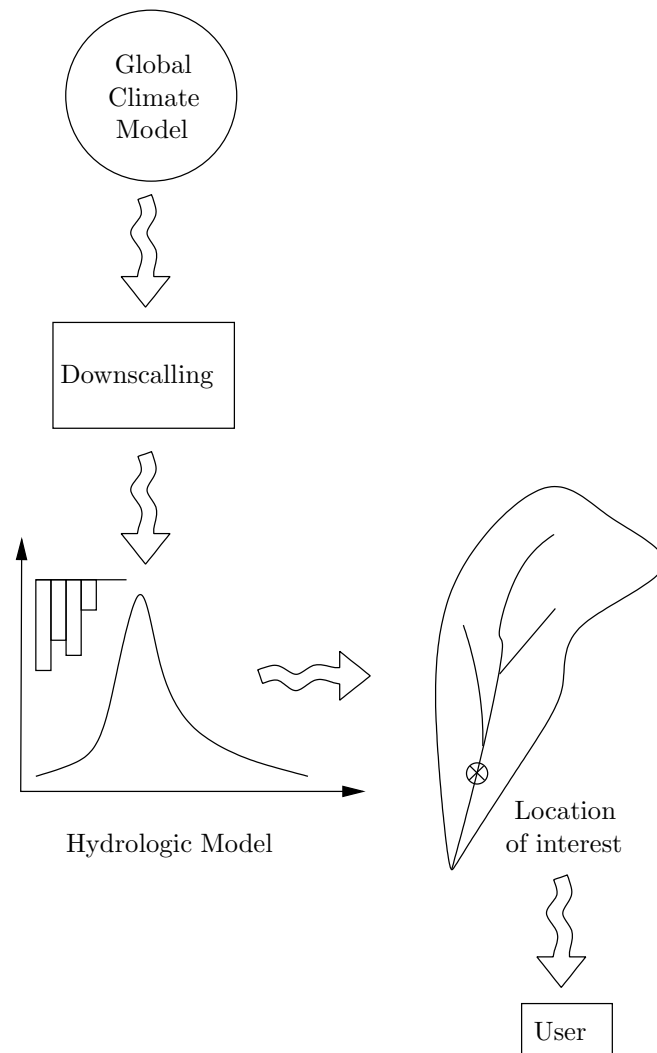
Figure 2: Traditional approach to climate change modelling

to this approach. First, the global models have temporal scales that are sometimes incompatible with temporal scales of river basins. For example, the rainfall-runoff modelling requires data with relatively short time steps (daily and/or hourly); the global models however, are only able to produce monthly outputs with a higher degree of accuracy (Cunderlik and Simonovic, 2006). This is problematic since we are often interested in changes in frequency of occurrence of short-duration high-intensity events, especially when studying the problem of flooding. Temporal downscaling of monthly global output must therefore be employed, and shorter duration events be estimated, thus compounding uncertainty. Second, spacial scales of global models can also be incompatible with spacial scales of river basins. The global models typically have resolutions between 3-5° in latitude and 5-10° in longitude, and are thus significantly larger than most river basins. Such coarse resolution is thus inadequate for the representation of many relevant smaller scale river basin phenomena.

The traditional approach is plagued by uncertainties in spacio-temporal downscaling from global to river basin scales. For example, Coulibaly and Dibike (2004) report significant differences between different downscaling methods (i.e., some downscaling techniques would produce an increase, while others a decrease in mean annual flow for the same global input). Therefore the choice of a downscaling technique can drive the outcome of the analysis, and thus mask the true system behaviour under the conditions of altered climate. Furthermore, the choice of climate models themselves (and their associated data sets) may also significantly impact the overall results. For instance, the work by Simonovic and Li (2003) shows that input data (obtained from three different climate models) is substantially different, thus explaining large variations in the final results. Consequently, the end-users (water management authorities, government policy makers and other stakeholders) became sceptical of such analysis, and have difficulties forming new guidelines as a result of them. What is needed is an

integrated water resources management framework, where the pitfalls of the classical approach are reduced as much as possible. One such framework is presented next.

1.3   Integrated Water Resources Management

This subsection provides background information on the concepts of integrated water resources management, and specifically how it applies to the Upper Thames River basin. It introduces the water resources planning and management framework from the systems point of view, generally ascribing to the position that individual system components are almost always tightly coupled, influencing, and being influenced by one another. Different system components within this framework are identified (such as climatic, hydrologic and socio-economic subsystems). The coupling of system components leads towards a systems thinking paradigm, a way of not only seeing systems and its sub components, but their interconnections as well. As water resources systems are generally of great complexity, a computer simulation framework is introduced to represent and link various system components, thus providing an approximation (or a model) of the basin under consideration. The methodology of system dynamics is introduced as a practical simulation framework able to achieve the above objectives.

General ideas relating to the concept of integrated water resources management have been developed some time ago as a result of increasing pressure on water and other natural resources by growing world population and its associated socio-economic development. The commonly accepted definition of integrated water resources management is one originally provided by the Global Water Partnership program (GWP, 2000) which states that:

Integrated water resources management is a process which promotes the coor-

dinated development and management of water, land and its related resources, in order to maximize the resultant economic and social welfare in an equitable manner without compromising the sustainability of vital ecosystems (p. 22).

One of the main advantages of integrated water resources management has been the move away from the traditional "top-down water master plan" approach, where purely technical (or engineering) measures are considered as solutions to water management problems. A growing realization of the suitability of a "bottom-up comprehensive water policy planning" approach is nowadays often made, which includes interactions among various bio-physical, socio-economic and institutional sectors for the purpose of building management capacity needed for effective administration of water resources (Loucks and van Beek, 2005, p. 26). The latter approach is becoming much more common, as it considers technical, socio-economic as well as administrative and institutional aspects of water management.

According to Loucks and van Beek (2005), today's water resources management involves the *interaction* of the following three interdependent subsystems or sectors:

1. the natural river subsystem in which physical, chemical and biological processes take place;

2. the socio-economic subsystem, which includes human activities related to the use of the natural river system;

3. the administrative and institutional subsystem of administration, legislation and regulation, where the decision and planning and management processes take place (p. 22).

Typically, projects start by identifying physical features of the water resources systems under consideration, such as meteorological and climatic characteristics, land cover and soil types, dominant land and water use patters, runoff in rivers and streams, as well as regional ground water and aquifer properties. Regional socio-economic

features are also studied, such as population and economic growth. All this is needed because the amount of runoff (and ground water recharge) depends both on physical (climate, land cover, soil types, vegetation, etc.) and socio-economic (population, economic activity, management practices, etc.) processes operating in the basin.

A framework that considers interaction of physical and socio-economic sectors is infinitely more valuable than inclusion of more detail in each such sector individually. This is because the interaction between the socio-economic and physical forces drive the overall functioning of the basin. For example, availability of clean water can place limits on economic development and growth of a region. Growth and increasing land development, if excessive, can increase timing and magnitude of flood peaks while reducing ground water recharge rates (through excessive conversion of agricultural and forested lands to residential and business uses), thus limiting the availability of clean water. The socio-economic forces are therefore inseparable from their physical counterparts, and must be included in any sound water management plan.

Water resources projects are ones that demand large financial investment for its development, operation and maintenance, and governments are the only ones who are able to provide it. In the Upper Thames River basin, water management infrastructure (consisting of three major dams as well as dykes in the cities of London and St. Marys) is estimated to have a replacement value of some 150 million dollars. As some of this infrastructure nears its useful lifespan, major replacements and/or retrofitting projects may be needed to ensure adequate protection to those affected from excessive floods and droughts. Compounding the problem of ageing water management infrastructure is climatic change, and its possible negative effects. For example, changed climatic signal is expected to alter regional hydrologic conditions (magnitude, timing and variability of extreme conditions like floods and droughts) on which all existing design standards and management guidelines are based on. Ageing water manage-

ment infrastructure and possible negative consequences of climatic change may force governments to alter practices in which design standards and management guidelines are set. These changes should be based on sound science, while using taxpayer money in a most efficient manner. This is not always easily achieved as conflicting viewpoints from different stakeholders, decision makers, and water users will almost certainly emerge and need to be taken into account. The integrated water resources management approach is one promising set of tools able to assist in such a process.

For a Canadian perspective on integrated water resources management, the reader is referred to the work of Mitchell (2005), where emphasis is placed on holistic or systemic view of water management. The systemic view of water management has recently shown great potential in coupling physical, socio-economic as well as institutional aspects of water management, thus making it ideal for addressing complex water resources management problems in the Upper Thames River basin. The strength of systems thinking applied to water resources lies in providing support for integration and interdisciplinary communication necessary in complex water resources projects. If properly implemented, it has the potential to enhance management capacity that may lead to better water resources management practice. The approach of systems thinking is introduced next.

1.3.1 Systems Thinking Approach

General systems theory, originally developed in the 1950's and 1960's, introduced a new way of thinking. In the broadest sense, system thinkers and systems theorists adopt a paradigm (or a world view) that no problem can be studied in isolation, but must be considered in the entire context it is embedded in (Phelan, 2001, p. 132). For historical development and evolution of systems thinking (and the concept of feedback in particular) from its early days to the present, the interested reader is referred to the

work of Richardson (1991). Jackson (2000) on the other hand, provides an outline of different systems approaches and how they can be applied to management in general.

The systems thinking paradigm implies that reductionist approaches—where a complex problem is divided into a number of components where each is studied on its own—are no longer sufficient. Furthermore, traditional (linear or event oriented) way of thinking that favours the belief that causes strictly precede effects is also thought to be no longer legitimate.

The systems thinking approach has always emphasized interdisciplinary, thus becoming useful in implementing principles of integrated water resources management. The aim of systems thinking is to allow interaction of disciplines that in the past would be completely disjointed (such as engineering, natural and social sciences). The interdisciplinary structure of systems thinking means that walls which once separated, are now semi-permeable membranes or meshed nets used to create open communication between system components. These can become open to influence in ways past organizational structures would not allow. For example, applying the systems thinking philosophy to integrated water resources planning and management would allow managers, policy makers, and stakeholders to consider not only physical, socio-economic, and environmental impacts of future strategies and plans, but provide novel means by which information may be shared among those that traditionally would not do so. The approach has potential for generating knowledge that may lead to better and more effective overall management of water.

In summary, the systemic way of viewing the world does not seek to eliminate or discredit classical approach to problem solving. It only wishes to complement it with tools that will enhance its capacity to deal with problems of increasing complexity in today's ever increasing dynamically complex world. In the words of Sterman (2000):

> The reductionist program [i.e., the traditional approach to water resources engineering] is very powerful, and should not be replaced by vague generalizations about systems and interconnectedness. The challenge is to design an education for ourselves and our children that preserves the power of specialized study while simultaneously teaching practical and rigorous approaches to complexity and cross-disciplinary communication—then using these systems thinking capabilities to address the pressing problems we face in our professional and personal lives (p. 901).

The systems thinking technique can, according to Simonovic and Fahmy (1999), separate policy questions from data, and help build models that are "functionally transparent to all parties involved in water policy analysis" (p. 295). As it consists of interconnecting elements, subsystems and components taken from teams with different perspectives, the systems thinking framework can generate strategies and formulate proposals that address concerns of all involved in the process. Trust in the quality of proposed strategies (as well as their actual implementation) has the potential to increase as a result of strong user participation. Therefore, there exists the possibility that smarter (and better) management decisions can be made in the future.

This is exactly what the present study attempts to do in addressing the problem of possible negative impacts of climatic change on the behaviour of the different physical and socio-economic system components in the Upper Thames River basin. Altered climatic signal has the possibility to modify (in some cases significantly) regional physical characteristics (such as magnitude, timing, and regularity of extremes such as floods and droughts). The physical characteristics may then affect socio-economic conditions that heavily rely on water (such as population, agricultural and industrial growth, health of ecosystems, species of fish and wildlife, etc). Presently, a systemic world view implementing the principles of integrated water resources management

is the only way to study impacts of climatic change on social, environmental and regional economic well being.

1.3.2 Systems Modelling and Simulation

One way to practically implement ideas of systems thinking in water resources management is to build a computer simulation model. However, before the details of any one specific simulation modelling school of thought is given, some introductory matter on generalities of computer modelling and simulation shall be presented.

A model is defined as a simplified representation of reality. Meadows et al. (1982) remind us that a model is not just any simplified representation, but one that extracts out of a complex world enough information to formulate an account of the problem to be studied (p. 7). Exactly how much information should be extracted, and how it should be represented depends on the purpose of the modelling project. For example, a hydrologic model of a river basin aiming to describe the rainfall-runoff characteristics of an area requires different kind of data and methods of representation then, say, a global climate model seeking to simulate global response to different carbon dioxide emission scenarios.

Especially since the advent of the computer, there emerged numerous applications of computer models in water resources management practice. Optimization models have typically been used in design and operation studies, while simulation models are oftentimes employed in long term planning. Determining an optimum set of reservoir releases subject to physical, environmental and social constraints represents a classic optimization problem, while studying impact of climate (and land use) changes on timing and magnitude of floods and droughts in a river basin constitutes an example of a simulation model. All further discussion in this report is restricted to simulation models, as the goal of the project is to study long term impacts of climatic change

on physical and socio-economic conditions in the Upper Thames River basin.

Computer simulation models, if properly made, can provide the modeller, the client or the user, with insight into how alternate policies or management practices can impact the modelled system over time. Computer simulation models are constructed by carefully extracting and assembling various levels of information contained in our mental models. One of the difficulties with decision making based on mental models alone is that often times we are not aware of our assumptions (they exist tacitly, operating below our level of understanding), nor capable of formulating logical implications of our assumptions. This is where computer models can help. They can include and mathematically represent a wide range of assumptions (or our mental models), as well as render precise implications of such assumptions. This implies that computer models are only as good as the assumptions that contain, no more and no less.

System dynamics methodology is a computer simulation and modelling framework based on the systems thinking paradigm. The methodology is grounded in the theory of feedback control, traditionally studied by electrical engineers. The fundamental building block of system dynamics is the feedback loop, and all its models contain multiple such loops which provide linkages between different components of the model. System dynamics modeller assembles such components individually, and then couples them together via feedback. This implies that model components have an ability to influence, and be influenced by, one another, thus facilitating communication between components that other modelling paradigms may miss. The strength of system dynamics lies in using mathematical models to formulate, test, and reformulate dynamic hypotheses of a real world problem, especially when social and physical interactions are present. The modelling can help the user gain insight into situations of dynamic complexity, and reveal causes of policy resistance. Another ma-

jor strength of system dynamics modelling is that it encourages not only modelling of physical and environmental, but also socio-economic systems. More on the system dynamics method is said in Section 2.4.

System dynamics methodology has shown promise as a methodological framework in practically implementing the idea of integrated water resources management. Its main advantage lies in the ability to integrate various processes needed for efficient and effective formulation of long term water management plans. With systems dynamics modelling, it is possible to link physical (temperature and precipitation patters, flood and drought flows, ground water recharge), environmental (water quality, ecosystem preservation and health) and socio-economic (population growth, water and land use, policy and management) aspects of river basin management. System dynamics has the capability to integrate such aspects in a single unifying framework, and thus generate understanding of the behaviour of the entire river basin system, its components and its interconnections, eventually leading to better and more effective management.

Growing pressures in both urban and rural communities within the Upper Thames River basin, coupled with the changing climate has a potential to fundamentally alter both physical and socio-economic characteristics, thus lending support for studying such problems using system dynamics. A changed climate signal may change the magnitude, frequency of occurrence, timing and regularity of extreme hydrologic exposures (such as floods and droughts), which may then force governments to review their rules, regulations, and management strategies in copying with the natural environment. In particular, the ageing water management infrastructure in the Upper Thames River basin (dams outside of St. Marys, Woodstock and London, dykes in the cities of St. Marys and London, as well as numerous sewer and drainage systems) may need to be retrofitted and/or completely replaced, as a result of its age and changing environmental conditions.

Socio-economic conditions in the region also stand to be affected from changing climatic and environmental conditions. There is tremendous amount of development occurring in parts of the basin (such as northern part of London), where agricultural land is being converted to industrial, institutional and residential uses at an alarming rate. Strong industrial and economic base is fuelling high amount of growth, and region's population is expected to continue to rise in the future. With increasing economic prosperity and larger population, the demand for water is also expected to increase. Currently, Middlesex County (of which the City of London is its largest urban centre) is supplied by both surface (extracted from Lakes Erie and Huron) and ground water, while counties of Oxford and Perth are sole users of ground water. It is important to note that Oxford County is the largest user of ground water, used to support growing manufacturing and agricultural industries.

The main objective of the study is to develop a model used for climate change impact assessment of both hydrologic and socio-economic conditions in the Upper Thames River basin. The integrated water resources model consists of hydrologic and socio-economic model components, tightly coupled via feedback—implying that each affects, and is affected by, the other. A number of scenarios (both climatic and socio-economic) are formulated and tested, and their logical consequences evaluated. Based on the insight provided by simulating the model possible changes in extreme hydrologic exposures, and well as shifting socio-economic patterns, are identified. Based on these changes, revisions of existing management practices and guidelines are suggested, in order to bridge the gap between the latest knowledge of climatic dynamics and overall basin behaviour.

1.4   Organization of the Report

The rest of this report is organized as follows: Section 2 presents in detail the inverse approach to climate change impact assessment. In particular, it outlines the steps of the inverse approach, together with giving background information on different modelling approaches used in the study. Section 3 gives a description of the Upper Thames River basin study area—located in southwestern Ontario, Canada—and includes physical as well as socio-economic characterization of the area. Simulation model components are presented next. The continuous hydrologic model component with all its modules is described in Section 4, while the socio-economic model component is described in Section 5. The coupling between the hydrologic and socio-economic model components is presented in Section 6, where descriptions of both climate and socio-economic scenarios is outlined. The simulation results and discussion is given in Section 7, and includes details of impacts of climatic change on timing, magnitude, and frequency of flood and drought flows. The model also shows outputs of a number of socio-economic variables in urban, rural, and land and water use sectors in the basin (such as population and business units; forest cover, agricultural residential and business land use; ground water recharge and use). Recommendations regarding possible revisions of existing water resources management policies and guidelines in the Upper Thames River basin are also included in this section. Study conclusions are presented in Section 8.

2 Methodology

2.1 Inverse Approach to Climate Change Impact Analysis

The inverse approach, originally developed by Cunderlik and Simonovic (2004c, 2006) takes a different route to climate change impact assessment than traditional or downscaling approach (see Section 1.2). The main strength of the inverse (or bottom up) approach is that it focuses on the end users of water resources systems, and keeps them involved in the process. The inverse approach includes the following steps (modified from Cunderlik and Simonovic, 2004c, 2006):

1. The critical hydrologic exposures (such as flood flows or droughts) leading to the failure of the water resources system under consideration are first identified, together with their risk/hazard levels. The end-users are involved in this stage, as they are most familiar with particulars of the water resource system in question. This stage may simply involve consultation with end users, and gathering information on extreme events of interest to them. In the case of flooding for example, the user may specify a particular threshold value that, when exceeded leads to an extensive flood damage. For the case of droughts, the logic is identical, except the focus is on the low flows.

2. The critical hydrologic exposures of the previous step are transformed into their corresponding critical meteorological conditions (such as maximum daily precipitation, monthly total precipitation) with an appropriate hydrologic model. Event based models are used for shorter, high-intensity flood producing storms, while continuous based hydrologic models are employed in studies where long term soil moisture balance is needed (i.e., water supply, droughts, irrigation). Relationships between critical hydrologic exposures and their corresponding

meteorological conditions for each location of interest are formulated. These are simply precipitation-discharge curves, which capture specific management practices of the end-users (such as flood/water control, environmental and river basin planning, source water protection, etc.). For studies considering flooding, a relationship between the peak flow and the total daily precipitation is constructed, while for the studies focusing on droughts monthly average precipitation-discharge relations are sought after. These relations are produced either from historic hydro-climatic data (if available), or using a simulation model where only climatic data are available. In this case, flows obtained by running a hydrologic model.

3. A weather generator is used next to simulate various meteorological conditions of present and future climates. Meteorological data is synthetically generated for an arbitrary long period that is statistically similar to the observed historical record. This data is then perturbed, in order to allow for the possibility of generating meteorological conditions not observed previously. The generated data is of high spacio-temporal resolution, as required by the hydrologic model. Altered climate scenarios are then generated that are conditioned upon the historical data (such as increased or decreased precipitation, warmer or wetter springs, etc) and are linked to the large scale global climate model outputs. This way, when new global data become available, this step can easily be repeated.

4. Using an ensemble of weather generator scenarios, frequency of occurrence curves are prepared for each generated scenario. Using a critical hydrologic exposure of interest (for example a flow causing extensive damage), the user finds the meteorological condition corresponding to that exposure using the relationship from step 2. Then, based on the frequency curves generated in this

step, the user can find the frequency of occurrence of the hydrologic event of interest for each of the formulated climate scenarios. Therefore, changes of frequency of occurrence in response to a changed climatic signal can be obtained with relative ease. Alternatively, the synthetically generated climatic signal can be also used as input to the hydrologic model, and its output analyzed. This is a preferred option when timing and regularity of flows are of interest, in addition to a flow frequency analysis.

5. The last step in the inverse approach requires the application of an integrated assessment model that captures the characteristics of both physical and socio-economic systems, and evaluates risk and vulnerability to changing climatic conditions within the study area. Different procedures are usually applied, depending whether flooding or drought studies are of interest. The overall goal of this step is to provide users with information that can be used in the formulation of new basin management guidelines.

Schematic of Figure 3 shows the steps employed in the inverse approach. The schematic also depicts the integrated assessment model (i.e., shown as an interface that combines hydrologic and socio-economic model components), that provides the user with an ability for integrated consideration of physical and socio-economic processes.

The focus of this report is on the development and description of an integrated water resources management model for use in the last step of the inverse approach. This kind of a model is absolutely necessary as the study objective is to examine dynamic impacts of alternate water resources management policies under the changing climatic signal. For example, a classical hydrologic analysis is able demonstrate what the potential impacts are due to a changed climate signal, and what management op-
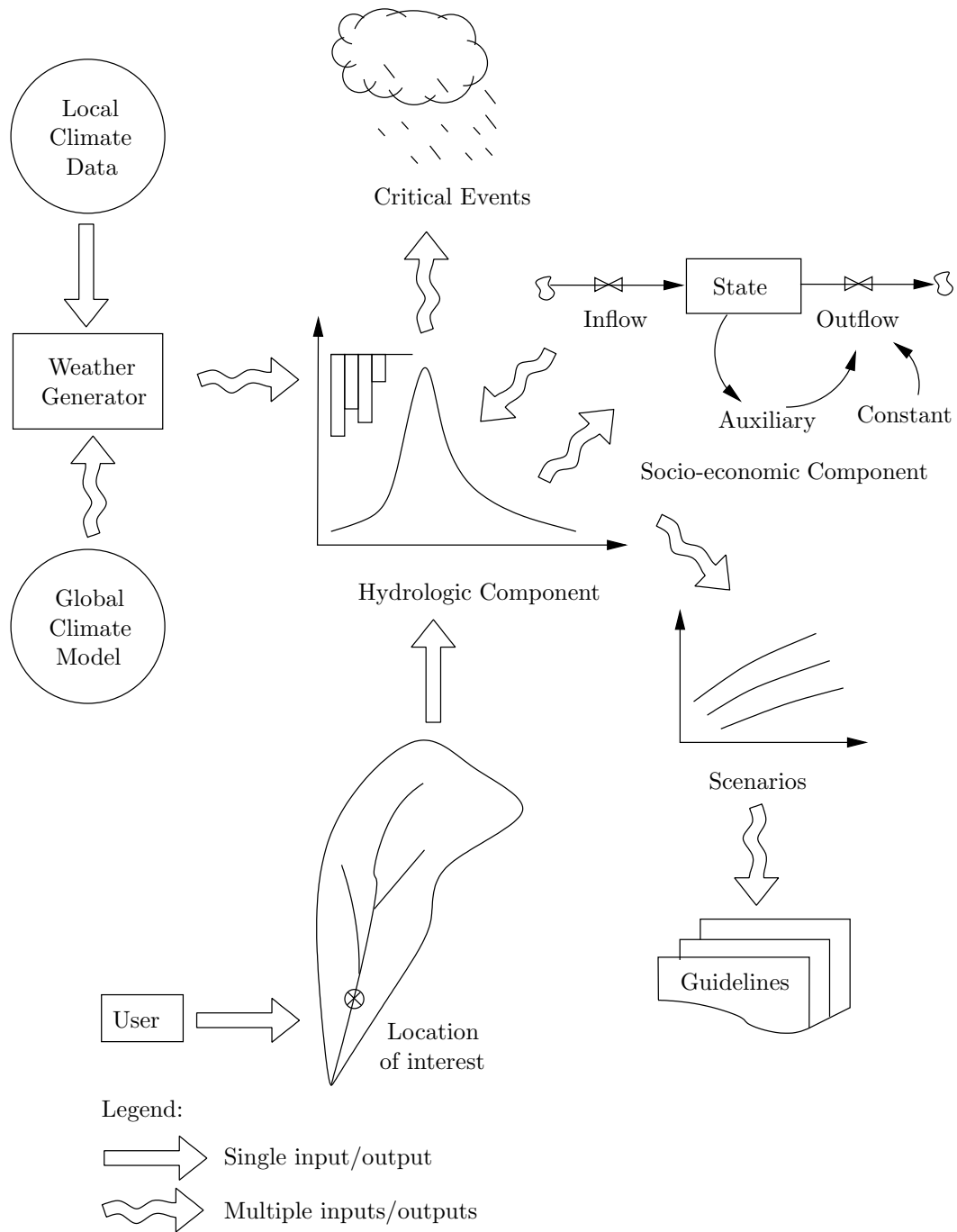
Figure 3: Overall schematic of the inverse approach

tions should be considered to alleviate possible negative consequences. However, the classical framework does not have the capability to show how a particular management option should be selected (among many available), nor show dynamic impacts over both the socio-economic and physical domains.

An integrated water resources model is needed for this, as the water resources management process involves not only processes that are physical (such as floods and droughts), but those that are social as well (such as land use planning and development, rural and urban business activity, population growth, etc.). This kind of model should be able to show the interactions between the physical and social domains, and capture dynamics of both socio-economic and physical impacts. It should also be able to evaluate the impacts of different management strategies (increasing water use, population and business growth, increasing land use and development, etc.) on the water resources system.

The management of water in the Upper Thames River basin stands to improve significantly by adopting the integrated water resources management framework. This is because the basin is currently experiencing growing pressures for development from both its urban and rural communities; it also faces increasing demand for environmental sustainability and protection, thus putting decision makers and land use planners in difficult situations. Ageing water management infrastructure (consisting of three reservoirs, an extensive dyking system in the cities of St. Marys and London, as well as numerous sewer and drainage systems) may need to be retrofitted (or in some cases completely replaced). Further compounding the problem are the possible negative consequences changed climatic conditions are expected to bring regarding magnitude, frequency, timing and regularity of hydrologic extremes (such as floods and droughts). Such changes are expected to force updating of current engineering design standards, and possibly even the revision of existing water management guidelines

and practices.

Growing socio-economic pressures such as increasing growth of population, industry and agriculture are expected to place growing strains on the environment currently sustaining them. Wetlands, floodplains, forests and other environmentally sensitive zones are expected to face growing development pressures, especially from industrial and residential uses. Growing socio-economic well being of the region is also expected to place increasing demands on the current water supplies. Even though majority of the population in the Upper Thames River basin is supplied by surface water drawn from Lakes Erie and Huron, a number of smaller communities heavily rely on ground water for residential and industrial uses. The strong manufacturing and agricultural sector in Oxford County is one of the largest ground water users in the region. Sustaining high rates of growth in this region may become difficult in the future if availability of ground water becomes scarce. This may place additional pressures to develop and expand pipelines currently in place that draw surface water from Lakes Erie and Huron.

An integrated water resources model of the Upper Thames River basin stands to highlight these and other problems the region is expected to face in the near future. By simulating the model and analyzing various climatic and socio-economic scenarios, the stakeholders, managers and users of water resources systems are provided with information which can be used to support changes in existing water management infrastructure design practices and standards, and even possible revision of current policies and management guidelines.

Such an integrated water resources model is developed in this work. The following sections describe in detail the methodologies needed to represent climatic, physical and socio-economic sectors of the model, respectively. The logic of the coupling of the model components via feedback loops is presented in Section 2.5, and forms the

major contribution of this work.

2.2 Weather Generator Modelling

The weather generator used has been originally developed by Sharif and Burn (2004, 2006a,b, 2007). A weather generator is a tool that synthetically creates climate information for an area by using both local and global weather data. The local data includes historically observed data taken from area weather stations in and around the study area, while the global data includes data obtained from the global climate models. Weather generators can be classified into two categories (see the paper of Sharif and Burn (2006a) for further details): parametric and non-parametric. The former are stochastic tools that generate weather data by assuming a probability distribution function and a large number of parameters (often site specific) for the variable of interest. The latter do not make distribution assumptions or have site specific parameters, but rely on various shuffling and sampling algorithms. A common limitation of the parametric weather generators is that they have difficulties representing persistent events such as droughts or prolonged rainfall (Sharif and Burn, 2006a, p. 181). The non-parametric version alleviate these and other drawbacks, and is thus adopted in this study.

In simple terms, the weather generator employed takes as input historical climate information for a number of weather stations in the area, as well as inputs from global climate models, and generates climatic information for an arbitrary long period of time. Sophisticated algorithms are used to shuffle the historical data, and thus generate statistically similar climate—here referred to as the historically identical, or base case scenario. Of course, the weather generator is not used solely for the replication of historical trends; it also contains various perturbation mechanisms

based on information from global climate models that force it to generate climatic information not necessarily observed in the historical record. This way, the weather generator may produce synthetic data of high spacio-temporal resolution corresponding to global climate model scenarios, and thus furnish data necessary for assessing local impacts of global climate change.

2.3   Continuous Hydrologic Modelling

Hydrologic models are mathematical representations of rainfall-runoff processes operating within a basin. Understanding the rainfall-runoff process is paramount because of its relevance to humans and their natural environment. By studying hydrologic processes via computer simulation models we get insight into behaviour of such natural phenomena as floods and droughts, and their impacts on people and their environment. This insight obtained from computer simulation models can be used as a basis for future decision making and formulation of management strategies and plans.

Continuous hydrologic models employ algorithms that describe the movement of water (or moisture) within a basin, and depict land based physical processes of the hydrologic cycle (Bennett, 1998). The continuous hydrologic model component used in this study is a semi-distributed rainfall-runoff model based on the computational engine of the widely popular HEC-HMS (USACE, 2000, 2006); its most detailed component, describing soil moisture accounting, is based on algorithms of Bennett (1998). The application of the HEC-HMS hydrologic model to this study has been originally developed by Cunderlik and Simonovic (2004b, 2005), and is used as the basis for the component representing hydrologic processes.

Continuous hydrologic models require a detailed account of long term movement of moisture within the basin, as well as an elaborate representation of moisture losses

(such as those due to evaporation and evapotranspiration). (By long term moisture movement we mean that the model simulations are done for a number of consecutive years, sometimes even decades.) A combination of methods used to describe rainfall transformation (conversion of excess rainfall into direct runoff), baseflow (portion of streamflow that comes from ground water), channel/reservoir routing (determining the shape of a flood hydrograph as it moves through a channel/reservoir), together with losses (detailed movement of water through vegetation, surface, soil and ground water) typically constitute continuous hydrologic models.

For the sake of completeness, we shall take the time and define event based hydrologic models. This category of hydrologic models simplifies the processes and relationships used to calculate the moisture balance. The losses in this category are represented by a simple function or a coefficient, as opposed to considering detailed movement of moisture as continuous hydrologic models do. Event based models, as they are capturing the basin response to short duration/high intensity rainfall, do not take into consideration the long term detailed movement of moisture, nor its associated evaporation or evapotranspiration losses (Bedient and Huber, 1988, p. 313). This is because evaporation and evapotranspiration losses are negligible during the course of an event; typical events last usually a few days at the most. The event based models are usually much easier to set up and use, as they do not require synthesis of a large number of hydrologic processes. Continuous based models on the other hand, are much more rich in number of captured processes than their event counterparts, and are best suited for studies where long term basin response is of interest (such as those of droughts). Even though there have been examples of using continuous based models for studies of flooding (Soong et al., 2005), they are still relatively rare.

2.4   System Dynamics Modelling

System dynamics simulation is a methodological framework closely based on ideas of systems thinking (see Section 1.3.1). In fact, the systems thinking approach is a prerequisite to formulation of sound system dynamics simulation models. Originating from development of general systems theory (and its paradigm of systems thinking), system dynamics is a school of thought that uses feedback control theory to build computer simulation models. Its fundamental building block is the feedback loop, and since the feedback loop is found in all complex systems, system dynamics has been used to study problems of all sorts. Some of these include physically based hydrologic processes (Li and Simonovic, 2002; Sehlke and Jacobson, 2005), organizational behaviour and management (Forrester, 1961; Richardson, 1991; Senge, 1990; Sterman, 2000), global change modelling (Meadows et al., 2004, 1982; Simonovic, 2002).

Recent times have seen system dynamics applied to a variety of water resources management problems. Matthias and Frederick (1994) have used system dynamics to model sea-level rise in coastal areas; Simonovic et al. (1997) and Simonovic and Fahmy (1999) applied it to modelling of long term planning and policy analysis in the Nile River Basin in Egypt; Ahmad and Simonovic (2000) used it in simulations of reservoir operation for flood control, while Simonovic and Li (2003) have used to evaluate impacts of climate change on a large scale flood protection system for the city of Winnipeg, Canada. Winz (2005) applies system dynamics for the purposes of studying availability, quality and distributions of water in urban areas of Auckland, New Zeland.

System dynamics modelling framework is different than other modelling constructs in the sense that its central point of view states that its model's feedback structure

(how feedback loops are arranged and linked together) is the source of its overall behaviour. This implies that different model structures cause different system behaviour. The behaviour we are referring to here is dynamic in nature; this means that we are interesting in studying how systems (or their components) change and evolve over time. Again, the model behaviour is determined by its feedback structure.

System dynamics is different than other computer simulation methodologies as it is equally capable of describing physical and social processes. For example, hydrologic models (see Section 2.3) only study the land based rainfall-runoff processes operating within a basin, but do not take into account social, economic, political or other equally relevant processes that affect the hydrologic cycle. It is known that socio-economic processes are absolutely critical in the management of water and other resources; this is because important management decisions are rarely based on purely physical (or technical) considerations alone.

The system dynamics methodology was the first computer simulation framework to emphasize the feedback structure of social systems, as well as its tight linkages to physical systems. Prior to the development of system dynamics, the explicit linkages between social and physical domains (although acknowledged as existent, significant, and sometimes even necessary) were not considered because an appropriate theory for such interaction was simply not available. Even today, some half a century after its initial introduction, system dynamics studies receive much unfounded criticism, mostly from people stuck in the traditional world view where compartmentalization, divide and conquer, and top-down organizational management strategies and attitudes are still prevalent.

System dynamics modelling, because it equally well applies to the study of both physical and social systems, is showing promise in being able to discover and generate knowledge that map ways the physical world interacts with its social realm. In the

study of management of water resources systems, this is extremely important, as the professional practice has much to gain from applying system dynamics to its complex water resources problems. Even thought various modelling efforts are nowadays used in the practice of water management for the purpose discovering better management decisions, very few of such efforts consider how the water resources systems studied affect, and are affected by, the social and environmental systems they are embedded in. For example, those studying flood, drought, water quality and sedimentation rarely build models where the impact of the water resources systems are felt in social and environmental systems, which can eventually influence the functioning of the water resources systems themeselves.

Perhaps one reason this is not done is because many of today models used in the water resources management practice are based on representation of very detailed physical processes, and are therefore very data intensive. This means that large amount of data must be gathered to adequately calibrate and verify the models in order to build confidence in the produced outputs. This has lead the water resources practice towards efforts of gathering data of increasing spacial and temporal resolution, but not necessarily better management decisions. The reason for this is that data (either directly measured or obtained by simulating a computer model) by itself is not enough to make decisions. What is needed is a way to distill information out of data, or as Beer (1975) puts it:

> I am fighting for way through to your ears. That is exactly what I am trying to differentiate, in you, between data and information. Data are a whole lot of meaningful patterns. We can generate data indefinitely; we can exchange data forever; we can store data, retrieve data, and file them away. All this is great fun: maybe useful, maybe lucrative. But we have to ask why. *The purpose is regulation.* And that means translating data into information. Information is what changes us. My purpose is to effect change, to impart information, not

data (p. 223, emphasis in original).

Data is important is modelling efforts of any kind—in fact, models could never be built if data was not available to initialize and parametrize them. The point we wish to emphasize is that data by itself is rarely useful, as it can not lead directly to better management decisions. If we consider the social realm complex water resources systems are embedded in, the availability of data is extremely scarce (time series data of population, economic activity and land cover are available on much larger time scales than hydrologic data). Some would use this as an argument against including social components to water resources models, despite knowing that social processes heavily influence the physical ones. The position adopted in this study is that all available data should be used, while in cases where data is not available, reasonable assumptions should be made that can easily be criticized and/or changed once relevant data becomes available.

The system dynamics approach can help bring about a new world view because it adopts a position that embraces finding and mapping relationships that link components of complex dynamic systems utilizing all available data. It also emphasizes long term dynamic continuous behaviour. This is entirely different from the static world view traditionally accepted which approximates dynamic change through a collection of snapshots, while all together neglecting ways of uncovering relationships that are the source of dynamics and change. The continuous view point means that the methodology is interested in variables responsible for gradual adjustments of processes and mental models that shape decisions, which through action bring about changes in future states of the world, thus leading to new decisions (Forrester, 1961).

2.4.1 Definition of Feedback

In order to fully understand the principles of system dynamics, we first define one of its building blocks, the feedback loop. Such a loop is commonly defined as a process where the system's output is related to its input in order to further regulate its output. A state of the system ordinarily provides information cues about a needed action, which then continuously acts and changes the system's state. As an example of a feedback consider the process of soil infiltration common in hydrologic analysis. The moisture content of the soil (i.e., its system state) communicates and provides information cues to the process of infiltration (i.e., its system action) by determining how much water can infiltrate, thus directly affecting the original storage of water in the soil. It is important to note that when we refer to action in this context, we emphasize that it represents a process operating continuously. This is contrasted with the every day meaning of the word, emphasizing an event oriented definition of the term.

There exist two types of feedback loops: positive and negative. The former are loops that are self reinforcing, meaning than an initial change is amplified in the same direction as all components are traced through the loop. For example, more people in an area implies that more development will need to take place, as people need adequate housing (houses, apartments), institutions (hospitals, schools, police and fire departments, etc.) and businesses (to provide jobs and a healthy regional economy). Therefore, as more development takes place, more people will be needed to maintain the current standard of living and ensure future growth, thus leading to higher levels of population.

The latter type of feedback loops are those that are balancing, or those that counteract and oppose initial change as its components are traced through the loop. This

means that an initial change in a variable eventually produces a result of the opposite sense in the same variable after all are traced through the loop. For example, as a neighbourhood is becoming more attractive (as in above example of development), more people will wish to move there. A high influx of people at any time can create high housing prices, crowding in schools and hospitals, and create traffic conditions that are intolerable. Development in the close vicinity to the flood plain may increase costs due to flood damage. Over a period of time, these conditions can make the area less attractive than initially thought (Sterman, 2000, p. 12), thus causing a decline of in migration. Such is an example of a self-regulating feature of social systems.

2.4.2   Definition of Delays

Another characteristic of system dynamics that must be mentioned is a delay. Delays are critical to dynamics of systems because they can "breed danger by creating instability and oscillation" (Sterman, 2000, p. 409). Furthermore, delays can help filter noise from variability and thus provide clearer signals upon which better management decisions can be made. Delays are found everywhere, and are inseparable from feedback loops—this is because the feedback loops could not exist if it wasn't for delays. In the definition of feedback we stated that a system state provides information cues to a decision, which then continuously alters the system state, eventually leading to adjustment of original decisions. The process by which information cues are generated include delays, as it always takes time to measure, transfer and report information about a system state. Despite the availability of information, decisions may not always be taken immediately upon receiving the information, but only after some time therefore creating more delays. This reporting, measuring and transfer of information, together with the process of carrying out the decisions, are referred to as delays.

In the system dynamics literature, there exist two types of delays: material and information. The former refer to processes that capture delays in physical or material flows. In this case, the inflow to a process (whether it be flow of water to a reservoir, or the influx of applications seeking water taking permits for example) is temporarily stored (in a reservoir, or on a desk of a government official) and remains in transit for some time. After a specified period (or a time delay), the quantities (reservoir release, or a permit evaluation rate) are released at a specified rate as output. Material delays emphasize physical flows, and include any quantities that have physical units. They obey conservation laws, meaning that the same amount flows out, as it originally flowed in.

The latter type of delays do not include any physical flows, but rather represent "gradual adjustment of perceptions or beliefs", and thus correspond to mental (or social) processes (Sterman, 2000, p. 412). An example of an information delay is a change between say, a normal inflow rate to a reservoir, and an operator's belief about the likely future inflow rate. Suppose that a large flood creates an inflow rate to a reservoir greater than the normal rate, and that such floods increase in magnitude and frequency as a result of climatic change. In this case, the operators are unlikely to immediately adjust their practice and instantly accept the higher inflows that may be the consequence of changing climate. If the floods of higher magnitudes persist, the expectations and beliefs of operators will also adjust to expect more such floods, but only after some time. The delay between the initial signal of change, and the time the operators realize climate change is bringing higher flood flows, represent information delays. Furthermore, as Sterman (2000) puts it:

> All beliefs, expectations, forecasts, and projections are based on information
> available to the decision maker at the time, which means information about the
> past. It takes time to gather the information needed to form judgements, and

people don't change their minds immediately on the receipt of new information. Reflection and deliberation often take considerable time. We often need still more time to adjust emotionally to a new situation before our beliefs and behaviour can change (p. 426).

Material and information delays constitute an important set of tools for systems analysis, as it provides us with better ways of representing both, the physical, and the mental processes that form the basis of all human decision making.

2.4.3 Elements of System Dynamics

All system dynamics model consists of only four basic elements, which are introduced in this section. Figure 4 shows the graphical representation of the four elements, together with their meaning. Accumulations in system dynamics models are represented with boxed variables, and are referred as levels, stocks and/or state variables. Level variables are used to hold any quantity that can accumulate—a storage of water in a reservoir may be represented by such a variable, as can the amount of knowledge in a person's head (despite the fact that it can't easily be measured). An advantage of the system dynamics method is that level variables can represent physical (such as volumes of water) as well as immaterial states such as knowledge, perceptions and beliefs.

The only way to change state variable is through their corresponding rates (or flow variables). This means that the only way to change the volume of a reservoir is by altering either its inflow, outflow, or regulating both. In the same sense, studying and learning can increase the knowledge content in a person's mind, while a rate of forgetting can decrease it. Rates are depicted by arrows with valves in the centre, to denote their responsibility in regulating level variables. Clouds are also part of the rate variables and to refer to inputs coming from the outside of the defined model

boundary. The units of rate variables are those of its corresponding level variables divided by time. This way, if the contents of reservoir volume is expressed in units of volume [m^3], the inflow and outflow rates must be represented in units of volume over time [m^3/s].

It should be noted that level and rate variables are the only variables necessary for building system dynamics simulation models. However, it often proves helpful to add additional variables (called auxiliary variables) as intermediate steps to keep track of other relevant information. For example, the gates of a reservoir may be operated in such a way that they depend on the rate of rise of water in the reservoir, which can be determined by converting the storage volume to elevation with an appropriate function (see Figure 5). Water elevation can then be used to determine the reservoir release; it therefore represents an auxiliary variable.

Lastly, single arrows connecting levels, rates, and auxiliary variables simply carry information from one to the other thus facilitating management of information. In the above example, the information from the reservoir level is carried (by the arrow) to the auxiliary variable representing reservoir water elevation, which, together with gate operation procedures determine the outflow.

2.5   Integrated Water Resources Modelling

The aim of this section to introduce a new modelling framework that dynamically integrates and links a number of components used in modelling of water resources. This approach extends the traditional way of water resources modelling, as it does not only include models of physical but also models of socio-economic processes. In other words, it presents the ideas of feedback, systems thinking and system dynamics, and how they can be implemented in integrated water resources modelling. One of
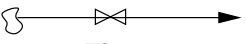
Symbol	Meaning

| Reserve | Level (or state variable), used to represent accumulation |

| Flow | Rate (or flow variable), used to represent change per unit time of the state variable |

| | Arrow, simply carries information from one variable to another |

| Auxiliary Variable | Variable, used to store information |

Figure 4: Elements of the system dynamics method

Storage [m^3]

Inflow [m^3/s]

Outflow [m^3/s]

Water Elevation [m]

Storage Elevation Function [m^3 vs. m]
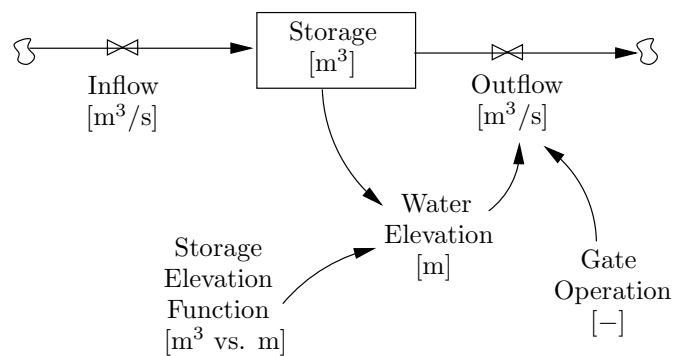
Gate Operation [−]

Figure 5: Reservoir operation via level and rate diagram

the goals of this report is to show that the system dynamics method can be used in coupling physically based hydrologic model component to its socio-economic counterpart. In current practice however, hydrologic modelling used in water resources management is entirely decoupled from the socio-economic framework it is embedded in. This is not to say that socio-economic characteristics are not (and were not) examined; they have (and are), but most have been decoupled from one another. In order for us to illustrate this, consider the following.

Typically water resources management studies consider a large number of socio-economic processes functioning in the basin. Such processes normally considered are population growth, urbanization, water use rates, or management processes such as upgrading or building new infrastructure, putting in place new educational programs, etc. The modelling framework of such studies typically include formulating a hydrologic (or water balance) models. These models, based on a conceptual representation of physical processes, estimate streamflow in rivers, storage in reservoirs, evaporation and ground water recharge rates, among other things. It should be pointed out that hydrologic models employed include the latest knowledge of physical processes, and represent the state of the art of the field of water resources modelling.

After hydrologic models are completed and hydrologic quantities estimated (usually water availability), they are compared against estimated water demands produced by socio-economic studies. These studies create and build models of their own to estimate population growth rates, industrial production, jobs, urbanization and development, among others. These models, upon which most urban master plans are based on, normally use social, political and administrative constructs of current structures of government as basis for its decision making regarding development and land use, social, economic and environmental prosperity, etc. The time horizon of such models is typically a few decades at most. It should be said that these socio-economic models

also use the latest methods and techniques of their field, and represent the state of the art.

In current water resources management practice, hydrologic and socio-economic models (or their results) are linked in the following way. Scenarios are used in both types of studies (where alternate hydro-climatic characteristics are assumed together with population and growth projections), and are compared against each other. For example, a changed climate scenario may produce a higher frequency of drought, and thus imply reduced availability of water. A socio-economic scenario projecting intensive growth in the region uses this information to draw conclusions and recommendations that suggest, say, guidelines for future water use. These conclusions and recommendations then form the foundation for future policies and management strategies.

Many in the current water resources management practice would see nothing wrong with a such an approach, as many indeed follow and practice it on a daily basis. But such an approach misses some of the key elements that is part of today's complex systems. In essence, it assumes that (from Simonovic and Davies, 2006):

1. We can predict the character of all interactions between the two systems, despite their [individual] nonlinear nature.

2. The interconnections between the systems do not fundamentally determine the future state of the [region]. In other words, the interactions between these systems are largely irrelevant to the behaviour of each.

3. The biophysical and socio-economic systems are essentially separate, so that feedbacks between the systems are external to both.

4. We can separate human effects ... from natural effects (p. 432).

The flow of information in the classical water resources management modelling is not of the kind that can help us better understand the functioning of complex water

resources systems, despite exchanges of information that take place when scenarios are compared. The fundamental building block of systems, the feedback loop, is not evident in the classical studies when physical and socio-economic links are considered; neither is the systemic framework of problem solving (see Section 1.3.1). Both of these are absolutely necessary for better understanding of the functioning of socio-economic processes and the physical world they are embedded in.

Hydrologic and socio-economic systems can not be studied in isolation, as both depend on, and are dependent on, one another; moreover, they continuously influence and interact with one another through a number of processes. Such influence and interaction is missing from traditional water resources management studies. As a result, knowledge and information is rarely generated that challenges ways in which the organizational structure of our current water resources management is set and operating. Often times defensive routines exist that protect the current status quo; in classical studies they are never brought to the surface, where they can be examined, questioned, or otherwise challenged. Reflection of the current organizational structures and management practices seldom take place, and experimentation that can bring change is rarely encouraged. The interested reader is referred to Argyris et al. (1985), where more information is provided on the context of how to initiate such a process.

Systemic way of thinking, coupled with system dynamics simulation modelling offers an innovative framework for integrated water resources management. This framework is believed to be particularly useful in studying impacts of climate change on water resources systems and their management. Integrated water resources modelling implemented using system dynamics has an ability to capture the complexity of both physical (hydrologic, climatic) and socio-economic (population, land and water use) systems, as well as their interactions. The flexibility of using scenarios,

together with an ability to check sensitivity of different management strategies, give the methodology an added practical benefit (Simonovic and Li, 2003, p. 365). The modelling framework can also help uncover key leverage points, where small changes (often found in unexpected places) can bring substantive positive and encouraging effects.

Progress by practitioners in this direction has indeed been slow. One encouraging fact is that the importance of interconnections between physical (or environmental) and socio-economic factors is being acknowledged, as evidenced by the following quotation from the Planning Policy Section of the Ontario's Ministry of Municipal Affairs and Housing:

> It is increasingly evident that environmental quality, once viewed in isolation, is inextricably linked with economic growth and prosperity. How and where future development occurs will have a direct impact on environmental quality in the study area, which in turn will directly affect the study area's economic potential (MMAH, 2002, p. ii).

Despite this realization, the report by the Policy Planning Section (which is some 320 pages long, including the appendices) provides no concrete means by which these feedback links could be formulated, tested, or otherwise examined. Systemic structure of the socio-economic systems is never explicitly defined—not with feedback loops or other means. Ample verbal descriptions and associated statistical information provide the basis for a description of such a systemic structure, despite the fact that it is never explicitly formulated or proposed in the above quoted document. This means that the systemic structure can never be questioned, nor its defencive routines (responsible for maintaining the status quo) exposed. The report does, however, provide ample socio-economic data containing "snapshots capturing information on demographics, local economic clusters and niche markets, land uses and the environ-

ment" (MMAH, 2002, p. 1). This simply illustrates the trend of classical studies in placing value of increasing data resolution of individual components, while altogether neglecting dynamic feedback interrelationships between different components. Flows of material and information linking socio-economic to environmental characteristics (which produces overall system behaviour) are never mentioned.

A computer simulation modelling framework is necessary that is able to capture such interactions, from which not only system behaviour can be deduced, but deeper understanding gained of how various policies and management strategies affect the entire system. Hidden assumption can be brought to surface, and thus properly examined. Evidence exists that shows people who use the systemic way of thinking coupled with the concept of feedback and computer simulation models, experience genuine learning not able to be produced otherwise (Senge, 1990; Senge et al., 1994; Sterman, 2000).

An integrated water resources model is developed in this work, and applied to the Upper Thames River basin for the purpose of examining long term regional risk and vulnerability of water resources systems to possible impacts of climatic change. Of particular emphasis are implications of changed hydro-climatic conditions, and how they might influence future engineering design standards and management guidelines. The model presented in the following sections consists of a continuous hydrologic component, coupled (via numerous feedback links) to a socio-economic component developed using system dynamics. In order to facilitate better understanding of the combined model (and its associated interconnections between the hydrologic and socio-economic processes), each individual component is described in detail. Following this, the feedback coupling of the hydrologic and socio-economic model components is presented, and consists of the major contribution of this research.

3 Upper Thames River Basin Case Study

The modelling approach described in Section 2 is applied to the Upper Thames River basin, located in southwestern Ontario, Canada. The relative size and location of the basin within the Province of Ontario are shown in Figure 6. This section provides a description of the study area (it outlines its physical and socio-economic characteristics) and presents the necessary data needed to practically implement the approach outlined in Section 2.

The integrated water resources management model developed in this study aims to address consequences altered climatic signal may have on the region's water resources. The emphasis is placed on changes in magnitude, frequency, timing and regularity of hydrologic extremes such as floods and droughts, as well as socio-economic impacts in demographic, job, labour, and land and water use sectors. Recommendations for possible revisions of basin's water management practices are presented in order to respond to climate change and its associated variability.

3.1 Physical Characteristics

The Thames River had formed after the retreat of the Wisconsinian Glacier from Ontario during our most recent ice age ending some 10,000 years ago. In fact, some of the northern branches still flow through the ancient spillways, while the southern reaches, with their lower gradients emerged after being a large glacial lake (Wilcox et al., 1998).

Originally called Askunessippi (the antlered river) by its original Algonquin and Iroquois inhabitants, the Thames River got its current name in 1793 after Lieutenant Governor John Graves Simcoe renamed it after the river of the same name in England. The renaming occurred after Simcoe proposed that a new capital of Upper
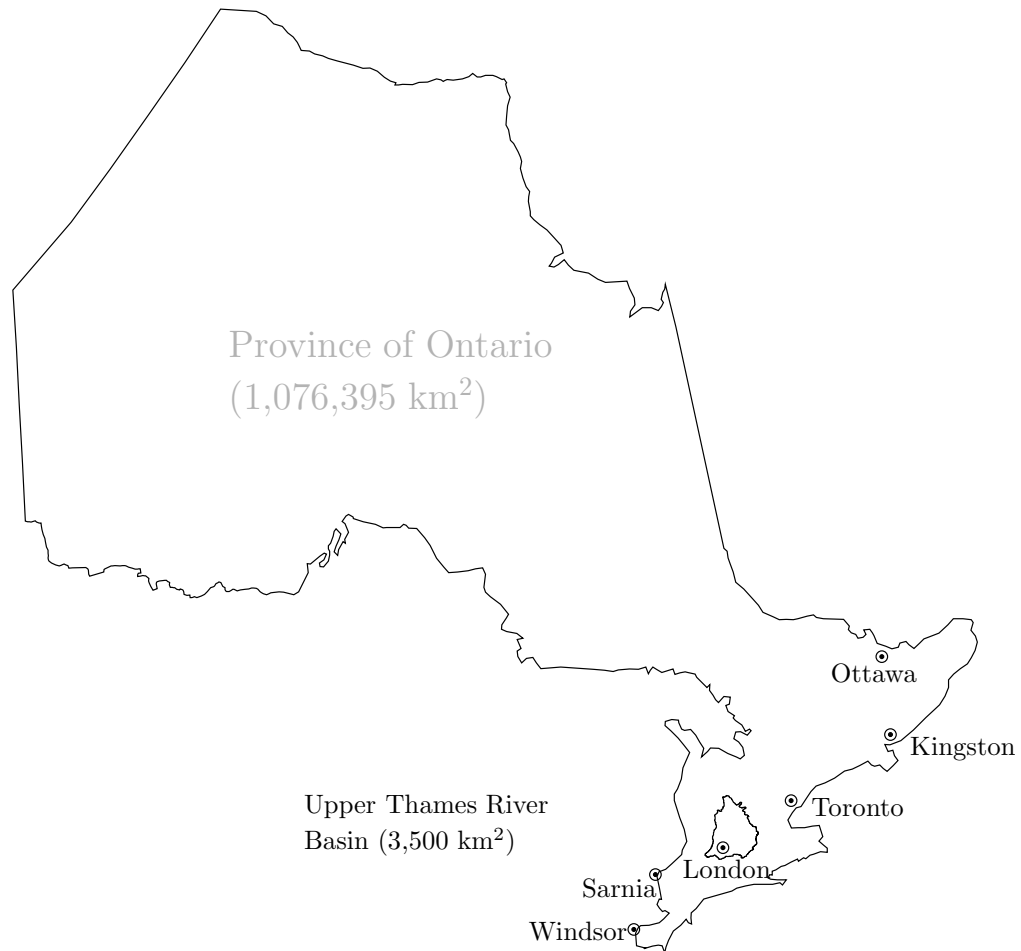
Figure 6: Location of the Upper Thames River basin within Ontario

Canada be called London, to be located between Lakes Huron and Erie at the forks of the Askunessippi River (in place of the original Algonquin village Kotequogong). Interestingly, the proposal for the capital was rejected in favour of Toronto, but the new names for the river and the city were kept.

A large part of the Thames river basin was originally part of deciduous Carolinian forests, most of which is now cleared to permit farming, agriculture and urban developments. Despite this however, the Thames River is still one of the country's richest basins in natural and human heritage. The basin is home to some 2,200 species of plants, including the Wood Poppy (found only in two locations in Canada). Its biodiversity consists of a large and diverse populations of clams, fish, and other wildlife. Most notably, the Eastern Spiny Soft Shell Turtle and the Queen Snake (now registered as endangered species in Ontario) call the Thames River their home (Wilcox et al., 1998).

The Thames river basin consists of two majors tributaries (see Figure 7) of the river Thames: the North Branch, flowing southward through Mitchell, St. Marys, and eventually into London, and the East Branch, flowing through Woodstock, Ingersoll, and east London. The two branches meet in London near the centre of the city, referred to as the Forks. The Thames then flows westwards and exits the basin near Byron, eventually draining into Lake St. Clair. The length of the Thames River (from Tavistock to its mouth at Lake St. Clair) is approximately 273 km, while its annual discharge (measured at the Byron stream gauge) is about 35.9 m^3/s. The Upper Thames River basin receives about 1,000 mm of annual precipitation, 60% of which is lost through evaporation and/or evapotranspiration, stored in ponds and wetlands, or recharged as ground water (after Wilcox et al., 1998, p. 15). The slope of the Thames River is about 1.9 m/km for most of its upper reaches, while its lower reaches are much flatter with a slope of less than 0.2 m/km.
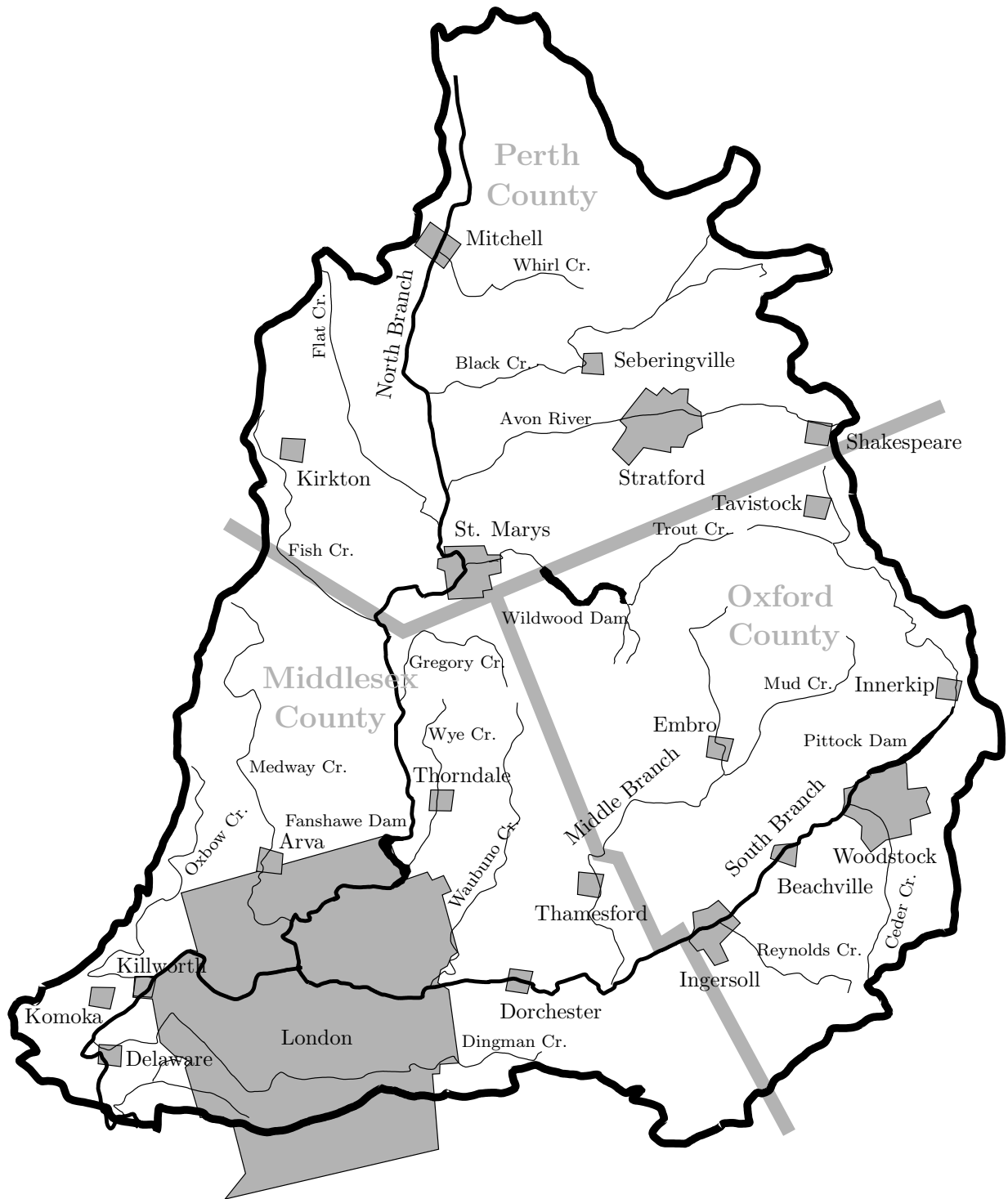
Figure 7: Map of the Upper Thames River basin

The Upper Thames River basin includes three major water management reservoirs: Wildwood (completed in 1965), Pittock (completed in 1965) and Fanshawe (completed in 1952) near St. Marys, Woodstock and London, respectively. The primary goal of all reservoirs has been flood management and protection. Since then however, the reservoirs have seen their purposes expand to low flow augmentation, as well as recreational uses. Other major flood management infrastructure in the Upper Thames River basin includes extensive dyking systems in the cities of St. Marys and London, as well as a diversion channel in the town of Ingersoll.

Floods and droughts represent major hydrologic hazards in the Upper Thames River basin. On average, 25% of all floods occur during March, while more than 50% of all floods happen between February and April (Cunderlik and Simonovic, 2004a). Early springs therefore see basin wide floods resulting from sudden warming and snowmelt on yearly basis. Times betwees December and April may also see severe floods resulting from a combination effect of snowmelt and intensive precipitation. Summer frontal storms are also known to produce severse flooding in and around the basin, but such storms are less frequent—even though they can exceed in magnitude (sometimes significantly) spring time flooding. In fact, some of the highest observed floods have occuring during the summer months.

Drought conditions are possible at any time of the year, although they are most frequent between the months of June and September. In the summer months the mean monthly flows are to the order of 25 to 30% of long term annual mean flow; as a reference, mean monthly flows in spring time are as large as 300% of the mean annual value (Cunderlik and Simonovic, 2004a). During the summer months a large percentage of baseflow in area rivers (located in the vicinity of sewage treatment plants) consists of treated effluent. As the Thames River is defined by Ontario's Ministry of Environment as a watercourse with total phosphorous concentration below

the minimum provincial standard, water quality during the course of low flow may be problematic.

3.2   Socio-Economic Characteristics

The human influences in the basin date back to 500 A.D., as the area was thought to have been inhibited by Aboriginal peoples, who were one of the first to practice agriculture in Canada. Subsequently, the area attracted a multitude of French fur traders and European settlers, as abundant fish and game were present. Since then however, agriculture and modern industry have overtaken once vast forested lands. (The reports by Wilcox et al. (1998) and UTRCA (2001) outline these, and many other interesting historical facts about the basin.)

Today, majority of the river basin is covered with agricultural lands (80%), with forest cover and urban uses taking about 10% each (basin's land area is approximately 3,500 km^2). The population of the basin is about 420,000, of which 350,000 are residents of the City of London, the largest urban center in the basin (see Figure 7). Other urban centers are those of Mitchell, St. Marys, Stratford, Ingersoll and Woodstock. The region is divided into three counties: Perth to the north, Oxford to the east, and Middlesex to the west.

Land use characteristics of the basin are shown in Table 1, and is based on information provided in UTRCA (2001). It is noted that data in the table represents only that portion of each county that lies in the river basin; in other words, the land area of each county located outside the basin's boundary is not shown here or otherwise considered.

The strength of the regional economy in and around the Upper Thames River basin is its diversification (MMAH, 2002). This is because the basin is located within

Table 1: Land cover in the Upper Thames River basin

County	Urban/Ind (km^2)	Agricultural (km^2)	Forest (km^2)	Total[†] (km^2)
Middlesex	204.4	760.6	152.0	1,117.0
Oxford	71.6	914.5	140.9	1,127.0
Perth	71.4	1,008.9	120.5	1,200.8
Totals[†]	347.3	2,684.1	413.4	3,444.8

[†] Numbers may not add up due to rounding.

the Highway 401 corridor (a major international land based transportation link), thus giving it easy access to many Canadian and US cities. The makeup of the regional economy consists of strong manufacturing, education, health care and agricultural sectors.

The regional economy depends on a number of manufacturing plants (particularly around London, Ingersoll and Woodstock). Educational sector also provides strong influence over the regional economic activity as it contains first class, internationally recognized, educational, medical and research institutions. The agricultural sector is classified as diverse, as it supports a variety of activities from specialized crops to major livestock operations. The counties in the study area provide some of the highest agricultural sales and productivity figures among all counties in the province (MMAH, 2002, p. 2). Although not as strong as other economic sectors, tourism forms a significant generator of economic activity, particularly in Stratford and other smaller towns, where parks, conservation areas and other cultural attractions draw visitors from within and abroad.

Table 2 provides additional information regarding socio-economic characteristics of the basin, and includes population, dwellings/farms, labour force and jobs for both urban and rural regions. The information in this table is based on data from Statistics Canada and its 2001 Community Profile Census (STATCAN, 2001). Water

use information on the other hand, is provided in Table 3 for both urban and rural sectors, and is divided into uses of surface and ground water. This information is taken from county ground water studies (Brown, 2001; Kell, 2004; Merry, 2003). As with land use, the socio-economic and water use characteristics are shown only for that part of each county that is within the basin. Both the 2001 census, and the county ground water reports provide information on a county wide basis, and thus represent a different physical boundary than our study area. The county based information is thus adjusted (based on land area, location of major urban centres, as well as population density) in order to produce data corresponding to the boundary of the basin.

It is interesting to note that detailed characteristics of water use data for Oxford County (the largest consumer of water in the area) is only available county wide. Details regarding consumption of water on smaller scale is not available to the public, and therefore was not used in the study.

Table 2: Demographics of the Upper Thames River basin

Urban Sector

County	Population (people)	Dwellings (no.)	Labour Force (people)	Jobs (people)
Middlesex	336,539	137,760	185,208	172,628
Oxford	44,038	17,395	31,642	29,671
Perth	35,969	14,640	27,040	25,970
Totals	416,546	169,795	243,890	228,269

Rural Sector

County	Population (people)	Farms (no.)	Labour Force (people)	Jobs (people)
Middlesex	21,410	889	2,915	2,752
Oxford	21,676	1,163	3,060	2,915
Perth	16,718	1,371	2,282	2,218
Totals	59,804	3,423	8,257	7,885

Table 3: Water use in the Upper Thames River basin

Urban Sector[†]

County	SW Res (m^3/yr)	SW Ind/Comm (m^3/yr)	GW Res (m^3/yr)	GW Ind/Comm (m^3/yr)
Middlesex	20,767,016	31,150,524	319,375	10,587,845
Oxford			14,419,325	55,179,970
Perth			4,172,414	4,580,542
Totals	20,767,016	31,150,524	18,911,114	70,348,357

Rural Sector[†]

County	SW Res (m^3/yr)	SW Ind/Comm (m^3/yr)	GW Res (m^3/yr)	GW Ind/Comm (m^3/yr)
Middlesex	155,037	232,555	1,783,824	2,428,579
Oxford			740,921	
Perth			783,393	148,584
Totals	155,037	232,555	3,308,138	2,577,163

Rural Sector Continued[†]

County		GW Livestock (m^3/yr)	GW Crops (m^3/yr)
Middlesex		592,749	1,502,946
Oxford		1,555,683	1,892,905
Perth		2,132,575	172,881
Totals		4,281,007	3,568,732

[†] SF = surface water; GW = ground water; Res = Residential; Ind/Comm = Industrial and commercial.

4 Continuous Hydrologic Model Component

The continuous hydrologic model component used in this report is based on the United States Army Corps of Engineers, Hydrologic Engineering Center's Hydrologic Modelling System (HEC-HMS). The modelling system is designed for rainfall-runoff modelling for a wide range of problems and geographic locations, although most of its applications have included North American basins. HEC-HMS has been around for more than three decades, and is well accepted by the hydrologic community.

HEC-HMS consists of three parts: (i) meteorologic module (which includes methods describing precipitation and/or evaporation); (ii) basin module (consisting of methods describing the physical properties of a catchment); and (iii) control module (where start and end times of a simulation are specified). The meteorologic and basin modules consist of a collection of methods allowing the user to specify and describe climatic and physical properties of the basin. For example, different loss methods (i.e., representing evaporation and/or evapotranspiration) are available depending if the user wishes to study short (event) or long (continuous) term hydrologic characteristics of the basin.

HEC-HMS has been applied to the Upper Thames River basin in the work of Cunderlik and Simonovic (2004b, 2005); from now on, their model is referred to as the continuous hydrologic model component (as it is a part of a larger model in this work). The original investigation provided a calibrated version of the hydrologic model component for the basin's outlet only. For the purposes of integrated analysis complete re-calibration has been performed to improve the performance of the model for locations other than basin outlet. For details of the calibration procedure regarding HEC-HMS models, the interested reader is referred to the manuals by USACE (2000, 2006), as well as to the work of Cunderlik and Simonovic (2004b, 2005).

The original hydrologic model component was created in HEC-HMS version 2.2.2, a program that has since then undergone a major update (USACE, 2006). Unfortunately the updated version of the software came after the bulk of the project work had already been completed. The requirements of the current project, however, demand tasks that would be difficult to complete even with the latest version of the software. For example, the continuous model (as applied to the Upper Thames River basin) requires consideration of seasonality—a model feature where a different parameter set is used for summer and winter seasons.

Furthermore, the main contribution of this work consists of linking the hydrologic model component to a socio-economic component via multiple feedback loops, where each component affects, and is affected by, the other. In order to represent such feedback links, a modelling framework should exist that attempts to mimic changes in hydrologic processes resulting from changes in socio-economic behaviour, as well as changes in socio-economic activity from altered physical (hydrologic, climatic) conditions.

Currently, the only way to represent such processes in HEC-HMS is to set up separate simulations representing different urbanization conditions, and then individually study their effects on regional hydrology. By doing this, dynamic characteristics of the overall basin behaviour (both hydrologic and socio-economic) become sacrificed. In order to change this a framework is needed where HEC-HMS (or any other hydrologic modelling system) can be made to respond dynamically to changes in socio-economic processes of the basin, and thus represent more accurately processes relevant for the system's overall behaviour.

The only way to dynamically link a hydrologic modelling framework like HEC-HMS to socio-economic models is to create an interface where communication and exchange of information between the two become possible. The communication of

information in this sense is one where one model provides a continuous signal to another model. Current version of HEC-HMS does not allow the user to do this. To cope with this problem, the computational engine of the continuous version of HEC-HMS was built from scratch in the Java programming language so that it could be later linked (see Section 6) to a socio-economic model component.

The continuous hydrologic modelling component used in this study is somewhat different than the original HEC-HMS framework, as it has been adapted for the needs of the current project. The hydrologic model component is divided into the following three major modules: (i) weather generator module (where the climatic input signal is generated); (ii) snow accumulation and melt module (where input precipitation is adjusted to account for snowfall and rainfall); and (iii) precipitation-runoff module (where hydrologic processes are described). Each module is described next.

4.1   Weather Generator Module

The weather generator used in this project was first developed by Sharif and Burn (2004, 2006a,b). The weather generator operates on a daily time step, and thus generates weather information each day, for each variable of interest—in our case, precipitation, minimum and maximum temperature. The synthetically generated data are produced for fifteen stations within and around the basin. (The spacial resolution of the generated data depends on the locally available climatic data used to condition the weather generator during model execution.) Sample output for one station is depicted in Figure 8 for a five year period of record, while the names of the stations (along with their coordinate points) are shown in Table 4.
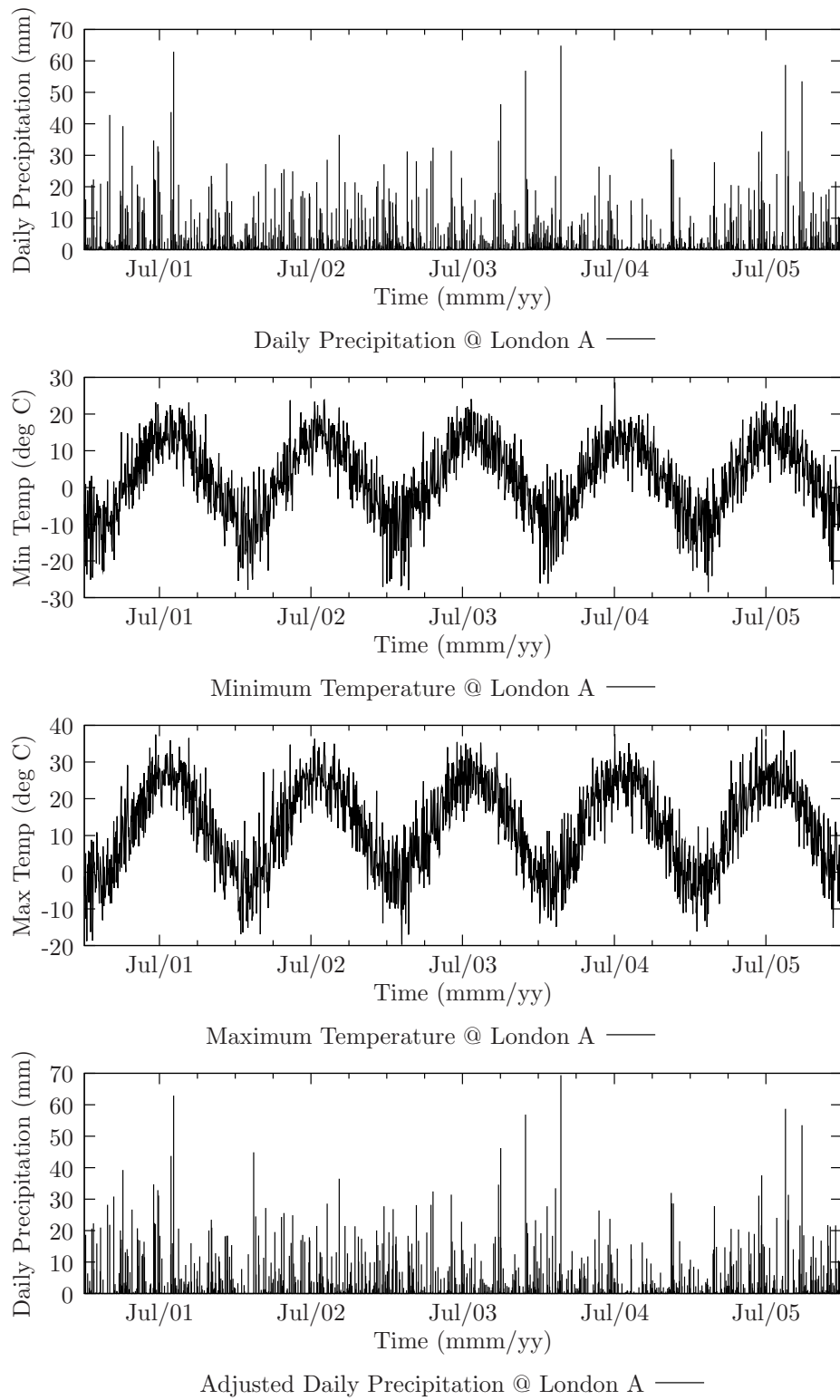
Figure 8: Weather generator/snow module output for the historic scenario

Table 4: Locations of generated meteorological data

No.	Station	Latitude	Longitude	No.	Station	Latitude	Longitude
1.	Blythe	43° 43'	-81° 22'	9.	London	43° 01'	-81° 09'
2.	Dorchester	43° 00'	-81° 01'	10	St. Thomas	43° 46'	-81° 12'
3.	Embro	43° 15'	-80° 55'	11.	Stratford	43° 22'	-81° 00'
4.	Exeter	43° 21'	-81° 30'	12.	Tavistock	43° 19'	-80° 49'
5.	Foldens	43° 01'	-80° 46'	13.	Waterloo	43° 28'	-80° 31'
6.	Fullarton	43° 23'	-81° 12'	14.	Woodstock	43° 08'	-80° 46'
7.	Glen Allan	43° 40'	-80° 43'	15.	Wroxeter	43° 52'	-81° 09'
8.	Ilderton	43° 03'	-81° 25'				

4.2   Snow Accumulation and Melt Module

The snow module (shown in Figure 9) takes time series data of temperature and precipitation provided by the weather generator and uses it to compute snow accumulation and melt based on the degree-day method (Cunderlik and Simonovic, 2004b, p. 65). An interpolation method is used to spacially interpolate the precipitation and temperature, as the climatic data provided by the weather generator is available only for fifteen stations within and around the basin. This is needed because the hydrologic model component used in this study contains thirty two sub catchments, which must receive precipitation as input. The interpolation of temperature and precipitation is accomplished with the Inverse Distance Weighting Method (USACE, 2000, p. 23). After the climatic data is interpolated, it is separated into solid and liquid forms of precipitation. The solid form of precipitation (or snowfall) is then subject to an accumulation and melt algorithm, eventually producing snowmelt. Snowmelt is then added to the liquid precipitation (or rainfall) producing a new variable defined as adjusted precipitation. It is this precipitation that is used next as input to the losses module.

The set of equations that represent snow accumulation and melt module are pre-

Weather
Generator
Data

Temperature   Precipitation

Interpolate
Process

Temperature   Precipitation

Separation   Snowmelt
Process   Snowfall   Process

Rainfall

Snowmelt

Adjusted
Precipitation
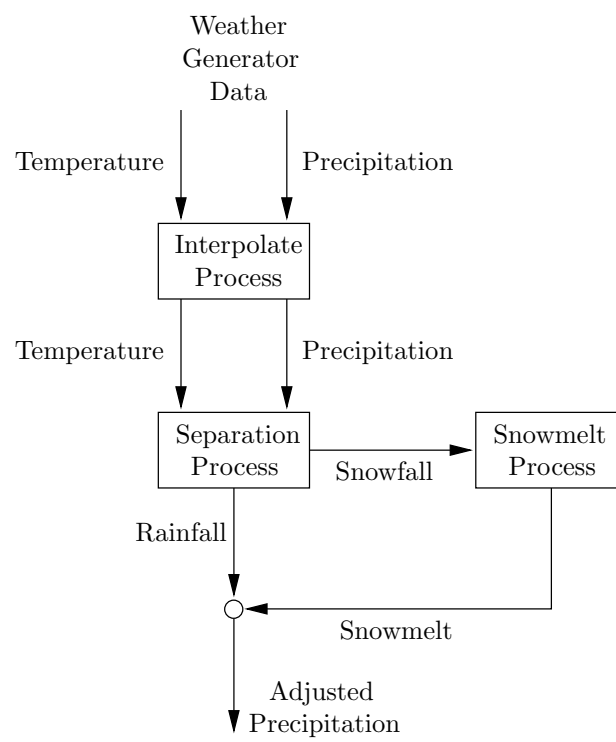
Figure 9: Snow model module

sented next. After the climatic data (temperature and precipitation) is interpolated, the algorithm can start. If the average daily temperature T_t is less than the minimum temperature for snowfall $T_- = -4°$ C, it means precipitation is falling as snow and:

$$S_t = P_t \tag{1}$$

$$R_t = 0.0 \tag{2}$$

where P_t is the measured amount of precipitation in [mm/day], while S_t and R_t are amounts of precipitation that fall as snow and rain respectively, in [mm/day]. The subscript t in above equations represents simulation time, where $t = 1, 2, 3, \cdots, n$ and n is the total number of days for which precipitation data is available. If the average daily temperature is somewhere between the minimum ($T_- = -4°$ C) and maximum ($T_+ = -2°$ C) temperature needed for snowmelt, the snowfall and rainfall are calculated as:

$$S_t = P_t \cdot \left[\frac{T_+ - T_t}{T_+ - T_-}\right] \tag{3}$$

$$R_t = P_t - S_t \tag{4}$$

Similarly, if the average daily temperature is greater than the maximum temperature for snowmelt, all precipitation is assumed to fall as rainfall, represented with:

$$S_t = 0.0 \tag{5}$$

$$R_t = P_t \tag{6}$$

The solid form of precipitation (or snowfall) is then subject to an accumulation and melt algorithm, eventually producing snowmelt. The daily amount of melt is

computed using:

$$M_t = MR \cdot (T_t - T_{cr}) \tag{7}$$

where MR is a parameter representing melt rate [mm/°C/day], and is set to 4.0; T_{cr} is a critical temperature for melt, and is set at zero. Snowmelt previously obtained is accumulated with the following equation:

$$S_t = S_t + S_{t-1} \tag{8}$$

from which adjusted precipitation is calculated. If snowmelt occurs (i.e., if $M_t > 0$) and if the accumulated snowmelt S_t is greater than the melt rate M_t ($S_t > M_t$), this implies that only a portion of the accumulated snow melts:

$$S_t = S_t - M_t \tag{9}$$

$$P_a = R_t + M_t \tag{10}$$

where P_a represents adjusted precipitation in [mm/day]. If, on the other hand, all accumulated snow melts, the adjusted precipitation becomes:

$$P_a = R_t + S_t \tag{11}$$

Lastly, if no snowmelt occurs, the adjusted precipitation takes on simply the vale of rainfall.

$$P_a = R_t \tag{12}$$

The adjusted precipitation is used next as input to the losses module. An example of a typical snow model output is shown on the bottom most plot in Figure 8.

4.3   Precipitation-Runoff Modules

The overall structure of the continuous hydrologic model component is shown in Figure 10, where each box represents a module that mathematically represents a physical processes functioning in the basin. Precipitation and temperature (obtained from the weather generator model) are used as inputs to a snow module, where adjustments are made to account for both solid and liquid precipitation. The output of the snow module is adjusted precipitation, used for computation of losses (used here in plural to emphasize existence of more than one kind of water loss). The losses module represents the movement of moisture through various conceptual reservoirs within a catchment, such as canopy, surface, soil, and ground water. One of the outputs of the losses module is evapotranspiration, or moisture that evaporates from the canopy, surface depressions, and/or the soil. Other outputs include baseflow (or flow being returned to the stream from ground water), surface excess (that portion of the flow that does not infiltrate into the soil), and ground water recharge (the flow that enters deep aquifers and does not return to the stream). The surface excess is then used by a transform module, or one which converts surface excess to direct runoff by convoluting a unit hydrograph. Its output is surface runoff, which is then combined with baseflow to produce direct runoff. Direct runoff, on the other hand, is used as input of a flood routing module, which calculates the propagation of a flood wave eventually producing streamflow.

4.3.1   Water Losses Module

The most complicated component of the hydrologic model component is the losses module. This is because it takes into consideration a large number of processes that are simultaneously functioning, which must be properly accounted for. The concep-

Temperature
Precipitation

Snow
Module

Evapotranspiration

Adjusted
Precipitation

Losses
Module

Transform
Module

Surface
Excess

Surface
Runoff

Baseflow

Direct
Runoff

GW Recharge

Routing
Module

Streamflow
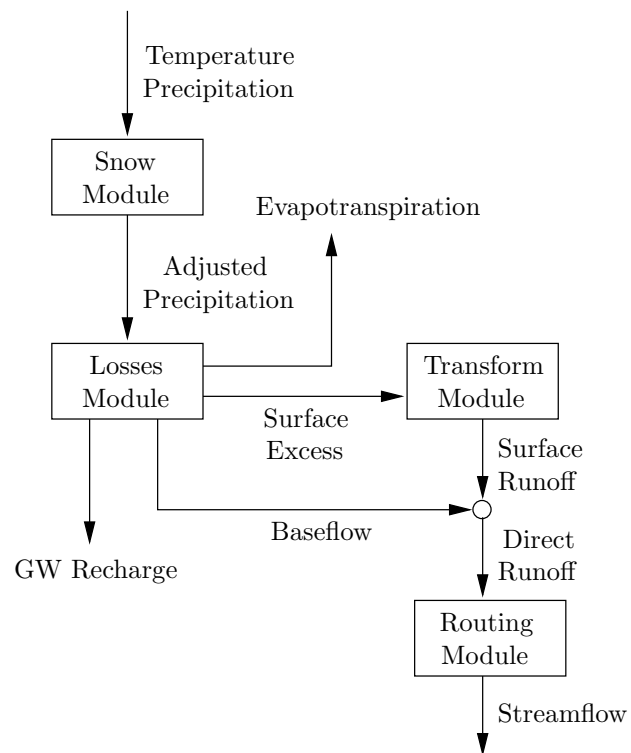
Figure 10: Hydrologic model component

tual diagram of this module is shown in Figure 11, and is based on the work of Bennett (1998), where details of the Soil Moisture Accounting algorithm are presented. Furthermore, Bennett's algorithm is based on the work of Leavesley et al. (1983), and their Precipitation-Runoff Modelling System.

The losses module uses a series of conceptual reservoirs to represent the storage and movement of water in each sub catchment of the basin. The storage reservoirs are: (i) canopy interception; (ii) surface interception; (iii) soil profile, and (iv) a number of ground water layers (only two shown here). The inflow and outflow rates between the reservoirs regulate the amount of water stored in each conceptual reservoir. They include evapotranspiration, infiltration, percolation, surface runoff and ground water flow.

The canopy storage layer includes the precipitation captured by vegetation (such as trees, shrubs, bushes, grasses, etc); furthermore, the precipitation is the only inflow that can fill this storage volume. When precipitation occurs, it fills this storage layer first, provided it is not already at capacity. The only process that can remove the moisture out of this layer is the process of evapotranspiration. After filling the canopy layer, the precipitation becomes available to fill the surface storage, and/or to infiltrate into the soil. The surface storage layer represents the volume of water held by shallow depressions and cracks on the ground surface. The storage of water in this layer (in addition to precipitation excess from the canopy) is one that is available to infiltrate into the soil, provided the soil is not fully saturated. The inflow to the surface storage layer is water that does not infiltrate the soil layer; in other words, it is a combination of the precipitation excess from the canopy layer, and its own volume left over after infiltration took place. The outflow from this layer is evaporation and surface runoff. The surface runoff is the flow produced when the surface storage layer is at capacity, and thus can not absorb water that previously has not infiltrated.
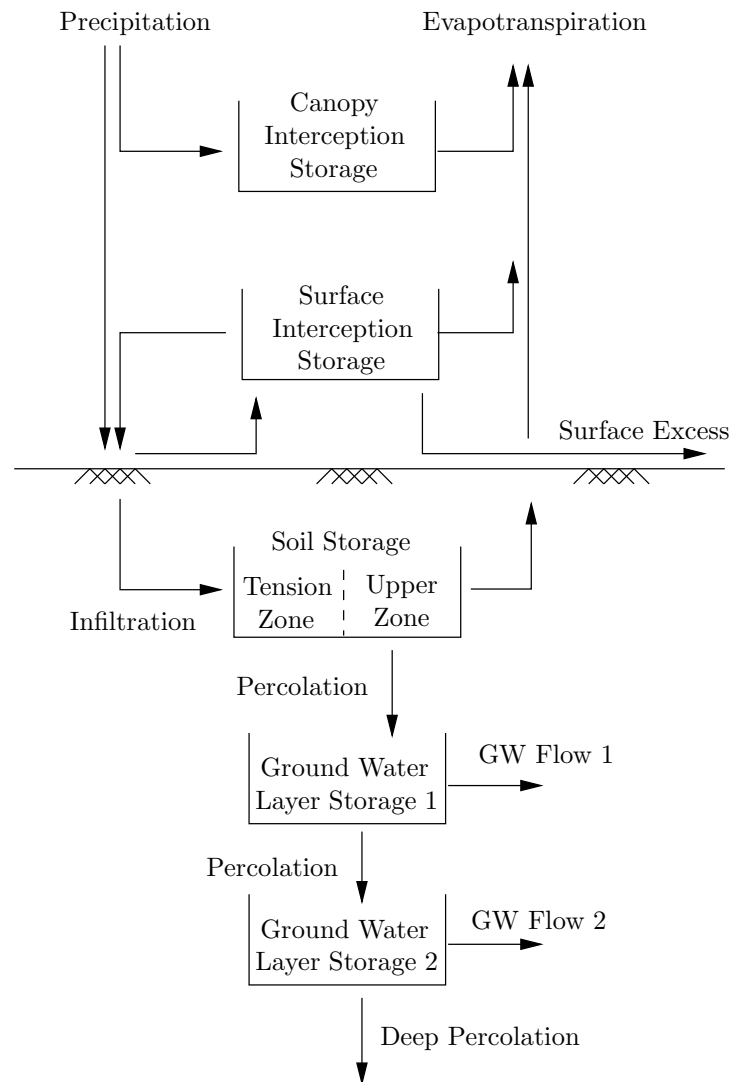
Figure 11: Soil Moisture Accounting losses module

During large precipitation events, the canopy and surface storage layer become full quickly thus producing high amounts of surface excess—as infiltration alone is usually never able to absorb all surplus precipitation.

The soil profile storage corresponds to the top layer of the soil. Water that infiltrates is the only inflow to this layer, while the outflows represent percolation to the lower ground water layer and evapotranspiration. The soil storage is further divided into two zones: the upper zone and the tension zone. The former is that portion of the soil layer that can lose water to both percolation and evapotranspiration, while the latter is one that looses water only through evaporation, but not percolation (Bennett, 1998, p. 5). This is because the upper zone represents water held in the pores of the soil (which can freely percolate and/or evaporate), while the tension zone constitutes water held by capillary tension, thus making it difficult to flow and move but can evaporate. It should be mentioned that evapotranspiration rates from the soil vary, as it is more difficult to remove water held by capillary tension than water held between the pores of the soil.

Evapotranspiration is the process that removes moisture from canopy, surface, and soil profile storage. In the Soil Moisture Accounting algorithm, evapotranspiration can only occur during times free of precipitation. Potential evapotranspiration is calculated based on maximum regional monthly evapotranspiration rates (specified by the user), multiplied by a pan coefficient. Actual evapotranspiration rates are realized through a loss of moisture from the canopy first, the surface second, and soil storage last, but can never exceed its potential rates.

Lastly, the water that percolates from the soil profile storage is used as an input to the ground water layer immediately beneath. The outflows from this layer represent the ground water flow (one that is returned to the stream channel as baseflow), and a further percolation to either another ground water layer or as deep percolation—

representing water entering a deep aquifer.

Since all equations for the Soil Moisture Algorithm can not be reproduced here (Bennett (1998) takes 90 pages to do it), only few key relationship are presented. They include infiltration, percolation and lateral ground water flow. The process of soil infiltration in the Soil Moisture Algorithm is captured by first calculating the potential soil infiltration, PSI in [mm/hr]:

$$PSI_t = MSI - (SoS_t/SoS_m)MSI \tag{13}$$

where MSI represents maximum soil infiltration, in [mm/hr]; SoS_t the volume of water in soil, in [mm], and SoS_m the maximum volume in soil storage, in [mm]. Equation (13) reflects that potential soil infiltration can never exceed the infiltration capacity (taken as the absolute maximum), and depends linearly on the current volume of water held in the soil. If the soil contains little or no water, the potential infiltration can be as high as the infiltration capacity; however, if the soil is at, or near saturation, the potential infiltration will be small. The actual soil infiltration ASI_t at time t is computed as the minimum value between the water available for infiltration WA_t and the potential soil infiltration PSI_t. Therefore,

$$ASI_t = \min(AW_t, PSI_t) \tag{14}$$

reflecting that the amount than actually infiltrates can never be more than is currently available, despite how high the potential infiltration is.

Similar to the process of infiltration is percolation, or the water that is transferred from the soil storage to a ground water layer immediately beneath. The potential soil

percolation PSP_t is computed as:

$$PSP_t = MSP \left(\frac{SoS_t}{SoS_m} \right) \cdot \left(1 - \frac{GWS_t}{GWS_m} \right) \tag{15}$$

where MSP stands for maximum soil percolation in [mm/hr]; SoS_t for water volume in soil storage in [mm]; SoS_m for soil storage capacity in [mm]; GWS_t for current ground water layer storage, and GWS_m for maximum ground water storage both in [mm]. Equation (15) indicates that potential soil percolation depends on amounts of water in soil and ground water. For example, if the ground water storage is at or near saturation, large amounts of water will not be able to percolate to the ground water aquifer. The potential percolation also depends on the amount of water currently in the soil, demonstrating that if soil is void of water, no percolation can occur. Actual soil percolation (ASP_t) is computed in the same manner as actual soil infiltration, and thus is the minimum between the potential soil percolation, and the amount of water available for percolation.

The outflows from the ground water layers represent the later ground water flow (one that is eventually returned to the stream channel as baseflow). The Soil Moisture Algorithm represents this lateral flow as:

$$GWF_t = \frac{(ASP_t \Delta t) + GWS_{t-1} - (PGWP_t \Delta t) - (0.5 GWF_{t-1} \Delta t)}{k + 0.5 \Delta t} \tag{16}$$

where GWF_t stands for ground water flow in [mm/hr]; $PGWP_t$ for potential ground water percolation in [mm/hr], and k for a ground water storage coefficient [hr]. The derivation of this equation is presented in (Bennett, 1998, p. 57). Such lateral flow from both ground water layers is averaged, before being routed with a series of linear reservoirs to produce baseflow.

4.3.2   Transform Module

Surface excess and ground water flow obtained from the losses module are two variables that must be further analyzed. Immediately after leaving the losses module they are used as an input to a transform module (i.e., to convert them to quantities that eventually enter the stream channel). In order to transform surface runoff (and ground water flow) into streamflow (and baseflow), unit hydrograph theory is typically employed. In the hydrologic model component of the present study, Clark's unit hydrograph is used to transform surface excess into direct runoff (Hoggan, 1996; USACE, 2000), while ground water outflow is transformed to baseflow using a series of linear reservoirs (Bennett, 1998, p. 57).

Clark's unit hydrograph method is shown in Figure 12, and represents a procedure used to conceptually represent the process of conversion between surface runoff (the water which is distributed on the ground surface after rainfall takes place) to direct runoff (the same water that drains from the catchment). The method starts by adopting a time-area relationship [Figure 12(a)], given by the following equation:

$$A_I = 1.414(t/T_c)^{3/2} \quad 0 \leq t/T_c < 0.5 \tag{17}$$

$$1 - A_I = 1.414(1 - t/T_c)^{3/2} \quad 0.5 < t/T_c < 1 \tag{18}$$

where A_I is the cumulative fraction of basin's area and t/T_c is a fraction of the time of concentration. Adoption of such a relationship assumes that velocity of overland flow is uniformly distributed over the catchment area, and that the time required for runoff to reach the outlet is directly proportional to distance travelled (Hoggan, 1996, p. 54). In other words, this relationship represents a temporal distribution of excess rainfall found on the surface.

(a) Time Area Map of Basin

(b) Translational Hydrograph

(d) Instantaneous Unit Hydrograph

(c) Linear Reservoir Routing
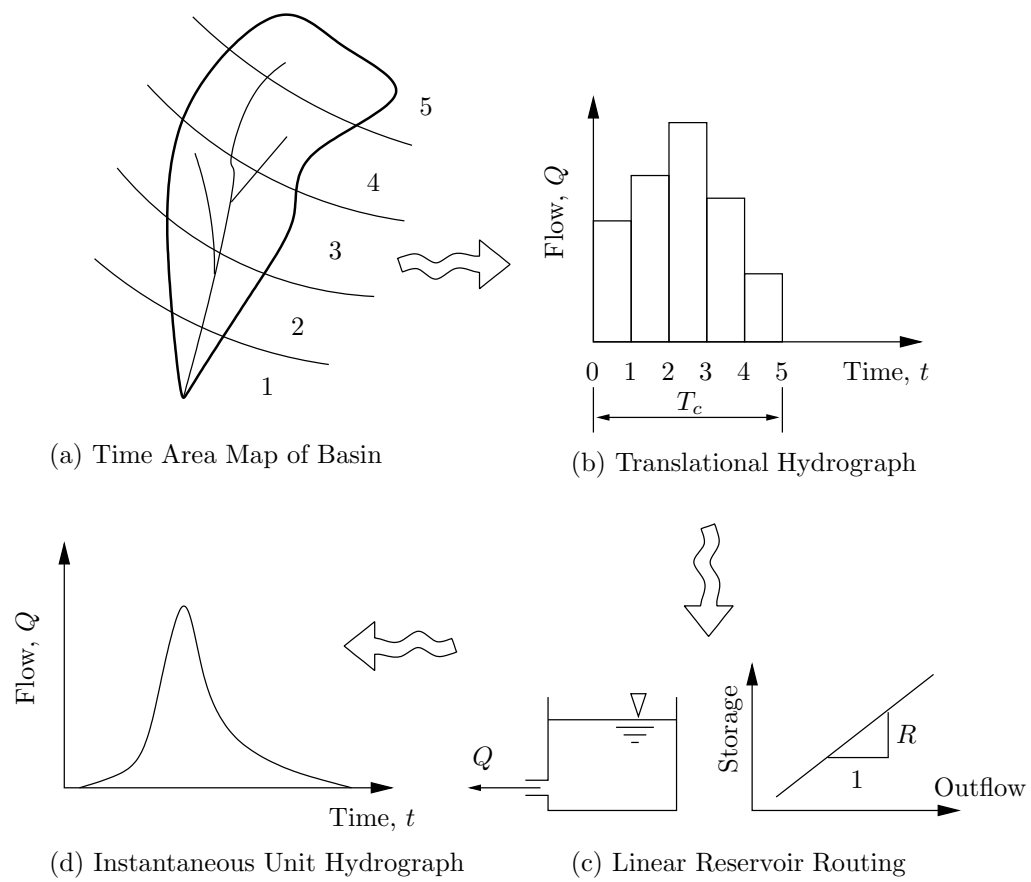
Figure 12: Clark's transform module

In order to convert this excess to direct runoff, a translational unit hydrograph [Figure 12(b)] is constructed based on the assumed time-area relationship, its catchment size, and its time of concentration (the time needed for a drop of water falling on the farthest point in the catchment to reach its outlet, denoted by T_c). Translational unit hydrograph describes the flow passing through the catchment outlet if no attenuation takes place. By knowing the time-area relationship, along with basin's total area, we can therefore compute a translational hydrograph. The volume of the translational hydrograph corresponds to a uniform rainfall of one unit (either mm or in.) falling over the basin for the duration equivalent to the time of concentration.

However, since hydrograph attenuation does occur, the translational hydrograph is thus routed through a linear reservoir [Figure 12(c)]. At this step the slope of the storage-outflow function is specified (represented by R) as it is needed to route the translational hydrograph through a linear reservoir to obtain an instantaneous unit hydrograph. The linear reservoir routing is represented by a discrete approximation of the continuity equation combined with a linear storage outflow function:

$$\bar{I}_t - \frac{O_{t-1} + O_t}{2} = \frac{S_t - S_{t-1}}{\Delta t} \tag{19}$$

where $\bar{I}_t$ represents an average inflow at time t, or the ordinate of the translational hydrograph, and O_t and S_t represent the outflow and storage during Δt, using the following linear relationship:

$$S_t = RO_t \tag{20}$$

Substituting (20) into (19) and simplifying obtains a solution for O_t, from which the equation for the unit hydrograph is derived:

$$U_t = \frac{O_t + O_{t-1}}{2} \tag{21}$$

where U_t represents the ordinate of the unit hydrograph. Finally, to obtain direct runoff of the basin Q_t, the unit hydrograph is transformed (via the convolution equation in discrete form):

$$Q_t = \sum_{i=1}^{t} E_i U_{t-i+1} \tag{22}$$

where E_i is the rainfall excess, computed from the Soil Moisture Accounting loss module. The direct runoff is then added to baseflow, and total watershed runoff obtained.

4.3.3 Routing Module

The last module used by the hydrologic model component is the routing module—representing a method that describes the movement of a flood wave as it passes through a river and/or a reservoir. The mechanism employed here is a simple hydrologic routing method (Chow, 1956; Hoggan, 1996; USACE, 2000), referred to as the Modified Puls method. Its computational procedure is analogous to one of described above for the linear reservoir method, with the only exception being that nonlinear storage-outflow functions are employed here. In essence, the method takes an hydrograph as input [Figure 13(a)], sends it through a nonlinear reservoir [Figure 13(b)], and produces a modified flood hydrograph [Figure 13(c)] as output. The assumption here is that channel (or a reservoir) provides enough storage and/or resistance that the original hydrograph is flattened and attenuated.

It must be acknowledged that reservoirs in the Upper Thames River Basin (especially Fanshawe) may not be operate according to the Modified Puls method. The calculated releases from the reservoirs using this method thus represent only approximations of management practices actually employed by those responsible for their operations. Since the basin response is extremely sensitive to reservoir operations (es-
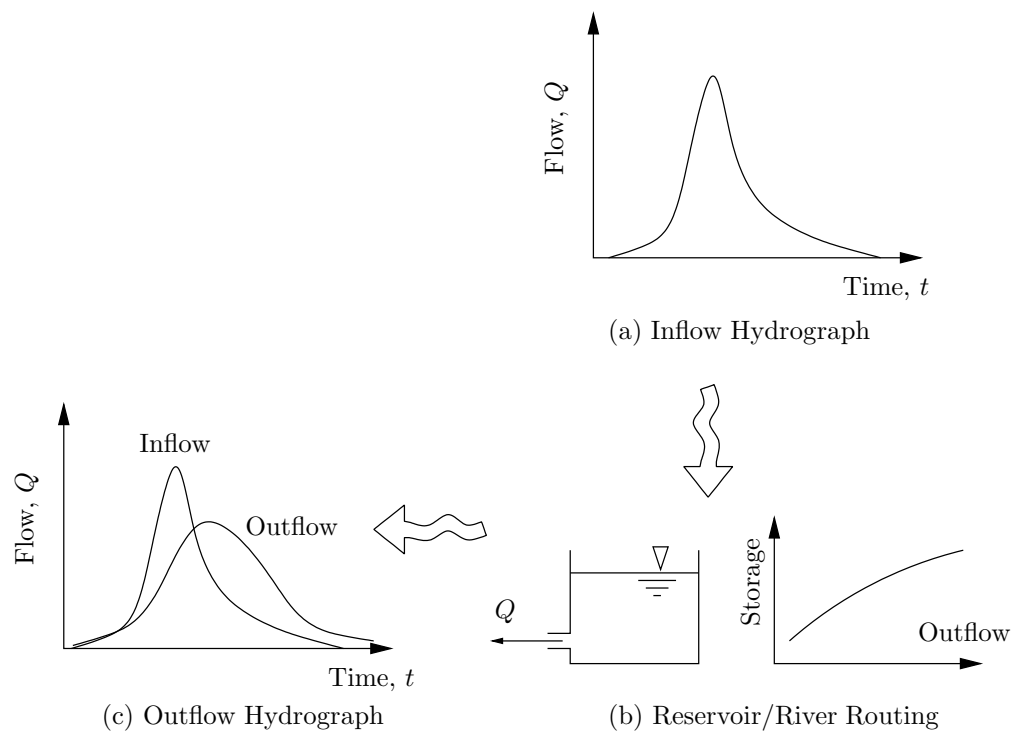
Figure 13: Modified Puls routing module

pecially to large reservoirs), accurate regional hydrologic conditions are only captured when the approximations match the actual releases. To try and correctly represent the reservoir operations is beyond the scope of this project, and should probably be a subject of a separate study. The encouraging fact is that the entire source code for the hydrologic model is available (see the Appendix), which can easily be adopted to custom modelling of the regional reservoir operating rules, and thus representing hydrologic conditions more accurately.

4.4   Hydrologic Modelling of the Upper Thames River Basin

The continuous hydrologic model component has been applied to the Upper Thames River basin for the purpose of studying long term regional risk and vulnerability to changing climatic conditions. The application of the model consists of thirty two sub basins, twenty one river reaches, and three major flood control reservoirs (Wildwood, Fanshawe, and Pittock). The schematic of the model is shown is Figure 14. Each sub basin in the schematic (represented by rectangles) is provided with climatic input from the weather generator. The outputs of each sub basin are flow hydrographs, joined by junctions (shown as circles) where flows are added together. River reaches represent the major rivers and streams in the basin; in the schematic, they are shown as thick lines connected between two junctions. The routing module described above is applied to each river reach, and thus describes a passage of a flood wave as it moves through the river system. Reservoirs are depicted as triangles, to whom same routing rules apply. A sample simulation output is shown in Figure 15. Note that the source code for the hydrologic model component is available in its entirety in the Appendix; code for all modules, structure as well as parameter values of all model components are also provided.
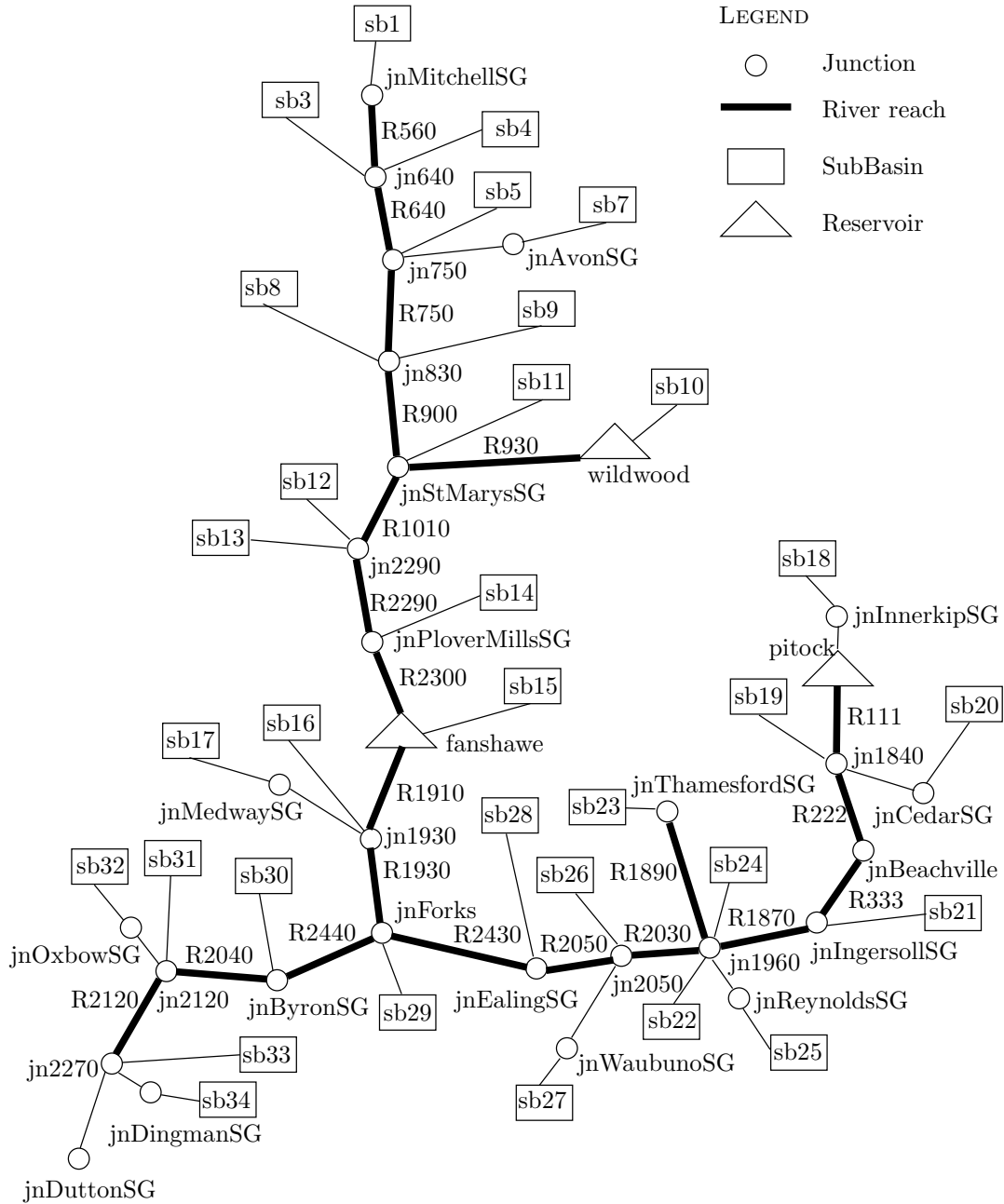
Figure 14: Schematic of the continuous hydrologic model component
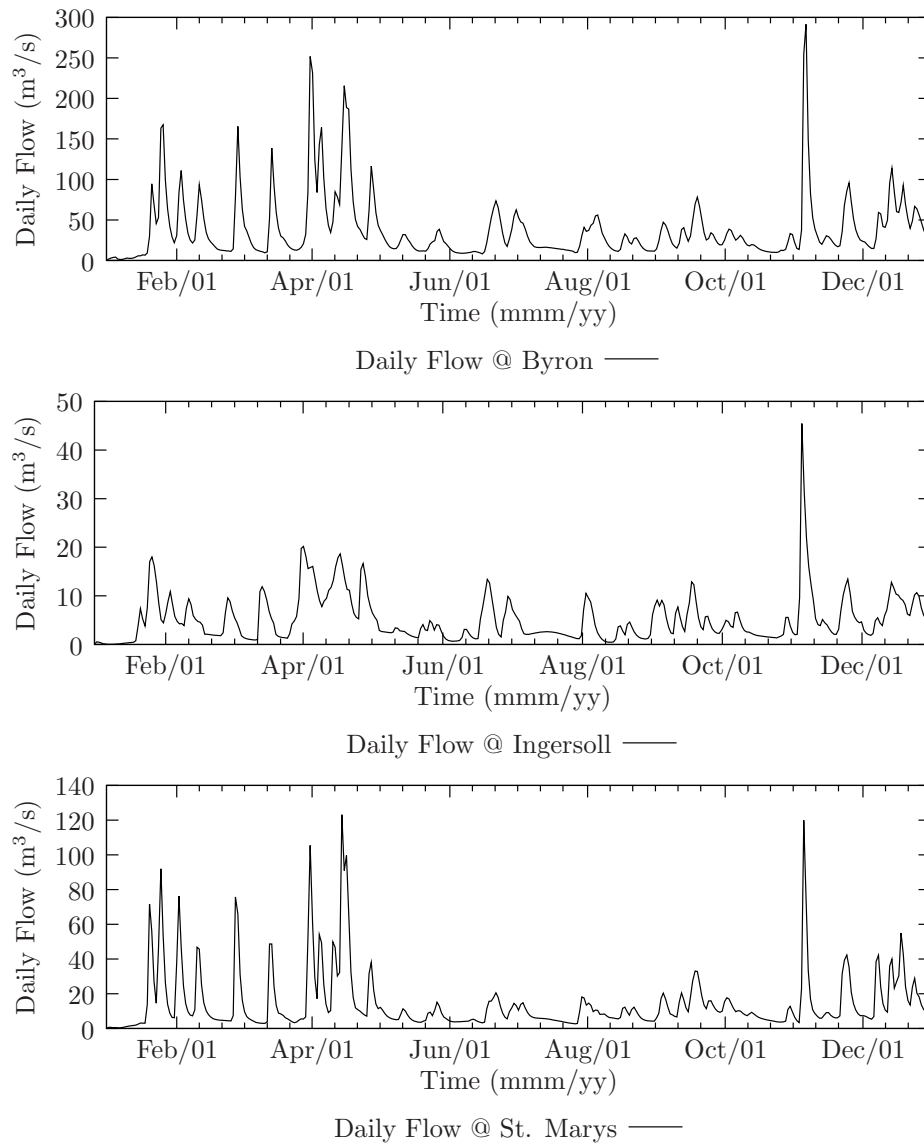
Figure 15: Daily flow for historically identical scenario

5 Socio-Economic Model Component

5.1 Upper Thames System Model Structure

The socio-economic model component is developed to help reveal dynamic behaviour of socio-economic conditions embedded and inextricably linked to physical conditions of water resources systems. The intention is to provide a practical framework where physical processes (climatic, hydrologic) can be linked to socio-economic processes (demographics, business activity, land use) for the purpose of evaluating risks and vulnerability of water resources systems to impacts of climatic change. The model is developed and applied to the Upper Thames River basin study area, although its structure is applicable to other regions as well with appropriate modification. The rest of this section describes the structure of the modelled system, and provides details on each sector considered. As the overall model contains a large number of equations, they will not all be listed here. The most significant functional forms are presented together with few selected equations, while the Appendix lists them all in their entirety.

The Upper Thames River Basin socio-economic model component is divided into three spacial units. These units are selected as regional economic activity is easier described using a larger aggregated area; in our case, the spacial units are defined as areas belonging to each county located in the Upper Thames River basin (see Section 3 for a map of the basin), and are thus referred as Middlesex, Oxford and Perth. County wide spacial units are appropriate as selected socio-economic data for the study area is only available on the county wide basis—see Tables 1-3 in Section 3 of this report.

Each spacial unit contains identical model structure, as it is assumed that socio-economic characteristics in each region are similar (based on the number of people,
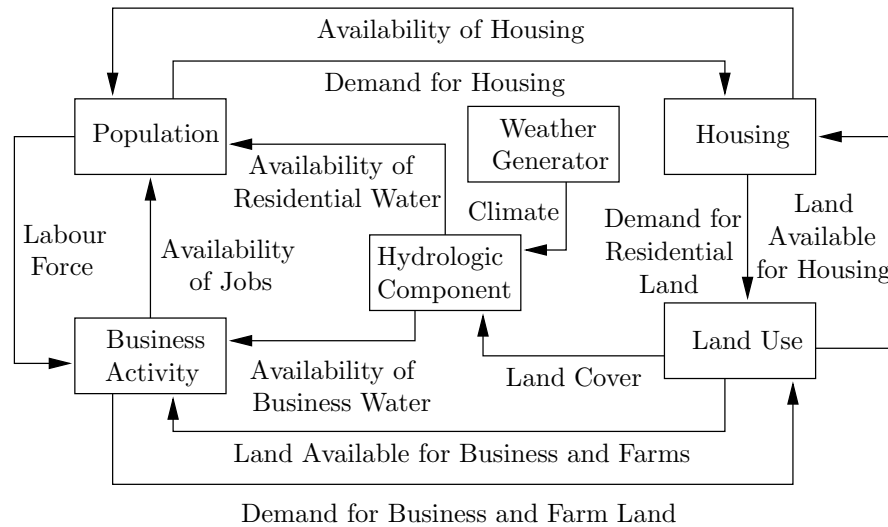
Figure 16: Socio-Economic model component architecture

unemployment level, jobs, etc.) As the region's economic output heavily depends on industrial as well as agricultural activity, separate sectors are built to account for such features. Each spacial unit therefore contains sectors describing urban and rural population, urban and rural business activity, housing, as well as land use. The socio-economic model component architecture is shown in Figure 16, where each major sector is depicted by a rectangle and linked to other sectors via variables shown adjacent to the solid lines.

Details regarding feedback links between the hydrologic to the socio-economic components of the overall model are only mentioned here; their full description is provided in Section 6. Description in this section shows information links between all sectors making up the socio-economic model component. Details of the individual sectors follow. It should be noted that the socio-economic model component is based on the work of Forrester (1969), Mass (1974), Schroeder et al. (1975) and Alfeld and Graham (1976).

The representation of population is subdivided into urban and rural sectors. These

sectors contains information that generate population dynamics based on current population, birth and death rates, as well as descriptions of in and out migration. Within the scope of the model, the population sectors depend on the availability of water, jobs and houses, illustrating the fact that each plays a prominent role in the dynamic of growth. Each of the three variables places a limit on the possibility of future growth and expansion. For example, the area can not experience population growth if adequate water, jobs or housing are not available. The business activity sector is likewise divided into urban and rural categories, representing dynamics of business and farm units, respectively, through investment and depreciation of capital. The business activity sector depends on a labour force provided by the population sector (where only a fraction of the total population represents labour force), water availability provided by the hydrologic model component, as well as availability of land allocated for business and farm use, obtained from the land use sector. Again, three conditions (available land, labour force, and water) regulate growth of economic activity, implying that each has the potential (independently and in combination) to affect regional economic growth.

The housing sector receives information cues from the population (through demand for housing) and land use (through land available for housing development), and thus simulates dynamics of the housing industry. Housing sector is only used in urban settings, as farm units are defined to represent places where rural families both live and work. Lastly, the land use sector is the most detailed part of the socio-economic model as it represents dynamics of change in terrestrial land cover—defined in this study as forest, agricultural, business and residential land use types. In essence, the residential and business communities exert pressure on the local government to rezone and allocate additional land to their respective uses if favourable conditions exist. An assumption is made that as business and population growth con-

tinues, remaining forest cover is converted to agricultural land, while agricultural land is developed into business and/or residential uses. A competition structure between residential and business land is established to cover cases when further conversion of agricultural land is not allowed and/or not possible (as in fully urbanized areas). In such cases, conversion between residential and business lands alternates depending on pressures from its respective communities.

In order to better understand the details of different model sectors, a general overview of level and rate variables for each sector are shown in Figures 17-19. The level and rate diagrams show major flows of material and information, which regulate the overall systems behaviour. The major level (or state) variables in the urban sectors (see Figure 17) are business structures, urban housing, and population (all defined in detail later). Their major flows include rates that increase the state variables (such as construction, births and in migration), or decrease it (through depreciation and demolition, deaths and out migration). The rural model sectors are similarly represented, and are shown in Figure 18.

The land use sector in Figure 19 shows the representation of land cover in the study area—forest, agricultural, residential and business land. This sector describes the political process used to rezone land cover from one type to another. For example, if demand for additional agricultural land is strong, the agricultural community will place pressure on the local government to clear remaining forest cover and thus convert it to farm land. Likewise, the same logic is applied if the urban communities demand additional land for residential and industrial development—in this case agricultural land gets converted to residential and business categories. In cases when an area contains only residential and business land (such as cities, or when all agricultural land disappears) a competition structure exists that converts one land use type to another.
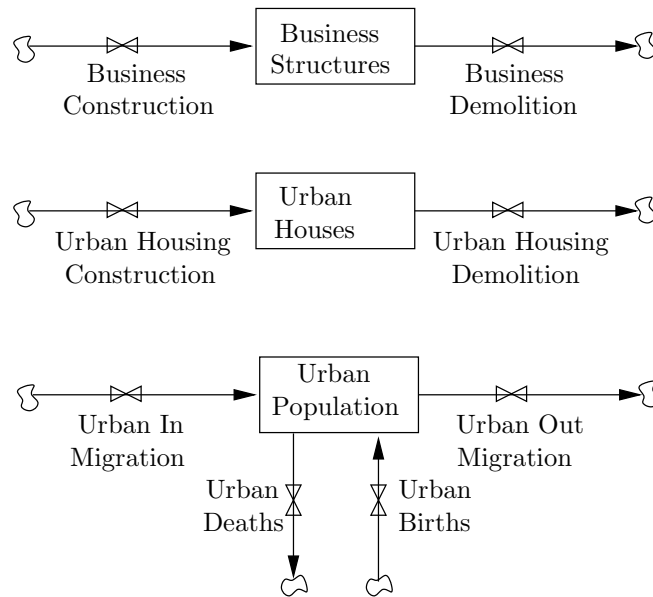
Figure 17: Levels and rates schematic of the urban sectors
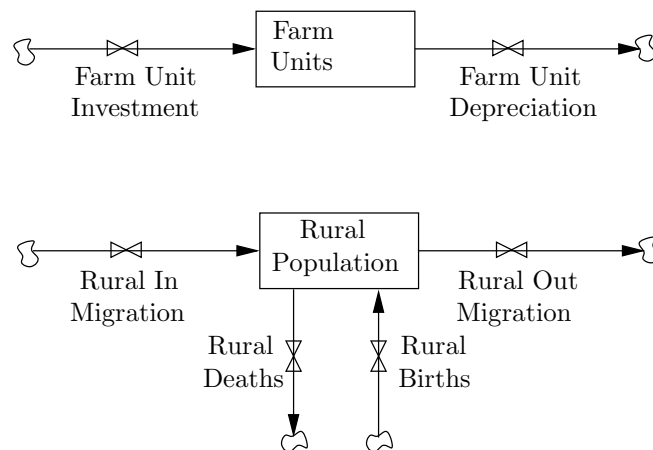
Figure 18: Levels and rates schematic of the rural sectors
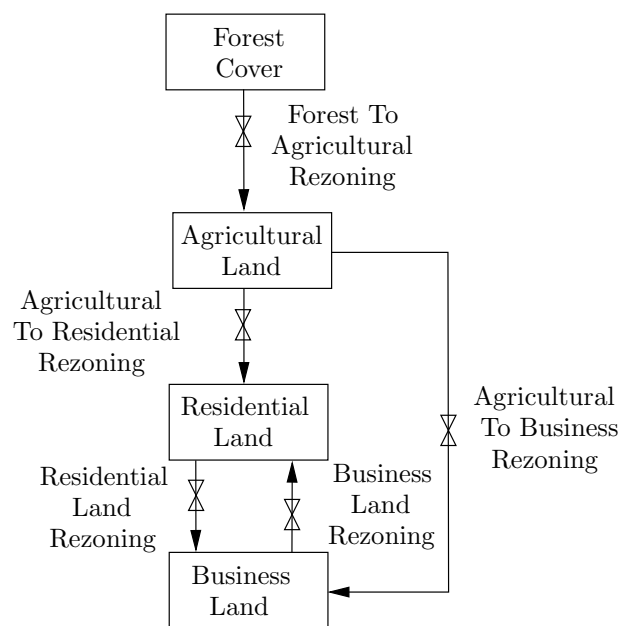
Figure 19: Levels and rates schematic of the land use sector

5.1.1 Urban Business Activity

This section describes inputs and structural relationships used in the urban business activity sector, as well as selected relationships with other model components. This is needed because certain information is shared by more than one sector of the model. The sector described here presents the underlying dynamics of urban economic activity in the area. The sector includes a detailed breakdown of surface and ground water use and availability, postulated here as a necessary determinant to future economic prosperity. The auxiliary variables providing information to the sector are shown in Figure 20, and are explained next.

The hydrologic model component provides monthly precipitation and flow variables (bolded), which are used to compute regional drought conditions based on the definition provided by Ontario's Low Water Response (OLWR, 2003) regulations. An information delay is applied to the drought level variable producing a perceived drought level, as it is assumed that action in response to drought is taken only after some period of time (i.e., it takes time to measure drought and convince people it will really cause a problem). In the model, this is represented as:

$$PDL_t = \text{Smooth3}(DL_t, DrT) \tag{23}$$

where PDL_t stands for perceived drought level, and DL_t for actual drought level, both as a dimensionless index between 0-3 (see OLWR, 2003); DrT represents a drought delay time in [yrs]. The function Smooth3 is defined in detail in (Sterman, 2000, p. 433), and corresponds to a third order information delay. This function has the responsibility of smoothing as well as delaying the input signal (actual drought level), based upon which future action is taken.

Response to drought in the model is accomplished through a water use reduction
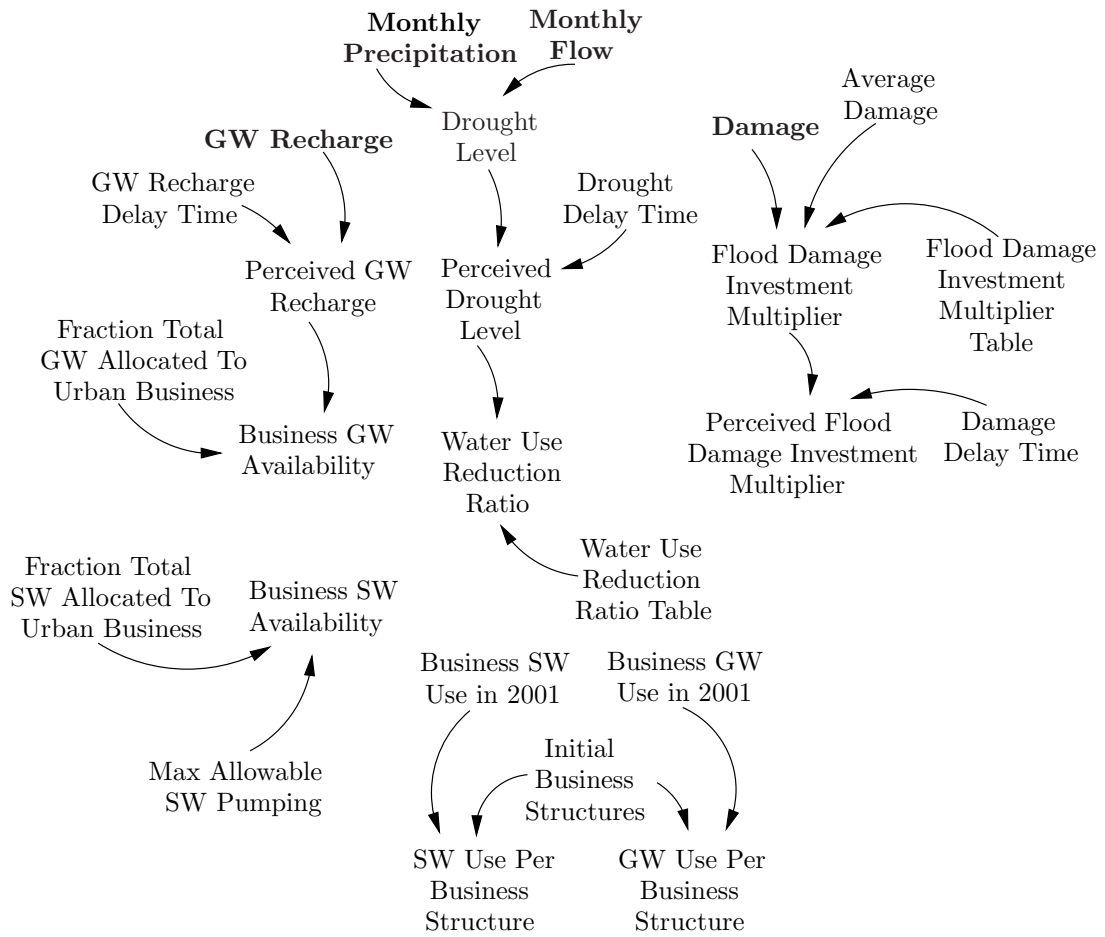
Figure 20: Auxiliary variables of the urban business activity sector

ratio ($WURR_t$), implying that area residents and businesses actually reduce their water consumption based on perceived drought conditions. The right plot in Figure 21 shows the functional relationship between the perceived drought level and the water use reduction ratio. The relationship is based on the measures recommended by OLWR (2003). As drought conditions worsen every water user is assumed (optimistically) to implement reduction measures leading to a maximum of 30% reduction during most severe droughts. This is represented mathematically with:

$$WURR_t = WURRT(PDL_t) \tag{24}$$

where $WURRT$ is the water use reduction ratio relationship that relates perceived drought level to reduced water use. Relationships like this simply convert an input (PDL_t) to an output ($WURR_t$).

Similar logic is applied to damage resulting from floods, where current monthly total damage (originally calculated from the hydrologic model component operating on a six hour time step and aggregating the results) is compared against a monthly average damage. This ratio of current to average damage is related to investment initiatives undertaken to rebuild and/or repair the damage. The functional relationship between current to average damage and its investment multiplier is shown on the left plot in Figure 21, and represented mathematically as:

$$FDIM_t = FDIMT(D_t/\overline{D}) \tag{25}$$

where $FDIMT$ stands for the flood damage investment multiplier table. This table relates a damage ratio (damage D in [\$] to average monthly damage $\overline{D}$ in [\$]) to a flood damage investment multiplier $FDIM_t$ in [-]. The logic of such a relationship is as follows: if the ratio of current to average damage increases (i.e., the area sees more flood damage) the investment multiplier will correspondingly increase, reflecting the
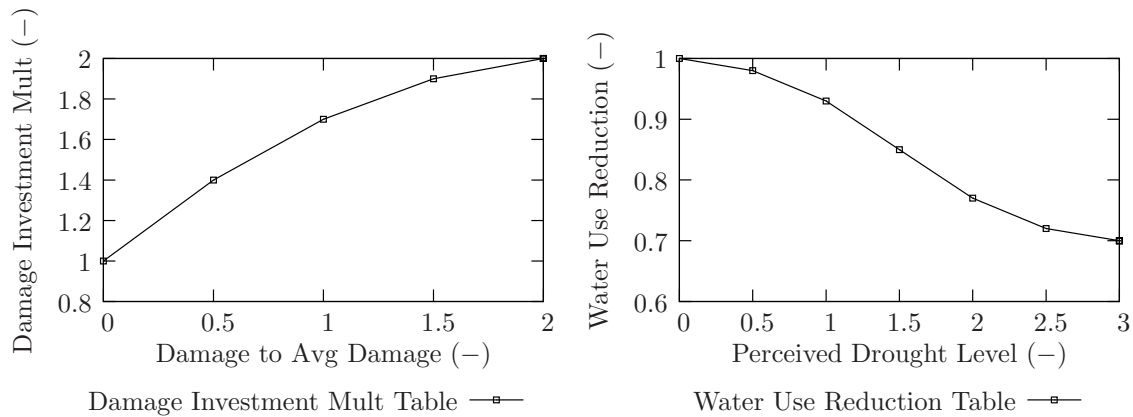
Figure 21: Table functions for flood and drought

undertaking of initiatives designed to rebuild, repair and/or replace assets damaged from a flood. Note that the function for this multiplier never dips below unity, and thus assumes that decisions to recover from flood damage are always initiated. By modifying this function we could easily build into the model an alternate formulation where increasing flood damage could force an actual decline of future investment in business. This would imply that flood investment multiplier curve would dip below unity for high levels of damage, thus discouraging future business investment.

The flood damage investment multiplier is also put through an information delay implying that it takes time to assess damage and come up with extra funds to rebuilt and/or repair the damage. Such a formulation takes the following form:

$$PFDIM_t = \text{Smooth3}(FDIM_t, DmDT) \tag{26}$$

where $PFDIM_t$ stands for the perceived flood damage investment multiplier in [-], and $DmDT$ for damage delay time in [yrs]. In the simulation model the flood damage is shown in the form of spikes (i.e., high damage amounts during short time intervals), while investment multiplier (delayed) provides smoother function where investment is spread over a longer time period.

Same logic is applied for the variable ground water recharge, obtained from the hydrologic model component (bolded in Figure 20). As current ground water conditions are never known precisely, an information delay is applied to this variable as well. (Information delays also filter out fluctuations, thus providing noise free signal for the quantity of interest.) Perceived ground water recharge $PGWR_t$ [10^6 m^3/yr] is therefore represented by:

$$PGWR_t = \text{Smooth3}(GWR_t, GWDT) \tag{27}$$

where GWR_t represents ground water recharge in [10^6 m^3/yr] and $GWDT$ ground water delay time. When multiplied with a fraction of total ground water allocated to urban business (shown in Figure 20 and denoted by $f_{GW}UB$ in [-]), business ground water availability $BGWA_t$ in [10^6 m^3/yr] is produced.

$$BGWA_t = f_{GW}UB \cdot PGWR_t \tag{28}$$

The same reasoning is applied to business surface water availability $BSWA_t$ in [10^6 m^3/yr], defined as maximum surface water allowed to be pumped from Lakes Huron and Erie MAP in [10^6 m^3/yr], multiplied by a fraction allocated to urban business $f_{SW}UB$ in [-].

$$BSWA_t = f_{SW}UB \cdot MAP \tag{29}$$

All water allocation fractions used in the model are calculated based on current use practices (see Table 3 in Section 3). Lastly, in order to estimate average ground and surface water use per business structure, we take the total respective water use

and divide it by the initial number of business structures (defined later).

$$GWUBS = BGWU_{2001}/BS_0 \qquad SWUBS = BSWU_{2001}/BS_0 \qquad (30)$$

where $GWUBS$ and $SWUBS$ stand for ground and surface water user per business structure, in $[10^6$ m^3/yr/unit]; $BGWU_{2001}$ and $BSWU_{2001}$ represents total ground and surface water used in year 2001 in $[10^6$ m^3/yr], and BS_0 for initial number of business structures.

The rest of the urban business activity sector is shown in Figure 22, where italicized variables represent information obtained from outside (either from other sectors, or other figures). The main state variable representing economic activity of the region is a business structure. Its definition is identical to one provided in the work Alfeld and Graham (1976), where the authors explore a number of different definitions of economic activity, but in the end select the business structure as the best one (p. 10). According to the authors, a number of businesses does not describe regional economic activity well because a single large company can create more activity than many smaller ones. A monetary measure is also not satisfactory, as this makes it difficult to relate economic activity to employment, migration, land use, and subsequently, taxes.

A definition of a business structure as building space occupied by a business is therefore adopted because of its relative ease in linking it to other important economic parameters. In today's business world building space (whether part of an office, warehouse, or a farm) requires a considerable investment of capital. In a viable business, an initial investment of capital usually generates more invested capital, thus potentially growing a business. With this definition, a large corporation (such as a manufacturing company or a university) employing a large number of people and

Figure 22: Level and rate diagram of the urban business activity sector

occupying a sizable land area is counted as a number of business structures (exactly how much depends on number of employees, and land area occupied). Growth of business activity, as it is defined here, takes place when area creates a demand for investment and construction of new business structures. Similarly, an increase in the rate of demolition and depreciation of business structures creates an economic down turn by eliminating places of employment, therefore lowering the number of jobs in the area. Business construction and demolition are therefore shown as rate variables in Figure 22. The relationship between the state and rate variables in this sector is represented by the following equation:

$$BS_t = BS_{t-1} + (BC_t - BD_t)\Delta t \tag{31}$$

where BS_t represents a number of business structures in [units], and BC_t and BD_t stand for business construction and demolition, respectively in [units/yr].

The main variable regulating dynamics in the business activity sector is business construction multiplier (BCM_t); it is defined as a product of business land (BLM_t), business labour force ($BLFM_t$), business water availability ($BWAM_t$), and the perceived flood damage multiplier ($PFDIM_t$).

$$BCM_t = BLM_t \cdot BLFM_t \cdot BWAM_t \cdot PFDIM_t \tag{32}$$

The multiplier effect works on the premise that any one of its dependent variables has the potential to significantly affect its outcome. Consider an example where water resources are depleted in an area, but labour force, jobs and business land are plentiful, and floods are not an issue. Despite the fact that other conditions are favourable, depleted water resources imply that growth can not be sustained. Similar

logic is applied for other combinations of the above variables (i.e., businesses can't generate additional structures if either labour force, land or water is unavailable).

The urban business activity sector sector consists of a number of feedback loops, which are discussed next. Water use in this sector (the same logic is used in all other sectors) is divided into surface and ground water uses. Each business structure is assumed to use a certain amount of water (both from surface and ground sources in Middlesex County, or from ground water alone in Oxford and Perth Counties). Combining water use per business structure ($SWUBS$, $GWUBS$) with the total number of structures (BS_t), multiplied with a water use reduction ratio $WURR_t$, results in variables named business surface ($BSWUD$) and ground ($BGWUD_t$) water use desired (see Figure 22). By dividing desired water use ($BSWUD$, $BGWUD$) by water availability ($BSWA_t$, $BGWA_t$) for each use type, desired business use to availability ratios are computed. Under simulation conditions emphasizing high growth it was discovered that desired water use to availability ratios can actually become larger than unity, a physical impossibility. In order to deal with situations such as this, a fuzzy minimum function is used to gradually convert desired use to availability ratios to (actual) use to availability ratios that never exceed unity. The logic behind this formulation is described in detail in Chapter 14 of Sterman (2000).

Water use effect is then computed for both use types from its respective use to availability ratios with a relationship shown on top left in Figure 23 and represented as:

$$BSWUE_t = WUET(BSWU_t/BSWA_t) \tag{33}$$

$$BGWUE_t = WUET(BGWU_t/BGWA_t) \tag{34}$$

where $BSWUE_t$ ($BGWUE_t$) stands for business surface (ground) water use effect;
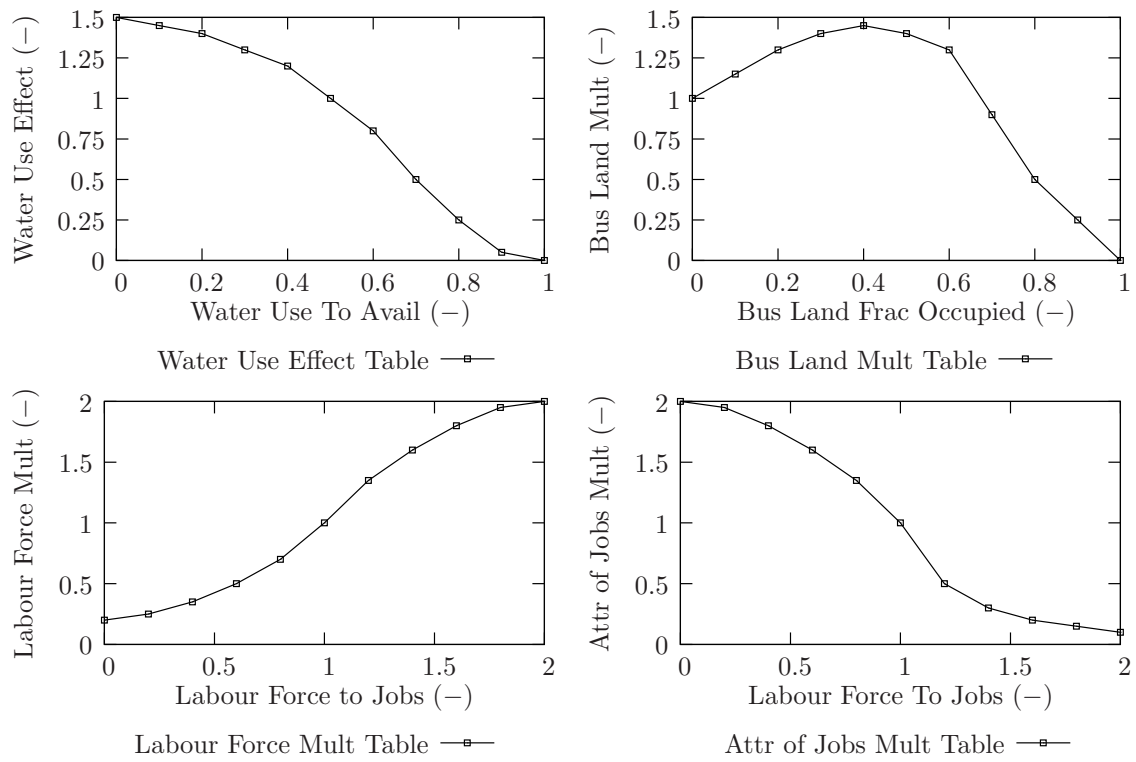
Figure 23: Table functions of the business activity sector

$WUET$ for water use effect table.

The logic of this relationship implies that as water use to availability ratios increase (and more available water gets used), further industrial growth can not be sustained because local governments are assumed to raise costs of servicing and delivering water to discourage growth that can potentially make the area run out of water. Use effects from surface and ground water are combined into a variable called business water availability multiplier ($BWAM$) by a weighted average, which is then used as one of the components in the business construction multiplier previously defined.

Business structures in an area are assumed to occupy some parcel of land. By having a number of business structures BS_t in an area, together with an amount of land occupied by each structure LBS [km^2/unit], we can compute a total amount of land occupied by business structures BLO_t in [km^2].

$$BLO_t = BS_t \cdot LBS \tag{35}$$

Next, the model obtains a total area of land zoned for business purposes (BL_t in [km^2]) from a land use sector (described later), from which a fraction of total business land occupied ($BLFO_t$) is calculated.

$$BLFO_t = BSO_t/BL_t \tag{36}$$

This ratio is used next to compute a business land multiplier, and thus influence construction of new business structures. Its relationship is shown in Figure 23 on top right and its rationale is the following: Small fractions of occupied land imply its wide availability; in such cases further increasing land occupancy leads to increased business activity. In other words, it is attractive for businesses to move to such area as

servicing (water, electricity, sewer, etc.) is assumed to have been already completed, with the added bonus of having plenty of available land for its operations. However, as a fraction of occupied land grows, multiplier effect is reduced to reflect "inhibiting effects on business construction of rising land prices and diminishing choices of location" (Alfeld and Graham, 1976, p. 38). This also suggests that denser occupancy leads to higher taxes, limited availability of storage and/or parking, as well as higher operating costs thus discouraging construction of new business structures. Eventually, as all land gets occupied, growth must be reduced to zero.

Another feedback loop regulating the behaviour of the business activity sector is one relating to the availability of area's labour force. Each business structure is assumed to employ a number of people (JBS in [jobs/unit]), which, when combined with a total number of business structures (BS_t [units]), produces a total amount of urban jobs (UJ_t [jobs]) in the area.

$$UJ_t = JBS \cdot BS_t \tag{37}$$

A total urban labour force (ULF_t [people]) is calculated by taking urban population (UP_t [people]) and multiplying it by an urban labour force participation factor $ULFPF$ in [-].

$$ULF_t = UP_t \cdot ULFPF \tag{38}$$

After this, by dividing total labour force (ULF_t) to available jobs (UJ_t), a variable labour force to jobs ratio is obtained ($ULFJR_t$).

$$ULFJR_t = ULF_t/UJ_t \tag{39}$$

This variable is used in a multiplier that regulates business construction through

the functional form given on bottom left in Figure 23. In this relationship, low labour force to jobs ratio implies that an area has less available labour than jobs, thus having the potential to hamper business investment (i.e., people just can not be found to fill available positions). On the other hand, high ratios of labour force to jobs in an area mean that there exists a labour force which exceeds available jobs. In such conditions businesses are assumed to react positively, as they can pick and choose among a large pool of available skilled workers to fill the needed positions.

Labour force to jobs ratio ($ULFJR_t$) is also used to determine an attractiveness of jobs ($AUJM_t$), used later by the population sector to regulate the number of people moving to an area. The relationship between labour force to jobs ratio and a jobs attractiveness multiplier is shown on bottom right in Figure 23, where small labour force to jobs ratio imply that labour force is smaller than available jobs—in other words the area has more jobs than people to fill them. Although not attractive for business investment, this is assumed to be an attractive feature for the population, which is expected to increase in migration to that area (i.e., people will see favourable conditions and move to that area). Under conditions of high labour force to jobs ratio (where labour force is greater than available jobs), the area stops being attractive for new families because the prospect of finding jobs becomes grim.

5.1.2   Urban Housing

Viability of an urban housing sector is recognized as an important determinant to regional well being. Adequacy of housing strongly affects regional population dynamics (without available housing people can not live or work in an area). Housing also influences regional economic health, where lack of housing means lack of people, reducing a potential labour force of skilled workers, thus halting potential growth of the economy. The housing sector used in this work is taken directly from the work

of Alfeld and Graham (1976). Systemic structure of the sector consists of a stock variable representing urban houses (UH_t [houses]), modulated by rates representing housing construction (HC_t [houses/yr]) and demolition (HD_t [houses/yr]), respectively. The level and rate diagram of the housing sector is shown in Figure 24, and is represented by:

$$UH_t = UH_{t-1} + (HC_t - HD_t)\Delta t \tag{40}$$

The feedback structure of the sector includes loops representing housing and land availability. The principle for determining the housing land multiplier is identical to one discussed in the business activity sector with the only exception that now houses are used instead of business structures. Therefore, the details will not be repeated here. The second feedback loop regulating dynamics is one that involves housing availability, and is described next. Knowing the number of houses in an area (UH_t [houses]), an average household size (HS [persons/household]), and the population (P [people]), a households to houses ratio (HHR_t [-]) is computed (it is simply a ratio between population and the product of houses and household size).

$$HHR_t = P_t/(UH_t \cdot HS) \tag{41}$$

According to Alfeld and Graham (1976), "the households to houses ratio represents an aggregate of more tangible variables that respond to housing market conditions, including price, vacancy rates, maintenance and service, finder's fees, and flexibility in choice of location (p. 153)." Similarly to the labour force to jobs ratio, households to houses ratio is used to provide a signal to the housing industry thus modulating the number of houses within an area by directly influencing housing construction.

The housing availability multiplier is a variable that receives the signal provided
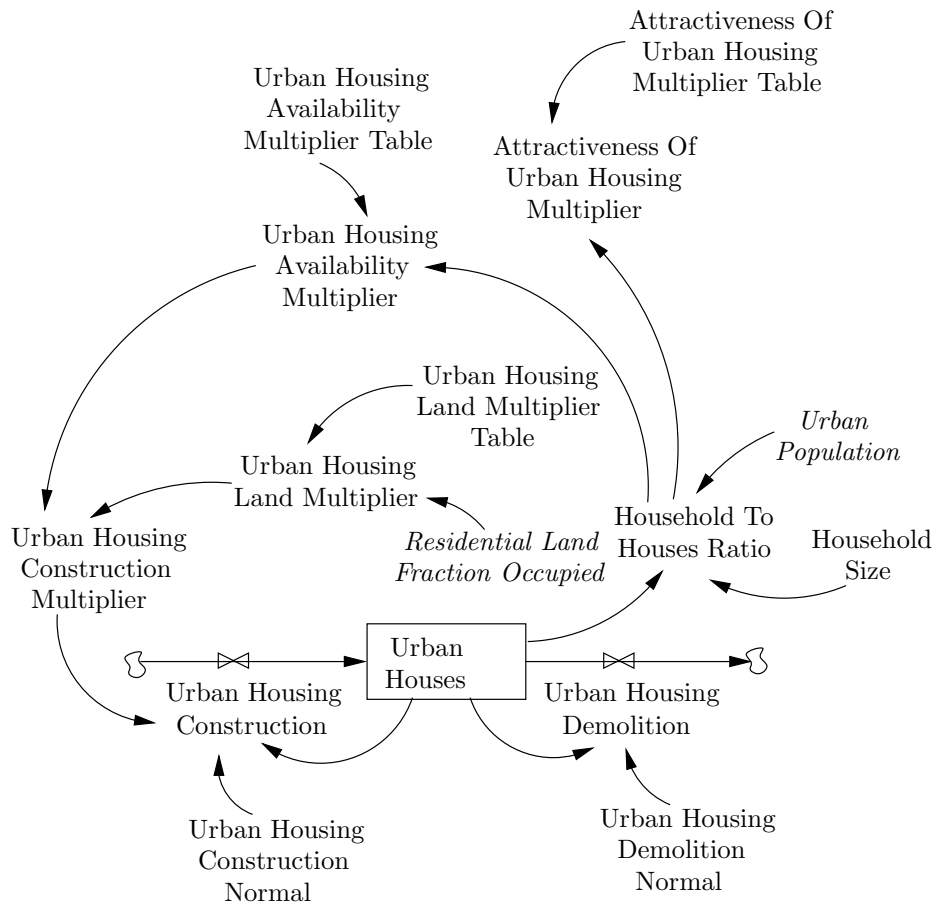
Figure 24: Level and rate diagram of the urban housing sector
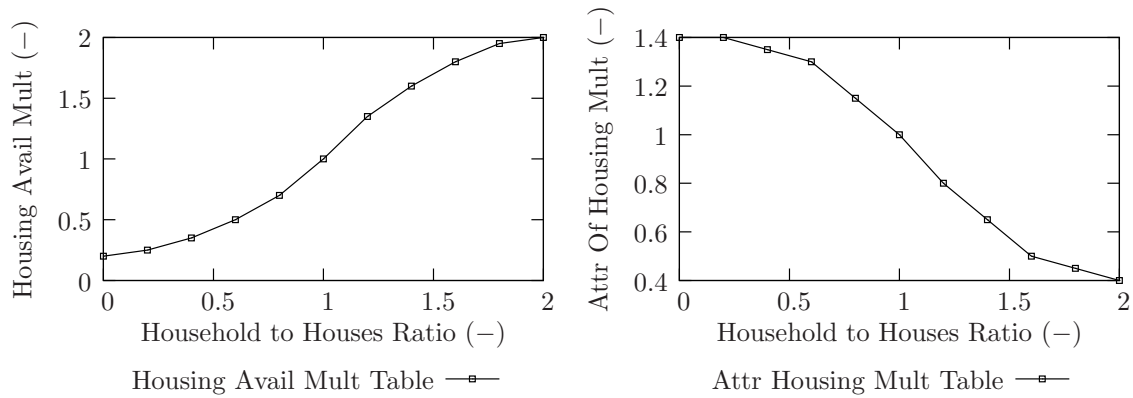
Figure 25: Table functions of the housing sector

from the households to houses ratio and uses it to compute a multiplier effect directly impacting housing construction. Its functional form is shown on the left plot in Figure 25. Low households to houses ratios imply that an area has more available houses than households, thus indicating plentiful housing conditions. Such cases therefore provide signals to the housing construction industry that more houses are not needed, and no additional funds are invested in construction. High households to houses ratios on the other hand, imply that the region has more households (i.e., families) than available homes, thus signalling the need for additional housing construction. Of course, an assumption is made that housing construction multiplier has a saturation effect, meaning that after a certain point no matter how high the demand for new houses, construction can take place only at some maximum rate.

Similarly, attractiveness arising from local housing conditions are computed from the same households to houses ratio. This attractiveness (shown on the right plot in Figure 25) is based on the following: Low households to houses ratio imply that an area consists of more available houses than it needs, thus making it attractive for those wishing to move there. On the other hand, high households to houses ratios stand for conditions where households (i.e, families) outnumber existing houses, thus
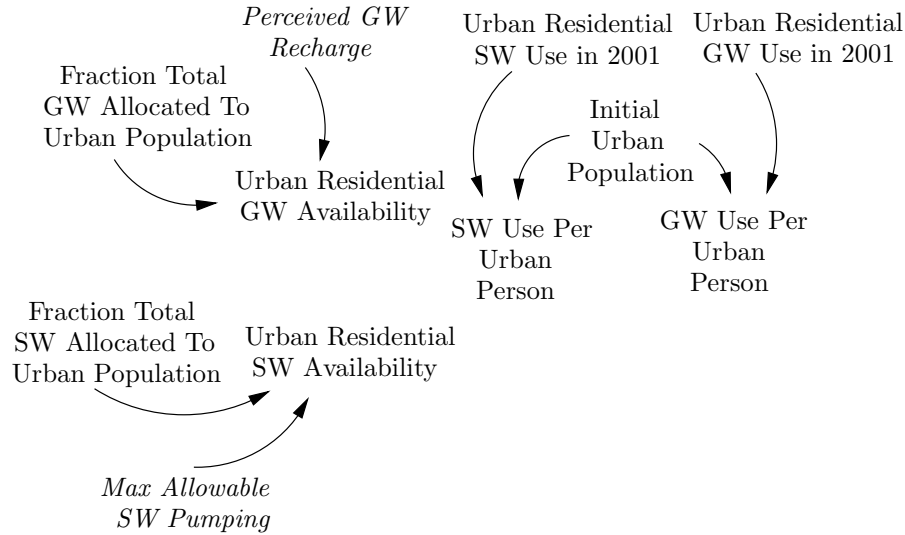
Figure 26: Auxiliary variables of the urban population sector

making it unattractive for newcomers. The same saturation effect is assumed in this relationship as well.

5.1.3 Urban Population

Urban population dynamics are described in this sector, which contain a number of variables previously discussed. The input variables obtained from elsewhere in the model (shown in italics) are depicted in Figure 26, and represent an identical formulation to one already discussed in the urban business activity sector. The level and rate diagram of the urban population sector is shown in Figure 27, and is described next.

The structure of the population sector consists of a state variable denoting a current level of urban population (UP_t [people]), with its rates being in (UIM_t) and out (UOM_t) migration, as well as birth (UB_t) and death (UB_t) rates, all in [people/yr]. Normal values of all variables are chosen from the 2001 Canadian census STATCAN (2001). The births and in migration have the ability to increase the population,
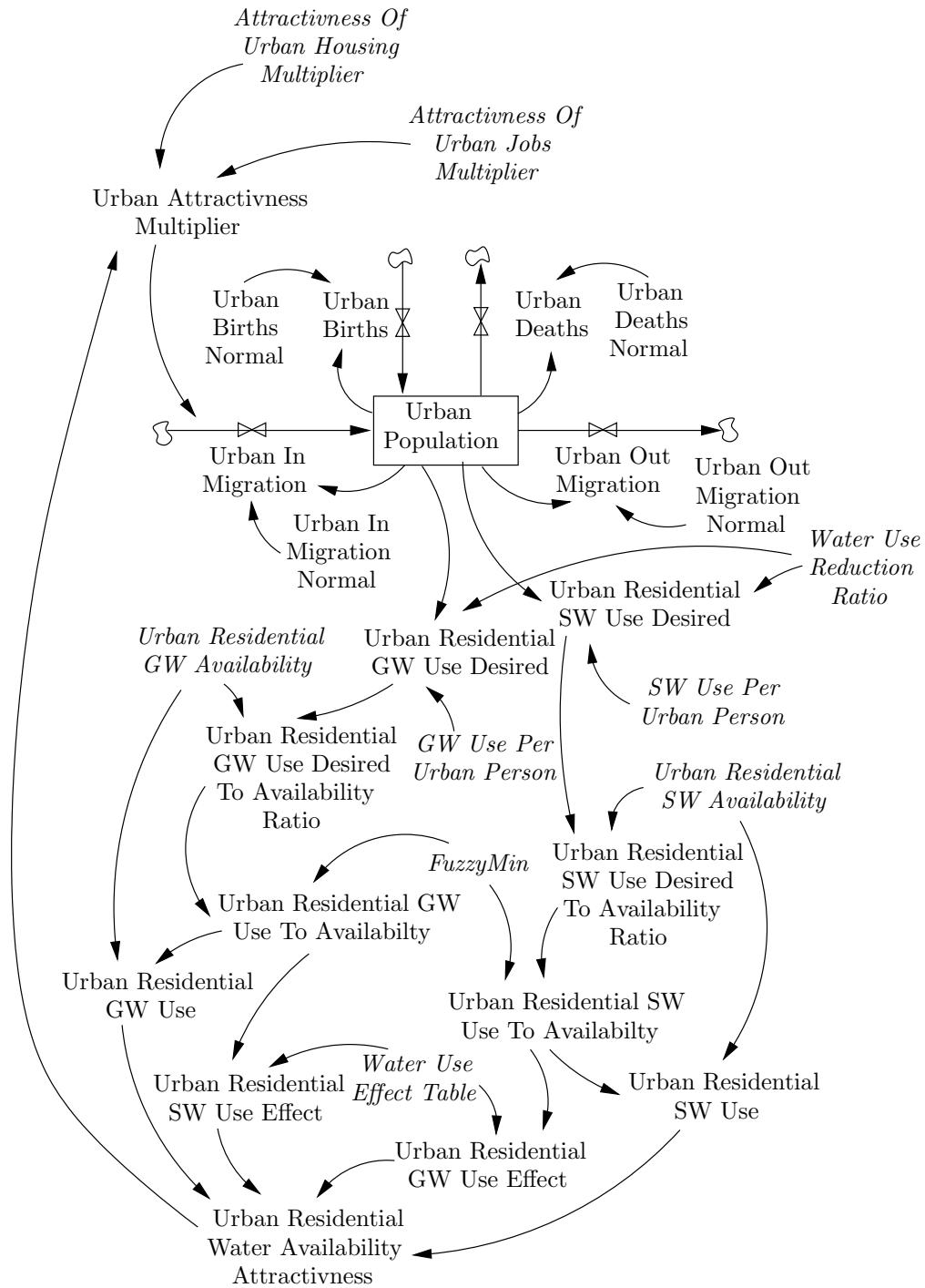
Figure 27: Level and rate diagram of the urban population sector

while out migration and deaths can reduce the population. This is depicted with the following equation:

$$UP_t = UP_{t-1} + (UIM_t + UB_t - UOM_t - UD_t)\Delta t \qquad (42)$$

The urban attractiveness multiplier is the only variable that has the ability to regulate the behaviour of the population sector. It consists of attractiveness multipliers representing availability of housing, jobs, and water. The structure used to determine the water availability multiplier is identical to one described above in the business activity sector, and thus will not be repeated. The only difference is that variables names are changed (where the word business is replaced by residential). The attractiveness of urban housing and urban jobs multipliers have been also described above, in the housing and business activity sectors, respectively. The urban attractiveness multiplier (a combination of of housing, jobs and water availability multipliers) is used to modulate the urban in migration. Thus, if favourable conditions exist, the in migration will cause the population to grow; if they are not, the rates of in migration will be reduced thus potentially lowering the population.

5.1.4   Rural Business Activity

The rural business activity sector parallels its urban counterpart, and uses the same principles in its formulation. The auxiliary variables of this sector are shown in Figure 28, and the level and rate diagram is presented in Figure 29. Some differences, however, are discussed next.

The state variable describing the rural business activity sector is selected to be a farm unit, defined similarly as a business structure. A farm unit is assumed to occupy a specified land area, which consists of a house (where its occupants reside) and buildings needed for agricultural operations (buildings housing grain/feed, live-

Perceived GW
Recharge

Fraction Total
GW Allocated To
Farm Units

Farm GW
Availability

Fraction Total
SW Allocated To
Farm Units

Farm SW
Availability

Farm SW
Use in 2001

Farm GW
Use in 2001

Initial
Farm
Units

Max Allowable
SW Pumping

SW Use Per
Farm Unit

GW Use Per
Farm Unit
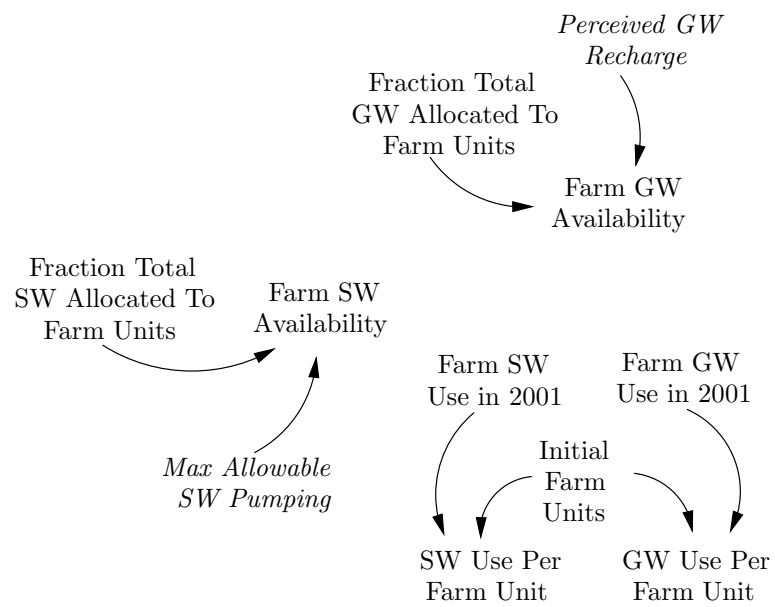
Figure 28: Auxiliary variables of the rural business activity sector
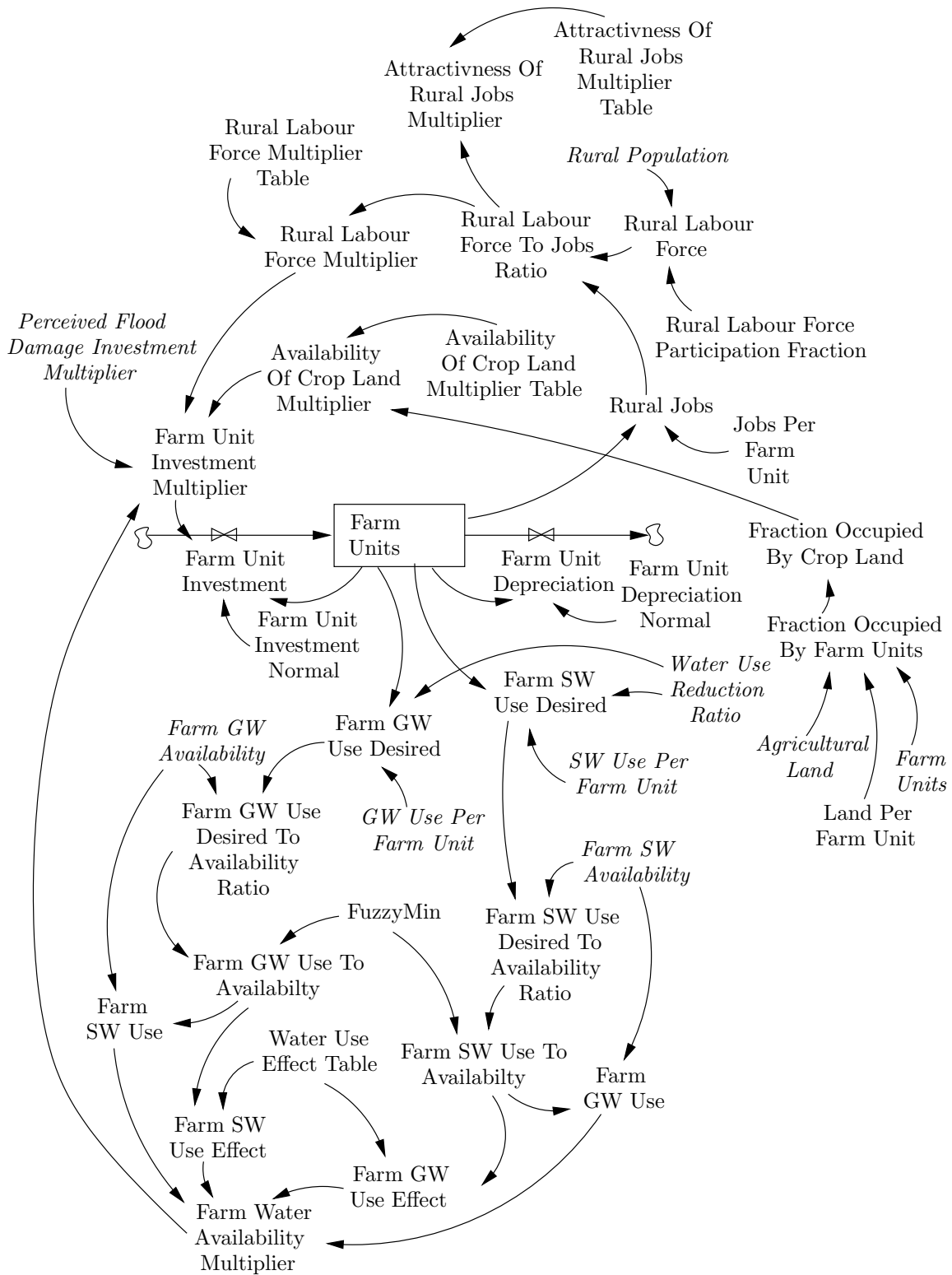
Figure 29: Level and rate diagram of the rural business activity sector

stock, equipment, etc). A farm unit thus consists of an aggregate unit responsible for agricultural operations. The total number of farm units can be modulated by investment (i.e., construction of more farms) and reduced by depreciation (where old farm structures are demolished).

The farm unit investment multiplier is defined in the same manner as its counterpart in the urban business activity sector. As before, it has been formulated to depend on availability of water, land, labour force, as well as the perceived flood damage investment multiplier (see Figure 29). The water use reduction ratio (previously calculated from inputs of the hydrologic model component) is applied to the desired water use, and has the effect of reducing water usage rates in response to regional drought conditions. The attractiveness of rural jobs is defined in the same manner as its urban equivalent, and thus provides a signal to the rural population sector.

5.1.5   Rural Population

The rural population sector is identically defined as its urban counterpart; it is shown in Figures 30 and 31 for completeness. Since a rural housing sector is not explicitly defined (it has been aggregated into a definition of farm units), a separate attractiveness multiplier is therefore not present in the formulation of this sector. The rural population therefore responds to two attractiveness multipliers, namely water availability attractiveness ($RWAA_t$) and attractiveness of rural jobs ($ARJM_t$), defined in the same way as its urban equivalents. All feedback relationships in this sector are indistinguishable in structure from ones used earlier, with the only difference that variable names have been changed, and populated using different initial conditions.

Perceived GW
Recharge

Rural Residential
SW Use in 2001

Rural Residential
GW Use in 2001

Fraction Total
GW Allocated To
Rural Population

Initial
Rural
Population

Rural Residential
GW Availability

SW Use Per
Rural
Person

GW Use Per
Rural
Person

Fraction Total
SW Allocated To
Rural Population

Rural Residential
SW Availability

Max Allowable
SW Pumping
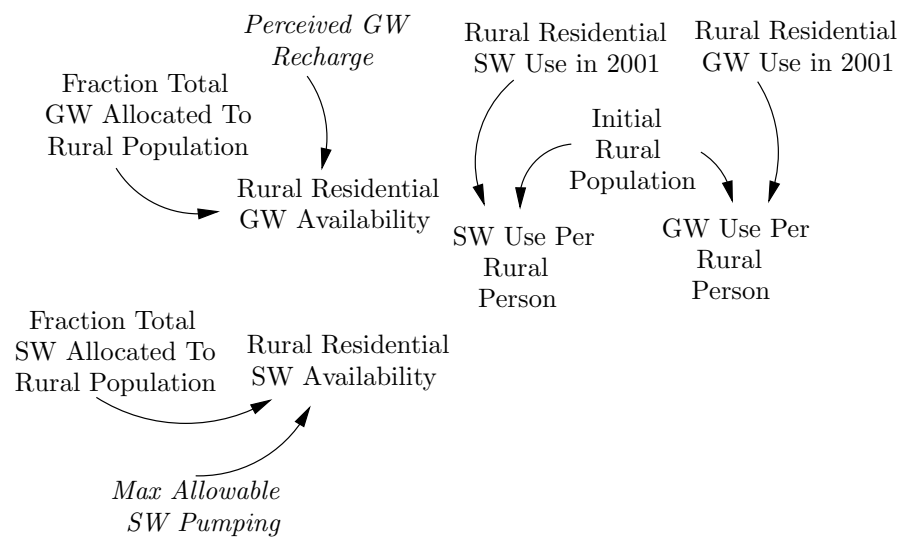
Figure 30: Auxiliary variables of the rural population sector
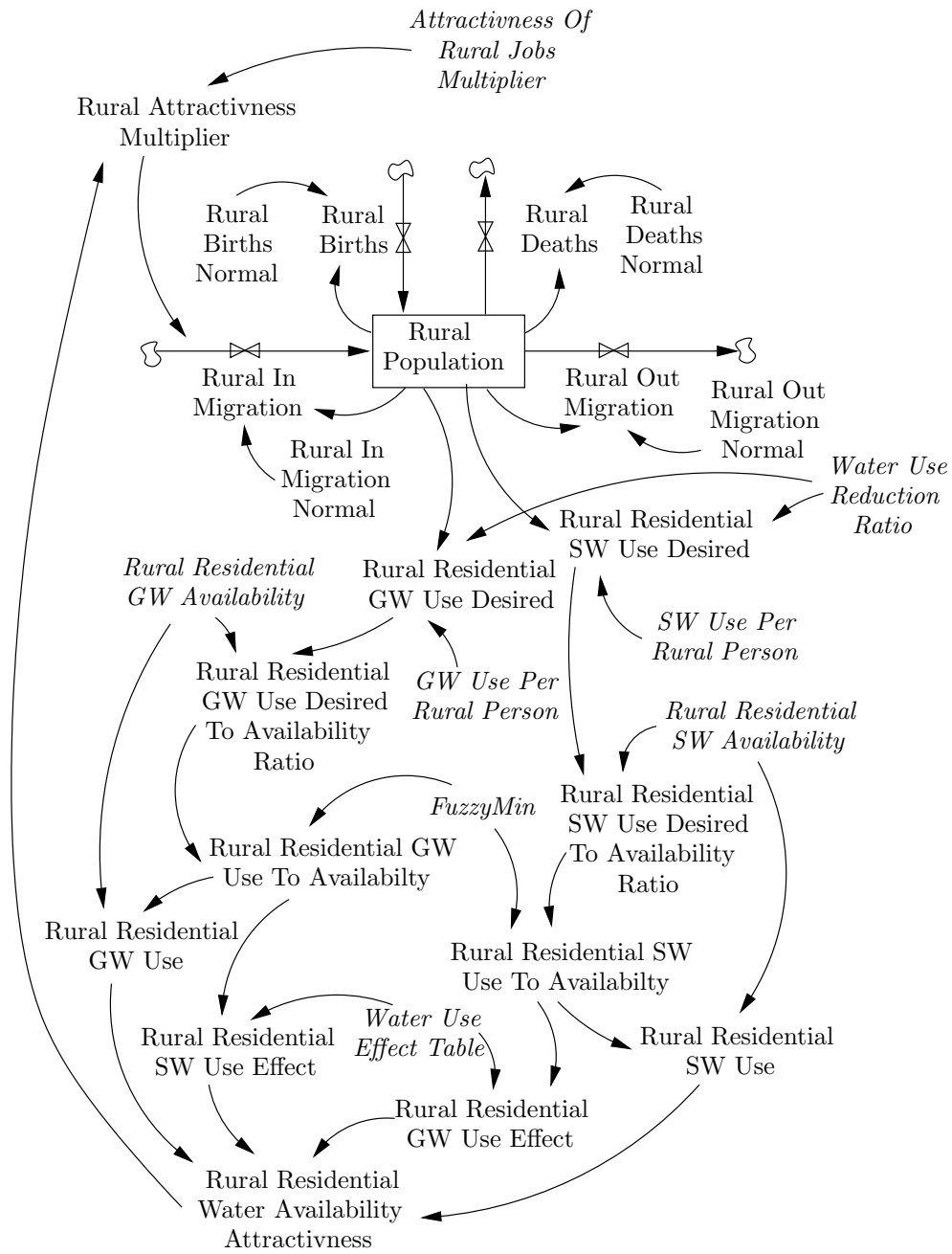
Figure 31: Level and rate diagram of the rural population sector

5.1.6 Land Use

The land use sector consists of the most detailed component of the socio-economic model component; it is formulated to respond to inputs from all other sectors. This model sector is an extension of the REZONE2 model, originally developed by (Schroeder et al., 1975, p. 134). The inputs received in the land use sector are those from urban and rural communities, and include pressures from each to rezone more land to suit its respective needs. For example, if the business community finds itself in need of additional land for development, it will place pressures on the local government and its land use planners to allow such zoning. In the structure of the land use sector, a conversion of land use types can come from a number of different sources. The land use sector is divided into four categories (see Figure 19): forest cover (FC_t), agricultural (AL_t), residential (RL_t) and business land (BL_t) all in [km^2].

The auxiliary variables of the land use sector are shown in Figure 32, which describe the competition structure between residential and business land uses. This is needed in cases if an area develops majority of it surrounding agricultural lands, or if conversion of such lands become strictly regulated. The variable residential land fraction rezoned ($RLFR_t$ [-]) is defined as a quantity that describes a fraction of residential land converted each year to business uses; likewise, the fraction of business land rezoned ($BLFR_t$ [-]) corresponds to a fraction of total business land converted to residential land each year. Residential land fraction rezoned depends on a normal rate of such conversion ($RLRN$ [1/km^2]), and multiplier effects named residential land availability multiplier ($RLAM_t$ [-]) and residential land rezoning multiplier ($RLRM_t$ [-]). Similar logic is applied to conversion of business to residential uses.

$$RLFR_t = RLRN \cdot RLAM_t \cdot RLRM_t \tag{43}$$
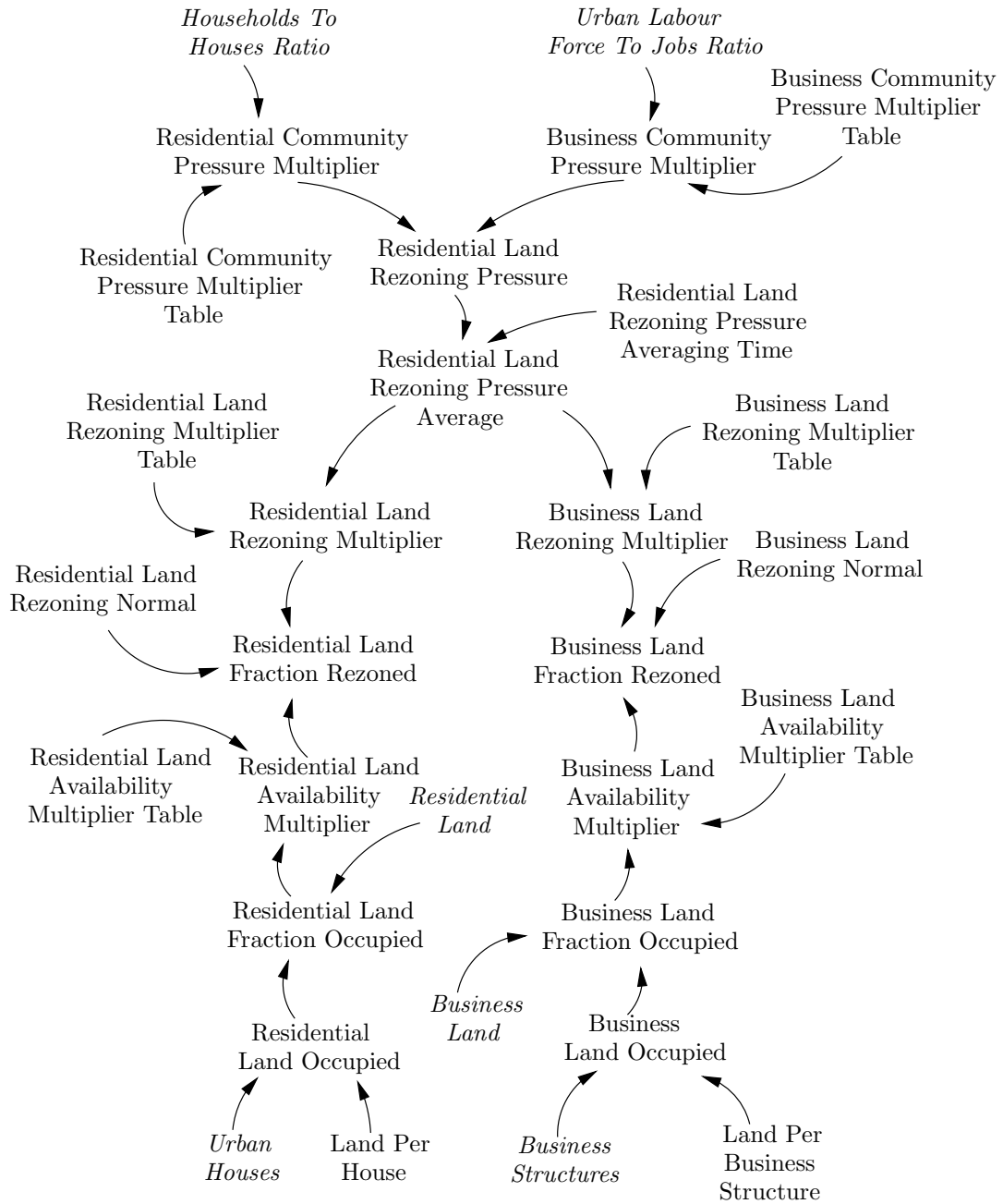
Figure 32: Auxiliary variables of the land use sector

Located near the top of Figure 32 are the variables that describe community pressure that eventually leads land use planners to change the land zoning. The residential community pressure multiplier ($RCPM_t$ [-]) is a variable that generates such pressure for the conversion of residential land to business uses. It responds to the households to houses ratio in the following manner (see top left plot in Figure 33): for increasing values of the ratio (meaning the area has more families than houses) the pressure to rezone residential land to business uses decreases. The business community pressure multiplier ($BCPM_t$ [-]) is defined in the same way. Top right plot in Figure 33 shows the relationship between an urban labour force to jobs ratio and its corresponding business community pressure multiplier. For increasing labour force to jobs ratios (implying a shortage of jobs generating the need for additional business land on which new business structures can be built thus generating jobs), the pressure to rezone residential land to business uses will correspondingly increase.

The residential and business multipliers describing pressures to rezone residential land ($RCPM_t$ and $BCPM_t$) are thus combined to form the variable residential land rezoning pressure ($RLRP_t$ [-]).

$$RLRP_t = RCPM_t \cdot BCPM_t \tag{44}$$

Since it takes considerable time for the community pressure to actually convert land from one type of use to another, an information delay is thus applied to the residential land rezoning pressure.

$$RLRPA_t = \text{Smooth3}(RLRP_t, RLRPAT) \tag{45}$$

where $RLRPA_t$ stands for residential land rezoning pressure average [-], and $RLRPAT$
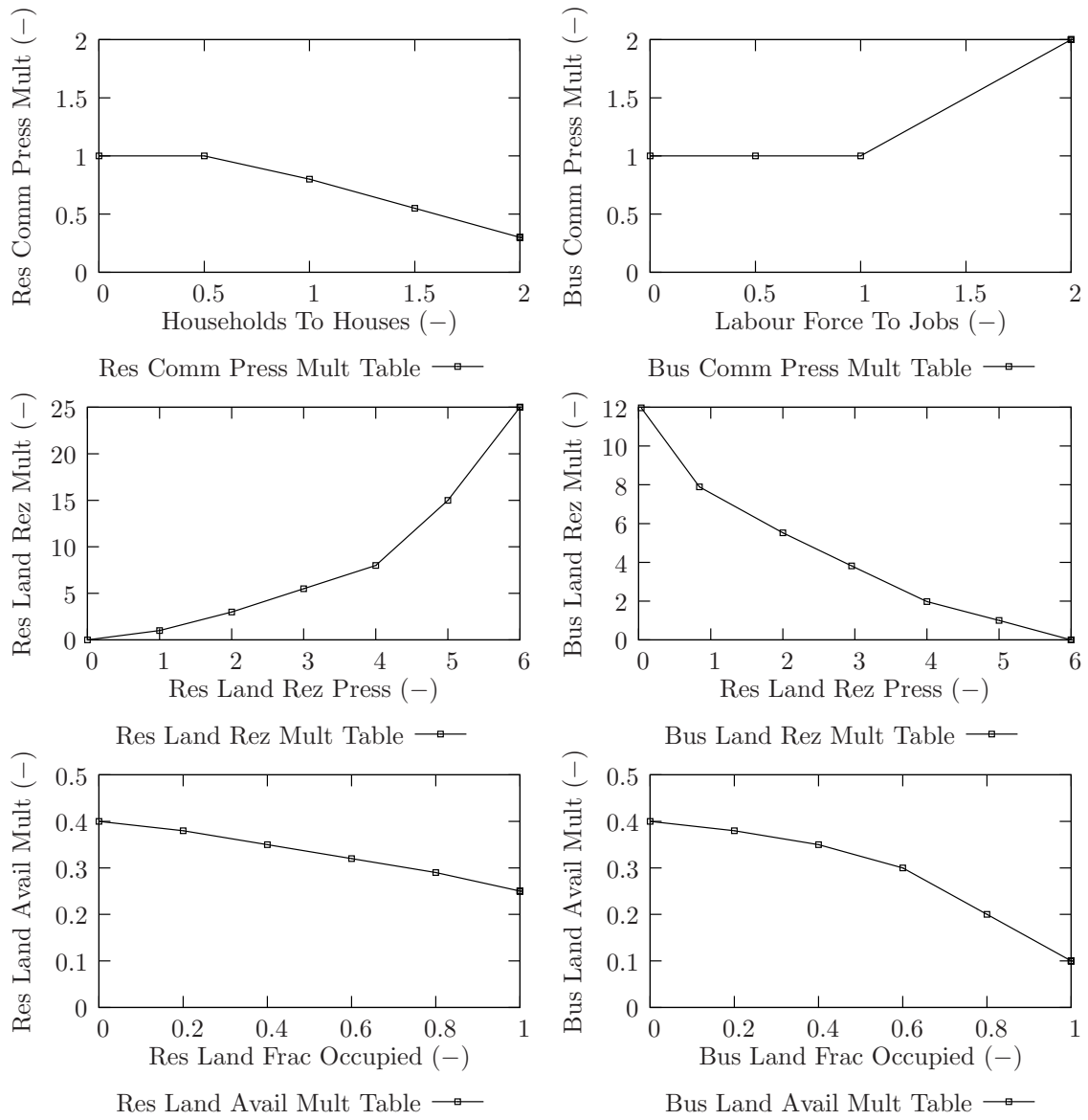
Figure 33: Table functions of the land use sector, part 1

for residential land rezoning pressure averaging time [yrs]. The delayed variable is one that helps determine the residential (and business) land rezoning multipliers. The relationship between the delayed pressure, and residential land rezoning multiplier is shown in the middle left plot in Figure 33. As the average pressure for the residential land to get rezoned increases, more residential land actually gets rezoned. Similar principle is used for the business land rezoning multiplier, shown in the middle right plot of Figure 33, where increased pressure to convert more residential land reduces the demand to rezone business land to residential use.

The variable residential land fraction rezoned also depends on the land that is actually available for rezoning. If land is occupied, the process of rezoning that land can not take place until the structure occupying it (whether it be a business, house or a farm unit) is demolished. Therefore, the residential (and business) land fraction rezoned also depends on the availability of land. This is discussed next.

Knowing the number of houses (UH_t [units]), and a land area that is taken up by each house (LPH [km^2/unit]), we can calculate a total area of residential land occupied (RLO [km^2]).

$$RLO_t = UH \cdot LPH \tag{46}$$

The amount of residential land occupied (RLO_t) is divided by the total land zoned for residential purposes (RL_t), to obtain a variable called residential land fraction occupied ($RLFO_t$):

$$RLFO_t = RLO_t / RL_t \tag{47}$$

The residential land availability multiplier ($RLAM_t$) is obtained from inputs of residential land occupied. The relationship for this multiplier is shown in the bottom left plot in Figure 33, and uses the following logic: as more residential land gets occupied, less is available for rezoning to business use types. An identical procedure is

employed when specifying the relationship for the business land availability multiplier $(BLAM_t)$, shown on the bottom right plot in Figure 33, where business land fraction occupied is obtained using the same way as its residential counterpart.

Residential and business land fraction rezoned ($RLFR_t$ and $BLFR_t$) are multiplied by the their corresponding state variables (residential (RL_t) and business (BL_t) land) to obtain rates of residential and business land rezoning (RLR_t and BLR_t), shown in Figure 34. The rate variables are the final outcome of the rezoning process between residential and business land, and represents the actual land rezoned each year. The rate of rezoning forest cover to agricultural land similarly depends on the fraction occupied by farm units, and is presented next. The functional relationship is shown in the left plot in Figure 35, and it assumes that after 30% of agricultural land is occupied by farm units, growing pressures develop for additional agricultural land. The actual rate variable forest to agricultural rezoning (FAR_t) is a product between the current level of forest cover (FC_t), a normal rate of conversion $FARN$, and its rezoning multiplier $FARM$. Forest cover in this work is assumed to be able to only be converted to agricultural land.

Agricultural land, on the other hand is allowed to be rezoned to either residential and business uses, depending on their respective occupancy ratios. The functional relationship used in the multipliers describing conversion of agricultural land to residential and business uses is shown on the right plot in Figure 35—the same relationship is used for both types. If more urban land is occupied (business or residential), there will be a growing trend to rezone the remaining agricultural land for urban uses. The relationship is assumed to saturate at high urban occupancy values, implying that a limit exists regarding how much (and how fast) such conversion can occur.
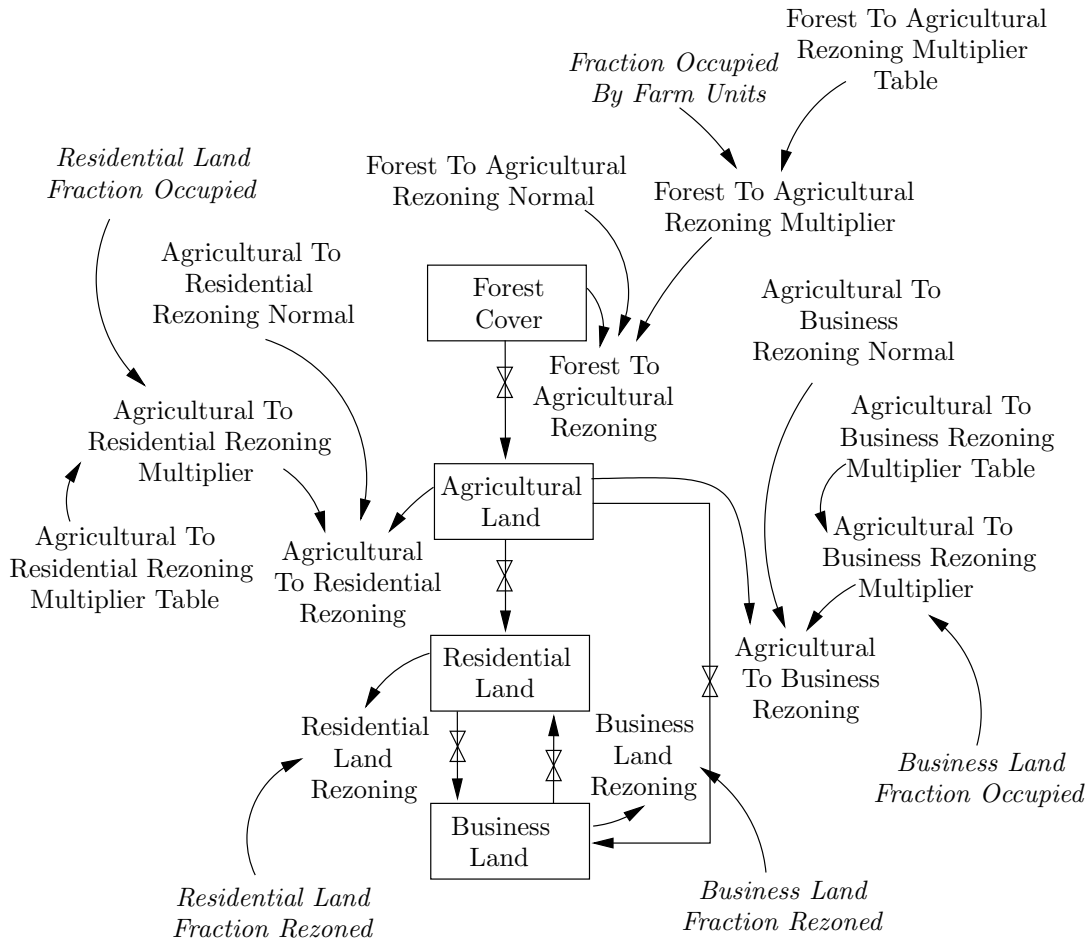
Figure 34: Level and rate diagram of the land use sector
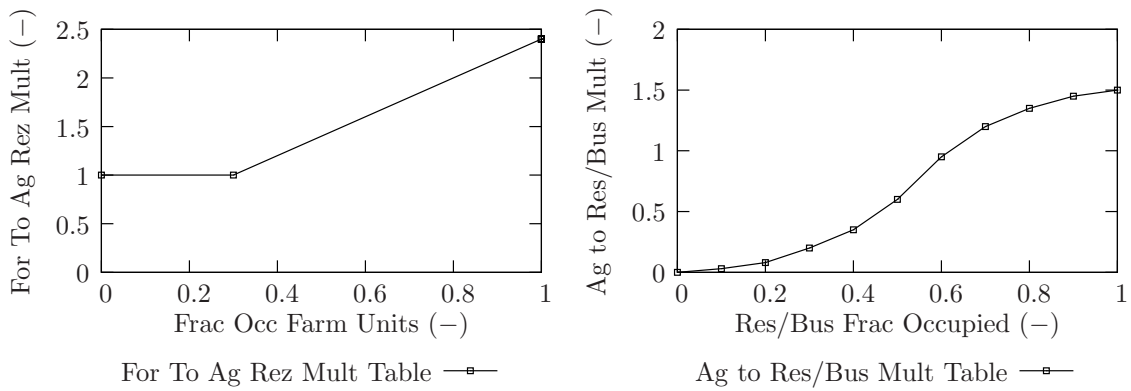
Figure 35: Table functions of the land use sector, part 2

6 Coupling of Model Components

This section describes the main contribution of the work presented in this report, namely to provide a framework where continuous hydrologic (HEC-HMS in our case) and socio-economic (system dynamics) model components are coupled via feedback. In such a modelling framework, physical and socio-economic processes depend on (and are dependent on) one another, thus closely mimicking their real world equivalents. In the area of environmental and water resources management, a number of past studies attempted to capture such feedback links. They include the following:

The research by Li and Simonovic (2002) demonstrates one of the first applications of system dynamics simulation and systems thinking to studying strictly hydrologic processes in North American prairie watersheds. The work of Saysel et al. (2002) develops a system dynamics model for the purpose of studying agricultural development in Turkey. The model consists of sectors representing water resources, land use, demography and agricultural pollution. Stave (2003) on the other hand, uses a very simple model to bring about public understanding of water management options in Las Vegas, Nevada. Problem definition similar to Saysel et al. (2002) is adopted by Fernández and Selma (2004), who studied water scarcity of irrigated landscapes in Spain. Sehlke and Jacobson (2005) proposed a model to investigate the utility of system dynamics approach in modelling large complex hydrologic systems and their interaction between the surface and ground water in the western United States.

Even though the above modelling efforts have been prepared for different purposes, all share one common feature: they study water resources management problems using the framework of systems thinking and feedback. Nearly all employ some sort of interactions between physical and socio-economic realms; this is indeed their largest contribution. However, all of the models share another commonality: they are not

widely accepted by end-users, practitioners and policy makers. There are still many who are unwilling to accept the systemic world view and consider implications it suggests. However, if systems models are constructed with the participation of its intended end-users, utilizing tools they are already familiar with, the outlook can be more promising. This is what the current study wishes to accomplish.

According to the best knowledge of the authors, combining a well known and widely accepted hydrologic model (like HEC-HMS) to a socio-economic model constructed using system dynamics has never been previously attempted. Furthermore, a modelling framework where physical and hydrologic processes influence, and are influenced by socio-economic processes has not been adequately treated in the current literature. The main contribution of this work develops a framework where detailed hydrologic processes are linked (via feedback) to larger scale socio-economic processes thus providing a single unifying framework where a systems model is able to address a variety of management questions and policy implications (including those from the potentially changing climate), upon which future management guidelines and strategies can be based.

6.1 Upper Thames River Basin Model Structure

The overall structure of the socio-economic model component is presented in Section 5.1, with its main sectors and inputs needed from the hydrologic model component. This is done as the socio-economic model component can not be simulated without hydrologic inputs—it is postulated that available water, flood and drought conditions are necessary determinants to regional social and economic well being of the river basin. The hydrologic model component on the other hand, could be run (this is how they are traditionally used) without inputs from the socio-economic model com-

ponent. Doing this implies that change in regional socio-economic character does not affect hydrologic processes. The Upper Thames Systems Model, referred to the model that joins the physical and socio-economic components via feedback, captures processes and produces conditions with more realism than either one of the model components by themselves.

The links between hydrologic and socio-economic model components used in this work are shown in Figure 36 via a causal loop diagram. The left part of the diagram shows feedback structure of the continuous hydrologic model component (and in particular the HEC-HMS Soil Moisture Accounting algorithm), while the right portion points to a select few variables of the socio-economic model component. The model consists of three major links between its components. Each is discussed next.

The hydrologic component of the model provides precipitation and temperature (originally obtained from the weather generator model), alongside with ground water recharge and direct runoff to the socio-economic model component. The links are shown as heavy lines in Figure 36. As mentioned previously, the hydrologic component operates on a six hours time step, while the socio-economic component uses a monthly time step. The two model components interact in the following way (the interested reader is encouraged to look at the code listing provided in the Appendix, where all model rules are unambiguously presented):

Before the simulation is started, the hydrologic and socio-economic components are initialized using latest available data (see Tables 1-3 in Section 3). Then, the hydrologic model component is executed with a time step of 6 hours until the end of the first month. At that time, the hydrologic component provides the following key pieces of information to the socio-economic model component: temperature, precipitation, flood damage, and ground water recharge. Flood damage provided is an aggregate of the damage resulting from the previous month—flood damage is a sum of
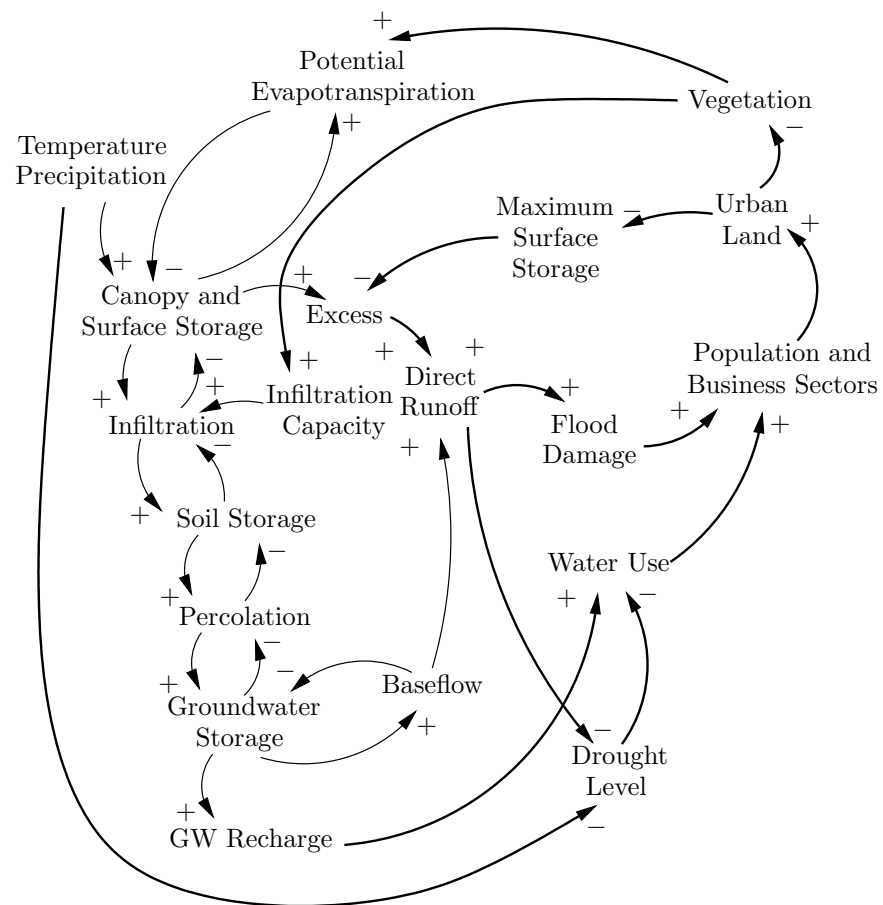
Figure 36: Feedbacks between hydrologic and socio-economic components

damages resulting from each six hour time step during the simulation of that month. Ground water recharge, temperature and precipitation information is communicated to the socio-economic model component, which uses it as input before initial execution. Therefore, at the end of the first month, the hydrologic model component provides the necessary information to the socio-economic model component, which is then executed.

The socio-economic model component then simulates dynamics of the social conditions of the study area, and thus updates it major state (population, land cover, business structures, farms, etc.) and rate variables (in migration, business and farm investment, rezoning rates, etc.). The socio-economic conditions resulting from the execution of the socio-economic component update the physical conditions of the hydrologic component through a number of parameters, embedding the assumption that regional hydrologic conditions dynamically change in light of changing socioeconomic conditions. The updating of the parameters is done before the execution of the hydrologic model (at the start of the second month) and includes the following (see Figure 36): potential evapotranspiration, maximum surface storage, and soil infiltration capacity. The input used to determine the degree of change to the hydrologic parameters come from the land use sector of the socio-economic model. Assumed functions that relate land use variables (such as urban and vegetative land) to hydrologic variables are shown in Figure 37, and discussed next.

The top plot in Figure 37 shows the assumed relationship between the fraction of paved land and effect paved land has on the degree of surface storage capacity (defined earlier in Section 4.3.1). The fraction of paved land is defined as the land area occupied by business structures and houses, divided by the total basin area. The relationship assumes that as a fraction of paved land increases, the surface storage capacity decreases and excess runoff increases. Higher runoff causes an increasing
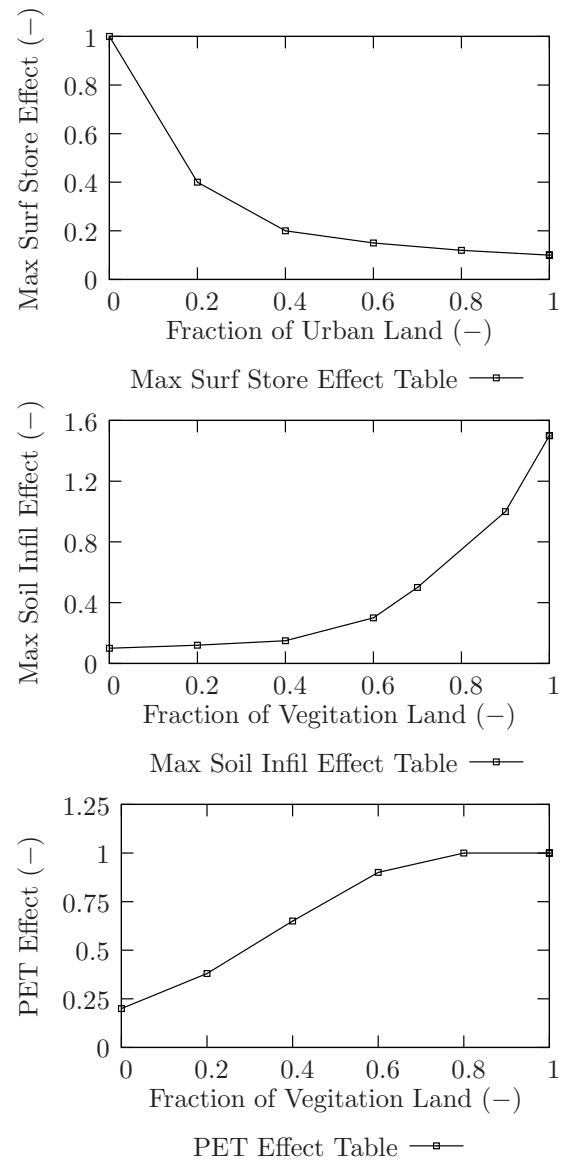
Figure 37: Combined model lookup tables

amount of rainfall (and snowfall) to end up in channels and ponds immediately after the storm, thus reducing the amount of water that can potentially infiltrate into the soil.

The middle plot in Figure 37 similarly shows the relationship between a fraction of vegetation cover (defined as a sum of forest cover and agricultural land divided by the total basin area) and maximum soil infiltration. The infiltration capacity of the soil depends on many factors, some of which are the soil type, moisture content, organic matter, vegetative cover, and season (Linsley et al., 1958). The soil type is by far the most important regulator of infiltration capacity, as soil porosity modulates the resistance of flow. More specifically, increasing soil porosity increases infiltration capacity (for example, infiltration is greater in sandy soils than ones containing large amounts of clay and silt). Vegetation also increases infiltration capacity of the soil, as it tends to increase soil porosity. Furthermore, Linsley et al. (1958) claim that:

> The effect of vegetation on infiltration capacity is difficult to determine, for it also influences [canopy] interception. Nevertheless, vegetal cover does increase infiltration as compared with barren soil because: (1) it retards surface flow, giving the water additional time to enter the soil; (2) the root system make the soil more previous; and (3) the foilage shields the soil from raindrop impact and reduces packing of the surface soil (p. 166).

The assumed relationship shows that as vegetative cover increases, so does the maximum soil infiltration. However, the reverse is also true: as the vegetative cover decreases, the infiltration capacity also decreases, thus increasing surface runoff and even further lowering ground water recharge.

The relationship between vegetative cover and potential evapotranspiration is shown on the bottom most plot in Figure 37. Potential evapotranspiration is a physical quantity typically calculated using empirical equations depending on many

climatic and other factors (for more details see Brutsaert, 1982). For the purposes of this study, it will suffice to say that evapotranspiration heavily depends on vegetation cover. Furthermore, as vegetation cover increases and an area sees more trees, brush, grasses and other plants, the total amount of water lost to the atmosphere from vegetation (i.e., evapotranspiration) also increases. The relationship in bottom plot of Figure 37 is formulated to reflect this fact. The interested reader is referred to Snyder et al. (2004), where more details are provided regarding vegetation cover and its impact on climate.

All relationship shown in Figure 37 have been formulated to capture accurately conditions of the study area. For example, the fraction of vegetative cover in the Upper Thames River basin at the start of the simulation is somewhere close to 0.9; therefore, the hydrologic model component receives a multiplier effect of unity at the start. However, as simulated time goes on and vegetation cover is reduced (through urbanization), soil infiltration capacity and potential evapotranspiration are lowered (by having a multiplier effect of less than unity). The same is true for the effect of maximum surface storage. At current low fractions of urban land cover, the surface storage multiplier effect is close to unity implying no alteration of parameters of the hydrologic model component. However, as the amount of urban land increases, the multiplier effect reduces the surface storage capacity, therefore producing more direct runoff.

6.2 Scenario Analysis in the Upper Thames River Basin

This section presents two types of simulation scenarios used in this work. They include combinations of climate change scenarios (using different climatic signals that regulate regional hydrologic behaviour) and socio-economic scenarios (which test dif-

ferent policies and management practices). Each simulation run of the model consists of a climate scenarios combined with a socio-economic scenario, thus showing impacts not only of climatic change, but also of policies and management strategies adopted to cope with changing climatic and environmental conditions.

6.2.1 Climate Scenarios

Three different climate scenarios are used in this study—the historical (or base case), and two scenarios based on outputs from the global climate models (B11 and B21). The historical (or base or reference) case is generated by simulating the weather generator using the observed record of regional climatic conditions for three variables (precipitation, maximum and minimum temperature) for years 1964-2001. Alternate climate scenarios produced by the weather generator use this historical data, as well as inputs from latest global climate model scenarios. The historical data set is perturbed and shuffled (guided by information provided by the global data), thus creating meteorological conditions not observed historically. Two such scenarios are considered in this study, B21 and B11, and are based on the work of Sharif and Burn (2004, 2006a,b).

The scenarios B21 and B11 are based on IPCC (2001) scenario story lines B2 and B1, reproduced in the Appendix for completeness. B21 and B11 scenarios use the information provided by outputs of CCSRNIES and CSIROM2kb global climate models for the grid cell where the Upper Thames River basin is located. The scenario B21 provides a plausible future where rainfall will tend to increase in both intensity and magnitude over the next century, while scenario B11 illustrates the future where dry spells and droughts become more frequent, in addition to all other environmental and economic forces. The B21 scenario has been specifically designed to test the basin's response to increasing incidents of flooding, while the B11 scenario examines

increasing incidents of drought conditions. By having a historically similar long term record of climate information, together with two widely different scenarios, an attempt is made to capture a range of possible future conditions of importance to policy makers, water resources managers and stewards, stakeholders and the general public.

6.2.2 Socio-Economic Scenarios

Five different socio-economic scenarios are considered in this work, and are briefly described next. The first scenario considered is referred to as the base case scenario. This scenario shows the system behaviour under normal (or no change) conditions. The usefulness of having a base case scenario is mainly for comparison of outcomes with other scenarios. In computer simulation modelling of physical and social systems the goal is usually to estimate behavioural implications of assumptions, decision rules, strategies and policies, that are embedded in the model. In performing scenario analysis, evolution of alternate (and sometimes very different) management polices can be acquired by simply simulating the model. The utility in having a simulation model is therefore derived from testing its different assumptions, policies and management strategies, and comparing such outcomes to a reference (or base case) scenario.

The second scenario tested in the model is one which considers area's water resources to be infinite. Even though this is a rather unrealistic scenario, it is worth testing as many area residents believe that water supply in the region can never be exhausted. This belief stems from the fact that our study area is surrounded by the Great Lakes, and the view that Great Lakes are an infinite source of water. This view is further exacerbated by low cost of lake water, and willingness of governments to build new pipelines to supply water to areas that had previously relied on ground water (as in the case of city of Strathroy, located west of London). The structure of this scenario is identical to the base case scenario, with the only exception that

water availability feedback loops in the socio-economic component are disabled (i.e., they are artificially set to unity for the entire simulation). This means that regional population and economic activity can grow regardless of the condition of its water resources.

The third scenario tested is one where area residents and businesses (in other words all users of ground and surface water) voluntary agree to reduce their water use by 30% when compared to the base case scenario. This scenario is aimed to show the behavioural implication of a strict water conservation policy. Within the socio-economic model component, this scenario is set by initially reducing per capita usage rates of urban and rural population, as well as reducing water usage per business structure and farm unit by 30%.

The fourth scenario is similar to the third scenario, as the same one time water use reduction of 30% is employed, together with land rezoning policies emphasizing conservation of forest and agricultural lands. This scenario aims to reduce rates of urbanization resulting from favourable socio-economic conditions (abundant land, water, jobs, housing, etc); it also aims to reduce total water use in the basin. This scenario achieves this by severely restricting the conversion of agricultural land to business and residential uses, as well as implementing strict water conservation policies. In the socio-economic model component, this scenario is implemented by reducing normal fractional rates of conversion of forest to agcricultural lands, as well as rezoning of agricultural to residential (and business) land categories in addition to lowering per capita (and per business/farm unit) water use.

The fifth and the final scenario considered is identical in structure to the base case scenario, with the exception that no ground water is used in the basin. In other words, this scenario looks to address the implications of performing a switch from ground to surface water supply basin wide. This means that counties that have historically

been using either combination of ground and surface water (like Middlesex), or only ground water (like Perth and Oxford), are assumed to implement policies and initiate construction projects that will provide water from the Great Lakes. This scenario is also unrealistic, but it shows implications of extending the view point that Great Lakes will always be there to provide the much needed water.

6.2.3   Summary of Scenarios

The simulation process undertaken in this study is performed in such a way to consider all possible combinations of climate and socio-economic scenarios. This means that for each climate scenario, five socio-economic scenarios are tested to determine the overall system behaviour. Figure 38 shows the schematic of all scenarios (fifteen in total), while Table 5 provides their brief summaries.

Climate
Scenarios

Socio-economic
Scenarios

Scenario

Hist
- Base
- InfWat
- RedWat
- RedWatLimLand
- SwitchToSW

B11
- Base
- InfWat
- RedWat
- RedWatLimLand
- SwitchToSW

B21
- Base
- InfWat
- RedWat
- RedWatLimLand
- SwitchToSW
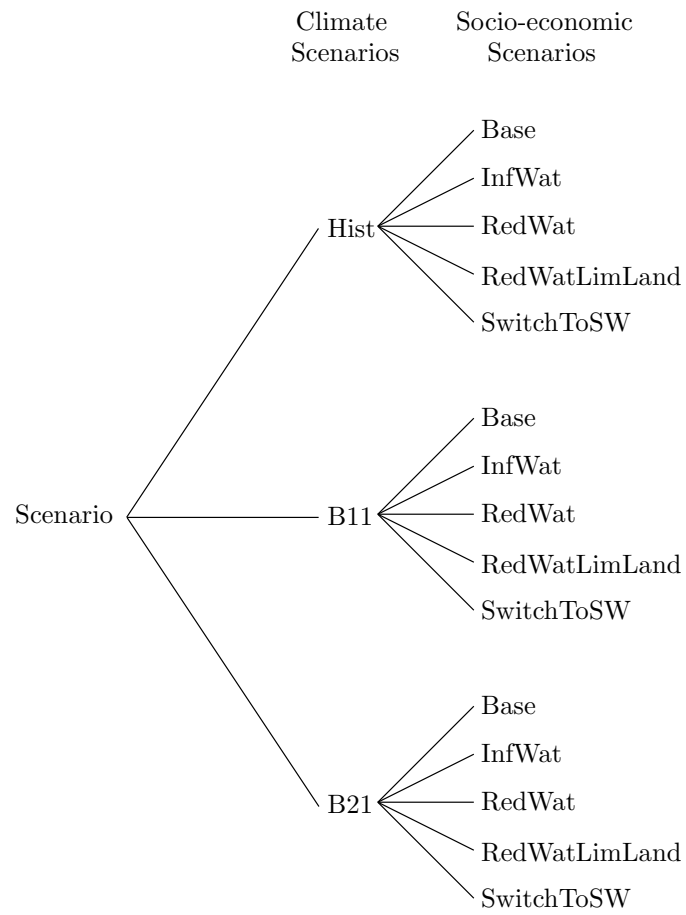
Figure 38: Schematic of scenarios

Table 5: Climatic and socio-economic simulation scenarios

Climate Scenarios		
Historic	CSIROM2kb B11	CCSRNIES B21
Based on regional hydro-climatic data for the period 1964-2001 and represents a historically identical (or base case) climate scenario.	Represents a scenario where higher temperature and dry spells become more frequent in the future.	Represents a scenario where magnitude and intensity of precipitation will increase in the next century.
Socio-economic Scenarios		
Base	InfWat	RedWat
Business as usual (or base) case regarding management strategies and policies; assumes future policies will be the same as those employed today.	A set of policies that assume population, industry and agriculture perceive that there exists an infinite amount of water available for consumption.	Policies that assume people, industry and agriculture reduce their per capita water use by 30%.
RedWatLimitLand	SwitchToSW	
Management strategies that assume a reduction of per capita water use by 30%, together with limiting conversion of agricultural (and forest) land to residential and business uses.	A scenario where a switch from ground water to surface water is made, as governments allow further extraction of Great Lakes water and abandon ground water sources completely.	

7   Results and Discussion

The results presented in this section show the application of the combined model to the Upper Thames River basin. The simulation output is divided into the following categories: First, a number of figures are presented that show the current state of the basin using the historically identical climate combined with the base case socio-economic scenario. This output shows the implications of following current policies and management practices.

Next, simulation output is shown that compares the effect of three different climate signals (combined with the base case socio-economic scenario) for a number of hydrologic variables. This choice of scenarios keeps the management practices and polices constant, while varying the climatic signal. This choice of combination scenarios is able to reveal impacts changed climate can have on regional hydrologic characteristics. After this, a number of socio-economic scenarios are tested, while keeping the climate signal fixed. With this set of scenarios, alternate polices and management strategies are tested in order to show socio-economic impacts of the Upper Thames River basin. Lastly, scenario pairs are tested which couple all combinations of climatic and socio-economic scenarios and its behavioural implications revealed. The climate change scenarios used are described in Section 6.2.1, while the socio-economic scenarios are outlined in Section 6.2.2. Figure 38 and Table 5 show the summaries of all scenarios considered in this study.

The output of the model is shown is this section for the Middlesex County only; outputs of other regions of Oxford and Perth Counties are available on the enclosed CD-ROM.

7.1   State of the Basin

The state of the physical (i.e., hydrologic) conditions are discussed first, and include flood and drought conditions in the basin (shown in Table 6 and Figures 39 and 40). The output is presented in terms of probabilistic frequency curves showing the return period versus a flow of specified duration, together with timing and regularity of hydrologic extremes (after Cunderlik and Burn, 2006). Frequency curves are obtained using procedures of classical statistical analysis where extreme hydrologic output (such as annual maximum/minimum flow) is analyzed. For this study, Gumbel and Log Pearson III statistical distributions are used for assessment of flooding conditions, while the Weibull distribution is employed for the study of droughts. For additional details on statistical analysis methods in hydrology, the reader is referred to the work of Benjamin and Cornell (1970), Linsley and Franzini (1979), Bedient and Huber (1988), and Hoggan (1996).

Indicators of hydrologic extremes in this study synthesize large amounts of hydrologic data and provide a single value as an index upon which change (as one induced by the altered climate) can be measured. The selected indicators measure the changes in magnitude, timing, and regularity of hydrologic extremes. The magnitude indicator selects typical return periods (100 year for high flows, and 20 year for low flows) used in practice with a specified duration (either one, seven or thirty day), and compares its relative change to a number of different scenarios. The timing of extremes is measured with an indicator taking values between 1 and 365, thus pointing out the average day of the year on which the extreme flow occurs. If mean day of flood turns out to be n, this means that, on average, highest annual flow is expected to occurs on the nth day of the year. The regularity indicator, on the other hand is a measure used to indicate the variability of timing on the scale between 0 and 1. Values close

Table 6: Timing and regularity of hydrologic extremes in Middlesex County

	Daily Max			Seven Day Min			Monthly Min		
Scenario	MDF[†] (day)	R[‡] (−)	$1Q_{100}$[§] (m^3/s)	MDD[†] (day)	R[‡] (−)	$7Q_{20}$[§] (m^3/s)	MDD[†] (day)	R[‡] (−)	$30Q_{20}$[§] (m^3/s)
Historic	36.9	0.56	714.01	213.07	0.69	1.61	217.55	0.73	3.37

[†] MDF (MDD) represents a mean day of occurrence of flood (drought) on a Julian calender with values between 1 and 365 (366 for a leap year).

[‡] R depicts regularity of extreme flows, with values between 0 (completely irregular) and 1 (completely regular).

[§] $1Q_{100}$ = maximum annual daily flow with a return period of 100 years fitted with a Log Pearson III distribution; $7Q_{20}$ $(30Q_{20})$ = minimum annual seven day (monthly) flow with a return period of 20 years fitted with a Weibull distribution.
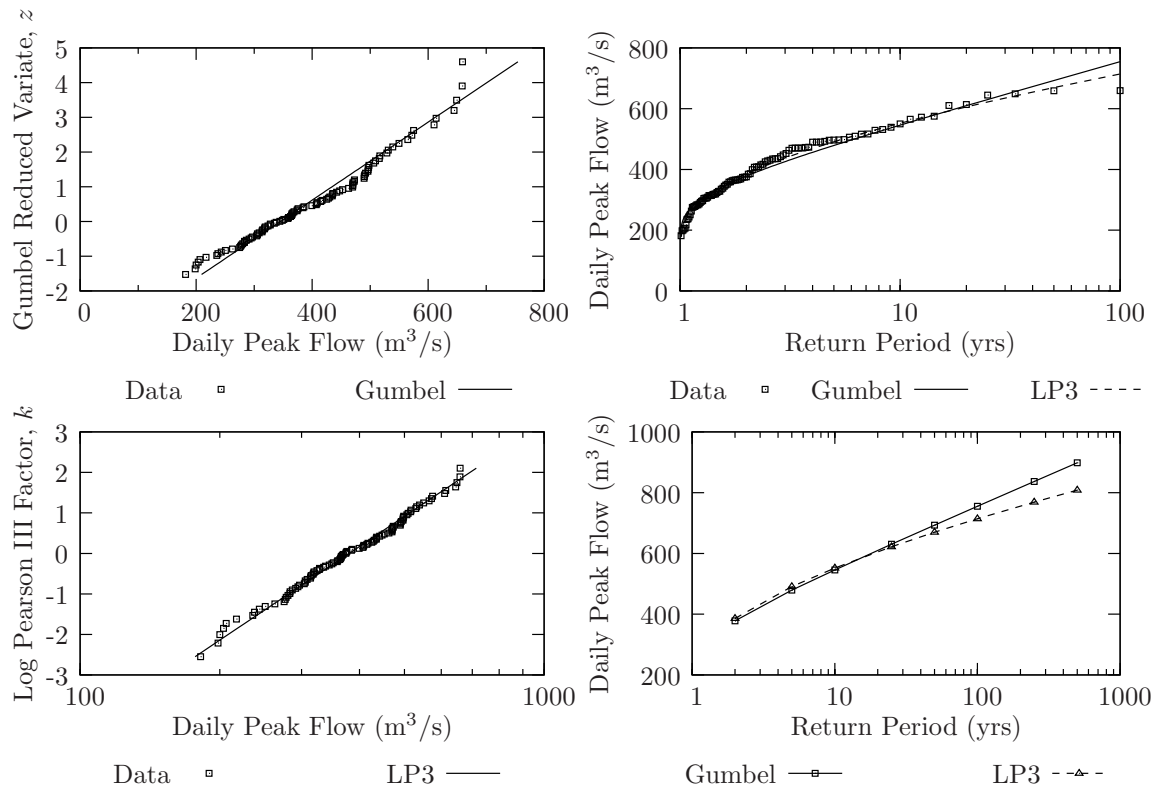
Figure 39: State of flooding conditions of Middlesex County
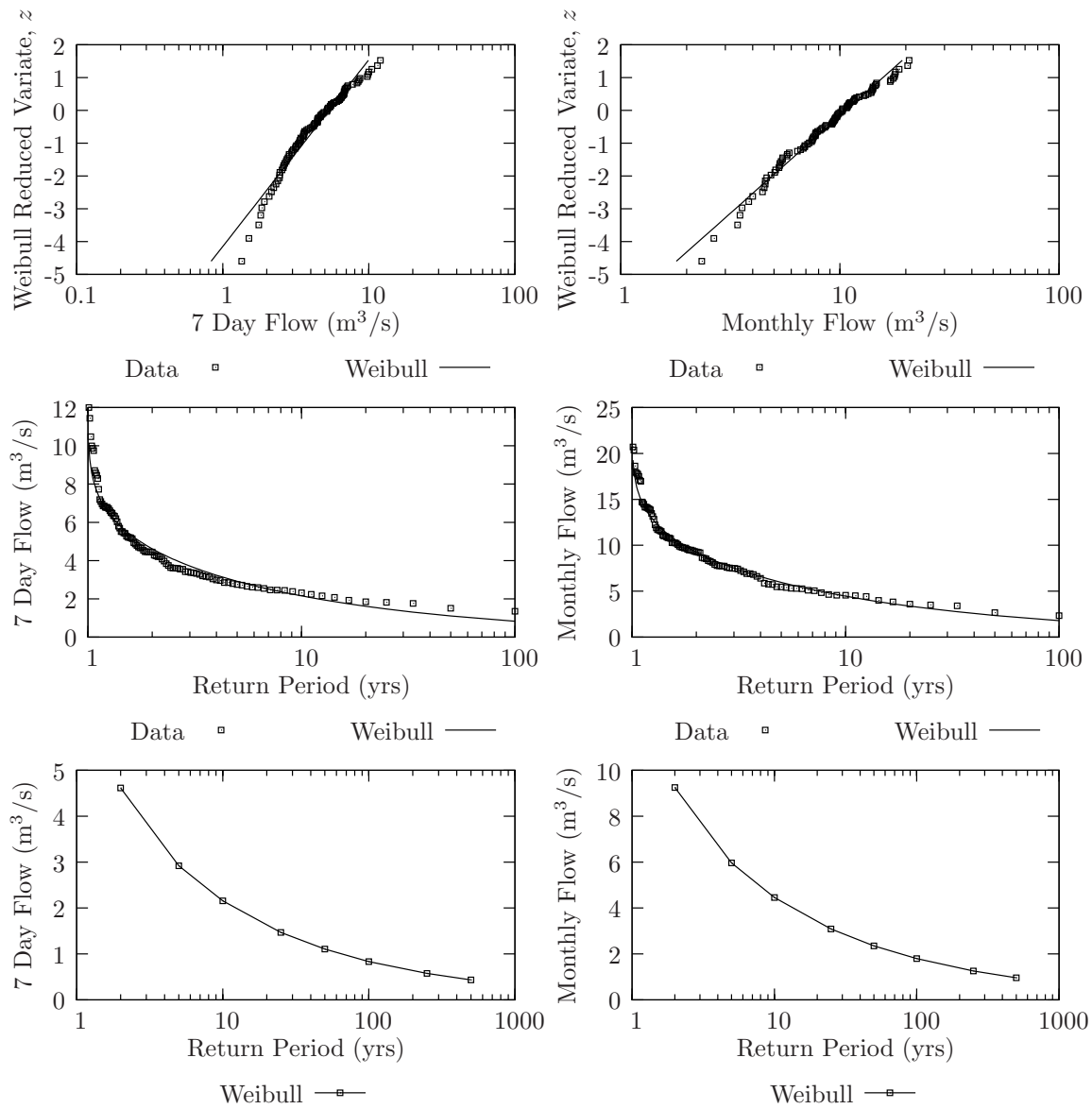
Figure 40: State of drought conditions of Middlesex County

to unity point out that extreme events occur roughly at the same time each year, while values close to zero suggest greater variability in terms of timing. For formal definitions of these indicators the interested reader is referred to a paper by Cunderlik and Burn (2006). It should be noted that magnitude, timing and regularity indicators are of limited value by themselves; their strength lies in comparing different hydro-climatic scenarios, and thus inferring conclusions from their relative changes (i.e., will extreme flows become larger/smaller, occur sooner/later, be more/less frequent, or be more/less regular?).

In analyzing flooding conditions (shown Figure 39), annual maximum daily flows are selected from the model output, and fitted to Gumbel and Log Pearson III statistical distributions. The outputs in Figure 39 are shown as probability paper plots, where model data (annual extremes) is shown for comparison with its fitted distributions. The probability paper plots are then converted to return period plots, to show the return period versus annual maximum daily flow, extended to an arbitrary return period of 500 years (see Figure 39).

The historically identical scenario shows the state of flooding conditions as they are represented by the model in Table 6 and Figure 39. All measures are based on annual maximum flows of daily duration, simulated for an arbitrary period of 100 years. Mean day of flood within this period occurs on 37th day of the year, with the regularity of 0.56. The interpretation of the numbers imply that largest annual daily flows are likely to occur both during snowmelt in early spring, and as a result of intense precipitation during later times of the year (particularly later on in the fall). Daily flood flow with the return period of 100 years, taken at the Byron stream gauge station just outside of London is estimated as 714 m^3/s by using Log Pearson III statistical distribution.

It is recognized that hydrologic data of daily duration is of limited use in design

of flood management infrastructure, as instantaneous peak flows are often larger and thus more critical. Even though daily peak flows are different in magnitude than instantaneous peak flows, studying flood flows of daily duration is still beneficial as they can reveal pattern of change applicable to instantaneous peak flows as well. For example, methods exist that are able to convert daily peak flows to instantaneous flows (Fill and Steiner, 2003). Alternatively, if flooding is the only issue of interest, use of event based rainfall runoff models may be justified (Prodanovic and Simonovic, 2006a).

Drought conditions are similarly analyzed (shown in Table 6 and Figure 40), with the only exception that low flows are considered. As droughts normally represent longer term phenomena, flow is aggregated to 7-day and monthly durations before analysis. Again, same statistical procedures are used, where annual minimum low flows are selected from the model output, and fitted to a Weibull probability distribution. This distribution is typically used in analysis of minimal extremes (i.e., low flows).

Average seven day and monthly annual minimum flows occur at the beginning of August (on 213th and 218th day of year, respectively). Their regularity index have values of 0.69 and 0.73, respectively, thus being more regular than flood flows previously analyzed. In the Upper Thames River basin, July and August have typically been the driest months of the year, sometimes bringing forth intense periods of drought. Water quality during the dry season is of particular concern to residents of the basin, as baseflow in area rivers can consist of a large percentage of treated sewage effluent. Water quality during dry seasons is also of concern, as the basin watercourses have consistently been below provincial standards for minimum phosphorous concentration. The parameter often used in design of municipal infrastructure is the $7Q_{20}$—seven day average flow with the return period of 20 years is

estimated as 1.61 m^3/s using the Weibull statistical distribution. Similarly, monthly averaged flow with a return period of 20 years is approximated as 3.37 m^3/s using the same low flow statistical distribution.

Figures 41 and 42 shows the evolution of the state of urban and rural conditions in the Middlesex County. The output is presented in terms of two columns of plots: the left column shows the major state variables used in the socio-economic model component (business structures, housing and population for urban, and farm units and population for rural sector), while the right column shows the variables influencing the state variables (such as availability of land, labour, water, crops and housing).

In the base case scenario, economic growth and expansion is observed to occur at a staggering pace. For example, the number of business structures in the City of London, and Middlesex County in general, is assumed to increase by some 60% within the next decade. Given the current strength of the Canadian economy, rapid increase in population growth and immigration, together with wide availability of water, labour force, and land, this is not an unreasonable assumption. For the purposes of this study, the chance of occurence of such a state is only important in that it provides a base case (or a reference) mode of behaviour, upon which alternate policies and management strategies can be tested.

As expansion of business structures continues, so must the population needed to sustain it. Rates of in migration (and possibly even birth rates) are expected to increase, mainly due to attractiveness of social and economic well being. During the period of growth, jobs are expected to be plentiful (and sometime even exceed the actual labour force), thus attracting even more people. Due to such high economic well being in both urban and rural sectors of the basin, the stock of housing must inevitably rise to be able to support such a rapid pace of growth.

However, as growth intensifies, so must the rates of urbanization and development
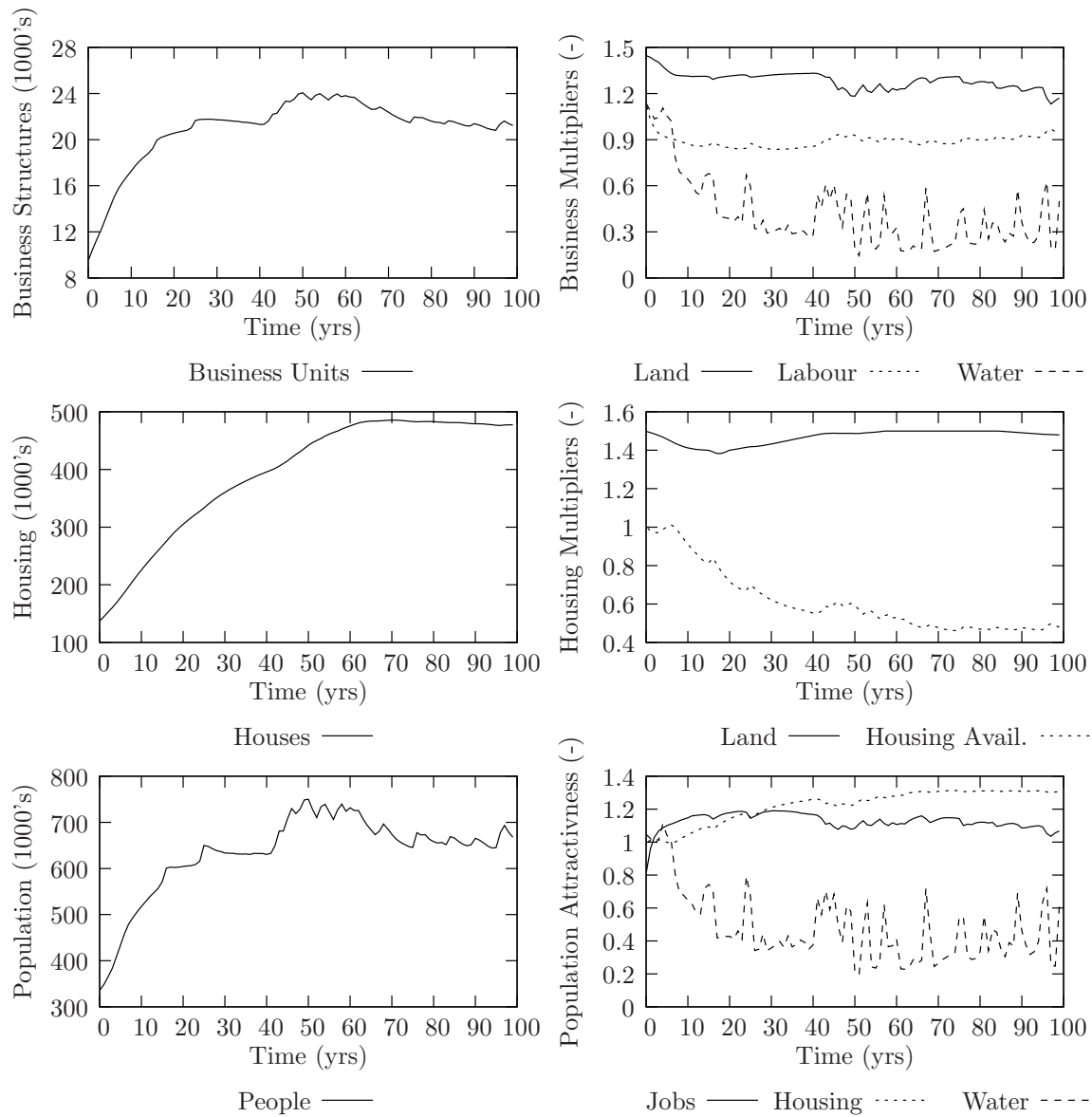
Figure 41: State of urban conditions in Middlesex County
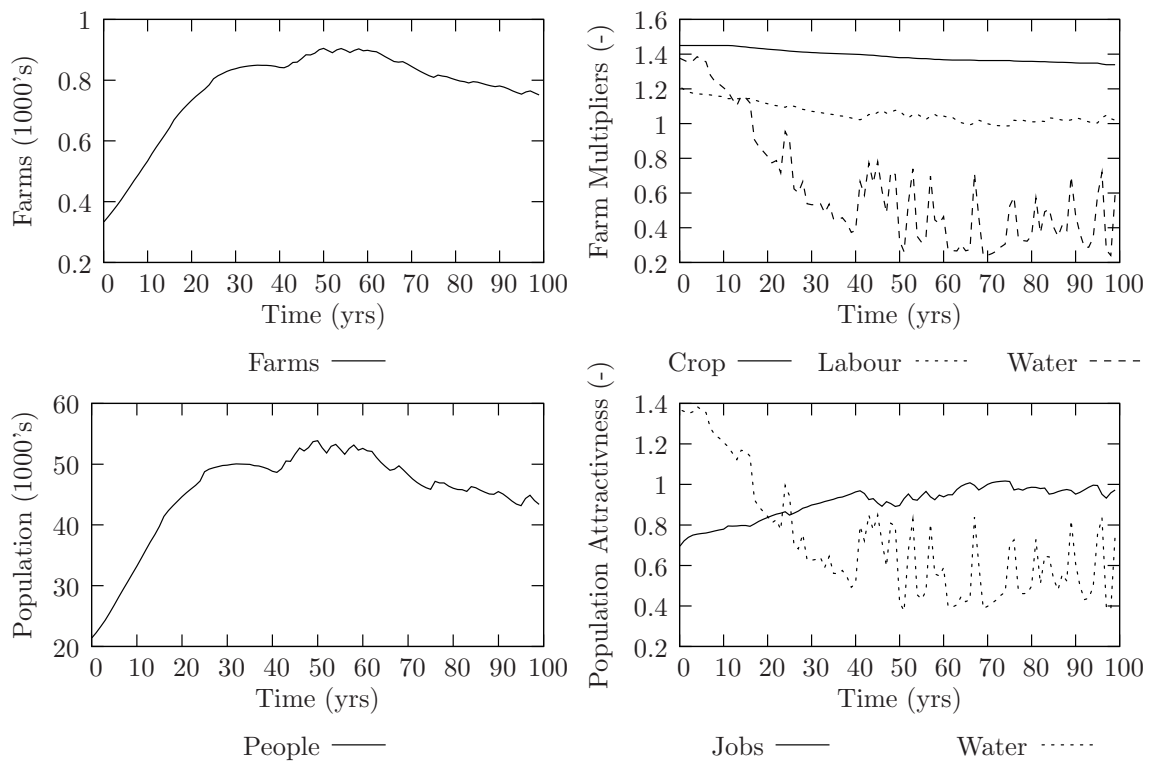
Figure 42: State of rural conditions in Middlesex County

(primarily of agricultural lands surrounding the urban centres). For example, the north part of the City of London is currently seeing such development, as mostly pristine agricultural lands are being converted to residential and business uses. The areas most likely to experience such development are ones closest to existing developed centres, as access (such as roads, highways, rail lines) and services (water, sewer, electricity, telecommunications) are already widely available. However, such a rapid rate of growth places additional stress on the already stressed environment. The stock of forested and agricultural lands starts to decline, as the the urbanized landscape increases (businesses, shops, malls, parking lots, homes).

As more land gets converted to urban uses (and impervious or paved areas increase), ground water recharge is expected to decrease. Ground water recharge is defined here as the fraction of precipitation that enters the ground water aquifers any given year. In this study it is assumed that ground water use can only be as large as the recharge rate, so that mining ground water supplies is avoided (extracting more from the reserves that is naturally replenished). Decreasing ground water recharge rates is a direct consequence of increased urbanization (i.e., paved and partially paved surfaces are unable to absorb precipitation into the soil that eventually recharges the aquifers). As a result of this, the precipitation is forced into channels, storage ponds, reservoirs, rivers and streams almost immediately after falling in the surface, instead of slowly infiltrating into the ground. As more water is diverted from infiltrating into the ground, less water actually recharges into the aquifers that many municipalities use as their water supplies (see Figure 43). Of course, as the area's available water resources decline, future growth and expansion may slow (or indeed stop all together).

Of all factors limiting growth that are considered (land, labour force, jobs, water availability, housing), water turns out to be the most important. In other words, due to high usage rates and declining supplies, water is observed as a limiting factor
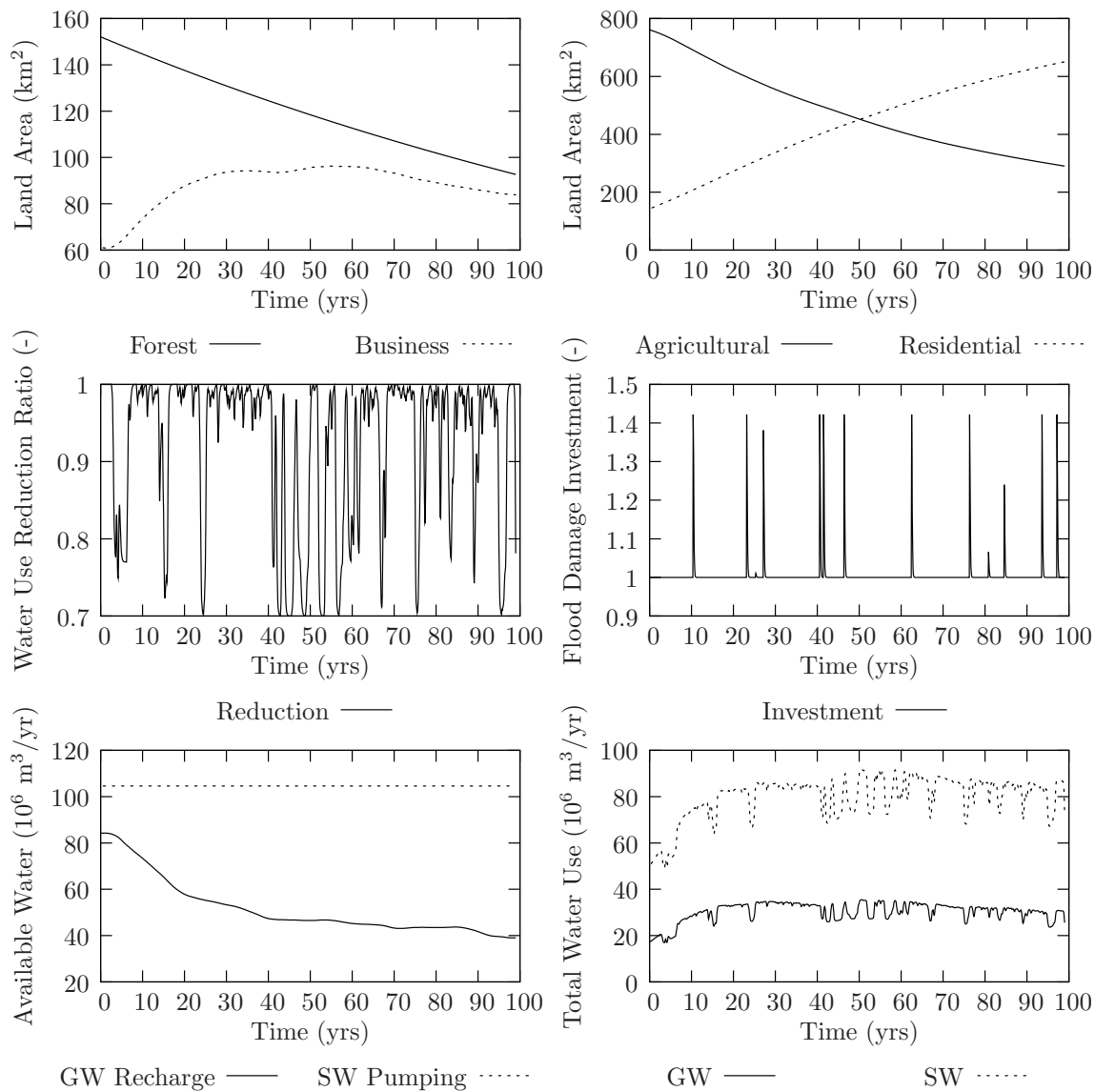
Figure 43: State of land and water use conditions in Middlesex County

to future growth and expansion. At such rapid rates of growth, development and continual expansion (of population, businesses structure, farm units, housing) can not be sustained past two to three decades. After reaching such time, despite plentiful housing conditions, availability of land, labour force and jobs, growth simply can not take place because water is not available (or if it is available, it might be too polluted and costly to treat).

The equilibrium population reaches about 650,000 for London and its surrounding urban areas, while rural population is seen to grow to 50,000 thus bringing the total population to 700,000 (see Figures 41 and 42). This is more than double the 2001 census amounts. The region's housing stock is expected to reach close to 500,000 (a three fold increase since 2001). Much of the housing stock is assumed to have been built during the times of rapid expansion, when families concentrated on accumulation of capital and wealth rather than on raising children.

Dynamics of land cover, flood and drought conditions, as well as availability and use of water are shown in Figure 43. The top left and top right plots show the state variables depicting land use, and thus show decline of agricultural and forest land as a result of increasing residential and business land use. The middle two plots in the same figure present signals received by the socio-economic model component from its hydrologic counterpart, and show the water use reduction ratio (determined from drought conditions) and a flood damage investment multiplier (calculated from perceived state of flooding). The variables showing drought and flood conditions reveal that drought has a potential to reduce water use significantly (that is, if residents actually conserve water during periods of drought), while flooding is not expected to pose a great risk. In other words, flood damage is restricted only to those living in close proximity to low lying areas prone to flooding, and alone is not enough to majorly impact region's overall economic well being.

Total available water, from ground and surface supplies is shown on the bottom left of Figure 43. The plots show the ground water recharge, as well as an maximum amount of water postulated to be allowed to be extracted from the Great Lakes for human consumption. An arbitrary maximum is set as twice the amount pumped in year 2001—implying that at most, people of Middlesex County can have twice what they had used in year 2001. Such an assumption illustrates that future international agreements between Canada and United States will cap the amount allowed to be drawn from the Great Lakes for uses by humans. The bottom right plot of Figure 43 shows total water use from ground and surface sources, and depicts that growth can be sustained only as long as available water exists. Sometimes large fluctuations in total water use are observed, resulting from an optimistic assumption that urban and rural water users actually reduce their consumption in light of drought conditions. The model assumes that intense drought conditions strengthen conservation efforts, thus reducing total water used.

7.2 Impacts of Climatic Scenarios

This section presents potential changes of hydrologic conditions (flood and drought regimes) in Middlesex County (and the Upper Thames River basin in general) as a result of altered climatic signal. All conclusions are derived from Table 7 which shows changes in timing, regularity and magnitude of hydrologic extremes for three different climate change scenarios (historic, B11 and B21), and Figure 44 which shows flow frequency plots for high and low flows for the same scenarios. Comments on potentially altered flooding regime are made by considering the climate change scenario B21—this scenarios represents a plausible future where intensity and magnitude of precipitation intensifies. Remarks regarding drought conditions are based on climate

Table 7: Comparisons of timing and regularity in Middlesex County

Scenario	Daily Max			Seven Day Min			Monthly Min		
	MDF[†] (day)	R[‡] (−)	$1Q_{100}$[§] (m³/s)	MDD[†] (day)	R[‡] (−)	$7Q_{20}$[§] (m³/s)	MDD[†] (day)	R[‡] (−)	$30Q_{20}$[§] (m³/s)
Historic	36.90	0.56	714.01	213.07	0.69	1.61	217.55	0.73	3.37
B11	20.68	0.63	601.71	214.94	0.68	1.56	216.83	0.72	3.31
B21	55.16	0.51	885.21	220.94	0.61	1.85	232.64	0.64	4.10

[†] MDF (MDD) represents a mean day of occurrence of flood (drought) on a Julian calender with values between 1 and 365 (366 for a leap year).

[‡] R depicts regularity of extreme flows, with values between 0 (completely irregular) and 1 (completely regular).

[§] $1Q_{100}$ = maximum annual daily flow with a return period of 100 years fitted with a Log Pearson III distribution; $7Q_{20}$ ($30Q_{20}$) = minimum annual seven day (monthly) flow with a return period of 20 years fitted with a Weibull distribution.
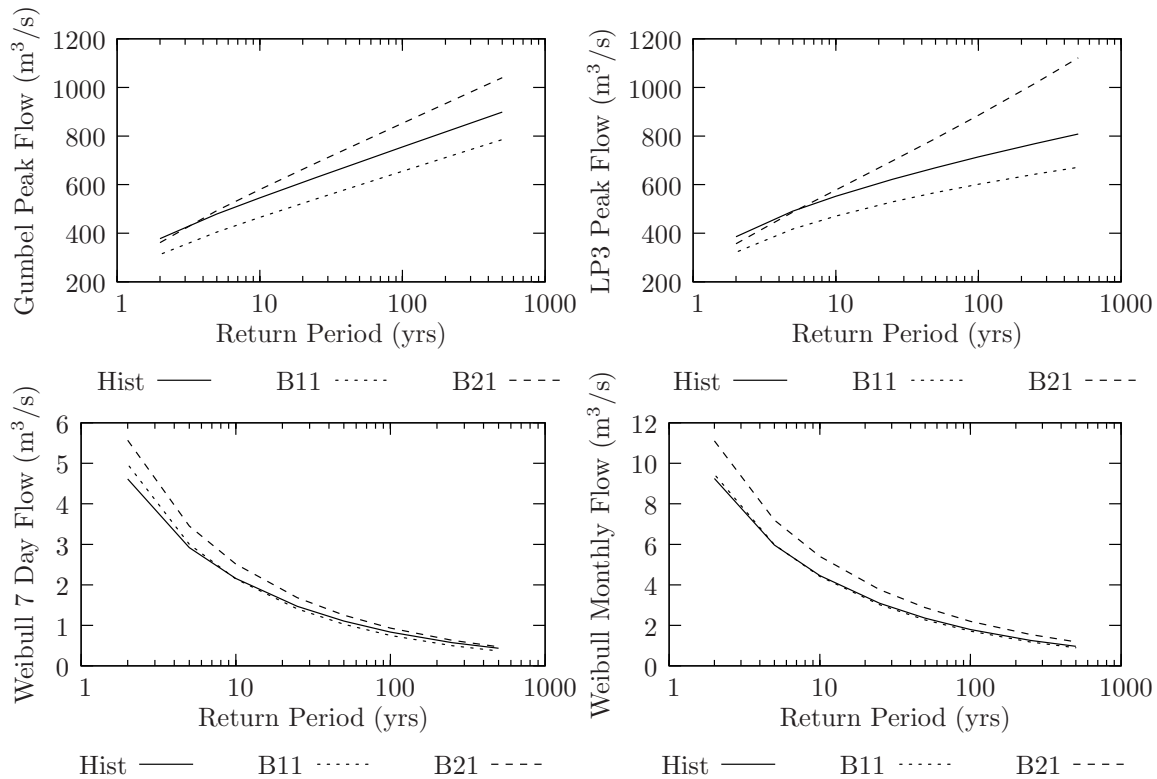
Figure 44: Effect of altered climate on floods and droughts in Middlesex County

scenario B11, formulated with the intent of studying a possible future where higher temperatures and dry spells become more frequent.

7.2.1 Floods

In assessing floods under changed climatic conditions, information from Table 7 and Figure 44 is used to compare relative changes in hydrologic extremes, and therefore infer possible consequences resulting from changed hydro-climatic conditions. In analyzing changes of magnitude of peak flows, the flow with a return period of 100 years is selected as a reference indicator, and is obtained from the Log Pearson III statistical distribution (see top right plot in Figure 44). This indicator is selected as majority of water management structures (such as dams, levees, dykes and diversions) are designed to withstand impacts of 100 year events. The 100 year daily peak flow magnitude for the historic scenario at the Byron stream gauge in Middlesex County is 714 m^3/s, while under scenario B21 (more intense precipitation regime) the flow of the same return period can increase to 885 m^3/s. This is a significant finding, signalling the need for possible revision of existing flood management guidelines and municipal design practices. Scenario B11 on the other hand—representing higher incidents of dry periods—shows an actual reduction of peak flow, with the magnitude of 602 m^3/s. The results of scenario B11 are shown here for comparison purposes only, as this scenario is designed specifically to study impacts of droughts.

Analysis of possible adverse changes in frequency of peak flows can be also performed by using Figure 44. For example, if top right graph is selected (showing the Log Pearson III fit) the peak flow of 714 m^3/s for the historic scenario has a return period of 100 years, while the same flow under scenario B21 may occur with a return period of 30 years. This means that a flow which currently has an annual exceedence probability of 0.01 (1 in 100), may, under future conditions have an annual exceedence

probability of only $0.0\dot{3}$ (1 in 30). Such a finding implies that incidents of flooding in Middlesex County may occur more frequently in the future. This is true not only for large daily peak flows, but lower flows as well.

The timing of annual maximum daily peak flows is also affected by the changing climate. For the historic scenario, the mean day of flood occurs on the 37th day of the year with a regularity index of 0.56 (see Table 7), implying that floods are likely to occur due to snowmelt in the spring, and later in the fall resulting from intense precipitation. Under the changed climate, the mean annual daily peak flow may occur as late as the 55th day of the year (for the B21 scenario), with a comparable regularity to the historic case. This implies that future flooding conditions (under scenario B21) are expected to be as variable as in the past, but may occur on average some 20 days later (i.e., floods from intense precipitation are expected to play more of a dominant role than in the past).

Altered hydro-climatic conditions resulting from climate change imply that flood flow frequency is altered as well, especially for the B21 scenario. This suggests that some of the current guidelines and management practices could be revised accordingly. For example, the Upper Thames River Conservation Authority, the government body responsible for establishing and mapping basin's floodplains, may need to revise and update their floodplain maps in light of the changed climatic signal. This work is the subject of a study by Mortsch (2006).

Further impacts of climatic change in the basin (especially for the B21 scenario) imply that flooding will occur more frequently in the future, regardless of the magnitude of floods. For example, this means that property flooding shall occur more frequently on average than it did in the past. This is a troubling fact, especially since the many towns currently experiences regular flooding resulting in property damage. The story is no different for the largest urban centre of the basin, the City of London,

which if scenario B21 plays out, shall experience flooding more frequently.

The City of London has a number of flood management works, namely the Fanshawe Reservoir as well as an elaborate dyking system for the Thames River. These flood management works have been originally designed to provide a certain level of safety to the city (i.e., able to handle a flows with return period of 100 years) against damaging floods. As a result of climatic change, this infrastructure may in the future provide a lower level of safety than it has been designed for. Therefore, appropriate government officials may need to consider retrofitting or upgrading the existing flood management works in the basin, which will undoubtedly result in increased budgets for flood management operations.

Climatic change in Middlesex County (and in the Upper Thames River basin in general), manifesting itself through changes in flood flow frequency, has the potential to alter capacities of storm water facilities and/or sewer treatment plants (through larger and more frequent flood volumes). In the City of London for example, older parts of the city (including the downtown core) have combined sanitary and storm sewers. As a result, in the event of an abnormally high precipitation causing large flood volumes, not all sewage is able to be transported to the area's treatment plants. The consequence of this is that raw sewage (combined with excess rainwater) is discharged directly into the Thames River without any treatment. Again, this is expected to occur more often (if the city sewer systems are not upgraded). A further consequence of potentially changed climate is that current design standards of bridges, roads, sewers, drains, storm water facilities and other municipal infrastructure should be reviewed, in order to assess their ability to cope with the potentially changed hydro-climatic conditions.

The following presents a summary of issues regarding possible changes of flood management guidelines as a result of climatic change in the Upper Thames River

basin. Some of the issues included in these recommendations are taken from comments and suggestions provided by basin's stakeholders during a stakeholders meeting (held on November 22, 2005) organized by team members of this project. The changes are categorized in the following three categories:

1. *Regulatory.* Changed hydro-climatic characteristics imply that rules and regulations currently employed by the Upper Thames River Conservation Authority may need revision. This recommendation is supported by the analysis showing that floods of all magnitudes will increase in frequency of occurrence (i.e., they will occur more often). Possible revisions of existing regulation include those pertaining to: (i) use of river and its adjacent park land for recreation (including boating, camping, swimming, fishing, jogging, etc.); (ii) removal of pumping stations near area rivers during the course of a flood; (iii) patrolling of river banks during periods of high water levels and monitoring performance of critical infrastructure (such as roads, bridges, culverts, drains, sewer systems); (iv) issuing of permits for floodplain development (including construction of roads, buildings, bridges, culverts, drains, sewer systems, etc); (v) revision of reservoir operating procedures (including smaller dams, weirs, outflows, etc.).

2. *Budgetary.* This set of guidelines look at possible changes in allocation of budgets for safe operation of existing flood management infrastructure, as well as budgetary constraints to be imposed by investment needed for future infrastructure still in the planning stages. If floods occur more frequently (as scenario B21 suggests), so should maintenance, retrofitting and upgrading of existing flood management infrastructure. For example, the over topping of dykes in London is expected to occur more often than in the past, implying that dykes will hold higher water levels more often. As this happens, development of

cracks, crevices and other imperfections will intensify, thus requiring increasing maintenance levels and leading to higher costs. An evaluation of current structural and non-structural measures used to reduce flood damage (such as infrastructure like reservoirs, dykes, floodwalls or implementation of land use zoning practices, flood warning systems, waterproofing, etc.) will need to take place. As flood frequency increases basin wide, so do costs allocated with maintenance of flood protection measures. Next to changed maintenance budgets of existing infrastructure, a need may arise for investment of additional infrastructure that may need to come on line in the future to address changed climatic conditions. As this new infrastructure comes on line, existing rules that determine budgets typically allocated to maintenance may not be appropriate and may need further consideration.

3. *Engineering.* Changes in this set of regulations aim to look at possible changes in design standards of municipal infrastructure (such as roads, buildings, bridges, culverts, drains, sewer systems, treatment plants, etc.). Changes are going to be necessary as climate change is expected to bring about changes in hydro-climatic characteristics. Since the current hydro-climatic characteristics have been used to formulate today's design standards, it only follows that under changed conditions different standards should be set, or at least old ones comprehensively reviewed.

In addition to required action by government officials and experts, information of the potential consequences of climatic change should also be provided to special interest groups, non-governmental organizations, interested individuals as well as the general public. The changed climate has the potential to effect everyone living in the basin, from possibly higher taxes to restricted floodplain development possibilities.

In addition, the interested groups and individuals should be allowed to participate in the planning process, and be included in the decision-making.

7.2.2 Droughts

Drought conditions are similarly analyzed under the altered climate. Hydrologic indicators of change are presented in Table 7 and Figure 44. As before, the results presented here are for Middlesex County only; similar results for Oxford and Perth Counties is presented in the enclosed CD ROM.

The magnitude indicators selected for the study of hydrologic drought conditions is the annual minimum seven day flows with the return period of 20 years (i.e., $7Q_{20}$). These are flows defined by Ontario's Ministry of Environment as ones providing a limiting condition for sewage treatment and wastewater disposal for a receiving water body (Pryce, 2004). The $7Q_{20}$ flows are therefore used to measure the stream's ability to accept (and dilute) point source discharge like treated sewage effluent, and consequently evaluate stream's water quality. Monthly minimum flows of the same return period are shown here for completeness, as they might be useful in studying longer term trends.

The $7Q_{20}$ parameter for the Middlesex County (taken at the Byron stream gauge) is computed as 1.61 m^3/s for the historic climate scenario. Little variation of this parameter are observed under alternate climatic signal (see Table 7 and Figure 44) implying that variation in magnitude of such flows are not expected to change significantly. Timing and regularity of low flows are also not significantly impacted by the changes in climate, as annual minimum lows flows are seen to occur in 213th day of year (end of July and/or beginning of August) for the historic scenario, while the B11 scenario (designed specifically to study impacts of drought) shows minimum annual low flows occurring on 215th day. Confidence in above conclusions is strengthened

by seeing that even scenario B21 (showing higher precipitation regimes) depicts the mean day of drought occurring on 220th day of year, only a week later from the historic case. Regularity indicators also show a consistent trend in supporting the conclusion that large changes are not expected in variation of timing of low flows.

Drought conditions are observed to be less frequent than those experienced in the past. For example, if we take a flow of 1.61 m^3/s on the bottom left plot of Figure 44 showing the annual minimum seven day flow versus the Weibull probability fit, we observe that under historically identical conditions the return period for such flow is about 20 yrs. Under the changed climate scenario B21, the same flow has a return period of 30 yrs, implying that such low flow is expected to occur (on average) less often. This means that the probability of observing an annual minimum seven day flow of 1.61 m^3/s or less any given year is currently 0.05 (1 in 20), but may be as little as $0.0\dot{3}$ (1 in 30) as a result of changed climate.

Low flows may not be the only indicators of regional drought conditions. Ways regulatory authorities manage dry periods is as important as their actual occurrence. Although the province of Ontario has a drought management framework that includes officials across all sectors of the government and incorporates various long and short term measures, local and/or regional drought management plans are still missing. This is further reinforced by the fact that negative drought related impacts in Middlesex County (and the Upper Thames River basin as a whole) are currently quite minor. Drought damage related studies in the basin are altogether lacking, despite the fact that potentially severe droughts are possible.

As Loucks and van Beek (2005) put it, drought management is tricky. This is because it is difficult to know what actions to take in light of potential drought conditions, as they usually occur over a long period of time and are not easily identified. It is difficult to obtain funding (or even the attention of the politicians or the general

public) for drought management when there always seems to be a more pressing immediate problem that demands human and financial resources. Loucks and van Beek (2005) claim that it is not easy to get a multi-agency support in drought response until a severe drought is imminent or occurring (p. 598). The authors also warn that getting various government agencies to work together during a crisis is not efficient, and that crisis oriented drought response is usually ineffective, poorly coordinated and untimely.

What is needed are local drought management guidelines that address differences in regional river basin characteristics such as predominant land use patters (urban, rural, agricultural, forest), water use (surface or ground water), population and industrial growth, as well as others deemed important. For example, different regions within the basin have different thresholds for drought triggers, which need to be carefully assessed and documented. Currently, the thresholds used are those suggested by the OLWR (2003) document, and are the same for all regions of the province of Ontario. Guidelines of this kind can potentially mask local (or smaller scale) drought conditions. Further exacerbating the problem is the fact that drought triggers can change with time, as in cases when infrastructure (such as reservoirs, canals, and agricultural tile drainage) is upgraded or if water demands change. Urban and rural drought triggers may also need to be set at different levels, as urban infrastructure can usually absorb the impact of shorter droughts (Loucks and van Beek, 2005, p. 596).

During meetings with various stakeholders in the Upper Thames River basin it was acknowledged that during the summer months a large percentage of the baseflow of the area rivers can consist of treated effluent. (The basin has a number of state of the art water treatment facilities and pollution control centres that discharge treated waste into the rivers.) This has the potential of deteriorating water quality downstream of the treatment plants.

It should be noted that water quality in the Thames River (both upstream and downstream of London) is below the provincial water quality standard that Ministry of Environment sets for Ontario's watercourses. Like many other watercourses in Ontario, Thames River is classified as a "Policy 2" watercourse, indicating that its total phosphorous concentration is below the targets set by the provincial government. Phosphorous is a nutrient necessary for plant and animal life, but elevated levels can cause excessive plant growth and algal concentrations, as well as eutrophication of surface water bodies. Furthermore, SNC-Lavalin and RV-Anderson (2005) state that basin's elevated phosphorous concentration results from:

- agricultural runoff and ground water flow from feedlots, pastures, fertilized fields, septic systems, etc.
- urban stormwater drainage including combined sewer outflows, and runoff containing street drainage, waste from resident and bird populations, residential fertilizer, etc.
- Pollution Control Plant effluent discharges (p. ES-1).

A report by SNC-Lavalin and RV-Anderson (2005) shows that mass balance of total phosphorous sources for London indicates that pollution control plants contribute about 17% of total annual phosphorous loading; tributary loads and stormwater runoff represent 14%; 30% and 38% of total annual phosphorous loading originate from North and South Branches of the Thames upstream from the city, respectively. It is also noted that total phosphorous loading in the Thames River downstream of London has either remained the same, or decreased since the start of record keeping in 1978, despite an increase in population of about 100,000 persons between 1978 and 2005. This is attributed to successful removal of phosphorous and other pollutants by area's pollution control plants, as well as other water quality improvements initiatives

(sewer separation, stormwater management, sewer use bylaws, etc.) by the City of London.

Despite an excellent record of London's pollution control plants, the Thames river consistently experiences water quality problems, particularly during dry seasons. Therefore, it is our recommendation that increased emphasis be placed on management practices (particularly in rural areas that drain most of the basin) and limit total phosphorous loading. This is because drought like conditions can exacerbate already poor water quality in area rivers.

Other issues of concern when droughts are considered are those of water use. Those living in smaller communities that are users of ground water may experience drying of wells, thus requiring drilling of costly new and deeper wells. This is further exacerbated by the fact that water use during times of drought increases (due to warmer temperatures), while ground water recharge (due to decreases in precipitation) usually decreases. This means that conditions are possible where water demand is higher, but the supply is lower. Careful management guidelines requiring strict conservation and possible restriction (of water uses in commercial, industrial, residential and agricultural sectors) are required to lower the impact of this potentially troubling situation. Furthermore, modification of reservoir operations and enforcing compliance of existing water taking permits may also need to take place.

A large amount of surface water is extracted from Lakes Erie and Huron and is currently used primarily by residents of City of London and its surrounding areas. City and other public officials often acknowledge a widespread belief of those living in London that there is virtually an unlimited supply of water for the city. This is troubling, since water levels in Lake Huron have recently been much lower than in the past. Furthermore, users may not fully appreciate the impact of increasing water withdrawals from the Great Lakes, especially in light of the knowledge that only about

of 2% of its water is renewable. Stricter drought education initiatives are needed (in addition to current information bulletins provided by staff of the Upper Thames River Conservation Authority) in order to make the residents more aware about drought, its potential impacts, and possible long term damage reduction mechanisms.

In light of the previous discussion regarding drought conditions in the Upper Thames River basin, four categories of recommendations are made, which include:

1. *Drought impact assessment.* Local and/or regional drought impact assessment are non-existent in the Upper Thames River basin, despite the fact that severe droughts can occur with a probability of 1 in 10 (sometimes less) any given year. A recommendation is made that a drought impact assessment study be commissioned, and look into drought impacts on: (i) agriculture (including lower crop yields, tilling practices, erosion control methods, selection of seed types, etc.); (ii) tourism and recreation (impact of stagnant, foul smelling water on trail and park visits for jogging, camping, fishing, swimming, etc.); (iii) wetlands (loss of habitat of fish, reptiles, birds and mammals; changes in vegetation cover); (iv) reservoir operations (including conflicting use of water for low flow augmentation and flood control); (v) ground water withdrawals (including increased costs during peak drought times, drilling of deeper wells); (vi) streamflow water quality (especially when flow consists of treated sewage).

2. *Local drought triggers.* Drought is a spacially variable phenomena, and therefore drought thresholds and/or triggers need to be locally defined and set—preferably on a sub watershed scale. Emphasis should be placed on drought triggers of urban and rural areas, as urban residents (especially those receiving water from the Great Lakes) are often unaware of drought conditions. Furthermore, as physical and socio-economic conditions change in the basin (such

as land development, population growth, water use patters, etc) so could the drought triggers. Careful long-term monitoring of such conditions is necessary as changes in drought triggers over time are possible. A flexible regulatory framework should be set to allow changes in definitions, should they be deemed necessary.

3. *Water quality management.* Drought and drought like conditions contribute to lower water quality. As the Upper Thames River basin watercourses systematically experience water quality problems (particularly with total phosphorous loading) new regulations should be placed to monitor water quality in both short and long term. A review of all existing water quality guidelines should also take place. Some of these should include a review of: (i) agricultural fertilizer use; (ii) storm water management techniques for current and future land development; (iii) combined sewer discharges; (iv) pollution control plant effluent strategies; (v) landfill use practices, including solid waste management.

4. *Education programs.* Increasing awareness of drought and its potentially devastating impact on people, plant and animal life is seen as paramount to the sustainability of the natural environment. Novel educational programs (targeting water resources managers and practitioners, municipal and other government officials, farmers, as well as the general public) are urgently needed. The following set of initiatives are suggested to achieve greater awareness of droughts and its adverse impacts: (i) enhance communication of current drought conditions (including specific goals and targets, and practical means to achieve them); (ii) introduce smart water use techniques (both via technology and modifying long held beliefs that water is a limitless resource); (iii) enhance interactions between socio-economic and physical domains (emphasizing a systemic approach where

a realization that structure of socio-economic systems has just as much to do with drought as current climatic and/or physical conditions).

Securing funds for such studies is recognized to be a challenge, especially since the ageing water management infrastructure (reservoirs, dykes, sewer systems, etc.) in the basin demands immediate attention. A fine balance between short and long term goals need to be achieved; after all, the sustainability of water resources is at stake. What is urgently needed is a systemic approach to water management.

7.3   Impacts of Socio-Economic Scenarios

This section presents the results of a number of socio-economic simulation scenarios, coupled with different climate scenarios. All possible combinations of climate and socio-economic scenarios are tested, and comments regarding their outcomes are provided.

By performing extensive model simulation and analysis, one conclusion is reached that warrants further discussion prior to addressing scenario specific results of the model. The conclusion in question states that socio-economic policies and management strategies are observed to have a greater impact on the long term socio-economic character of the region than impacts of climate change alone. Of course this does not mean that climate change will have no impact on socio-economic behaviour in the region; it will, but its impacts may not be as severe as those resulting from adoption of certain policies which have the potential to significantly alter the socio-economic landscape in the region.

This conclusion was reached by performing the following simulations and analyzing its outputs: (i) keeping the climate signal fixed while varying socio-economic policies, and (ii) keeping the socio-economic scenarios fixed while changing the climatic signal.

Simulation results in (i) showed a larger variation in socio-economic behaviour when compared to those performed in (ii).

The rest of this section therefore focuses on scenarios that test different socio-economic scenarios while keeping the climatic signal fixed. The reader is referred to Table 5 for a quick summary of the scenarios considered in this work. All scenarios studied in this report start with the same assumptions as those in the base case scenario (described in Section 7.1), and then make appropriate adjustments. Therefore all simulation results are compared to the base case scenario.

7.3.1 Infinite Water Availability

Even though highly unrealistic, this scenario shows behavioural implications of the belief that water resources are limitless in the basin. It shows what could be expected to happen if people believe water will always be available and never in short supply. This assumption reflects the present belief of many in the basin who think that region's close proximity to the Great Lakes make it immune to any possible future water shortages. The output of a number of selected variables for this scenario is shown in Figures 45, 46 and 47.

Figure 45 shows the evolution of the urban and rural sectors resulting from the comparison of infinite water availability scenario with the base case. In this scenario tremendous growth is observed (even more than in the base case), as water is no longer believed as the limiting factor. Equilibrium values are sometimes four times higher than observed in the base case scenario. Urban population in Middlesex County (London and its surroundings) is therefore seen to grow to about 2 million, while rural sector of the population to more than a 100,000. This implies that the City of London could become a large metropolitan centre within the next two to three decades. Tremendous development would be seen in the area, particularly in
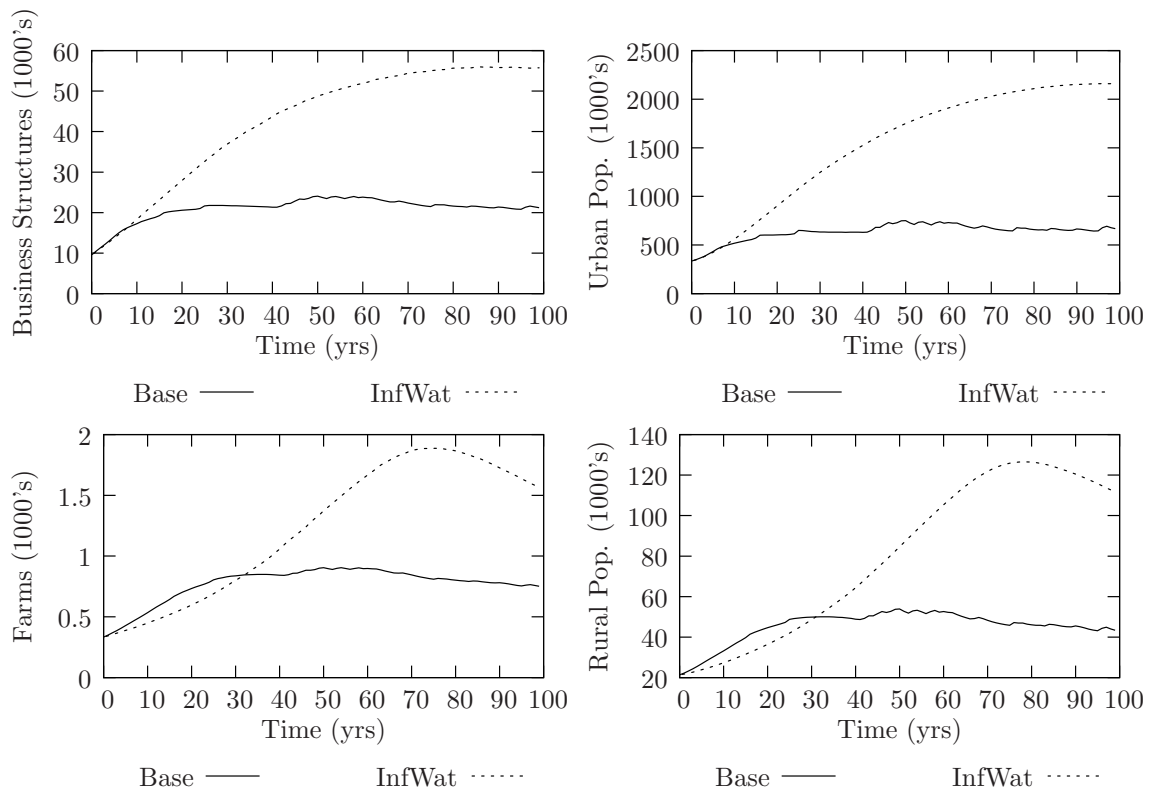
Figure 45: Middlesex urban and rural conditions under InfWat
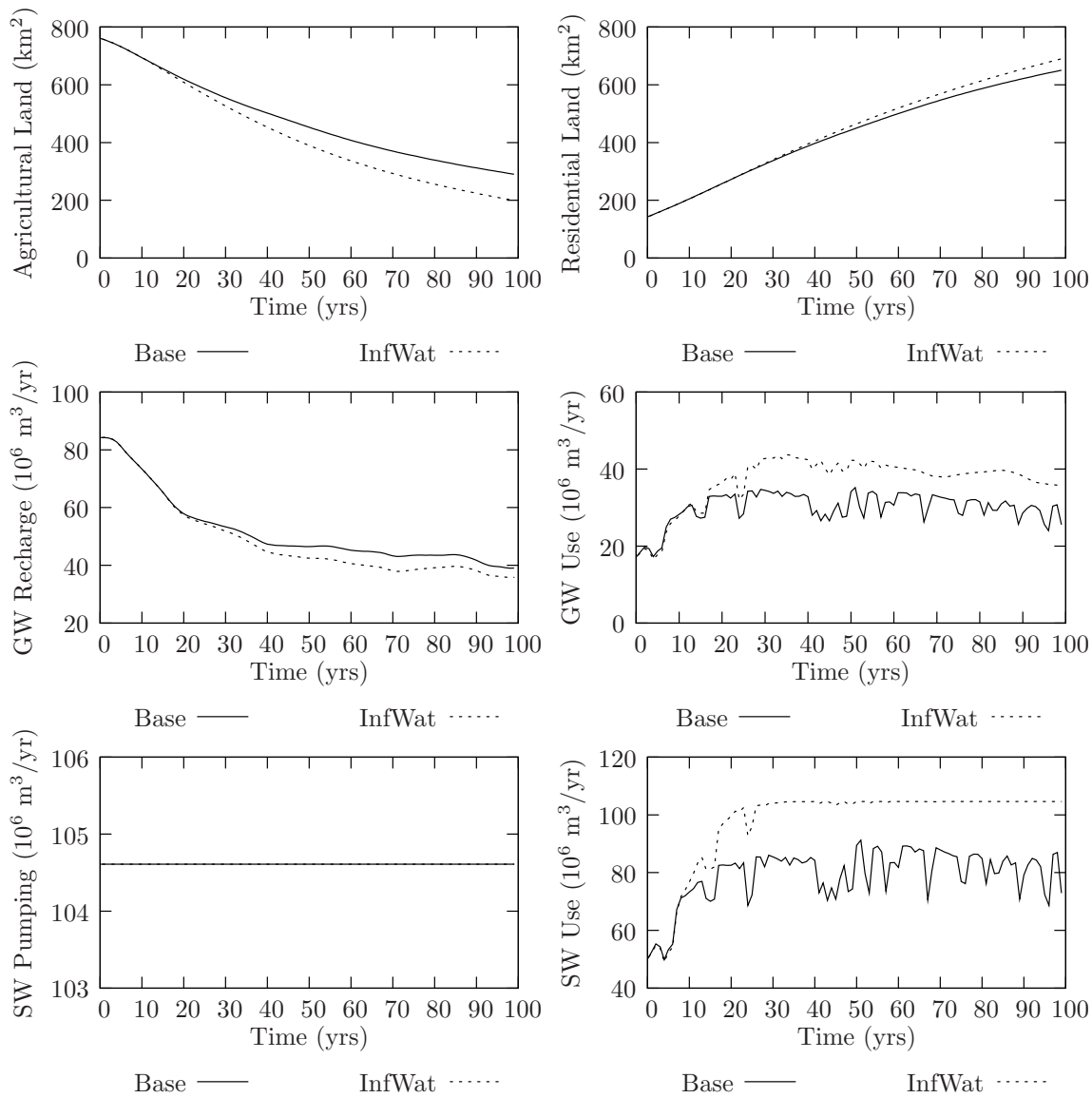
Figure 46: Middlesex land and water use conditions under InfWat
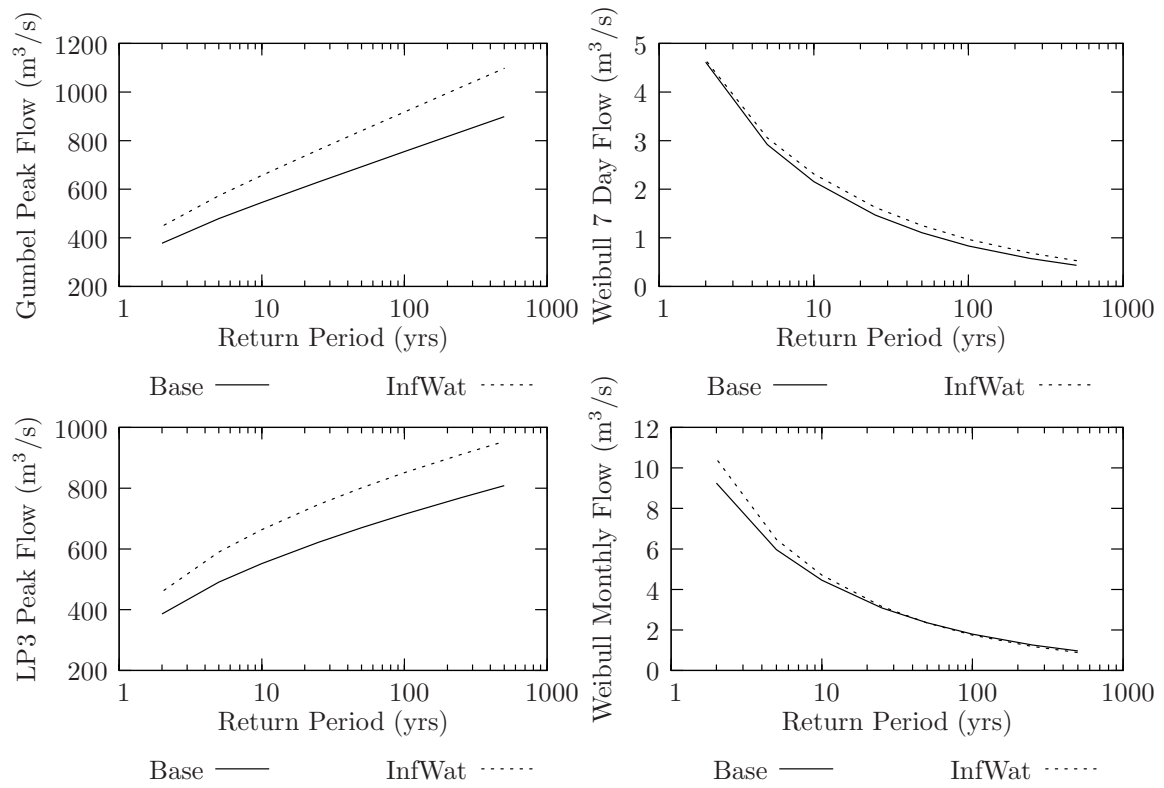
Figure 47: Middlesex flood and drought conditions under InfWat

expansion of existing industrial, commercial and institutional economic sectors. The agricultural sector is also expected to see a boom, as increasing population depends on it for the supply of food.

However, land and water use conditions shown in Figure 46 indicate a different living condition than originally observed in the base case scenarios. A large number of businesses (and people) must be sustained on smaller land areas, consistent with metropolitan like settings. Furthermore, per capita water use in all sectors is significantly reduced, as area businesses and residents can not use more than is currently available (dictated by yearly ground water recharge rates and amount pumped from the Great Lakes). Ironically in this scenario, growth reduces the amount of ground water recharge from the base case, thus forcing the area residents to increasingly rely on more surface water supplies. However, the maximum surface water drawn from the Great Lakes (still assumed to be capped at about twice the total amount pumped in 2001) is reached in about two decades, thus making further growth absolutely senseless.

It is interesting to point changes in regional hydrology that could be expected from such high rates of growth. As an increased area of residential and business land becomes occupied (and subsequently paved over to make room for roads, houses, parking lots), a larger portion of rainfall is anticipated to immediately end up in area rivers and streams, thus increasing the amount surface runoff. At times when area rivers are flowing at or near capacity (such as during snowmelt for example), this additional runoff (that could have infiltrated into the ground) is now added to the flood water, thus further exacerbating flooding conditions. The result of this are increased frequencies of flood flows, for all magnitudes of flow (see Figure 47). Drought conditions under this scenario are unchanged compared to those of the base case.

7.3.2 Reduced Water Use

This scenario shows an implementation of an intensive water use conservation programs for all water users in the area. In testing this scenario, all urban and rural residents are assumed to reduce their per capita water use by 30%; business and farms are also expected to reduce water use per business structure and farm unit by the same amount.

Simulation output revealed by Figures 48 and 49 shows the impacts on the overall system. The behaviour is observed to be of counter-intuitive nature. This is because it challenges the widely accepted view that strict conservation measures benefit the area by reducing total water use. Immediately after the implementation of the conservation policy, total water use is indeed reduced. However, since the area now has an greater initial water availability (people and businesses are using less), the restriction on future growth from water availability is not as severe (as more water is now available). This means that higher growth rates in the regional economy, and in the population can be sustained, thus amounting to more growth than in the base case scenario. Eventually, the state variables equilibriate with higher levels of population, businesses, farm units and houses, and therefore increase the area's overall water use in comparison to the base case. What was thought to be a policy whose aim was to reduce total water use, in the end accomplished the exact opposite.

A further comment is offered at this point to illustrate that complex nonlinear feedback systems (the model contained in this work is of that type) have a tremendous insensitivity to parameter and policy changes. In such systems, a change in one variable (such as per capita and per business water use) is often offset by changes in other places (support for additional growth and development through increased water availability) that in the end make the performance of the initial policy totally
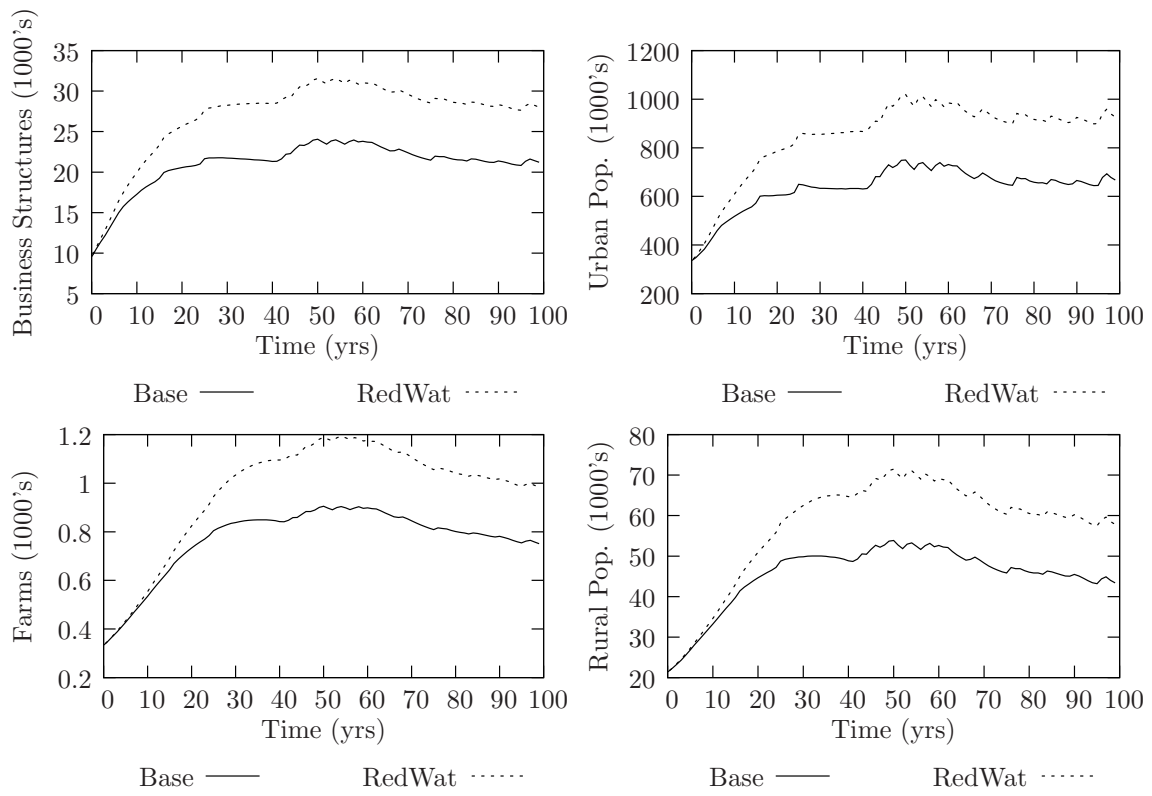
Figure 48: Middlesex urban and rural conditions under RedWat
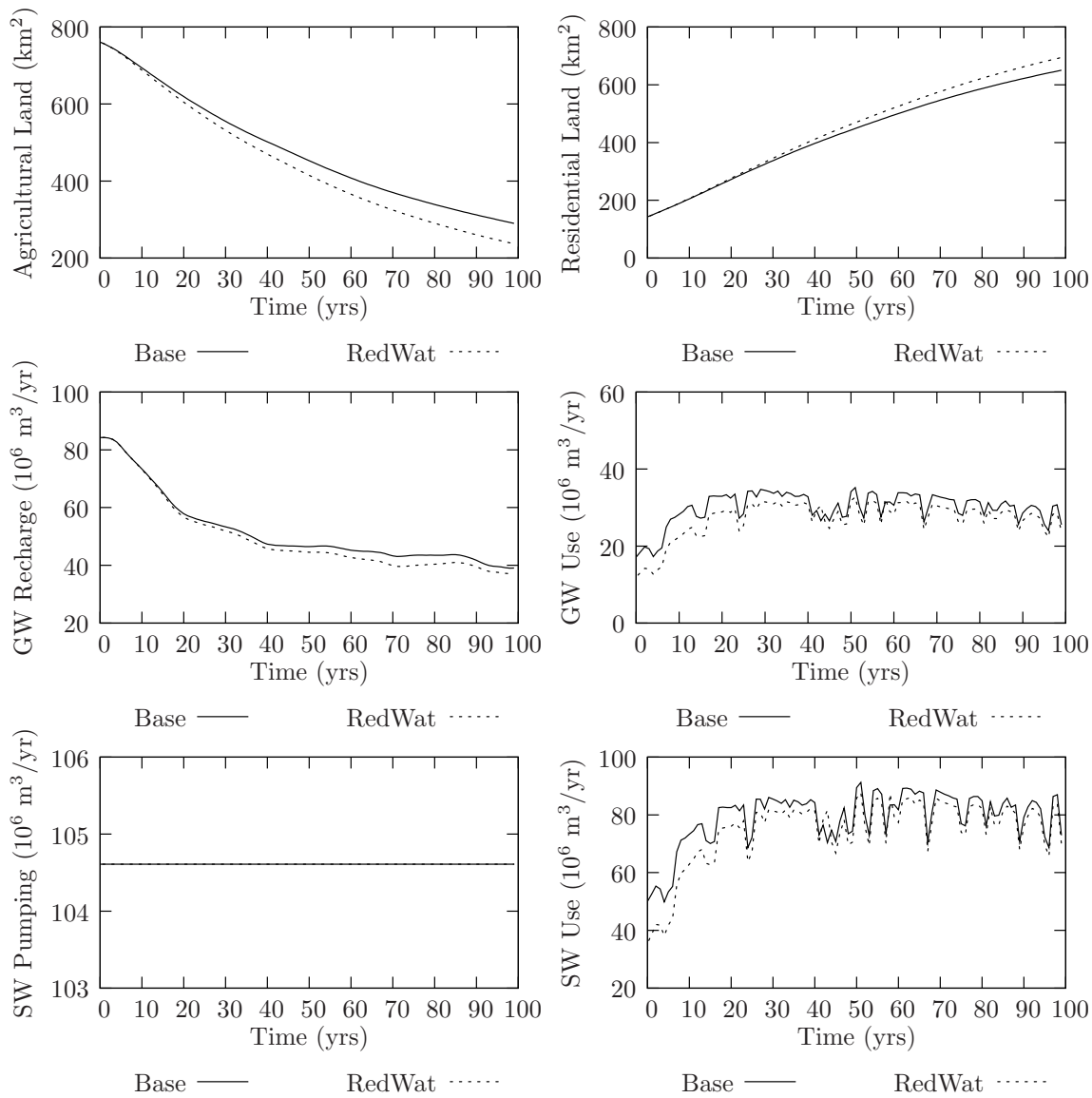
Figure 49: Middlesex land and water conditions under RedWat

counteractive.

7.3.3 Reduced Water Use and Limited Land Rezoning

If reducing water use alone can not accomplish the intended consequence of reducing the limiting impact of water availability, perhaps a policy that restricts water use in combination with placing limits on land rezoning can perform better. That is exactly what this current scenario investigates. This scenario is identical to the one previously discussed, with added policies that limit conversion of agricultural land to residential (and business) uses.

Figures 50 and 51 show the simulation output for this scenario. A notable reduction is observed in the number of business structures over time, although the level of urban population remains nearly as high as in the base case. The state variables in the rural sector increase, as there is more freedom for those involved in agriculture to expand without pressures from the urban community. The urban community is now restricted in development opportunities, as emphasis is placed on environmental preservation. The urban population grows to nearly twice as it was in 2001, while expansion of residential land uses occurs at a much smaller rate. As water and land conservation programs dominate the management practice, the region is expected to adopt an increasing sustainable outlook for its future. Smarter homes are expected to be designed and built that require use of much less resources (such as water, land and energy) for sustenance. Careful land use planning leading to smart development and expansion can occur, but only with an aim of utmost environmental sustainability and protection of ecosystems, people and businesses.

Figure 51 shows an effect of preserving much of the original agricultural land while increasing residential only by a modest amount. The implication of having more agricultural land available over time increases ground water recharge, meaning that
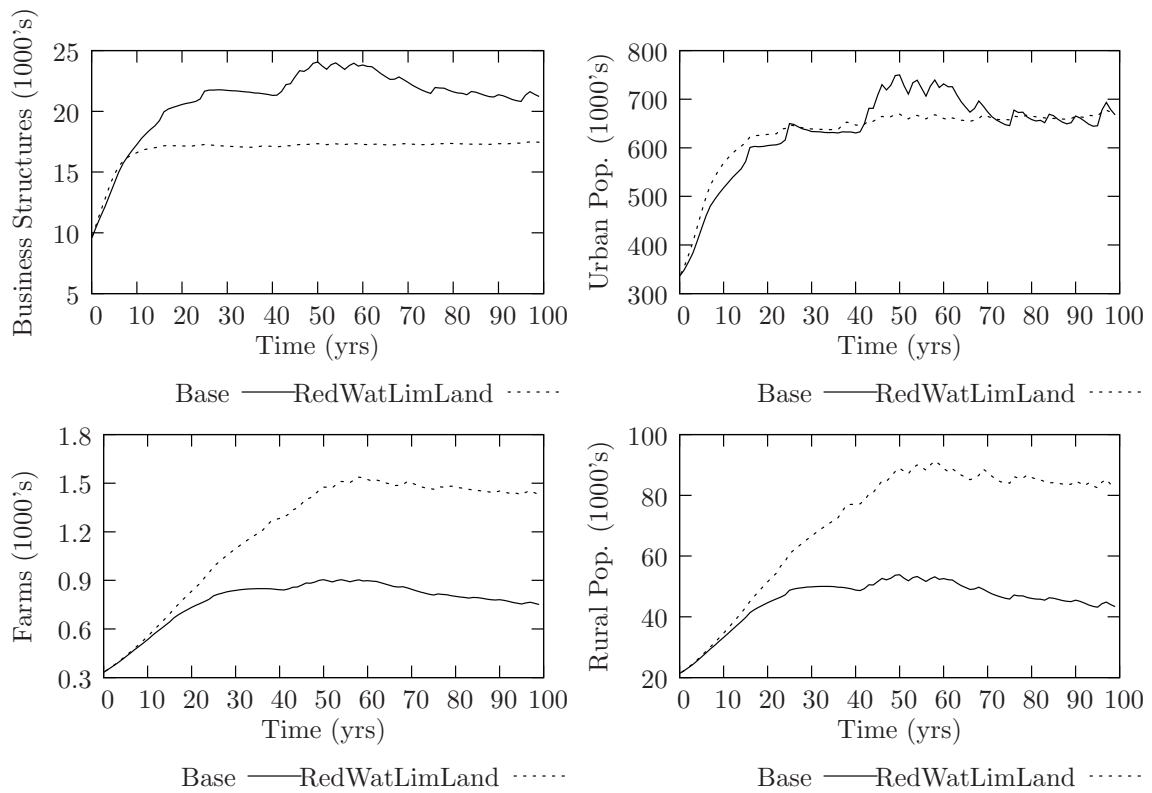
Figure 50: Middlesex urban and rural conditions under RedWatLimLand
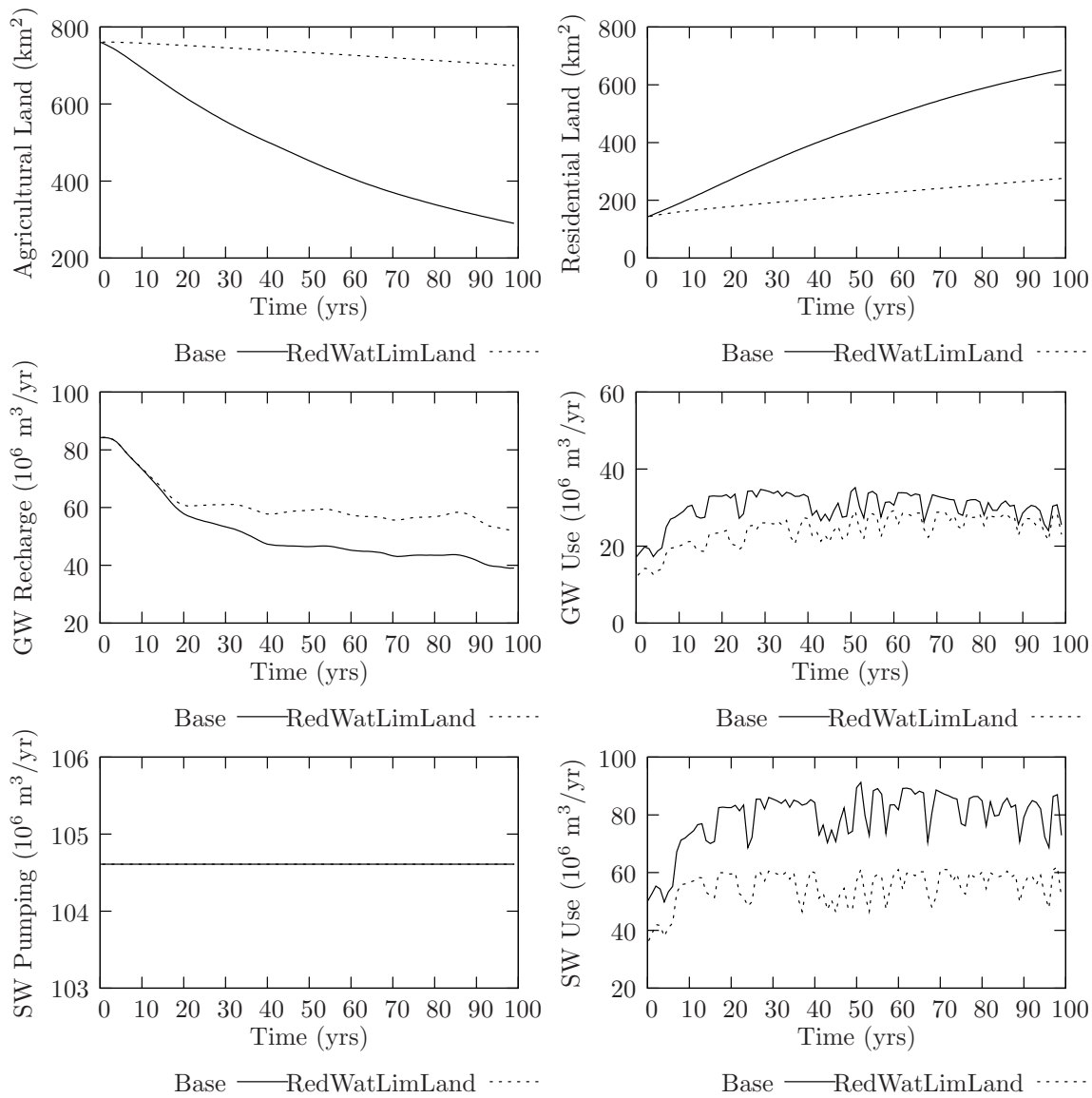
Figure 51: Middlesex land and water conditions under RedWatLimLand

more water becomes available for use than compared to the base case scenario. Even though more ground water is available, the restriction policy successfully achieves a reduction of total water use (both for ground and surface water).

Of course, such a future can not be expected to occur unless major shifts in attitudes occurs from economic and social sectors within the region. Major efforts will be needed to achieve public support for such policies, while educational efforts towards environmental sustainability and protection shall need to strengthen significantly. Economic developments at the expense of the environment (with increased degradation of land, reduced water availability, increased water quality problems) should be limited.

7.3.4 Switch to Ground Water Use

The last of the socio-economic scenarios involves the testing of a policy where a shift from ground to surface water use is made basin wide. The simulation output for this scenario is shown in Figures 52 and 53, and simply represents a policy change that eliminates use of ground water entirely and replaces it with water pumped from the Great Lakes. It assumes that massive investment efforts are initiated by all levels of government for the construction of infrastructure (water treatment plants, pumps, pipelines, etc) that would eventually supply the entire Upper Thames River basin with lake water.

This policy change results in the reduction of urban population and the number of business structures when compared to the base case scenario. In the base case scenario, the combination of surface and ground water provided more total water than the increased supply of surface water alone available for this scenario, thus lowering population and the number business units. The rural population and the number of farm units grows modestly, mainly because of the wider availability of land and water.
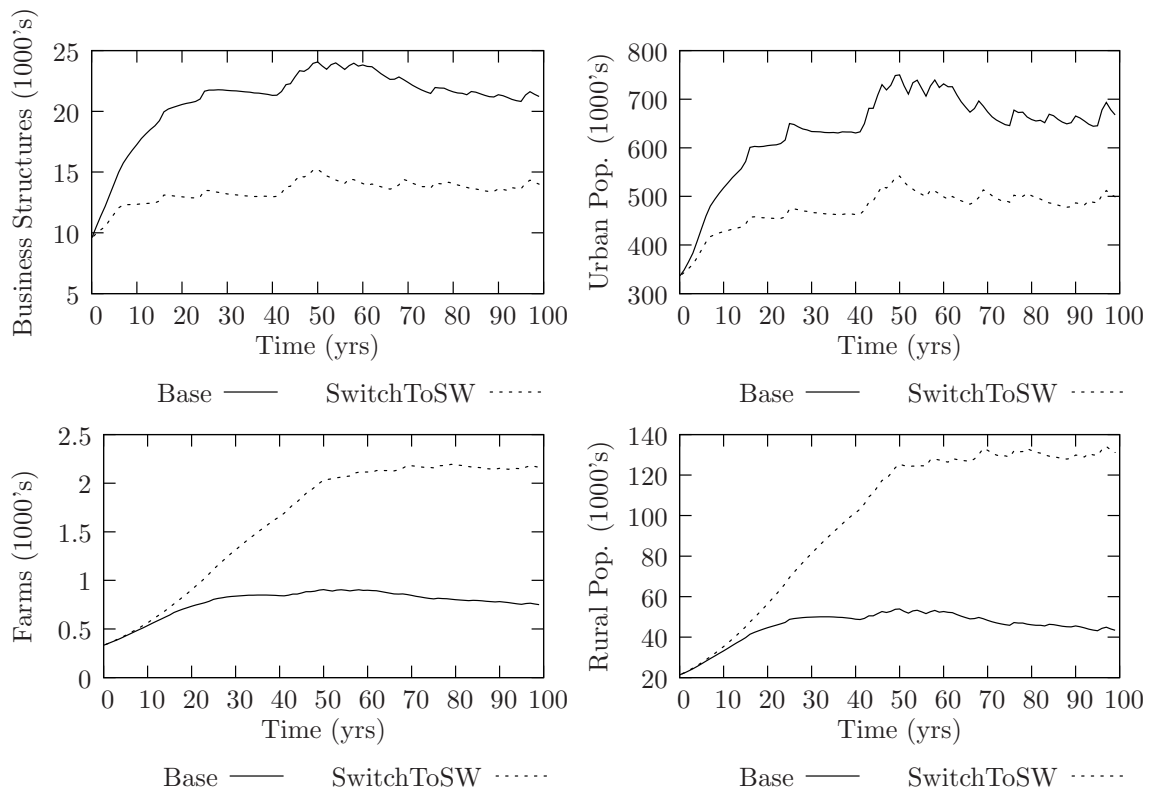
Figure 52: Middlesex urban and rural conditions under SwitchToSW
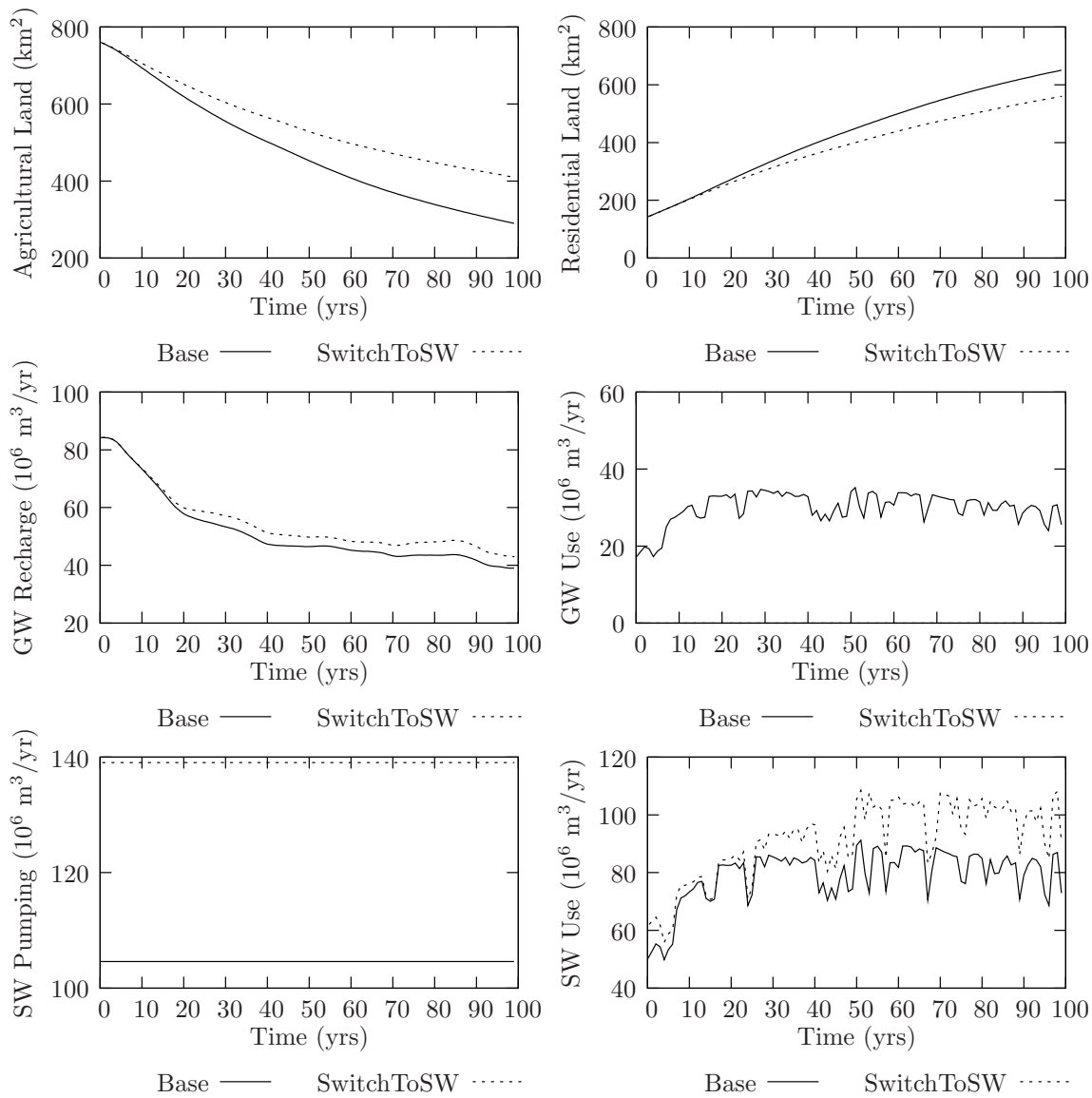
Figure 53: Middlesex land and water use conditions under SwitchToSW

Even though ground water use in this scenario is reduced to zero, surface water use is correspondingly higher. When considering the cost associated with converting an entire basin to surface water supplies, especially when simulation shows that better socio-economic conditions do not emerge, makes this scenario rather unattractive for future consideration.

8   Conclusions

The main objective of this project is to study risk and vulnerability of water resources systems to changing climatic conditions. Since water resources problems can not be studied without the socio-economic context they are embedded in, this report presents a framework able to integrate physical aspects of water resources management with its socio-economic counterparts. New modelling framework is developed using the system dynamics approach where a continuous hydrologic model component describing physical processes is coupled via feedback to a socio-economic model component representing regional socio-economic characteristics. Two model components are linked in such a way that each influences, and is influenced by, the other, thereby capturing the same processes operating on larger scales in the real world.

The model used in the study represents a tool used to practically implement the idea of integrated water resources management. The model is general enough to be able to test a wide range of policies and management strategies (like changes in demographics, housing, jobs and labour force, land use practices, etc), as well as produce detailed hydrologic output on a daily time scale that can be used for impact analysis and/or a variety of engineering design purposes.

Hydrologic extremes (such as floods and droughts) are assessed using different climate signals with a continuous hydrologic model component, while socio-economic impacts are evaluated using a model component of the same name. Simulation scenarios are used to investigate effects of changed climatic and socio-economic conditions on the overall basin behaviour.

After simulating a number of different climate change and socio-economic scenario, together with evaluating their impacts, a number of concluding remarks are made regarding the future of regional physical and socio-economic well being of the

Upper Thames River basin. These conclusions relate to possible changes in hydrologic extremes (floods and droughts), together with changes in regional economic activity (urban and rural population, business and farm units, land use development, water availability and use, etc). The conclusions outline ways in which climatic change, coupled with policies and management strategies has the potential to significantly alter the physical and socio-economic landscape of the basin.

Climate change is expected to intensify flooding in the basin, thus bringing flows of higher magnitudes with more frequent occurrence. Such conditions may demand additional investment in flood management infrastructure and may require complete revision of budgets for flood management and yearly maintenance. It may even require retrofitting and replacing current (or even building additional) flood management infrastructure. Engineering design standards in light of changed climatic conditions may also need revision.

Drought conditions are expected to remain at their current levels, with no appreciable shifts in magnitude, frequency, timing, or regularity. Despite this, severe droughts in the region are common, therefore requiring serious attention from water resources managers, stakeholders and users. This is of particular importance, especially since drought impact assessments in the basin are altogether lacking. Commissioning of such studies forms one of the major recommendations regarding future drought initiatives in the Upper Thames River basin. Such initiatives are needed because deteriorating water quality during summer periods may become a serious concern, to the health of both humans and ecosystems.

In addition to droughts, socio-economic characteristics of the Upper Thames River basin have the potential to become significantly altered as a result of both climatic change and management policies. In testing various combinations of climatic and socio-economic scenarios using the simulation model revealed that availability of water

is the most important factor for future regional economic development. In other words, of all factors postulated to place limits on economic development (availability of water, land, labour force, jobs and housing) availability of water turned out to be the most important. This is because residents and businesses in the basin use such staggering amount of water, that currently known sources are expected to become either echausted or become unavailable within the next two to three decades. If such time is reached, economic growth may slow, or altogether stop.

In order to cope with limited water availability, a socio-economic scenario is considered which implements a strict water conservation policy with the aim of reducing total water use and elevating regional socio-economic well being in the long term. Simulation reveals that implementation of such a policy actually has *negative* long term consequences on regional socio-economic behaviour. This is because reducing per capita (and per business/farm) water use lowers the total amount used in the short term, while increasing overall water availability (i.e., if less gets used, more becomes available). However, higher water availability implies that the region can actually increase its economic base, therefore making additional business investments more lucrative. Over time, the increased economic activity eventually ends up using more water, eventually reaching water use patterns comparable to the base case scenario.

A possible way to cope with keeping total water use at acceptable levels may be to implement a combination policy where a reduction in per capita water use is required, together with a policy that limits further expansion and development (or allow only smart development). Such a case is tested in the simulation model, and encouraging results are obtained. Total water use can indeed be lowered, while keeping most forest and agricultural land intact. Ground water recharge rates are increased when compared to the base case scenario, while expansion of existing residential and business

lands occurs at modest amount without negatively impacting the population.

Such a conclusion points out that the future of the Upper Thames River basin lies in the hands of those currently residing within its borders.

References

Ahmad, S. and Simonovic, S. P. (2000). "System dynamics modeling of reservoir operations for flood management." *ASCE Journal of Computing in Civil Engineering*, 14(3), 190–198.

Alfeld, L. E. and Graham, A. K. (1976). *Introduction to Urban Dynamics.* Wright–Allen Press, Inc., Cambridge, Massachusetts.

Argyris, C., Putnam, R., and Smith, D. M. (1985). *Action Science: Concepts, Methods and Skills for Research and Intervention.* Jossey-Bass Publishers, San Francisco, California.

Bedient, P. B. and Huber, W. C. (1988). *Hydrology and floodplain analysis.* Addison-Wesley Publishing, Upper Saddle River, New Jersay.

Beer, S. (1975). *Platform for change: a message from Stafford Beer.* John Wiley & Sons Ltd., London, United Kingdom.

Benjamin, J. and Cornell, C. A. (1970). *Probability, Statistics and Decision for Civil Engineers.* McGraw-Hill, New York.

Bennett, T. (1998). *Development and application of a continuous soil moisture accounting algorithm for the Hydrologic Engineering Center Hydrologic Modeling System (HEC-HMS).* Masters Thesis, Department of Civil and Environmental Engineering, University of California, Davis, California.

Bronstert, A. (2004). "Rainfall-runoff modelling for assessing impacts of climate and land-use change." *Hydrological Processes*, 18, 567–570.

Brown, D. (2001). *Groundwater Protection Study: Phase II, County of Oxford, Ontario.* Golder Associates Ltd., London, Ontario, Canada.

Bruce, J., Burton, I., Martin, H., Mills, B., and Mortsch, L. (2000). *Water Sector: Vulnerability and Adaptation to Climate Change.* Global Strategies International Inc., and Meteorological Service of Canada, Ottawa, Canada.

Brutsaert, W. (1982). *Evaporation into the Atmosphere: theory, history and applications.* Kluwer Academic Publishers, Dordrecht, Netherlands.

Chow, V. T. (1956). *Open-Channel Hydraulics.* McGraw-Hill Book Company, New York.

Clark, I. D. (2006). "An open letter to Prime Minister Steven Harper." Originally published in the National Post on April 6, 2006.

Coulibaly, P. and Dibike, Y. B. (2004). *Downscaling of Global Climate Model Outputs for Flood Frequency Analysis in the Saguenay River System.* Final Project Report prepared for the Canadian Climate Change Action Fund, Environment Canada, Hamilton, Ontario, Canada.

Cunderlik, J. M. and Burn, D. H. (2006). "Switching the pooling similarity distances: Mahalanobis for euclidean." *Water Resources Research*, 42, W03409.

Cunderlik, J. M. and Simonovic, S. P. (2004a). "Assessment of water resources risk and vulnerability to changing climatic conditions: Calibration and verification data for the HEC-HMS hydrologic model." *Report No. II*, Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

Cunderlik, J. M. and Simonovic, S. P. (2004b). "Assessment of water resources risk and vulnerability to changing climatic conditions: Calibration, verification and sensitivity analysis of the HEC-HMS hydrologic model." *Report No. IV*, Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

Cunderlik, J. M. and Simonovic, S. P. (2004c). "Inverse modeling of water resources risk and vulnerability to changing climatic conditions." *57th Canadian Water Resources Association Annual Congress*, Montreal, Quebec, Canada.

Cunderlik, J. M. and Simonovic, S. P. (2005). "Hydrological extremes in a southwestern Ontario river basin under future climate conditions." *Hydrological Sciences*, 50(4), 631–654.

Cunderlik, J. M. and Simonovic, S. P. (2006). "Inverse flood risk modeling under changing climatic conditions." *Hydrological Processes*, (accepted for publication).

Fernández, J. M. and Selma, M. A. E. (2004). "The dynamics of water scarcity on irrigated landscapes: Mazarrón and Aguilas in south-eastern Spain." *System Dynamics Review*, 20(2), 117–137.

Fill, H. D. and Steiner, A. A. (2003). "Estimating instantaneous peak flow from mean daily flow data." *ASCE Journal of Hydrologic Engineering*, 8(6), 365–369.

Flannery, T. (2006). *The Weather Makers: How Man is Changing the Climate and What it Means for Life on Earth.* Atlantic Monthly Press, New York.

Forrester, J. W. (1961). *Industrial Dynamics.* The MIT Press, Cambridge, Massachusetts.

Forrester, J. W. (1969). *Urban Dynamics.* The MIT Press, Cambridge, Massachusetts.

GWP (2000). *Integrated Water Resources Management.* Global Water Partnership TAC Background Paper 4, Stockholm, Sweeden.

Hoggan, D. (1996). *Computer-assisted floodplain hydrology and hydraulics.* McGraw Hill, New York.

IPCC (2001). *Climate Change 2001: Scientific Basis. Contribution of the Working Group I to the Third Assessment Report of the Intergovernmental Panel on Climate Change.* Cambridge University Press, Cambridge, United Kingdom.

Jackson, M. C. (2000). *Systems Approaches To Management.* Kluwer Academic/Plenum Publishers, New York.

Kell, R. (2004). *Middlesex Elgin Groundwater Study Final Report.* Dillon Consulting, London, Ontario, Canada.

Kolbert, E. (2006). *Field notes from a catastrophe: man, nature, and climate change.* Bloomsbury Publishing, New York.

Leavesley, G., Lichty, R., Troutman, B., and Saindon, L. (1983). "Precipitation-Runoff Modeling System (PRMS), User's manual." *Report No. 83-4238*, U.S. Geological Survey Water-Resources Investigations, Denver, Colorado.

Li, L. and Simonovic, S. (2002). "System dynamics model for predicting floods from snowmelt in North American prairie watersheds." *Hydrological Processes*, 16, 2645–2666.

Linsley, R., Kohler, M., and Paulhus, J. (1958). *Hydrology for Engineers.* McGraw-Hill Book Company, Inc., New York.

Linsley, R. K. and Franzini, J. B. (1979). *Water Resources Engineering.* McGraw-Hill Book Company, New York, third edition.

Loucks, D. P. and van Beek, E. (2005). *Water Resources Systems Planning and Management: An Introduction to Methods, Models and Applications.* United Nations Educational, Scientific and Cultural Organization, Paris, France and Delft, Netherlands.

Mass, N. J. (1974). *Readings in Urban Dynamics: Volume 1.* Wright-Allen Press, Inc., Cambridge, Massachusetts.

Matthias, R. and Frederick, P. (1994). "Modeling spatial dynamics of sea-level rise in coastal areas." *System Dynamics Review*, 10(4), 375–389.

McBean, G. (2006). "An open letter to the Prime Minister of Canada on climate change science." Originally published on the Canadian Foundation for Climate and Atmospheric Sciences website (http://www.cfcas.org) on April 18, 2006.

Meadows, D. H., Randers, J., and Meadows, D. L. (1972). *The Limits to Growth: A Report for the Club of Rome's project on the predicament of Mankind.* Universe Books Publishers, New York.

Meadows, D. H., Randers, J., and Meadows, D. L. (1992). *Beyond the Limits: Confronting Global Collapse and Envisioning a Sustainable Future.* Chelsea Green Publishing Company, Vermont.

Meadows, D. H., Randers, J., and Meadows, D. L. (2004). *Limits to Growth: The 30 year update.* Chelsea Green Publishing Company, Vermont.

Meadows, D. H., Richardson, J., and Bruckman, G. (1982). *Groping in the dark: the first decade of global modelling.* John Wiley & Sons Ltd., New York.

Merry, A. (2003). *Perth County Groundwater Study Final Report.* Waterloo Hydrogeologic, Inc., Waterloo, Ontario, Canada.

Mitchell, B. (2005). "Integrated water resource management, institutional arrangements, and land-use planning." *Environment and Planning A*, 37, 1335–1352.

MMAH (2002). *London Regional Analysis.* Planning Policy Section, Provincial Planning and Environmental Services Branch, Ministry of Municipal Affairs and Housing, London, Ontario, Canada.

Mortsch, L. (2006). "Assessment of water resources risk and vulnerability to changing climatic conditions: Floodplain mapping in the Upper Thames River basin as a result of changed climate." *Report No. XI*, Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

OLWR (2003). *Ontario Low Water Responce Manual.* Ontario Ministry of Natural Resources, Queen's Printer for Ontario.

Palmer, R. N., Clancy, E., VanRheenen, N. T., and Wiley, M. W. (2004). *The Impacts of Climate Change on The Tualatin River Basin Water Supply: An Investigation into Projected Hydrologic and Management Impacts.* Department of Civil and Environmental Engineering, University of Washington, Seattle, Washingtom.

Phelan, S. (2001). "What is complexity science, really?." *Emergence*, 30(1), 120–136.

Prodanovic, P. and Simonovic, S. P. (2006a). "Assessment of water resources risk and vulnerability to changing climatic conditions: Inverse flood risk modelling of the Upper Thames River basin." *Report No. VIII*, Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

Prodanovic, P. and Simonovic, S. P. (2006b). "Systems approach to assessment of climatic change in small river basins." *XXIII Conference of the Danubian Countries on the Hydrological Forecasting and Hydrological Bases of Water Management*, Belgrade, Republic of Serbia.

Pryce, R. (2004). "Hydrological low flow indices and their uses." *Report No. WSC 04-2004*, Watershed Science Centre, Trent University, Peterborough, Ontario.

Richardson, G. P. (1991). *Feedback Thought in Social Science and Systems Theory.* University of Pennsylvania Press, Philadelphia, Pennsylvania.

Saysel, A. K., Barlas, Y., and Yenigün, O. (2002). "Environmental sustainability in an agricultural development project: a System Dynamics approach." *Journal of Environmental Management*, 64, 247–260.

Schroeder, W. W., Sweeney, R. E., and Alfeld, L. E. (1975). *Readings in Urban Dynamics: Volume 2.* Wright-Allen Press, Inc., Cambridge, Massachusetts.

Sehlke, G. and Jacobson, J. (2005). "System dynamics modeling of transboundary systems: The Bear River Basin model." *Ground Water*, 43(5), 722–730.

Senge, P. M. (1990). *The Fifth Discipline: The Art & Practise of the Learning Organization.* Doubleday Currency Press, New York.

Senge, P. M., Kleiner, A., Robers, C., Ross, R. B., and Smith, B. J. (1994). *The Fifth Discipline Fieldbook: Strategies and tools for building a learning organization.* Doubleday Currency Press, New York.

Sharif, M. and Burn, D. H. (2004). "Assessment of water resources risk and vulnerability to changing climatic conditions: Development and application of a K-NN weather generating model." *Report No. III*, Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

Sharif, M. and Burn, D. H. (2006a). "Simulating climate change scenarios using an improved K-nearest neighbir model." *Journal of Hydrology*, 325, 179–196.

Sharif, M. and Burn, D. H. (2006b). "Vulnerability assessment of Upper Thames Basin to climate change scenarios predicted by global circulation models." *EWRI-ASCE International Perspective on Environmental and Water Resources Conference*, New Delhi, India.

Sharif, M. and Burn, D. H. (2007). "Improved k-nearest neighbor weather generating model." *ASCE Journal of Hydrologic Engineering*, 12(1), 42–51.

Simonovic, S. P. (2002). "World water dynamics: global modeling of water resources." *Journal of Environmental Management*, 66, 249–267.

Simonovic, S. P. and Davies, E. G. (2006). "Are we modelling impacts of climatic change properly." *Hydrological Processes*, 20, 431–433.

Simonovic, S. P. and Fahmy, H. (1999). "A new modling approach for water resources policy analysis." *Water Resources Research*, 35(1), 295–304.

Simonovic, S. P., Fahmy, H., and El-Shorbagy, A. (1997). "The use of object-oriented modeling for water resources planning in Egypt." *Water Resources Management*, 11, 243–261.

Simonovic, S. P. and Li, L. (2003). "Methodology for assessment of climate change impacts on large-scale flood protection system." *ASCE Journal of Water Resources Planning and Management*, 129(5), 361–371.

SNC-Lavalin and RV-Anderson (2005). *Pollution Control Plant (PCP) Discharge Strategy (for the City of London)*. SNC Lavalin Engineers and Constructors and RV Anderson Associated Limited, London, Ontario, Canada.

Snyder, P., Delire, D., and Foley, J. (2004). "Evaluating the influence of different vegitation biomes on the global climate." *Climate Dynamics*, 23, 279–302.

Soong, D., Straub, T., and Murphy, E. (2005). "Continuous hydrologic simulation and flood-frequency, hydraulic, and flood-hazard analysis of the Blackberry Creek Watershed, Kane County, Illinois." *Report No. 2005-5270*, United States Geological Survey, Reston, Virginia.

STATCAN (2001). *2001 Census: Community Profiles*. Statistics Canada, Ottawa, Ontario, Canada.

Stave, K. A. (2003). "A system dynamics model to facilitate public understanding of water management options in Las Vegas, Nevada." *Journal of Environmental Management*, 67, 303–313.

Sterman, J. D. (2000). *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Irwin McGraw-Hill, Boston, Massachusetts.

USACE (2000). *Hydrologic Modelling System HEC–HMS, Technical reference manual*. United States Army Corps of Engineers, Hydrologic Engineering Center, Davis, California.

USACE (2006). *Hydrologic Modelling System HEC–HMS, User's Manual for version 3.0.1*. United States Army Corps of Engineers, Hydrologic Engineering Center, Davis, California.

UTRCA (2001). *The Upper Thames River Watershed: Report Cards 2001*. Upper Thames River Conservation Authority, London, Ontario, Canada.

Wilcox, I., Quinlan, C., Rogers, C., Troughton, M., McCallum, I., Quenneville, A., Heagy, E., and Dool, D. (1998). *The Thames River Watershed: A Background Study for Nomination under the Canadian Heritage Rivers System.* Upper Thames River Conservation Authority, London, Ontario, Canada.

Winz, I. (2005). "A system dynamics approach to sustainable urban development." *The 23rd International Conference of the System Dynamics Society*, Cambridge, Massachusetts.

Appendix

A   IPCC Scenarios

The following is taken from IPCC (2001) and represent four main families of climate change scenarios. The scenarios used in this report are based on B1 and B2.

The A1 storyline and scenario family describes a future world of very rapid economic growth, global population that peaks in mid-century and declines thereafter, and the rapid introduction of new and more efficient technologies. Major underlying themes are convergence among regions, capacity building, and increased cultural and social interactions, with a substantial reduction in regional differences in per capita income. The A1 scenario family develops into three groups that describe alternative directions of technological change in the energy system. The three A1 groups are distinguished by their technological emphasis: fossil intensive (A1FI), non-fossil energy sources (A1T), or a balance across all sources (A1B).

The A2 storyline and scenario family describes a very heterogeneous world. The underlying theme is self-reliance and preservation of local identities. Fertility patterns across regions converge very slowly, which results in continuously increasing global population. Economic development is primarily regionally oriented and per capita economic growth and technological change are more fragmented and slower than in other story lines.

The B1 storyline and scenario family describes a convergent world with the same global population that peaks in mid- century and declines thereafter, as in the A1 storyline, but with rapid changes in economic structures toward a service and information economy, with reductions in material intensity, and the introduction of clean and resource-efficient technologies. The emphasis is on global solutions to economic, social, and environmental sustainability, including improved equity, but without additional climate initiatives.

The B2 storyline and scenario family describes a world in which the emphasis is on local solutions to economic, social, and environmental sustainability. It is a world with continuously increasing global population at a rate lower than A2, intermediate levels of economic development, and less rapid and more diverse technological change than in the B1 and A1 story lines. While the scenario is also oriented toward environmental protection and social equity, it focuses on local and regional levels.

B   Model Manual

B.1   Introduction and Background

This manual is meant to provide fine technical details on the Upper Thames Systems Model developed as part of this study. As such, general descriptions, and justification of methodologies and techniques used will not be provided here. These items are covered in the main body of the report, which this appendix supplements.

B.2   Overall Model Architecture

This manual does cover, however, the details needed to better understand the inner functioning of the model. A causal loop diagram, representing a simplified view of the overall model structure, is shown in Figure 36. The diagram shows major feedback loops between the hydrologic (left) and the socio-economic (right) model components. For example, the hydrologic component is used to compute direct runoff, of the flow in area's rivers. This runoff is then used in the computation of flood damage, which influence functioning of the population and business sectors through an investment function, where flood damage leads to increased investment to rebuild.

Growth of population and business sectors places emphasis on conversion of agricultural to urban land, thus directly increasing runoff and contributing to further increases of flood damage. The same logic is used to provide explanations to feedback links between land use and hydrology (through urban land and vegetative cover), and hydrology to socio-economic factors (through available ground water recharge, floods and droughts).

B.3   Hydrologic Model Component

The hydrological model schematic is given in Figure 14. The component is of continuous hydrologic nature, representing detailed relationships of movement of moisture from the canopy, surface, soil and ground. It contains thirty two spacial units, or sub-catchments, covering an area of about 3,500 km^2. The main components of the model are junctions, river reaches, sub basins and reservoirs. The model operates on a six hour time step.

B.4   Socio-Economic Model Component

The socio-economic model component is presented in Figure 16. This figure can be used to gain understanding of the interconnectedness between different sectors of the model. Detailed level and rate diagrams (see Figures 20-35) can be useful if one wishes to interpret socio-economic model equations. This model component contains three spacial units: Middlesex, Oxford and Perth Counties. It is noted that each

spacial unit represents that part of a county that is in the basin. The socio-economic model component operates on a monthly time step.

B.5 Class Diagrams

Lastly, class diagrams are presented in Figures 54-56. These are needed by those wishing to understand the computer code generated as part of this project.

B.6 Executing the Model

The complete code need to run and execute the model is listed in this Appendix, and is available for download through the project web site (it can also be obtained by personally contacting the author).

Before the model can be executed, the Java programming language must be installed. The reader is referred to Sun's website: `http://java.sun.com` for latest versions of the compiler, as well as installation instructions.

The code was written in a program called JEdit, a simple and powerful Java text editor. It is available for free from: `http://www.jedit.org/`. The code written as part of this project utilizes JEdit's folds feature—which involves temporarily hiding chunks of code. It is strongly recommended that this text editor be used to view the code.

B.7 Directory Structure and Model Files

The files needed to run the model are located in the directory called `JavaContinuous`, under the root directory. Table 8 shows the directory structure of the model, together with its brief description.

Table 9-10 provides a list of file names, together with a brief description of their contents.

B.8 Socio-Economic Scenarios

This section explains the modification to the code that are necessary in order to reproduce the results of the socio-economic scenarios. The file `SysModel.java` contains the structure for the scenario Base. To produce scenarios presented in the report, few lines of code need to be modified from this file.

B.8.1 Scenario: InfWat

This is the scenario where it is assumed that actors in the system think infinitely amount of water is available for their use. In essence, this means that all water availability multipliers are set to unity. The following code needs to be added to the file `SysModel.java`, after the specified line number:

Class: mainCont9.java
Executes the combined model; specifies input files; contains all feedback links; processes and outputs all results

Class: HydModel.java
Hydrologic component assembly; all model parameters and hydrologic input data

Class: ModelDate.java
Keeps track of the date and time as simulation goes on

Class: SysModel.java
Socio-Economic component; all equations, parameters and table functions are here

Class: AnnualSeries2.java
Extracts annual extremes from the HydModel for selected stations only

Class: AnnualSeries.java
Extracts annual extremes from the HydModel for all stream gauge stations

Class: contUtils.java
Collection of various utilities for pre/post processing model output

Class: DataWriter.java
Writes model output to .txt and .csv files

Figure 54: Overall model class structure
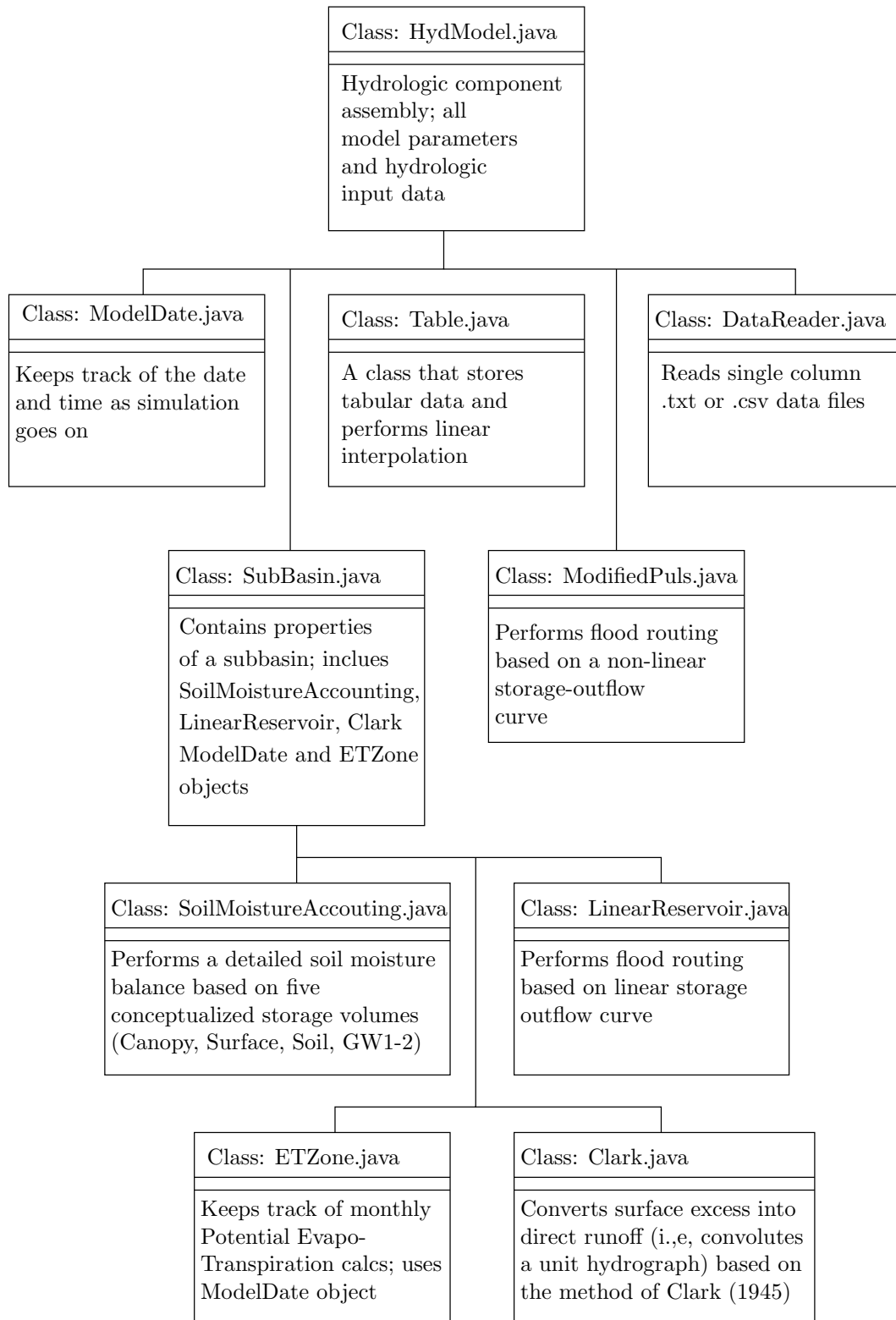
Figure 55: Class structure of the hydrologic model component
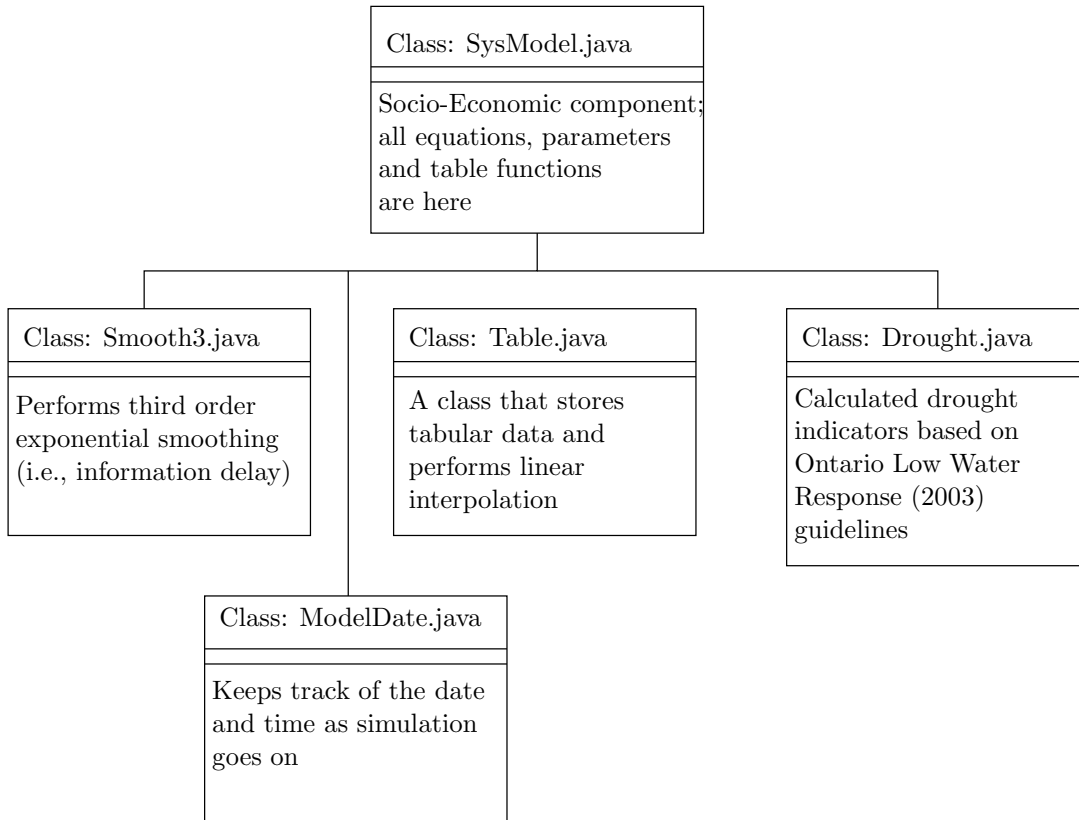
Figure 56: Class structure of the socio-economic model component

Table 8: Directory structure of the model

Name	Description
`JavaContinuous`	Root directory
`Results`	Simulation results
`SDScenarios`	Socio-economic scenario files
`Data`	Input data
`Daily`	Daily data
`Historical`	Historical daily data (1964-2001)
`HistoricalSmall`	Historical daily data (1984-2001)
`WGScenarios`	Synthetic Data
`B11`	Decreasing WG Scenario
`B21`	Increasing WG Scenario
`Historic`	Historic WG Scenario

Table 9: Model files

Name	Description
clearClassFiles.bat	Clears unnecessary files after a run is made.
jarThisFolder.bat	Makes a backup file of all .java files.
AnnualSeries.java	Extracts annual extremes from all stream gauge stations.
AnnualSeries2.java	Extracts annual extremes from Byron, St. Marys, and Ingersoll stations only.
Clark.java	Convolutes the Unit Hydrograph based upon the method of Clark (1945).
contUtils.java	Various static methods that are used for pre processing input data or post processing results.
DataReader.java	Reads input data from a file. When reading numbers, make sure the file contains a single column of numbers only.
DataWriter.java	Writes the data to a .txt file.
Drought.java	Calculates drought indicators based on local guidelines.
ETZone.java	Keeps track of monthly potential evapotranspiration.
eventUtils.java†	Various static method used for post processing event model output.
HydModel.java	Hydrologic model assembled, with all parameters.
LinearReservoir.java	Perform flood routing based on a series of linear reservoirs.
mainCont8.java	Execution file for the hydrologic model by itself, without any feedback links.
mainCont9.java	Execution file for the combined model, with all feedback links included.
mainEvent.java†	Execution file for the post processing event model output.
mainStorm5.java†	Execution file for the model that creates spacial event storms.

† Not used in the Upper Thames Systems Model, but may be useful when processing various types of hydrologic data.

Table 10: Model files, part II

Name	Description
mainUtils.java	Execution file for post processing.
ModelDate.java	Keeps track of date and time as simulation goes on.
ModifiedPuls.java	Performs reservoir routing based on the level pool method.
Smooth3.java	Performs third order exponential smoothing; used as information delay in system dynamics model.
SoilMoistureAccounting.java	Keeps track of a detailed soil moisture balance. Based on the algorithms under the same name from USACE HEC.
SubBasin.java	Contains properties of each subbasin used in the hydrologic model.
SysModel.java	Socio-economic model component assembled, with all parameters and table functions.
Table.java	Keeps track of tabular data; interpolates linearly tabular data.

- After line 807, add BusinessWaterAvailabilityMultiplier[i] = 1.0;

- After line 903, add UrbanResidentialWaterAvailabilityAttractivness[i] = 1.0;

- After line 1090, add RuralResidentialWaterAvailabilityAttractivness[i] = 1.0;

- After line 1168, add FarmWaterAvailabilityMultiplier[i] = 1.0;

B.8.2 Scenario: RedWat

This scenario is one which assumes the actors in the system voluntary agree to reduce their water consumption at time zero by thirty percent. In file `SysModel.java` multiply the arrays in variables BusinessSWUse2001, BusinessGWUse2001, UrbanResidentialSWUse2001, UrbanResidentialGWUse2001, FarmSWUse2001, FarmGWUse2001, RuralResidentialSWUse2001, RuralResidentialGWUse2001 by 0.7.

B.8.3 Scenario: RedWatLimLand

This scenario is the same as RedWat, with the following addition: The variables AgriculturalToResidentialRezoningNormal and AgriculturalToBusinessRezoningNormal are divided by 10 to reduce the normal rate of conversion of agricultural land to either business or residential purposes.

B.8.4 Scenario: SwitchToSW

This scenario is one where a switch from GW to SW is made at the start of the simulation. This involved adjustment of the following variables:

- SWPumpedIn2001 = {69520449, 73788804, 11990388};

- PumpingRatioRelativeTo2001 = {2.0, 2.0, 2.0};

- BusinessSWUse2001 = {41738369, 55179970, 4580542};

- BusinessGWUse2001 = {0.0, 0.0, 0.0};

- FractionTotalSWAllocatedToUrbanBusiness ={0.380, 0.748, 0.382};

- FractionTotalGWAllocatedToUrbanBusiness = {0.0, 0.0, 0.0};

- UrbanResidentialSWUse2001 = {21086391, 14419325, 4172414};

- UrbanResidentialGWUse2001 = {0.0, 0.0, 0.0};

- FractionTotalSWAllocatedToUrbanPopulation = {0.253, 0.195, 0.348};

- FractionTotalGWAllocatedToUrbanPopulation = {0.0, 0.0, 0.0};

- RuralResidentialSWUse2001 = {1938861, 740921, 783393};

- RuralResidentialGWUse2001 = {0.0, 0.0, 0.0};

- FractionTotalSWAllocatedToRuralResidential = {0.104, 0.010, 0.065};

- FractionTotalGWAllocatedToRuralResidential = {0.0, 0.0, 0.0};

- FarmSWUse2001 = {4756829, 3448588, 2454040};

- FarmGWUse2001 = {0.0,0.0, 0.0};

- FractionTotalSWAllocatedToFarmUnits = {0.263, 0.047, 0.205};

- FractionTotalGWAllocatedToFarmUnits = {0.0, 0.0, 0.0};

C Model Results

Model results are available for all combinations of climatic and socio-economic scenarios, for three spacial units (Middlesex, Oxford and Perth Counties). In the file ResultsAppendix.pdf, the state of the basin plots are shown first for each combination of socio-economic and climatic scenarios. After this, comparison scenarios are presented. The first set shows results where where socio-economic scenarios are kept fixed, while varying the climatic scenarios; the second set shows results when climatic scenarios are kept fixed, while altering the socio-economic scenarios. Tables showing page numbers of each possible scenario and comparison plot are presented for convenience at the beginning of the ResultsAppendix.pdf file.

D   Program Listing

D.1   mainCont9.java

```java
import java.io.*;

/**
 *  This is a class that executes the combined model, and performs all analysis
 *
 * @author      Predrag Prodanovic
 * @created     June 15, 2006
 */
public class mainCont9 {
/**
 *  The main program for the mainCont9 class
 *
 * @param  args              The command line arguments
 * @exception  IOException  Input Output Exception
 */
public static void main(String[] args) throws IOException {

// to measure ellapsed time
// Get current time
long start = System.currentTimeMillis();
System.out.println("Simulation running ... please wait ...");

// main will have only two objects; HydModel and SysModel;
// the links between the two will be here

// set initial date as 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 1;
int startHour = 1;
int startMinute = 0;
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

// time step of the HydModel, in [hrs]; must be an integer value
// and must stay at 6 [hrs], because for some of the subbasins
// Clark's time of concentration is 6 [hrs].
int userTimeStep = 6;

// the first number is a number of days in a year, multiplied
// by the number of years, then a number of leap years per century
// is added to this number (24 leap years per century), multiplied by the number of time
// steps per day
int num6hrTimeSteps = ((365 * 99) + 24) * (24 / userTimeStep);

// WG scenario name: Historic, B11, or B21
String scenario = "Historic";

// get the current directory (where the main file is located)
String curDir = new File(new File("mainCont9.java").
getAbsolutePath()).getParentFile().getAbsolutePath();

// this is the data for historically identical WG Scenario
String inputDir = curDir + "//Data//Daily//WGScenarios//" +
scenario + "//";
String outputDir = curDir + "//Data//Daily//WGScenarios//" +
scenario + "//IntermediateFiles//";
```

```
// the number of user time steps in any given month of the year
int userTimeStepsInMonth = 0;

// time step of the SysModel, in [months]
int sysTimeStep = 1;

// so that the output files plot years instead of date
double yearCounter = 1.0 / 12.0;

// feedback link variable declaration; these variables get sent
// to the SysModel, and are calculated once a month
//{{{
// the stream gauges that the SysModel gets from the HydModel
// these gauges are in units of [$1,000/yr]; these are calculated
// once every month
double jnByronSGDamage = 0.0;
double jnIngersollSGDamage = 0.0;
double jnStMarysSGDamage = 0.0;

// the GWRecharge values that the SysModel gets from the HydModel
// the units are in [m3/yr]; these are calculated once every month
double GWRechargeMiddlesex = 0.0;
double GWRechargeOxford = 0.0;
double GWRechargePerth = 0.0;

// the drought indicators that the SysModel gets from the HydModel
// as before, these are computed once every month
// computed monthly flow, in units of [cms]
double jnByronSG = 0.0;
double jnIngersollSG = 0.0;
double jnStMarysSG = 0.0;

// this is the accumulated monthly ppt
// for Byron, take sb30ppt, for Ingersoll take sb21ppt, and for
// StMarys take sb11ppt
double jnByronPPT = 0.0;
double jnIngersollPPT = 0.0;
double jnStMarysPPT = 0.0;

// this is where the time of concentration, percent of
// imperviousness and PET reduction ratio values are outputted
double[] TCValues;
double[] ImpValues;
double[] MSSValues;
double[] MSIValues;
double[] PETValues;
//}}}

// daily output variable declarations, in [cms]
//{{{
double jnMitchellDaily = 0.0;
double jnAvonDaily = 0.0;
double jnDownStreamWildwood = 0.0;
double jnStMarysDaily = 0.0;
double jnPloverMillsDaily = 0.0;
double jnDownStreamFanshawe = 0.0;
double jnMedwayDaily = 0.0;
double jnInnerkipDaily = 0.0;
double jnDownStreamPittock = 0.0;
double jnCedarDaily = 0.0;
double jnIngersollDaily = 0.0;
double jnThamesfordDaily = 0.0;
double jnReynoldsDaily = 0.0;
```

```
double jnWaubunoDaily = 0.0;
double jnEalingDaily = 0.0;
double jnByronDaily = 0.0;
double jnOxbowDaily = 0.0;
double jnDingmanDaily = 0.0;
//}}}

// define the HydModel, and initialize it
HydModel utHyd = new HydModel(currentDate, userTimeStep);
utHyd.initialize(inputDir, outputDir);

// define the SysModel object, but initialize it only after a month
SysModel utSys = new SysModel(currentDate, sysTimeStep);

// define AnnualSeries and AnnualSeries2 objects
//{{{

// these are the AnnualSeries2 objects, which simply extract
// annual extremes one junction at a time; because of how the
// AnnualSeries2 object is set up, the file names can not be
// changed---they are embedded into the object itself.  This is
// bad programming, but I am too lazy to make it more efficient

// daily maximum flows
AnnualSeries2 saDailyByron = new AnnualSeries2(utHyd,
currentDate, outputDir, "ByronAnnualMaxDailyFlow.txt");
AnnualSeries2 saDailyStMarys = new AnnualSeries2(utHyd,
currentDate, outputDir, "StMarysAnnualMaxDailyFlow.txt");
AnnualSeries2 saDailyIngersoll = new AnnualSeries2(utHyd,
currentDate, outputDir, "IngersollAnnualMaxDailyFlow.txt");

// 7 day minimum flows
AnnualSeries2 sa7DayByron = new AnnualSeries2(utHyd,
currentDate, outputDir, "ByronAnnualMin7DayFlow.txt");
AnnualSeries2 sa7DayStMarys = new AnnualSeries2(utHyd,
currentDate, outputDir, "StMarysAnnualMin7DayFlow.txt");
AnnualSeries2 sa7DayIngersoll = new AnnualSeries2(utHyd,
currentDate, outputDir, "IngersollAnnualMin7DayFlow.txt");

// monthly minimum flows
AnnualSeries2 saMonthlyByron = new AnnualSeries2(utHyd,
currentDate, outputDir, "ByronAnnualMinMonthlyFlow.txt");
AnnualSeries2 saMonthlyStMarys = new AnnualSeries2(utHyd,
currentDate, outputDir, "StMarysAnnualMinMonthlyFlow.txt");
AnnualSeries2 saMonthlyIngersoll = new AnnualSeries2(utHyd,
currentDate, outputDir, "IngersollAnnualMinMonthlyFlow.txt");

// these are the original AnnualSeries objects, that extract
// annual extremes for all junctions, and compute their timing
// and regularity
AnnualSeries saDaily = new AnnualSeries(utHyd,
currentDate, outputDir + "DailyMaxFlow.txt");

AnnualSeries sa7Day = new AnnualSeries(utHyd,
currentDate, outputDir + "SevenDayMinFlow.txt");

AnnualSeries saMonthly = new AnnualSeries(utHyd,
currentDate, outputDir + "MonthlyMinFlow.txt");
//}}}

// DataWriter objects
//{{{
DataWriter jnOutDaily = new DataWriter(outputDir + "junctionsDaily.txt");
```

```
jnOutDaily.writeData("Date, MitchellSG, jnAvon, jnDownStreamWildwood, " +
"StMarysSG, jnPloverMillsSG, jnDownStreamFanshawe, " +
"jnMedwaySG, jnInnerkipSG, jnDownStreamPittock, " +
"jnCedarSG, jnIngersollSG, jnThamesfordSG, " +
"jnReynoldsSG, jnWaubunoSG, JnEalingSG, jnByronSG, " +
"jnOxbowSG, jnDingmanSG, sb1MaxSoilInfil");

DataWriter jnOutMonthly = new DataWriter(outputDir + "monthlyDamages.txt");
jnOutMonthly.writeData("Date,  " +
"FDIM M, PFDIM M, FDIM O, PFDIM O, FDIM P, PFDIM P, " +
"ByronDamage, IngersollDamage, " +
"StMarysDamage, GWR M (m3/yr), " +
"GWR O (m3/yr), GWR P (m3/yr), ByronPPT, " +
"IngersollPPT, St.MarysPPT, ByronSG, IngersollSG, " +
"StMarysSG, sb1 MaxSurfStore");

// these files output daily ppt and flow for three locations in
// the basin for use in the continuous inverse link
DataWriter flowDailyStMarys = new DataWriter(outputDir +
"StMarysDailyFlow.txt");
DataWriter flowDailyIngersoll = new DataWriter(outputDir +
"IngersollDailyFlow.txt");
DataWriter flowDailyByron = new DataWriter(outputDir +
"ByronDailyFlow.txt");

DataWriter sysOut = new DataWriter(outputDir + "SystemDynamicsOutput.txt");
sysOut.writeData("Year, BS M, BS O, BS P, BLM M, BLM O, BLM P, " +
"BLFM M, BLFM O, BLFM P, BWAM M, BWAM O, BWAM P, " +
"UH M, UH O, UH P, UHAM M, UHAM O, UHAM P, " +
"UHLM M, UHLM O, UHLM P, UP M, UP O, UP P, " +
"AUJM M, AUJM O, AUJM P, AUHM M, AUHM O, AUHM P, " +
"URWAA M, URWAA O, URWAA P, FC M, FC O, FC P, " +
"AL M, AL O, AL P, RL M, RL O, RL P, BL M, BL O, BL P, " +
"PGWR M, PGWR O, PGWR P, MASWP M, MASWP O, MASWP P, " +
"WURR M, WURR O, WURR P, PFDM M, PFDM O, PFDM P, " +
"FU M, FU O, FU P, ACLM M, ACLM O, ACLM P, " +
"RLFM M, RLFM O, RLFM P, FWAM M, FWAM O, FWAM P, " +
"RP M, RP O, RP P, ARJM M, ARJM O, ARJM P, " +
"RRWAM M, RRWAM O, RRWAM P, TGWU M, TGWU O, TGWU P, " +
"TSWU M, TSWU O, TSWU P");
//}}}

for (int i = 1; i < num6hrTimeSteps; i++) {

currentDate.incrementDateByHours(userTimeStep);
utHyd.update(currentDate);

// to extract annual statistics from the hydrological model
//{{{
// AnnualSeries2 objects are updated here
// daily maximum flows
saDailyByron.updateDaily(utHyd, currentDate);
saDailyStMarys.updateDaily(utHyd, currentDate);
saDailyIngersoll.updateDaily(utHyd, currentDate);

// 7 day minimum flows
sa7DayByron.update7Day(utHyd, currentDate);
sa7DayStMarys.update7Day(utHyd, currentDate);
sa7DayIngersoll.update7Day(utHyd, currentDate);

// monthly minimum flows
saMonthlyByron.updateMonthly(utHyd, currentDate);
saMonthlyStMarys.updateMonthly(utHyd, currentDate);
saMonthlyIngersoll.updateMonthly(utHyd, currentDate);
```

```
// AnnualSeries objects are updated here
saDaily.updateDaily(utHyd, currentDate);
sa7Day.update7Day(utHyd, currentDate);
saMonthly.updateMonthly(utHyd, currentDate);
//}}}

// daily hydrologic model output
//{{{
// accumulates the 6hr hydrological output to daily values
jnMitchellDaily = jnMitchellDaily + utHyd.getJnMitchellSG();
jnAvonDaily = jnAvonDaily + utHyd.getJnAvonSG();
jnDownStreamWildwood = jnDownStreamWildwood + utHyd.getJnDownStreamWildwood();
jnStMarysDaily = jnStMarysDaily + utHyd.getJnStMarysSG();
jnPloverMillsDaily = jnPloverMillsDaily + utHyd.getJnPloverMillsSG();
jnDownStreamFanshawe = jnDownStreamFanshawe + utHyd.getJnDownStreamFanshawe();
jnMedwayDaily = jnMedwayDaily + utHyd.getJnMedwaySG();
jnInnerkipDaily = jnInnerkipDaily + utHyd.getJnUpStreamPittock();
jnDownStreamPittock = jnDownStreamPittock + utHyd.getJnDownStreamPittock();
jnCedarDaily = jnCedarDaily + utHyd.getJnCedarSG();
jnIngersollDaily = jnIngersollDaily + utHyd.getJnIngersollSG();
jnThamesfordDaily = jnThamesfordDaily + utHyd.getJnThamesfordSG();
jnReynoldsDaily = jnReynoldsDaily + utHyd.getJnReynoldsSG();
jnWaubunoDaily = jnWaubunoDaily + utHyd.getJnWaubunoSG();
jnEalingDaily = jnEalingDaily + utHyd.getJnEalingSG();
jnByronDaily = jnByronDaily + utHyd.getJnByronSG();
jnOxbowDaily = jnOxbowDaily + utHyd.getJnOxbowSG();
jnDingmanDaily = jnDingmanDaily + utHyd.getJnDingmanSG();

if (currentDate.getHour() >= 19) {
// this averages four 6hr intervals into one
// daily value
jnMitchellDaily = jnMitchellDaily / 4.0;
jnAvonDaily = jnAvonDaily / 4.0;
jnDownStreamWildwood = jnDownStreamWildwood / 4.0;
jnStMarysDaily = jnStMarysDaily / 4.0;
jnPloverMillsDaily = jnPloverMillsDaily / 4.0;
jnDownStreamFanshawe = jnDownStreamFanshawe / 4.0;
jnMedwayDaily = jnMedwayDaily / 4.0;
jnInnerkipDaily = jnInnerkipDaily / 4.0;
jnDownStreamPittock = jnDownStreamPittock / 4.0;
jnCedarDaily = jnCedarDaily / 4.0;
jnIngersollDaily = jnIngersollDaily / 4.0;
jnThamesfordDaily = jnThamesfordDaily / 4.0;
jnReynoldsDaily = jnReynoldsDaily / 4.0;
jnWaubunoDaily = jnWaubunoDaily / 4.0;
jnEalingDaily = jnEalingDaily / 4.0;
jnByronDaily = jnByronDaily / 4.0;
jnOxbowDaily = jnOxbowDaily / 4.0;
jnDingmanDaily = jnDingmanDaily / 4.0;

// outputs the daily flow values
// outputs the daily flows for three stations only
flowDailyStMarys.writeData(jnStMarysDaily);
flowDailyIngersoll.writeData(jnIngersollDaily);
flowDailyByron.writeData(jnByronDaily);

// outputs the daily flow values to all stations
/*
 *  jnOutDaily.writeData(currentDate.getDate() + ", " +
 *  jnMitchellDaily + ", " +
 *  jnAvonDaily + ", " +
 *  jnDownStreamWildwood + ", " +
 *  jnStMarysDaily + ", " +
```

```
 *   jnPloverMillsDaily + ", " +
 *   jnDownStreamFanshawe + ", " +
 *   jnMedwayDaily + ", " +
 *   jnInnerkipDaily + ", " +
 *   jnDownStreamPittock + ", " +
 *   jnCedarDaily + ", " +
 *   jnIngersollDaily + ", " +
 *   jnThamesfordDaily + ", " +
 *   jnReynoldsDaily + ", " +
 *   jnWaubunoDaily + ", " +
 *   jnEalingDaily + ", " +
 *   jnByronDaily + ", " +
 *   jnOxbowDaily + ", " +
 *   jnDingmanDaily);
 */
// resets the flow variable
jnMitchellDaily = 0.0;
jnAvonDaily = 0.0;
jnDownStreamWildwood = 0.0;
jnStMarysDaily = 0.0;
jnPloverMillsDaily = 0.0;
jnDownStreamFanshawe = 0.0;
jnMedwayDaily = 0.0;
jnInnerkipDaily = 0.0;
jnDownStreamPittock = 0.0;
jnCedarDaily = 0.0;
jnIngersollDaily = 0.0;
jnThamesfordDaily = 0.0;
jnReynoldsDaily = 0.0;
jnWaubunoDaily = 0.0;
jnEalingDaily = 0.0;
jnByronDaily = 0.0;
jnOxbowDaily = 0.0;
jnDingmanDaily = 0.0;
}
//}}}


// accumulated the 6hr hydrological output to monthly values
//{{{
// sums up damages each 6hr time step
jnByronSGDamage = jnByronSGDamage + utHyd.getJnByronSGDamage();
jnIngersollSGDamage = jnIngersollSGDamage + utHyd.getJnIngersollSGDamage();
jnStMarysSGDamage = jnStMarysSGDamage + utHyd.getJnStMarysSGDamage();

// sums up GWRechage each 6 hr time step
GWRechargeMiddlesex = GWRechargeMiddlesex + utHyd.getGWRechargeMiddlesex();
GWRechargeOxford = GWRechargeOxford + utHyd.getGWRechargeOxford();
GWRechargePerth = GWRechargePerth + utHyd.getGWRechargePerth();

// sums up the stream gauge flow
jnByronSG = jnByronSG + utHyd.getJnByronSG();
jnIngersollSG = jnIngersollSG + utHyd.getJnIngersollSG();
jnStMarysSG = jnStMarysSG + utHyd.getJnStMarysSG();

// sums up the ppt values
jnByronPPT = jnByronPPT + utHyd.getJnByronPPT();
jnIngersollPPT = jnIngersollPPT + utHyd.getJnIngersollPPT();
jnStMarysPPT = jnStMarysPPT + utHyd.getJnStMarysPPT();
//}}}

// this is where all the feedback links are; this if
// statement is true only once per month;
//{{{
if ((currentDate.getDay() == currentDate.getDaysInMonth()) &&
```

```
(currentDate.getHour() >= 19)) {

// obtains monthly averages for the hydrologic output
//{{{
// computes the number of user times steps this month
userTimeStepsInMonth = currentDate.getDaysInMonth() *
(24 / userTimeStep);

// this is average monthly damage, in [$1000/yr]
jnByronSGDamage = (jnByronSGDamage / userTimeStepsInMonth);
jnIngersollSGDamage = (jnIngersollSGDamage / userTimeStepsInMonth);
jnStMarysSGDamage = (jnStMarysSGDamage / userTimeStepsInMonth);

// this is average monthly GWRecharge, in [m3/yr]
GWRechargeMiddlesex = GWRechargeMiddlesex / userTimeStepsInMonth;
GWRechargeOxford = GWRechargeOxford / userTimeStepsInMonth;
GWRechargePerth = GWRechargePerth / userTimeStepsInMonth;

// this is average monthly flow, in [cms]
jnByronSG = jnByronSG / userTimeStepsInMonth;
jnIngersollSG = jnIngersollSG / userTimeStepsInMonth;
jnStMarysSG = jnStMarysSG / userTimeStepsInMonth;

// the monthly ppt only gets accumulated, so there
// is not need to average it here
//}}}

// the SysModel is initialized at the end of the first month
// or at 31 Jan 2001, @ (or after) 19:00 hrs
if ((currentDate.getYear() == startYear) &&
(currentDate.getMonth() == startMonth) &&
(currentDate.getDay() > 28) &&
(currentDate.getHour() >= 19)) {

// the system dynamics model gets the flow
// this must be called before initialize()
utSys.setDamage(jnByronSGDamage, jnIngersollSGDamage,
jnStMarysSGDamage);

utSys.setGWRecharge(GWRechargeMiddlesex,
GWRechargeOxford,
GWRechargePerth);

utSys.setFlow(jnByronSG, jnIngersollSG,
jnStMarysSG);

utSys.setPPT(jnByronPPT, jnIngersollPPT,
jnStMarysPPT);

// system dynamics model is initialized
utSys.initialize();

// updates the parameters of the HydModel

// setting of the time of concentration tables
utHyd.setTimeOfConcentrationTables(
utSys.getFractionPavedLandMiddlesex(),
utSys.getFractionPavedLandOxford(),
utSys.getFractionPavedLandPerth());
/*
 *  / setting of time of concentration is done once per month
 *  utHyd.setTimeOfConcentration(
 *  utSys.getFractionPavedLandMiddlesex(),
 *  utSys.getFractionPavedLandOxford(),
```

```
 *  utSys.getFractionPavedLandPerth() );
 */
TCValues = utHyd.getTimeOfConcentration();

// setting of the percent of imperviousness tables
utHyd.setPercentImperviousnessTables(
utSys.getFractionPavedLandMiddlesex(),
utSys.getFractionPavedLandOxford(),
utSys.getFractionPavedLandPerth());
/*
 *  / setting of percent of imperviousness is done once per month
 *  utHyd.setPercentImperviousness(
 *  utSys.getFractionPavedLandMiddlesex(),
 *  utSys.getFractionPavedLandOxford(),
 *  utSys.getFractionPavedLandPerth() );
 */
ImpValues = utHyd.getImperviousness();

// setting of the reduction of max surface storage
utHyd.setMaxSurfStoreReductionTables(
utSys.getFractionPavedLandMiddlesex(),
utSys.getFractionPavedLandOxford(),
utSys.getFractionPavedLandPerth());

utHyd.setMaxSurfStoreReduction(
utSys.getFractionPavedLandMiddlesex(),
utSys.getFractionPavedLandOxford(),
utSys.getFractionPavedLandPerth());

MSSValues = utHyd.getMaxSurfStoreReduction();

utHyd.setMaxSoilInfilReductionTables(
utSys.getFractionVegetationMiddlesex(),
utSys.getFractionVegetationOxford(),
utSys.getFractionVegetationPerth());

utHyd.setMaxSoilInfilReduction(
utSys.getFractionVegetationMiddlesex(),
utSys.getFractionVegetationOxford(),
utSys.getFractionVegetationPerth());

MSIValues = utHyd.getMaxSoilInfilReduction();

utHyd.setPETTables(
utSys.getFractionVegetationMiddlesex(),
utSys.getFractionVegetationOxford(),
utSys.getFractionVegetationPerth());

utHyd.setPET(
utSys.getFractionVegetationMiddlesex(),
utSys.getFractionVegetationOxford(),
utSys.getFractionVegetationPerth());

PETValues = utHyd.getPETReductionCoeff();

// writes the initial conditions for the system
// dynamics model
//{{{
sysOut.writeData(yearCounter + ", " +
utSys.getBusStrMiddlesex() + ", " +
utSys.getBusStrOxford() + ", " +
utSys.getBusStrPerth() + ", " +
utSys.getBusinessLandMultiplierMiddlesex() + ", " +
utSys.getBusinessLandMultiplierOxford() + ", " +
```

```
        utSys.getBusinessLandMultiplierPerth() + ", " +
        utSys.getBusinessLabourForceMultiplierMiddlesex() + ", " +
        utSys.getBusinessLabourForceMultiplierOxford() + ", " +
        utSys.getBusinessLabourForceMultiplierPerth() + ", " +
        utSys.getBusinessWaterAvailabilityMultiplierMiddlesex() + ", " +
        utSys.getBusinessWaterAvailabilityMultiplierOxford() + ", " +
        utSys.getBusinessWaterAvailabilityMultiplierPerth() + ", " +
        utSys.getUrbHouMiddlesex() + ", " +
        utSys.getUrbHouOxford() + ", " +
        utSys.getUrbHouPerth() + ", " +
        utSys.getUrbanHousingAvailabilityMultiplierMiddlesex() + ", " +
        utSys.getUrbanHousingAvailabilityMultiplierOxford() + ", " +
        utSys.getUrbanHousingAvailabilityMultiplierPerth() + ", " +
        utSys.getUrbanHousingLandMultiplierMiddlesex() + ", " +
        utSys.getUrbanHousingLandMultiplierOxford() + ", " +
        utSys.getUrbanHousingLandMultiplierPerth() + ", " +
        utSys.getUrbPopMiddlesex() + ", " +
        utSys.getUrbPopOxford() + ", " +
        utSys.getUrbPopPerth() + ", " +
        utSys.getAttractivnessOfUrbanJobsMultiplierMiddlesex() + ", " +
        utSys.getAttractivnessOfUrbanJobsMultiplierOxford() + ", " +
        utSys.getAttractivnessOfUrbanJobsMultiplierPerth() + ", " +
        utSys.getAttractivnessOfUrbanHousingMultiplierMiddlesex() + ", " +
        utSys.getAttractivnessOfUrbanHousingMultiplierOxford() + ", " +
        utSys.getAttractivnessOfUrbanHousingMultiplierPerth() + ", " +
        utSys.getUrbanResidentialWaterAvailabilityAttractivnessMiddlesex() + ", " +
        utSys.getUrbanResidentialWaterAvailabilityAttractivnessOxford() + ", " +
        utSys.getUrbanResidentialWaterAvailabilityAttractivnessPerth() + ", " +
        utSys.getForCovMiddlesex() + ", " +
        utSys.getForCovOxford() + ", " +
        utSys.getForCovPerth() + ", " +
        utSys.getAgLandMiddlesex() + ", " +
        utSys.getAgLandOxford() + ", " +
        utSys.getAgLandPerth() + ", " +
        utSys.getResLandMiddlesex() + ", " +
        utSys.getResLandOxford() + ", " +
        utSys.getResLandPerth() + ", " +
        utSys.getBusLandMiddlesex() + ", " +
        utSys.getBusLandOxford() + ", " +
        utSys.getBusLandPerth() + ", " +
        utSys.getPerceivedGWRechargeMiddlesex() + ", " +
        utSys.getPerceivedGWRechargeOxford() + ", " +
        utSys.getPerceivedGWRechargePerth() + ", " +
        utSys.getMaxAllowableSWPumpingMiddlesex() + ", " +
        utSys.getMaxAllowableSWPumpingOxford() + ", " +
        utSys.getMaxAllowableSWPumpingPerth() + ", " +
        utSys.getWaterUseReductionRatioMiddlesex() + ", " +
        utSys.getWaterUseReductionRatioOxford() + ", " +
        utSys.getWaterUseReductionRatioPerth() + ", " +
        utSys.getPerceivedFloodDamageMultiplierMiddlesex() + ", " +
        utSys.getPerceivedFloodDamageMultiplierOxford() + ", " +
        utSys.getPerceivedFloodDamageMultiplierPerth() + ", " +
        utSys.getFarmsMiddlesex() + ", " +
        utSys.getFarmsOxford() + ", " +
        utSys.getFarmsPerth() + ", " +
        utSys.getAvailabilityOfCropLandMultiplierMiddlesex() + ", " +
        utSys.getAvailabilityOfCropLandMultiplierOxford() + ", " +
        utSys.getAvailabilityOfCropLandMultiplierPerth() + ", " +
        utSys.getRuralLabourForceMultiplierMiddlesex() + ", " +
        utSys.getRuralLabourForceMultiplierOxford() + ", " +
        utSys.getRuralLabourForceMultiplierPerth() + ", " +
        utSys.getFarmWaterAvailabilityMultiplierMiddlesex() + ", " +
        utSys.getFarmWaterAvailabilityMultiplierOxford() + ", " +
        utSys.getFarmWaterAvailabilityMultiplierPerth() + ", " +
```

```
utSys.getRurPopMiddlesex() + ", " +
utSys.getRurPopOxford() + ", " +
utSys.getRurPopPerth() + ", " +
utSys.getAttractivnessOfRuralJobsMultiplierMiddlesex() + ", " +
utSys.getAttractivnessOfRuralJobsMultiplierOxford() + ", " +
utSys.getAttractivnessOfRuralJobsMultiplierPerth() + ", " +
utSys.getRuralResidentialWaterAvailabilityAttractivnessMiddlesex() + ", " +
utSys.getRuralResidentialWaterAvailabilityAttractivnessOxford() + ", " +
utSys.getRuralResidentialWaterAvailabilityAttractivnessPerth() + ", " +
utSys.getTotalGWUseMiddlesex() + ", " +
utSys.getTotalGWUseOxford() + ", " +
utSys.getTotalGWUsePerth() + ", " +
utSys.getTotalSWUseMiddlesex() + ", " +
utSys.getTotalSWUseOxford() + ", " +
utSys.getTotalSWUsePerth());

// writes the monthly damage values, in [$1000/yr]
jnOutMonthly.writeData(yearCounter + ", " +
utSys.getFloodDamageMultiplierMiddlesex() + ", " +
utSys.getPerceivedFloodDamageMultiplierMiddlesex() + ", " +
utSys.getFloodDamageMultiplierOxford() + ", " +
utSys.getPerceivedFloodDamageMultiplierOxford() + ", " +
utSys.getFloodDamageMultiplierPerth() + ", " +
utSys.getPerceivedFloodDamageMultiplierPerth() + ", " +
jnByronSGDamage + ", " +
jnIngersollSGDamage + ", " +
jnStMarysSGDamage + ", " +
GWRechargeMiddlesex + ", " +
GWRechargeOxford + ", " +
GWRechargePerth + ", " +
jnByronPPT + ", " +
jnIngersollPPT + ", " +
jnStMarysPPT + ", " +
jnByronSG + ", " +
jnIngersollSG + ", " +
jnStMarysSG + ", " + utHyd.getMaxSurfStore());
//}}}

// set these to zero, so that the next time step
// can properly sum them
//{{{
jnByronSGDamage = 0.0;
jnIngersollSGDamage = 0.0;
jnStMarysSGDamage = 0.0;
GWRechargeMiddlesex = 0.0;
GWRechargeOxford = 0.0;
GWRechargePerth = 0.0;
jnByronSG = 0.0;
jnIngersollSG = 0.0;
jnStMarysSG = 0.0;
jnByronPPT = 0.0;
jnIngersollPPT = 0.0;
jnStMarysPPT = 0.0;
//}}}


} else {
// must be called immediately before update()
utSys.setDamage(jnByronSGDamage, jnIngersollSGDamage,
jnStMarysSGDamage);

utSys.setGWRecharge(GWRechargeMiddlesex,
GWRechargeOxford,
GWRechargePerth);
```

```
utSys.setFlow(jnByronSG, jnIngersollSG,
jnStMarysSG);

utSys.setPPT(jnByronPPT, jnIngersollPPT,
jnStMarysPPT);

utSys.update(currentDate);

// setting of time of concentration is done once per month
/*
 *  utHyd.setTimeOfConcentration(
 *  utSys.getFractionPavedLandMiddlesex(),
 *  utSys.getFractionPavedLandOxford(),
 *  utSys.getFractionPavedLandPerth() );
 *  / setting of percent of imperviousness is done once per month
 *  utHyd.setPercentImperviousness(
 *  utSys.getFractionPavedLandMiddlesex(),
 *  utSys.getFractionPavedLandOxford(),
 *  utSys.getFractionPavedLandPerth() );
 */
utHyd.setMaxSurfStoreReduction(
utSys.getFractionPavedLandMiddlesex(),
utSys.getFractionPavedLandOxford(),
utSys.getFractionPavedLandPerth());

utHyd.setMaxSoilInfilReduction(
utSys.getFractionVegetationMiddlesex(),
utSys.getFractionVegetationOxford(),
utSys.getFractionVegetationPerth());

// setting of PET reduction coefficient
utHyd.setPET(
utSys.getFractionVegetationMiddlesex(),
utSys.getFractionVegetationOxford(),
utSys.getFractionVegetationPerth());

TCValues = utHyd.getTimeOfConcentration();
ImpValues = utHyd.getImperviousness();
MSSValues = utHyd.getMaxSurfStoreReduction();
MSIValues = utHyd.getMaxSoilInfilReduction();
PETValues = utHyd.getPETReductionCoeff();

yearCounter = yearCounter + (1.0 / 12.0);
// writes the model output at the end of each year
//{{{
if (currentDate.getDayOfYear() ==
currentDate.getDaysInYear()) {

sysOut.writeData(yearCounter + ", " +
utSys.getBusStrMiddlesex() + ", " +
utSys.getBusStrOxford() + ", " +
utSys.getBusStrPerth() + ", " +
utSys.getBusinessLandMultiplierMiddlesex() + ", " +
utSys.getBusinessLandMultiplierOxford() + ", " +
utSys.getBusinessLandMultiplierPerth() + ", " +
utSys.getBusinessLabourForceMultiplierMiddlesex() + ", " +
utSys.getBusinessLabourForceMultiplierOxford() + ", " +
utSys.getBusinessLabourForceMultiplierPerth() + ", " +
utSys.getBusinessWaterAvailabilityMultiplierMiddlesex() + ", " +
utSys.getBusinessWaterAvailabilityMultiplierOxford() + ", " +
utSys.getBusinessWaterAvailabilityMultiplierPerth() + ", " +
utSys.getUrbHouMiddlesex() + ", " +
utSys.getUrbHouOxford() + ", " +
```

```
                    utSys.getUrbHouPerth() + ", " +
                    utSys.getUrbanHousingAvailabilityMultiplierMiddlesex() + ", " +
                    utSys.getUrbanHousingAvailabilityMultiplierOxford() + ", " +
                    utSys.getUrbanHousingAvailabilityMultiplierPerth() + ", " +
                    utSys.getUrbanHousingLandMultiplierMiddlesex() + ", " +
                    utSys.getUrbanHousingLandMultiplierOxford() + ", " +
                    utSys.getUrbanHousingLandMultiplierPerth() + ", " +
                    utSys.getUrbPopMiddlesex() + ", " +
                    utSys.getUrbPopOxford() + ", " +
                    utSys.getUrbPopPerth() + ", " +
                    utSys.getAttractivnessOfUrbanJobsMultiplierMiddlesex() + ", " +
                    utSys.getAttractivnessOfUrbanJobsMultiplierOxford() + ", " +
                    utSys.getAttractivnessOfUrbanJobsMultiplierPerth() + ", " +
                    utSys.getAttractivnessOfUrbanHousingMultiplierMiddlesex() + ", " +
                    utSys.getAttractivnessOfUrbanHousingMultiplierOxford() + ", " +
                    utSys.getAttractivnessOfUrbanHousingMultiplierPerth() + ", " +
                    utSys.getUrbanResidentialWaterAvailabilityAttractivnessMiddlesex() + ", " +
                    utSys.getUrbanResidentialWaterAvailabilityAttractivnessOxford() + ", " +
                    utSys.getUrbanResidentialWaterAvailabilityAttractivnessPerth() + ", " +
                    utSys.getForCovMiddlesex() + ", " +
                    utSys.getForCovOxford() + ", " +
                    utSys.getForCovPerth() + ", " +
                    utSys.getAgLandMiddlesex() + ", " +
                    utSys.getAgLandOxford() + ", " +
                    utSys.getAgLandPerth() + ", " +
                    utSys.getResLandMiddlesex() + ", " +
                    utSys.getResLandOxford() + ", " +
                    utSys.getResLandPerth() + ", " +
                    utSys.getBusLandMiddlesex() + ", " +
                    utSys.getBusLandOxford() + ", " +
                    utSys.getBusLandPerth() + ", " +
                    utSys.getPerceivedGWRechargeMiddlesex() + ", " +
                    utSys.getPerceivedGWRechargeOxford() + ", " +
                    utSys.getPerceivedGWRechargePerth() + ", " +
                    utSys.getMaxAllowableSWPumpingMiddlesex() + ", " +
                    utSys.getMaxAllowableSWPumpingOxford() + ", " +
                    utSys.getMaxAllowableSWPumpingPerth() + ", " +
                    utSys.getWaterUseReductionRatioMiddlesex() + ", " +
                    utSys.getWaterUseReductionRatioOxford() + ", " +
                    utSys.getWaterUseReductionRatioPerth() + ", " +
                    utSys.getPerceivedFloodDamageMultiplierMiddlesex() + ", " +
                    utSys.getPerceivedFloodDamageMultiplierOxford() + ", " +
                    utSys.getPerceivedFloodDamageMultiplierPerth() + ", " +
                    utSys.getFarmsMiddlesex() + ", " +
                    utSys.getFarmsOxford() + ", " +
                    utSys.getFarmsPerth() + ", " +
                    utSys.getAvailabilityOfCropLandMultiplierMiddlesex() + ", " +
                    utSys.getAvailabilityOfCropLandMultiplierOxford() + ", " +
                    utSys.getAvailabilityOfCropLandMultiplierPerth() + ", " +
                    utSys.getRuralLabourForceMultiplierMiddlesex() + ", " +
                    utSys.getRuralLabourForceMultiplierOxford() + ", " +
                    utSys.getRuralLabourForceMultiplierPerth() + ", " +
                    utSys.getFarmWaterAvailabilityMultiplierMiddlesex() + ", " +
                    utSys.getFarmWaterAvailabilityMultiplierOxford() + ", " +
                    utSys.getFarmWaterAvailabilityMultiplierPerth() + ", " +
                    utSys.getRurPopMiddlesex() + ", " +
                    utSys.getRurPopOxford() + ", " +
                    utSys.getRurPopPerth() + ", " +
                    utSys.getAttractivnessOfRuralJobsMultiplierMiddlesex() + ", " +
                    utSys.getAttractivnessOfRuralJobsMultiplierOxford() + ", " +
                    utSys.getAttractivnessOfRuralJobsMultiplierPerth() + ", " +
                    utSys.getRuralResidentialWaterAvailabilityAttractivnessMiddlesex() + ", " +
                    utSys.getRuralResidentialWaterAvailabilityAttractivnessOxford() + ", " +
                    utSys.getRuralResidentialWaterAvailabilityAttractivnessPerth() + ", " +
```

```
                utSys.getTotalGWUseMiddlesex() + ", " +
                utSys.getTotalGWUseOxford() + ", " +
                utSys.getTotalGWUsePerth() + ", " +
                utSys.getTotalSWUseMiddlesex() + ", " +
                utSys.getTotalSWUseOxford() + ", " +
                utSys.getTotalSWUsePerth());

                // writes the monthly damage values, in [$1000/yr]
                jnOutMonthly.writeData(yearCounter + ", " +
                utSys.getFloodDamageMultiplierMiddlesex() + ", " +
                utSys.getPerceivedFloodDamageMultiplierMiddlesex() + ", " +
                utSys.getFloodDamageMultiplierOxford() + ", " +
                utSys.getPerceivedFloodDamageMultiplierOxford() + ", " +
                utSys.getFloodDamageMultiplierPerth() + ", " +
                utSys.getPerceivedFloodDamageMultiplierPerth() + ", " +
                jnByronSGDamage + ", " +
                jnIngersollSGDamage + ", " +
                jnStMarysSGDamage + ", " +
                GWRechargeMiddlesex + ", " +
                GWRechargeOxford + ", " +
                GWRechargePerth + ", " +
                jnByronPPT + ", " +
                jnIngersollPPT + ", " +
                jnStMarysPPT + ", " +
                jnByronSG + ", " +
                jnIngersollSG + ", " +
                jnStMarysSG + ", " + utHyd.getMaxSurfStore());
                }
                //}}}

                // set these to zero, so that the next time step
                // can properly sum them
                //{{{
                jnByronSGDamage = 0.0;
                jnIngersollSGDamage = 0.0;
                jnStMarysSGDamage = 0.0;
                GWRechargeMiddlesex = 0.0;
                GWRechargeOxford = 0.0;
                GWRechargePerth = 0.0;
                jnByronSG = 0.0;
                jnIngersollSG = 0.0;
                jnStMarysSG = 0.0;
                jnByronPPT = 0.0;
                jnIngersollPPT = 0.0;
                jnStMarysPPT = 0.0;
                //}}}

                }

                }
                //}}}
                }

                // all analysis of output is here
                //{{{
                // close the daily flow output files for the continuous inverse
                // link; these have to be closed here because they are used as
                // input in the method calcContRainfallRunoffMonthly()
                //{{{
                flowDailyStMarys.closeFile();
                flowDailyIngersoll.closeFile();
                flowDailyByron.closeFile();
                //}}}
```

```
// to compute stuff needed for the continuous model inverse link
// together with some drought initial conditions
//{{{

// computes weighted average daily ppt for three locations
// output file names are: {StMarysAvgDailyPPT,
// IngersollAvgDailyPPT, ByronAvgDailyPPT}.txt
contUtils.computeWeightedDailyPPT(outputDir);

// St. Marys
String pptFileStMarys = "StMarysAvgDailyPPT.txt";
String flowFileStMarys = "StMarysDailyFlow.txt";
String outputFileStMarysDrought = "StMarysDroughtInitCond.txt";
String outputFileStMarysPPT = "StMarysSummerPPT.txt";
String outputFileStMarysFlow = "StMarysSummerFlow.txt";
String outputFileStMarysPPTFlow = "StMarysSummerPPTFlow.txt";
contUtils.calcContRainfallRunoffMonthly(pptFileStMarys, flowFileStMarys,
21, outputDir, outputFileStMarysPPT,
outputFileStMarysFlow, outputFileStMarysPPTFlow);
contUtils.fitStatisticalDistribution(outputDir,
"StMarysSummerPPT.txt", "Weibull");
contUtils.fitStatisticalDistribution(outputDir,
"StMarysSummerFlow.txt", "Weibull");
contUtils.calcHistDroughtInfo(outputDir + pptFileStMarys,
outputDir + flowFileStMarys,
outputDir + outputFileStMarysDrought);

// Ingersoll
String pptFileIngersoll = "IngersollAvgDailyPPT.txt";
String flowFileIngersoll = "IngersollDailyFlow.txt";
String outputFileIngersollDrought = "IngersollDroughtInitCond.txt";
String outputFileIngersollPPT = "IngersollSummerPPT.txt";
String outputFileIngersollFlow = "IngersollSummerFlow.txt";
String outputFileIngersollPPTFlow = "IngersollSummerPPTFlow.txt";
contUtils.calcContRainfallRunoffMonthly(pptFileIngersoll, flowFileIngersoll,
21, outputDir, outputFileIngersollPPT,
outputFileIngersollFlow, outputFileIngersollPPTFlow);
contUtils.fitStatisticalDistribution(outputDir,
"IngersollSummerPPT.txt", "Weibull");
contUtils.fitStatisticalDistribution(outputDir,
"IngersollSummerFlow.txt", "Weibull");
contUtils.calcHistDroughtInfo(outputDir + pptFileIngersoll,
outputDir + flowFileIngersoll,
outputDir + outputFileIngersollDrought);

// Byron
String pptFileByron = "ByronAvgDailyPPT.txt";
String flowFileByron = "ByronDailyFlow.txt";
String outputFileByronDrought = "ByronDroughtInitCond.txt";
String outputFileByronPPT = "ByronSummerPPT.txt";
String outputFileByronFlow = "ByronSummerFlow.txt";
String outputFileByronPPTFlow = "ByronSummerPPTFlow.txt";
contUtils.calcContRainfallRunoffMonthly(pptFileByron, flowFileByron,
21, outputDir, outputFileByronPPT,
outputFileByronFlow, outputFileByronPPTFlow);
contUtils.fitStatisticalDistribution(outputDir,
"ByronSummerPPT.txt", "Weibull");
contUtils.fitStatisticalDistribution(outputDir,
"ByronSummerFlow.txt", "Weibull");
contUtils.calcHistDroughtInfo(outputDir + pptFileByron,
outputDir + flowFileByron,
outputDir + outputFileByronDrought);

//}}}
```

```
// close all other output files; these have to be closed as they
// are used as input to the method fitStatisticalDistribution()
//{{{
// to close AnnualSeries2 objects

// daily maximum flows
saDailyByron.closeFile();
saDailyStMarys.closeFile();
saDailyIngersoll.closeFile();

// 7 day minimum flows
sa7DayByron.closeFile();
sa7DayStMarys.closeFile();
sa7DayIngersoll.closeFile();

// monthly minimum flows
saMonthlyByron.closeFile();
saMonthlyStMarys.closeFile();
saMonthlyIngersoll.closeFile();

// to close AnnualSeries objects
saDaily.closeFile();
sa7Day.closeFile();
saMonthly.closeFile();

// to close all other files
jnOutMonthly.closeFile();
sysOut.closeFile();
jnOutDaily.closeFile();
//}}}

// to fit the statistical distributions to the annual extremes
// this is strictly the output of the continuous model, and has
// nothing to do with the inverse link
//{{{

// annual maximum flows are fit to a LP3 and Gumbel distributions
contUtils.fitStatisticalDistribution(outputDir,
"ByronAnnualMaxDailyFlow.txt", "Gumbel");
contUtils.fitStatisticalDistribution(outputDir,
"ByronAnnualMaxDailyFlow.txt", "LP3");
contUtils.fitStatisticalDistribution(outputDir,
"StMarysAnnualMaxDailyFlow.txt", "Gumbel");
contUtils.fitStatisticalDistribution(outputDir,
"StMarysAnnualMaxDailyFlow.txt", "LP3");
contUtils.fitStatisticalDistribution(outputDir,
"IngersollAnnualMaxDailyFlow.txt", "Gumbel");
contUtils.fitStatisticalDistribution(outputDir,
"IngersollAnnualMaxDailyFlow.txt", "LP3");

// 7 day minimum flows are fit to a Weibull
contUtils.fitStatisticalDistribution(outputDir,
"ByronAnnualMin7DayFlow.txt", "Weibull");
contUtils.fitStatisticalDistribution(outputDir,
"StMarysAnnualMin7DayFlow.txt", "Weibull");
contUtils.fitStatisticalDistribution(outputDir,
"IngersollAnnualMin7DayFlow.txt", "Weibull");

// monthly minimum flows are also fit to a Weibull
contUtils.fitStatisticalDistribution(outputDir,
"ByronAnnualMinMonthlyFlow.txt", "Weibull");
contUtils.fitStatisticalDistribution(outputDir,
"StMarysAnnualMinMonthlyFlow.txt", "Weibull");
```

```
contUtils.fitStatisticalDistribution(outputDir,
"IngersollAnnualMinMonthlyFlow.txt", "Weibull");
//}}}

//}}}

// Get elapsed time in milliseconds
long elapsedTimeMillis = System.currentTimeMillis() - start;

// Get elapsed time in seconds
float elapsedTimeSec = elapsedTimeMillis / (1000F);
System.out.println("Elapsed time (sec): " + elapsedTimeSec);

}
}
```

D.2 HydModel.java

```java
import java.io.*;

/**
 *  This is the hydrological model object it is here that we assemble the
 * model with SubBasin objects (SoilMoistureAccounting, LinearReservoir and
 * Clark objects) together with ModifiedPuls objects; Junctions are not
 * explicitely modelled they are simply variables in HydModel class
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class HydModel {

// hydrological model constants and variable declaration
//{{{

// this is what the HydModel gets from the SysModel; stands for
// fraction of paved land in each county; Clark's time of concentration
// will be determined from this value, as will the percent of imperviousness
private double FPLMiddlesex = 0.0;
private double FPLOxford = 0.0;
private double FPLPerth = 0.0;

// same as above, except that the variables stand for fraction of
// vegetative land
private double FVLMiddlesex = 0.0;
private double FVLOxford = 0.0;
private double FVLPerth = 0.0;

// the instance variables that are the tables to update the time of
// concentration parameters of the SubBasin objects
private Table TCTableMiddlesex24hr, TCTableOxford24hr, TCTablePerth24hr;
private Table TCTableMiddlesex18hr, TCTableOxford18hr, TCTablePerth18hr;
private Table TCTableMiddlesex12hr, TCTableOxford12hr, TCTablePerth12hr;

// the instance variables that are the tables to update the percent
// of impervious parameters of the SubBasin objects
private Table ImpTableMiddlesex0, ImpTableMiddlesex5;
private Table ImpTableMiddlesex40, ImpTableMiddlesex30;
private Table ImpTablePerth2, ImpTablePerth0;
private Table ImpTableOxford0;

// the instance variables that are the tables to update the
// maximum surface storage reduction multiplier
private Table maxSurfStoreRedMultMiddlesex, maxSurfStoreRedMultOxford,
maxSurfStoreRedMultPerth;

// the instance variables that are the tables to update the amount of
// PET of the SubBasin objects
private Table PETTableMiddlesex, PETTableOxford, PETTablePerth;

// the instance variables that are the tables to update the infiltration
// capacity of the SubBasin objects
private Table infilTableMiddlesex, infilTableOxford, infilTablePerth;

// input and output directories of the temp and ppt data that is obtained
// from the main
private String inputDir, outputDir;

// time step of the PPT data, in [hrs]
private final int dataTimeStep = 24;
```

```
// instance variables that the constructor initializes
private ModelDate currentDate;

// this is the first time the HydModel receives information from the
// SysModel
private int firstYear, firstMonth, firstDay, firstHour;

private int userTimeStep;

// flow damage table for Byron, Ingersoll and StMarys
Table FlowDamageByronSGTable;
Table FlowDamageIngersollSGTable;
Table FlowDamageStMarysSGTable;

// DataReader objects that read in SubBasin PPT previously genereated
// by contUtils.interpolate() method
private DataReader sb1PPT;
private DataReader sb3PPT;
private DataReader sb4PPT;
private DataReader sb5PPT;
private DataReader sb7PPT;
private DataReader sb8PPT;
private DataReader sb9PPT;
private DataReader sb10PPT;
private DataReader sb11PPT;
private DataReader sb12PPT;
private DataReader sb13PPT;
private DataReader sb14PPT;
private DataReader sb15PPT;
private DataReader sb16PPT;
private DataReader sb17PPT;
private DataReader sb18PPT;
private DataReader sb19PPT;
private DataReader sb20PPT;
private DataReader sb21PPT;
private DataReader sb22PPT;
private DataReader sb23PPT;
private DataReader sb24PPT;
private DataReader sb25PPT;
private DataReader sb26PPT;
private DataReader sb27PPT;
private DataReader sb28PPT;
private DataReader sb29PPT;
private DataReader sb30PPT;
private DataReader sb31PPT;
private DataReader sb32PPT;
private DataReader sb33PPT;
private DataReader sb34PPT;

// numberical values of the PPT that are read from the object above
// in units of [mm/day]
private double sb1PPTVal;
private double sb3PPTVal;
private double sb4PPTVal;
private double sb5PPTVal;
private double sb7PPTVal;
private double sb8PPTVal;
private double sb9PPTVal;
private double sb10PPTVal;
private double sb11PPTVal;
private double sb12PPTVal;
private double sb13PPTVal;
private double sb14PPTVal;
private double sb15PPTVal;
```

```
private double sb16PPTVal;
private double sb17PPTVal;
private double sb18PPTVal;
private double sb19PPTVal;
private double sb20PPTVal;
private double sb21PPTVal;
private double sb22PPTVal;
private double sb23PPTVal;
private double sb24PPTVal;
private double sb25PPTVal;
private double sb26PPTVal;
private double sb27PPTVal;
private double sb28PPTVal;
private double sb29PPTVal;
private double sb30PPTVal;
private double sb31PPTVal;
private double sb32PPTVal;
private double sb33PPTVal;
private double sb34PPTVal;

// SubBasin objects
private SubBasin sb1;
private SubBasin sb3;
private SubBasin sb4;
private SubBasin sb5;
private SubBasin sb7;
private SubBasin sb8;
private SubBasin sb9;
private SubBasin sb10;
private SubBasin sb11;
private SubBasin sb12;
private SubBasin sb13;
private SubBasin sb14;
private SubBasin sb15;
private SubBasin sb16;
private SubBasin sb17;
private SubBasin sb18;
private SubBasin sb19;
private SubBasin sb20;
private SubBasin sb21;
private SubBasin sb22;
private SubBasin sb23;
private SubBasin sb24;
private SubBasin sb25;
private SubBasin sb26;
private SubBasin sb27;
private SubBasin sb28;
private SubBasin sb29;
private SubBasin sb30;
private SubBasin sb31;
private SubBasin sb32;
private SubBasin sb33;
private SubBasin sb34;

// PET variables
private double[] evapNOR;
private double[] evapMID;
private double[] evapSUD;
private double panCoeff;

// junction variables
private double[] jnMitchellSG;
private double[] jn640;
private double[] jn750;
```

```
private double[] jn830;
private double[] jnUpStreamWildwood;
private double[] jnDownStreamWildwood;
private double[] jnStMarysSG;
private double[] jn2290;
private double[] jnPloverMillsSG;
private double[] jnUpStreamFanshawe;
private double[] jnDownStreamFanshawe;
private double[] jn1930;
private double[] jnUpStreamPittock;
private double[] jnDownStreamPittock;
private double[] jn1840;
private double[] jnBeachville;
private double[] jnIngersollSG;
private double[] jnThamesfordSG;
private double[] jn1960;
private double[] jn2050;
private double[] jnEalingSG;
private double[] jnForks;
private double[] jnByronSG;
private double[] jn2120;
private double[] jn2270;

// ModifiedPuls objects
private ModifiedPuls R560;
private ModifiedPuls R640;
private ModifiedPuls R750;
private ModifiedPuls R900;
private ModifiedPuls wildwood;
private ModifiedPuls R930;
private ModifiedPuls R1010;
private ModifiedPuls R2290;
private ModifiedPuls R2300;
private ModifiedPuls fanshawe;
private ModifiedPuls R1910;
private ModifiedPuls R1930;
private ModifiedPuls pittock;
private ModifiedPuls R111;
private ModifiedPuls R222;
private ModifiedPuls R333;
private ModifiedPuls R1870;
private ModifiedPuls R1890;
private ModifiedPuls R2030;
private ModifiedPuls R2050;
private ModifiedPuls R2430;
private ModifiedPuls R2440;
private ModifiedPuls R2040;
private ModifiedPuls R2120;


//}}}

/**
 *  Constructor for the HydModel object
 *
 * @param  currentDate   Current date
 * @param  userTimeStep  User time step, in [hrs]
 */
public HydModel(ModelDate currentDate, int userTimeStep) {
this.currentDate = currentDate;
this.userTimeStep = userTimeStep;

// this is the first time HydModel receives information
// from the SysModel; essentially, this is the end of the first
```

```
// month of the simulation
this.firstYear = this.currentDate.getYear();
this.firstMonth = this.currentDate.getMonth();
this.firstDay = this.currentDate.getDaysInMonth();
this.firstHour = 19;
}


/**
 *  Initializes everything in the HydModel object
 *
 * @param  inputDir        Input directory of the data set
 * @param  outputDir       Output (or temporary) directory where some
 *     intermediate data gets saved
 * @exception  IOException  Input Output Exception
 */
public void initialize(String inputDir, String outputDir)
 throws IOException {
//{{{

// flow damage table relationships; in units of Q (cms) versus
// Damage ($1,000) 2005 Canadian Dollars; these are the curves that
// were prepared by Helsten and Davidge (2005); the flow values
// seems to be way out of range
double[] QByronSG = {0.0, 593, 843, 1070, 1170, 1370, 1498, 1834, 2094};
double[] DByronSG = {0.0, 0, 125.3, 2066.02, 2986.27, 4951.29, 7667.86,
12219.75, 32314.85};
this.FlowDamageByronSGTable = new Table(QByronSG, DByronSG);

double[] QIngersollSG = {0.0, 108, 154, 179, 204, 223, 239, 276.15, 304.25};
double[] DIngersollSG = {0.0, 163.83, 1084.89, 2244.61, 3532.3, 3999.15,
4516.36, 15432.18, 22246.28};
this.FlowDamageIngersollSGTable = new Table(QIngersollSG, DIngersollSG);

double[] QStMarysSG = {0.0, 377, 518, 613, 731.42, 821, 910, 1029.56, 1120};
double[] DStMarysSG = {0.0, 450.01, 574.29, 718.49, 1068.67, 1816.17,
5211.38, 9935.44, 13627.41};
this.FlowDamageStMarysSGTable = new Table(QStMarysSG, DStMarysSG);

// directory where original raw data is
// this data is first temporally interpolated, the adjusted
// for snow accumulation and melt, then spacially interpolated
// to obtain ppt for each sub catchment.

this.inputDir = inputDir;
this.outputDir = outputDir;

// the calls to contUtils are best done with an outside class

// formats the WG output from a 301 year data set so that it can
// be used in this work
//contUtils.formatAllWGInput(this.inputDir);

// applies the snow algorithm to our 15 ppt gauges
// gauge data and all snow melt parameters are embedded into the
// method because of the large number of parameters
// if parameters and/or data sources need to be changed, go the
// method and change it there

// This is used in case we need to interpolate the data
// temporally, in order to fill in the blanks
//contUtils.interpolateTemporally(this.inputDir, this.outputDir);

// only need to do this once per WG scenario
```

```
// the method reads files from the same directory where it
// outputs them
//contUtils.adjustForSnow(this.inputDir, this.outputDir);

// interpolates the adjusted ppt to the centroids of each
// subcatchment; again, this needs to be done once per WG scenario
// as before, all parameters are within the the method
//contUtils.interpolateSpacially(this.outputDir, this.outputDir);

// to create DataReader objects that read in the interpolated
// SubBasin PPT that contUtils.interpolateSpacially() just generated
this.sb1PPT = new DataReader(
this.outputDir + "sb1PPT.csv");
this.sb1PPTVal = sb1PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb3PPT = new DataReader(
this.outputDir + "sb3PPT.csv");
this.sb3PPTVal = sb3PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb4PPT = new DataReader(
this.outputDir + "sb4PPT.csv");
this.sb4PPTVal = sb4PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb5PPT = new DataReader(
this.outputDir + "sb5PPT.csv");
this.sb5PPTVal = sb5PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb7PPT = new DataReader(
this.outputDir + "sb7PPT.csv");
this.sb7PPTVal = sb7PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb8PPT = new DataReader(
this.outputDir + "sb8PPT.csv");
this.sb8PPTVal = sb8PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb9PPT = new DataReader(
this.outputDir + "sb9PPT.csv");
this.sb9PPTVal = sb9PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb10PPT = new DataReader(
this.outputDir + "sb10PPT.csv");
this.sb10PPTVal = sb10PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb11PPT = new DataReader(
this.outputDir + "sb11PPT.csv");
this.sb11PPTVal = sb11PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb12PPT = new DataReader(
this.outputDir + "sb12PPT.csv");
this.sb12PPTVal = sb12PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb13PPT = new DataReader(
this.outputDir + "sb13PPT.csv");
this.sb13PPTVal = sb13PPT.readInitialData() /
```

```
(dataTimeStep / this.userTimeStep);

this.sb14PPT = new DataReader(
this.outputDir + "sb14PPT.csv");
this.sb14PPTVal = sb14PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb15PPT = new DataReader(
this.outputDir + "sb15PPT.csv");
this.sb15PPTVal = sb15PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb16PPT = new DataReader(
this.outputDir + "sb16PPT.csv");
this.sb16PPTVal = sb16PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb17PPT = new DataReader(
this.outputDir + "sb17PPT.csv");
this.sb17PPTVal = sb17PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb18PPT = new DataReader(
this.outputDir + "sb18PPT.csv");
this.sb18PPTVal = sb18PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb19PPT = new DataReader(
this.outputDir + "sb19PPT.csv");
this.sb19PPTVal = sb19PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb20PPT = new DataReader(
this.outputDir + "sb20PPT.csv");
this.sb20PPTVal = sb20PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb21PPT = new DataReader(
this.outputDir + "sb21PPT.csv");
this.sb21PPTVal = sb21PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb22PPT = new DataReader(
this.outputDir + "sb22PPT.csv");
this.sb22PPTVal = sb22PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb23PPT = new DataReader(
this.outputDir + "sb23PPT.csv");
this.sb23PPTVal = sb23PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb24PPT = new DataReader(
this.outputDir + "sb24PPT.csv");
this.sb24PPTVal = sb24PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb25PPT = new DataReader(
this.outputDir + "sb25PPT.csv");
this.sb25PPTVal = sb25PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb26PPT = new DataReader(
this.outputDir + "sb26PPT.csv");
```

```
this.sb26PPTVal = sb26PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb27PPT = new DataReader(
this.outputDir + "sb27PPT.csv");
this.sb27PPTVal = sb27PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb28PPT = new DataReader(
this.outputDir + "sb28PPT.csv");
this.sb28PPTVal = sb28PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb29PPT = new DataReader(
this.outputDir + "sb29PPT.csv");
this.sb29PPTVal = sb29PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb30PPT = new DataReader(
this.outputDir + "sb30PPT.csv");
this.sb30PPTVal = sb30PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb31PPT = new DataReader(
this.outputDir + "sb31PPT.csv");
this.sb31PPTVal = sb31PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb32PPT = new DataReader(
this.outputDir + "sb32PPT.csv");
this.sb32PPTVal = sb32PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb33PPT = new DataReader(
this.outputDir + "sb33PPT.csv");
this.sb33PPTVal = sb33PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

this.sb34PPT = new DataReader(
this.outputDir + "sb34PPT.csv");
this.sb34PPTVal = sb34PPT.readInitialData() /
(dataTimeStep / this.userTimeStep);

// these are monthly evapotranspiration rates for the three
// evaporation zones within the Upper Thames Basin, in [mm/month]

// only sb1 is in NOR
double[] tempNOR = {0.1, 0.0, 6.5, 59.8, 96.8, 116.3, 127.6,
99.5, 64.8, 29.7, 8.6, 0.0};

// sb2-sb14, sb18, sb23 are in MID
double[] tempMID = {0.1, 0.0, 11.0, 62.7, 99.1, 119.6, 128.6,
100.9, 65.6, 31.1, 8.9, 0.0};

// sb15-sb17, sb19-sb22, sb24-sb34 are in SUD
double[] tempSUD = {0.1, 0.0, 19.5, 66.4, 101.8, 124.6, 134.6,
105.6, 68.4, 31.8, 8.8, 0.1};

this.evapNOR = tempNOR;
this.evapMID = tempMID;
this.evapSUD = tempSUD;

// this is the pan coefficient for all sub catchments in the basin
this.panCoeff = 0.7;
```

```
// routing reach data; storages in [thousands m^3] and
// outflows in [m^3/s]
double[] sR560 = {0, 389.67, 657.93, 1421.86, 2607.09, 3508.72,
3860.86, 5109.41, 5650, 5984.53, 7505.39};
double[] oR560 = {0, 20.7, 41.4, 104, 207, 292, 355, 445,
508, 550, 768};
this.R560 = new ModifiedPuls(sR560, oR560, this.userTimeStep);

double[] sR640 = {0.0, 271.15, 440.87, 924.18, 1663.83, 2256.23,
2499.1, 3106.65, 3440.88, 3621.61, 4737.15};
double[] oR640 = {0.0, 25.3, 50.5, 126, 253, 356, 426, 534,
610, 660, 937};
this.R640 = new ModifiedPuls(sR640, oR640, this.userTimeStep);

double[] sR750 = {0.0, 104.9, 177.99, 377.73, 668.41, 876.97,
1028.77, 1280.78, 1332.32, 1391.07, 1649.61};
double[] oR750 = {0.0, 27.5, 55, 138, 270, 380, 461, 578,
660, 715, 1021};
this.R750 = new ModifiedPuls(sR750, oR750, this.userTimeStep);

double[] sR900 = {0.0, 182.9, 297.24, 603.72, 1058.99, 1369.71,
1578.41, 1866.99, 2093.57, 2184.92, 2782.47};
double[] oR900 = {0.0, 31.9, 64, 159, 345, 487, 566, 657, 716, 785, 1183};
this.R900 = new ModifiedPuls(sR900, oR900, this.userTimeStep);

double[] sR930 = {0.0, 67, 95, 115, 138.17, 208.56, 319.53, 543.7,
1132.43, 3021.35, 5241.65};
double[] oR930 = {0.0, 3, 5, 7, 10, 20, 30, 50, 100, 151, 180};
this.R930 = new ModifiedPuls(sR930, oR930, this.userTimeStep);

double[] sWildwood = {2430, 2430.1, 3050, 3730, 4470, 5310, 6280, 7350,
8520, 9780, 11120, 12580, 14180, 15880, 17730, 18100, 18470,
18840, 19250, 19660, 20470, 21290, 22110, 22930, 23800, 24670};
double[] oWildwood = {0.0, 0.79, 0.82, 0.86, 0.89, 0.92, 0.95, 0.98,
1.01, 1.03, 1.06, 1.08, 1.1, 3, 3, 3, 4.33, 5.66, 7.37, 18.6,
23.55, 29.35, 35.87, 43.02, 60.66, 68.92};
this.wildwood = new ModifiedPuls(sWildwood, oWildwood, this.userTimeStep);

double[] sR1010 = {0.0, 309.23, 503.96, 985.75, 1726, 2222.59,
2558.16, 2945.41, 3234.45, 3522.67, 4514.07};
double[] oR1010 = {0.0, 33.5, 67, 168, 335, 450, 530, 630, 705, 784, 1057};
this.R1010 = new ModifiedPuls(sR1010, oR1010, this.userTimeStep);

double[] sR2290 = {0.0, 230.88, 371.24, 716.78, 1233.15, 1606.14,
1852.44, 2149.65, 2376.79, 2599.42, 3319.45};
double[] oR2290 = {0.0, 35.5, 71, 178, 355, 477, 562, 668, 748, 833, 1121};
this.R2290 = new ModifiedPuls(sR2290, oR2290, this.userTimeStep);

double[] sR2300 = {0.0, 522.14, 788.88, 1430.77, 2343.38, 2964.62,
3366.77, 9099.35, 9331.21, 9567.45, 10373};
double[] oR2300 = {0.0, 37, 74, 184, 369, 496, 584, 690, 776, 864, 1164};
this.R2300 = new ModifiedPuls(sR2300, oR2300, this.userTimeStep);

double[] sFanshawe = {12350, 12350.1, 12900, 12900, 13450, 13450, 14000,
14550, 15150, 15700, 16250, 16850, 17400, 18000, 19300, 20600,
21950, 23250, 23250, 24650, 26600, 28550, 28550, 30600, 32700,
34950, 37200, 37200, 42050, 47250, 47250, 52300};
double[] oFanshawe = {0.0, 1, 3, 5.76, 5.76, 29.94, 50.55, 75.3, 103.76,
135.65, 141.8, 155.4, 167.6, 178.8, 199, 217, 234, 248.7, 321, 341,
365, 388, 475, 502, 530, 558, 586, 694, 763, 836, 1335, 1453};
this.fanshawe = new ModifiedPuls(sFanshawe, oFanshawe, this.userTimeStep);

double[] sR1910 = {0.0, 837.87, 1279.23, 2592.06, 4971.75, 6159.72,
```

```
7120.41, 7712.24, 8615.57, 9436.37, 14583.16};
double[] oR1910 = {0.0, 36.1, 72.2, 181, 361, 447, 535, 579, 624, 744.7, 1121.5};
this.R1910 = new ModifiedPuls(sR1910, oR1910, this.userTimeStep);

double[] sR1930 = {0.0, 225.94, 363.43, 681.95, 1264.37, 1728.5,
1967.24, 2205.7, 2461.66, 2611.69, 4386.31};
double[] oR1930 = {0.0, 44, 88, 220, 445, 591, 669, 731, 801, 815, 1367.7};
this.R1930 = new ModifiedPuls(sR1930, oR1930, this.userTimeStep);

double[] sPittock = {100, 100.1, 100, 260, 470, 680, 890, 890, 1500, 2240,
3070, 4040, 5110, 6340, 7700, 7700, 9250, 9250, 10950, 12880,
14930, 14930, 17160, 17160, 18940};
double[] oPittock = {0.0, 0.4, 2.7, 2.9, 3, 3.2, 3.33, 5.9, 7.3, 8.5, 9.5,
10.4, 11.3, 12.1, 14.3, 27.1, 35, 48.1, 59, 72, 86, 101.12, 117,
180, 196};
this.pittock = new ModifiedPuls(sPittock, oPittock, this.userTimeStep);

double[] sR111 = {0.0, 50.5, 86.16, 196.43, 329.36, 580.92, 673.32,
740.4, 794.03, 1058.11, 1655.93};
double[] oR111 = {0.0, 4.4, 8.8, 21.6, 34, 74, 83.5, 90, 93.5, 130, 211};
this.R111 = new ModifiedPuls(sR111, oR111, this.userTimeStep);

double[] sR222 = {0.0, 95.13, 156.33, 378.48, 690.3, 1162.93, 1378.07,
1642.92, 1800.79, 2213.88, 3732.2};
double[] oR222 = {0.0, 7, 14, 35, 70, 107, 125, 140, 156, 200, 307.2};
this.R222 = new ModifiedPuls(sR222, oR222, this.userTimeStep);

double[] sR333 = {0.0, 107.93, 169.42, 311.18, 513.11, 655.22, 733.83,
809.86, 876.07, 906.99, 1542.14};
double[] oR333 = {0.0, 10.8, 21.6, 54, 108, 154, 179, 204, 223, 239, 414};
this.R333 = new ModifiedPuls(sR333, oR333, this.userTimeStep);

double[] sR1870 = {0.0, 264.88, 474.8, 1321.19, 1810.16, 3810.08,
4278.23, 4675.72, 5832.38, 6291.38, 8737.16};
double[] oR1870 = {0.0, 13.2, 26.4, 67, 120, 214, 249, 280, 320, 360, 572};
this.R1870 = new ModifiedPuls(sR1870, oR1870, this.userTimeStep);

double[] sR1890 = {0.0, 404.07, 543.7, 958.47, 1558.6, 2047.74,
2327.52, 2642.25, 2881.41, 3097.13, 3846.54};
double[] oR1890 = {0.0, 9.1, 18.1, 45.3, 90, 133, 159, 190, 214, 236, 314};
this.R1890 = new ModifiedPuls(sR1890, oR1890, this.userTimeStep);

double[] sR2030 = {0.0, 585.96, 913.82, 1829.99, 3183.52, 5691.37,
6608.67, 7421.55, 8496.58, 9469.84, 14140.82};
double[] oR2030 = {0.0, 21, 41, 104, 188, 333, 387, 435, 500, 560, 870};
this.R2030 = new ModifiedPuls(sR2030, oR2030, this.userTimeStep);

double[] sR2050 = {0.0, 265.55, 421.7, 908.26, 1637.43, 2468.77,
2946.06, 3412.74, 3915.06, 4339.66, 6021.55};
double[] oR2050 = {0.0, 24, 48, 122, 224, 391, 455, 516, 580, 661, 1019};
this.R2050 = new ModifiedPuls(sR2050, oR2050, this.userTimeStep);

double[] sR2430 = {0.0, 175, 344.31, 582.38, 976.73, 1361.44, 1615.79,
1885.8, 2285.38, 2656.99, 4934.06};
double[] oR2430 = {0.0, 24, 48, 122, 224, 391, 455, 516, 580, 661, 1019};
this.R2430 = new ModifiedPuls(sR2430, oR2430, this.userTimeStep);

double[] sR2440 = {0.0, 691.75, 1024.01, 1820.28, 3114.94, 4105.96,
4766.86, 5373.9, 6136.58, 6776.65, 9264.95};
double[] oR2440 = {0.0, 59.3, 118.6, 296.5, 593, 843, 1010, 1170, 1370,
1489, 2200};
this.R2440 = new ModifiedPuls(sR2440, oR2440, this.userTimeStep);

double[] sR2040 = {0.0, 440.04, 658.15, 1007.31, 3191.6, 4271.63,
```

```
4961.46, 5716.29, 6300.11, 6857.22, 8951.47};
double[] oR2040 = {0.0, 33, 65, 130, 651, 926, 1110, 1320, 1490, 1658, 2340};
this.R2040 = new ModifiedPuls(sR2040, oR2040, this.userTimeStep);

double[] sR2120 = {0.0, 736.63, 1110.68, 1721.57, 6531.27, 10598.2,
14004.89, 17214.6, 19991.89, 22381.16, 33771.6};
double[] oR2120 = {0.0, 33, 65, 130, 651, 926, 1110, 1320, 1490, 1658, 2340};
this.R2120 = new ModifiedPuls(sR2120, oR2120, this.userTimeStep);

// instantiate all SubBasin objects here; the ETZone objects have
// reduction coefficients set to unity at this point
this.sb1 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb3 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb4 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb5 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb7 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb8 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb9 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb10 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb11 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb12 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb13 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb14 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb15 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb16 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb17 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb18 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb19 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb20 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb21 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb22 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb23 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb24 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb25 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb26 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb27 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb28 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb29 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb30 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb31 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb32 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb33 = new SubBasin(this.currentDate, this.userTimeStep);
this.sb34 = new SubBasin(this.currentDate, this.userTimeStep);

// this is where all the parameters are set initially
initializeParams();

// this is where all SubBasin objects are initialized
this.sb1.initialize();
this.sb3.initialize();
this.sb4.initialize();
this.sb5.initialize();
this.sb7.initialize();
this.sb8.initialize();
this.sb9.initialize();
this.sb10.initialize();
this.sb11.initialize();
this.sb12.initialize();
this.sb13.initialize();
this.sb14.initialize();
this.sb15.initialize();
this.sb16.initialize();
this.sb17.initialize();
this.sb18.initialize();
```

```
this.sb19.initialize();
this.sb20.initialize();
this.sb21.initialize();
this.sb22.initialize();
this.sb23.initialize();
this.sb24.initialize();
this.sb25.initialize();
this.sb26.initialize();
this.sb27.initialize();
this.sb28.initialize();
this.sb29.initialize();
this.sb30.initialize();
this.sb31.initialize();
this.sb32.initialize();
this.sb33.initialize();
this.sb34.initialize();

// this is where the parameters are adjusted due to their
// seasonal nature
if (currentDate.getMonth() > 3 &&
currentDate.getMonth() < 10) {
setSummerParams();
updateParams();
} else {
setWinterParams();
updateParams();
}

// MODEL ASSEMBLY STARTS HERE

// NORTH BRANCH STARTS HERE
// sb1 drains into the jnMitchellSG stream gauge
this.jnMitchellSG = new double[2];
this.jnMitchellSG[0] = 0.0;
this.jnMitchellSG[1] = this.sb1.getInitTotalFlow();

// jnMitchellSG gets routed by R560
this.R560.initialize(this.jnMitchellSG[1]);

// jn640 is the sum of sb3, sb4, and R560
this.jn640 = new double[2];
this.jn640[0] = 0.0;
this.jn640[1] = this.R560.getInitOutflow() +
this.sb3.getInitTotalFlow() +
this.sb4.getInitTotalFlow();

// jn640 gets routed by R640
this.R640.initialize(this.jn640[1]);

// jn750 is the sum sb5, sb7, and R640
this.jn750 = new double[2];
this.jn750[0] = 0.0;
this.jn750[1] = this.R640.getInitOutflow() +
this.sb5.getInitTotalFlow() +
this.sb7.getInitTotalFlow();

// jn750 gets routed by R750
this.R750.initialize(this.jn750[1]);

// jn830 is the sum of sb8, sb9 and R750
this.jn830 = new double[2];
this.jn830[0] = 0.0;
this.jn830[1] = this.R750.getInitOutflow() +
this.sb8.getInitTotalFlow() +
```

```
this.sb9.getInitTotalFlow();

// jn830 gets routed by R900
this.R900.initialize(this.jn830[1]);

// jnUpStramWildwood is the sum of sb10
this.jnUpStreamWildwood = new double[2];
this.jnUpStreamWildwood[0] = 0.0;
this.jnUpStreamWildwood[1] = this.sb10.getInitTotalFlow();

// jnUpStramWildwood gets routed by wildwood
this.wildwood.initialize(this.jnUpStreamWildwood[1]);

// jnDownStreamWildwood is the outflow of wildwood
this.jnDownStreamWildwood = new double[2];
this.jnDownStreamWildwood[0] = 0.0;
this.jnDownStreamWildwood[1] = this.wildwood.getInitOutflow();

// jnDownStramWildwood gets routed by R930
this.R930.initialize(this.jnDownStreamWildwood[1]);

// jnStMarysSG is the sum of sb11, R900 and R930
this.jnStMarysSG = new double[2];
this.jnStMarysSG[0] = 0.0;
this.jnStMarysSG[1] = this.R900.getInitOutflow() +
this.R930.getInitOutflow() +
this.sb11.getInitTotalFlow();

// jnStMarysSG gets routed by R1010
this.R1010.initialize(this.jnStMarysSG[1]);

// jn2290 is the sum of sb12, sb13 and R1010
this.jn2290 = new double[2];
this.jn2290[0] = 0.0;
this.jn2290[1] = this.R1010.getInitOutflow() +
this.sb12.getInitTotalFlow() +
this.sb13.getInitTotalFlow();

// jn2290 gets routed by R2290
this.R2290.initialize(this.jn2290[1]);

// jnPloverMillsSG is the sum of R2290 and sb14
this.jnPloverMillsSG = new double[2];
this.jnPloverMillsSG[0] = 0.0;
this.jnPloverMillsSG[1] = this.R2290.getInitOutflow() +
this.sb14.getInitTotalFlow();

// jnPloverMillsSG gets routed by R2300
this.R2300.initialize(this.jnPloverMillsSG[1]);

// jnUpStreamFanshawe is the sum of R2300 and sb15
this.jnUpStreamFanshawe = new double[2];
this.jnUpStreamFanshawe[0] = 0.0;
this.jnUpStreamFanshawe[1] = this.R2300.getInitOutflow() +
this.sb15.getInitTotalFlow();

// jnUpStreamFanshawe is routed by fanshawe
this.fanshawe.initialize(this.jnUpStreamFanshawe[1]);

// jnDownStreamFanshawe is the outflow from fanshawe
this.jnDownStreamFanshawe = new double[2];
this.jnDownStreamFanshawe[0] = 0.0;
this.jnDownStreamFanshawe[1] = this.fanshawe.getInitOutflow();
```

```
// jnDownStreamFanshawe is routed with R1910
this.R1910.initialize(this.jnDownStreamFanshawe[1]);

// jn1930 is the sum of R1910, sb16, sb17
this.jn1930 = new double[2];
this.jn1930[0] = 0.0;
this.jn1930[1] = this.R1910.getInitOutflow() +
this.sb16.getInitTotalFlow() +
this.sb17.getInitTotalFlow();

// jn1930 is routed with R1930
this.R1930.initialize(this.jn1930[1]);

// SOUTH BRANCH STARTS HERE
// jnUpStreamPittock is sb18
this.jnUpStreamPittock = new double[2];
this.jnUpStreamPittock[0] = 0.0;
this.jnUpStreamPittock[1] = this.sb18.getInitTotalFlow();

// jnUpStreamPittock is routed by pittock
this.pittock.initialize(this.jnUpStreamPittock[1]);

// jnDownStreamPittock is the outflow from pittock
this.jnDownStreamPittock = new double[2];
this.jnDownStreamPittock[0] = 0.0;
this.jnDownStreamPittock[1] = this.pittock.getInitOutflow();

// jnDownStreamPittock is routed with R111
this.R111.initialize(jnDownStreamPittock[1]);

// jn1840 is the sum of R111, sb19, sb20
this.jn1840 = new double[2];
this.jn1840[0] = 0.0;
this.jn1840[1] = this.R111.getInitOutflow() +
this.sb19.getInitTotalFlow() +
this.sb20.getInitTotalFlow();

// jn1840 is routed by R222
this.R222.initialize(this.jn1840[1]);

// jnBeachville is outflow of R222
this.jnBeachville = new double[2];
this.jnBeachville[0] = 0.0;
this.jnBeachville[1] = this.R222.getInitOutflow();

// jnBeachville is routed by R333
this.R333.initialize(this.jnBeachville[1]);

// jnIngersollSG is the sum of R333 and sb21
this.jnIngersollSG = new double[2];
this.jnIngersollSG[0] = 0.0;
this.jnIngersollSG[1] = this.R333.getInitOutflow() +
this.sb21.getInitTotalFlow();

// jnIngersollSG is routed by R1870
this.R1870.initialize(this.jnIngersollSG[1]);

// jnThamesfordSG is the sum of sb23
this.jnThamesfordSG = new double[2];
this.jnThamesfordSG[0] = 0.0;
this.jnThamesfordSG[1] = this.sb23.getInitTotalFlow();

// jnThamesfordSG is routed by R1890
this.R1890.initialize(this.jnThamesfordSG[1]);
```

```
// jn1960 is the sum of R1870, R1890, sb22, sb24 and sb25
this.jn1960 = new double[2];
this.jn1960[0] = 0.0;
this.jn1960[1] = this.R1870.getInitOutflow() +
this.R1890.getInitOutflow() +
this.sb22.getInitTotalFlow() +
this.sb24.getInitTotalFlow() +
this.sb25.getInitTotalFlow();

// jn1960 is routed by R2030
this.R2030.initialize(this.jn1960[1]);

// jn2050 is the sum of R2030, sb26, and sb27
this.jn2050 = new double[2];
this.jn2050[0] = 0.0;
this.jn2050[1] = this.R2030.getInitOutflow() +
this.sb26.getInitTotalFlow() +
this.sb27.getInitTotalFlow();

// jn2050 is routed by R2050
this.R2050.initialize(this.jn2050[1]);

// jnEalingSG is the sum of R2050 and sb28
this.jnEalingSG = new double[2];
this.jnEalingSG[0] = 0.0;
this.jnEalingSG[1] = this.R2050.getInitOutflow() +
this.sb28.getInitTotalFlow();

// jnEalingSG is routed by R2430
this.R2430.initialize(this.jnEalingSG[1]);

// THIS IS WHERE NORTH AND SOUTH BRANCH OF THE THAMES RIVER MEET
// jnForks is the sum of R1930, R2430 and sb29
this.jnForks = new double[2];
this.jnForks[0] = 0.0;
this.jnForks[1] = this.R1930.getInitOutflow() +
this.R2430.getInitOutflow() +
this.sb29.getInitTotalFlow();

// jnForks is routed by R2440
this.R2440.initialize(this.jnForks[1]);

// jnByronSG is the sum of R2440 and sb30
this.jnByronSG = new double[2];
this.jnByronSG[0] = 0.0;
this.jnByronSG[1] = this.R2440.getInitOutflow() +
this.sb30.getInitTotalFlow();

// jnByronSG is routed by R2040
this.R2040.initialize(this.jnByronSG[1]);

// jn2120 is the sum of R2040, sb31 and sb32
this.jn2120 = new double[2];
this.jn2120[0] = 0.0;
this.jn2120[1] = this.R2040.getInitOutflow() +
this.sb31.getInitTotalFlow() +
this.sb32.getInitTotalFlow();

// jn2120 is routed by R2120
this.R2120.initialize(this.jn2120[1]);

// jn2270 is the sum of R2120, sb33 and sb34
this.jn2270 = new double[2];
```

```
this.jn2270[0] = 0.0;
this.jn2270[1] = this.R2120.getInitOutflow() +
this.sb33.getInitTotalFlow() +
this.sb34.getInitTotalFlow();

// reset the junctions
this.jnMitchellSG[0] = this.jnMitchellSG[1];
this.jn640[0] = this.jn640[1];
this.jn750[0] = this.jn750[1];
this.jn830[0] = this.jn830[1];
this.jnUpStreamWildwood[0] = this.jnUpStreamWildwood[1];
this.jnDownStreamWildwood[0] = this.jnDownStreamWildwood[1];
this.jnStMarysSG[0] = this.jnStMarysSG[1];
this.jn2290[0] = this.jn2290[1];
this.jnPloverMillsSG[0] = this.jnPloverMillsSG[1];
this.jnUpStreamFanshawe[0] = this.jnUpStreamFanshawe[1];
this.jnDownStreamFanshawe[0] = this.jnDownStreamFanshawe[1];
this.jn1930[0] = this.jn1930[1];
this.jnUpStreamPittock[0] = this.jnUpStreamPittock[1];
this.jnDownStreamPittock[0] = this.jnDownStreamPittock[1];
this.jn1840[0] = this.jn1840[1];
this.jnBeachville[0] = this.jnBeachville[1];
this.jnIngersollSG[0] = this.jnIngersollSG[1];
this.jnThamesfordSG[0] = this.jnThamesfordSG[1];
this.jn1960[0] = this.jn1960[1];
this.jn2050[0] = this.jn2050[1];
this.jnEalingSG[0] = this.jnEalingSG[1];
this.jnForks[0] = this.jnForks[1];
this.jnByronSG[0] = this.jnByronSG[1];
this.jn2120[0] = this.jn2120[1];
this.jn2270[0] = this.jn2270[1];
//}}}
}


/**
 *  Updates everything in the HydModel object; if historical data set is
 * used, make sure the readHistData() method is used; if WG data is used,
 * and adjustments have to be made for leap years, make sure method
 * readWGData() is used.
 *
 * @param   currentDate   Current date
 */
public void update(ModelDate currentDate) {
//{{{

// updates the currentDate instance variable
this.currentDate = currentDate;

// this is where all the parameters are set
// in java days start at index 1, and months start at index 0
// Summer season is between 01 May -- 31 Oct
// Winter season is between 01 Nov -- 30 Apr

// to take care of seasonal parameters

// methods setSummerParams() and setWinterParams() update the
// SubBasin objects with the new parameters, while the method
// updateParams() updates the objects SubBasin object consists
// of, namely SoilMoistureAccounting, Clark and LinearReservoir

if (currentDate.getMonth() > 3 &&
currentDate.getMonth() < 10) {
setSummerParams();
```

```
updateParams();
} else {
setWinterParams();
updateParams();
}

// to read the current PPT data
// this takes the daily data (i.e., 24 hrs time interval) and
// converts it into four even 6 hr values

// if using historical data set, then call the method readHistData()
// and if using WG data that need to be adjusted for leap years,
// call readWGData()

readWGData();
//readHistData();

// to update the SubBasin objects
this.sb1.update(this.sb1PPTVal, this.currentDate);
this.sb3.update(this.sb3PPTVal, this.currentDate);
this.sb4.update(this.sb4PPTVal, this.currentDate);
this.sb5.update(this.sb5PPTVal, this.currentDate);
this.sb7.update(this.sb7PPTVal, this.currentDate);
this.sb8.update(this.sb8PPTVal, this.currentDate);
this.sb9.update(this.sb9PPTVal, this.currentDate);
this.sb10.update(this.sb10PPTVal, this.currentDate);
this.sb11.update(this.sb11PPTVal, this.currentDate);
this.sb12.update(this.sb12PPTVal, this.currentDate);
this.sb13.update(this.sb13PPTVal, this.currentDate);
this.sb14.update(this.sb14PPTVal, this.currentDate);
this.sb15.update(this.sb15PPTVal, this.currentDate);
this.sb16.update(this.sb16PPTVal, this.currentDate);
this.sb17.update(this.sb17PPTVal, this.currentDate);
this.sb18.update(this.sb18PPTVal, this.currentDate);
this.sb19.update(this.sb19PPTVal, this.currentDate);
this.sb20.update(this.sb20PPTVal, this.currentDate);
this.sb21.update(this.sb21PPTVal, this.currentDate);
this.sb22.update(this.sb22PPTVal, this.currentDate);
this.sb23.update(this.sb23PPTVal, this.currentDate);
this.sb24.update(this.sb24PPTVal, this.currentDate);
this.sb25.update(this.sb25PPTVal, this.currentDate);
this.sb26.update(this.sb26PPTVal, this.currentDate);
this.sb27.update(this.sb27PPTVal, this.currentDate);
this.sb28.update(this.sb28PPTVal, this.currentDate);
this.sb29.update(this.sb29PPTVal, this.currentDate);
this.sb30.update(this.sb30PPTVal, this.currentDate);
this.sb31.update(this.sb31PPTVal, this.currentDate);
this.sb32.update(this.sb32PPTVal, this.currentDate);
this.sb33.update(this.sb33PPTVal, this.currentDate);
this.sb34.update(this.sb34PPTVal, this.currentDate);

// MODEL ASSEMBLY STARTS HERE

// NORTH BRANCH STARTS HERE
// sb1 drains into the jnMitchellSG stream gauge
this.jnMitchellSG[1] = this.sb1.getTotalFlow();

// jnMitchellSG gets routed by R560
this.R560.update(this.jnMitchellSG[0], this.jnMitchellSG[1]);

// jn640 is the sum of sb3, sb4, and routed R560
this.jn640[1] = this.R560.getOutflow() +
this.sb3.getTotalFlow() +
this.sb4.getTotalFlow();
```

```
// jn640 gets routed by R640
this.R640.update(this.jn640[0], this.jn640[1]);

// jn750 is the sum sb5, sb7, and R640
this.jn750[1] = this.R640.getOutflow() +
this.sb5.getTotalFlow() +
this.sb7.getTotalFlow();

// jn750 gets routed by R750
this.R750.update(this.jn750[0], this.jn750[1]);

// jn830 is the sum of sb8, sb9 and R750
this.jn830[1] = this.R750.getOutflow() +
this.sb8.getTotalFlow() +
this.sb9.getTotalFlow();

// jn830 gets routed by R900
this.R900.update(this.jn830[0], this.jn830[1]);

// jnUpStramWildwood is the sum of sb10
this.jnUpStreamWildwood[1] = this.sb10.getTotalFlow();

// jnUpStramWildwood gets routed by wildwood
this.wildwood.update(this.jnUpStreamWildwood[0],
this.jnUpStreamWildwood[1]);

// jnDownStreamWildwood is the outflow of wildwood
this.jnDownStreamWildwood[1] = this.wildwood.getOutflow();

// jnDownStramWildwood gets routed by R930
this.R930.update(this.jnDownStreamWildwood[0],
this.jnDownStreamWildwood[1]);

// jnStMarysSG is the sum of sb11, R900 and R930
this.jnStMarysSG[1] = this.R900.getOutflow() +
this.R930.getOutflow() +
this.sb11.getTotalFlow();

// jnStMarysSG gets routed by R1010
this.R1010.update(this.jnStMarysSG[0], this.jnStMarysSG[1]);

// jn2290 is the sum of sb12, sb13 and R1010
this.jn2290[1] = this.R1010.getOutflow() +
this.sb12.getTotalFlow() +
this.sb13.getTotalFlow();

// jn2290 gets routed by R2290
this.R2290.update(this.jn2290[0], this.jn2290[1]);

// jnPloverMillsSG is the sum of R2290 and sb14
this.jnPloverMillsSG[1] = this.R2290.getOutflow() +
this.sb14.getTotalFlow();

// jnPloverMillsSG gets routed by R2300
this.R2300.update(this.jnPloverMillsSG[0],
this.jnPloverMillsSG[1]);

// jnUpStreamFanshawe is the sum of R2300 and sb15
this.jnUpStreamFanshawe[1] = this.R2300.getOutflow() +
this.sb15.getTotalFlow();

// jnUpStreamFanshawe is routed by fanshawe
this.fanshawe.update(this.jnUpStreamFanshawe[0],
```

```
                    this.jnUpStreamFanshawe[1]);

                    // jnDownStreamFanshawe is the outflow from fanshawe
                    this.jnDownStreamFanshawe[1] = this.fanshawe.getOutflow();

                    // jnDownStreamFanshawe is routed with R1910
                    this.R1910.update(this.jnDownStreamFanshawe[0],
                    this.jnDownStreamFanshawe[1]);

                    // jn1930 is the sum of R1910, sb16, sb17
                    this.jn1930[1] = this.R1910.getOutflow() +
                    this.sb16.getTotalFlow() +
                    this.sb17.getTotalFlow();

                    // jn1930 is routed with R1930
                    this.R1930.update(this.jn1930[0], this.jn1930[1]);

                    // SOUTH BRANCH STARTS HERE
                    // jnUpStreamPittock is sb18
                    this.jnUpStreamPittock[1] = this.sb18.getTotalFlow();

                    // jnUpStreamPittock is routed by pittock
                    this.pittock.update(this.jnUpStreamPittock[0],
                    this.jnUpStreamPittock[1]);

                    // jnDownStreamPittock is the outflow from pittock
                    this.jnDownStreamPittock[1] = this.pittock.getOutflow();

                    // jnDownStreamPittock is routed with R111
                    this.R111.update(jnDownStreamPittock[0], jnDownStreamPittock[1]);

                    // jn1840 is the sum of R111, sb19, sb20
                    this.jn1840[1] = this.R111.getOutflow() +
                    this.sb19.getTotalFlow() +
                    this.sb20.getTotalFlow();

                    // jn1840 is routed by R222
                    this.R222.update(this.jn1840[0], this.jn1840[1]);

                    // jnBeachville is outflow of R222
                    this.jnBeachville[1] = this.R222.getOutflow();

                    // jnBeachville is routed by R333
                    this.R333.update(this.jnBeachville[0], this.jnBeachville[1]);

                    // jnIngersollSG is the sum of R333 and sb21
                    this.jnIngersollSG[1] = this.R333.getOutflow() +
                    this.sb21.getTotalFlow();

                    // jnIngersollSG is routed by R1870
                    this.R1870.update(this.jnIngersollSG[0], this.jnIngersollSG[1]);

                    // jnThamesfordSG is the sum of sb23
                    this.jnThamesfordSG[1] = this.sb23.getTotalFlow();

                    // jnThamesfordSG is routed by R1890
                    this.R1890.update(this.jnThamesfordSG[0], this.jnThamesfordSG[1]);

                    // jn1960 is the sum of R1870, R1890, sb22, sb24 and sb25
                    this.jn1960[1] = this.R1870.getOutflow() +
                    this.R1890.getOutflow() +
                    this.sb22.getTotalFlow() +
                    this.sb24.getTotalFlow() +
                    this.sb25.getTotalFlow();
```

```
// jn1960 is routed by R2030
this.R2030.update(this.jn1960[0], this.jn1960[1]);

// jn2050 is the sum of R2030, sb26, and sb27
this.jn2050[1] = this.R2030.getOutflow() +
this.sb26.getTotalFlow() +
this.sb27.getTotalFlow();

// jn2050 is routed by R2050
this.R2050.update(this.jn2050[0], this.jn2050[1]);

// jnEalingSG is the sum of R2050 and sb28
this.jnEalingSG[1] = this.R2050.getOutflow() +
this.sb28.getTotalFlow();

// jnEalingSG is routed by R2430
this.R2430.update(this.jnEalingSG[0], this.jnEalingSG[1]);

// THIS IS WHERE NORTH AND SOUTH BRANCH OF THE THAMES RIVER MEET
// jnForks is the sum of R1930, R2430 and sb29
this.jnForks[1] = this.R1930.getOutflow() +
this.R2430.getOutflow() +
this.sb29.getTotalFlow();

// jnForks is routed by R2440
this.R2440.update(this.jnForks[0], this.jnForks[1]);

// jnByronSG is the sum of R2440 and sb30
this.jnByronSG[1] = this.R2440.getOutflow() +
this.sb30.getTotalFlow();

// jnByronSG is routed by R2040
this.R2040.update(this.jnByronSG[0], this.jnByronSG[1]);

// jn2120 is the sum of R2040, sb31 and sb32
this.jn2120[1] = this.R2040.getOutflow() +
this.sb31.getTotalFlow() +
this.sb32.getTotalFlow();

// jn2120 is routed by R2120
this.R2120.update(this.jn2120[0], this.jn2120[1]);

// jn2270 is the sum of R2120, sb33 and sb34
this.jn2270[1] = this.R2120.getOutflow() +
this.sb33.getTotalFlow() +
this.sb34.getTotalFlow();

// reset the junctions
this.jnMitchellSG[0] = this.jnMitchellSG[1];
this.jn640[0] = this.jn640[1];
this.jn750[0] = this.jn750[1];
this.jn830[0] = this.jn830[1];
this.jnUpStreamWildwood[0] = this.jnUpStreamWildwood[1];
this.jnDownStreamWildwood[0] = this.jnDownStreamWildwood[1];
this.jnStMarysSG[0] = this.jnStMarysSG[1];
this.jn2290[0] = this.jn2290[1];
this.jnPloverMillsSG[0] = this.jnPloverMillsSG[1];
this.jnUpStreamFanshawe[0] = this.jnUpStreamFanshawe[1];
this.jnDownStreamFanshawe[0] = this.jnDownStreamFanshawe[1];
this.jn1930[0] = this.jn1930[1];
this.jnUpStreamPittock[0] = this.jnUpStreamPittock[1];
this.jnDownStreamPittock[0] = this.jnDownStreamPittock[1];
this.jn1840[0] = this.jn1840[1];
```

```
this.jnBeachville[0] = this.jnBeachville[1];
this.jnIngersollSG[0] = this.jnIngersollSG[1];
this.jnThamesfordSG[0] = this.jnThamesfordSG[1];
this.jn1960[0] = this.jn1960[1];
this.jn2050[0] = this.jn2050[1];
this.jnEalingSG[0] = this.jnEalingSG[1];
this.jnForks[0] = this.jnForks[1];
this.jnByronSG[0] = this.jnByronSG[1];
this.jn2120[0] = this.jn2120[1];
this.jn2270[0] = this.jn2270[1];
//}}}

}


// these are the methods that update the parameters of the HydModel
// as simulated time goes on
//{{{
/**
 *  A method that simply initializes the parameters of the SubBasin
 * objects that have been previously instantiated.  These parameters are
 * for the summer season.
 */
private void initializeParams() {
this.sb1.setPhysicalProps(305.505, 0.0);
this.sb1.setMaxStores(2.0, 22.0, 60.0, 15.0, 45.0, 40.0);
this.sb1.setMaxRates(4.8, 3.0, 1.0, 1.0);
this.sb1.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb1.setRoutGWStorage(150.0, 290.0);
this.sb1.setBaseflowParams(25.0, 1, 35.0, 7);
this.sb1.setClarkParams(24.0, 22.0);
this.sb1.setETZoneParams(this.evapNOR, this.panCoeff);
this.sb1.setPETReductionCoeff(1.0);

this.sb3.setPhysicalProps(47.745, 0.0);
this.sb3.setMaxStores(2.0, 34.0, 57.0, 21.0, 40.0, 40.0);
this.sb3.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb3.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb3.setRoutGWStorage(180.0, 260.0);
this.sb3.setBaseflowParams(25.0, 1, 40.0, 6);
this.sb3.setClarkParams(6.0, 8.0);
this.sb3.setETZoneParams(this.evapMID, this.panCoeff);
this.sb3.setPETReductionCoeff(1.0);

this.sb4.setPhysicalProps(151.189, 0.0);
this.sb4.setMaxStores(2.0, 23.0, 70.0, 15.0, 50.0, 50.0);
this.sb4.setMaxRates(4.8, 3.0, 2.0, 1.0);
this.sb4.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb4.setRoutGWStorage(175.0, 230.0);
this.sb4.setBaseflowParams(25.0, 1, 60.0, 7);
this.sb4.setClarkParams(24.0, 30.0);
this.sb4.setETZoneParams(this.evapMID, this.panCoeff);
this.sb4.setPETReductionCoeff(1.0);

this.sb5.setPhysicalProps(76.82, 0.0);
this.sb5.setMaxStores(2.0, 23.0, 70.0, 15.0, 50.0, 50.0);
this.sb5.setMaxRates(4.8, 3.0, 2.0, 1.0);
this.sb5.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb5.setRoutGWStorage(175.0, 230.0);
this.sb5.setBaseflowParams(25.0, 1, 55.0, 7);
this.sb5.setClarkParams(18.0, 18.0);
this.sb5.setETZoneParams(this.evapMID, this.panCoeff);
this.sb5.setPETReductionCoeff(1.0);
```

```
this.sb7.setPhysicalProps(144.0, 2.0);
this.sb7.setMaxStores(2.0, 23.0, 70.0, 15.0, 50.0, 50.0);
this.sb7.setMaxRates(4.8, 3.0, 2.0, 1.0);
this.sb7.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb7.setRoutGWStorage(175.0, 230.0);
this.sb7.setBaseflowParams(25.0, 1, 65.0, 10);
this.sb7.setClarkParams(12.0, 24.0);
this.sb7.setETZoneParams(this.evapMID, this.panCoeff);
this.sb7.setPETReductionCoeff(1.0);


this.sb8.setPhysicalProps(88.355, 0.0);
this.sb8.setMaxStores(2.0, 34.0, 57.0, 21.0, 40.0, 40.0);
this.sb8.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb8.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb8.setRoutGWStorage(180.0, 260.0);
this.sb8.setBaseflowParams(25.0, 1, 45.0, 6);
// originally this was (16,20) but I changed it so that I
// wouldn't have to do another table function for Tc = 16 hrs
this.sb8.setClarkParams(18.0, 20.0);
this.sb8.setETZoneParams(this.evapMID, this.panCoeff);
this.sb8.setPETReductionCoeff(1.0);


this.sb9.setPhysicalProps(78.476, 0.0);
this.sb9.setMaxStores(2.0, 38.0, 72.0, 22.0, 45.0, 50.0);
this.sb9.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb9.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb9.setRoutGWStorage(170.0, 250.0);
this.sb9.setBaseflowParams(25.0, 1, 55.0, 8);
this.sb9.setClarkParams(12.0, 20.0);
this.sb9.setETZoneParams(this.evapMID, this.panCoeff);
this.sb9.setPETReductionCoeff(1.0);


this.sb10.setPhysicalProps(141.118, 0.0);
this.sb10.setMaxStores(2.0, 38.0, 72.0, 22.0, 45.0, 50.0);
this.sb10.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb10.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb10.setRoutGWStorage(170.0, 250.0);
this.sb10.setBaseflowParams(25.0, 1, 55.0, 8);
// originally this was (22,24) but I changed it so that I
// wouldn't have to do another table function for Tc = 22 hrs
this.sb10.setClarkParams(24.0, 24.0);
this.sb10.setETZoneParams(this.evapMID, this.panCoeff);
this.sb10.setPETReductionCoeff(1.0);


this.sb11.setPhysicalProps(28.942, 0.0);
this.sb11.setMaxStores(2.0, 38.0, 72.0, 22.0, 45.0, 50.0);
this.sb11.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb11.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb11.setRoutGWStorage(170.0, 250.0);
this.sb11.setBaseflowParams(25.0, 1, 55.0, 7);
this.sb11.setClarkParams(12.0, 18.0);
this.sb11.setETZoneParams(this.evapMID, this.panCoeff);
this.sb11.setPETReductionCoeff(1.0);


this.sb12.setPhysicalProps(35.466, 0.0);
this.sb12.setMaxStores(2.0, 32.0, 55.0, 17.0, 40.0, 45.0);
this.sb12.setMaxRates(5.0, 3.0, 2.0, 1.0);
this.sb12.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb12.setRoutGWStorage(150.0, 280.0);
this.sb12.setBaseflowParams(25.0, 1, 70.0, 7);
this.sb12.setClarkParams(18.0, 18.0);
this.sb12.setETZoneParams(this.evapMID, this.panCoeff);
this.sb12.setPETReductionCoeff(1.0);
```

```
this.sb13.setPhysicalProps(153.721, 0.0);
this.sb13.setMaxStores(2.0, 34.0, 57.0, 21.0, 40.0, 40.0);
this.sb13.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb13.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb13.setRoutGWStorage(180.0, 260.0);
this.sb13.setBaseflowParams(25.0, 1, 50.0, 6);
this.sb13.setClarkParams(24.0, 24.0);
this.sb13.setETZoneParams(this.evapMID, this.panCoeff);
this.sb13.setPETReductionCoeff(1.0);

this.sb14.setPhysicalProps(84.539, 0.0);
this.sb14.setMaxStores(2.0, 32.0, 55.0, 17.0, 40.0, 45.0);
this.sb14.setMaxRates(5.0, 3.0, 2.0, 1.0);
this.sb14.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb14.setRoutGWStorage(150.0, 280.0);
this.sb14.setBaseflowParams(25.0, 1, 55.0, 7);
this.sb14.setClarkParams(18.0, 18.0);
this.sb14.setETZoneParams(this.evapMID, this.panCoeff);
this.sb14.setPETReductionCoeff(1.0);

this.sb15.setPhysicalProps(94.198, 0.0);
this.sb15.setMaxStores(2.0, 32.0, 55.0, 17.0, 40.0, 45.0);
this.sb15.setMaxRates(5.0, 3.0, 2.0, 1.0);
this.sb15.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb15.setRoutGWStorage(150.0, 280.0);
this.sb15.setBaseflowParams(25.0, 1, 75.0, 7);
this.sb15.setClarkParams(12.0, 18.0);
this.sb15.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb15.setPETReductionCoeff(1.0);

this.sb16.setPhysicalProps(75.363, 5.0);
this.sb16.setMaxStores(2.0, 26.0, 58.0, 18.0, 58.0, 55.0);
this.sb16.setMaxRates(5.0, 3.0, 2.0, 1.0);
this.sb16.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb16.setRoutGWStorage(145.0, 290.0);
this.sb16.setBaseflowParams(25.0, 1, 75.0, 7);
this.sb16.setClarkParams(12.0, 18.0);
this.sb16.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb16.setPETReductionCoeff(1.0);

this.sb17.setPhysicalProps(202.478, 0.0);
this.sb17.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
this.sb17.setMaxRates(2.5, 3.0, 2.0, 1.0);
this.sb17.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb17.setRoutGWStorage(225.0, 275.0);
this.sb17.setBaseflowParams(25.0, 1, 30.0, 6);
this.sb17.setClarkParams(18.0, 20.0);
this.sb17.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb17.setPETReductionCoeff(1.0);

this.sb18.setPhysicalProps(148.318, 0.0);
this.sb18.setMaxStores(2.0, 34.0, 70.0, 30.0, 50.0, 40.0);
this.sb18.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb18.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb18.setRoutGWStorage(225.0, 275.0);
this.sb18.setBaseflowParams(25.0, 1, 75.0, 7);
this.sb18.setClarkParams(18.0, 24.0);
this.sb18.setETZoneParams(this.evapMID, this.panCoeff);
this.sb18.setPETReductionCoeff(1.0);

this.sb19.setPhysicalProps(96.84, 0.0);
this.sb19.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
this.sb19.setMaxRates(4.9, 2.0, 2.0, 1.0);
this.sb19.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
```

```
        this.sb19.setRoutGWStorage(140.0, 300.0);
        this.sb19.setBaseflowParams(25.0, 1, 75.0, 7);
        this.sb19.setClarkParams(6.0, 10.0);
        this.sb19.setETZoneParams(this.evapSUD, this.panCoeff);
        this.sb19.setPETReductionCoeff(1.0);

        this.sb20.setPhysicalProps(97.91, 0.0);
        this.sb20.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
        this.sb20.setMaxRates(4.9, 2.0, 2.0, 1.0);
        this.sb20.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
        this.sb20.setRoutGWStorage(140.0, 300.0);
        this.sb20.setBaseflowParams(25.0, 1, 70.0, 7);
        this.sb20.setClarkParams(18.0, 26.0);
        this.sb20.setETZoneParams(this.evapSUD, this.panCoeff);
        this.sb20.setPETReductionCoeff(1.0);

        this.sb21.setPhysicalProps(170.704, 0.0);
        this.sb21.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
        this.sb21.setMaxRates(4.9, 2.0, 2.0, 1.0);
        this.sb21.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
        this.sb21.setRoutGWStorage(140.0, 300.0);
        this.sb21.setBaseflowParams(25.0, 1, 80.0, 7);
        this.sb21.setClarkParams(18.0, 22.0);
        this.sb21.setETZoneParams(this.evapSUD, this.panCoeff);
        this.sb21.setPETReductionCoeff(1.0);

        this.sb22.setPhysicalProps(42.859, 0.0);
        this.sb22.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
        this.sb22.setMaxRates(4.9, 2.0, 2.0, 1.0);
        this.sb22.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
        this.sb22.setRoutGWStorage(140.0, 300.0);
        this.sb22.setBaseflowParams(25.0, 1, 65.0, 7);
        this.sb22.setClarkParams(12.0, 18.0);
        this.sb22.setETZoneParams(this.evapSUD, this.panCoeff);
        this.sb22.setPETReductionCoeff(1.0);

        this.sb23.setPhysicalProps(291.08, 0.0);
        this.sb23.setMaxStores(2.0, 23.0, 80.0, 50.0, 70.0, 50.0);
        this.sb23.setMaxRates(4.8, 3.5, 2.0, 1.0);
        this.sb23.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
        this.sb23.setRoutGWStorage(130.0, 290.0);
        this.sb23.setBaseflowParams(25.0, 1, 75.0, 7);
        this.sb23.setClarkParams(24.0, 25.0);
        this.sb23.setETZoneParams(this.evapMID, this.panCoeff);
        this.sb23.setPETReductionCoeff(1.0);

        this.sb24.setPhysicalProps(35.861, 0.0);
        this.sb24.setMaxStores(2.0, 23.0, 80.0, 50.0, 70.0, 50.0);
        this.sb24.setMaxRates(4.8, 3.5, 2.0, 1.0);
        this.sb24.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
        this.sb24.setRoutGWStorage(130.0, 290.0);
        this.sb24.setBaseflowParams(25.0, 1, 65.0, 7);
        this.sb24.setClarkParams(12.0, 18.0);
        this.sb24.setETZoneParams(this.evapSUD, this.panCoeff);
        this.sb24.setPETReductionCoeff(1.0);

        this.sb25.setPhysicalProps(165.973, 0.0);
        this.sb25.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
        this.sb25.setMaxRates(4.9, 2.0, 2.0, 1.0);
        this.sb25.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
        this.sb25.setRoutGWStorage(140.0, 300.0);
        this.sb25.setBaseflowParams(25.0, 1, 75.0, 7);
        this.sb25.setClarkParams(24.0, 24.0);
        this.sb25.setETZoneParams(this.evapSUD, this.panCoeff);
```

```
this.sb25.setPETReductionCoeff(1.0);

this.sb26.setPhysicalProps(120.935, 0.0);
this.sb26.setMaxStores(2.0, 37.0, 60.0, 40.0, 70.0, 50.0);
this.sb26.setMaxRates(5.0, 3.5, 2.0, 1.0);
this.sb26.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb26.setRoutGWStorage(130.0, 300.0);
this.sb26.setBaseflowParams(25.0, 1, 75.0, 7);
this.sb26.setClarkParams(12.0, 24.0);
this.sb26.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb26.setPETReductionCoeff(1.0);

this.sb27.setPhysicalProps(104.945, 0.0);
this.sb27.setMaxStores(2.0, 37.0, 60.0, 40.0, 70.0, 50.0);
this.sb27.setMaxRates(5.0, 3.5, 2.0, 1.0);
this.sb27.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb27.setRoutGWStorage(130.0, 300.0);
this.sb27.setBaseflowParams(25.0, 1, 70.0, 7);
this.sb27.setClarkParams(18.0, 22.0);
this.sb27.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb27.setPETReductionCoeff(1.0);

this.sb28.setPhysicalProps(61.195, 0.0);
this.sb28.setMaxStores(2.0, 37.0, 60.0, 40.0, 70.0, 50.0);
this.sb28.setMaxRates(5.0, 3.5, 2.0, 1.0);
this.sb28.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb28.setRoutGWStorage(130.0, 300.0);
this.sb28.setBaseflowParams(25.0, 1, 70.0, 7);
this.sb28.setClarkParams(12.0, 18.0);
this.sb28.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb28.setPETReductionCoeff(1.0);

this.sb29.setPhysicalProps(22.556, 40.0);
this.sb29.setMaxStores(2.0, 26.0, 58.0, 18.0, 58.0, 55.0);
this.sb29.setMaxRates(5.0, 3.0, 2.0, 1.0);
this.sb29.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb29.setRoutGWStorage(145.0, 290.0);
this.sb29.setBaseflowParams(25.0, 1, 50.0, 6);
this.sb29.setClarkParams(6.0, 8.0);
this.sb29.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb29.setPETReductionCoeff(1.0);

this.sb30.setPhysicalProps(30.002, 30.0);
this.sb30.setMaxStores(2.0, 26.0, 58.0, 18.0, 58.0, 55.0);
this.sb30.setMaxRates(5.0, 3.0, 2.0, 1.0);
this.sb30.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb30.setRoutGWStorage(145.0, 290.0);
this.sb30.setBaseflowParams(25.0, 1, 45.0, 6);
this.sb30.setClarkParams(6.0, 11.0);
this.sb30.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb30.setPETReductionCoeff(1.0);

this.sb31.setPhysicalProps(32.409, 0.0);
this.sb31.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
this.sb31.setMaxRates(2.5, 3.0, 2.0, 1.0);
this.sb31.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb31.setRoutGWStorage(225.0, 275.0);
this.sb31.setBaseflowParams(25.0, 1, 45.0, 6);
this.sb31.setClarkParams(6.0, 8.0);
this.sb31.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb31.setPETReductionCoeff(1.0);

this.sb32.setPhysicalProps(88.145, 0.0);
this.sb32.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
```

```
this.sb32.setMaxRates(2.5, 3.0, 2.0, 1.0);
this.sb32.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb32.setRoutGWStorage(225.0, 275.0);
this.sb32.setBaseflowParams(25.0, 1, 65.0, 7);
this.sb32.setClarkParams(18.0, 18.0);
this.sb32.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb32.setPETReductionCoeff(1.0);

this.sb33.setPhysicalProps(50.486, 0.0);
this.sb33.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
this.sb33.setMaxRates(2.5, 3.0, 2.0, 1.0);
this.sb33.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb33.setRoutGWStorage(225.0, 275.0);
this.sb33.setBaseflowParams(25.0, 1, 45.0, 6);
this.sb33.setClarkParams(6.0, 10.0);
this.sb33.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb33.setPETReductionCoeff(1.0);

this.sb34.setPhysicalProps(168.719, 2.0);
this.sb34.setMaxStores(2.0, 31.0, 55.0, 10.0, 60.0, 45.0);
this.sb34.setMaxRates(4.9, 3.0, 2.0, 1.0);
this.sb34.setInitStores(0.0, 0.0, 0.0, 5.0, 100.0);
this.sb34.setRoutGWStorage(140.0, 290.0);
this.sb34.setBaseflowParams(25.0, 1, 55.0, 7);
this.sb34.setClarkParams(24.0, 28.0);
this.sb34.setETZoneParams(this.evapSUD, this.panCoeff);
this.sb34.setPETReductionCoeff(1.0);
}


/**
 *  Sets the summerParams attribute of the HydModel object; this is
 * where summer season parameters are set.
 */
private void setSummerParams() {
this.sb1.setMaxStores(2.0, 22.0, 60.0, 15.0, 45.0, 40.0);
this.sb1.setMaxRates(4.8, 3.0, 1.0, 1.0);

this.sb3.setMaxStores(2.0, 34.0, 57.0, 21.0, 40.0, 40.0);
this.sb3.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb4.setMaxStores(2.0, 23.0, 65.0, 15.0, 50.0, 50.0);
this.sb4.setMaxRates(4.8, 3.0, 2.0, 1.0);

this.sb5.setMaxStores(2.0, 23.0, 70.0, 15.0, 50.0, 50.0);
this.sb5.setMaxRates(4.8, 3.0, 2.0, 1.0);

this.sb7.setMaxStores(2.0, 23.0, 65.0, 15.0, 50.0, 50.0);
this.sb7.setMaxRates(4.8, 3.0, 2.0, 1.0);

this.sb8.setMaxStores(2.0, 34.0, 57.0, 21.0, 40.0, 40.0);
this.sb8.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb9.setMaxStores(2.0, 38.0, 72.0, 22.0, 45.0, 50.0);
this.sb9.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb10.setMaxStores(2.0, 38.0, 72.0, 22.0, 45.0, 50.0);
this.sb10.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb11.setMaxStores(2.0, 38.0, 72.0, 22.0, 45.0, 50.0);
this.sb11.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb12.setMaxStores(2.0, 32.0, 55.0, 17.0, 40.0, 45.0);
this.sb12.setMaxRates(5.0, 3.0, 2.0, 1.0);
```

```
this.sb13.setMaxStores(2.0, 34.0, 57.0, 21.0, 40.0, 40.0);
this.sb13.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb14.setMaxStores(2.0, 32.0, 55.0, 17.0, 40.0, 45.0);
this.sb14.setMaxRates(5.0, 3.0, 2.0, 1.0);

this.sb15.setMaxStores(2.0, 32.0, 55.0, 17.0, 40.0, 45.0);
this.sb15.setMaxRates(5.0, 3.0, 2.0, 1.0);

this.sb16.setMaxStores(2.0, 26.0, 58.0, 18.0, 58.0, 55.0);
this.sb16.setMaxRates(5.0, 3.0, 2.0, 1.0);

this.sb17.setMaxStores(2.0, 30.0, 60.0, 20.0, 40.0, 40.0);
this.sb17.setMaxRates(5.0, 3.0, 2.0, 1.0);

this.sb18.setMaxStores(2.0, 34.0, 70.0, 50.0, 50.0, 40.0);
this.sb18.setMaxRates(4.9, 3.0, 2.0, 1.0);

this.sb19.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
this.sb19.setMaxRates(4.9, 2.0, 2.0, 1.0);

this.sb20.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
this.sb20.setMaxRates(4.9, 2.0, 2.0, 1.0);

this.sb21.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
this.sb21.setMaxRates(4.9, 2.0, 2.0, 1.0);

this.sb22.setMaxStores(2.0, 37.0, 80.0, 40.0, 60.0, 40.0);
this.sb22.setMaxRates(4.9, 2.0, 2.0, 1.0);

this.sb23.setMaxStores(2.0, 23.0, 80.0, 50.0, 70.0, 50.0);
this.sb23.setMaxRates(4.8, 3.5, 2.0, 1.0);

this.sb24.setMaxStores(2.0, 23.0, 80.0, 50.0, 70.0, 50.0);
this.sb24.setMaxRates(4.8, 3.5, 2.0, 1.0);

this.sb25.setMaxStores(2.0, 37.0, 90.0, 40.0, 60.0, 40.0);
this.sb25.setMaxRates(4.9, 2.0, 2.0, 1.0);

this.sb26.setMaxStores(2.0, 37.0, 60.0, 40.0, 70.0, 50.0);
this.sb26.setMaxRates(5.0, 3.5, 2.0, 1.0);

this.sb27.setMaxStores(2.0, 37.0, 60.0, 40.0, 70.0, 50.0);
this.sb27.setMaxRates(5.0, 3.5, 2.0, 1.0);

this.sb28.setMaxStores(2.0, 37.0, 60.0, 40.0, 70.0, 50.0);
this.sb28.setMaxRates(5.0, 3.5, 2.0, 1.0);

this.sb29.setMaxStores(2.0, 26.0, 58.0, 18.0, 58.0, 55.0);
this.sb29.setMaxRates(5.0, 3.0, 2.0, 1.0);

this.sb30.setMaxStores(2.0, 26.0, 58.0, 18.0, 58.0, 55.0);
this.sb30.setMaxRates(5.0, 3.0, 2.0, 1.0);

this.sb31.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
this.sb31.setMaxRates(2.5, 3.0, 2.0, 1.0);

this.sb32.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
this.sb32.setMaxRates(2.5, 3.0, 2.0, 1.0);

this.sb33.setMaxStores(2.0, 30.0, 50.0, 20.0, 40.0, 40.0);
this.sb33.setMaxRates(2.5, 3.0, 2.0, 1.0);
```

```
this.sb34.setMaxStores(2.0, 31.0, 55.0, 10.0, 60.0, 45.0);
this.sb34.setMaxRates(4.9, 3.0, 2.0, 1.0);


}


/**
 *  Sets the winterParams attribute of the HydModel object; this is
 * where the winter parameters are set.
 */
private void setWinterParams() {
this.sb1.setMaxStores(2.0, 11.0, 50.0, 25.0, 45.0, 40.0);
this.sb1.setMaxRates(0.2, 3.0, 1.0, 1.0);

this.sb3.setMaxStores(2.0, 15.0, 57.0, 27.0, 40.0, 40.0);
this.sb3.setMaxRates(0.36, 3.0, 2.0, 1.0);

this.sb4.setMaxStores(2.0, 10.0, 70.0, 20.0, 50.0, 50.0);
this.sb4.setMaxRates(0.35, 3.0, 2.0, 1.0);

this.sb5.setMaxStores(2.0, 12.0, 70.0, 20.0, 50.0, 50.0);
this.sb5.setMaxRates(0.35, 3.0, 2.0, 1.0);

this.sb7.setMaxStores(2.0, 8.0, 70.0, 20.0, 50.0, 50.0);
this.sb7.setMaxRates(0.35, 3.0, 2.0, 1.0);

this.sb8.setMaxStores(2.0, 17.0, 57.0, 25.0, 40.0, 40.0);
this.sb8.setMaxRates(0.36, 3.0, 2.0, 1.0);

this.sb9.setMaxStores(2.0, 12.0, 72.0, 25.0, 45.0, 50.0);
this.sb9.setMaxRates(0.36, 3.0, 2.0, 1.0);

this.sb10.setMaxStores(2.0, 17.0, 72.0, 25.0, 45.0, 50.0);
this.sb10.setMaxRates(0.36, 3.0, 2.0, 1.0);

this.sb11.setMaxStores(2.0, 17.0, 72.0, 25.0, 45.0, 50.0);
this.sb11.setMaxRates(0.36, 3.0, 2.0, 1.0);

this.sb12.setMaxStores(2.0, 19.0, 55.0, 17.0, 40.0, 45.0);
this.sb12.setMaxRates(0.37, 3.0, 2.0, 1.0);

this.sb13.setMaxStores(2.0, 16.0, 57.0, 21.0, 40.0, 40.0);
this.sb13.setMaxRates(0.36, 3.0, 2.0, 1.0);

this.sb14.setMaxStores(2.0, 17.0, 55.0, 17.0, 40.0, 45.0);
this.sb14.setMaxRates(0.37, 3.0, 2.0, 1.0);

this.sb15.setMaxStores(2.0, 16.0, 55.0, 17.0, 40.0, 45.0);
this.sb15.setMaxRates(0.37, 3.0, 2.0, 1.0);

this.sb16.setMaxStores(2.0, 16.0, 58.0, 18.0, 58.0, 55.0);
this.sb16.setMaxRates(0.37, 3.0, 2.0, 1.0);

this.sb17.setMaxStores(2.0, 13.0, 60.0, 20.0, 40.0, 40.0);
this.sb17.setMaxRates(0.35, 3.0, 2.0, 1.0);

this.sb18.setMaxStores(2.0, 17.0, 60.0, 40.0, 50.0, 40.0);
this.sb18.setMaxRates(0.35, 3.0, 2.0, 1.0);

this.sb19.setMaxStores(2.0, 19.0, 85.0, 40.0, 60.0, 40.0);
this.sb19.setMaxRates(0.8, 2.0, 2.0, 1.0);

this.sb20.setMaxStores(2.0, 19.0, 85.0, 40.0, 60.0, 40.0);
this.sb20.setMaxRates(0.8, 2.0, 2.0, 1.0);
```

```
        this.sb21.setMaxStores(2.0, 19.0, 80.0, 40.0, 60.0, 40.0);
        this.sb21.setMaxRates(1.0, 2.0, 2.0, 1.0);

        this.sb22.setMaxStores(2.0, 19.0, 80.0, 40.0, 60.0, 40.0);
        this.sb22.setMaxRates(0.36, 2.0, 2.0, 1.0);

        this.sb23.setMaxStores(2.0, 12.0, 80.0, 50.0, 70.0, 50.0);
        this.sb23.setMaxRates(0.35, 3.5, 2.0, 1.0);

        this.sb24.setMaxStores(2.0, 12.0, 80.0, 50.0, 70.0, 50.0);
        this.sb24.setMaxRates(0.35, 3.5, 2.0, 1.0);

        this.sb25.setMaxStores(2.0, 19.0, 90.0, 40.0, 60.0, 40.0);
        this.sb25.setMaxRates(0.36, 2.0, 2.0, 1.0);

        this.sb26.setMaxStores(2.0, 19.0, 60.0, 40.0, 70.0, 50.0);
        this.sb26.setMaxRates(0.37, 3.5, 2.0, 1.0);

        this.sb27.setMaxStores(2.0, 19.0, 60.0, 50.0, 70.0, 50.0);
        this.sb27.setMaxRates(0.37, 3.5, 2.0, 1.0);

        this.sb28.setMaxStores(2.0, 19.0, 60.0, 40.0, 70.0, 50.0);
        this.sb28.setMaxRates(0.37, 3.5, 2.0, 1.0);

        this.sb29.setMaxStores(2.0, 13.0, 58.0, 18.0, 58.0, 55.0);
        this.sb29.setMaxRates(0.37, 3.0, 2.0, 1.0);

        this.sb30.setMaxStores(2.0, 13.0, 58.0, 18.0, 58.0, 55.0);
        this.sb30.setMaxRates(0.37, 3.0, 2.0, 1.0);

        this.sb31.setMaxStores(2.0, 15.0, 50.0, 20.0, 40.0, 40.0);
        this.sb31.setMaxRates(0.18, 3.0, 2.0, 1.0);

        this.sb32.setMaxStores(2.0, 15.0, 50.0, 20.0, 40.0, 40.0);
        this.sb32.setMaxRates(0.18, 3.0, 2.0, 1.0);

        this.sb33.setMaxStores(2.0, 15.0, 50.0, 20.0, 40.0, 40.0);
        this.sb33.setMaxRates(0.18, 3.0, 2.0, 1.0);

        this.sb34.setMaxStores(2.0, 16.0, 55.0, 10.0, 60.0, 45.0);
        this.sb34.setMaxRates(0.36, 3.0, 2.0, 1.0);
        }


    /**
     *  Once the parameters of the SubBasin objects change, they must
     * communicate to the SoilMoistureAccounting objects and tell them to
     * change as well.  This is accomplished with this method.  See a
     * description of the same method under SoilMoistureAccounting.
     */
    private void updateParams() {
    this.sb1.updateParams();
    this.sb3.updateParams();
    this.sb4.updateParams();
    this.sb5.updateParams();
    this.sb7.updateParams();
    this.sb8.updateParams();
    this.sb9.updateParams();
    this.sb10.updateParams();
    this.sb11.updateParams();
    this.sb12.updateParams();
    this.sb13.updateParams();
    this.sb14.updateParams();
```

```
this.sb15.updateParams();
this.sb16.updateParams();
this.sb17.updateParams();
this.sb18.updateParams();
this.sb19.updateParams();
this.sb20.updateParams();
this.sb21.updateParams();
this.sb22.updateParams();
this.sb23.updateParams();
this.sb24.updateParams();
this.sb25.updateParams();
this.sb26.updateParams();
this.sb27.updateParams();
this.sb28.updateParams();
this.sb29.updateParams();
this.sb30.updateParams();
this.sb31.updateParams();
this.sb32.updateParams();
this.sb33.updateParams();
this.sb34.updateParams();
}


/**
 *  This method reads data that were generated previously by the
 * contUtils.java methods.  This method is used to read the WG data
 * for the case when the data is given for a long time series, but does
 * not include the leap year values.  For example, if 100 years of daily
 * data is given without consideration of leap years (i.e., 36500 data
 * points), this would cause problems on the timing and magnitude of
 * extreme events.  As there are 24 leap years per century, after
 * 100 years the data would lag behind about a month.  For 300 years, it
 * would lag about 3 months, which would mean that spring snow melt
 * would now be occuring in January, which is not right.  This method
 * simply takes the 28 Feb value and assigns it to 29 Feb, provided it
 * is a leap year.
 */
private void readWGData() {

// this reason for <=2 is that the model starts at 01:00 hrs
// and because of daylight savings time, the hours shift
// since the model operates on the userTimeStep of 6 hrs, which
// is  different from dataTimeStep of 24 hrs, the data is read
// from the file one once every day and divided by
// 4 or by (dataTimeStep / userTimeStep)

// since the WG scenarios do not generate data for leap years
// 29 Feb data will be equated to the data of the previous day
// this feature should be disabled when running the model for
// historically observed ppt because the historical record will
// contain ppt values for 29 Feb

if ((this.currentDate.getMonth() == 1) &&
(this.currentDate.getDay() == 29) &&
(this.currentDate.getHour() <= 2)) {
// do nothing; actually, for this time step, it will
// use the previous data value

} else {

if (this.currentDate.getHour() <= 2) {
this.sb1PPTVal = this.sb1PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);
```

```
this.sb3PPTVal = this.sb3PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb4PPTVal = this.sb4PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb5PPTVal = this.sb5PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb7PPTVal = this.sb7PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb8PPTVal = this.sb8PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb9PPTVal = this.sb9PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb10PPTVal = this.sb10PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb11PPTVal = this.sb11PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb12PPTVal = this.sb12PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb13PPTVal = this.sb13PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb14PPTVal = this.sb14PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb15PPTVal = this.sb15PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb16PPTVal = this.sb16PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb17PPTVal = this.sb17PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb18PPTVal = this.sb18PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb19PPTVal = this.sb19PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb20PPTVal = this.sb20PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb21PPTVal = this.sb21PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb22PPTVal = this.sb22PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb23PPTVal = this.sb23PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb24PPTVal = this.sb24PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb25PPTVal = this.sb25PPT.readCurrentData() /
```

```
(dataTimeStep / this.userTimeStep);

this.sb26PPTVal = this.sb26PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb27PPTVal = this.sb27PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb28PPTVal = this.sb28PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb29PPTVal = this.sb29PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb30PPTVal = this.sb30PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb31PPTVal = this.sb31PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb32PPTVal = this.sb32PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb33PPTVal = this.sb33PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb34PPTVal = this.sb34PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);
}
}
}


/**
 *  This method reads data that were generated previously by the
 * contUtils.java methods.  This method is used to read the historical
 * data, that already has leap year values in the historical record.
 */
private void readHistData() {

// this reason for <=2 is that the model starts at 01:00 hrs
// and because of daylight savings time, the hours shift
// since the model operates on the userTimeStep of 6 hrs, which
// is  different from dataTimeStep of 24 hrs, the data is read
// from the file one once every day and divided by
// 4 or by (dataTimeStep / userTimeStep)

if (this.currentDate.getHour() <= 2) {
this.sb1PPTVal = this.sb1PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb3PPTVal = this.sb3PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb4PPTVal = this.sb4PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb5PPTVal = this.sb5PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb7PPTVal = this.sb7PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb8PPTVal = this.sb8PPT.readCurrentData() /
```

```
(dataTimeStep / this.userTimeStep);

this.sb9PPTVal = this.sb9PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb10PPTVal = this.sb10PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb11PPTVal = this.sb11PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb12PPTVal = this.sb12PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb13PPTVal = this.sb13PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb14PPTVal = this.sb14PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb15PPTVal = this.sb15PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb16PPTVal = this.sb16PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb17PPTVal = this.sb17PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb18PPTVal = this.sb18PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb19PPTVal = this.sb19PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb20PPTVal = this.sb20PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb21PPTVal = this.sb21PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb22PPTVal = this.sb22PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb23PPTVal = this.sb23PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb24PPTVal = this.sb24PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb25PPTVal = this.sb25PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb26PPTVal = this.sb26PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb27PPTVal = this.sb27PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb28PPTVal = this.sb28PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb29PPTVal = this.sb29PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);
```

```
this.sb30PPTVal = this.sb30PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb31PPTVal = this.sb31PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb32PPTVal = this.sb32PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb33PPTVal = this.sb33PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);

this.sb34PPTVal = this.sb34PPT.readCurrentData() /
(dataTimeStep / this.userTimeStep);
}
}


//}}}

// set methods; these are the places where the SysModel gives the info
// to the HydModel
//{{{
/**
 *  Sets the maxSurfStoreReductionTables attribute of the HydModel object
 *
 * @param  FPLMiddlesex  The new maxSurfStoreReductionTables value
 * @param  FPLOxford     The new maxSurfStoreReductionTables value
 * @param  FPLPerth      The new maxSurfStoreReductionTables value
 */
public void setMaxSurfStoreReductionTables(double FPLMiddlesex,
double FPLOxford, double FPLPerth) {

this.FPLMiddlesex = FPLMiddlesex;
this.FPLOxford = FPLOxford;
this.FPLPerth = FPLPerth;
/*
 *  // original tables
 *  double[] xFRLMiddlesex = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
 *  xFRLMiddlesex[0] = this.FPLMiddlesex;
 *  double[] yMaxSurfStoreRedMultMiddlesex =
 *  {1.0, 0.9, 0.7, 0.4, 0.2, 0.1};
 *  this.maxSurfStoreRedMultMiddlesex = new Table(
 *  xFRLMiddlesex, yMaxSurfStoreRedMultMiddlesex);
 *  double[] xFRLOxford = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
 *  xFRLOxford[0] = this.FPLOxford;
 *  double[] yMaxSurfStoreRedMultOxford =
 *  {1.0, 0.9, 0.7, 0.4, 0.2, 0.1};
 *  this.maxSurfStoreRedMultOxford = new Table(
 *  xFRLOxford, yMaxSurfStoreRedMultOxford);
 *  double[] xFRLPerth = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
 *  xFRLPerth[0] = this.FPLPerth;
 *  double[] yMaxSurfStoreRedMultPerth =
 *  {1.0, 0.9, 0.7, 0.4, 0.2, 0.1};
 *  this.maxSurfStoreRedMultPerth = new Table(
 *  xFRLPerth, yMaxSurfStoreRedMultPerth);
 */
// new tables, as of 01 June 2006
double[] xFPLMiddlesex = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
xFPLMiddlesex[0] = this.FPLMiddlesex;
double[] yMaxSurfStoreRedMultMiddlesex =
{1.0, 0.4, 0.2, 0.15, 0.12, 0.1};
this.maxSurfStoreRedMultMiddlesex = new Table(
```

```
xFPLMiddlesex, yMaxSurfStoreRedMultMiddlesex);

double[] xFPLOxford = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
xFPLOxford[0] = this.FPLOxford;
double[] yMaxSurfStoreRedMultOxford =
{1.0, 0.4, 0.2, 0.15, 0.12, 0.1};
this.maxSurfStoreRedMultOxford = new Table(
xFPLOxford, yMaxSurfStoreRedMultOxford);

double[] xFPLPerth = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
xFPLPerth[0] = this.FPLPerth;
double[] yMaxSurfStoreRedMultPerth =
{1.0, 0.4, 0.2, 0.15, 0.12, 0.1};
this.maxSurfStoreRedMultPerth = new Table(
xFPLPerth, yMaxSurfStoreRedMultPerth);

}


/**
 *  Sets the maxSurfStoreReduction attribute of the HydModel object
 *
 * @param  FPLMiddlesex  The new maxSurfStoreReduction value
 * @param  FPLOxford     The new maxSurfStoreReduction value
 * @param  FPLPerth      The new maxSurfStoreReduction value
 */
public void setMaxSurfStoreReduction(double FPLMiddlesex, double FPLOxford,
double FPLPerth) {
this.FPLMiddlesex = FPLMiddlesex;
this.FPLOxford = FPLOxford;
this.FPLPerth = FPLPerth;

// this is where the max surface store values get adjusted
// based on the Fraction of Paved Land (FPL) in each county

// Perth County
this.sb1.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb3.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb4.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb5.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb7.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb8.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb9.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb10.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb11.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb12.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));
this.sb13.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultPerth.lookup(this.FPLPerth));

// Oxford county
this.sb18.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb19.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb20.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb21.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb22.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb23.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb24.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb25.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb26.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));
this.sb27.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultOxford.lookup(this.FPLOxford));

// Middlesex County
this.sb14.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb15.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb16.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
```

```
this.sb17.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb28.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb29.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb30.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb31.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb32.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb33.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));
this.sb34.setMaxSurfStoreReductionCoeff(this.maxSurfStoreRedMultMiddlesex.lookup(this.FPLMiddlesex));

// this is so that SubBasin objects communicate the above to their
// own objects
updateParams();
}



/**
 *  Gets the maxSurfStoreReduction attribute of the HydModel object
 *
 * @return     The maxSurfStoreReduction value
 */
public double[] getMaxSurfStoreReduction() {
double[] MSSValues = new double[32];

MSSValues[0] = this.sb1.getMaxSurfStoreReduction();
MSSValues[1] = this.sb3.getMaxSurfStoreReduction();
MSSValues[2] = this.sb4.getMaxSurfStoreReduction();
MSSValues[3] = this.sb5.getMaxSurfStoreReduction();
MSSValues[4] = this.sb7.getMaxSurfStoreReduction();
MSSValues[5] = this.sb8.getMaxSurfStoreReduction();
MSSValues[6] = this.sb9.getMaxSurfStoreReduction();
MSSValues[7] = this.sb10.getMaxSurfStoreReduction();
MSSValues[8] = this.sb11.getMaxSurfStoreReduction();
MSSValues[9] = this.sb12.getMaxSurfStoreReduction();
MSSValues[10] = this.sb13.getMaxSurfStoreReduction();
MSSValues[11] = this.sb14.getMaxSurfStoreReduction();
MSSValues[12] = this.sb15.getMaxSurfStoreReduction();
MSSValues[13] = this.sb16.getMaxSurfStoreReduction();
MSSValues[14] = this.sb17.getMaxSurfStoreReduction();
MSSValues[15] = this.sb18.getMaxSurfStoreReduction();
MSSValues[16] = this.sb19.getMaxSurfStoreReduction();
MSSValues[17] = this.sb20.getMaxSurfStoreReduction();
MSSValues[18] = this.sb21.getMaxSurfStoreReduction();
MSSValues[19] = this.sb22.getMaxSurfStoreReduction();
MSSValues[20] = this.sb23.getMaxSurfStoreReduction();
MSSValues[21] = this.sb24.getMaxSurfStoreReduction();
MSSValues[22] = this.sb25.getMaxSurfStoreReduction();
MSSValues[23] = this.sb26.getMaxSurfStoreReduction();
MSSValues[24] = this.sb27.getMaxSurfStoreReduction();
MSSValues[25] = this.sb28.getMaxSurfStoreReduction();
MSSValues[26] = this.sb29.getMaxSurfStoreReduction();
MSSValues[27] = this.sb30.getMaxSurfStoreReduction();
MSSValues[28] = this.sb31.getMaxSurfStoreReduction();
MSSValues[29] = this.sb32.getMaxSurfStoreReduction();
MSSValues[30] = this.sb33.getMaxSurfStoreReduction();
MSSValues[31] = this.sb34.getMaxSurfStoreReduction();

return MSSValues;
}



/**
 *  Sets the maxSoilInfilTables attribute of the HydModel object
 *
```

```
 * @param  FVLMiddlesex  The new maxSoilInfilTables value
 * @param  FVLOxford     The new maxSoilInfilTables value
 * @param  FVLPerth      The new maxSoilInfilTables value
 */
public void setMaxSoilInfilReductionTables(double FVLMiddlesex, double FVLOxford,
double FVLPerth) {

this.FVLMiddlesex = FVLMiddlesex;
this.FVLOxford = FVLOxford;
this.FVLPerth = FVLPerth;

// there are three tables for this, one for each county
double[] xFVLM = {0.0, 0.2, 0.4, 0.6, 0.7, 1.0, 1.0};
double[] yInfilM = {0.1, 0.12, 0.15, 0.3, 0.5, 1.0, 1.5};
xFVLM[5] = this.FVLMiddlesex;
this.infilTableMiddlesex = new Table(xFVLM, yInfilM);

double[] xFVLO = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.0};
double[] yInfilO = {0.1, 0.12, 0.15, 0.2, 0.35, 1.0, 1.2};
xFVLO[5] = this.FVLOxford;
this.infilTableOxford = new Table(xFVLO, yInfilO);

double[] xFVLP = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.0};
double[] yInfilP = {0.1, 0.12, 0.15, 0.2, 0.35, 1.0, 1.2};
xFVLP[5] = this.FVLPerth;
this.infilTablePerth = new Table(xFVLP, yInfilP);

}


/**
 *  Sets the maxSoilInfil attribute of the HydModel object
 *
 * @param  FVLMiddlesex  The new maxSoilInfil value
 * @param  FVLOxford     The new maxSoilInfil value
 * @param  FVLPerth      The new maxSoilInfil value
 */
public void setMaxSoilInfilReduction(double FVLMiddlesex, double FVLOxford,
double FVLPerth) {

this.FVLMiddlesex = FVLMiddlesex;
this.FVLOxford = FVLOxford;
this.FVLPerth = FVLPerth;

// Perth County
this.sb1.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb3.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb4.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb5.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb7.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb8.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb9.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb10.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb11.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb12.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));
this.sb13.setMaxSoilInfilReductionCoeff(this.infilTablePerth.lookup(this.FVLPerth));

// Oxford county
this.sb18.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb19.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb20.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb21.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb22.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb23.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
```

```
this.sb24.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb25.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb26.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));
this.sb27.setMaxSoilInfilReductionCoeff(this.infilTableOxford.lookup(this.FVLOxford));

// Middlesex County
this.sb14.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb15.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb16.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb17.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb28.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb29.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb30.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb31.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb32.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb33.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb34.setMaxSoilInfilReductionCoeff(this.infilTableMiddlesex.lookup(this.FVLMiddlesex));

// this is so that SubBasin objects communicate the above to their
// own objects
updateParams();

}


/**
 *  Gets the maxSoilInfilReduction attribute of the HydModel object
 *
 * @return     The maxSoilInfilReduction value
 */
public double[] getMaxSoilInfilReduction() {
double[] MSIValues = new double[32];

MSIValues[0] = this.sb1.getMaxSoilInfilReduction();
MSIValues[1] = this.sb3.getMaxSoilInfilReduction();
MSIValues[2] = this.sb4.getMaxSoilInfilReduction();
MSIValues[3] = this.sb5.getMaxSoilInfilReduction();
MSIValues[4] = this.sb7.getMaxSoilInfilReduction();
MSIValues[5] = this.sb8.getMaxSoilInfilReduction();
MSIValues[6] = this.sb9.getMaxSoilInfilReduction();
MSIValues[7] = this.sb10.getMaxSoilInfilReduction();
MSIValues[8] = this.sb11.getMaxSoilInfilReduction();
MSIValues[9] = this.sb12.getMaxSoilInfilReduction();
MSIValues[10] = this.sb13.getMaxSoilInfilReduction();
MSIValues[11] = this.sb14.getMaxSoilInfilReduction();
MSIValues[12] = this.sb15.getMaxSoilInfilReduction();
MSIValues[13] = this.sb16.getMaxSoilInfilReduction();
MSIValues[14] = this.sb17.getMaxSoilInfilReduction();
MSIValues[15] = this.sb18.getMaxSoilInfilReduction();
MSIValues[16] = this.sb19.getMaxSoilInfilReduction();
MSIValues[17] = this.sb20.getMaxSoilInfilReduction();
MSIValues[18] = this.sb21.getMaxSoilInfilReduction();
MSIValues[19] = this.sb22.getMaxSoilInfilReduction();
MSIValues[20] = this.sb23.getMaxSoilInfilReduction();
MSIValues[21] = this.sb24.getMaxSoilInfilReduction();
MSIValues[22] = this.sb25.getMaxSoilInfilReduction();
MSIValues[23] = this.sb26.getMaxSoilInfilReduction();
MSIValues[24] = this.sb27.getMaxSoilInfilReduction();
MSIValues[25] = this.sb28.getMaxSoilInfilReduction();
MSIValues[26] = this.sb29.getMaxSoilInfilReduction();
MSIValues[27] = this.sb30.getMaxSoilInfilReduction();
MSIValues[28] = this.sb31.getMaxSoilInfilReduction();
MSIValues[29] = this.sb32.getMaxSoilInfilReduction();
MSIValues[30] = this.sb33.getMaxSoilInfilReduction();
```

```
MSIValues[31] = this.sb34.getMaxSoilInfilReduction();

return MSIValues;
}


/**
 *  Sets the TimeOfConcentrationTables attribute of the HydModel object.
 *
 * @param  FPLMiddlesex  Fraction of paved land in Middlesex, in [-]
 * @param  FPLOxford     Fraction of paved land in Oxford, in [-]
 * @param  FPLPerth      Fraction of paved land in Perth, in [-]
 */
public void setTimeOfConcentrationTables(double FPLMiddlesex, double FPLOxford,
double FPLPerth) {
this.FPLMiddlesex = FPLMiddlesex;
this.FPLOxford = FPLOxford;
this.FPLPerth = FPLPerth;

// if a SubBasin has a Tc = 6 hrs, nothing is done about it; this
// is because the HydModel must operate on 6 hr time step, and
// nothing less; the tables for each county are the same in this
// code, but this leaves the possibility that a model user may
// want to change it;

// all tables take in an initial fraction of paved land from the
// SysModel so that at initial time step, all time of concentrations
// are as they were originally set in the HydModel.  This is needed
// because a sudden jump between time zero and time one is not wanted

// TABLES FOR PERTH COUNTY
double[] xFRLPerth24 = {0.0, 0.03, 0.1, 0.2, 0.3, 0.4,
0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
xFRLPerth24[1] = this.FPLPerth;
double[] yTCTablePerth24hr = {24, 24, 23.3, 21.9,
20, 16.42, 12.10, 9.3, 7.5, 6.3, 6.0, 6.0};
this.TCTablePerth24hr = new Table(xFRLPerth24, yTCTablePerth24hr);

double[] xFPLPerth18 = xFRLPerth24;
xFPLPerth18[1] = this.FPLPerth;
double[] yTCTablePerth18hr = {18, 18, 17.75, 17.11, 15.58, 13.53,
10.47, 8.68, 7.32, 6.58, 6.21, 6.0};
this.TCTablePerth18hr = new Table(xFPLPerth18, yTCTablePerth18hr);

double[] xFPLPerth12 = xFRLPerth24;
xFPLPerth12[1] = this.FPLPerth;
double[] yTCTablePerth12hr = {12, 12, 11.84, 11.58, 10.8, 9.6,
8.2, 7.2, 6.48, 6.2, 6.1, 6.0};
this.TCTablePerth12hr = new Table(xFPLPerth12, yTCTablePerth12hr);

// TABLES FOR OXFORD COUNTY
double[] xFPLOxford24 = {0.0, 0.03, 0.1, 0.2, 0.3, 0.4,
0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
xFPLOxford24[1] = this.FPLOxford;
double[] yTCTableOxford24hr = {24, 24, 23.3, 21.9,
20, 16.42, 12.10, 9.3, 7.5, 6.3, 6.0, 6.0};
this.TCTableOxford24hr = new Table(xFPLOxford24, yTCTableOxford24hr);

double[] xFPLOxford18 = xFPLOxford24;
xFPLOxford18[1] = this.FPLOxford;
double[] yTCTableOxford18hr = {18, 18, 17.75, 17.11, 15.58, 13.53,
10.47, 8.68, 7.32, 6.58, 6.21, 6.0};
this.TCTableOxford18hr = new Table(xFPLOxford18, yTCTableOxford18hr);
```

```
double[] xFPLOxford12 = xFPLOxford24;
xFPLOxford12[1] = this.FPLOxford;
double[] yTCTableOxford12hr = {12, 12, 11.84, 11.58, 10.8, 9.6,
8.2, 7.2, 6.48, 6.2, 6.1, 6.0};
this.TCTableOxford12hr = new Table(xFPLOxford12, yTCTableOxford12hr);

// TABLES FOR MIDDLESEX COUNTY
double[] xFPLMiddlesex24 = {0.0, 0.1, 0.2, 0.3, 0.4,
0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
xFPLMiddlesex24[1] = this.FPLMiddlesex;
double[] yTCTableMiddlesex24hr = {25, 24, 22,
19.6, 16.42, 12.10, 9.3, 7.5, 6.3, 6.0, 6.0};
this.TCTableMiddlesex24hr = new Table(xFPLMiddlesex24, yTCTableMiddlesex24hr);

double[] xFPLMiddlesex18 = xFPLMiddlesex24;
xFPLMiddlesex18[1] = this.FPLMiddlesex;
double[] yTCTableMiddlesex18hr = {18.5, 18.0, 16.9, 15.58, 13.53,
10.47, 8.68, 7.32, 6.58, 6.21, 6.0};
this.TCTableMiddlesex18hr = new Table(xFPLMiddlesex18, yTCTableMiddlesex18hr);

double[] xFPLMiddlesex12 = xFPLMiddlesex24;
xFPLMiddlesex12[1] = this.FPLMiddlesex;
double[] yTCTableMiddlesex12hr = {12.3, 12.0, 11.58, 10.8, 9.6,
8.2, 7.2, 6.48, 6.2, 6.1, 6.0};
this.TCTableMiddlesex12hr = new Table(xFPLMiddlesex12, yTCTableMiddlesex12hr);
}


/**
 *  Sets the timeOfConcentration attribute of the HydModel object
 *
 * @param  FPLMiddlesex  The new timeOfConcentration value
 * @param  FPLOxford     The new timeOfConcentration value
 * @param  FPLPerth      The new timeOfConcentration value
 */
public void setTimeOfConcentration(double FPLMiddlesex, double FPLOxford,
double FPLPerth) {
this.FPLMiddlesex = FPLMiddlesex;
this.FPLOxford = FPLOxford;
this.FPLPerth = FPLPerth;

// this is where the Tc values get adjusted based on the Fraction
// of Paved Land (FPL) in each county

// Perth County; it didn't update sb3, because sb3 has Tc = 6hrs, and this
// never changes
this.sb1.setTimeOfConcentration(this.TCTablePerth24hr.lookup(this.FPLPerth));
this.sb4.setTimeOfConcentration(this.TCTablePerth24hr.lookup(this.FPLPerth));
this.sb5.setTimeOfConcentration(this.TCTablePerth18hr.lookup(this.FPLPerth));
this.sb7.setTimeOfConcentration(this.TCTablePerth12hr.lookup(this.FPLPerth));
this.sb8.setTimeOfConcentration(this.TCTablePerth18hr.lookup(this.FPLPerth));
this.sb9.setTimeOfConcentration(this.TCTablePerth12hr.lookup(this.FPLPerth));
this.sb10.setTimeOfConcentration(this.TCTablePerth24hr.lookup(this.FPLPerth));
this.sb11.setTimeOfConcentration(this.TCTablePerth12hr.lookup(this.FPLPerth));
this.sb12.setTimeOfConcentration(this.TCTablePerth18hr.lookup(this.FPLPerth));
this.sb13.setTimeOfConcentration(this.TCTablePerth24hr.lookup(this.FPLPerth));

// Oxford county; it didn't update sb19, because sb19 has Tc = 6hrs, and this
// never changes
this.sb18.setTimeOfConcentration(this.TCTableOxford18hr.lookup(this.FPLOxford));
this.sb20.setTimeOfConcentration(this.TCTableOxford18hr.lookup(this.FPLOxford));
this.sb21.setTimeOfConcentration(this.TCTableOxford18hr.lookup(this.FPLOxford));
this.sb22.setTimeOfConcentration(this.TCTableOxford12hr.lookup(this.FPLOxford));
this.sb23.setTimeOfConcentration(this.TCTableOxford24hr.lookup(this.FPLOxford));
```

```
this.sb24.setTimeOfConcentration(this.TCTableOxford12hr.lookup(this.FPLOxford));
this.sb25.setTimeOfConcentration(this.TCTableOxford24hr.lookup(this.FPLOxford));
this.sb26.setTimeOfConcentration(this.TCTableOxford12hr.lookup(this.FPLOxford));
this.sb27.setTimeOfConcentration(this.TCTableOxford18hr.lookup(this.FPLOxford));

// Middlesex County; it didn't update sb29, 30, 31, 33 because they
// all have Tc = 6hrs, and this can't change
this.sb14.setTimeOfConcentration(this.TCTableMiddlesex18hr.lookup(this.FPLMiddlesex));
this.sb15.setTimeOfConcentration(this.TCTableMiddlesex12hr.lookup(this.FPLMiddlesex));
this.sb16.setTimeOfConcentration(this.TCTableMiddlesex12hr.lookup(this.FPLMiddlesex));
this.sb17.setTimeOfConcentration(this.TCTableMiddlesex18hr.lookup(this.FPLMiddlesex));
this.sb28.setTimeOfConcentration(this.TCTableMiddlesex12hr.lookup(this.FPLMiddlesex));
this.sb32.setTimeOfConcentration(this.TCTableMiddlesex18hr.lookup(this.FPLMiddlesex));
this.sb34.setTimeOfConcentration(this.TCTableMiddlesex24hr.lookup(this.FPLMiddlesex));

// this is so that SubBasin objects communicate the above to their
// own objects
updateParams();
}


/**
 *  Gets the timeOfConcentration attribute of the HydModel object
 *
 * @return    The timeOfConcentration value, in [hrs] for all SubBasin objects.
 */
public double[] getTimeOfConcentration() {
double[] TCValues = new double[32];

TCValues[0] = this.sb1.getTimeOfConcentration();
TCValues[1] = this.sb3.getTimeOfConcentration();
TCValues[2] = this.sb4.getTimeOfConcentration();
TCValues[3] = this.sb5.getTimeOfConcentration();
TCValues[4] = this.sb7.getTimeOfConcentration();
TCValues[5] = this.sb8.getTimeOfConcentration();
TCValues[6] = this.sb9.getTimeOfConcentration();
TCValues[7] = this.sb10.getTimeOfConcentration();
TCValues[8] = this.sb11.getTimeOfConcentration();
TCValues[9] = this.sb12.getTimeOfConcentration();
TCValues[10] = this.sb13.getTimeOfConcentration();
TCValues[11] = this.sb14.getTimeOfConcentration();
TCValues[12] = this.sb15.getTimeOfConcentration();
TCValues[13] = this.sb16.getTimeOfConcentration();
TCValues[14] = this.sb17.getTimeOfConcentration();
TCValues[15] = this.sb18.getTimeOfConcentration();
TCValues[16] = this.sb19.getTimeOfConcentration();
TCValues[17] = this.sb20.getTimeOfConcentration();
TCValues[18] = this.sb21.getTimeOfConcentration();
TCValues[19] = this.sb22.getTimeOfConcentration();
TCValues[20] = this.sb23.getTimeOfConcentration();
TCValues[21] = this.sb24.getTimeOfConcentration();
TCValues[22] = this.sb25.getTimeOfConcentration();
TCValues[23] = this.sb26.getTimeOfConcentration();
TCValues[24] = this.sb27.getTimeOfConcentration();
TCValues[25] = this.sb28.getTimeOfConcentration();
TCValues[26] = this.sb29.getTimeOfConcentration();
TCValues[27] = this.sb30.getTimeOfConcentration();
TCValues[28] = this.sb31.getTimeOfConcentration();
TCValues[29] = this.sb32.getTimeOfConcentration();
TCValues[30] = this.sb33.getTimeOfConcentration();
TCValues[31] = this.sb34.getTimeOfConcentration();

return TCValues;
}
```

```java
/**
 *  Sets the percentImperviousnessTables attribute of the HydModel object
 *
 * @param  FPLMiddlesex  The new percentImperviousnessTables value
 * @param  FPLOxford     The new percentImperviousnessTables value
 * @param  FPLPerth      The new percentImperviousnessTables value
 */
public void setPercentImperviousnessTables(double FPLMiddlesex,
double FPLOxford, double FPLPerth) {
this.FPLMiddlesex = FPLMiddlesex;
this.FPLOxford = FPLOxford;
this.FPLPerth = FPLPerth;

// table for Middlesex County when the initial percent of
// imperviousness is zero
double[] mx0 = {0.0, 0.1, 1.0};
mx0[1] = this.FPLMiddlesex;
double[] my0 = {0.0, 0.0, 100.0};
this.ImpTableMiddlesex0 = new Table(mx0, my0);

// table for Middlesex County when the initial percent of
// imperviousness is 5
double[] mx5 = {0.0, 0.1, 1.0};
mx5[1] = this.FPLMiddlesex;
double[] my5 = {0.0, 5.0, 100.0};
this.ImpTableMiddlesex5 = new Table(mx5, my5);

// table for Middlesex County when the initial percent of
// imperviousness is 40
double[] mx40 = {0.0, 0.1, 1.0};
mx40[1] = this.FPLMiddlesex;
double[] my40 = {0.0, 40.0, 100.0};
this.ImpTableMiddlesex40 = new Table(mx40, my40);

// table for Middlesex County when the initial percent of
// imperviousness is 30
double[] mx30 = {0.0, 0.1, 1.0};
mx30[1] = this.FPLMiddlesex;
double[] my30 = {0.0, 30.0, 100.0};
this.ImpTableMiddlesex30 = new Table(mx30, my30);

// table for Perth County when the initial percent of
// imperviousness is 0
double[] px0 = {0.0, 0.01, 1.0};
px0[1] = this.FPLPerth;
double[] py0 = {0.0, 0.0, 100.0};
this.ImpTablePerth0 = new Table(px0, py0);

// table for Perth County when the initial percent of
// imperviousness is 2
double[] px2 = {0.0, 0.01, 1.0};
px2[1] = this.FPLPerth;
double[] py2 = {0.0, 2.0, 100.0};
this.ImpTablePerth2 = new Table(px2, py2);

// table for Oxford County when the initial percent of
// imperviousness is 0
double[] ox0 = {0.0, 0.01, 1.0};
ox0[1] = this.FPLOxford;
double[] oy0 = {0.0, 0.0, 100.0};
this.ImpTableOxford0 = new Table(ox0, oy0);
}
```

```
/**
 *  Sets the percentImperviousness attribute of the HydModel object
 *
 * @param  FPLMiddlesex  The new percentImperviousness value
 * @param  FPLOxford     The new percentImperviousness value
 * @param  FPLPerth      The new percentImperviousness value
 */
public void setPercentImperviousness(double FPLMiddlesex,
double FPLOxford, double FPLPerth) {
this.FPLMiddlesex = FPLMiddlesex;
this.FPLOxford = FPLOxford;
this.FPLPerth = FPLPerth;

// this is where the percent of imperviousness values get
// adjusted based on the Fraction of Paved Land (FPL) in each county

// Perth County
this.sb1.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb3.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb4.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb5.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb7.setImperviousness(this.ImpTablePerth2.lookup(this.FPLPerth));
this.sb8.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb9.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb10.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb11.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb12.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));
this.sb13.setImperviousness(this.ImpTablePerth0.lookup(this.FPLPerth));

// Oxford county
this.sb18.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb19.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb20.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb21.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb22.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb23.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb24.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb25.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb26.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));
this.sb27.setImperviousness(this.ImpTableOxford0.lookup(this.FPLOxford));

// Middlesex County
this.sb14.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb15.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb16.setImperviousness(this.ImpTableMiddlesex5.lookup(this.FPLMiddlesex));
this.sb17.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb28.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb29.setImperviousness(this.ImpTableMiddlesex40.lookup(this.FPLMiddlesex));
this.sb30.setImperviousness(this.ImpTableMiddlesex30.lookup(this.FPLMiddlesex));
this.sb31.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb32.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb33.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));
this.sb34.setImperviousness(this.ImpTableMiddlesex0.lookup(this.FPLMiddlesex));

// this is so that SubBasin objects communicate the above to their
// own objects
updateParams();
}


/**
 *  Gets the imperviousness attribute of the HydModel object
```

```
 *
 * @return    The imperviousness value
 */
public double[] getImperviousness() {
double[] ImpValues = new double[32];

ImpValues[0] = this.sb1.getImperviousness();
ImpValues[1] = this.sb3.getImperviousness();
ImpValues[2] = this.sb4.getImperviousness();
ImpValues[3] = this.sb5.getImperviousness();
ImpValues[4] = this.sb7.getImperviousness();
ImpValues[5] = this.sb8.getImperviousness();
ImpValues[6] = this.sb9.getImperviousness();
ImpValues[7] = this.sb10.getImperviousness();
ImpValues[8] = this.sb11.getImperviousness();
ImpValues[9] = this.sb12.getImperviousness();
ImpValues[10] = this.sb13.getImperviousness();
ImpValues[11] = this.sb14.getImperviousness();
ImpValues[12] = this.sb15.getImperviousness();
ImpValues[13] = this.sb16.getImperviousness();
ImpValues[14] = this.sb17.getImperviousness();
ImpValues[15] = this.sb18.getImperviousness();
ImpValues[16] = this.sb19.getImperviousness();
ImpValues[17] = this.sb20.getImperviousness();
ImpValues[18] = this.sb21.getImperviousness();
ImpValues[19] = this.sb22.getImperviousness();
ImpValues[20] = this.sb23.getImperviousness();
ImpValues[21] = this.sb24.getImperviousness();
ImpValues[22] = this.sb25.getImperviousness();
ImpValues[23] = this.sb26.getImperviousness();
ImpValues[24] = this.sb27.getImperviousness();
ImpValues[25] = this.sb28.getImperviousness();
ImpValues[26] = this.sb29.getImperviousness();
ImpValues[27] = this.sb30.getImperviousness();
ImpValues[28] = this.sb31.getImperviousness();
ImpValues[29] = this.sb32.getImperviousness();
ImpValues[30] = this.sb33.getImperviousness();
ImpValues[31] = this.sb34.getImperviousness();

return ImpValues;
}


/**
 *  Sets the PETTables attribute of the HydModel object
 *
 * @param  FVLMiddlesex  The new pETTables value
 * @param  FVLOxford     The new pETTables value
 * @param  FVLPerth      The new pETTables value
 */
public void setPETTables(double FVLMiddlesex, double FVLOxford,
double FVLPerth) {

this.FVLMiddlesex = FVLMiddlesex;
this.FVLOxford = FVLOxford;
this.FVLPerth = FVLPerth;

// there three tables for this, one for each county
double[] xFVLM = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
double[] yPETM = {0.2, 0.38, 0.65, 0.9, 1.0, 1.0};
xFVLM[4] = this.FVLMiddlesex;
this.PETTableMiddlesex = new Table(xFVLM, yPETM);

double[] xFVLO = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
```

```
double[] yPETO = {0.2, 0.38, 0.65, 0.9, 1.0, 1.0};
xFVLO[4] = this.FVLOxford;
this.PETTableOxford = new Table(xFVLO, yPETO);

double[] xFVLP = {0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
double[] yPETP = {0.2, 0.38, 0.65, 0.9, 1.0, 1.0};
xFVLP[4] = this.FVLPerth;
this.PETTablePerth = new Table(xFVLP, yPETP);

}


/**
 *  Sets the PET attribute of the HydModel object
 *
 * @param  FVLMiddlesex  The new pET value
 * @param  FVLOxford     The new pET value
 * @param  FVLPerth      The new pET value
 */
public void setPET(double FVLMiddlesex, double FVLOxford,
double FVLPerth) {

this.FVLMiddlesex = FVLMiddlesex;
this.FVLOxford = FVLOxford;
this.FVLPerth = FVLPerth;

// Perth County
this.sb1.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb3.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb4.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb5.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb7.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb8.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb9.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb10.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb11.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb12.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));
this.sb13.setPETReductionCoeff(this.PETTablePerth.lookup(this.FVLPerth));

// Oxford county
this.sb18.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb19.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb20.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb21.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb22.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb23.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb24.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb25.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb26.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));
this.sb27.setPETReductionCoeff(this.PETTableOxford.lookup(this.FVLOxford));

// Middlesex County
this.sb14.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb15.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb16.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb17.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb28.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb29.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb30.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb31.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb32.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb33.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
this.sb34.setPETReductionCoeff(this.PETTableMiddlesex.lookup(this.FVLMiddlesex));
```

```
// this is so that SubBasin objects communicate the above to their
// own objects
updateParams();

}


/**
 *  Gets the PETReductionCoeff attribute of the HydModel object
 *
 * @return     The PETReductionCoeff value
 */
public double[] getPETReductionCoeff() {
double[] PETValues = new double[32];

PETValues[0] = this.sb1.getPETReductionCoeff();
PETValues[1] = this.sb3.getPETReductionCoeff();
PETValues[2] = this.sb4.getPETReductionCoeff();
PETValues[3] = this.sb5.getPETReductionCoeff();
PETValues[4] = this.sb7.getPETReductionCoeff();
PETValues[5] = this.sb8.getPETReductionCoeff();
PETValues[6] = this.sb9.getPETReductionCoeff();
PETValues[7] = this.sb10.getPETReductionCoeff();
PETValues[8] = this.sb11.getPETReductionCoeff();
PETValues[9] = this.sb12.getPETReductionCoeff();
PETValues[10] = this.sb13.getPETReductionCoeff();
PETValues[11] = this.sb14.getPETReductionCoeff();
PETValues[12] = this.sb15.getPETReductionCoeff();
PETValues[13] = this.sb16.getPETReductionCoeff();
PETValues[14] = this.sb17.getPETReductionCoeff();
PETValues[15] = this.sb18.getPETReductionCoeff();
PETValues[16] = this.sb19.getPETReductionCoeff();
PETValues[17] = this.sb20.getPETReductionCoeff();
PETValues[18] = this.sb21.getPETReductionCoeff();
PETValues[19] = this.sb22.getPETReductionCoeff();
PETValues[20] = this.sb23.getPETReductionCoeff();
PETValues[21] = this.sb24.getPETReductionCoeff();
PETValues[22] = this.sb25.getPETReductionCoeff();
PETValues[23] = this.sb26.getPETReductionCoeff();
PETValues[24] = this.sb27.getPETReductionCoeff();
PETValues[25] = this.sb28.getPETReductionCoeff();
PETValues[26] = this.sb29.getPETReductionCoeff();
PETValues[27] = this.sb30.getPETReductionCoeff();
PETValues[28] = this.sb31.getPETReductionCoeff();
PETValues[29] = this.sb32.getPETReductionCoeff();
PETValues[30] = this.sb33.getPETReductionCoeff();
PETValues[31] = this.sb34.getPETReductionCoeff();

return PETValues;
}


//}}}

// PPT output methods
//{{{

/**
 *  Gets the jnByronPPT attribute of the HydModel object.  The method
 * computes precipitation based on the weighted average (by area).  This
 * is simply a method to aggregate ppt from a number of SubBasins.
 *
 * @return     The jnByronPPT value, in [mm]
 */
```

```
public double getJnByronPPT() {

// area draining to Byron stream gauge
// all subbasins except 31, 32, 33, 34
double byronArea = (this.sb1.getSubBasinArea() +
this.sb3.getSubBasinArea() +
this.sb4.getSubBasinArea() +
this.sb5.getSubBasinArea() +
this.sb7.getSubBasinArea() +
this.sb8.getSubBasinArea() +
this.sb9.getSubBasinArea() +
this.sb10.getSubBasinArea() +
this.sb11.getSubBasinArea() +
this.sb12.getSubBasinArea() +
this.sb13.getSubBasinArea() +
this.sb14.getSubBasinArea() +
this.sb15.getSubBasinArea() +
this.sb16.getSubBasinArea() +
this.sb17.getSubBasinArea() +
this.sb18.getSubBasinArea() +
this.sb19.getSubBasinArea() +
this.sb20.getSubBasinArea() +
this.sb21.getSubBasinArea() +
this.sb22.getSubBasinArea() +
this.sb23.getSubBasinArea() +
this.sb24.getSubBasinArea() +
this.sb25.getSubBasinArea() +
this.sb26.getSubBasinArea() +
this.sb27.getSubBasinArea() +
this.sb28.getSubBasinArea() +
this.sb29.getSubBasinArea() +
this.sb30.getSubBasinArea() );

return ((this.sb1.getSubBasinArea() / byronArea) * this.sb1PPTVal +
(this.sb3.getSubBasinArea() / byronArea) * this.sb3PPTVal +
(this.sb4.getSubBasinArea() / byronArea) * this.sb4PPTVal +
(this.sb5.getSubBasinArea() / byronArea) * this.sb5PPTVal +
(this.sb7.getSubBasinArea() / byronArea) * this.sb7PPTVal +
(this.sb8.getSubBasinArea() / byronArea) * this.sb8PPTVal +
(this.sb9.getSubBasinArea() / byronArea) * this.sb9PPTVal +
(this.sb10.getSubBasinArea() / byronArea) * this.sb10PPTVal +
(this.sb11.getSubBasinArea() / byronArea) * this.sb11PPTVal +
(this.sb12.getSubBasinArea() / byronArea) * this.sb12PPTVal +
(this.sb13.getSubBasinArea() / byronArea) * this.sb13PPTVal +
(this.sb14.getSubBasinArea() / byronArea) * this.sb14PPTVal +
(this.sb15.getSubBasinArea() / byronArea) * this.sb15PPTVal +
(this.sb16.getSubBasinArea() / byronArea) * this.sb16PPTVal +
(this.sb17.getSubBasinArea() / byronArea) * this.sb17PPTVal +
(this.sb18.getSubBasinArea() / byronArea) * this.sb18PPTVal +
(this.sb19.getSubBasinArea() / byronArea) * this.sb19PPTVal +
(this.sb20.getSubBasinArea() / byronArea) * this.sb20PPTVal +
(this.sb21.getSubBasinArea() / byronArea) * this.sb21PPTVal +
(this.sb22.getSubBasinArea() / byronArea) * this.sb22PPTVal +
(this.sb23.getSubBasinArea() / byronArea) * this.sb23PPTVal +
(this.sb24.getSubBasinArea() / byronArea) * this.sb24PPTVal +
(this.sb25.getSubBasinArea() / byronArea) * this.sb25PPTVal +
(this.sb26.getSubBasinArea() / byronArea) * this.sb26PPTVal +
(this.sb27.getSubBasinArea() / byronArea) * this.sb27PPTVal +
(this.sb28.getSubBasinArea() / byronArea) * this.sb28PPTVal +
(this.sb29.getSubBasinArea() / byronArea) * this.sb29PPTVal +
(this.sb30.getSubBasinArea() / byronArea) * this.sb30PPTVal );
}
```

```
/**
 * Gets the jnIngersollPPT attribute of the HydModel object.  The method
 * computes precipitation based on the weighted average (by area).  This
 * is simply a method to aggregate ppt from a number of SubBasins.
 *
 * @return    The jnIngersollPPT value, in [mm]
 */
public double getJnIngersollPPT() {

// area draining to Ingersoll stream gauge
// sb18, 19, 20, 21
double ingersollArea = (this.sb18.getSubBasinArea() +
this.sb19.getSubBasinArea() +
this.sb20.getSubBasinArea() +
this.sb21.getSubBasinArea() );

return ((this.sb18.getSubBasinArea() / ingersollArea) * this.sb18PPTVal +
(this.sb19.getSubBasinArea() / ingersollArea) * this.sb19PPTVal +
(this.sb20.getSubBasinArea() / ingersollArea) * this.sb20PPTVal +
(this.sb21.getSubBasinArea() / ingersollArea) * this.sb21PPTVal);
}


/**
 * Gets the jnStMarysPPT attribute of the HydModel object. The method
 * computes precipitation based on the weighted average (by area).  This
 * is simply a method to aggregate ppt from a number of SubBasins.
 *
 * @return    The jnStMarysPPT value, in [mm]
 */
public double getJnStMarysPPT() {

// area draining to St.Marys stream gauge; note this is a
// different area than one computed by getGWRecharge methods

// sb1, 3, 4, 5, 7, 8, 9, 10, 11
double stMarysArea = (this.sb1.getSubBasinArea() +
this.sb3.getSubBasinArea() +
this.sb4.getSubBasinArea() +
this.sb5.getSubBasinArea() +
this.sb7.getSubBasinArea() +
this.sb8.getSubBasinArea() +
this.sb9.getSubBasinArea() +
this.sb10.getSubBasinArea() +
this.sb11.getSubBasinArea() );

return ((this.sb1.getSubBasinArea() / stMarysArea) * this.sb1PPTVal +
(this.sb3.getSubBasinArea() / stMarysArea) * this.sb3PPTVal +
(this.sb4.getSubBasinArea() / stMarysArea) * this.sb4PPTVal +
(this.sb5.getSubBasinArea() / stMarysArea) * this.sb5PPTVal +
(this.sb7.getSubBasinArea() / stMarysArea) * this.sb7PPTVal +
(this.sb8.getSubBasinArea() / stMarysArea) * this.sb8PPTVal +
(this.sb9.getSubBasinArea() / stMarysArea) * this.sb9PPTVal +
(this.sb10.getSubBasinArea() / stMarysArea) * this.sb10PPTVal +
(this.sb11.getSubBasinArea() / stMarysArea) * this.sb11PPTVal);
}
//}}}

// GWRecharge output methods
//{{{
/**
 * Gets the GWRechargePerth attribute of the HydModel object
 *
 * @return    The GWRechargePerth value, in [m<SUP>3</SUP>/yr]
```

```java
 */
public double getGWRechargePerth() {
// sb1, 3, 4, 5, 7, 8, 9, 10, 11, 12, 13 are assumed to be
// in Perth county

double totalSubBasinsArea = (this.sb1.getSubBasinArea() +
this.sb3.getSubBasinArea() +
this.sb4.getSubBasinArea() +
this.sb5.getSubBasinArea() +
this.sb7.getSubBasinArea() +
this.sb8.getSubBasinArea() +
this.sb9.getSubBasinArea() +
this.sb10.getSubBasinArea() +
this.sb11.getSubBasinArea() +
this.sb12.getSubBasinArea() +
this.sb13.getSubBasinArea());

// returns a weighted averaged value in [cms], based on the
// area of the SubBasin
return ((this.sb1.getSubBasinArea() *
this.sb1.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb3.getSubBasinArea() *
this.sb3.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb4.getSubBasinArea() *
this.sb4.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb5.getSubBasinArea() *
this.sb5.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb7.getSubBasinArea() *
this.sb7.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb8.getSubBasinArea() *
this.sb8.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb9.getSubBasinArea() *
this.sb9.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb10.getSubBasinArea() *
this.sb10.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb11.getSubBasinArea() *
this.sb11.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb12.getSubBasinArea() *
this.sb12.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb13.getSubBasinArea() *
this.sb13.getActGW2Perc() /
totalSubBasinsArea));
}



/**
 *  Gets the GWRechargeOxford attribute of the HydModel object
 *
 * @return    The GWRechargeOxford value, in [m<SUP>3</SUP>/yr]
 */
public double getGWRechargeOxford() {
// sb18,19,20,21,22,23,24,25,26,27 are assumed to be
// in Oxford county
```

```
double totalSubBasinsArea = (this.sb18.getSubBasinArea() +
this.sb19.getSubBasinArea() +
this.sb20.getSubBasinArea() +
this.sb21.getSubBasinArea() +
this.sb22.getSubBasinArea() +
this.sb23.getSubBasinArea() +
this.sb24.getSubBasinArea() +
this.sb25.getSubBasinArea() +
this.sb26.getSubBasinArea() +
this.sb27.getSubBasinArea());

// returns a weighted averaged value in [m3/yr], based on the
// area of the SubBasin
return ((this.sb18.getSubBasinArea() *
this.sb18.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb19.getSubBasinArea() *
this.sb19.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb20.getSubBasinArea() *
this.sb20.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb21.getSubBasinArea() *
this.sb21.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb22.getSubBasinArea() *
this.sb22.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb23.getSubBasinArea() *
this.sb23.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb24.getSubBasinArea() *
this.sb24.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb25.getSubBasinArea() *
this.sb25.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb26.getSubBasinArea() *
this.sb26.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb27.getSubBasinArea() *
this.sb27.getActGW2Perc() /
totalSubBasinsArea));
}




/**
 *  Gets the GWRechargeMiddlesex attribute of the HydModel object
 *
 * @return    The GWRechargeMiddlesex value, in [m<SUP>3</SUP>/yr]
 */
public double getGWRechargeMiddlesex() {
// sb14,15,16,17,28,29,30,31,32,33,34 are assumed to be
// in Middlesex county

double totalSubBasinsArea = (this.sb14.getSubBasinArea() +
this.sb15.getSubBasinArea() +
this.sb16.getSubBasinArea() +
this.sb17.getSubBasinArea() +
this.sb28.getSubBasinArea() +
this.sb29.getSubBasinArea() +
this.sb30.getSubBasinArea() +
```

```
this.sb31.getSubBasinArea() +
this.sb32.getSubBasinArea() +
this.sb33.getSubBasinArea() +
this.sb34.getSubBasinArea());

// returns a weighted averaged value in [m3/yr], based on the
// area of the SubBasin
return ((this.sb14.getSubBasinArea() *
this.sb14.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb15.getSubBasinArea() *
this.sb15.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb16.getSubBasinArea() *
this.sb16.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb17.getSubBasinArea() *
this.sb17.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb28.getSubBasinArea() *
this.sb28.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb29.getSubBasinArea() *
this.sb29.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb30.getSubBasinArea() *
this.sb30.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb31.getSubBasinArea() *
this.sb31.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb32.getSubBasinArea() *
this.sb32.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb33.getSubBasinArea() *
this.sb33.getActGW2Perc() /
totalSubBasinsArea) +
(this.sb34.getSubBasinArea() *
this.sb34.getActGW2Perc() /
totalSubBasinsArea));
}


//}}}

// junction output methods
//{{{
/**
 *  Gets the jnMitchellSG attribute of the HydModel object
 *
 * @return    The jnMitchellSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnMitchellSG() {
return this.jnMitchellSG[0];
}

/**
 *  Gets the JnMitchellSGBaseflow attribute of the HydModel object
 *
 * @return    The JnMitchellSGBaseflow value, in [m<SUP>3</SUP>/s]
 */
public double getJnMitchellSGBaseflow(){
return this.sb1.getBaseflow();
}
```

```java
/**
 *  Gets the jnAvonSG attribute of the HydModel object
 *
 * @return    The jnAvonSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnAvonSG() {
return this.sb7.getTotalFlow();
}


/**
 *  Gets the jnStMarysSG attribute of the HydModel object
 *
 * @return    The jnStMarysSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnStMarysSG() {
return this.jnStMarysSG[0];
}


/**
 *  Gets the jnStMarysSGDamage attribute of the HydModel object; from
 * flow [cms] it computes damage in [$1000/s], and then it converts this to
 * [$1000/yr].
 *
 * @return    The jnStMarysSGDamage value, in [$1000/yr]
 */
public double getJnStMarysSGDamage() {
return (this.FlowDamageStMarysSGTable.lookup(
this.jnStMarysSG[0]) * 3600 * 24 *
this.currentDate.getDaysInMonth() * 12);
}


/**
 *  Gets the jnPloverMillsSG attribute of the HydModel object
 *
 * @return    The jnPloverMillsSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnPloverMillsSG() {
return this.jnPloverMillsSG[0];
}


/**
 *  Gets the jnMedwaySG attribute of the HydModel object
 *
 * @return    The jnMedwaySG value, in [m<SUP>3</SUP>/s]
 */
public double getJnMedwaySG() {
return this.sb17.getTotalFlow();
}


/**
 *  Gets the jnCedarSG attribute of the HydModel object
 *
 * @return    The jnCedarSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnCedarSG() {
return this.sb20.getTotalFlow();
```

```
}


/**
 *  Gets the jnIngersollSG attribute of the HydModel object
 *
 * @return    The jnIngersollSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnIngersollSG() {
return this.jnIngersollSG[0];
}


/**
 *  Gets the jnIngersollSGDamage attribute of the HydModel object; from
 * flow [cms] it computes damage in [$1000/s], and then it converts this to
 * [$1000/yr].
 *
 * @return    The jnIngersollSGDamage value, in [$1000/yr]
 */
public double getJnIngersollSGDamage() {
return (this.FlowDamageIngersollSGTable.lookup(
this.jnIngersollSG[0]) * 3600 * 24 *
this.currentDate.getDaysInMonth() * 12);
}


/**
 *  Gets the jnReynoldsSG attribute of the HydModel object
 *
 * @return    The jnReynoldsSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnReynoldsSG() {
return this.sb25.getTotalFlow();
}


/**
 *  Gets the jnThamesfordSG attribute of the HydModel object
 *
 * @return    The jnThamesfordSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnThamesfordSG() {
return this.jnThamesfordSG[0];
}


/**
 *  Gets the jnWaubunoSG attribute of the HydModel object
 *
 * @return    The jnWaubunoSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnWaubunoSG() {
return this.sb27.getTotalFlow();
}


/**
 *  Gets the jnEalingSG attribute of the HydModel object
 *
 * @return    The jnEalingSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnEalingSG() {
return this.jnEalingSG[0];
```

```
}


/**
 *  Gets the jnByronSG attribute of the HydModel object
 *
 * @return     The jnByronSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnByronSG() {
return this.jnByronSG[0];
}


/**
 *  Gets the jnByronSGDamage attribute of the HydModel object; from
 * flow [cms] it computes damage in [$1000/s], and then it converts this to
 * [$1000/yr].
 *
 * @return     The jnByronSGDamage value, in [$1000/yr]
 */
public double getJnByronSGDamage() {
return (this.FlowDamageByronSGTable.lookup(
this.jnByronSG[0]) * 3600 * 24 *
this.currentDate.getDaysInMonth() * 12);
}


/**
 *  Gets the jnOxbowSG attribute of the HydModel object
 *
 * @return     The jnOxbowSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnOxbowSG() {
return this.sb32.getTotalFlow();
}


/**
 *  Gets the jnDingmanSG attribute of the HydModel object
 *
 * @return     The jnDingmanSG value, in [m<SUP>3</SUP>/s]
 */
public double getJnDingmanSG() {
return this.sb34.getTotalFlow();
}


//}}}

// reservoir output methods
//{{{
/**
 *  Gets the jnUpStreamWildwood attribute of the HydModel object
 *
 * @return     The jnUpStreamWildwood value, in [m<SUP>3</SUP>/s]
 */
public double getJnUpStreamWildwood() {
return this.jnUpStreamWildwood[0];
}


/**
 *  Gets the jnDownStreamWildwood attribute of the HydModel object
 *
```

```
 * @return    The jnDownStreamWildwood value, in [m<SUP>3</SUP>/s]
 */
public double getJnDownStreamWildwood() {
return this.jnDownStreamWildwood[0];
}


/**
 *  Gets the wildwoodStorage attribute of the HydModel object
 *
 * @return    The wildwoodStorage value, in [m<SUP>3</SUP>/s]
 */
public double getWildwoodStorage() {
return this.wildwood.getStorage();
}


/**
 *  Gets the jnUpStreamFanshawe attribute of the HydModel object
 *
 * @return    The jnUpStreamFanshawe value, in [m<SUP>3</SUP>/s]
 */
public double getJnUpStreamFanshawe() {
return this.jnUpStreamFanshawe[0];
}


/**
 *  Gets the jnDownStreamFanshawe attribute of the HydModel object
 *
 * @return    The jnDownStreamFanshawe value, in [m<SUP>3</SUP>/s]
 */
public double getJnDownStreamFanshawe() {
return this.jnDownStreamFanshawe[0];
}


/**
 *  Gets the fanshaweStorage attribute of the HydModel object
 *
 * @return    The fanshaweStorage value, in [m<SUP>3</SUP>/s]
 */
public double getFanshaweStorage() {
return this.fanshawe.getStorage();
}


/**
 *  Gets the jnUpStreamPittock attribute of the HydModel object
 *
 * @return    The jnUpStreamPittock value, in [m<SUP>3</SUP>/s]
 */
public double getJnUpStreamPittock() {
return this.jnUpStreamPittock[0];
}


/**
 *  Gets the jnDownStreamPittock attribute of the HydModel object
 *
 * @return    The jnDownStreamPittock value, in [m<SUP>3</SUP>/s]
 */
public double getJnDownStreamPittock() {
return this.jnDownStreamPittock[0];
```

```
}


/**
 *  Gets the pittockStorage attribute of the HydModel object
 *
 * @return    The pittockStorage value, in [m<SUP>3</SUP>/s]
 */
public double getPittockStorage() {
return this.pittock.getStorage();
}

//}}}

// other output methods
//{{{
/**
 *  Gets the maxSurfStore attribute of the HydModel object
 *
 * @return    The maxSurfStore value
 */
public double getMaxSurfStore() {
return this.sb1.getMaxSurfStore();
}

public double getMaxSoilInfil(){
return this.sb1.getMaxSoilInfil();
}


/**
 *  Gets the canStore attribute of the HydModel object
 *
 * @return    The canStore value
 */
public double getCanStore() {
return this.sb1.getCanStore();
}


/**
 *  Gets the surfStore attribute of the HydModel object
 *
 * @return    The surfStore value
 */
public double getSurfStore() {
return this.sb1.getSurfStore();
}


/**
 *  Gets the soilStore attribute of the HydModel object
 *
 * @return    The soilStore value
 */
public double getSoilStore() {
return this.sb1.getSoilStore();
}


/**
 *  Gets the gW1Store attribute of the HydModel object
 *
 * @return    The gW1Store value
```

```
 */
public double getGW1Store() {
return this.sb1.getGW1Store();
}


/**
 *  Gets the gW2Store attribute of the HydModel object
 *
 * @return     The gW2Store value
 */
public double getGW2Store() {
return this.sb1.getGW2Store();
}


/**
 *  Gets the excess attribute of the HydModel object
 *
 * @return     The excess value
 */
public double getExcess() {
return this.sb1.getExcess();
}


/**
 *  Gets the totalFlow attribute of the HydModel object
 *
 * @return     The totalFlow value
 */
public double getTotalFlow() {
return this.sb1.getTotalFlow();
}


/**
 *  Gets the actSoilInfil attribute of the HydModel object
 *
 * @return     The actSoilInfil value
 */
public double getActSoilInfil() {
return this.sb1.getActSoilInfil();
}


/**
 *  Gets the actSoilPerc attribute of the HydModel object
 *
 * @return     The actSoilPerc value
 */
public double getActSoilPerc() {
return this.sb1.getActSoilPerc();
}


/**
 *  Gets the potEvapTransVol attribute of the HydModel object
 *
 * @return     The potEvapTransVol value
 */
public double getPotEvapTransVol() {
return this.sb1.getPotEvapTransVol();
}
```

```
/**
 *  Gets the baseflow attribute of the HydModel object
 *
 * @return    The baseflow value
 */
public double getBaseflow() {
return this.sb1.getBaseflow();
}

//}}}

}
```

D.3 ModelDate.java

```java
// all these are needed for the date
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.Date;
import java.text.DateFormat;
import java.util.GregorianCalendar;

/**
 *  A class that keeps track of simulation time
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class ModelDate {

// instance variables
private int year;
private int month;
private int day;
private int hour;
private int minute;

// these get computed by the constructor
private SimpleDateFormat dateFormat;
private GregorianCalendar cal;
private Date currentDate;


/**
 *Constructor for the ModelDate object
 *
 * @param  year    Starting year
 * @param  month   Starting month
 * @param  day     Starting day
 * @param  hour    Starting hour
 * @param  minute  Starting minute
 */
public ModelDate(int year, int month, int day,
int hour, int minute) {

this.year = year;
this.month = month;
this.day = day;
this.hour = hour;
this.minute = minute;

this.dateFormat = new SimpleDateFormat("dd MMM yyyy HH:mm");
this.cal = new GregorianCalendar(year, month, day, hour, minute);
this.currentDate = cal.getTime();
}


/**
 *  Gets the year attribute of the ModelDate object
 *
 * @return    The year value
 */
public int getYear() {
return this.cal.get(GregorianCalendar.YEAR);
}
```

```
/**
 *  Gets the month attribute of the ModelDate object
 *
 * @return     The month value
 */
public int getMonth() {
return this.cal.get(GregorianCalendar.MONTH);
}


/**
 *  Gets the day attribute of the ModelDate object
 *
 * @return     The day value
 */
public int getDay() {
return this.cal.get(GregorianCalendar.DAY_OF_MONTH);
}


/**
 *  Gets the dayOfYear attribute of the ModelDate object
 *
 * @return     The dayOfYear value
 */
public int getDayOfYear() {
return this.cal.get(GregorianCalendar.DAY_OF_YEAR);
}


/**
 *  Gets the dayOfWeek attribute of the ModelDate object
 *
 * @return     The dayOfWeek value
 */
public int getDayOfWeek() {
return this.cal.get(GregorianCalendar.DAY_OF_WEEK);
}


/**
 *  Gets the weekOfYear attribute of the ModelDate object
 *
 * @return     The weekOfYear value
 */
public int getWeekOfYear() {
return this.cal.get(GregorianCalendar.WEEK_OF_YEAR);
}



/**
 *  Gets the hour attribute of the ModelDate object
 *
 * @return     The hour value
 */
public int getHour() {
return this.cal.get(GregorianCalendar.HOUR_OF_DAY);
}


/**
 *  Gets the daysInMonth attribute of the ModelDate object.
 * Note that this method calls the method getMonth() from above
```

```
 *
 * @return    The daysInMonth value
 */
public int getDaysInMonth() {

int[] daysInMonths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
if (this.cal.isLeapYear(getYear())) {
daysInMonths[1] = 29;
}
// no need to put this.getMonth() because getMonth() already uses this
return (daysInMonths[getMonth()]);
}


/**
 *  Gets the daysInYear attribute of the ModelDate object
 *
 * @return    The daysInYear value
 */
public int getDaysInYear() {

int daysInYear;
if (this.cal.isLeapYear(getYear())) {
daysInYear = 366;
} else {
daysInYear = 365;
}
return daysInYear;
}


/**
 *  Gets the date attribute of the ModelDate object
 *
 * @return    The date value
 */
public String getDate() {
return this.dateFormat.format(this.currentDate);
}


/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByMonths(int val) {
this.cal.add(GregorianCalendar.MONTH, val);
this.currentDate = this.cal.getTime();
}


/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByDays(int val) {
this.cal.add(GregorianCalendar.DATE, val);
this.currentDate = this.cal.getTime();

}
```

```
/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByHours(int val) {
this.cal.add(GregorianCalendar.HOUR, val);
this.currentDate = this.cal.getTime();
}


/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByMinutes(int val) {
this.cal.add(GregorianCalendar.MINUTE, val);
this.currentDate = this.cal.getTime();
}

}
```

D.4 SysModel.java

```java
import java.io.*;

/**
 *  This is the System Dynamics model for the counties of Middlesex, Oxford, and
 * Perth.  Middlesex, Oxford and Perth counties are represented as arrays
 * with indices of 0, 1, and 2 respectivelly.  Variables in this class do not
 * use the keyword this, because parameters are not passed into the class, so
 * it not necessary.
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class SysModel {

private ModelDate currentDate;
private double sysTimeStep;

/*
 *  MODEL CONTSTANTS
 */
//{{{
/*
 *  INITIAL CONDITIONS OF STATE VARIABLES
 */
// ALL INITIAL CONDITIONS ARE SET FOR YEAR 2001, AS THIS WAS THE
// LATEST STATSCAN CENSUS AVAILABLE;

// estimated as 2001 jobs divided by 18 persons per business unit
// in [business units]
private final double[] initBusinessStructures = {9590, 1648, 1443};

// the number of dwellings in each county, in [houses]
private final double[] initUrbanHouses = {137760, 17395, 14640};

// the number of urban persons in each county, in [persons]
private final double[] initUrbanPopulation = {336539, 44038, 35969};

// land use types, as they were in 2001, in [km^2]
private final double[] initForestCover = {152.02, 140.88, 120.5};
private final double[] initAgriculturalLand = {760.62, 914.54, 1008.92};

// assumes 70% of urban land use type is residential, in [km^2]
private final double[] initResidentialLand = {0.7 * 204.36, 0.7 * 71.58,
0.7 * 71.4};
// assumes 30% of urban land use type is business, in [km^2]
private final double[] initBusinessLand = {0.3 * 204.36, 0.3 * 71.58,
0.3 * 71.4};

// the number of rural persons in each county, in [persons]
private final double[] initRuralPopulation = {21410, 21676, 16718};

// // estimated as 2001 jobs divided by 8 persons per farm unit
// in [farm units]
private final double[] initFarmUnits = {334, 364, 277};


/*
 *  OTHER MODEL CONSTANTS AND TABLE DEFINITIONS
 */
// most of these constants are the same for each county; this is because
// the data was estimated as it wasn't available at the time of modelling

// NOTE ALL COUNTIES ARE ASSUMED TO HAVE IDENTICAL RELATIONSHIPS IN THE
```

```
// TABLES; IF THIS NEEDS TO BE CHANGED, BUILD A NEW TABLE, AND PROPERLY
// SET IT UP IN initialize() AND update() METHODS
// t stands for a table value of a particular variable

/*
 *  CROSS SECTOR LINKS, CONSTANTS AND TABLE DEFINITIONS
 */
// amount of SW pumped from the Great Lakes in 2001, in [m^3/year]
private final double[] SWPumpedIn2001 = {52305132, 0.0, 0.0};
private final double[] PumpingRatioRelativeTo2001 = {2.0, 0.0, 0.0};

// damage delay time, in [years]; chosen as 0.3333 years because 0.25 years
// gives numerical instability in the smoothing function
private double[] DamageDelayTime = {0.3333, 0.3333, 0.3333};

// GWRecharge delay time smooths the GWRecharge; assumed to be 10 years
private double[] GWRechargeDelayTime = {10.0, 10.0, 10.0};

// an assumed averaged damage; Middlesex County has little damage, while
// Oxford and Perth Counties have much more
private final double[] AvgDamage = {1.0E6, 2.0E8, 4.0E8};

// business and farm investment multiplier based on the amount
// of flood damage
private final double[] txDamageToAvgDamageRatio = {0.0, 0.5, 1.0, 1.5, 2.0};
private final double[] tyInvestmentMultiplier = {1.0, 1.4, 1.7, 1.9, 2.0};
private final Table FloodDamageInvestmentMultiplierTable = new Table(
txDamageToAvgDamageRatio,
tyInvestmentMultiplier);

// these are the initial conditions generated by the method
// calcHistDroughtInfo found in the contUtils class.  The conditions here
// are historical properties of a particular location needed for the
// calculation of drought indicators; letters M, O, P stand for the
// county of interest (Middlesex, Oxford, Perth)

// historical drought information for Middlesex County
private final double[] histOneMonthPPTAvgTotalM = {2038.27, 2345.28, 4014.21,
3474.15, 3103.98, 3257.96, 2990.27, 3382.13, 3520.16,
3068.79, 3589.65, 2826.84};
private double[] histThreeMonthPPTAvgTotalM = {3392.86, 3161.76, 2740.74,
2361.03, 2799.25, 3277.88, 3530.78, 3278.69, 3117.4,
3210.12, 3297.52, 3323.69};
private double[] histEighteenMonthPPTAvgTotalM = {3075.52, 3125.63, 3102.56,
3081.32, 3111.49, 3085.69, 2950.46, 2894.81, 2925.59,
2949.16, 2926.66, 2949.54};
private double histLowAvgSumMonthFlowTotalM = 288.32;
private double[] pptLastEighteenMonthsM = {119.98, 88.93, 133.58, 47.56,
65.82, 36.61, 45.79, 102.97, 97.05, 63.71, 93.79, 66.44,
16.11, 56.37, 116.77, 168.63, 124.77, 52.2};

// historical drought information for Oxford County
private double[] histOneMonthPPTAvgTotalO = {1876.82, 2231.74, 3530.18, 3237.42,
2922.45, 3261.08, 3398.53, 3488.42, 3384.95, 2923.28, 3354.52, 2633.21};
private double[] histThreeMonthPPTAvgTotalO = {3220.91, 2970.34, 2552.03, 2207.84,
2546.25, 2999.78, 3230.02, 3140.32, 3194.02, 3382.68, 3423.96, 3265.55};
private double[] histEighteenMonthPPTAvgTotalO = {2972.91, 3033.29, 3028.58, 3012.29,
3039.33, 3002.28, 2844.83, 2775.66, 2783.03, 2801.06, 2781.35, 2816.83};
private double histLowAvgSumMonthFlowTotalO = 34.06;
private double[] pptLastEighteenMonthsO = {184.75, 62.78, 114.04, 33.58, 58.01,
49.44, 33.88, 107.19, 83.09, 59.67, 89.24, 49.42, 16.3, 53.78, 73.59, 162.5,
106.14, 48.73};

// historical drought information for Perth County
```

```
private double[] histOneMonthPPTAvgTotalP = {1995.46, 2361.46, 4465.12,
3900.91, 3042.83, 3093.75, 3282.3, 3358.82, 3674.65, 3072, 3673.44, 2729.66};
private double[] histThreeMonthPPTAvgTotalP = {3473.36, 3158.36, 2728.01,
2324, 2940.68, 3575.83, 3802.95, 3345.83, 3139.63, 3244.96, 3438.59, 3368.49};
private double[] histEighteenMonthPPTAvgTotalP = {3187.12, 3233.5, 3193.21,
3151.26, 3189.9, 3168.33, 3014.13, 2959.74, 3005.42, 3052.61, 3021.92,
3042.57};
private double histLowAvgSumMonthFlowTotalP = 122.63;
private double[] pptLastEighteenMonthsP = {167.74, 77.11, 132.14, 36.43,
79.98, 44.61, 45.22, 95.32, 98.2, 112.94, 98.97, 67.19, 17.86, 67.48,
91.7, 161.97, 118.11, 47.45};

// the number of years upon which the calculations have been made; this
// is the same for all three counties
private final int numYears = 38;

// drought delay time is set to 1/2 year
private final double[] DroughtDelayTime = {0.5, 0.5, 0.5};

// water use reduction function (based on perceived drought level)
private final double[] txPerceivedDroughtLevel = {0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0};
private final double[] tyWaterUseReductionRatio = {1.0, 0.98, 0.93, 0.85, 0.77, 0.72, 0.7};
private final Table WaterUseReductionRatioTable = new Table(
txPerceivedDroughtLevel, tyWaterUseReductionRatio);

// the fuzzyMin function, modified from Sterman (2000)
private final double[] txFuzzyMin = {0.0, 0.25, 0.5, 0.75, 1.0, 1.3,
1.5, 2.0};
private final double[] tyFuzzyMin = {0.0, 0.25, 0.48, 0.65, 0.8, 0.93,
0.98, 1.0};
private final Table FuzzyMin = new Table(txFuzzyMin, tyFuzzyMin);

/*
 *  this is what I had it like initially
 *  private final double[] txFuzzyMin = {0.0, 0.25, 0.5, 0.75, 1.0, 1.25,
 *  1.5, 1.75, 2.0, 2.25, 2.5};
 *  private final double[] tyFuzzyMin = {0.0, 0.25, 0.48, 0.66, 0.78, 0.85,
 *  0.9, 0.94, 0.97, 0.99, 1.0};
 *  private final Table FuzzyMin = new Table(txFuzzyMin, tyFuzzyMin);
 */
// all water uses have the same effect table, for all counties

private final double[] tWaterUseToAvailability = {0.0, 0.1, 0.2, 0.3, 0.4,
0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
private final double[] tWaterUseEffect = {1.5, 1.45, 1.4, 1.3, 1.2,
1.0, 0.8, 0.5, 0.25, 0.05, 0.0};
private final Table WaterUseEffectTable = new Table(
tWaterUseToAvailability, tWaterUseEffect);

/*
 *  URBAN BUSINESS ACTIVITY SECTOR CONSTANTS AND TABLE DEFINITIONS
 */
// in units of [fraction/year], or 7 new businesses per 100 existing
// businesses per year
private final double[] BusinessConstructionNormal = {0.07, 0.07, 0.07};
private final double[] BusinessDemolitionNormal = {0.025, 0.025, 0.025};

// in units of [jobs/business unit]
private final double[] JobsPerBusinessStructure = {18, 18, 18};

// calculated as LabourForce/UrbanPopulation for year 2001
private final double[] UrbanLabourForceParticipationFraction =
{0.55, 0.72, 0.75};
```

```
// amount of water used in the urban business sector in 2001, in [m^3/yr]
private final double[] BusinessSWUse2001 = {31150524.0, 0.0, 0.0};
private final double[] BusinessGWUse2001 = {10587845, 55179970, 4580542};

// water use allocation fractions
private final double[] FractionTotalSWAllocatedToUrbanBusiness =
{0.596, 0.0, 0.0};
private final double[] FractionTotalGWAllocatedToUrbanBusiness =
{0.615, 0.748, 0.382};

private final double[] tUrbanLabourForceToJobsRatio =
{0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0};
private final double[] tBusinessLabourForceMultiplier =
{0.2, 0.25, 0.35, 0.5, 0.7, 1.0, 1.35, 1.6, 1.8, 1.95, 2.0};
private final Table BusinessLabourForceMultiplierTable = new Table(
tUrbanLabourForceToJobsRatio,
tBusinessLabourForceMultiplier);

private final double[] tBusinessLandFractionOccupied =
{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
private final double[] tBusinessLandMultiplier =
{1.0, 1.15, 1.3, 1.4, 1.45, 1.4, 1.3, 0.9, 0.5, 0.25, 0.0};
private final Table BusinessLandMultiplierTable = new Table(
tBusinessLandFractionOccupied,
tBusinessLandMultiplier);

private final double[] tAttractivnessOfUrbanJobsMultiplier =
{2.0, 1.95, 1.8, 1.6, 1.35, 1.0, 0.5, 0.3, 0.2, 0.15, 0.1};
private final Table AttractivnessOfUrbanJobsMultiplierTable = new Table(
tUrbanLabourForceToJobsRatio,
tAttractivnessOfUrbanJobsMultiplier);

/*
 *  URBAN HOUSING SECTOR CONSTANTS AND TABLE DEFINITIONS
 */
private final double[] UrbanHousingConstructionNormal = {0.07, 0.07, 0.07};
private final double[] UrbanHousingDemolitionNormal = {0.05, 0.05, 0.05};

// obtained by dividing urban population by dwellings
private final double[] HouseholdSize = {2.44, 2.53, 2.46};

private final double[] tResidentialLandFractionOccupied = {0.0, 0.1, 0.2,
0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
private final double[] tUrbanHousingLandMultiplier = {0.4, 0.7, 1.0, 1.25,
1.45, 1.5, 1.5, 1.4, 1.0, 0.5, 0.0};
private final Table UrbanHousingLandMultiplierTable = new Table(
tResidentialLandFractionOccupied,
tUrbanHousingLandMultiplier);

private final double[] tHouseholdsToHousesRatio = {0.0, 0.2, 0.4, 0.6,
0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0};
private final double[] tUrbanHousingAvailabilityMultiplier = {0.2, 0.25,
0.35, 0.5, 0.7, 1.0, 1.35, 1.6, 1.8, 1.95, 2.0};
private final Table UrbanHousingAvailabilityMultiplierTable = new Table(
tHouseholdsToHousesRatio,
tUrbanHousingAvailabilityMultiplier);

private final double[] tAttractivnessOfUrbanHousingMultiplier = {1.4,
1.4, 1.35, 1.3, 1.15, 1.0, 0.8, 0.65, 0.5, 0.45, 0.4};
private final Table AttractivnessOfUrbanHousingMultiplierTable = new Table(
tHouseholdsToHousesRatio,
tAttractivnessOfUrbanHousingMultiplier);
/*
 *  URBAN POPULATION SECTOR CONSTANTS AND TABLE DEFINITIONS
```

```
 */
// water use allocation fractions
private final double[] FractionTotalSWAllocatedToUrbanPopulation =
{0.397, 0.0, 0.0};
private final double[] FractionTotalGWAllocatedToUrbanPopulation =
{0.019, 0.195, 0.348};
private final double[] UrbanInMigrationNormal = {0.1, 0.1, 0.1};
private final double[] UrbanOutMigrationNormal = {0.07, 0.07, 0.07};
private final double[] UrbanBirthsNormal = {0.03, 0.03, 0.03};
private final double[] UrbanDeathsNormal = {0.015, 0.015, 0.015};


// amount of water used in the urban business sector in 2001, in [m^3/yr]
private final double[] UrbanResidentialSWUse2001 = {20767016.0, 0.0, 0.0};
private final double[] UrbanResidentialGWUse2001 = {319375, 14419325, 4172414};


/*
 *  LAND USE SECTOR CONSTANTS AND TABLE DEFINITIONS
 */
private final double[] ForestToAgriculturalRezoningNormal =
{0.005, 0.005, 0.005};
private final double[] AgriculturalToResidentialRezoningNormal =
{0.005, 0.005, 0.005};
private final double[] AgriculturalToBusinessRezoningNormal =
{0.007, 0.007, 0.007};
private final double[] BusinessLandRezoningNormal =
{0.015, 0.015, 0.015};
private final double[] ResidentialLandRezoningNormal =
{0.002, 0.002, 0.002};
private final double[] ResidentialLandRezoningPressureAveragingTime =
{5.0, 5.0, 5.0};


// values in [km^2]; assumes 25m x25m for houses, and 50m x50m for businesses
private final double[] LandPerHouse = {0.000625, 0.000625, 0.000625};
private final double[] LandPerBusinessStructure = {0.0025, 0.0025, 0.0025};


private final double[] tHouseholdsToHousesRatio2 =
{0.0, 0.5, 1.0, 1.5, 2.0};
private final double[] tResidentialCommunityPressureMultiplier =
{1.0, 1.0, 0.8, 0.55, 0.3};
private final Table ResidentialCommunityPressureMultiplierTable = new Table(
tHouseholdsToHousesRatio2,
tResidentialCommunityPressureMultiplier);


private final double[] tUrbanLabourForceToJobsRatio2 =
{0.0, 0.5, 1.0, 4.0};
private final double[] tBusinessCommunityPressureMultiplier =
{1.0, 1.0, 1.0, 4.0};
private final Table BusinessCommunityPressureMultiplierTable = new Table(
tUrbanLabourForceToJobsRatio2,
tBusinessCommunityPressureMultiplier);


private final double[] tFractionOccupiedByFarmUnits =
{0.0, 0.3, 1.0};
private final double[] tForestToAgriculturalRezoningMultiplier =
{1.0, 1.0, 2.4};
private final Table ForestToAgriculturalRezoningMultiplierTable = new Table(
tFractionOccupiedByFarmUnits,
tForestToAgriculturalRezoningMultiplier);


private final double[] tAgriculturalToResidentialRezoningMultiplier =
{0.0, 0.03, 0.08, 0.2, 0.35, 0.6, 0.95, 1.2, 1.35, 1.45, 1.5};
private final Table AgriculturalToResidentialRezoningMultiplierTable = new Table(
tResidentialLandFractionOccupied,
tAgriculturalToResidentialRezoningMultiplier);
```

```
private final double[] tAgriculturalToBusinessRezoningMultiplier =
{0.0, 0.03, 0.08, 0.2, 0.35, 0.6, 0.95, 1.2, 1.35, 1.45, 1.5};
private final Table AgriculturalToBusinessRezoningMultiplierTable = new Table(
tBusinessLandFractionOccupied,
tAgriculturalToBusinessRezoningMultiplier);

private final double[] tResidentialLandRezoningPressureAverage =
{0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0};
private final double[] tResidentialLandRezoningMultiplier =
{0.0, 1.0, 3.0, 5.5, 8.0, 15.0, 25.0};
private final Table ResidentialLandRezoningMultiplierTable = new Table(
tResidentialLandRezoningPressureAverage,
tResidentialLandRezoningMultiplier);

private final double[] tResidentialLandRezoningPressureAverage2 =
{0.0367, 0.844, 2.0, 2.954, 4.0, 5.0, 6.0};
private final double[] tBusinessLandRezoningMultiplier =
{11.97, 7.895, 5.526, 3.816, 1.974, 1.0, 0.0};
private final Table BusinessLandRezoningMultiplierTable = new Table(
tResidentialLandRezoningPressureAverage2,
tBusinessLandRezoningMultiplier);

private final double[] tResidentialLandFractionOccupied2 =
{0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
private final double[] tResidentialLandAvailabilityMultiplier =
{0.4, 0.38, 0.35, 0.32, 0.29, 0.25};
private final Table ResidentialLandAvailabilityMultiplierTable = new Table(
tResidentialLandFractionOccupied2,
tResidentialLandAvailabilityMultiplier);

private final double[] tBusinessLandFractionOccupied2 =
{0.0, 0.2, 0.4, 0.6, 0.8, 1.0};
private final double[] tBusinessLandAvailabilityMultiplier =
{0.4, 0.38, 0.35, 0.3, 0.2, 0.1};
private final Table BusinessLandAvailabilityMultiplierTable = new Table(
tBusinessLandFractionOccupied2,
tBusinessLandAvailabilityMultiplier);

/*
 *  RURAL POPULATION SECTOR CONSTANTS AND TABLE DEFINITIONS
 */
// water use allocation fractions
private final double[] FractionTotalSWAllocatedToRuralResidential =
{0.003, 0.0, 0.0};
private final double[] FractionTotalGWAllocatedToRuralResidential =
{0.104, 0.010, 0.065};
private final double[] RuralInMigrationNormal = {0.1, 0.1, 0.1};
private final double[] RuralOutMigrationNormal = {0.07, 0.07, 0.07};
private final double[] RuralBirthsNormal = {0.03, 0.03, 0.03};
private final double[] RuralDeathsNormal = {0.015, 0.015, 0.015};

// amount of water used in the rural residential sector in 2001, in [m^3/yr]
private final double[] RuralResidentialSWUse2001 = {155037, 0.0, 0.0};
private final double[] RuralResidentialGWUse2001 = {1783824, 740921, 783393};

/*
 *  RURAL BUSINESS ACTIVITY SECTOR AND TABLE DEFINITIONS
 */
private final double[] FarmUnitInvestmentNormal = {0.03, 0.03, 0.03};
private final double[] FarmUnitDepreciationNormal = {0.02, 0.02, 0.02};

// in units of [jobs/business unit]
private final double[] JobsPerFarmUnit = {8, 8, 8};
```

```
// calculated as LabourForce/Population for year 2001
// it actually worked out to be 0.14 for all three counties; strange;
private final double[] RuralLabourForceParticipationFraction =
{0.14, 0.14, 0.14};

// amount of water used in the farm business sector in 2001, in [m^3/yr]
private final double[] FarmSWUse2001 = {232555.0, 0.0, 0.0};
private final double[] FarmGWUse2001 = {4524268, 4189509, 2454040};

// water use allocation fractions
private final double[] FractionTotalSWAllocatedToFarmUnits =
{0.004, 0.0, 0.0};
private final double[] FractionTotalGWAllocatedToFarmUnits =
{0.263, 0.047, 0.205};

private final Table AttractivnessOfRuralJobsMultiplierTable =
AttractivnessOfUrbanJobsMultiplierTable;
private final Table RuralLabourForceMultiplierTable =
BusinessLabourForceMultiplierTable;
private final double[] LandPerFarmUnit = {0.12, 0.12, 0.12};

private final double[] tFractionOccupiedByFarmUnits2 =
{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0};
private final double[] tAvailabilityOfCropLandMultiplier =
{0.0, 0.3, 0.55, 0.8, 1.0, 1.15, 1.25, 1.35, 1.4, 1.45, 1.45};
private final Table AvailabilityOfCropLandMultiplierTable = new Table(
tFractionOccupiedByFarmUnits2,
tAvailabilityOfCropLandMultiplier);

//}}}

/*
 *  VARIABLE DECLARATION
 */
//{{{

/*
 *  STATE VARIABLES
 */
private double[][] BusinessStructures = new double[3][2];
private double[][] UrbanHouses = new double[3][2];
private double[][] UrbanPopulation = new double[3][2];
private double[][] ForestCover = new double[3][2];
private double[][] AgriculturalLand = new double[3][2];
private double[][] ResidentialLand = new double[3][2];
private double[][] BusinessLand = new double[3][2];
private double[][] RuralPopulation = new double[3][2];
private double[][] FarmUnits = new double[3][2];

/*
 *  CROSS SECTOR LINKS OBTAINED FROM THE HYDROLOGICAL MODEL
 */
private double[] MaxAllowableSWPumping = new double[3];
private double[] Damage = new double[3];
private Smooth3[] smDamage = new Smooth3[3];
private double[] PerceivedDamage = new double[3];
private double[] FloodDamageInvestmentMultiplier = new double[3];
private double[] PerceivedFloodDamageInvestmentMultiplier = new double[3];

private double[] GWRecharge = new double[3];
private Smooth3[] smRecharge = new Smooth3[3];
private double[] PerceivedGWRecharge = new double[3];
```

```
private double[] Flow = new double[3];
private double[] PPT = new double[3];

private Drought[] DroughtObj = new Drought[3];
private int[] DroughtLevel = new int[3];
private Smooth3[] smDrought = new Smooth3[3];
private double[] PerceivedDroughtLevel = new double[3];
private double[] WaterUseReductionRatio = new double[3];
private double[] TotalGWUse = new double[3];
private double[] TotalSWUse = new double[3];

/*
 *  URBAN BUSINESS ACTIVITY SECTOR
 */
private double[] BusinessConstruction = new double[3];
private double[] BusinessDemolition = new double[3];
private double[] UrbanJobs = new double[3];
private double[] UrbanLabourForceToJobsRatio = new double[3];
private double[] UrbanLabourForce = new double[3];
private double[] BusinessLabourForceMultiplier = new double[3];
private double[] BusinessConstructionMultiplier = new double[3];
private double[] BusinessLandMultiplier = new double[3];
private double[] AttractivnessOfUrbanJobsMultiplier = new double[3];
private double[] SWUsePerBusinessStructure = new double[3];
private double[] GWUsePerBusinessStructure = new double[3];
private double[] BusinessSWUseDesired = new double[3];
private double[] BusinessGWUseDesired = new double[3];
private double[] BusinessSWUse = new double[3];
private double[] BusinessGWUse = new double[3];
private double[] BusinessSWAvailability = new double[3];
private double[] BusinessGWAvailability = new double[3];
private double[] BusinessSWUseDesiredToAvailability = new double[3];
private double[] BusinessGWUseDesiredToAvailability = new double[3];
private double[] BusinessSWUseToAvailability = new double[3];
private double[] BusinessGWUseToAvailability = new double[3];
private double[] BusinessTotalWaterUse = new double[3];
private double[] BusinessSWUseEffect = new double[3];
private double[] BusinessGWUseEffect = new double[3];
private double[] BusinessWaterAvailabilityMultiplier = new double[3];

/*
 *  URBAN HOUSING SECTOR
 */
private double[] UrbanHousingConstruction = new double[3];
private double[] UrbanHousingDemolition = new double[3];
private double[] HouseholdsToHousesRatio = new double[3];
private double[] AttractivnessOfUrbanHousingMultiplier = new double[3];
private double[] UrbanHousingAvailabilityMultiplier = new double[3];
private double[] UrbanHousingLandMultiplier = new double[3];
private double[] HousingConstructionMultiplier = new double[3];

/*
 *  URBAN POPULATION SECTOR
 */
private double[] UrbanInMigration = new double[3];
private double[] UrbanOutMigration = new double[3];
private double[] UrbanBirths = new double[3];
private double[] UrbanDeaths = new double[3];
private double[] UrbanAttractivnessMultiplier = new double[3];
private double[] SWUsePerUrbanPerson = new double[3];
private double[] GWUsePerUrbanPerson = new double[3];
private double[] UrbanResidentialSWUseDesired = new double[3];
private double[] UrbanResidentialGWUseDesired = new double[3];
private double[] UrbanResidentialSWUse = new double[3];
```

```
private double[] UrbanResidentialGWUse = new double[3];
private double[] UrbanResidentialSWAvailability = new double[3];
private double[] UrbanResidentialGWAvailability = new double[3];
private double[] UrbanResidentialSWUseDesiredToAvailability = new double[3];
private double[] UrbanResidentialGWUseDesiredToAvailability = new double[3];
private double[] UrbanResidentialSWUseToAvailability = new double[3];
private double[] UrbanResidentialGWUseToAvailability = new double[3];
private double[] UrbanResidentialTotalWaterUse = new double[3];
private double[] UrbanResidentialSWUseEffect = new double[3];
private double[] UrbanResidentialGWUseEffect = new double[3];
private double[] UrbanResidentialWaterAvailabilityAttractivness = new double[3];

/*
 * LAND USE SECTOR
 */
private double[] BusinessLandOccupied = new double[3];
private double[] ResidentialLandOccupied = new double[3];
private double[] BusinessLandFractionOccupied = new double[3];
private double[] ResidentialLandFractionOccupied = new double[3];
private double[] ResidentialLandAvailabilityMultiplier = new double[3];
private double[] BusinessLandAvailabilityMultiplier = new double[3];
private double[] BusinessLandFractionRezoned = new double[3];
private double[] ResidentialLandFractionRezoned = new double[3];
private double[] ResidentialLandRezoningMultiplier = new double[3];
private double[] BusinessLandRezoningMultiplier = new double[3];
private double[] ResidentialCommunityPressureMultiplier = new double[3];
private double[] BusinessCommunityPressureMultiplier = new double[3];
private double[] ResidentialLandRezoningPressure = new double[3];
private double[] ResidentialLandRezoningPressureAverage = new double[3];
private double[] ForestToAgriculturalRezoningMultiplier = new double[3];
private double[] AgriculturalToResidentialRezoningMultiplier = new double[3];
private double[] AgriculturalToBusinessRezoningMultiplier = new double[3];

private double[] ForestToAgriculturalRezoning = new double[3];
private double[] AgriculturalToResidentialRezoning = new double[3];
private double[] AgriculturalToBusinessRezoning = new double[3];
private double[] ResidentialLandRezoning = new double[3];
private double[] BusinessLandRezoning = new double[3];

private Smooth3[] resLandRezPres = new Smooth3[3];

/*
 * RURAL POPULAION SECTOR
 */
private double[] RuralInMigration = new double[3];
private double[] RuralOutMigration = new double[3];
private double[] RuralBirths = new double[3];
private double[] RuralDeaths = new double[3];
private double[] RuralAttractivnessMultiplier = new double[3];
private double[] SWUsePerRuralPerson = new double[3];
private double[] GWUsePerRuralPerson = new double[3];
private double[] RuralResidentialSWUseDesired = new double[3];
private double[] RuralResidentialGWUseDesired = new double[3];
private double[] RuralResidentialSWUse = new double[3];
private double[] RuralResidentialGWUse = new double[3];
private double[] RuralResidentialSWAvailability = new double[3];
private double[] RuralResidentialGWAvailability = new double[3];
private double[] RuralResidentialSWUseDesiredToAvailability = new double[3];
private double[] RuralResidentialGWUseDesiredToAvailability = new double[3];
private double[] RuralResidentialSWUseToAvailability = new double[3];
private double[] RuralResidentialGWUseToAvailability = new double[3];
private double[] RuralResidentialTotalWaterUse = new double[3];
private double[] RuralResidentialSWUseEffect = new double[3];
private double[] RuralResidentialGWUseEffect = new double[3];
```

```
        private double[] RuralResidentialWaterAvailabilityAttractivness = new double[3];

        /*
         *  RURAL BUSINESS ACTIVITY SECTOR
         */
        private double[] FractionOccupiedByFarmUnits = new double[3];
        private double[] FractionOccupiedByCropLand = new double[3];
        private double[] AttractivnessOfRuralJobsMultiplier = new double[3];
        private double[] AvailabilityOfCropLandMultiplier = new double[3];
        private double[] LandYieldInvestmentMultiplier = new double[3];
        private double[] FarmUnitInvestmentMultiplier = new double[3];
        private double[] FarmJobs = new double[3];
        private double[] RuralLabourForce = new double[3];
        private double[] RuralLabourForceToJobsRatio = new double[3];
        private double[] RuralLabourForceMultiplier = new double[3];
        private double[] FarmUnitInvestment = new double[3];
        private double[] FarmUnitDepreciation = new double[3];

        private double[] SWUsePerFarmUnit = new double[3];
        private double[] GWUsePerFarmUnit = new double[3];
        private double[] FarmSWUseDesired = new double[3];
        private double[] FarmGWUseDesired = new double[3];
        private double[] FarmSWUse = new double[3];
        private double[] FarmGWUse = new double[3];
        private double[] FarmSWAvailability = new double[3];
        private double[] FarmGWAvailability = new double[3];
        private double[] FarmSWUseDesiredToAvailability = new double[3];
        private double[] FarmGWUseDesiredToAvailability = new double[3];
        private double[] FarmSWUseToAvailability = new double[3];
        private double[] FarmGWUseToAvailability = new double[3];
        private double[] FarmTotalWaterUse = new double[3];
        private double[] FarmSWUseEffect = new double[3];
        private double[] FarmGWUseEffect = new double[3];
        private double[] FarmWaterAvailabilityMultiplier = new double[3];


        //}}}

        /**
         *  Constructor for the SysModel object
         *
         * @param  currentDate  Current date
         * @param  sysTimeStep  Time step of the System Dynamics model, in [months]
         */
        public SysModel(ModelDate currentDate, double sysTimeStep) {
        this.currentDate = currentDate;

        // convert it to years here
        this.sysTimeStep = sysTimeStep / 12.0;
        }


        /**
         *  Method that initializes all variables
         */
        public void initialize() {
        //{{{

        // CROSS SECTOR LINKS ARE INITIALIZED FIRST

        // to initialize the Drought objects
        // Middlesex
        DroughtObj[0] = new Drought(this.currentDate, this.numYears);
        DroughtObj[0].setHistOneMonthPPT(histOneMonthPPTAvgTotalM);
```

```
DroughtObj[0].setHistThreeMonthPPT(histThreeMonthPPTAvgTotalM);
DroughtObj[0].setHistEighteenMonthPPT(histEighteenMonthPPTAvgTotalM);
DroughtObj[0].setHistLowAvgSumMonthFlow(histLowAvgSumMonthFlowTotalM);
DroughtObj[0].setPPTLastEighteenMonths(pptLastEighteenMonthsM);

// Oxford
DroughtObj[1] = new Drought(this.currentDate, this.numYears);
DroughtObj[1].setHistOneMonthPPT(histOneMonthPPTAvgTotalO);
DroughtObj[1].setHistThreeMonthPPT(histThreeMonthPPTAvgTotalO);
DroughtObj[1].setHistEighteenMonthPPT(histEighteenMonthPPTAvgTotalO);
DroughtObj[1].setHistLowAvgSumMonthFlow(histLowAvgSumMonthFlowTotalO);
DroughtObj[1].setPPTLastEighteenMonths(pptLastEighteenMonthsO);

// Perth
DroughtObj[2] = new Drought(this.currentDate, this.numYears);
DroughtObj[2].setHistOneMonthPPT(histOneMonthPPTAvgTotalP);
DroughtObj[2].setHistThreeMonthPPT(histThreeMonthPPTAvgTotalP);
DroughtObj[2].setHistEighteenMonthPPT(histEighteenMonthPPTAvgTotalP);
DroughtObj[2].setHistLowAvgSumMonthFlow(histLowAvgSumMonthFlowTotalP);
DroughtObj[2].setPPTLastEighteenMonths(pptLastEighteenMonthsP);

// damage values are already initialized by calling the getDamage()
// method prior to initialize, so that is why we can do this:
for (int i = 0; i < 3; i++) {
MaxAllowableSWPumping[i] = SWPumpedIn2001[i] *
PumpingRatioRelativeTo2001[i];
FloodDamageInvestmentMultiplier[i] =
FloodDamageInvestmentMultiplierTable.lookup(
Damage[i] / AvgDamage[i]);

smDamage[i] = new Smooth3(this.sysTimeStep);
smDamage[i].initialize(FloodDamageInvestmentMultiplier[i],
DamageDelayTime[i]);
PerceivedFloodDamageInvestmentMultiplier[i] =
smDamage[i].getOutput();

// setGWRecharge() must be called before this is executed
smRecharge[i] = new Smooth3(this.sysTimeStep);
smRecharge[i].initialize(GWRecharge[i], GWRechargeDelayTime[i]);
PerceivedGWRecharge[i] = smRecharge[i].getOutput();

DroughtLevel[i] = DroughtObj[i].getDroughtLevel();
smDrought[i] = new Smooth3(this.sysTimeStep);
smDrought[i].initialize(DroughtLevel[i], DroughtDelayTime[i]);
PerceivedDroughtLevel[i] = smDrought[i].getOutput();

WaterUseReductionRatio[i] = WaterUseReductionRatioTable.lookup(
PerceivedDroughtLevel[i]);
}

for (int i = 0; i < 3; i++) {
/*
 *  STATE VARIABLES ARE INITIALIZED
 */
BusinessStructures[i][0] = initBusinessStructures[i];
UrbanHouses[i][0] = initUrbanHouses[i];
UrbanPopulation[i][0] = initUrbanPopulation[i];
ForestCover[i][0] = initForestCover[i];
AgriculturalLand[i][0] = initAgriculturalLand[i];
ResidentialLand[i][0] = initResidentialLand[i];
BusinessLand[i][0] = initBusinessLand[i];
RuralPopulation[i][0] = initRuralPopulation[i];
FarmUnits[i][0] = initFarmUnits[i];
```

```
/*
 *  CROSS SECTOR LINKS
 */
BusinessLandOccupied[i] = BusinessStructures[i][0] *
LandPerBusinessStructure[i];
ResidentialLandOccupied[i] = UrbanHouses[i][0] *
LandPerHouse[i];
FractionOccupiedByFarmUnits[i] = (LandPerFarmUnit[i] *
FarmUnits[i][0]) / AgriculturalLand[i][0];

BusinessLandFractionOccupied[i] = BusinessLandOccupied[i] /
BusinessLand[i][0];
ResidentialLandFractionOccupied[i] = ResidentialLandOccupied[i] /
ResidentialLand[i][0];

FarmJobs[i] = FarmUnits[i][0] * JobsPerFarmUnit[i];
RuralLabourForce[i] = RuralPopulation[i][0] *
RuralLabourForceParticipationFraction[i];
RuralLabourForceToJobsRatio[i] = RuralLabourForce[i] /
FarmJobs[i];
AttractivnessOfRuralJobsMultiplier[i] =
AttractivnessOfRuralJobsMultiplierTable.lookup(
RuralLabourForceToJobsRatio[i]);

/*
 *  URBAN BUSINESS ACTIVITY SECTOR
 */
UrbanJobs[i] = BusinessStructures[i][0] *
JobsPerBusinessStructure[i];
UrbanLabourForce[i] = UrbanPopulation[i][0] *
UrbanLabourForceParticipationFraction[i];
UrbanLabourForceToJobsRatio[i] = UrbanLabourForce[i] /
UrbanJobs[i];
BusinessLabourForceMultiplier[i] =
BusinessLabourForceMultiplierTable.lookup(
UrbanLabourForceToJobsRatio[i]);
AttractivnessOfUrbanJobsMultiplier[i] =
AttractivnessOfUrbanJobsMultiplierTable.lookup(
UrbanLabourForceToJobsRatio[i]);
BusinessLandMultiplier[i] =
BusinessLandMultiplierTable.lookup(
BusinessLandFractionOccupied[i]);
GWUsePerBusinessStructure[i] = BusinessGWUse2001[i] /
initBusinessStructures[i];
SWUsePerBusinessStructure[i] = BusinessSWUse2001[i] /
initBusinessStructures[i];
BusinessSWUseDesired[i] = BusinessStructures[i][0] *
SWUsePerBusinessStructure[i] * WaterUseReductionRatio[i];
BusinessGWUseDesired[i] = BusinessStructures[i][0] *
GWUsePerBusinessStructure[i] * WaterUseReductionRatio[i];
BusinessSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToUrbanBusiness[i];
BusinessGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToUrbanBusiness[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
BusinessSWUseDesiredToAvailability[i] =
BusinessSWUseDesired[i] /
BusinessSWAvailability[i];
BusinessGWUseDesiredToAvailability[i] =
BusinessGWUseDesired[i] /
BusinessGWAvailability[i];
```

```
// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(BusinessSWUseDesiredToAvailability[i])) {
BusinessSWUseDesiredToAvailability[i] = 0.0;
BusinessSWUseEffect[i] = 0.0;
} else {
BusinessSWUseToAvailability[i] = FuzzyMin.lookup(
BusinessSWUseDesiredToAvailability[i]);
BusinessSWUseEffect[i] = WaterUseEffectTable.lookup(
BusinessSWUseToAvailability[i]);
}

if (Double.isNaN(BusinessGWUseDesiredToAvailability[i])) {
BusinessGWUseDesiredToAvailability[i] = 0.0;
BusinessGWUseEffect[i] = 0.0;
} else {
BusinessGWUseToAvailability[i] = FuzzyMin.lookup(
BusinessGWUseDesiredToAvailability[i]);
BusinessGWUseEffect[i] = WaterUseEffectTable.lookup(
BusinessGWUseToAvailability[i]);
}

BusinessSWUse[i] = BusinessSWAvailability[i] *
BusinessSWUseToAvailability[i];
BusinessGWUse[i] = BusinessGWAvailability[i] *
BusinessGWUseToAvailability[i];

BusinessTotalWaterUse[i] = BusinessSWUse[i] +
BusinessGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
BusinessWaterAvailabilityMultiplier[i] =
BusinessSWUseEffect[i] * (BusinessSWUse[i] /
BusinessTotalWaterUse[i]) +
BusinessGWUseEffect[i] * (BusinessGWUse[i] /
BusinessTotalWaterUse[i]);

BusinessConstructionMultiplier[i] =
PerceivedFloodDamageInvestmentMultiplier[i] *
BusinessLabourForceMultiplier[i] *
BusinessLandMultiplier[i] *
BusinessWaterAvailabilityMultiplier[i];
BusinessConstruction[i] = BusinessConstructionMultiplier[i] *
BusinessConstructionNormal[i] *
BusinessStructures[i][0];
BusinessDemolition[i] = BusinessDemolitionNormal[i] *
BusinessStructures[i][0];
/*
 *  URBAN HOUSING SECTOR
 */
UrbanHousingDemolition[i] = UrbanHouses[i][0] *
UrbanHousingDemolitionNormal[i];
HouseholdsToHousesRatio[i] = UrbanPopulation[i][0] /
(UrbanHouses[i][0] * HouseholdSize[i]);
AttractivnessOfUrbanHousingMultiplier[i] =
AttractivnessOfUrbanHousingMultiplierTable.lookup(
HouseholdsToHousesRatio[i]);
UrbanHousingAvailabilityMultiplier[i] =
UrbanHousingAvailabilityMultiplierTable.lookup(
HouseholdsToHousesRatio[i]);
UrbanHousingLandMultiplier[i] =
```

```
UrbanHousingLandMultiplierTable.lookup(
ResidentialLandFractionOccupied[i]);
HousingConstructionMultiplier[i] =
UrbanHousingLandMultiplier[i] *
UrbanHousingAvailabilityMultiplier[i];
UrbanHousingConstruction[i] = UrbanHousingConstructionNormal[i] *
UrbanHouses[i][0] * HousingConstructionMultiplier[i];

/*
 *  URBAN POPULATION SECTOR
 */
GWUsePerUrbanPerson[i] = UrbanResidentialGWUse2001[i] /
initUrbanPopulation[i];
SWUsePerUrbanPerson[i] = UrbanResidentialSWUse2001[i] /
initUrbanPopulation[i];
UrbanResidentialSWUseDesired[i] = UrbanPopulation[i][0] *
SWUsePerUrbanPerson[i] * WaterUseReductionRatio[i];
UrbanResidentialGWUseDesired[i] = UrbanPopulation[i][0] *
GWUsePerUrbanPerson[i] * WaterUseReductionRatio[i];
UrbanResidentialSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToUrbanPopulation[i];
UrbanResidentialGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToUrbanPopulation[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
UrbanResidentialSWUseDesiredToAvailability[i] =
UrbanResidentialSWUseDesired[i] /
UrbanResidentialSWAvailability[i];
UrbanResidentialGWUseDesiredToAvailability[i] =
UrbanResidentialGWUseDesired[i] /
UrbanResidentialGWAvailability[i];

// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(UrbanResidentialSWUseDesiredToAvailability[i])) {
UrbanResidentialSWUseDesiredToAvailability[i] = 0.0;
UrbanResidentialSWUseEffect[i] = 0.0;
} else {
UrbanResidentialSWUseToAvailability[i] = FuzzyMin.lookup(
UrbanResidentialSWUseDesiredToAvailability[i]);
UrbanResidentialSWUseEffect[i] = WaterUseEffectTable.lookup(
UrbanResidentialSWUseToAvailability[i]);
}

if (Double.isNaN(UrbanResidentialGWUseDesiredToAvailability[i])) {
UrbanResidentialGWUseDesiredToAvailability[i] = 0.0;
UrbanResidentialGWUseEffect[i] = 0.0;
} else {
UrbanResidentialGWUseToAvailability[i] = FuzzyMin.lookup(
UrbanResidentialGWUseDesiredToAvailability[i]);
UrbanResidentialGWUseEffect[i] = WaterUseEffectTable.lookup(
UrbanResidentialGWUseToAvailability[i]);
}

UrbanResidentialSWUse[i] = UrbanResidentialSWAvailability[i] *
UrbanResidentialSWUseToAvailability[i];
UrbanResidentialGWUse[i] = UrbanResidentialGWAvailability[i] *
UrbanResidentialGWUseToAvailability[i];

UrbanResidentialTotalWaterUse[i] = UrbanResidentialSWUse[i] +
UrbanResidentialGWUse[i];
```

```
// this multiplier is a weighted average of both SW and GW
// used in each county
UrbanResidentialWaterAvailabilityAttractivness[i] =
UrbanResidentialSWUseEffect[i] * (UrbanResidentialSWUse[i] /
UrbanResidentialTotalWaterUse[i]) +
UrbanResidentialGWUseEffect[i] * (UrbanResidentialGWUse[i] /
UrbanResidentialTotalWaterUse[i]);

UrbanAttractivnessMultiplier[i] = AttractivnessOfUrbanHousingMultiplier[i] *
AttractivnessOfUrbanJobsMultiplier[i] *
UrbanResidentialWaterAvailabilityAttractivness[i];
UrbanInMigration[i] = UrbanInMigrationNormal[i] * UrbanPopulation[i][0] *
UrbanAttractivnessMultiplier[i];
UrbanBirths[i] = UrbanBirthsNormal[i] * UrbanPopulation[i][0];
UrbanDeaths[i] = UrbanDeathsNormal[i] * UrbanPopulation[i][0];
UrbanOutMigration[i] = UrbanOutMigrationNormal[i] * UrbanPopulation[i][0];

/*
 *  LAND USE SECTOR
 */
AgriculturalToBusinessRezoningMultiplier[i] =
AgriculturalToBusinessRezoningMultiplierTable.lookup(
BusinessLandFractionOccupied[i]);
AgriculturalToBusinessRezoning[i] = AgriculturalLand[i][0] *
AgriculturalToBusinessRezoningNormal[i] *
AgriculturalToBusinessRezoningMultiplier[i];

AgriculturalToResidentialRezoningMultiplier[i] =
AgriculturalToResidentialRezoningMultiplierTable.lookup(
ResidentialLandFractionOccupied[i]);
AgriculturalToResidentialRezoning[i] = AgriculturalLand[i][0] *
AgriculturalToResidentialRezoningNormal[i] *
AgriculturalToResidentialRezoningMultiplier[i];

ResidentialLandAvailabilityMultiplier[i] =
ResidentialLandAvailabilityMultiplierTable.lookup(
ResidentialLandFractionOccupied[i]);
BusinessLandAvailabilityMultiplier[i] =
BusinessLandAvailabilityMultiplierTable.lookup(
BusinessLandFractionOccupied[i]);

BusinessCommunityPressureMultiplier[i] =
BusinessCommunityPressureMultiplierTable.lookup(
UrbanLabourForceToJobsRatio[i]);
ResidentialCommunityPressureMultiplier[i] =
ResidentialCommunityPressureMultiplierTable.lookup(
HouseholdsToHousesRatio[i]);

ResidentialLandRezoningPressure[i] =
BusinessCommunityPressureMultiplier[i] *
ResidentialCommunityPressureMultiplier[i];

resLandRezPres[i] = new Smooth3(this.sysTimeStep);
resLandRezPres[i].initialize(ResidentialLandRezoningPressure[i],
ResidentialLandRezoningPressureAveragingTime[i]);

ResidentialLandRezoningPressureAverage[i] =
resLandRezPres[i].getOutput();

BusinessLandRezoningMultiplier[i] =
BusinessLandRezoningMultiplierTable.lookup(
ResidentialLandRezoningPressureAverage[i]);
ResidentialLandRezoningMultiplier[i] =
```

```
ResidentialLandRezoningMultiplierTable.lookup(
ResidentialLandRezoningPressureAverage[i]);

ResidentialLandFractionRezoned[i] =
ResidentialLandRezoningNormal[i] *
ResidentialLandAvailabilityMultiplier[i] *
ResidentialLandRezoningMultiplier[i];
BusinessLandFractionRezoned[i] =
BusinessLandRezoningNormal[i] *
BusinessLandAvailabilityMultiplier[i] *
BusinessLandRezoningMultiplier[i];

ResidentialLandRezoning[i] =
ResidentialLand[i][0] *
ResidentialLandFractionRezoned[i];

BusinessLandRezoning[i] =
BusinessLand[i][0] *
BusinessLandFractionRezoned[i];

ForestToAgriculturalRezoningMultiplier[i] =
ForestToAgriculturalRezoningMultiplierTable.lookup(
FractionOccupiedByFarmUnits[i]);
ForestToAgriculturalRezoning[i] = ForestCover[i][0] *
ForestToAgriculturalRezoningNormal[i] *
ForestToAgriculturalRezoningMultiplier[i];

/*
 *  RURAL POPULATION SECTOR
 */
GWUsePerRuralPerson[i] = RuralResidentialGWUse2001[i] /
initRuralPopulation[i];
SWUsePerRuralPerson[i] = RuralResidentialSWUse2001[i] /
initRuralPopulation[i];
RuralResidentialSWUseDesired[i] = RuralPopulation[i][0] *
SWUsePerRuralPerson[i] * WaterUseReductionRatio[i];
RuralResidentialGWUseDesired[i] = RuralPopulation[i][0] *
GWUsePerRuralPerson[i] * WaterUseReductionRatio[i];
RuralResidentialSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToRuralResidential[i];
RuralResidentialGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToRuralResidential[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
RuralResidentialSWUseDesiredToAvailability[i] =
RuralResidentialSWUseDesired[i] /
RuralResidentialSWAvailability[i];
RuralResidentialGWUseDesiredToAvailability[i] =
RuralResidentialGWUseDesired[i] /
RuralResidentialGWAvailability[i];

// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(RuralResidentialSWUseDesiredToAvailability[i])) {
RuralResidentialSWUseDesiredToAvailability[i] = 0.0;
RuralResidentialSWUseEffect[i] = 0.0;
} else {
RuralResidentialSWUseToAvailability[i] = FuzzyMin.lookup(
RuralResidentialSWUseDesiredToAvailability[i]);
RuralResidentialSWUseEffect[i] = WaterUseEffectTable.lookup(
RuralResidentialSWUseToAvailability[i]);
```

```
}

if (Double.isNaN(RuralResidentialGWUseDesiredToAvailability[i])) {
RuralResidentialGWUseDesiredToAvailability[i] = 0.0;
RuralResidentialGWUseEffect[i] = 0.0;
} else {
RuralResidentialGWUseToAvailability[i] = FuzzyMin.lookup(
RuralResidentialGWUseDesiredToAvailability[i]);
RuralResidentialGWUseEffect[i] = WaterUseEffectTable.lookup(
RuralResidentialGWUseToAvailability[i]);
}

RuralResidentialSWUse[i] = RuralResidentialSWAvailability[i] *
RuralResidentialSWUseToAvailability[i];
RuralResidentialGWUse[i] = RuralResidentialGWAvailability[i] *
RuralResidentialGWUseToAvailability[i];

RuralResidentialTotalWaterUse[i] = RuralResidentialSWUse[i] +
RuralResidentialGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
RuralResidentialWaterAvailabilityAttractivness[i] =
RuralResidentialSWUseEffect[i] * (RuralResidentialSWUse[i] /
RuralResidentialTotalWaterUse[i]) +
RuralResidentialGWUseEffect[i] * (RuralResidentialGWUse[i] /
RuralResidentialTotalWaterUse[i]);

RuralAttractivnessMultiplier[i] =
AttractivnessOfRuralJobsMultiplier[i] *
RuralResidentialWaterAvailabilityAttractivness[i];
RuralInMigration[i] = RuralInMigrationNormal[i] * RuralPopulation[i][0] *
RuralAttractivnessMultiplier[i];
RuralBirths[i] = RuralBirthsNormal[i] * RuralPopulation[i][0];
RuralDeaths[i] = RuralDeathsNormal[i] * RuralPopulation[i][0];
RuralOutMigration[i] = RuralOutMigrationNormal[i] * RuralPopulation[i][0];

/*
 *  RURAL BUSINESS ACTIVITY SECTOR
 */
RuralLabourForceMultiplier[i] = RuralLabourForceMultiplierTable.lookup(
RuralLabourForceToJobsRatio[i]);

FractionOccupiedByCropLand[i] = 1.0 -
FractionOccupiedByFarmUnits[i];
AvailabilityOfCropLandMultiplier[i] =
AvailabilityOfCropLandMultiplierTable.lookup(
FractionOccupiedByCropLand[i]);
LandYieldInvestmentMultiplier[i] =
AvailabilityOfCropLandMultiplier[i];

GWUsePerFarmUnit[i] = FarmGWUse2001[i] / initFarmUnits[i];
SWUsePerFarmUnit[i] = FarmSWUse2001[i] / initFarmUnits[i];
FarmSWUseDesired[i] = FarmUnits[i][0] * SWUsePerFarmUnit[i] *
WaterUseReductionRatio[i];
FarmGWUseDesired[i] = FarmUnits[i][0] * GWUsePerFarmUnit[i] *
WaterUseReductionRatio[i];
FarmSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToFarmUnits[i];
FarmGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToFarmUnits[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
```

```
FarmSWUseDesiredToAvailability[i] =
FarmSWUseDesired[i] / FarmSWAvailability[i];
FarmGWUseDesiredToAvailability[i] =
FarmGWUseDesired[i] / FarmGWAvailability[i];

// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(FarmSWUseDesiredToAvailability[i])) {
FarmSWUseDesiredToAvailability[i] = 0.0;
FarmSWUseEffect[i] = 0.0;
} else {
FarmSWUseToAvailability[i] = FuzzyMin.lookup(
FarmSWUseDesiredToAvailability[i]);
FarmSWUseEffect[i] = WaterUseEffectTable.lookup(
FarmSWUseToAvailability[i]);
}

if (Double.isNaN(FarmGWUseDesiredToAvailability[i])) {
FarmGWUseDesiredToAvailability[i] = 0.0;
FarmGWUseEffect[i] = 0.0;
} else {
FarmGWUseToAvailability[i] = FuzzyMin.lookup(
FarmGWUseDesiredToAvailability[i]);
FarmGWUseEffect[i] = WaterUseEffectTable.lookup(
FarmGWUseToAvailability[i]);
}

FarmSWUse[i] = FarmSWAvailability[i] * FarmSWUseToAvailability[i];
FarmGWUse[i] = FarmGWAvailability[i] * FarmGWUseToAvailability[i];

FarmTotalWaterUse[i] = FarmSWUse[i] + FarmGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
FarmWaterAvailabilityMultiplier[i] =
FarmSWUseEffect[i] * (FarmSWUse[i] /
FarmTotalWaterUse[i]) +
FarmGWUseEffect[i] * (FarmGWUse[i] /
FarmTotalWaterUse[i]);

FarmUnitInvestmentMultiplier[i] =
PerceivedFloodDamageInvestmentMultiplier[i] *
RuralLabourForceMultiplier[i] *
FarmWaterAvailabilityMultiplier[i] *
LandYieldInvestmentMultiplier[i];
FarmUnitInvestment[i] = FarmUnits[i][0] *
FarmUnitInvestmentNormal[i] *
FarmUnitInvestmentMultiplier[i];
FarmUnitDepreciation[i] = FarmUnits[i][0] *
FarmUnitDepreciationNormal[i];

// total county water use
TotalGWUse[i] = BusinessGWUse[i] + UrbanResidentialGWUse[i] +
RuralResidentialGWUse[i] + FarmGWUse[i];

TotalSWUse[i] = BusinessSWUse[i] + UrbanResidentialSWUse[i] +
RuralResidentialSWUse[i] + FarmSWUse[i];

}
//}}}
}
```

```
/**
 *  Method that updates all variables, and resets current state to
 * previous state.  Auxiliary variables need not have saved current and
 * previous value; only the state variables do.
 *
 * @param   currentDate  Description of the Parameter
 */
public void update(ModelDate currentDate) {
//{{{

this.currentDate = currentDate;
for (int i = 0; i < 3; i++) {

// updates the smoothed damage function; note the function
// getDamage() is called in the main before this one, so
// this update method has new values each time step

FloodDamageInvestmentMultiplier[i] =
FloodDamageInvestmentMultiplierTable.lookup(
Damage[i] / AvgDamage[i]);

smDamage[i].update(FloodDamageInvestmentMultiplier[i],
DamageDelayTime[i]);
PerceivedFloodDamageInvestmentMultiplier[i] =
smDamage[i].getOutput();

// setGWRecharge() must be called before this is executed
smRecharge[i].update(GWRecharge[i], GWRechargeDelayTime[i]);
PerceivedGWRecharge[i] = smRecharge[i].getOutput();

// updates the drought objects
DroughtObj[i].update(this.currentDate, PPT[i], Flow[i]);

DroughtLevel[i] = DroughtObj[i].getDroughtLevel();
smDrought[i].update(DroughtLevel[i], DroughtDelayTime[i]);
PerceivedDroughtLevel[i] = smDrought[i].getOutput();

WaterUseReductionRatio[i] = WaterUseReductionRatioTable.lookup(
PerceivedDroughtLevel[i]);

// updates the state variables
BusinessStructures[i][1] = BusinessStructures[i][0] +
(BusinessConstruction[i] -
BusinessDemolition[i]) * sysTimeStep;
UrbanHouses[i][1] = UrbanHouses[i][0] +
(UrbanHousingConstruction[i] -
UrbanHousingDemolition[i]) * sysTimeStep;
UrbanPopulation[i][1] = UrbanPopulation[i][0] +
(UrbanInMigration[i] + UrbanBirths[i] -
UrbanDeaths[i] - UrbanOutMigration[i]) *
sysTimeStep;

ForestCover[i][1] = ForestCover[i][0] +
(-1.0 * ForestToAgriculturalRezoning[i]) *
sysTimeStep;
AgriculturalLand[i][1] = AgriculturalLand[i][0] +
(ForestToAgriculturalRezoning[i] -
AgriculturalToBusinessRezoning[i] -
AgriculturalToResidentialRezoning[i]) *
sysTimeStep;
ResidentialLand[i][1] = ResidentialLand[i][0] +
(AgriculturalToResidentialRezoning[i] +
BusinessLandRezoning[i] -
```

```
ResidentialLandRezoning[i]) * sysTimeStep;
BusinessLand[i][1] = BusinessLand[i][0] +
(ResidentialLandRezoning[i] +
AgriculturalToBusinessRezoning[i] -
BusinessLandRezoning[i]) * sysTimeStep;

RuralPopulation[i][1] = RuralPopulation[i][0] +
(RuralInMigration[i] + RuralBirths[i] -
RuralDeaths[i] - RuralOutMigration[i]) *
sysTimeStep;
FarmUnits[i][1] = FarmUnits[i][0] +
(FarmUnitInvestment[i] -
FarmUnitDepreciation[i]) * sysTimeStep;

// updates the auxiliary variables
/*
 *  CROSS SECTOR LINKS
 */
BusinessLandOccupied[i] = BusinessStructures[i][1] *
LandPerBusinessStructure[i];
ResidentialLandOccupied[i] = UrbanHouses[i][1] *
LandPerHouse[i];
FractionOccupiedByFarmUnits[i] = (LandPerFarmUnit[i] *
FarmUnits[i][1]) / AgriculturalLand[i][1];

BusinessLandFractionOccupied[i] = BusinessLandOccupied[i] /
BusinessLand[i][1];
ResidentialLandFractionOccupied[i] = ResidentialLandOccupied[i] /
ResidentialLand[i][1];

FarmJobs[i] = FarmUnits[i][1] * JobsPerFarmUnit[i];
RuralLabourForce[i] = RuralPopulation[i][1] *
RuralLabourForceParticipationFraction[i];
RuralLabourForceToJobsRatio[i] = RuralLabourForce[i] /
FarmJobs[i];
AttractivnessOfRuralJobsMultiplier[i] =
AttractivnessOfRuralJobsMultiplierTable.lookup(
RuralLabourForceToJobsRatio[i]);

/*
 *  URBAN BUSINESS ACTIVITY SECTOR
 */
UrbanJobs[i] = BusinessStructures[i][1] *
JobsPerBusinessStructure[i];
UrbanLabourForce[i] = UrbanPopulation[i][1] *
UrbanLabourForceParticipationFraction[i];
UrbanLabourForceToJobsRatio[i] = UrbanLabourForce[i] /
UrbanJobs[i];
BusinessLabourForceMultiplier[i] =
BusinessLabourForceMultiplierTable.lookup(
UrbanLabourForceToJobsRatio[i]);
AttractivnessOfUrbanJobsMultiplier[i] =
AttractivnessOfUrbanJobsMultiplierTable.lookup(
UrbanLabourForceToJobsRatio[i]);
BusinessLandMultiplier[i] =
BusinessLandMultiplierTable.lookup(
BusinessLandFractionOccupied[i]);
GWUsePerBusinessStructure[i] = BusinessGWUse2001[i] /
initBusinessStructures[i];
SWUsePerBusinessStructure[i] = BusinessSWUse2001[i] /
initBusinessStructures[i];
BusinessSWUseDesired[i] = BusinessStructures[i][1] *
SWUsePerBusinessStructure[i] * WaterUseReductionRatio[i];
BusinessGWUseDesired[i] = BusinessStructures[i][1] *
```

```
GWUsePerBusinessStructure[i] * WaterUseReductionRatio[i];
BusinessSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToUrbanBusiness[i];
BusinessGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToUrbanBusiness[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
BusinessSWUseDesiredToAvailability[i] =
BusinessSWUseDesired[i] /
BusinessSWAvailability[i];
BusinessGWUseDesiredToAvailability[i] =
BusinessGWUseDesired[i] /
BusinessGWAvailability[i];

// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(BusinessSWUseDesiredToAvailability[i])) {
BusinessSWUseDesiredToAvailability[i] = 0.0;
BusinessSWUseEffect[i] = 0.0;
} else {
BusinessSWUseToAvailability[i] = FuzzyMin.lookup(
BusinessSWUseDesiredToAvailability[i]);
BusinessSWUseEffect[i] = WaterUseEffectTable.lookup(
BusinessSWUseToAvailability[i]);
}

if (Double.isNaN(BusinessGWUseDesiredToAvailability[i])) {
BusinessGWUseDesiredToAvailability[i] = 0.0;
BusinessGWUseEffect[i] = 0.0;
} else {
BusinessGWUseToAvailability[i] = FuzzyMin.lookup(
BusinessGWUseDesiredToAvailability[i]);
BusinessGWUseEffect[i] = WaterUseEffectTable.lookup(
BusinessGWUseToAvailability[i]);
}

BusinessSWUse[i] = BusinessSWAvailability[i] *
BusinessSWUseToAvailability[i];
BusinessGWUse[i] = BusinessGWAvailability[i] *
BusinessGWUseToAvailability[i];

BusinessTotalWaterUse[i] = BusinessSWUse[i] +
BusinessGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
BusinessWaterAvailabilityMultiplier[i] =
BusinessSWUseEffect[i] * (BusinessSWUse[i] /
BusinessTotalWaterUse[i]) +
BusinessGWUseEffect[i] * (BusinessGWUse[i] /
BusinessTotalWaterUse[i]);

BusinessConstructionMultiplier[i] =
PerceivedFloodDamageInvestmentMultiplier[i] *
BusinessLabourForceMultiplier[i] *
BusinessLandMultiplier[i] *
BusinessWaterAvailabilityMultiplier[i];
BusinessConstruction[i] = BusinessConstructionMultiplier[i] *
BusinessConstructionNormal[i] *
BusinessStructures[i][1];
BusinessDemolition[i] = BusinessDemolitionNormal[i] *
```

```
                           BusinessStructures[i][1];
                           /*
                            *  URBAN HOUSING SECTOR
                            */
                           UrbanHousingDemolition[i] = UrbanHouses[i][1] *
                           UrbanHousingDemolitionNormal[i];
                           HouseholdsToHousesRatio[i] = UrbanPopulation[i][1] /
                           (UrbanHouses[i][1] * HouseholdSize[i]);
                           AttractivnessOfUrbanHousingMultiplier[i] =
                           AttractivnessOfUrbanHousingMultiplierTable.lookup(
                           HouseholdsToHousesRatio[i]);
                           UrbanHousingAvailabilityMultiplier[i] =
                           UrbanHousingAvailabilityMultiplierTable.lookup(
                           HouseholdsToHousesRatio[i]);
                           UrbanHousingLandMultiplier[i] =
                           UrbanHousingLandMultiplierTable.lookup(
                           ResidentialLandFractionOccupied[i]);
                           HousingConstructionMultiplier[i] =
                           UrbanHousingLandMultiplier[i] *
                           UrbanHousingAvailabilityMultiplier[i];
                           UrbanHousingConstruction[i] = UrbanHousingConstructionNormal[i] *
                           UrbanHouses[i][1] * HousingConstructionMultiplier[i];

                           /*
                            *  URBAN POPULATION SECTOR
                            */
                           GWUsePerUrbanPerson[i] = UrbanResidentialGWUse2001[i] /
                           initUrbanPopulation[i];
                           SWUsePerUrbanPerson[i] = UrbanResidentialSWUse2001[i] /
                           initUrbanPopulation[i];
                           UrbanResidentialSWUseDesired[i] = UrbanPopulation[i][1] *
                           SWUsePerUrbanPerson[i] * WaterUseReductionRatio[i];
                           UrbanResidentialGWUseDesired[i] = UrbanPopulation[i][1] *
                           GWUsePerUrbanPerson[i] * WaterUseReductionRatio[i];
                           UrbanResidentialSWAvailability[i] = MaxAllowableSWPumping[i] *
                           FractionTotalSWAllocatedToUrbanPopulation[i];
                           UrbanResidentialGWAvailability[i] = PerceivedGWRecharge[i] *
                           FractionTotalGWAllocatedToUrbanPopulation[i];

                           // in case denominator is zero, the result is infinity
                           // and so we'll set it to zero
                           UrbanResidentialSWUseDesiredToAvailability[i] =
                           UrbanResidentialSWUseDesired[i] /
                           UrbanResidentialSWAvailability[i];
                           UrbanResidentialGWUseDesiredToAvailability[i] =
                           UrbanResidentialGWUseDesired[i] /
                           UrbanResidentialGWAvailability[i];

                           // applies the FuzzyMin table because it is possible
                           // for desired water use to be larger than available;
                           // this just makes sure the system goes to a smooth
                           // transition to the available
                           if (Double.isNaN(UrbanResidentialSWUseDesiredToAvailability[i])) {
                           UrbanResidentialSWUseDesiredToAvailability[i] = 0.0;
                           UrbanResidentialSWUseEffect[i] = 0.0;
                           } else {
                           UrbanResidentialSWUseToAvailability[i] = FuzzyMin.lookup(
                           UrbanResidentialSWUseDesiredToAvailability[i]);
                           UrbanResidentialSWUseEffect[i] = WaterUseEffectTable.lookup(
                           UrbanResidentialSWUseToAvailability[i]);
                           }

                           if (Double.isNaN(UrbanResidentialGWUseDesiredToAvailability[i])) {
                           UrbanResidentialGWUseDesiredToAvailability[i] = 0.0;
```

```
UrbanResidentialGWUseEffect[i] = 0.0;
} else {
UrbanResidentialGWUseToAvailability[i] = FuzzyMin.lookup(
UrbanResidentialGWUseDesiredToAvailability[i]);
UrbanResidentialGWUseEffect[i] = WaterUseEffectTable.lookup(
UrbanResidentialGWUseToAvailability[i]);
}

UrbanResidentialSWUse[i] = UrbanResidentialSWAvailability[i] *
UrbanResidentialSWUseToAvailability[i];
UrbanResidentialGWUse[i] = UrbanResidentialGWAvailability[i] *
UrbanResidentialGWUseToAvailability[i];

UrbanResidentialTotalWaterUse[i] = UrbanResidentialSWUse[i] +
UrbanResidentialGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
UrbanResidentialWaterAvailabilityAttractivness[i] =
UrbanResidentialSWUseEffect[i] * (UrbanResidentialSWUse[i] /
UrbanResidentialTotalWaterUse[i]) +
UrbanResidentialGWUseEffect[i] * (UrbanResidentialGWUse[i] /
UrbanResidentialTotalWaterUse[i]);

UrbanAttractivnessMultiplier[i] = AttractivnessOfUrbanHousingMultiplier[i] *
AttractivnessOfUrbanJobsMultiplier[i] *
UrbanResidentialWaterAvailabilityAttractivness[i];
UrbanInMigration[i] = UrbanInMigrationNormal[i] * UrbanPopulation[i][1] *
UrbanAttractivnessMultiplier[i];
UrbanBirths[i] = UrbanBirthsNormal[i] * UrbanPopulation[i][1];
UrbanDeaths[i] = UrbanDeathsNormal[i] * UrbanPopulation[i][1];
UrbanOutMigration[i] = UrbanOutMigrationNormal[i] * UrbanPopulation[i][1];

/*
 *  LAND USE SECTOR
 */
AgriculturalToBusinessRezoningMultiplier[i] =
AgriculturalToBusinessRezoningMultiplierTable.lookup(
BusinessLandFractionOccupied[i]);
AgriculturalToBusinessRezoning[i] = AgriculturalLand[i][1] *
AgriculturalToBusinessRezoningNormal[i] *
AgriculturalToBusinessRezoningMultiplier[i];

AgriculturalToResidentialRezoningMultiplier[i] =
AgriculturalToResidentialRezoningMultiplierTable.lookup(
ResidentialLandFractionOccupied[i]);
AgriculturalToResidentialRezoning[i] = AgriculturalLand[i][1] *
AgriculturalToResidentialRezoningNormal[i] *
AgriculturalToResidentialRezoningMultiplier[i];

ResidentialLandAvailabilityMultiplier[i] =
ResidentialLandAvailabilityMultiplierTable.lookup(
ResidentialLandFractionOccupied[i]);
BusinessLandAvailabilityMultiplier[i] =
BusinessLandAvailabilityMultiplierTable.lookup(
BusinessLandFractionOccupied[i]);

BusinessCommunityPressureMultiplier[i] =
BusinessCommunityPressureMultiplierTable.lookup(
UrbanLabourForceToJobsRatio[i]);
ResidentialCommunityPressureMultiplier[i] =
ResidentialCommunityPressureMultiplierTable.lookup(
HouseholdsToHousesRatio[i]);
```

```
ResidentialLandRezoningPressure[i] =
BusinessCommunityPressureMultiplier[i] *
ResidentialCommunityPressureMultiplier[i];

resLandRezPres[i].update(ResidentialLandRezoningPressure[i],
ResidentialLandRezoningPressureAveragingTime[i]);

ResidentialLandRezoningPressureAverage[i] =
resLandRezPres[i].getOutput();

BusinessLandRezoningMultiplier[i] =
BusinessLandRezoningMultiplierTable.lookup(
ResidentialLandRezoningPressureAverage[i]);
ResidentialLandRezoningMultiplier[i] =
ResidentialLandRezoningMultiplierTable.lookup(
ResidentialLandRezoningPressureAverage[i]);

ResidentialLandFractionRezoned[i] =
ResidentialLandRezoningNormal[i] *
ResidentialLandAvailabilityMultiplier[i] *
ResidentialLandRezoningMultiplier[i];
BusinessLandFractionRezoned[i] =
BusinessLandRezoningNormal[i] *
BusinessLandAvailabilityMultiplier[i] *
BusinessLandRezoningMultiplier[i];

ResidentialLandRezoning[i] =
ResidentialLand[i][1] *
ResidentialLandFractionRezoned[i];

BusinessLandRezoning[i] =
BusinessLand[i][1] *
BusinessLandFractionRezoned[i];

ForestToAgriculturalRezoningMultiplier[i] =
ForestToAgriculturalRezoningMultiplierTable.lookup(
FractionOccupiedByFarmUnits[i]);
ForestToAgriculturalRezoning[i] = ForestCover[i][1] *
ForestToAgriculturalRezoningNormal[i] *
ForestToAgriculturalRezoningMultiplier[i];

/*
 *  RURAL POPULATION SECTOR
 */
GWUsePerRuralPerson[i] = RuralResidentialGWUse2001[i] /
initRuralPopulation[i];
SWUsePerRuralPerson[i] = RuralResidentialSWUse2001[i] /
initRuralPopulation[i];
RuralResidentialSWUseDesired[i] = RuralPopulation[i][1] *
SWUsePerRuralPerson[i] * WaterUseReductionRatio[i];
RuralResidentialGWUseDesired[i] = RuralPopulation[i][1] *
GWUsePerRuralPerson[i] * WaterUseReductionRatio[i];
RuralResidentialSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToRuralResidential[i];
RuralResidentialGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToRuralResidential[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
RuralResidentialSWUseDesiredToAvailability[i] =
RuralResidentialSWUseDesired[i] /
RuralResidentialSWAvailability[i];
RuralResidentialGWUseDesiredToAvailability[i] =
RuralResidentialGWUseDesired[i] /
```

```
RuralResidentialGWAvailability[i];

// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(RuralResidentialSWUseDesiredToAvailability[i])) {
RuralResidentialSWUseDesiredToAvailability[i] = 0.0;
RuralResidentialSWUseEffect[i] = 0.0;
} else {
RuralResidentialSWUseToAvailability[i] = FuzzyMin.lookup(
RuralResidentialSWUseDesiredToAvailability[i]);
RuralResidentialSWUseEffect[i] = WaterUseEffectTable.lookup(
RuralResidentialSWUseToAvailability[i]);
}

if (Double.isNaN(RuralResidentialGWUseDesiredToAvailability[i])) {
RuralResidentialGWUseDesiredToAvailability[i] = 0.0;
RuralResidentialGWUseEffect[i] = 0.0;
} else {
RuralResidentialGWUseToAvailability[i] = FuzzyMin.lookup(
RuralResidentialGWUseDesiredToAvailability[i]);
RuralResidentialGWUseEffect[i] = WaterUseEffectTable.lookup(
RuralResidentialGWUseToAvailability[i]);
}

RuralResidentialSWUse[i] = RuralResidentialSWAvailability[i] *
RuralResidentialSWUseToAvailability[i];
RuralResidentialGWUse[i] = RuralResidentialGWAvailability[i] *
RuralResidentialGWUseToAvailability[i];

RuralResidentialTotalWaterUse[i] = RuralResidentialSWUse[i] +
RuralResidentialGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
RuralResidentialWaterAvailabilityAttractivness[i] =
RuralResidentialSWUseEffect[i] * (RuralResidentialSWUse[i] /
RuralResidentialTotalWaterUse[i]) +
RuralResidentialGWUseEffect[i] * (RuralResidentialGWUse[i] /
RuralResidentialTotalWaterUse[i]);

RuralAttractivnessMultiplier[i] =
AttractivnessOfRuralJobsMultiplier[i] *
RuralResidentialWaterAvailabilityAttractivness[i];
RuralInMigration[i] = RuralInMigrationNormal[i] * RuralPopulation[i][1] *
RuralAttractivnessMultiplier[i];
RuralBirths[i] = RuralBirthsNormal[i] * RuralPopulation[i][1];
RuralDeaths[i] = RuralDeathsNormal[i] * RuralPopulation[i][1];
RuralOutMigration[i] = RuralOutMigrationNormal[i] * RuralPopulation[i][1];

/*
 *  RURAL BUSINESS ACTIVITY SECTOR
 */
RuralLabourForceMultiplier[i] = RuralLabourForceMultiplierTable.lookup(
RuralLabourForceToJobsRatio[i]);

FractionOccupiedByCropLand[i] = 1.0 -
FractionOccupiedByFarmUnits[i];
AvailabilityOfCropLandMultiplier[i] =
AvailabilityOfCropLandMultiplierTable.lookup(
FractionOccupiedByCropLand[i]);
LandYieldInvestmentMultiplier[i] =
AvailabilityOfCropLandMultiplier[i];
```

```
GWUsePerFarmUnit[i] = FarmGWUse2001[i] / initFarmUnits[i];
SWUsePerFarmUnit[i] = FarmSWUse2001[i] / initFarmUnits[i];
FarmSWUseDesired[i] = FarmUnits[i][1] * SWUsePerFarmUnit[i] *
WaterUseReductionRatio[i];
FarmGWUseDesired[i] = FarmUnits[i][1] * GWUsePerFarmUnit[i] *
WaterUseReductionRatio[i];
FarmSWAvailability[i] = MaxAllowableSWPumping[i] *
FractionTotalSWAllocatedToFarmUnits[i];
FarmGWAvailability[i] = PerceivedGWRecharge[i] *
FractionTotalGWAllocatedToFarmUnits[i];

// in case denominator is zero, the result is infinity
// and so we'll set it to zero
FarmSWUseDesiredToAvailability[i] =
FarmSWUseDesired[i] / FarmSWAvailability[i];
FarmGWUseDesiredToAvailability[i] =
FarmGWUseDesired[i] / FarmGWAvailability[i];

// applies the FuzzyMin table because it is possible
// for desired water use to be larger than available;
// this just makes sure the system goes to a smooth
// transition to the available
if (Double.isNaN(FarmSWUseDesiredToAvailability[i])) {
FarmSWUseDesiredToAvailability[i] = 0.0;
FarmSWUseEffect[i] = 0.0;
} else {
FarmSWUseToAvailability[i] = FuzzyMin.lookup(
FarmSWUseDesiredToAvailability[i]);
FarmSWUseEffect[i] = WaterUseEffectTable.lookup(
FarmSWUseToAvailability[i]);
}

if (Double.isNaN(FarmGWUseDesiredToAvailability[i])) {
FarmGWUseDesiredToAvailability[i] = 0.0;
FarmGWUseEffect[i] = 0.0;
} else {
FarmGWUseToAvailability[i] = FuzzyMin.lookup(
FarmGWUseDesiredToAvailability[i]);
FarmGWUseEffect[i] = WaterUseEffectTable.lookup(
FarmGWUseToAvailability[i]);
}

FarmSWUse[i] = FarmSWAvailability[i] * FarmSWUseToAvailability[i];
FarmGWUse[i] = FarmGWAvailability[i] * FarmGWUseToAvailability[i];

FarmTotalWaterUse[i] = FarmSWUse[i] + FarmGWUse[i];

// this multiplier is a weighted average of both SW and GW
// used in each county
FarmWaterAvailabilityMultiplier[i] =
FarmSWUseEffect[i] * (FarmSWUse[i] /
FarmTotalWaterUse[i]) +
FarmGWUseEffect[i] * (FarmGWUse[i] /
FarmTotalWaterUse[i]);

FarmUnitInvestmentMultiplier[i] =
PerceivedFloodDamageInvestmentMultiplier[i] *
RuralLabourForceMultiplier[i] *
FarmWaterAvailabilityMultiplier[i] *
LandYieldInvestmentMultiplier[i];
FarmUnitInvestment[i] = FarmUnits[i][1] *
FarmUnitInvestmentNormal[i] *
FarmUnitInvestmentMultiplier[i];
```

```
FarmUnitDepreciation[i] = FarmUnits[i][1] *
FarmUnitDepreciationNormal[i];

// total county water use
TotalGWUse[i] = BusinessGWUse[i] + UrbanResidentialGWUse[i] +
RuralResidentialGWUse[i] + FarmGWUse[i];

TotalSWUse[i] = BusinessSWUse[i] + UrbanResidentialSWUse[i] +
RuralResidentialSWUse[i] + FarmSWUse[i];

// reset the state variables
BusinessStructures[i][0] = BusinessStructures[i][1];
UrbanHouses[i][0] = UrbanHouses[i][1];
UrbanPopulation[i][0] = UrbanPopulation[i][1];
ForestCover[i][0] = ForestCover[i][1];
AgriculturalLand[i][0] = AgriculturalLand[i][1];
ResidentialLand[i][0] = ResidentialLand[i][1];
BusinessLand[i][0] = BusinessLand[i][1];
RuralPopulation[i][0] = RuralPopulation[i][1];
FarmUnits[i][0] = FarmUnits[i][1];

}


//}}}
}


// methods that set Damage, GWRecharge, Flow and PPT for each county
//{{{
/**
 *  Sets the damage attribute of the SysModel object
 *
 * @param  DamageMiddlesex  In units of [$1,000/yr]
 * @param  DamageOxford     In units of [$1,000/yr]
 * @param  DamagePerth      In units of [$1,000/yr]
 */
public void setDamage(double DamageMiddlesex,
double DamageOxford, double DamagePerth) {
this.Damage[0] = DamageMiddlesex;
this.Damage[1] = DamageOxford;
this.Damage[2] = DamagePerth;

}


/**
 *  Sets the GWRecharge attribute of the SysModel object
 *
 * @param  GWRechargeMiddlesex  In units of [m<SUP>3</SUP>/yr]
 * @param  GWRechargeOxford     In units of [m<SUP>3</SUP>/yr]
 * @param  GWRechargePerth      In units of [m<SUP>3</SUP>/yr]
 */
public void setGWRecharge(double GWRechargeMiddlesex,
double GWRechargeOxford, double GWRechargePerth) {
this.GWRecharge[0] = GWRechargeMiddlesex;
this.GWRecharge[1] = GWRechargeOxford;
this.GWRecharge[2] = GWRechargePerth;
}


/**
 *  Sets the flow attribute of the SysModel object
 *
 * @param  FlowMiddlesex  The new flow value, [m<SUP>3</SUP>/s]
```

```
 * @param  FlowOxford     The new flow value, [m<SUP>3</SUP>/s]
 * @param  FlowPerth      The new flow value, [m<SUP>3</SUP>/s]
 */
public void setFlow(double FlowMiddlesex, double FlowOxford,
double FlowPerth) {
this.Flow[0] = FlowMiddlesex;
this.Flow[1] = FlowOxford;
this.Flow[2] = FlowPerth;
}


/**
 *  Sets the PPT attribute of the SysModel object
 *
 * @param  PPTMiddlesex  The new PPT value, in [mm]
 * @param  PPTOxford     The new PPT value, in [mm]
 * @param  PPTPerth      The new PPT value, in [mm]
 */
public void setPPT(double PPTMiddlesex, double PPTOxford,
double PPTPerth) {
this.PPT[0] = PPTMiddlesex;
this.PPT[1] = PPTOxford;
this.PPT[2] = PPTPerth;
}


//}}}

// methods that calculate fraction of paved and vegetative land
//{{{
public double getFractionPavedLandMiddlesex() {
return (LandPerBusinessStructure[0] * BusinessStructures[0][0] +
LandPerHouse[0] * UrbanHouses[0][0]) /
(ForestCover[0][0] + AgriculturalLand[0][0] +
BusinessLand[0][0] + ResidentialLand[0][0]);
}

public double getFractionPavedLandOxford() {
return (LandPerBusinessStructure[1] * BusinessStructures[1][0] +
LandPerHouse[1] * UrbanHouses[1][0]) /
(ForestCover[1][0] + AgriculturalLand[1][0] +
BusinessLand[1][0] + ResidentialLand[1][0]);
}

public double getFractionPavedLandPerth() {
return (LandPerBusinessStructure[2] * BusinessStructures[2][0] +
LandPerHouse[2] * UrbanHouses[2][0]) /
(ForestCover[2][0] + AgriculturalLand[2][0] +
BusinessLand[2][0] + ResidentialLand[2][0]);
}

public double getFractionVegetationMiddlesex() {
return (ForestCover[0][0] + AgriculturalLand[0][0]) /
(ForestCover[0][0] + AgriculturalLand[0][0] +
BusinessLand[0][0] + ResidentialLand[0][0]);
}

public double getFractionVegetationOxford() {
return (ForestCover[1][0] + AgriculturalLand[1][0]) /
(ForestCover[1][0] + AgriculturalLand[1][0] +
BusinessLand[1][0] + ResidentialLand[1][0]);
}

public double getFractionVegetationPerth() {
```

```
return (ForestCover[2][0] + AgriculturalLand[2][0]) /
(ForestCover[2][0] + AgriculturalLand[2][0] +
BusinessLand[2][0] + ResidentialLand[2][0]);
}
//}}}

// these methods output its values to the main, so that they can be
// written to a file as final output of the System Dynamics model
//{{{

// urban business
public double getBusStrMiddlesex() {
return BusinessStructures[0][0] / 1.0E3;
}

public double getBusStrOxford() {
return BusinessStructures[1][0] / 1.0E3;
}

public double getBusStrPerth() {
return BusinessStructures[2][0] / 1.0E3;
}

public double getBusinessLandMultiplierMiddlesex() {
return BusinessLandMultiplier[0];
}

public double getBusinessLandMultiplierOxford() {
return BusinessLandMultiplier[1];
}

public double getBusinessLandMultiplierPerth() {
return BusinessLandMultiplier[2];
}

public double getBusinessLabourForceMultiplierMiddlesex() {
return BusinessLabourForceMultiplier[0];
}

public double getBusinessLabourForceMultiplierOxford() {
return BusinessLabourForceMultiplier[1];
}

public double getBusinessLabourForceMultiplierPerth() {
return BusinessLabourForceMultiplier[2];
}

public double getBusinessWaterAvailabilityMultiplierMiddlesex() {
return BusinessWaterAvailabilityMultiplier[0];
}

public double getBusinessWaterAvailabilityMultiplierOxford() {
return BusinessWaterAvailabilityMultiplier[1];
}

public double getBusinessWaterAvailabilityMultiplierPerth() {
return BusinessWaterAvailabilityMultiplier[2];
}


// urban housing
public double getUrbHouMiddlesex() {
return UrbanHouses[0][0] / 1.0E3;
}
```

```
public double getUrbHouOxford() {
return UrbanHouses[1][0] / 1.0E3;
}

public double getUrbHouPerth() {
return UrbanHouses[2][0] / 1.0E3;
}

public double getUrbanHousingAvailabilityMultiplierMiddlesex() {
return UrbanHousingAvailabilityMultiplier[0];
}

public double getUrbanHousingAvailabilityMultiplierOxford() {
return UrbanHousingAvailabilityMultiplier[1];
}

public double getUrbanHousingAvailabilityMultiplierPerth() {
return UrbanHousingAvailabilityMultiplier[2];
}

public double getUrbanHousingLandMultiplierMiddlesex() {
return UrbanHousingLandMultiplier[0];
}

public double getUrbanHousingLandMultiplierOxford() {
return UrbanHousingLandMultiplier[1];
}

public double getUrbanHousingLandMultiplierPerth() {
return UrbanHousingLandMultiplier[2];
}


// urban population
public double getUrbPopMiddlesex() {
return UrbanPopulation[0][0] / 1.0E3;
}

public double getUrbPopOxford() {
return UrbanPopulation[1][0] / 1.0E3;
}

public double getUrbPopPerth() {
return UrbanPopulation[2][0] / 1.0E3;
}

public double getAttractivnessOfUrbanJobsMultiplierMiddlesex() {
return AttractivnessOfUrbanJobsMultiplier[0];
}

public double getAttractivnessOfUrbanJobsMultiplierOxford() {
return AttractivnessOfUrbanJobsMultiplier[1];
}

public double getAttractivnessOfUrbanJobsMultiplierPerth() {
return AttractivnessOfUrbanJobsMultiplier[2];
}

public double getAttractivnessOfUrbanHousingMultiplierMiddlesex() {
return AttractivnessOfUrbanHousingMultiplier[0];
}

public double getAttractivnessOfUrbanHousingMultiplierOxford() {
```

```
return AttractivnessOfUrbanHousingMultiplier[1];
}

public double getAttractivnessOfUrbanHousingMultiplierPerth() {
return AttractivnessOfUrbanHousingMultiplier[2];
}

public double getUrbanResidentialWaterAvailabilityAttractivnessMiddlesex() {
return UrbanResidentialWaterAvailabilityAttractivness[0];
}

public double getUrbanResidentialWaterAvailabilityAttractivnessOxford() {
return UrbanResidentialWaterAvailabilityAttractivness[1];
}

public double getUrbanResidentialWaterAvailabilityAttractivnessPerth() {
return UrbanResidentialWaterAvailabilityAttractivness[2];
}


// land use
public double getForCovMiddlesex() {
return ForestCover[0][0];
}

public double getForCovOxford() {
return ForestCover[1][0];
}

public double getForCovPerth() {
return ForestCover[2][0];
}

public double getAgLandMiddlesex() {
return AgriculturalLand[0][0];
}

public double getAgLandOxford() {
return AgriculturalLand[1][0];
}

public double getAgLandPerth() {
return AgriculturalLand[2][0];
}

public double getResLandMiddlesex() {
return ResidentialLand[0][0];
}

public double getResLandOxford() {
return ResidentialLand[1][0];
}

public double getResLandPerth() {
return ResidentialLand[2][0];
}

public double getBusLandMiddlesex() {
return BusinessLand[0][0];
}

public double getBusLandOxford() {
return BusinessLand[1][0];
}
```

```
public double getBusLandPerth() {
return BusinessLand[2][0];
}


// rural business
public double getFarmsMiddlesex() {
return FarmUnits[0][0] / 1.0E3;
}

public double getFarmsOxford() {
return FarmUnits[1][0] / 1.0E3;
}

public double getFarmsPerth() {
return FarmUnits[2][0] / 1.0E3;
}

public double getAvailabilityOfCropLandMultiplierMiddlesex() {
return AvailabilityOfCropLandMultiplier[0];
}

public double getAvailabilityOfCropLandMultiplierOxford() {
return AvailabilityOfCropLandMultiplier[1];
}

public double getAvailabilityOfCropLandMultiplierPerth() {
return AvailabilityOfCropLandMultiplier[2];
}

public double getRuralLabourForceMultiplierMiddlesex() {
return RuralLabourForceMultiplier[0];
}

public double getRuralLabourForceMultiplierOxford() {
return RuralLabourForceMultiplier[1];
}

public double getRuralLabourForceMultiplierPerth() {
return RuralLabourForceMultiplier[2];
}

public double getFarmWaterAvailabilityMultiplierMiddlesex() {
return FarmWaterAvailabilityMultiplier[0];
}

public double getFarmWaterAvailabilityMultiplierOxford() {
return FarmWaterAvailabilityMultiplier[1];
}

public double getFarmWaterAvailabilityMultiplierPerth() {
return FarmWaterAvailabilityMultiplier[2];
}


// rural population
public double getRurPopMiddlesex() {
return RuralPopulation[0][0] / 1.0E3;
}

public double getRurPopOxford() {
return RuralPopulation[1][0] / 1.0E3;
}
```

```
public double getRurPopPerth() {
return RuralPopulation[2][0] / 1.0E3;
}

public double getAttractivnessOfRuralJobsMultiplierMiddlesex() {
return AttractivnessOfRuralJobsMultiplier[0];
}

public double getAttractivnessOfRuralJobsMultiplierOxford() {
return AttractivnessOfRuralJobsMultiplier[1];
}

public double getAttractivnessOfRuralJobsMultiplierPerth() {
return AttractivnessOfRuralJobsMultiplier[2];
}

public double getRuralResidentialWaterAvailabilityAttractivnessMiddlesex() {
return RuralResidentialWaterAvailabilityAttractivness[0];
}

public double getRuralResidentialWaterAvailabilityAttractivnessOxford() {
return RuralResidentialWaterAvailabilityAttractivness[1];
}

public double getRuralResidentialWaterAvailabilityAttractivnessPerth() {
return RuralResidentialWaterAvailabilityAttractivness[2];
}


// water
public double getDroughtLevelMiddlesex() {
return DroughtLevel[0];
}

public double getDroughtLevelOxford() {
return DroughtLevel[1];
}

public double getDroughtLevelPerth() {
return DroughtLevel[2];
}

public double getPerceivedDroughtLevelMiddlesex() {
return PerceivedDroughtLevel[0];
}

public double getPerceivedDroughtLevelOxford() {
return PerceivedDroughtLevel[1];
}

public double getPerceivedDroughtLevelPerth() {
return PerceivedDroughtLevel[2];
}

public double getWaterUseReductionRatioMiddlesex() {
return WaterUseReductionRatio[0];
}

public double getWaterUseReductionRatioOxford() {
return WaterUseReductionRatio[1];
}

public double getWaterUseReductionRatioPerth() {
```

```java
    return WaterUseReductionRatio[2];
}

public double getPerceivedGWRechargeMiddlesex() {
    return PerceivedGWRecharge[0] / 1.0E6;
}

public double getGWRechargeMiddlesex() {
    return GWRecharge[0] / 1.0E6;
}

public double getPerceivedGWRechargeOxford() {
    return PerceivedGWRecharge[1] / 1.0E6;
}

public double getGWRechargeOxford() {
    return GWRecharge[1] / 1.0E6;
}

public double getPerceivedGWRechargePerth() {
    return PerceivedGWRecharge[2] / 1.0E6;
}

public double getGWRechargePerth() {
    return GWRecharge[2] / 1.0E6;
}

public double getMaxAllowableSWPumpingMiddlesex() {
    return MaxAllowableSWPumping[0] / 1.0E6;
}

public double getMaxAllowableSWPumpingOxford() {
    return MaxAllowableSWPumping[1] / 1.0E6;
}

public double getMaxAllowableSWPumpingPerth() {
    return MaxAllowableSWPumping[2] / 1.0E6;
}

public double getFloodDamageMultiplierMiddlesex() {
    return this.FloodDamageInvestmentMultiplier[0];
}

public double getPerceivedFloodDamageMultiplierMiddlesex() {
    return this.PerceivedFloodDamageInvestmentMultiplier[0];
}

public double getFloodDamageMultiplierOxford() {
    return this.FloodDamageInvestmentMultiplier[1];
}

public double getPerceivedFloodDamageMultiplierOxford() {
    return this.PerceivedFloodDamageInvestmentMultiplier[1];
}

public double getFloodDamageMultiplierPerth() {
    return this.FloodDamageInvestmentMultiplier[2];
}

public double getPerceivedFloodDamageMultiplierPerth() {
    return this.PerceivedFloodDamageInvestmentMultiplier[2];
}

public double getBusinessSWUseMiddlesex() {
```

```java
return this.BusinessSWUse[0] / 1.0E6;
}

public double getBusinessSWAvailabilityMiddlesex() {
return this.BusinessSWAvailability[0] / 1.0E6;
}

public double getBusinessGWUseMiddlesex() {
return this.BusinessGWUse[0] / 1.0E6;
}

public double getBusinessGWAvailabilityMiddlesex() {
return this.BusinessGWAvailability[0] / 1.0E6;
}

public double getBusinessGWUseOxford() {
return this.BusinessGWUse[1] / 1.0E6;
}

public double getBusinessSWAvailabilityOxford() {
return this.BusinessGWAvailability[1] / 1.0E6;
}

public double getBusinessGWUsePerth() {
return this.BusinessGWUse[2] / 1.0E6;
}

public double getBusinessSWAvailabilityPerth() {
return this.BusinessGWAvailability[2] / 1.0E6;
}

public double getTotalGWUseMiddlesex(){
return this.TotalGWUse[0] / 1.0E6;
}

public double getTotalGWUseOxford(){
return this.TotalGWUse[1] / 1.0E6;
}

public double getTotalGWUsePerth(){
return this.TotalGWUse[2] / 1.0E6;
}

public double getTotalSWUseMiddlesex(){
return this.TotalSWUse[0] / 1.0E6;
}

public double getTotalSWUseOxford(){
return this.TotalSWUse[1] / 1.0E6;
}

public double getTotalSWUsePerth(){
return this.TotalSWUse[2] / 1.0E6;
}

//}}}

}
```

D.5 AnnualSeries2.java

```java
import java.io.*;
/**
 *  This class is the same as AnnualSeries, except that it outputs the results
 * one station at the time, and it does not compute timing and regularity of the
 * the flows.  This is being used simply to be able to use the
 * fitStatisticalDistribution() method from the contUtils class within main.
 *
 * @author     Predrag Prodanovic
 * @created    June 15, 2006
 */
public class AnnualSeries2 {

// set up a DataWriter objects
DataWriter outFile;
String fileName;

// flow instance vars
/*
 *  jnMitchellDaily = 0
 *  jnAvonDaily = 1
 *  jnDownStreamWildwood = 2
 *  jnStMarysDaily = 3
 *  jnPloverMillsDaily = 4
 *  jnDownStreamFanshawe = 5
 *  jnMedwayDaily = 6
 *  jnInnerkipDaily = 7
 *  jnDownStreamPittock = 8
 *  jnCedarDaily = 9
 *  jnIngersollDaily = 10
 *  jnThamesfordDaily = 11
 *  jnReynoldsDaily = 12
 *  jnWaubunoDaily = 13
 *  jnEalingDaily = 14
 *  jnByronDaily = 15
 *  jnOxbowDaily = 16
 *  jnDingmanDaily = 17
 */
// the junctions corresponding to the array locations are shown above
double[] jnOut = new double[18];

// annual maximums
double[] jnOutMax = new double[18];

// annual minimums
double[] jnOutMin = new double[18];

// theta values (used for computation of timing and regularity)
// see Cunderlik and Burn (2006) paper in Water Resources Research, v.42
int yearCount = 0;
double[] theta = new double[18];
double[] sumX = new double[18];
double[] sumY = new double[18];
double[] xBar = new double[18];
double[] yBar = new double[18];
double[] MDF = new double[18];
double[] R = new double[18];

private HydModel utHyd;
private ModelDate currentDate;
private int userTimeStepsInMonth = 0;
```

```
/**
 * Constructor for the AnnualSeries2 object
 *
 * @param   utHyd           HydModel object is ised as input
 * @param   currentDate     The current date
 * @param   fileName        The file name that will be used to output results
 * @param   outputDir       Description of the Parameter
 * @exception   IOException  An Input Output Exception
 */
public AnnualSeries2(HydModel utHyd, ModelDate currentDate,
String outputDir, String fileName) throws IOException {
//{{{

this.utHyd = utHyd;
this.currentDate = currentDate;
this.fileName = fileName;

// to initialize
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
this.jnOutMax[i] = -9999.9;
this.jnOutMin[i] = 9999.9;
this.theta[i] = 0.0;
}
this.outFile = new DataWriter(outputDir + this.fileName);
//}}}
}


/**
 *  This method computes the annual maximum daily flow at all of the
 * stream gauges in the basin.  The outputs of this method are used in
 * the flood flow frequency analysis.
 *
 * @param   utHyd           The Hydrologic model
 * @param   currentDate     The current date
 * @exception   IOException  Input Output Exception
 */
public void updateDaily(HydModel utHyd, ModelDate currentDate)
 throws IOException {
//{{{
this.utHyd = utHyd;
this.currentDate = currentDate;

// accumulates the 6hr hydrological output to daily values
this.jnOut[0] = this.jnOut[0] + this.utHyd.getJnMitchellSG();
this.jnOut[1] = this.jnOut[1] + this.utHyd.getJnAvonSG();
this.jnOut[2] = this.jnOut[2] + this.utHyd.getJnDownStreamWildwood();
this.jnOut[3] = this.jnOut[3] + this.utHyd.getJnStMarysSG();
this.jnOut[4] = this.jnOut[4] + this.utHyd.getJnPloverMillsSG();
this.jnOut[5] = this.jnOut[5] + this.utHyd.getJnDownStreamFanshawe();
this.jnOut[6] = this.jnOut[6] + this.utHyd.getJnMedwaySG();
this.jnOut[7] = this.jnOut[7] + this.utHyd.getJnUpStreamPittock();
this.jnOut[8] = this.jnOut[8] + this.utHyd.getJnDownStreamPittock();
this.jnOut[9] = this.jnOut[9] + this.utHyd.getJnCedarSG();
this.jnOut[10] = this.jnOut[10] + this.utHyd.getJnIngersollSG();
this.jnOut[11] = this.jnOut[11] + this.utHyd.getJnThamesfordSG();
this.jnOut[12] = this.jnOut[12] + this.utHyd.getJnReynoldsSG();
this.jnOut[13] = this.jnOut[13] + this.utHyd.getJnWaubunoSG();
this.jnOut[14] = this.jnOut[14] + this.utHyd.getJnEalingSG();
this.jnOut[15] = this.jnOut[15] + this.utHyd.getJnByronSG();
this.jnOut[16] = this.jnOut[16] + this.utHyd.getJnOxbowSG();
this.jnOut[17] = this.jnOut[17] + this.utHyd.getJnDingmanSG();
```

```
        if (this.currentDate.getHour() >= 19) {

for (int i = 0; i < 18; i++) {
// this averages four 6hr intervals into one
// daily value
this.jnOut[i] = this.jnOut[i] / 4.0;

// finds the maximum
if (this.jnOut[i] > this.jnOutMax[i]) {
this.jnOutMax[i] = this.jnOut[i];

// compute the theta
this.theta[i] = this.currentDate.getDayOfYear() *
2.0 * Math.PI / this.currentDate.getDaysInYear();
}
}

if (this.currentDate.getDayOfYear() ==
this.currentDate.getDaysInYear()) {

computeTimingAndRegularity();
// to output the annual maximum
if (this.fileName == "ByronAnnualMaxDailyFlow.txt") {
this.outFile.writeData(this.jnOutMax[15]);
} else if (this.fileName == "StMarysAnnualMaxDailyFlow.txt") {
this.outFile.writeData(this.jnOutMax[3]);
} else if (this.fileName == "IngersollAnnualMaxDailyFlow.txt") {
this.outFile.writeData(this.jnOutMax[10]);
}

// reset the maximums
for (int i = 0; i < 18; i++) {
this.jnOutMax[i] = -9999.9;
}

}

// resets the flow variable
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
}
}
//}}}
}


/**
 *  This method computes the annual minimum 7 day flow at all of the
 * stream gauges in the basin.  The outputs of this method are used in
 * the low flow frequency analysis.
 *
 * @param   utHyd          The Hydrologic model
 * @param   currentDate     The current date
 * @exception  IOException  Input Output Exception
 */
public void update7Day(HydModel utHyd, ModelDate currentDate)
 throws IOException {
//{{{
this.utHyd = utHyd;
this.currentDate = currentDate;

// accumulates the 6hr hydrological output 7 day averages
this.jnOut[0] = this.jnOut[0] + this.utHyd.getJnMitchellSG();
this.jnOut[1] = this.jnOut[1] + this.utHyd.getJnAvonSG();
```

```
this.jnOut[2] = this.jnOut[2] + this.utHyd.getJnDownStreamWildwood();
this.jnOut[3] = this.jnOut[3] + this.utHyd.getJnStMarysSG();
this.jnOut[4] = this.jnOut[4] + this.utHyd.getJnPloverMillsSG();
this.jnOut[5] = this.jnOut[5] + this.utHyd.getJnDownStreamFanshawe();
this.jnOut[6] = this.jnOut[6] + this.utHyd.getJnMedwaySG();
this.jnOut[7] = this.jnOut[7] + this.utHyd.getJnUpStreamPittock();
this.jnOut[8] = this.jnOut[8] + this.utHyd.getJnDownStreamPittock();
this.jnOut[9] = this.jnOut[9] + this.utHyd.getJnCedarSG();
this.jnOut[10] = this.jnOut[10] + this.utHyd.getJnIngersollSG();
this.jnOut[11] = this.jnOut[11] + this.utHyd.getJnThamesfordSG();
this.jnOut[12] = this.jnOut[12] + this.utHyd.getJnReynoldsSG();
this.jnOut[13] = this.jnOut[13] + this.utHyd.getJnWaubunoSG();
this.jnOut[14] = this.jnOut[14] + this.utHyd.getJnEalingSG();
this.jnOut[15] = this.jnOut[15] + this.utHyd.getJnByronSG();
this.jnOut[16] = this.jnOut[16] + this.utHyd.getJnOxbowSG();
this.jnOut[17] = this.jnOut[17] + this.utHyd.getJnDingmanSG();

if ((this.currentDate.getHour() >= 19) &&
(this.currentDate.getDayOfWeek() == 7)) {

for (int i = 0; i < 18; i++) {
// this averages 28 6hr intervals into one
// 7 day value
this.jnOut[i] = this.jnOut[i] / 28.0;

// finds the minimum
if (this.jnOut[i] < this.jnOutMin[i]) {
this.jnOutMin[i] = this.jnOut[i];

// compute the theta
this.theta[i] = this.currentDate.getDayOfYear() *
2.0 * Math.PI / this.currentDate.getDaysInYear();
}
}

// for the last week in a year
if (this.currentDate.getWeekOfYear() == 52) {

computeTimingAndRegularity();

// outputs the 7 day minimum flow, the timing and
// regularity
//{{{
if (this.fileName == "ByronAnnualMin7DayFlow.txt") {
this.outFile.writeData(this.jnOutMin[15]);
} else if (this.fileName == "StMarysAnnualMin7DayFlow.txt") {
this.outFile.writeData(this.jnOutMin[3]);
} else if (this.fileName == "IngersollAnnualMin7DayFlow.txt") {
this.outFile.writeData(this.jnOutMin[10]);
}
//}}}

// reset the minimums
for (int i = 0; i < 18; i++) {
this.jnOutMin[i] = 9999.9;
}

}

// resets the flow variable
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
}
}
```

```
//}}}
}


/**
 *  This method computes the annual minimum 7 day flow at all of the
 * stream gauges in the basin.  The outputs of this method are used in
 * the low flow frequency analysis.
 *
 * @param   utHyd          The Hydrologic model
 * @param   currentDate     The current date
 * @exception  IOException  Input Output Exception
 */
public void updateMonthly(HydModel utHyd, ModelDate currentDate)
 throws IOException {
//{{{
this.utHyd = utHyd;
this.currentDate = currentDate;

// accumulates the 6hr hydrological output 7 day averages
this.jnOut[0] = this.jnOut[0] + this.utHyd.getJnMitchellSG();
this.jnOut[1] = this.jnOut[1] + this.utHyd.getJnAvonSG();
this.jnOut[2] = this.jnOut[2] + this.utHyd.getJnDownStreamWildwood();
this.jnOut[3] = this.jnOut[3] + this.utHyd.getJnStMarysSG();
this.jnOut[4] = this.jnOut[4] + this.utHyd.getJnPloverMillsSG();
this.jnOut[5] = this.jnOut[5] + this.utHyd.getJnDownStreamFanshawe();
this.jnOut[6] = this.jnOut[6] + this.utHyd.getJnMedwaySG();
this.jnOut[7] = this.jnOut[7] + this.utHyd.getJnUpStreamPittock();
this.jnOut[8] = this.jnOut[8] + this.utHyd.getJnDownStreamPittock();
this.jnOut[9] = this.jnOut[9] + this.utHyd.getJnCedarSG();
this.jnOut[10] = this.jnOut[10] + this.utHyd.getJnIngersollSG();
this.jnOut[11] = this.jnOut[11] + this.utHyd.getJnThamesfordSG();
this.jnOut[12] = this.jnOut[12] + this.utHyd.getJnReynoldsSG();
this.jnOut[13] = this.jnOut[13] + this.utHyd.getJnWaubunoSG();
this.jnOut[14] = this.jnOut[14] + this.utHyd.getJnEalingSG();
this.jnOut[15] = this.jnOut[15] + this.utHyd.getJnByronSG();
this.jnOut[16] = this.jnOut[16] + this.utHyd.getJnOxbowSG();
this.jnOut[17] = this.jnOut[17] + this.utHyd.getJnDingmanSG();

if ((currentDate.getDay() == currentDate.getDaysInMonth()) &&
(currentDate.getHour() >= 19)) {

// computes the number of user times steps this month
this.userTimeStepsInMonth =
this.currentDate.getDaysInMonth() * 4;

for (int i = 0; i < 18; i++) {
// this averages userTimeStepsInMonth 6hr
//intervals into one monthly value
this.jnOut[i] = this.jnOut[i] /
this.userTimeStepsInMonth;

// finds the minimum
if (this.jnOut[i] < this.jnOutMin[i]) {
this.jnOutMin[i] = this.jnOut[i];

// compute the theta
this.theta[i] = this.currentDate.getDayOfYear() *
2.0 * Math.PI / this.currentDate.getDaysInYear();
}
}

if (this.currentDate.getDayOfYear() ==
this.currentDate.getDaysInYear()) {
```

```
computeTimingAndRegularity();

// outputs the monthly minimum flow, the timing and
// regularity
//{{{
if (this.fileName == "ByronAnnualMinMonthlyFlow.txt") {
this.outFile.writeData(this.jnOutMin[15]);
} else if (this.fileName == "StMarysAnnualMinMonthlyFlow.txt") {
this.outFile.writeData(this.jnOutMin[3]);
} else if (this.fileName == "IngersollAnnualMinMonthlyFlow.txt") {
this.outFile.writeData(this.jnOutMin[10]);
}
//}}}

// reset the minimums
for (int i = 0; i < 18; i++) {
this.jnOutMin[i] = 9999.9;
}

}

// resets the flow variable
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
}
}
//}}}
}


/**
 *  This method updates the timing and regularity of the object
 */
private void computeTimingAndRegularity() {
//{{{

// increments the year counted
this.yearCount++;

// updates the MDF and R values (i.e., timing and regularity of
// events)
for (int i = 0; i < 18; i++) {
this.sumX[i] = this.sumX[i] + Math.cos(this.theta[i]);
this.sumY[i] = this.sumY[i] + Math.sin(this.theta[i]);

this.xBar[i] = this.sumX[i] / yearCount;
this.yBar[i] = this.sumY[i] / yearCount;

// note that a century has 36524 days, which works out
// as 365.24 days per year
if ((xBar[i] >= 0) && (yBar[i] >= 0)) {
// quadrant I
this.MDF[i] = Math.atan(this.yBar[i] /
this.xBar[i]) * 365.24 / (2.0 * Math.PI);
} else if ((xBar[i] <= 0) && (yBar[i] >= 0)) {
// quadrant II
this.MDF[i] = (Math.PI - Math.atan(Math.abs(this.yBar[i]) /
Math.abs(this.xBar[i]))) * 365.24 / (2.0 * Math.PI);
} else if ((xBar[i] <= 0) && (yBar[i] <= 0)) {
// quadrant III
this.MDF[i] = (Math.PI + Math.atan(Math.abs(this.yBar[i]) /
Math.abs(this.xBar[i]))) * 365.24 / (2.0 * Math.PI);
} else if ((xBar[i] >= 0) && (yBar[i] <= 0)) {
```

```
// quadrant IV
this.MDF[i] = (2.0 * Math.PI - Math.atan(Math.abs(this.yBar[i]) /
Math.abs(this.xBar[i]))) * 365.24 / (2.0 * Math.PI);
}

this.R[i] = Math.sqrt(Math.pow(this.xBar[i], 2.0) +
Math.pow(this.yBar[i], 2.0));
}
//}}}
}




/**
 *  This method closes the file
 */
public void closeFile() {
outFile.closeFile();
}

}
```

D.6   AnnualSeries.java

```
import java.io.*;

/**
 *  This class extract annual extremes (of flows only) from the Hydrologic
 * portion of the model.  It can compute annual daily maximum, annual 7 day
 * minimum, and annual monthly minimum flows, together with timing and regularity
 * of said flows. Please note that a separate instance of this object should
 * call only one method---this is how the code was written.  This means that is
 * it is wished to perfom a statistical analysis on daily, 7 day, and monthly
 * flows, three separate AnnualSeries objects must be instantiated.  For
 * example, the following code does this:
 *
 * // declares the objects
 * AnnualSeries saDaily = new AnnualSeries(utHyd,
 * currentDate, outputDir + "DailyAnnualMax.txt");
 * AnnualSeries sa7Day = new AnnualSeries(utHyd,
 * currentDate, outputDir + "SevenDayAnnualMin.txt");
 * AnnualSeries saMonthly = new AnnualSeries(utHyd,
 * currentDate, outputDir + "MonthlyAnnualMin.txt");
 *
 * Then, in the loop immediately right after utHyd.update(currentDate), the
 * following code must be called:
 *
 * saDaily.updateDaily(utHyd, currentDate);
 * sa7Day.update7Day(utHyd, currentDate);
 * saMonthly.updateMonthly(utHyd, currentDate);
 *
 * @author     Predrag Prodanovic
 * @created    April 25, 2006
 */
public class AnnualSeries {

// set up a DataWriter objects
// for the output of flows for all stations
DataWriter outFile;

// flow instance vars
/*
 *  jnMitchellDaily = 0
 *  jnAvonDaily = 1
 *  jnDownStreamWildwood = 2
 *  jnStMarysDaily = 3
 *  jnPloverMillsDaily = 4
 *  jnDownStreamFanshawe = 5
 *  jnMedwayDaily = 6
 *  jnInnerkipDaily = 7
 *  jnDownStreamPittock = 8
 *  jnCedarDaily = 9
 *  jnIngersollDaily = 10
 *  jnThamesfordDaily = 11
 *  jnReynoldsDaily = 12
 *  jnWaubunoDaily = 13
 *  jnEalingDaily = 14
 *  jnByronDaily = 15
 *  jnOxbowDaily = 16
 *  jnDingmanDaily = 17
 */
// the junctions corresponding to the array locations are shown above
double[] jnOut = new double[18];

// annual maximums
double[] jnOutMax = new double[18];
```

```java
// annual minimums
double[] jnOutMin = new double[18];

// theta values (used for computation of timing and regularity)
// see Cunderlik and Burn (2006) paper in Water Resources Research, v.42
int yearCount = 0;
double[] theta = new double[18];
double[] sumX = new double[18];
double[] sumY = new double[18];
double[] xBar = new double[18];
double[] yBar = new double[18];
double[] MDF = new double[18];
double[] R = new double[18];

private HydModel utHyd;
private ModelDate currentDate;
private int userTimeStepsInMonth = 0;


/**
 * Constructor for the AnnualSeries object
 *
 * @param   utHyd             HydModel object is ised as input
 * @param   currentDate       The current date
 * @param   fileName          The file name that will be used to output results
 * @exception  IOException  An Input Output Exception
 */
public AnnualSeries(HydModel utHyd, ModelDate currentDate,
String fileName) throws IOException {
//{{{

this.utHyd = utHyd;
this.currentDate = currentDate;

// to initialize
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
this.jnOutMax[i] = -9999.9;
this.jnOutMin[i] = 9999.9;
this.theta[i] = 0.0;
}

this.outFile = new DataWriter(fileName);

// writes the headers to the output files; it outputs the
// variable of interest (Maximum Daily Annual Flow, Minimum Monthly
// Annual Flow, Minimum 7 Day Annual Flow) together with their
// timing and regularity indicators
this.outFile.writeData("Date, MitchellSG, jnAvonSG, jnDownStreamWildwood, " +
"StMarysSG, jnPloverMillsSG, jnDownStreamFanshawe, " +
"jnMedwaySG, jnInnerkipSG, jnDownStreamPittock, " +
"jnCedarSG, jnIngersollSG, jnThamesfordSG, " +
"jnReynoldsSG, jnWaubunoSG, JnEalingSG, jnByronSG, " +
"jnOxbowSG, jnDingmanSG, " +
"MitchellMDF, jnAvonMDF, jnDownStreamWildwoodMDF, " +
"StMarysMDF, jnPloverMillsMDF, jnDownStreamFanshaweMDF, " +
"jnMedwayMDF, jnInnerkipMDF, jnDownStreamPittockMDF, " +
"jnCedarMDF, jnIngersollMDF, jnThamesfordMDF, " +
"jnReynoldsMDF, jnWaubunoMDF, JnEalingMDF, jnByronMDF, " +
"jnOxbowMDF, jnDingmanMDF, " +
"MitchellR, jnAvonR, jnDownStreamWildwood, " +
"StMarysR, jnPloverMillsR, jnDownStreamFanshawe, " +
"jnMedwayR, jnInnerkipR, jnDownStreamPittock, " +
```

```
"jnCedarR, jnIngersollR, jnThamesfordR, " +
"jnReynoldsR, jnWaubunoR, JnEalingR, jnByronR, " +
"jnOxbowR, jnDingmanR");
//}}}
}


/**
 *  This method computes the annual maximum daily flow at all of the
 * stream gauges in the basin.  The outputs of this method are used in
 * the flood flow frequency analysis.
 *
 * @param   utHyd           The Hydrologic model
 * @param   currentDate     The current date
 * @exception  IOException  Input Output Exception
 */
public void updateDaily(HydModel utHyd, ModelDate currentDate)
 throws IOException {
//{{{
this.utHyd = utHyd;
this.currentDate = currentDate;

// accumulates the 6hr hydrological output to daily values
this.jnOut[0] = this.jnOut[0] + this.utHyd.getJnMitchellSG();
this.jnOut[1] = this.jnOut[1] + this.utHyd.getJnAvonSG();
this.jnOut[2] = this.jnOut[2] + this.utHyd.getJnDownStreamWildwood();
this.jnOut[3] = this.jnOut[3] + this.utHyd.getJnStMarysSG();
this.jnOut[4] = this.jnOut[4] + this.utHyd.getJnPloverMillsSG();
this.jnOut[5] = this.jnOut[5] + this.utHyd.getJnDownStreamFanshawe();
this.jnOut[6] = this.jnOut[6] + this.utHyd.getJnMedwaySG();
this.jnOut[7] = this.jnOut[7] + this.utHyd.getJnUpStreamPittock();
this.jnOut[8] = this.jnOut[8] + this.utHyd.getJnDownStreamPittock();
this.jnOut[9] = this.jnOut[9] + this.utHyd.getJnCedarSG();
this.jnOut[10] = this.jnOut[10] + this.utHyd.getJnIngersollSG();
this.jnOut[11] = this.jnOut[11] + this.utHyd.getJnThamesfordSG();
this.jnOut[12] = this.jnOut[12] + this.utHyd.getJnReynoldsSG();
this.jnOut[13] = this.jnOut[13] + this.utHyd.getJnWaubunoSG();
this.jnOut[14] = this.jnOut[14] + this.utHyd.getJnEalingSG();
this.jnOut[15] = this.jnOut[15] + this.utHyd.getJnByronSG();
this.jnOut[16] = this.jnOut[16] + this.utHyd.getJnOxbowSG();
this.jnOut[17] = this.jnOut[17] + this.utHyd.getJnDingmanSG();

if (this.currentDate.getHour() >= 19) {

for (int i = 0; i < 18; i++) {
// this averages four 6hr intervals into one
// daily value
this.jnOut[i] = this.jnOut[i] / 4.0;

// finds the maximum
if (this.jnOut[i] > this.jnOutMax[i]) {
this.jnOutMax[i] = this.jnOut[i];

// compute the theta
this.theta[i] = this.currentDate.getDayOfYear() *
2.0 * Math.PI / this.currentDate.getDaysInYear();
}
}

if (this.currentDate.getDayOfYear() ==
this.currentDate.getDaysInYear()) {

computeTimingAndRegularity();
// to output the annual maximum, the timing and
```

```
// regularity
//{{{
this.outFile.writeData(currentDate.getDate() + ", " +
this.jnOutMax[0] + ", " +
this.jnOutMax[1] + ", " +
this.jnOutMax[2] + ", " +
this.jnOutMax[3] + ", " +
this.jnOutMax[4] + ", " +
this.jnOutMax[5] + ", " +
this.jnOutMax[6] + ", " +
this.jnOutMax[7] + ", " +
this.jnOutMax[8] + ", " +
this.jnOutMax[9] + ", " +
this.jnOutMax[10] + ", " +
this.jnOutMax[11] + ", " +
this.jnOutMax[12] + ", " +
this.jnOutMax[13] + ", " +
this.jnOutMax[14] + ", " +
this.jnOutMax[15] + ", " +
this.jnOutMax[16] + ", " +
this.jnOutMax[17] + ", " +
this.MDF[0] + ", " +
this.MDF[1] + ", " +
this.MDF[2] + ", " +
this.MDF[3] + ", " +
this.MDF[4] + ", " +
this.MDF[5] + ", " +
this.MDF[6] + ", " +
this.MDF[7] + ", " +
this.MDF[8] + ", " +
this.MDF[9] + ", " +
this.MDF[10] + ", " +
this.MDF[11] + ", " +
this.MDF[12] + ", " +
this.MDF[13] + ", " +
this.MDF[14] + ", " +
this.MDF[15] + ", " +
this.MDF[16] + ", " +
this.MDF[17] + ", " +
this.R[0] + ", " +
this.R[1] + ", " +
this.R[2] + ", " +
this.R[3] + ", " +
this.R[4] + ", " +
this.R[5] + ", " +
this.R[6] + ", " +
this.R[7] + ", " +
this.R[8] + ", " +
this.R[9] + ", " +
this.R[10] + ", " +
this.R[11] + ", " +
this.R[12] + ", " +
this.R[13] + ", " +
this.R[14] + ", " +
this.R[15] + ", " +
this.R[16] + ", " +
this.R[17]);
//}}}

// reset the maximums
for (int i = 0; i < 18; i++) {
this.jnOutMax[i] = -9999.9;
}
```

```
}

// resets the flow variable
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
}
}
//}}}
}


/**
 *  This method computes the annual minimum 7 day flow at all of the
 * stream gauges in the basin.  The outputs of this method are used in
 * the low flow frequency analysis.
 *
 * @param   utHyd           The Hydrologic model
 * @param   currentDate     The current date
 * @exception  IOException  Input Output Exception
 */
public void update7Day(HydModel utHyd, ModelDate currentDate)
 throws IOException {
//{{{
this.utHyd = utHyd;
this.currentDate = currentDate;

// accumulates the 6hr hydrological output 7 day averages
this.jnOut[0] = this.jnOut[0] + this.utHyd.getJnMitchellSG();
this.jnOut[1] = this.jnOut[1] + this.utHyd.getJnAvonSG();
this.jnOut[2] = this.jnOut[2] + this.utHyd.getJnDownStreamWildwood();
this.jnOut[3] = this.jnOut[3] + this.utHyd.getJnStMarysSG();
this.jnOut[4] = this.jnOut[4] + this.utHyd.getJnPloverMillsSG();
this.jnOut[5] = this.jnOut[5] + this.utHyd.getJnDownStreamFanshawe();
this.jnOut[6] = this.jnOut[6] + this.utHyd.getJnMedwaySG();
this.jnOut[7] = this.jnOut[7] + this.utHyd.getJnUpStreamPittock();
this.jnOut[8] = this.jnOut[8] + this.utHyd.getJnDownStreamPittock();
this.jnOut[9] = this.jnOut[9] + this.utHyd.getJnCedarSG();
this.jnOut[10] = this.jnOut[10] + this.utHyd.getJnIngersollSG();
this.jnOut[11] = this.jnOut[11] + this.utHyd.getJnThamesfordSG();
this.jnOut[12] = this.jnOut[12] + this.utHyd.getJnReynoldsSG();
this.jnOut[13] = this.jnOut[13] + this.utHyd.getJnWaubunoSG();
this.jnOut[14] = this.jnOut[14] + this.utHyd.getJnEalingSG();
this.jnOut[15] = this.jnOut[15] + this.utHyd.getJnByronSG();
this.jnOut[16] = this.jnOut[16] + this.utHyd.getJnOxbowSG();
this.jnOut[17] = this.jnOut[17] + this.utHyd.getJnDingmanSG();

if ((this.currentDate.getHour() >= 19) &&
(this.currentDate.getDayOfWeek() == 7)) {

for (int i = 0; i < 18; i++) {
// this averages 28 6hr intervals into one
// 7 day value
this.jnOut[i] = this.jnOut[i] / 28.0;

// finds the minimum
if (this.jnOut[i] < this.jnOutMin[i]) {
this.jnOutMin[i] = this.jnOut[i];

// compute the theta
this.theta[i] = this.currentDate.getDayOfYear() *
2.0 * Math.PI / this.currentDate.getDaysInYear();
}
}
```

```
// for the last week in a year
if (this.currentDate.getWeekOfYear() == 52) {

computeTimingAndRegularity();

// outputs the 7 day minimum flow, the timing and
// regularity
//{{{
this.outFile.writeData(currentDate.getDate() + ", " +
this.jnOutMin[0] + ", " +
this.jnOutMin[1] + ", " +
this.jnOutMin[2] + ", " +
this.jnOutMin[3] + ", " +
this.jnOutMin[4] + ", " +
this.jnOutMin[5] + ", " +
this.jnOutMin[6] + ", " +
this.jnOutMin[7] + ", " +
this.jnOutMin[8] + ", " +
this.jnOutMin[9] + ", " +
this.jnOutMin[10] + ", " +
this.jnOutMin[11] + ", " +
this.jnOutMin[12] + ", " +
this.jnOutMin[13] + ", " +
this.jnOutMin[14] + ", " +
this.jnOutMin[15] + ", " +
this.jnOutMin[16] + ", " +
this.jnOutMin[17] + ", " +
this.MDF[0] + ", " +
this.MDF[1] + ", " +
this.MDF[2] + ", " +
this.MDF[3] + ", " +
this.MDF[4] + ", " +
this.MDF[5] + ", " +
this.MDF[6] + ", " +
this.MDF[7] + ", " +
this.MDF[8] + ", " +
this.MDF[9] + ", " +
this.MDF[10] + ", " +
this.MDF[11] + ", " +
this.MDF[12] + ", " +
this.MDF[13] + ", " +
this.MDF[14] + ", " +
this.MDF[15] + ", " +
this.MDF[16] + ", " +
this.MDF[17] + ", " +
this.R[0] + ", " +
this.R[1] + ", " +
this.R[2] + ", " +
this.R[3] + ", " +
this.R[4] + ", " +
this.R[5] + ", " +
this.R[6] + ", " +
this.R[7] + ", " +
this.R[8] + ", " +
this.R[9] + ", " +
this.R[10] + ", " +
this.R[11] + ", " +
this.R[12] + ", " +
this.R[13] + ", " +
this.R[14] + ", " +
this.R[15] + ", " +
this.R[16] + ", " +
this.R[17]);
//}}}
```

```
// reset the minimums
for (int i = 0; i < 18; i++) {
this.jnOutMin[i] = 9999.9;
}


}

// resets the flow variable
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
}
}
//}}}
}


/**
 *  This method computes the annual minimum 7 day flow at all of the
 * stream gauges in the basin.  The outputs of this method are used in
 * the low flow frequency analysis.
 *
 * @param  utHyd          The Hydrologic model
 * @param  currentDate     The current date
 * @exception  IOException  Input Output Exception
 */
public void updateMonthly(HydModel utHyd, ModelDate currentDate)
 throws IOException {
//{{{
this.utHyd = utHyd;
this.currentDate = currentDate;

// accumulates the 6hr hydrological output 7 day averages
this.jnOut[0] = this.jnOut[0] + this.utHyd.getJnMitchellSG();
this.jnOut[1] = this.jnOut[1] + this.utHyd.getJnAvonSG();
this.jnOut[2] = this.jnOut[2] + this.utHyd.getJnDownStreamWildwood();
this.jnOut[3] = this.jnOut[3] + this.utHyd.getJnStMarysSG();
this.jnOut[4] = this.jnOut[4] + this.utHyd.getJnPloverMillsSG();
this.jnOut[5] = this.jnOut[5] + this.utHyd.getJnDownStreamFanshawe();
this.jnOut[6] = this.jnOut[6] + this.utHyd.getJnMedwaySG();
this.jnOut[7] = this.jnOut[7] + this.utHyd.getJnUpStreamPittock();
this.jnOut[8] = this.jnOut[8] + this.utHyd.getJnDownStreamPittock();
this.jnOut[9] = this.jnOut[9] + this.utHyd.getJnCedarSG();
this.jnOut[10] = this.jnOut[10] + this.utHyd.getJnIngersollSG();
this.jnOut[11] = this.jnOut[11] + this.utHyd.getJnThamesfordSG();
this.jnOut[12] = this.jnOut[12] + this.utHyd.getJnReynoldsSG();
this.jnOut[13] = this.jnOut[13] + this.utHyd.getJnWaubunoSG();
this.jnOut[14] = this.jnOut[14] + this.utHyd.getJnEalingSG();
this.jnOut[15] = this.jnOut[15] + this.utHyd.getJnByronSG();
this.jnOut[16] = this.jnOut[16] + this.utHyd.getJnOxbowSG();
this.jnOut[17] = this.jnOut[17] + this.utHyd.getJnDingmanSG();

if ((currentDate.getDay() == currentDate.getDaysInMonth()) &&
(currentDate.getHour() >= 19)) {

// computes the number of user times steps this month
this.userTimeStepsInMonth =
this.currentDate.getDaysInMonth() * 4;

for (int i = 0; i < 18; i++) {
// this averages userTimeStepsInMonth 6hr
//intervals into one monthly value
this.jnOut[i] = this.jnOut[i] /
this.userTimeStepsInMonth;
```

```
// finds the minimum
if (this.jnOut[i] < this.jnOutMin[i]) {
this.jnOutMin[i] = this.jnOut[i];

// compute the theta
this.theta[i] = this.currentDate.getDayOfYear() *
2.0 * Math.PI / this.currentDate.getDaysInYear();
}
}

if (this.currentDate.getDayOfYear() ==
this.currentDate.getDaysInYear()) {

computeTimingAndRegularity();

// outputs the monthly minimum flow, the timing and
// regularity
//{{{
this.outFile.writeData(currentDate.getDate() + ", " +
this.jnOutMin[0] + ", " +
this.jnOutMin[1] + ", " +
this.jnOutMin[2] + ", " +
this.jnOutMin[3] + ", " +
this.jnOutMin[4] + ", " +
this.jnOutMin[5] + ", " +
this.jnOutMin[6] + ", " +
this.jnOutMin[7] + ", " +
this.jnOutMin[8] + ", " +
this.jnOutMin[9] + ", " +
this.jnOutMin[10] + ", " +
this.jnOutMin[11] + ", " +
this.jnOutMin[12] + ", " +
this.jnOutMin[13] + ", " +
this.jnOutMin[14] + ", " +
this.jnOutMin[15] + ", " +
this.jnOutMin[16] + ", " +
this.jnOutMin[17] + ", " +
this.MDF[0] + ", " +
this.MDF[1] + ", " +
this.MDF[2] + ", " +
this.MDF[3] + ", " +
this.MDF[4] + ", " +
this.MDF[5] + ", " +
this.MDF[6] + ", " +
this.MDF[7] + ", " +
this.MDF[8] + ", " +
this.MDF[9] + ", " +
this.MDF[10] + ", " +
this.MDF[11] + ", " +
this.MDF[12] + ", " +
this.MDF[13] + ", " +
this.MDF[14] + ", " +
this.MDF[15] + ", " +
this.MDF[16] + ", " +
this.MDF[17] + ", " +
this.R[0] + ", " +
this.R[1] + ", " +
this.R[2] + ", " +
this.R[3] + ", " +
this.R[4] + ", " +
this.R[5] + ", " +
this.R[6] + ", " +
this.R[7] + ", " +
```

```
this.R[8] + ", " +
this.R[9] + ", " +
this.R[10] + ", " +
this.R[11] + ", " +
this.R[12] + ", " +
this.R[13] + ", " +
this.R[14] + ", " +
this.R[15] + ", " +
this.R[16] + ", " +
this.R[17]);
//}}}

// reset the minimums
for (int i = 0; i < 18; i++) {
this.jnOutMin[i] = 9999.9;
}

}

// resets the flow variable
for (int i = 0; i < 18; i++) {
this.jnOut[i] = 0.0;
}
}
//}}}
}


/**
 *  This method updates the timing and regularity of the object
 */
private void computeTimingAndRegularity() {
//{{{

// increments the year counted
this.yearCount++;

// updates the MDF and R values (i.e., timing and regularity of
// events)
for (int i = 0; i < 18; i++) {
this.sumX[i] = this.sumX[i] + Math.cos(this.theta[i]);
this.sumY[i] = this.sumY[i] + Math.sin(this.theta[i]);

this.xBar[i] = this.sumX[i] / yearCount;
this.yBar[i] = this.sumY[i] / yearCount;

// note that a century has 36524 days, which works out
// as 365.24 days per year
if ((xBar[i] >= 0) && (yBar[i] >= 0)) {
// quadrant I
this.MDF[i] = Math.atan(this.yBar[i] /
this.xBar[i]) * 365.24 / (2.0 * Math.PI);
} else if ((xBar[i] <= 0) && (yBar[i] >= 0)) {
// quadrant II
this.MDF[i] = (Math.PI - Math.atan(Math.abs(this.yBar[i]) /
Math.abs(this.xBar[i]))) * 365.24 / (2.0 * Math.PI);
} else if ((xBar[i] <= 0) && (yBar[i] <= 0)) {
// quadrant III
this.MDF[i] = (Math.PI + Math.atan(Math.abs(this.yBar[i]) /
Math.abs(this.xBar[i]))) * 365.24 / (2.0 * Math.PI);
} else if ((xBar[i] >= 0) && (yBar[i] <= 0)) {
// quadrant IV
this.MDF[i] = (2.0 * Math.PI - Math.atan(Math.abs(this.yBar[i]) /
Math.abs(this.xBar[i]))) * 365.24 / (2.0 * Math.PI);
```

```
}

this.R[i] = Math.sqrt(Math.pow(this.xBar[i], 2.0) +
Math.pow(this.yBar[i], 2.0));
}
//}}}
}




/**
 *  This method closes the file
 */
public void closeFile() {
outFile.closeFile();
}

}
```

D.7   contUtils.java

```
// this is needed for reading/writing files
import java.io.*;

/**
 *  This class is a list of static utilities needed by the project.
 *
 * @author     Predrag Prodanovic
 * @created    February 12, 2006
 */
public class contUtils {

/**
 *  A method that finds a minimum in an array
 *
 * @param  a  An array of numbers
 * @return    The minimum value of array a
 */
public static int arrayMin(int[] a) {
//{{{
int minimum = a[0];

for (int i = 1; i < a.length; i++) {
if (a[i] < minimum) {
minimum = a[i];
}
}
return minimum;
//}}}
}


/**
 *  A method that finds a minimum in an array
 *
 * @param  a  An array of numbers
 * @return    The minimum value of array a
 */
public static double arrayMin(double[] a) {
//{{{
double minimum = a[0];

for (int i = 1; i < a.length; i++) {
if (a[i] < minimum) {
minimum = a[i];
}
}
return minimum;
//}}}
}


/**
 *  This method takes an input from a file that was generated by the WG
 * and it formats it according to what is needed by SoilMoistureAccounting
 * algorithm.  This corresponds to the data generated for 300 years by
 * P. Prodanovic with Sharif's WG at FIDS.  Code written on 24 Apr 2006.
 *
 * @param  dir            Directory of the file to be read
 * @param  file           File name
 * @exception  IOException  Input Output Exception
 */
```

```
public static void formatWGInput(String dir, String file)
 throws IOException {
//{{{

// to declare input and output DataReader objects
// object in is the original file provided by the WG, and
// out is the object that has some information stripped, as it
// is not needed
DataReader in = new DataReader(dir + file + ".out");
DataWriter out = new DataWriter(dir + file + ".out2");

int n = in.countDataPoints();

String line = new String();

// only writes the new file with the information that is needed
for (int i = 0; i < 119287; i++) {
line = in.readRecord();
if (i > 9786) {
out.writeData(line);
}
}

out.closeFile();

// now to split the above file into three
DataWriter outMax = new DataWriter(dir + file + "TempMax" + ".csv");
DataWriter outMin = new DataWriter(dir + file + "TempMin" + ".csv");
DataWriter outPPT = new DataWriter(dir + file + "PPT" + ".csv");

// DataReader to read the file just written; this is what reads
// the variable line
DataReader temp = new DataReader(dir + file + ".out2");

// reset line to blank
line = "";

// to extract from a 4 column file, columsn 2, 3 and 4, as this
// is the data that is needed

// the index number of first separation
int indexNum1;

// create an array of characters
char[] charArray1 = new char[50];
String line1;
char[] charArray2 = new char[50];
String line2;
char[] charArray3 = new char[50];
String line3;
char[] charArray4 = new char[50];
String line4;

for (int i = 0; i < 109500; i++) {
line = temp.readRecord();

line1 = line;
indexNum1 = line1.indexOf("\t");
line1.getChars(0, line1.length(), charArray1, 0);
line1 = new String(charArray1, 0, indexNum1);

line = new String(charArray1, indexNum1 + 1,
charArray1.length - (indexNum1 + 1));
```

```
line2 = line;
indexNum1 = line2.indexOf("\t");
line2.getChars(0, line2.length(), charArray2, 0);
line2 = new String(charArray2, 0, indexNum1);

line = new String(charArray2, indexNum1 + 1,
charArray2.length - (indexNum1 + 1));

line3 = line;
indexNum1 = line3.indexOf("\t");
line3.getChars(0, line3.length(), charArray3, 0);
line3 = new String(charArray3, 0, indexNum1);

line = new String(charArray3, indexNum1 + 1,
charArray3.length - (indexNum1 + 1));

line4 = line;
indexNum1 = line4.indexOf("\t");
line4.getChars(0, line4.length(), charArray4, 0);
line4 = new String(charArray4, 0, indexNum1);

line = new String(charArray4, indexNum1 + 1,
charArray4.length - (indexNum1 + 1));

outMax.writeData(line2);
outMin.writeData(line3);
outPPT.writeData(line4);

line = " ";
}

// to close the files
outMax.closeFile();
outMin.closeFile();
outPPT.closeFile();

//}}}
}


/**
 *  This method takes an input from a file that was generated by the WG
 * and it formats it according to what is needed by SoilMoistureAccounting
 * algorithm.  This corresponds to the WG output data generated for 100
 * and 300 years by Sharif and given to P. Prodanovic.  Code written on
 * 14-20 Jun 2006.
 *
 * @param   dir             Directory
 * @param   file            File name
 * @exception  IOException  Input Output Exception
 */
public static void formatWGInputNew(String dir, String file)
 throws IOException {
//{{{

// to declare input and output DataReader objects
// object in is the original file provided by the WG, and
// out is the object that has some information stripped, as it
// is not needed
DataReader in = new DataReader(dir + file + ".out");
DataWriter out = new DataWriter(dir + file + ".out2");

// don't count the line numbers; only interested in first 100 yrs
//int n = in.countDataPoints();
```

```
int n = 36500;

String line = new String();

// only writes the new file with the information that is needed
for (int i = 0; i < n; i++) {
line = in.readRecord();
out.writeData(line);
}

out.closeFile();

// now to split the above file into three
DataWriter outMax = new DataWriter(dir + file + "TempMax" + ".csv");
DataWriter outMin = new DataWriter(dir + file + "TempMin" + ".csv");
DataWriter outPPT = new DataWriter(dir + file + "PPT" + ".csv");

// DataReader to read the file just written; this is what reads
// the variable line
DataReader temp = new DataReader(dir + file + ".out2");

// reset line to blank
line = "";

// to extract from a 4 column file, columsn 2, 3 and 4, as this
// is the data that is needed

// the index number of first separation
int indexNum1;

// create an array of characters
char[] charArray1 = new char[50];
String line1;
char[] charArray2 = new char[50];
String line2;
char[] charArray3 = new char[50];
String line3;
char[] charArray4 = new char[50];
String line4;

for (int i = 0; i < 36500; i++) {
line = temp.readRecord();

line1 = line;
indexNum1 = line1.indexOf("\t");
line1.getChars(0, line1.length(), charArray1, 0);
line1 = new String(charArray1, 0, indexNum1);

line = new String(charArray1, indexNum1 + 1,
charArray1.length - (indexNum1 + 1));

line2 = line;
indexNum1 = line2.indexOf("\t");
line2.getChars(0, line2.length(), charArray2, 0);
line2 = new String(charArray2, 0, indexNum1);

line = new String(charArray2, indexNum1 + 1,
charArray2.length - (indexNum1 + 1));

line3 = line;
indexNum1 = line3.indexOf("\t");
line3.getChars(0, line3.length(), charArray3, 0);
line3 = new String(charArray3, 0, indexNum1);
```

```
line = new String(charArray3, indexNum1 + 1,
charArray3.length - (indexNum1 + 1));

line4 = line;
indexNum1 = line4.indexOf("\t");
line4.getChars(0, line4.length(), charArray4, 0);
line4 = new String(charArray4, 0, indexNum1);

line = new String(charArray4, indexNum1 + 1,
charArray4.length - (indexNum1 + 1));

outMax.writeData(line2);
outMin.writeData(line3);
outPPT.writeData(line4);

line = " ";
}

// to close the files
outMax.closeFile();
outMin.closeFile();
outPPT.closeFile();

//}}}
}


/**
 *  This method simply takes a file as input, and adds line numbers
 * at the start of each line.
 *
 * @param   inputFile       Name of the input file
 * @param   outputFile      Name of the output file
 * @param   inDir           Input Directory
 * @param   outDir          Output Directory
 * @exception  IOException   Input Output Exception
 */
public static void addLineNumbers(String inDir, String outDir,
String inputFile, String outputFile) throws IOException {
//{{{
DataReader in = new DataReader(inDir + inputFile);
DataWriter out = new DataWriter(outDir + outputFile);

// number of lines in a file
int n = in.countDataPoints();

// the line that is read from the file
String line = "";

// the actual line number
int lineNo = 0;

for (int i = 0; i < n; i++) {
lineNo = i + 1;
if (i < 9) {
line = in.readRecord();
out.writeData("    " + lineNo + ": " + line);
}
if ((i >= 9) && (i < 99)) {
line = in.readRecord();
out.writeData("   " + lineNo + ": " + line);
}
if ((i >= 99) && (i < 999)) {
line = in.readRecord();
```

```
out.writeData(" " + lineNo + ": " + line);
}
if ((i >= 999) && (i < 9999)) {
line = in.readRecord();
out.writeData("" + lineNo + ": " + line);
}
}


// to close the file
out.closeFile();
//}}}
}



/**
 *  This method formats an entire scenario at once
 *
 * @param  dir               Description of the Parameter
 * @exception  IOException   Input Output Exception
 */
public static void formatAllWGInput(String dir) throws IOException {

// there are 15 input files per scenario that need to be formatted
String[] file = {"Blythe", "Dorchester", "Embro", "Exeter",
"Folden", "Fullarton", "GlenAllan", "Illderton",
"LondonA", "StThomas", "Stratford", "Tavistock",
"WaterlooA", "Woodstock", "Wroxeter"};

for (int i = 0; i < file.length; i++) {
formatWGInputNew(dir, file[i]);
}

}



/**
 *  In order to calculate the adjusted ppt, and save it to a file
 * this method takes in three file names (max and min temp, together
 * with precipitation, and it outputs the adjusted ppt according to the
 * Juro's snow melt algorithm.  The parameters of the algorithm are
 * embedded into this code, as they don't change from gauge to gauge.
 *
 * @param  pptFile          Name of the PPT file
 * @param  maxTempFile      Name of the maxTemp file
 * @param  minTempFile      Name of the minTemp file
 * @param  adjPPTFile       Name of the adjPPT file (i.e., the output)
 * @exception  IOException  Input Output Exception
 */
private static void Snow(String pptFile, String maxTempFile,
String minTempFile, String adjPPTFile) throws IOException {
//{{{

// these are the input files
DataReader oMaxTemp = new DataReader(maxTempFile);
DataReader oMinTemp = new DataReader(minTempFile);
DataReader oPPT = new DataReader(pptFile);

// this is the output file
DataWriter oAdjPPT = new DataWriter(adjPPTFile);

// count the number of points in the data file
int n = oMaxTemp.countDataPoints();
int m = oMinTemp.countDataPoints();
int o = oPPT.countDataPoints();
```

```
// to check that all data files are of the same size
if ((n != m) || (n != o) || (m != o)) {
System.out.print("Number of data points in method ");
System.out.println("adjustForSnow do not match");
System.out.println("Simulation Terminated!!!");
System.exit(0);
}

// input data
// daily precipitation [mm]
double P;
// average daily temperature [deg C]
double T;

// output data
// separated rainfall [mm]
double R = -999.0;
// separated snowfall [mm]
double[] S = new double[2];
// adjusted precipitation [mm]
double NP;
// daily melt amount [mm]
double M;

// model parameters
// melt rate [mm/degree/day]
final double MR = 4.0;
// critital temp for melt [deg C]
final double Tcrit = 0.0;
// lower temp bound [deg C]
final double Tmin = -4.0;
// upper temp bound [deg C]
final double Tmax = -2.0;

M = 0.0;

for (int i = 0; i < n; i++) {

// reads the initial data from the files
P = oPPT.readCurrentData();
// maximumum and minimum temperature is averaged
T = (oMinTemp.readCurrentData() +
oMaxTemp.readCurrentData()) / 2.0;

// separates ppt into rainfall and snowfall
if (T <= Tmin) {
S[1] = P;
R = 0.0;
} else if ((Tmin < T) & (T < Tmax)) {
S[1] = P * ((Tmax - T) / (Tmax - Tmin));
R = P - S[1];

} else if (T >= Tmax) {
S[1] = 0.0;
R = P;
}

// accumulates snowfall
if (i > 0) {
S[1] = S[1] + S[0];
}

// accumulating snowmelt
```

```
M = MR * (T - Tcrit);

// calculate adjusted ppt
if (M > 0.0) {
if (S[1] > 0.0) {
if (S[1] > M) {
S[1] = S[1] - M;
NP = R + M;
} else {
NP = R + S[1];
S[1] = 0.0;
}
} else {
NP = R;
}
} else {
NP = R;
}

// reset S
if (i > 0) {
S[0] = S[1];
}

// writes the data to a file
oAdjPPT.writeData(NP);

}

oAdjPPT.closeFile();
//}}}
}


/**
 *  This method simply calls the Snow method above with all data files.
 *
 * @param   inputDir        Directory of input files
 * @param   outputDir       Directory of output files
 * @exception  IOException  Input Output Exception
 */
public static void adjustForSnow(String inputDir, String outputDir)
 throws IOException {
//{{{

Snow(inputDir + "BlythePPT.csv",
inputDir + "BlytheTempMax.csv",
inputDir + "BlytheTempMin.csv",
outputDir + "adjBlythePPT.csv");

Snow(inputDir + "DorchesterPPT.csv",
inputDir + "DorchesterTempMax.csv",
inputDir + "DorchesterTempMin.csv",
outputDir + "adjDorchesterPPT.csv");

Snow(inputDir + "EmbroPPT.csv",
inputDir + "EmbroTempMax.csv",
inputDir + "EmbroTempMin.csv",
outputDir + "adjEmbroPPT.csv");

Snow(inputDir + "ExeterPPT.csv",
inputDir + "ExeterTempMax.csv",
inputDir + "ExeterTempMin.csv",
outputDir + "adjExeterPPT.csv");
```

```
        Snow(inputDir + "FoldenPPT.csv",
        inputDir + "FoldenTempMax.csv",
        inputDir + "FoldenTempMin.csv",
        outputDir + "adjFoldenPPT.csv");

        Snow(inputDir + "FullartonPPT.csv",
        inputDir + "FullartonTempMax.csv",
        inputDir + "FullartonTempMin.csv",
        outputDir + "adjFullartonPPT.csv");

        Snow(inputDir + "GlenAllanPPT.csv",
        inputDir + "GlenAllanTempMax.csv",
        inputDir + "GlenAllanTempMin.csv",
        outputDir + "adjGlenAllanPPT.csv");

        Snow(inputDir + "IlldertonPPT.csv",
        inputDir + "IlldertonTempMax.csv",
        inputDir + "IlldertonTempMin.csv",
        outputDir + "adjIlldertonPPT.csv");

        Snow(inputDir + "LondonAPPT.csv",
        inputDir + "LondonATempMax.csv",
        inputDir + "LondonATempMin.csv",
        outputDir + "adjLondonAPPT.csv");

        Snow(inputDir + "StratfordPPT.csv",
        inputDir + "StratfordTempMax.csv",
        inputDir + "StratfordTempMin.csv",
        outputDir + "adjStratfordPPT.csv");

        Snow(inputDir + "StThomasPPT.csv",
        inputDir + "StThomasTempMax.csv",
        inputDir + "StThomasTempMin.csv",
        outputDir + "adjStThomasPPT.csv");

        Snow(inputDir + "TavistockPPT.csv",
        inputDir + "TavistockTempMax.csv",
        inputDir + "TavistockTempMin.csv",
        outputDir + "adjTavistockPPT.csv");

        Snow(inputDir + "WaterlooAPPT.csv",
        inputDir + "WaterlooATempMax.csv",
        inputDir + "WaterlooATempMin.csv",
        outputDir + "adjWaterlooAPPT.csv");

        Snow(inputDir + "WoodstockPPT.csv",
        inputDir + "WoodstockTempMax.csv",
        inputDir + "WoodstockTempMin.csv",
        outputDir + "adjWoodstockPPT.csv");

        Snow(inputDir + "WroxeterPPT.csv",
        inputDir + "WroxeterTempMax.csv",
        inputDir + "WroxeterTempMin.csv",
        outputDir + "adjWroxeterPPT.csv");
        //}}}
        }


        /**
         * This method interpolates the 15 ppt gauges in the Upper Thames Basin
         * and generates ppt files for each of the 32 SubBasins in the HydModel
         * based upon the Inverse Distance method of HEC-HMS.
         * The method could be written to be more general, but rather than passing
```

```
 * parameters of station names and their lat/lon, together with SubBasin
 * names with their lat/lon this is done by vectors at the start of the
 * method.  The method can only be run after method adjustForSnow
 * generated the 15 adjusted ppt files.
 *
 * @param   inputDir        Directory of input files
 * @param   outputDir       Directory of output files
 * @exception  IOException  Input Output Exception
 */
public static void interpolateSpacially(String inputDir, String outputDir)
 throws IOException {
//{{{

// ppt gauges (input)
final String[] inputFiles = {"adjBlythePPT.csv",
"adjDorchesterPPT.csv", "adjEmbroPPT.csv", "adjExeterPPT.csv",
"adjFoldenPPT.csv", "adjFullartonPPT.csv", "adjGlenAllanPPT.csv",
"adjIlldertonPPT.csv", "adjLondonAPPT.csv", "adjStThomasPPT.csv",
"adjStratfordPPT.csv", "adjTavistockPPT.csv", "adjWaterlooAPPT.csv",
"adjWoodstockPPT.csv", "adjWroxeterPPT.csv"};

// longitudes of the ppt gauges
final double[] lonGauges = {-81.3666666666667, -81.0166666666667,
-80.9166666666667, -81.5, -80.7666666666667,
-81.2, -80.7166666666667, -81.4166666666667, -81.15,
-81.2, -81.0, -80.8166666666667, -80.5166666666667,
-80.7666666666667, -81.15};

// lattitudes of the ppt gauges
final double[] latGauges = {43.7166666666667, 43.0, 43.25,
43.35, 43.0166666666667, 43.3833333333333, 43.6666666666667,
43.05, 43.0166666666667, 43.7666666666667, 43.3666666666667,
43.3166666666667, 43.4666666666667, 43.1333333333333,
43.8666666666667};

// SubBasin ppt (output)
final String[] outputFiles = {"sb1PPT.csv", "sb3PPT.csv",
"sb4PPT.csv", "sb5PPT.csv", "sb7PPT.csv", "sb8PPT.csv",
"sb9PPT.csv", "sb10PPT.csv", "sb11PPT.csv", "sb12PPT.csv",
"sb13PPT.csv", "sb14PPT.csv", "sb15PPT.csv", "sb16PPT.csv",
"sb17PPT.csv", "sb18PPT.csv", "sb19PPT.csv", "sb20PPT.csv",
"sb21PPT.csv", "sb22PPT.csv", "sb23PPT.csv", "sb24PPT.csv",
"sb25PPT.csv", "sb26PPT.csv", "sb27PPT.csv", "sb28PPT.csv",
"sb29PPT.csv", "sb30PPT.csv", "sb31PPT.csv", "sb32PPT.csv",
"sb33PPT.csv", "sb34PPT.csv"};

// longitudes of the SubBasins
final double[] lonSubBasins = {-81.110, -81.202, -81.019, -81.154,
-81.000, -81.255, -81.065, -80.960, -81.102, -81.175, -81.290,
-81.153, -81.150, -81.226, -81.283, -80.817, -80.711, -80.734,
-80.825, -80.906, -80.919, -80.981, -80.879, -81.043, -81.059,
-81.152, -81.217, -81.280, -81.341, -81.363, -81.450, -81.233};

// lattitudes of the SubBasins
final double[] latSubBasins = {43.517, 43.419, 43.428, 43.358,
43.367, 43.380, 43.310, 43.271, 43.253, 43.246, 43.314, 43.204,
43.119, 43.039, 43.143, 43.292, 43.189, 43.068, 43.101, 43.035,
43.177, 43.042, 42.955, 43.002, 43.111, 43.032, 42.974, 42.977,
42.982, 43.039, 42.937, 42.922};

// to make sure the input and output data is entered correctly
if ((inputFiles.length != lonGauges.length) ||
(inputFiles.length != latGauges.length) ||
(lonGauges.length != latGauges.length)) {
```

```
System.out.print("Number of files in input data in ");
System.out.println("method interpolateSpacially() doesn't match");
System.out.println("Simulation Terminated!");
System.exit(0);
}

if ((outputFiles.length != lonSubBasins.length) ||
(outputFiles.length != latSubBasins.length) ||
(lonSubBasins.length != latSubBasins.length)) {
System.out.print("Number of files in output data in");
System.out.println("method interpolate doesn't match");
System.out.println("Simulation Terminated!");
System.exit(0);
}

// number of gauges and subBasins
int numGauges = inputFiles.length;
int numSubBasins = outputFiles.length;

// distance between a subBasin and a Gauge
double dist = -9999.99;

double xDist = -9999.99;
double yDist = -9999.99;

// arrays that represents a minimum distance for each subBasin
// for each of the four quadrants
double[] min1 = new double[numSubBasins];
double[] min2 = new double[numSubBasins];
double[] min3 = new double[numSubBasins];
double[] min4 = new double[numSubBasins];

// weights vector array that stores weights of each subBasin
double[] w1 = new double[numSubBasins];
double[] w2 = new double[numSubBasins];
double[] w3 = new double[numSubBasins];
double[] w4 = new double[numSubBasins];

// gauges vector that tells which gauge is the minimum for
// each subBasin (for each quadrant)
int[] g1 = new int[numSubBasins];
int[] g2 = new int[numSubBasins];
int[] g3 = new int[numSubBasins];
int[] g4 = new int[numSubBasins];

// initialize the min and weights arrays to a large number
for (int i = 0; i < numSubBasins; i++) {
min1[i] = 9999.99;
min2[i] = 9999.99;
min3[i] = 9999.99;
min4[i] = 9999.99;
w1[i] = 9999.99;
w2[i] = 9999.99;
w3[i] = 9999.99;
w4[i] = 9999.99;

// to initialize the gauge minimum vector
g1[i] = 0;
g2[i] = 0;
g3[i] = 0;
g4[i] = 0;
}

// computes the distance between each subBasin and each gauge
```

```
for (int i = 0; i < numSubBasins; i++) {
for (int j = 0; j < numGauges; j++) {
xDist = lonSubBasins[i] - lonGauges[j];
yDist = latSubBasins[i] - latGauges[j];

dist = Math.sqrt(Math.pow(xDist, 2.0) +
Math.pow(yDist, 2.0));

// for each subBasin, place a gauge in a quadrant
// quadrants are numbered as follows
// 2 | 1
// 3 | 4

// quadrant 1
if ((lonGauges[j] >= lonSubBasins[i]) &&
(latGauges[j] >= latSubBasins[i])) {

if (dist < min1[i]) {
min1[i] = dist;
g1[i] = j;
}
}

// quadrant 2
if ((lonGauges[j] <= lonSubBasins[i]) &&
(latGauges[j] >= latSubBasins[i])) {

if (dist < min2[i]) {
min2[i] = dist;
g2[i] = j;
}
}
// quadrant 3
if ((lonGauges[j] <= lonSubBasins[i]) &&
(latGauges[j] <= latSubBasins[i])) {

if (dist < min3[i]) {
min3[i] = dist;
g3[i] = j;
}
}
// quadrant 4
if ((lonGauges[j] >= lonSubBasins[i]) &&
(latGauges[j] <= latSubBasins[i])) {

if (dist < min4[i]) {
min4[i] = dist;
g4[i] = j;
}
}
}
}
/*
 *  // THIS IS NOT NEEDED BECAUSE MIN1-4 REMAINS HIGH (I.E., 9999.99)
 *  // AND WHEN THIS GETS DIVIDED BY UNITY, AND SQUARED, IT
 *  // PRACTICALLY BECOMES ZERO IN THE WEIGHTS CALCULATIONS
 *  // SO NO NEED TO CONSIDER THIS AT ALL
 *  // in case when a subBasin just doesn't have a gauge in one of
 *  // its quadrants, that quadrant just gets ignored and its weight
 *  // is assigned to zero
 *  for (int i = 0; i < numSubBasins; i++){
 *  if (min1[i] == 9999.99){
 *  min1[i] = 0.0;
 *  // assigns weight to be zero
```

```
 *  w1[i] = 0.0;
 *  // it needs to have a gauge, so just give it the
 *  // first one in the list.  Note that this gauge
 *  // just gets multiplied by zero, so it never
 *  // actually gets used; it just needs to have
 *  // something there so that syntax works ok
 *  g1[i] = inputFiles[0];
 *  }
 *  if (min2[i] == 9999.99){
 *  min2[i] = 0.0;
 *  w2[i] = 0.0;
 *  g2[i] = inputFiles[0];
 *  }
 *  if (min3[i] == 9999.99){
 *  min3[i] = 0.0;
 *  w3[i] = 0.0;
 *  g3[i] = inputFiles[0];
 *  }
 *  if (min4[i] == 9999.99){
 *  min4[i] = 0.0;
 *  w4[i] = 0.0;
 *  g4[i] = inputFiles[0];
 *  }
 *  }
 */
// to calculate weights vector
for (int i = 0; i < numSubBasins; i++) {
w1[i] = (1.0 / Math.pow(min1[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w2[i] = (1.0 / Math.pow(min2[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w3[i] = (1.0 / Math.pow(min3[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w4[i] = (1.0 / Math.pow(min4[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
}

// now that we have weights, we can calculate the interpolated
// precipitation

// read in the input files
// declare an array of DataReader objects and instantiate them
DataReader[] in = new DataReader[numGauges];
for (int i = 0; i < numGauges; i++) {
in[i] = new DataReader(inputDir + inputFiles[i]);

}

// for output files
// declare an array of DataWriter objects and instantiate them
DataWriter[] out = new DataWriter[numSubBasins];
for (int i = 0; i < numSubBasins; i++) {
```

```
        out[i] = new DataWriter(outputDir + outputFiles[i]);

    }

    // this assumes that all data files have the same number of data
    // points as the first file in the list
    int numDataPoints = in[0].countDataPoints();

    // values that are read from the input files
    double[] inVal = new double[numGauges];

    // interpolated ppt
    double[] interpPPT = new double[numSubBasins];

    for (int k = 0; k < numDataPoints; k++) {
    for (int j = 0; j < numGauges; j++) {
    inVal[j] = in[j].readCurrentData();
    }
    for (int i = 0; i < numSubBasins; i++) {
    interpPPT[i] = inVal[g1[i]] * w1[i] +
    inVal[g2[i]] * w2[i] +
    inVal[g3[i]] * w3[i] +
    inVal[g4[i]] * w4[i];
    out[i].writeData(interpPPT[i]);
    }
    }
    // to close the output files
    for (int i = 0; i < numSubBasins; i++) {
    out[i].closeFile();
    }
    //}}}
    }


    /**
     *  This method interpolates historical data, and fills in the missing
     * values.  It is very similar in structure to interpolateSpacially()
     * method, except that this method does only temporal interpolation.
     * Again, as with other methods in this class, this method takes no
     * parameters as input; its parameters are embedded into the code.  The
     * method currently takes the Upper Thames Basin ppt and temp data for
     * years 1964-2001, and simply fills the gaps in records with values
     * estimated by HEC's inverse square distance method.
     * This method must have different input and output directories.
     *
     * @param   inputDir       Directory of input files
     * @param   outputDir      Directory of output files
     * @exception  IOException  Input Output Exception
     */
    public static void interpolateTemporally(String inputDir,
    String outputDir) throws IOException {
    //{{{

    // the data files where ppt is stored
    final String[] inputFilesPPT = {"BlythePPT.csv",
    "DorchesterPPT.csv", "EmbroPPT.csv", "ExeterPPT.csv",
    "FoldenPPT.csv", "FullartonPPT.csv", "GlenAllanPPT.csv",
    "IlldertonPPT.csv", "LondonAPPT.csv", "StThomasPPT.csv",
    "StratfordPPT.csv", "TavistockPPT.csv", "WaterlooAPPT.csv",
    "WoodstockPPT.csv", "WroxeterPPT.csv"};

    // the data files where TempMax is stored
    final String[] inputFilesTempMax = {"BlytheTempMax.csv",
    "DorchesterTempMax.csv", "EmbroTempMax.csv", "ExeterTempMax.csv",
```

```
"FoldenTempMax.csv", "FullartonTempMax.csv", "GlenAllanTempMax.csv",
"IlldertonTempMax.csv", "LondonATempMax.csv", "StThomasTempMax.csv",
"StratfordTempMax.csv", "TavistockTempMax.csv", "WaterlooATempMax.csv",
"WoodstockTempMax.csv", "WroxeterTempMax.csv"};

// the data files where TempMin is stored
final String[] inputFilesTempMin = {"BlytheTempMin.csv",
"DorchesterTempMin.csv", "EmbroTempMin.csv", "ExeterTempMin.csv",
"FoldenTempMin.csv", "FullartonTempMin.csv", "GlenAllanTempMin.csv",
"IlldertonTempMin.csv", "LondonATempMin.csv", "StThomasTempMin.csv",
"StratfordTempMin.csv", "TavistockTempMin.csv", "WaterlooATempMin.csv",
"WoodstockTempMin.csv", "WroxeterTempMin.csv"};

// longitudes of the gauges
final double[] lonGauges = {-81.3666666666667, -81.0166666666667,
-80.9166666666667, -81.5, -80.7666666666667,
-81.2, -80.7166666666667, -81.4166666666667, -81.15,
-81.2, -81.0, -80.8166666666667, -80.5166666666667,
-80.7666666666667, -81.15};

// lattitudes of the gauges
final double[] latGauges = {43.7166666666667, 43.0, 43.25,
43.35, 43.0166666666667, 43.3833333333333, 43.6666666666667,
43.05, 43.0166666666667, 43.7666666666667, 43.3666666666667,
43.3166666666667, 43.4666666666667, 43.1333333333333,
43.8666666666667};

// the data files where interpolated (Int) ppt is stored
final String[] outputFilesPPT = inputFilesPPT;

// the data files where interpolated TempMax is stored
final String[] outputFilesTempMax = inputFilesTempMax;

// the data files where interpolated TempMin is stored
final String[] outputFilesTempMin = inputFilesTempMin;

// to make sure the input and output data is entered correctly
if ((inputFilesPPT.length != lonGauges.length) ||
(inputFilesPPT.length != latGauges.length) ||
(lonGauges.length != latGauges.length) ||
(inputFilesPPT.length != inputFilesTempMin.length) ||
(inputFilesPPT.length != inputFilesTempMax.length) ||
(inputFilesTempMin.length != inputFilesTempMax.length)) {
System.out.print("Number of files in input data in ");
System.out.println("method interpolateTemporally() doesn't match");
System.out.println("Simulation Terminated!");
System.exit(0);
}

// number of gauges
int numGauges = inputFilesPPT.length;

// distance between a Gauge i and a Gauge j
double dist = -9999.99;

// a 2-d array that stores the distances
double xDist = -9999.99;
double yDist = -9999.99;

// arrays that represents a minimum distance for each gauge
// for each of the four quadrants
double[] min1 = new double[numGauges];
double[] min2 = new double[numGauges];
double[] min3 = new double[numGauges];
```

```
                double[] min4 = new double[numGauges];

                // weights vector array that stores weights of each gauge
                double[] w1 = new double[numGauges];
                double[] w2 = new double[numGauges];
                double[] w3 = new double[numGauges];
                double[] w4 = new double[numGauges];

                // gauges vector that tells which gauge is the minimum for
                // each gauge (for each quadrant)
                int[] g1 = new int[numGauges];
                int[] g2 = new int[numGauges];
                int[] g3 = new int[numGauges];
                int[] g4 = new int[numGauges];

                // MAXIMUM TEMPERATURE INTERPOLATION STARTS HERE
                // initialize the min and weights arrays to a large number
                for (int i = 0; i < numGauges; i++) {
                min1[i] = 99999999.99;
                min2[i] = 99999999.99;
                min3[i] = 99999999.99;
                min4[i] = 99999999.99;
                w1[i] = 99999999.99;
                w2[i] = 99999999.99;
                w3[i] = 99999999.99;
                w4[i] = 99999999.99;

                // to initialize the gauge minimum vector
                g1[i] = 0;
                g2[i] = 0;
                g3[i] = 0;
                g4[i] = 0;
                }

                // values that are read from the input files
                double[] inValPPT = new double[numGauges];

                // interpolated ppt
                double[] interpPPT = new double[numGauges];

                // read in the PPT input files
                // declare an array of DataReader objects and instantiate them
                DataReader[] inPPT = new DataReader[numGauges];
                for (int i = 0; i < numGauges; i++) {
                inPPT[i] = new DataReader(inputDir + inputFilesPPT[i]);
                }

                // this assumes that all data files have the same number of data
                // points as the first file in the list
                int numDataPoints = inPPT[0].countDataPoints();

                // for PPT output files
                // declare an array of DataWriter objects and instantiate them
                DataWriter[] outPPT = new DataWriter[numGauges];
                for (int i = 0; i < numGauges; i++) {
                outPPT[i] = new DataWriter(outputDir + outputFilesPPT[i]);
                }

                for (int k = 0; k < numDataPoints; k++) {

                // this loop reads the current data value for each gauge
                // and stores it to inValPPT[x]
                for (int x = 0; x < numGauges; x++) {
                inValPPT[x] = inPPT[x].readCurrentData();
```

```
}

for (int i = 0; i < numGauges; i++) {
if (inValPPT[i] > 777) {

for (int j = 0; j < numGauges; j++) {
xDist = lonGauges[i] - lonGauges[j];
yDist = latGauges[i] - latGauges[j];

dist = Math.sqrt(Math.pow(xDist, 2.0) +
Math.pow(yDist, 2.0));

// for each subBasin, place a gauge in a quadrant
// quadrants are numbered as follows
// 2 | 1
// 3 | 4

// computes the distance between each gauge
// this is different from the interpolateSpacially() in that
// equality constraint is removed; this means that a gauge will
// not be able to say that zero (or itself) is its nearest
// neighbour

// quadrant 1
if ((lonGauges[j] > lonGauges[i]) &&
(latGauges[j] > latGauges[i])) {

if ((dist < min1[i]) && (inValPPT[j] < 777)) {
min1[i] = dist;
g1[i] = j;
}
}

// quadrant 2
if ((lonGauges[j] < lonGauges[i]) &&
(latGauges[j] > latGauges[i])) {

if (dist < min2[i] && (inValPPT[j] < 777)) {
min2[i] = dist;
g2[i] = j;
}
}
// quadrant 3
if ((lonGauges[j] < lonGauges[i]) &&
(latGauges[j] < latGauges[i])) {

if (dist < min3[i] && (inValPPT[j] < 777)) {
min3[i] = dist;
g3[i] = j;
}
}
// quadrant 4
if ((lonGauges[j] > lonGauges[i]) &&
(latGauges[j] < latGauges[i])) {

if (dist < min4[i] && (inValPPT[j] < 777)) {
min4[i] = dist;
g4[i] = j;
}
}
}

w1[i] = (1.0 / Math.pow(min1[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
```

```
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w2[i] = (1.0 / Math.pow(min2[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w3[i] = (1.0 / Math.pow(min3[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w4[i] = (1.0 / Math.pow(min4[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));

// sometimes the value(s) upon which the
// interpolated values is to be computed
// is large (because it was based on the
// missing values).  In this case, set
// the weight of that value is set to zero

// in case the interpolated value is to be
// based on closest values > 200 mm / day,
// the user should investiage this
if (inValPPT[g1[i]] > 200) {
w1[i] = 0.0;
}
if (inValPPT[g2[i]] > 200) {
w2[i] = 0.0;
}
if (inValPPT[g3[i]] > 200) {
w3[i] = 0.0;
}
if (inValPPT[g4[i]] > 200) {
w4[i] = 0.0;
}

interpPPT[i] = inValPPT[g1[i]] * w1[i] +
inValPPT[g2[i]] * w2[i] +
inValPPT[g3[i]] * w3[i] +
inValPPT[g4[i]] * w4[i];

if (interpPPT[i] > 200) {
System.out.println("Impossible Value; Exiting!");
System.out.print("Gauge :" + outputFilesPPT[i]);
System.exit(0);
}

outPPT[i].writeData(interpPPT[i]);
} else {
outPPT[i].writeData(inValPPT[i]);
}
}
}

// to close the output files
for (int i = 0; i < numGauges; i++) {
outPPT[i].closeFile();
}
```

```
// MAXIMUM TEMPERATURE INTERPOLATION STARTS HERE
// reset all of the variables used in the k loop
for (int i = 0; i < numGauges; i++) {
min1[i] = 99999999.99;
min2[i] = 99999999.99;
min3[i] = 99999999.99;
min4[i] = 99999999.99;
w1[i] = 99999999.99;
w2[i] = 99999999.99;
w3[i] = 99999999.99;
w4[i] = 99999999.99;

// to initialize the gauge minimum vector
g1[i] = 0;
g2[i] = 0;
g3[i] = 0;
g4[i] = 0;
}
xDist = 99999999.99;
yDist = 99999999.99;
dist = 99999999.99;

// values that are read from the input files
double[] inValTempMax = new double[numGauges];

// interpolated TempMax
double[] interpTempMax = new double[numGauges];

// read in the TempMax input files
// declare an array of DataReader objects and instantiate them
DataReader[] inTempMax = new DataReader[numGauges];
for (int i = 0; i < numGauges; i++) {
inTempMax[i] = new DataReader(inputDir + inputFilesTempMax[i]);
}

// declare an array of DataWriter objects and instantiate them
DataWriter[] outTempMax = new DataWriter[numGauges];
for (int i = 0; i < numGauges; i++) {
outTempMax[i] = new DataWriter(outputDir + outputFilesTempMax[i]);
}

for (int k = 0; k < numDataPoints; k++) {
// this loop reads the current data value for each gauge
for (int x = 0; x < numGauges; x++) {
inValTempMax[x] = inTempMax[x].readCurrentData();
}

for (int i = 0; i < numGauges; i++) {
if (inValTempMax[i] > 777) {

for (int j = 0; j < numGauges; j++) {
xDist = lonGauges[i] - lonGauges[j];
yDist = latGauges[i] - latGauges[j];

dist = Math.sqrt(Math.pow(xDist, 2.0) +
Math.pow(yDist, 2.0));

// for each subBasin, place a gauge in a quadrant
// quadrants are numbered as follows
// 2 | 1
// 3 | 4

// computes the distance between each gauge
// this is different from the interpolateSpacially() in that
```

```
// equality constraint is removed; this means that a gauge will
// not be able to say that zero (or itself) is its nearest
// neighbour

// quadrant 1
if ((lonGauges[j] > lonGauges[i]) &&
(latGauges[j] > latGauges[i])) {

if ((dist < min1[i]) && (inValTempMax[j] < 777)) {
min1[i] = dist;
g1[i] = j;
}
}

// quadrant 2
if ((lonGauges[j] < lonGauges[i]) &&
(latGauges[j] > latGauges[i])) {

if (dist < min2[i] && (inValTempMax[j] < 777)) {
min2[i] = dist;
g2[i] = j;
}
}
// quadrant 3
if ((lonGauges[j] < lonGauges[i]) &&
(latGauges[j] < latGauges[i])) {

if (dist < min3[i] && (inValTempMax[j] < 777)) {
min3[i] = dist;
g3[i] = j;
}
}
// quadrant 4
if ((lonGauges[j] > lonGauges[i]) &&
(latGauges[j] < latGauges[i])) {

if (dist < min4[i] && (inValTempMax[j] < 777)) {
min4[i] = dist;
g4[i] = j;
}
}
}

w1[i] = (1.0 / Math.pow(min1[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w2[i] = (1.0 / Math.pow(min2[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w3[i] = (1.0 / Math.pow(min3[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w4[i] = (1.0 / Math.pow(min4[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
```

```
// sometimes the value(s) upon which the
// interpolated values is to be computed
// is large (because it was based on the
// missing values).  In this case, set
// the weight of that value is set to zero

// in case the interpolated value is to be
// based on closest values > 100 deg C,
// the user should investiage this
if (inValTempMax[g1[i]] > 100) {
w1[i] = 0.0;
}
if (inValTempMax[g2[i]] > 100) {
w2[i] = 0.0;
}
if (inValTempMax[g3[i]] > 100) {
w3[i] = 0.0;
}
if (inValTempMax[g4[i]] > 100) {
w4[i] = 0.0;
}

interpTempMax[i] = inValTempMax[g1[i]] * w1[i] +
inValTempMax[g2[i]] * w2[i] +
inValTempMax[g3[i]] * w3[i] +
inValTempMax[g4[i]] * w4[i];

if (interpTempMax[i] > 100) {
System.out.println("Impossible Value; Exiting!");
System.out.print("Gauge :" + outputFilesTempMax[i]);
System.exit(0);
}

outTempMax[i].writeData(interpTempMax[i]);
} else {
outTempMax[i].writeData(inValTempMax[i]);
}
}

}

// to close the output files
for (int i = 0; i < numGauges; i++) {
outTempMax[i].closeFile();
}

// MINIMUM TEMPERATURE INTERPOLATION STARTS HERE
// reset all of the variables used in the k loop
for (int i = 0; i < numGauges; i++) {
min1[i] = 99999999.99;
min2[i] = 99999999.99;
min3[i] = 99999999.99;
min4[i] = 99999999.99;
w1[i] = 99999999.99;
w2[i] = 99999999.99;
w3[i] = 99999999.99;
w4[i] = 99999999.99;

// to initialize the gauge minimum vector
g1[i] = 0;
g2[i] = 0;
g3[i] = 0;
g4[i] = 0;
}
```

```
xDist = 99999999.99;
yDist = 99999999.99;
dist = 99999999.99;

// values that are read from the input files
double[] inValTempMin = new double[numGauges];

// interpolated TempMin
double[] interpTempMin = new double[numGauges];

// read in the TempMin input files
// declare an array of DataReader objects and instantiate them
DataReader[] inTempMin = new DataReader[numGauges];
for (int i = 0; i < numGauges; i++) {
inTempMin[i] = new DataReader(inputDir + inputFilesTempMin[i]);
}

// declare an array of DataWriter objects and instantiate them
DataWriter[] outTempMin = new DataWriter[numGauges];
for (int i = 0; i < numGauges; i++) {
outTempMin[i] = new DataWriter(outputDir + outputFilesTempMin[i]);
}

for (int k = 0; k < numDataPoints; k++) {
// this loop reads the current data value for each gauge
for (int x = 0; x < numGauges; x++) {
inValTempMin[x] = inTempMin[x].readCurrentData();
}

for (int i = 0; i < numGauges; i++) {
if (inValTempMin[i] > 777) {

for (int j = 0; j < numGauges; j++) {
xDist = lonGauges[i] - lonGauges[j];
yDist = latGauges[i] - latGauges[j];

dist = Math.sqrt(Math.pow(xDist, 2.0) +
Math.pow(yDist, 2.0));

// for each subBasin, place a gauge in a quadrant
// quadrants are numbered as follows
// 2 | 1
// 3 | 4

// computes the distance between each gauge
// this is different from the interpolateSpacially() in that
// equality constraint is removed; this means that a gauge will
// not be able to say that zero (or itself) is its nearest
// neighbour

// quadrant 1
if ((lonGauges[j] > lonGauges[i]) &&
(latGauges[j] > latGauges[i])) {

if ((dist < min1[i]) && (inValTempMin[j] < 777)) {
min1[i] = dist;
g1[i] = j;
}
}

// quadrant 2
if ((lonGauges[j] < lonGauges[i]) &&
(latGauges[j] > latGauges[i])) {
```

```
if (dist < min2[i] && (inValTempMin[j] < 777)) {
min2[i] = dist;
g2[i] = j;
}
}
// quadrant 3
if ((lonGauges[j] < lonGauges[i]) &&
(latGauges[j] < latGauges[i])) {

if (dist < min3[i] && (inValTempMin[j] < 777)) {
min3[i] = dist;
g3[i] = j;
}
}
// quadrant 4
if ((lonGauges[j] > lonGauges[i]) &&
(latGauges[j] < latGauges[i])) {

if (dist < min4[i] && (inValTempMin[j] < 777)) {
min4[i] = dist;
g4[i] = j;
}
}
}

w1[i] = (1.0 / Math.pow(min1[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w2[i] = (1.0 / Math.pow(min2[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w3[i] = (1.0 / Math.pow(min3[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));
w4[i] = (1.0 / Math.pow(min4[i], 2.0)) /
((1.0 / Math.pow(min1[i], 2.0)) +
(1.0 / Math.pow(min2[i], 2.0)) +
(1.0 / Math.pow(min3[i], 2.0)) +
(1.0 / Math.pow(min4[i], 2.0)));

// sometimes the value(s) upon which the
// interpolated values is to be computed
// is large (because it was based on the
// missing values).  In this case, set
// the weight of that value is set to zero

// in case the interpolated value is to be
// based on closest values > 100 deg C,
// the user should investiage this
if (inValTempMin[g1[i]] > 100) {
w1[i] = 0.0;
}
if (inValTempMin[g2[i]] > 100) {
w2[i] = 0.0;
}
if (inValTempMin[g3[i]] > 100) {
w3[i] = 0.0;
}
```

```
if (inValTempMin[g4[i]] > 100) {
w4[i] = 0.0;
}

interpTempMin[i] = inValTempMin[g1[i]] * w1[i] +
inValTempMin[g2[i]] * w2[i] +
inValTempMin[g3[i]] * w3[i] +
inValTempMin[g4[i]] * w4[i];

if (interpTempMin[i] > 100) {
System.out.println("Impossible Value; Exiting!");
System.out.print("Gauge :" + outputFilesTempMin[i]);
System.exit(0);
}

outTempMin[i].writeData(interpTempMin[i]);
} else {
outTempMin[i].writeData(inValTempMin[i]);
}
}


}

// to close the output files
for (int i = 0; i < numGauges; i++) {
outTempMin[i].closeFile();
}
//}}}
}


/**
 *  This method takes in interpolated precipitation, and computes an
 * average daily ppt for St. Marys, Ingersoll and Byron stream gauges based
 * on the weighted average method.  This is the same as the methods in
 * the HydModel.java that output the ppt.
 *
 * @param  dir              The name of the directory where the
 * interpolated ppt files are
 * @exception  IOException  Input Output Exception
 */
public static void computeWeightedDailyPPT(String dir) throws IOException {
//{{{
// the interpolated input files
String[] inputFiles = {"sb1PPT.csv", "sb3PPT.csv",
"sb4PPT.csv", "sb5PPT.csv", "sb7PPT.csv", "sb8PPT.csv",
"sb9PPT.csv", "sb10PPT.csv", "sb11PPT.csv", "sb12PPT.csv",
"sb13PPT.csv", "sb14PPT.csv", "sb15PPT.csv", "sb16PPT.csv",
"sb17PPT.csv", "sb18PPT.csv", "sb19PPT.csv", "sb20PPT.csv",
"sb21PPT.csv", "sb22PPT.csv", "sb23PPT.csv", "sb24PPT.csv",
"sb25PPT.csv", "sb26PPT.csv", "sb27PPT.csv", "sb28PPT.csv",
"sb29PPT.csv", "sb30PPT.csv", "sb31PPT.csv", "sb32PPT.csv",
"sb33PPT.csv", "sb34PPT.csv"};

// the sub basin areas, in [km^2]
double[] area = {305.505, 47.745, 151.189, 76.82, 144.0, 88.355,
78.476, 141.118, 28.942, 35.466, 153.721, 84.539,
94.198, 75.363, 202.478, 148.318, 96.84, 97.91, 170.704,
42.859, 291.08, 35.861, 165.973, 120.935, 104.945, 61.195,
22.556, 30.002, 32.409, 88.145, 50.486, 168.719};

int numSubBasins = area.length;

// DataReader objects
```

```
DataReader[] in = new DataReader[numSubBasins];

// DataWriter objects
DataWriter stMarysOut = new DataWriter(dir + "StMarysAvgDailyPPT.txt");
DataWriter ingersollOut = new DataWriter(dir + "IngersollAvgDailyPPT.txt");
DataWriter byronOut = new DataWriter(dir + "ByronAvgDailyPPT.txt");

// instantiate the DataReader objects
for (int i = 0; i < numSubBasins; i++) {
in[i] = new DataReader(dir + inputFiles[i]);
}

// the number of days in the file; assumes same for each station
int n = in[0].countDataPoints();

// calculated total basin area
double totalBasinArea = 0.0;
for (int i = 0; i < numSubBasins; i++) {
totalBasinArea = totalBasinArea + area[i];
}

// area draining to St.Marys stream gauge
// sb1, 3, 4, 5, 7, 8, 9, 10, 11
double stMarysArea = area[0] + area[1] + area[2] + area[3] +
area[4] + area[5] + area[6] + area[7] +
area[8] + area[9];

// area draining to Ingersoll stream gauge
// sb18, 19, 20, 21
double ingersollArea = area[16] + area[17] + area[18] + area[19];

// area draining to Byron stream gauge
// all subbasins except 31, 32, 33, 34
double byronArea = totalBasinArea -
(area[31] + area[30] + area[29] + area[28]);

// to create aggregated ppt for our three locations of interest
double stMarysPPT = 0.0;
double ingersollPPT = 0.0;
double byronPPT = 0.0;

// the current value of the ppt for each sub basin
double[] ppt = new double[numSubBasins];

// to aggregate the subbasin ppt
for (int i = 0; i < n; i++) {
for (int j = 0; j < numSubBasins; j++) {
// reads ppt of all subbasins
ppt[j] = in[j].readCurrentData();
}
stMarysPPT = (area[0] / stMarysArea) * ppt[0] +
(area[1] / stMarysArea) * ppt[1] +
(area[2] / stMarysArea) * ppt[2] +
(area[3] / stMarysArea) * ppt[3] +
(area[4] / stMarysArea) * ppt[4] +
(area[5] / stMarysArea) * ppt[5] +
(area[6] / stMarysArea) * ppt[6] +
(area[7] / stMarysArea) * ppt[7] +
(area[8] / stMarysArea) * ppt[8] +
(area[9] / stMarysArea) * ppt[9];
stMarysOut.writeData(stMarysPPT);

ingersollPPT = (area[16] / ingersollArea) * ppt[16] +
(area[17] / ingersollArea) * ppt[17] +
```

```
(area[18] / ingersollArea) * ppt[18] +
(area[19] / ingersollArea) * ppt[19];
ingersollOut.writeData(ingersollPPT);

byronPPT = (area[0] / byronArea) * ppt[0] +
(area[1] / byronArea) * ppt[1] +
(area[2] / byronArea) * ppt[2] +
(area[3] / byronArea) * ppt[3] +
(area[4] / byronArea) * ppt[4] +
(area[5] / byronArea) * ppt[5] +
(area[6] / byronArea) * ppt[6] +
(area[7] / byronArea) * ppt[7] +
(area[8] / byronArea) * ppt[8] +
(area[9] / byronArea) * ppt[9] +
(area[10] / byronArea) * ppt[10] +
(area[11] / byronArea) * ppt[11] +
(area[12] / byronArea) * ppt[12] +
(area[13] / byronArea) * ppt[13] +
(area[14] / byronArea) * ppt[14] +
(area[15] / byronArea) * ppt[15] +
(area[16] / byronArea) * ppt[16] +
(area[17] / byronArea) * ppt[17] +
(area[18] / byronArea) * ppt[18] +
(area[19] / byronArea) * ppt[19] +
(area[20] / byronArea) * ppt[20] +
(area[21] / byronArea) * ppt[21] +
(area[22] / byronArea) * ppt[22] +
(area[23] / byronArea) * ppt[23] +
(area[24] / byronArea) * ppt[24] +
(area[25] / byronArea) * ppt[25] +
(area[26] / byronArea) * ppt[26] +
(area[27] / byronArea) * ppt[27] +
(area[28] / byronArea) * ppt[28];
byronOut.writeData(byronPPT);
}

// to close the files
stMarysOut.closeFile();
ingersollOut.closeFile();
byronOut.closeFile();
//}}}
}


/**
 *  This method takes in a daily ppt and a daily flow file, and computes
 * historical drought information.  This information is to be used as an
 * initial condition to the System Dynamics model.
 *
 * @param  pptFile          Daily PPT file [mm], starting with 01 Jan 1964
 * @param  flowFile         Daily Flow file [cms], starting with 01 Jan 1964
 * @param  outFile          Contains historical drought characteristics
 * @exception  IOException  Input Output Exception
 */
public static void calcHistDroughtInfo(String pptFile, String flowFile,
String outFile) throws IOException {
//{{{
// the is the starting date of the historical record:
// 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 1;
int startHour = 1;
int startMinute = 0;
```

```
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

// to define DataReader and DataWriter objects
DataReader ppt = new DataReader(pptFile);
DataReader flow = new DataReader(flowFile);
DataWriter out = new DataWriter(outFile);

// variables that accumulate daily values into monthly values
double tempMonthlyPPT = 0.0;
double tempMonthlyFlow = 0.0;
/*
 *  / to make sure the input files are of the same size
 *  if (ppt.countDataPoints() != flow.countDataPoints()) {
 *  System.out.println("Input files are different sizes!");
 *  System.out.println("Simulation terminated!");
 *  System.exit(0);
 *  }
 */
// the number of days in the file
int numDays = flow.countDataPoints();

// compute the number of years; note that Java rounds down
// integer division
int numYears = numDays / 365;

// numer of months in the file
int numMonths = numYears * 12;

// arrays where monthly data is stored
double[] monthlyPPT = new double[numMonths];
double[] monthlyFlow = new double[numMonths];
double[] threeMonthPPT = new double[numMonths];
double[] eighteenMonthPPT = new double[numMonths];

// arrays where the historical 1 month averages are stored
// totals
double[] histMonthlyAvgTotalPPT = new double[12];
double[] histMonthlyAvgTotalFlow = new double[12];
// averages
double[] histMonthlyAvgPPT = new double[12];
double[] histMonthlyAvgFlow = new double[12];

// arrays where the historical 3 month averages are stored
// totals
double[] hist3MonthAvgTotalPPT = new double[12];
// averages
double[] hist3MonthAvgPPT = new double[12];

// arrays where the historical 18 month averages are stored
// totals
double[] hist18MonthAvgTotalPPT = new double[12];
// averages
double[] hist18MonthAvgPPT = new double[12];

// the lowest summer month flow is an array of size numYears,
// where each value represents the lowest monthly stream flow
// that year
double[] lowestAvgSummerMonthFlow = new double[numYears];

// a variable that counts the total number of months
int totalMonthCount = 0;

out.writeData("Date, Monthly PPT, Monthly Flow");
```

```
// the loop that computes (i.e., accumulates daily values into
// monthly averages)
for (int i = 0; i < numDays; i++) {

// this is here because the first day is already set by
// the constructor of the ModelDate object
if (i != 0) {
// increments the date by one day
currentDate.incrementDateByDays(1);
}

tempMonthlyPPT = tempMonthlyPPT + ppt.readCurrentData();
tempMonthlyFlow = tempMonthlyFlow + flow.readCurrentData();

// at the end of the month, store monthly values in
// arrays
if (currentDate.getDay() == currentDate.getDaysInMonth()) {

monthlyPPT[totalMonthCount] = tempMonthlyPPT;
monthlyFlow[totalMonthCount] = tempMonthlyFlow /
currentDate.getDaysInMonth();

out.writeData(currentDate.getDate() + ", " +
monthlyPPT[totalMonthCount] + ", " +
monthlyFlow[totalMonthCount]);
totalMonthCount++;

// to reset
tempMonthlyPPT = 0.0;
tempMonthlyFlow = 0.0;
}
}

// comptes 1 month average historical ppt and flow

// a variable that counts the months within a year
int monthCount = 0;

// accumulate the flow and ppt
for (int i = 0; i < numMonths; i++) {

histMonthlyAvgTotalPPT[monthCount] =
histMonthlyAvgTotalPPT[monthCount] +
monthlyPPT[i];

histMonthlyAvgTotalFlow[monthCount] =
histMonthlyAvgTotalFlow[monthCount] +
monthlyFlow[i];

monthCount++;
if (monthCount > 11) {
monthCount = 0;
}
}
out.writeData("");
out.writeData("Historical averages based on the historical record:");
out.writeData("oneMonthPPTAvg, oneMonthFlowAvg,  PPT Total, Flow Total");

// average the accumulated flow and ppt
for (int i = 0; i < 12; i++) {

histMonthlyAvgPPT[i] = histMonthlyAvgTotalPPT[i] /
numYears;
```

```
histMonthlyAvgFlow[i] = histMonthlyAvgTotalFlow[i] /
numYears;

out.writeData(histMonthlyAvgPPT[i] + ", " +
histMonthlyAvgFlow[i] + ", " +
histMonthlyAvgTotalPPT[i] + ", " +
histMonthlyAvgTotalFlow[i]);
}

// to compute 3 month average historical ppt
monthCount = 2;
for (int i = 2; i < numMonths; i++) {
for (int j = 0; j < 3; j++) {
threeMonthPPT[i] = threeMonthPPT[i] +
monthlyPPT[i - j];

}

threeMonthPPT[i] = threeMonthPPT[i] / 3.0;
}

// accumulate the 3 month average historical ppt
for (int i = 0; i < numMonths; i++) {

hist3MonthAvgTotalPPT[monthCount] =
hist3MonthAvgTotalPPT[monthCount] +
threeMonthPPT[i];
monthCount++;
if (monthCount > 11) {
monthCount = 0;
}
}
out.writeData("");
out.writeData("threeMonthPPTAvg, PPT Total");
// average the accumulated 3 month ppt for the 38 years of record
for (int i = 0; i < 12; i++) {

hist3MonthAvgPPT[i] = hist3MonthAvgTotalPPT[i] /
numYears;

out.writeData(hist3MonthAvgPPT[i] + ", " +
hist3MonthAvgTotalPPT[i]);
}

// to compute 18 month average historical ppt
monthCount = 6;
for (int i = 18; i < numMonths; i++) {
for (int j = 0; j < 18; j++) {
eighteenMonthPPT[i] = eighteenMonthPPT[i] +
monthlyPPT[i - j];

}

eighteenMonthPPT[i] = eighteenMonthPPT[i] / 18.0;
}

// accumulate the 18 month average historical ppt
for (int i = 0; i < numMonths; i++) {

hist18MonthAvgTotalPPT[monthCount] =
hist18MonthAvgTotalPPT[monthCount] +
eighteenMonthPPT[i];
monthCount++;
```

```
        if (monthCount > 11) {
        monthCount = 0;
        }
        }
        out.writeData("");
        out.writeData("eighteenMonthPPTAvg, PPT Total");
        // average the accumulated 18 month ppt for the 38 years of record
        for (int i = 0; i < 12; i++) {

        hist18MonthAvgPPT[i] = hist18MonthAvgTotalPPT[i] /
        numYears;

        out.writeData(hist18MonthAvgPPT[i] + ", " +
        hist18MonthAvgTotalPPT[i]);
        }

        // to initialize the minimum
        double minimum = monthlyFlow[0];

        // to compute a running total
        double sumLowestAvgSummerMonthFlow = 0.0;

        out.writeData("");
        out.writeData("Lowest average summer month flow");
        // i represents the year count
        for (int i = 0; i < numYears; i++) {

        for (int j = 11 * i + i; j <= (11 * i + i) + 11; j++) {
        if (monthlyFlow[j] < minimum) {
        minimum = monthlyFlow[j];
        }
        }
        lowestAvgSummerMonthFlow[i] = minimum;

        // to compute the running total
        sumLowestAvgSummerMonthFlow =
        sumLowestAvgSummerMonthFlow +
        lowestAvgSummerMonthFlow[i];

        // to reset the minimum
        minimum = 9999.9;
        out.writeData(lowestAvgSummerMonthFlow[i]);
        }

        out.writeData("");
        out.writeData(numYears + " year average");
        out.writeData(sumLowestAvgSummerMonthFlow / numYears);
        out.writeData("");
        out.writeData(numYears + " year total");
        out.writeData(sumLowestAvgSummerMonthFlow);

        out.closeFile();
        //}}}
        }


        /**
         *  This method takes in daily ppt and daily flow, together with a lag
         * time (in days), and computes the monthly rainfall-runoff
         * relationship for the summer season (i.e., Jun, Jul, Aug, Sep).  The
         * ppt is the variable that is lagged.  If the lag is 7 days for example,
         * the method will start the ppt sum 7 days before the start of that
         * month and finish 7 days before the end of that month.
         *
```

```
 * @param  pptFile          Daily ppt input file name
 * @param  flowFile         Daily flow input file name
 * @param  outPPTFile       Output ppt file name
 * @param  outFlowFile      Output flow file name
 * @param  lagTime          Lag time, in days
 * @param  dir              Directory
 * @param  outFile          Monthly ppt-flow file
 * @exception  IOException  Input Output Exception
 */
public static void calcContRainfallRunoffMonthly(String pptFile,
String flowFile, int lagTime, String dir, String outPPTFile,
String outFlowFile, String outFile) throws IOException {
//{{{

// the is the starting date of the historical record:
// 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 1;
int startHour = 1;
int startMinute = 0;
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

// this is the data that is read from the files
double currentPPT = 0.0;
double currentFlow = 0.0;

// to define the variables needed for the monthly output
double monthlyPPT = 0.0;
double tempMonthlyPPT = 0.0;
double monthlyFlow = 0.0;

// to define the variables that will hold the flow each season
// the arrays are of size 4 because it holds Jun, Jul, Aug, Sep
double[] seasonalPPT = new double[4];
double[] seasonalFlow = new double[4];

// the index of the min flow---this is needed because the ppt
// corresponding to the min flow is to be outputted
int indexOfMinFlow = 0;
double seasonalMinFlow = 9999.9;
double correspondingPPT = 9999.9;

// initialize the arrays to some big number
for (int i = 0; i < 4; i++) {
seasonalPPT[i] = 9999.9;
seasonalFlow[i] = 9999.9;
}

// to define DataReader and DataWriter objects
DataReader ppt = new DataReader(dir + pptFile);
DataReader flow = new DataReader(dir + flowFile);

DataWriter outMonthlyPPT = new DataWriter(dir + outPPTFile);
DataWriter outMonthlyFlow = new DataWriter(dir + outFlowFile);
DataWriter outMonthly = new DataWriter(dir + outFile);
DataWriter outMonthlyMin = new DataWriter(dir + "Min" + outFile);

// to write the headers
outMonthly.writeData("Date, MonthlyPPT, MonthlyFlow");
outMonthlyMin.writeData("Date, MonthlyPPT, MonthlyFlow");

// the number of days in the file
```

```
int numDays = flow.countDataPoints();

// the number of days to hold
int numDaysToHold = 90;

// the array that holds ppt for the previous 365 days
double[] previousPPT = new double[numDaysToHold];

// this is used for shifting
double[] tempPreviousPPT = new double[numDaysToHold];

// the loop that computes (i.e., accumulates daily values into
// monthly averages)
for (int i = 0; i < numDays; i++) {

// this is here because the first day is already set by
// the constructor of the ModelDate object
if (i != 0) {
// increments the date by one day
currentDate.incrementDateByDays(1);
}

// to read the data
currentPPT = ppt.readCurrentData();
currentFlow = flow.readCurrentData();

// to store the previous ppt array and update it
//{{{
if (i < numDaysToHold) {
// to fill the arrays holding the ppt
previousPPT[i] = currentPPT;
tempPreviousPPT[i] = previousPPT[i];
} else {
// shift the ppt so that the first value in the
// array is deleted
for (int j = 1; j < numDaysToHold; j++) {
tempPreviousPPT[j - 1] = previousPPT[j];
}
tempPreviousPPT[numDaysToHold - 1] = currentPPT;

// to update the previousPPT
for (int j = 0; j < numDaysToHold; j++) {
previousPPT[j] = tempPreviousPPT[j];
}
}
//}}}

// for the monthly rainfall-runoff
// to accumulate the monthly ppt and flow; note how the
// ppt is delayed by the variable lagTime
tempMonthlyPPT = tempMonthlyPPT +
previousPPT[numDaysToHold - lagTime];

monthlyFlow = monthlyFlow + currentFlow;

if (currentDate.getDay() == currentDate.getDaysInMonth()) {

monthlyFlow = monthlyFlow /
currentDate.getDaysInMonth();
monthlyPPT = tempMonthlyPPT;

// only write the summer season
if ((currentDate.getMonth() >= 5) &&
(currentDate.getMonth() <= 8)) {
```

```
// writes the seasonal ppt-flow curve
outMonthly.writeData(
currentDate.getMonth() + ", " +
monthlyPPT + ", " +
monthlyFlow);

// to store the data into its arrays
// the value of currentDate.getMonth()
// here is always 5,6,7,8, so that
// currentDate.getMonth()-5 will always
// be 0,1,2,3---exactly what the arrays need

seasonalFlow[currentDate.getMonth() - 5] =
monthlyFlow;

seasonalPPT[currentDate.getMonth() - 5] =
monthlyPPT;

if (currentDate.getMonth() == 8) {
// finds the min
for (int j = 0; j < 4; j++) {
if (seasonalFlow[j] < seasonalMinFlow) {
seasonalMinFlow = seasonalFlow[j];
correspondingPPT = seasonalPPT[j];
indexOfMinFlow = j;
}
}

// writes the annual minimum data
outMonthlyMin.writeData(
(indexOfMinFlow + 5) + ", " +
correspondingPPT + ", " +
seasonalMinFlow);

outMonthlyPPT.writeData(correspondingPPT);
outMonthlyFlow.writeData(seasonalMinFlow);

// reset the seasonalFlow and correspondingPPT
seasonalMinFlow = 9999.9;
correspondingPPT = 9999.9;

}
}

// to reset
tempMonthlyPPT = 0.0;
monthlyFlow = 0.0;
}
}

// to close the files
outMonthlyPPT.closeFile();
outMonthlyFlow.closeFile();
outMonthly.closeFile();
outMonthlyMin.closeFile();
//}}}
}


/**
 *  This method is similar to the calcHistDroughtInfo(), but just
 * outputs the monthly ppt and flow data for a particular month.  This
 * is because the precipitation-discharge relationship needs to be done
```

```
 * separetely for each month of the year.  This is in line with what
 * Linsley and Franzini (1979) say on p.49.  The method takes in daily
 * values of ppt and flow, and it produces monthly totals (for Jun, Jul,
 * Aug, Sep).  The monthly ppt is delayed by one month. THIS WILL NOT BE
 * USED.
 *
 * @param  pptFile          Daily ppt input file name
 * @param  flowFile         Daily flow input file name
 * @param  outFile          Output of montly ppt and fLow together
 * @param  month            Month for which to compute the relationship
 * @param  outDir           Output directory
 * @param  outPPTFile       Output of monthly ppt only
 * @param  outFlowFile      Output of monthly flow only
 * @exception  IOException  Input Output Exception
 */
public static void calcContInvLink(String pptFile, String flowFile,
String outDir, String outPPTFile, String outFlowFile, String outFile,
int month) throws IOException {
//{{{
// the is the starting date of the historical record:
// 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 1;
int startHour = 1;
int startMinute = 0;
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

// to define DataReader and DataWriter objects
String monthID = "";
if (month == 5) {
monthID = "Jun";
} else if (month == 6) {
monthID = "Jul";
} else if (month == 7) {
monthID = "Aug";
} else if (month == 8) {
monthID = "Sep";
}

// reset the monthID because this is going to be done once per
// season, and not once per month; I'll still keep the identifier
// as I still might need it
monthID = "";

DataReader ppt = new DataReader(outDir + pptFile);
DataReader flow = new DataReader(outDir + flowFile);
DataWriter outPPT = new DataWriter(outDir + monthID +
outPPTFile);
DataWriter outFlow = new DataWriter(outDir + monthID +
outFlowFile);
DataWriter out = new DataWriter(outDir + monthID +
outFile);

// variables that accumulate daily values into monthly values
double tempMonthlyPPT = 0.0;
double tempMonthlyFlow = 0.0;
/*
 *  // to make sure the input files are of the same size
 *  if (ppt.countDataPoints() != flow.countDataPoints()) {
 *  System.out.println("Input files are different sizes!");
 *  System.out.println("Simulation terminated!");
 *  System.exit(0);
```

```
 *  }
 */
// the number of days in the file
int numDays = flow.countDataPoints();

// compute the number of years; note that Java rounds down
// integer division
int numYears = numDays / 365;

// numer of months in the file
int numMonths = numYears * 12;

// arrays where monthly data is stored
double[] monthlyPPT = new double[numMonths];
double[] monthlyFlow = new double[numMonths];

// a variable that counts the total number of months
int totalMonthCount = 0;

out.writeData("Date, MonthlyPPT, MonthlyFlow");

// the loop that computes (i.e., accumulates daily values into
// monthly averages)
for (int i = 0; i < numDays; i++) {

// this is here because the first day is already set by
// the constructor of the ModelDate object
if (i != 0) {
// increments the date by one day
currentDate.incrementDateByDays(1);
}

tempMonthlyPPT = tempMonthlyPPT + ppt.readCurrentData();
tempMonthlyFlow = tempMonthlyFlow + flow.readCurrentData();

// at the end of the month, store monthly values in
// arrays
if (currentDate.getDay() == currentDate.getDaysInMonth()) {

monthlyPPT[totalMonthCount] = tempMonthlyPPT;
monthlyFlow[totalMonthCount] = tempMonthlyFlow /
currentDate.getDaysInMonth();

// this is to output the Jun, Jul, Aug and Sep
// precipitation and flow for use in the inverse
// link; added 15 Jun 2006

if ((currentDate.getMonth() >= 6) &&
(currentDate.getMonth() <= 8)) {
outPPT.writeData(monthlyPPT[totalMonthCount - 1]);
outFlow.writeData(monthlyFlow[totalMonthCount]);

// the previous month's ppt is outputted
// with this month's flow
out.writeData(currentDate.getMonth() + ", " +
monthlyPPT[totalMonthCount - 1] + ", " +
monthlyFlow[totalMonthCount]);
}

// to increment the totalMonthCount
totalMonthCount++;

// to reset
tempMonthlyPPT = 0.0;
```

```
tempMonthlyFlow = 0.0;
}
}

// to close the files
outPPT.closeFile();
outFlow.closeFile();
out.closeFile();
//}}}
}


/**
 *  Similar calculations as above, except that it produces yearly
 * precipitation-discharge curve.  THIS WILL NOT BE USED.
 *
 * @param   pptFile          Daily ppt input file name
 * @param   flowFile         Daily flow input file name
 * @param   outFile          Output of montly ppt and fLow together
 * @param   outDir           Output directory
 * @param   outPPTFile       Output of yearly ppt only
 * @param   outFlowFile      Output of yearly flow only
 * @exception  IOException   Input Output Exception
 */
public static void calcContInvLinkYearly(String pptFile, String flowFile,
String outDir, String outPPTFile, String outFlowFile,
String outFile) throws IOException {
//{{{
// the is the starting date of the historical record:
// 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 1;
int startHour = 1;
int startMinute = 0;
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

DataReader ppt = new DataReader(outDir + pptFile);
DataReader flow = new DataReader(outDir + flowFile);
DataWriter outPPT = new DataWriter(outDir + outPPTFile);
DataWriter outFlow = new DataWriter(outDir + outFlowFile);
DataWriter out = new DataWriter(outDir + outFile);

// variables that accumulate daily values into monthly values
double tempYearlyPPT = 0.0;
double tempYearlyFlow = 0.0;

// the number of days in the file
int numDays = flow.countDataPoints();

// compute the number of years; note that Java rounds down
// integer division
int numYears = numDays / 365;

// numer of months in the file
int numMonths = numYears * 12;

// arrays where monthly data is stored
double[] yearlyPPT = new double[numMonths];
double[] yearlyFlow = new double[numMonths];

// a variable that counts the total number of years
int totalYearCount = 0;
```

```
out.writeData("Year, YearlyPPT, YearlyFlow");

// the loop that computes (i.e., accumulates daily values into
// monthly averages)
for (int i = 0; i < numDays; i++) {

// this is here because the first day is already set by
// the constructor of the ModelDate object
if (i != 0) {
// increments the date by one day
currentDate.incrementDateByDays(1);
}

tempYearlyPPT = tempYearlyPPT + ppt.readCurrentData();
tempYearlyFlow = tempYearlyFlow + flow.readCurrentData();

// at the end of the year, store values in arrays
if (currentDate.getDayOfYear() == currentDate.getDaysInYear()) {

yearlyPPT[totalYearCount] = tempYearlyPPT;
yearlyFlow[totalYearCount] = tempYearlyFlow /
currentDate.getDaysInYear();

// this is to output the Jun, Jul, Aug and Sep
// precipitation and flow for use in the inverse
// link; added 15 Jun 2006

outPPT.writeData(yearlyPPT[totalYearCount]);
outFlow.writeData(yearlyFlow[totalYearCount]);

out.writeData(currentDate.getYear() + ", " +
yearlyPPT[totalYearCount] + ", " +
yearlyFlow[totalYearCount]);

// to increment the totalYearCount
totalYearCount++;

// to reset
tempYearlyPPT = 0.0;
tempYearlyFlow = 0.0;
}
}

// to close the files
outPPT.closeFile();
outFlow.closeFile();
out.closeFile();
//}}}
}


/**
 *  This method takes a file with a monthly time step, and extracts the
 * values for that month only, and stores them in a separate file. I
 * needed this when I was trying to do an inverse link with the drought
 * level.  THIS WILL NOT BE USED.
 *
 * @param  month            Month to be extracted from the record, 0=Jan, 1=Feb...
 * @param  dir              Directory of the original file
 * @param  fileName         Name of the original file
 * @exception  IOException  Description of the Exception
 */
public static void extractMonthlyData(String dir, String fileName,
```

```
int month) throws IOException {
//{{{
// the is the starting date of the historical record:
// 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 31;
int startHour = 1;
int startMinute = 0;
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

double droughtLevel = 0.0;

// to define DataReader and DataWriter objects
String monthID = "";
if (month == 5) {
monthID = "Jun";
} else if (month == 6) {
monthID = "Jul";
} else if (month == 7) {
monthID = "Aug";
} else if (month == 8) {
monthID = "Sep";
}

DataReader in = new DataReader(dir + fileName);
int n = in.countDataPoints();

DataWriter out = new DataWriter(dir + monthID + fileName);

// to extract the values
for (int i = 0; i < n; i++) {
// this is here because the first day is already set by
// the constructor of the ModelDate object
if (i != 0) {
// increments the date by one day
currentDate.incrementDateByMonths(1);
}

droughtLevel = in.readCurrentData();

if (currentDate.getMonth() == month) {
// write to a file
out.writeData((int) droughtLevel);
}
}

out.closeFile();
//}}}
}


/**
 *  This method extracts the annual series of the ppt input data.  This
 * method is needed because the AnnualSeries object was written to compute
 * statistics of flows only.  The methods finds annual maximum daily and
 * annual minimum monthly ppt, and writes it to a file.
 *
 * @param  fileName        File name of the daily ppt
 * @param  dir             Directory where the ppt file is located
 * @exception  IOException  Input Output Exception
 */
public static void computePrecipStatsDaily(String dir, String fileName)
```

```
 throws IOException {
//{{{

// to read the data from a file
DataReader pptIn = new DataReader(dir + fileName);

// to write the data to a file
DataWriter pptOutDaily = new DataWriter(dir + "MaxDaily" + fileName);
DataWriter pptOutMonthly = new DataWriter(dir + "MinMonthly" + fileName);

// set initial date as 01 Jan 2001 01:00
int startYear = 2001;
int startMonth = 0;
int startDay = 1;
int startHour = 1;
int startMinute = 0;
ModelDate currentDate = new ModelDate(startYear, startMonth,
startDay, startHour, startMinute);

int numDays = pptIn.countDataPoints();
double pptCurrent = 0.0;
double pptMax = -9999.9;

double pptCurrentMonthly = 0.0;
double pptMinMonthly = 9999.9;

for (int i = 0; i < numDays; i++) {

pptCurrent = pptIn.readCurrentData();

// for the daily annual maximums
if (pptCurrent > pptMax) {
pptMax = pptCurrent;
}

if (currentDate.getDayOfYear() == currentDate.getDaysInYear()) {
// output the ppt value
pptOutDaily.writeData(pptMax);

// to reset the pptMax
pptMax = -9999.9;
}

// for the monthly annual min
// sums the daily into monthly
pptCurrentMonthly = pptCurrentMonthly + pptCurrent;

if (currentDate.getDay() == currentDate.getDaysInMonth()) {

// this if statement was added so that summer
// annual monthly minimums would be picked out
// this if statement works; verified; OK!

//if ((currentDate.getMonth() > 4) &&
// (currentDate.getMonth() < 9)) {

if (pptCurrentMonthly < pptMinMonthly) {
pptMinMonthly = pptCurrentMonthly;
}
//}

if (currentDate.getDayOfYear() ==
currentDate.getDaysInYear()) {
pptOutMonthly.writeData(pptMinMonthly);
```

```
// reset the min
pptMinMonthly = 9999.9;
}

// reset the monthly
pptCurrentMonthly = 0.0;
}

currentDate.incrementDateByDays(1);
}
pptOutDaily.closeFile();
pptOutMonthly.closeFile();
//}}}
}


/**
 *  This method takes in annual data (like annual maximum daily
 * precipitation, annual maximum daily flow, annual minimum monthly flow,
 * annual minimum 7 day flow) and fits it to a distribution.  It prepares
 * the file so that it can be readily plotted in Grapher or Excel.
 *
 * @param   dir             Directory of the input file
 * @param   fileName        File name of the annual data
 * @param   distribution    Distribution either Gumbel, Weibull, LP3
 * @exception  IOException  Input Output Exception
 */
public static void fitStatisticalDistribution(String dir, String fileName,
String distribution) throws IOException {
//{{{
// create a file to write the fitted x
DataWriter out = new DataWriter(dir + distribution + fileName);

// this is the fit for the UTRCA return periods
DataWriter outUTRCA = new DataWriter(dir + distribution + "UTRCA" + fileName);

// input file
DataReader in = new DataReader(dir + fileName);

int n = in.countDataPoints();
double[] x = new double[n];
double[] xSorted = new double[n];

for (int i = 0; i < n; i++) {
x[i] = in.readCurrentData();
}

// sorts the array x from smallest to largest
xSorted = sortArray(x);

// find mean and stDev of the x
double mean;
double stDev;

// the temporary variables that are used to get mean and stDev
double sumX = 0.0;
double sumXMinusMeanSquared = 0.0;

// to calculate the mean
for (int i = 0; i < n; i++) {
sumX = sumX + x[i];
}
mean = sumX / n;
```

```java
// to calculate the stDev
for (int i = 0; i < n; i++) {
sumXMinusMeanSquared = sumXMinusMeanSquared +
Math.pow((x[i] - mean), 2.0);
}
stDev = Math.sqrt(sumXMinusMeanSquared / n);

// the probability and the return period of the x
// F is the probability of exceedence---used for max
double[] F = new double[n];
double[] RP = new double[n];

for (int i = 0; i < n; i++) {
F[i] = (i + 1) / (double) (n + 1);
}

// for the return periods that UTRCA uses
double[] utrcaRP = {2.0, 5.0, 10.0, 25.0, 50.0, 100.0,
250.0, 500.0};
int nSmall = utrcaRP.length;

if (distribution == "Gumbel") {

// to calculate the parameters of the Gumbel distribution
// based on the method of moments
double alpha = 1.282 / stDev;
double u = mean - (0.577 / alpha);

// the Gumbel variate for the original x
double[] z = new double[n];

double[] xFitted = new double[n];

for (int i = 0; i < n; i++) {

// in Java, log = base with natural logarithm e
// and log10 = base 10 logarithm
z[i] = -1 * Math.log(-1 * Math.log(F[i]));
xFitted[i] = (z[i] / alpha) + u;

// exceedence return period
RP[i] = 1.0 / (1.0 - F[i]);
}

out.writeData("Rank, Data, DataFitted, F, z, RP");
for (int i = 0; i < n; i++) {
out.writeData((i + 1) + ", " + xSorted[i] + ", " +
xFitted[i] + ", " + F[i] + ", " + z[i] + ", " + RP[i]);
}

// for the UTRCA return periods
double[] utrcaF = new double[nSmall];
double[] utrcaZ = new double[nSmall];
double[] utrcaXFitted = new double[nSmall];

for (int i = 0; i < nSmall; i++) {
utrcaF[i] = 1.0 - (1.0 / utrcaRP[i]);
utrcaZ[i] = -1 * Math.log(-1 * Math.log(utrcaF[i]));
utrcaXFitted[i] = (utrcaZ[i] / alpha) + u;
}
outUTRCA.writeData("utrcaXFitted, RP");
for (int i = 0; i < nSmall; i++) {
outUTRCA.writeData(utrcaXFitted[i] + ", " + utrcaRP[i]);
```

```
    }

    } else if (distribution == "Weibull") {
    // to calculate the parameters of the distribution based
    // on the method of least squares---this is easier than
    // using the method of moments, which requires an iterative
    // schemes and the approximation to the gamma function

    // for explanation of the equations, see my graduate notes
    // from ES520 Statistics and Reliability, p.90

    double[] z = new double[n];

    // b = sumLnX
    double sumLnX = 0.0;

    // a = sumLnXSquared
    double sumLnXSquared = 0.0;

    double sumF = 0.0;

    // d = sumZ
    double sumZ = 0.0;

    // c = sumLnXZ
    double sumLnXZ = 0.0;

    for (int i = 0; i < n; i++) {

    // the F used here is one for exceedence
    z[i] = Math.log(-1 * Math.log(1 - F[i]));
    sumLnX = sumLnX + Math.log(xSorted[i]);
    sumLnXSquared = sumLnXSquared +
    Math.pow(Math.log(xSorted[i]), 2.0);
    sumF = sumF + F[i];
    sumZ = sumZ + z[i];
    sumLnXZ = sumLnXZ + (Math.log(xSorted[i]) * z[i]);
    }

    double b = sumLnX;
    double a = sumLnXSquared;
    double d = sumZ;
    double c = sumLnXZ;

    // parameters of the distribution
    double u = Math.exp((c * b - d * a) / (c * n - b * d));
    double k = d / (b - n * Math.log(u));

    // to get the fitted x
    double[] xFitted = new double[n];
    for (int i = 0; i < n; i++) {
    xFitted[i] = Math.exp((z[i] / k) + Math.log(u));

    // non-exceedence return period; used for plotting
    RP[i] = 1.0 / (F[i]);
    }

    out.writeData("Rank, Data, DataFitted, F, z, RP");
    for (int i = 0; i < n; i++) {
    out.writeData((i + 1) + ", " + xSorted[i] + ", " +
    xFitted[i] + ", " + F[i] + ", " + z[i] + ", " + RP[i]);
    }

    // for the UTRCA return periods
```

```
double[] utrcaF = new double[nSmall];
double[] utrcaZ = new double[nSmall];
double[] utrcaXFitted = new double[nSmall];

for (int i = 0; i < nSmall; i++) {
utrcaF[i] = 1.0 / utrcaRP[i];
utrcaZ[i] = Math.log(-1 * Math.log(1 - utrcaF[i]));
utrcaXFitted[i] = Math.exp((utrcaZ[i] / k) + Math.log(u));
}
outUTRCA.writeData("utrcaXFitted, RP");
for (int i = 0; i < nSmall; i++) {
outUTRCA.writeData(utrcaXFitted[i] + ", " + utrcaRP[i]);
}

} else if (distribution == "LP3") {

// calculations here are from Maidment's Water Resources
// Engineering (2005), p. 325

// for Log Pearson III we need the data sorted from
// largest to smallest
double[] xSortedTemp = new double[n];
for (int i = 0; i < n; i++) {
xSortedTemp[i] = xSorted[n - i - 1];
}

double sumLogX = 0.0;
double sumLogXSquared = 0.0;
double sumLogXCubed = 0.0;

for (int i = 0; i < n; i++) {
sumLogX = sumLogX + Math.log10(xSortedTemp[i]);
sumLogXSquared = sumLogXSquared +
Math.pow(Math.log10(xSortedTemp[i]), 2.0);
sumLogXCubed = sumLogXCubed +
Math.pow(Math.log10(xSortedTemp[i]), 3.0);
}

double meanLogX = sumLogX / n;
double stDevLogX = Math.sqrt((sumLogXSquared -
Math.pow(sumLogX, 2.0) / n) / (n - 1));

double skewnessLogX = (n * n * sumLogXCubed - 3 * n *
sumLogX * sumLogXSquared +
2 * Math.pow(sumLogX, 3.0)) / (n * (n - 1) * (n - 2) *
Math.pow(stDevLogX, 3.0));

// converts a return period to a standard normal variate
// this is needed for the estimation of the frequency
// factor K, based on an approximation
double k = skewnessLogX / 6.0;
double[] z = new double[n];
double[] K = new double[n];
double[] logQFitted = new double[n];
double[] QFitted = new double[n];

for (int i = 0; i < n; i++) {

// exceedence return period
RP[i] = 1.0 / (1.0 - F[i]);

// converts the return period to the standard
// normal variate
z[i] = NORMSINV(1.0 - (1.0 / RP[i]));
```

```
// approximation for the frequency factor
K[i] = z[i] + (Math.pow(z[i], 2.0) - 1.0) * k +
(1.0 / 3.0) * (Math.pow(z[i], 3.0) -
6.0 * z[i]) * Math.pow(k, 2.0) -
(Math.pow(z[i], 2.0) - 1) *
Math.pow(k, 3.0) + z[i] *
Math.pow(k, 4.0) + (1.0 / 3.0) *
Math.pow(k, 5.0);

logQFitted[i] = meanLogX + K[i] * stDevLogX;
QFitted[i] = Math.pow(10.0, logQFitted[i]);


}

double[] utrcaZ = new double[nSmall];
double[] utrcaK = new double[nSmall];
double[] utrcaLogQFitted = new double[nSmall];
double[] utrcaQFitted = new double[nSmall];

for (int i = 0; i < nSmall; i++) {
utrcaZ[i] = NORMSINV(1.0 - (1.0 / utrcaRP[i]));
utrcaK[i] = utrcaZ[i] + (Math.pow(utrcaZ[i], 2.0) - 1.0) * k +
(1.0 / 3.0) * (Math.pow(utrcaZ[i], 3.0) -
6.0 * utrcaZ[i]) * Math.pow(k, 2.0) -
(Math.pow(utrcaZ[i], 2.0) - 1) *
Math.pow(k, 3.0) + utrcaZ[i] *
Math.pow(k, 4.0) + (1.0 / 3.0) *
Math.pow(k, 5.0);

utrcaLogQFitted[i] = meanLogX + (utrcaK[i] * stDevLogX);
utrcaQFitted[i] = Math.pow(10.0, utrcaLogQFitted[i]);
}

out.writeData("Rank, Data, DataFitted, K, RP");
for (int i = 0; i < n; i++) {
out.writeData((i + 1) + ", " + xSorted[i] + ", " +
QFitted[i] + ", " + K[i] + ", " + RP[i]);
}

outUTRCA.writeData("utrcaQFitted, utrcaRP");
for (int i = 0; i < nSmall; i++) {
outUTRCA.writeData(utrcaQFitted[i] + ", " + utrcaRP[i]);
}

} else {
System.out.println("Unknown Distribution!");
System.out.println("Please enter valid distribution.");
}

out.closeFile();
outUTRCA.closeFile();
//}}}
}


/**
 *  An approximation of the inverse of the normal distribution function;
 * The input parameter is the probability, or the area under the normal
 * distribution function.  The output is the standard normal variate, z.
 * The approximation comes from 26.2.23 in Abramowitz, M. and I. A.
 * Stegun, Handbook of Mathematical Functions, Dover, 1972, p. 933.
 *
 * @param  a  Probability
```

```
 * @return     Standard normal variate, z
 */
static double NORMSINV(double a) {
//{{{
double out = 0.0;

if (a <= 0.5) {
double t = Math.sqrt(Math.log(1.0 / (Math.pow(a, 2.0))));
out = (-t + (2.515517 + 0.802853 * t + 0.010328 * t * t) /
(1.0 + 1.432788 * t + 0.189269 * t * t + 0.001308 *
t * t * t));
} else if (a > 0.5) {
double t = Math.sqrt(Math.log(1.0 / (Math.pow((1.0 - a), 2.0))));
out = (t - (2.515517 + 0.802853 * t + 0.010328 * t * t) /
(1.0 + 1.432788 * t + 0.189269 * t * t + 0.001308 * t * t * t));
}
return out;
//}}}
}


/**
 *  This method takes in an array, and returns the array sorted from
 * smallest to largest
 *
 * @param   a  Array to be sorted
 * @return     The sorted value of the array
 */
static double[] sortArray(double[] a) {
//{{{
for (int i = 0; i < a.length - 1; i++) {
for (int j = i + 1; j < a.length; j++) {
if (a[i] > a[j]) {
double temp = a[i];
a[i] = a[j];
a[j] = temp;
}
}
}
return a;
//}}}
}


/**
 *  Method that takes a data file with dataTimeStep
 * and converts it to userTimeStep in an number of
 * intervals.  THIS WILL NOT BE USED.
 *
 * @param   fileNameIn       Name of input file
 * @param   fileNameOut      Name of output file
 * @param   dataTimeStep     Time step of data
 * @param   userTimeStep     Time step specified by the user
 * @exception  IOException   Input Output Exception
 */
public static void adjustInputFile(String fileNameIn,
String fileNameOut, int dataTimeStep, int userTimeStep)
 throws IOException {
//{{{

double p = 0.0;

DataReader in = new DataReader(fileNameIn);
//[mm] in dataTimeStep
```

```
DataWriter out = new DataWriter(fileNameOut);
//[mm] in userTimeStep

int numDataPointsInFile = in.countDataPoints();

// main loop goes by the userTimeStep
for (int i = 0; i < numDataPointsInFile; i++) {
p = in.readCurrentData();
for (int j = 0; j < (dataTimeStep / userTimeStep); j++) {
out.writeData(p / (dataTimeStep / userTimeStep));
}
}
out.closeFile();
//}}}
}

}
```

D.8 DataWriter.java

```java
import java.io.*;

/**
 *  This class provides an interface from which data is written to a file
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class DataWriter {

// instance variables for text writing
String fileName;
PrintWriter FileOut;


/**
 * Constructor for the DataWriter object
 *
 * @param   fileName         File name to which data is written
 * @exception   IOException  Input Output Exception
 */
public DataWriter(String fileName) throws IOException {
this.fileName = fileName;

this.FileOut = new PrintWriter(
new BufferedWriter(new FileWriter(
this.fileName)));
}


/**
 *  Method writes a single double value to a file
 *
 * @param   dataPoint  Value to be written to a file
 */
public void writeData(double dataPoint) {
this.FileOut.println(dataPoint);
}


/**
 *  Method writes a single String value to a file.  This is the method
 * that will be used most often.
 *
 * @param   dataPoint  Value to be written to a file
 */
public void writeData(String dataPoint) {
this.FileOut.println(dataPoint);
}


/**
 *  A method to simply close the file
 */
public void closeFile() {
(this.FileOut).close();
}
}
```

D.9 DataReader.java

```java
import java.io.*;

/**
 *  This class provides an interface from which data can be read from a file
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class DataReader {

// instance variables
String fileName;
FileReader fr;
BufferedReader br;


/**
 * Constructor for the DataReader object
 *
 * @param  fileName         File name from which data is read from
 * @exception  IOException  Input Output Exception
 */
public DataReader(String fileName) throws IOException {
this.fileName = fileName;

fr = new FileReader(this.fileName);
br = new BufferedReader(fr);
}


/**
 *  This is a method that simply counts the number of data points in a file
 *
 * @return     The number of data points in the file
 */
public int countDataPoints() {

// This is the string that the program reads line by line
String record = null;

// This is the total number of records
int totRecords = 0;

// this method uses its own FileReader and BufferedReader
// because it only needs to do a count once
try {
FileReader frCount = new FileReader(this.fileName);
BufferedReader brCount = new BufferedReader(frCount);
record = new String();

while ((record = brCount.readLine()) != null) {
totRecords++;
}
} catch (IOException e) {
// To catch errors from readLine
System.out.println("IOException error occured");
e.printStackTrace();
}

return totRecords;
}
```

```
/**
 *  This method simply reads the first value of the time series data
 *
 * @return    The initial data point read from the file
 */
public double readInitialData() {
// just to intialize the output variable
double PPT = -9999.9;

// uses the instance variable br and fr because
// it needs to do this every time step and thus
// maintain continuity
try {
String record = new String();
record = (this.br).readLine();
PPT = Double.valueOf(record).doubleValue();

}
catch (IOException e) {
// To catch errors from readLine
System.out.println("IOException error occured");
e.printStackTrace();
}
return PPT;
}


/**
 *  This method reads the current value of the time series file when
 * the number to be read is of type double
 *
 * @return    Current value of the time series data as double
 */
public double readCurrentData() {
double PPT = -9999.9;

// uses the instance variable br and fr because
// it needs to do this every time step and thus
// maintain continuity
try {
String record = new String();
record = (this.br).readLine();
PPT = Double.valueOf(record).doubleValue();
}
// end try
catch (IOException e) {
// To catch errors from readLine
System.out.println("IOException error occured");
e.printStackTrace();
}
// end catch

return PPT;
}


/**
 *  This method reads the current value of the time series file when
 * the line to be read is of type String
 *
 * @return    Current value of the time series data as String
 */
public String readRecord() {
```

```
        double PPT = -9999.9;
        String line = "";

        // uses the instance variable br and fr because
        // it needs to do this every time step and thus
        // maintain continuity
        try {
        String record = new String();
        record = (this.br).readLine();
        line = record;
        //PPT = Double.valueOf(record).doubleValue();
        }
        // end try
        catch (IOException e) {
        // To catch errors from readLine
        System.out.println("IOException error occured");
        e.printStackTrace();
        }
        // end catch

        return line;
        }


        }
```

D.10   Table.java

```java
/**
 *  This class provides an interface where tabular functions can be stored
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class Table {
// instance variables
double[] x;
double[] y;


/**
 * Constructor for the Table object
 *
 * @param  x  An array of x values
 * @param  y  An array of y values
 */
public Table(double[] x, double[] y) {
// this.x refers to the x of the instance variable
this.x = x;
this.y = y;

if (this.x.length != this.y.length) {
System.out.println("Table input data not of same size.");
System.out.println("Terminating simulation!");
System.exit(0);
}
}


/**
 *  This method takes as input a single x value, and
 * outputs a singe y value based on linear interpolation
 * of the table.
 *
 * @param  xhat  x value to be looked up
 * @return       Resulting y value
 */
public double lookup(double xhat) {

int n = this.x.length;
double yhat;
// interpolated value

// if xhat exceeds x given in the table, then interpolated
// value becomes the last x value in table
if (xhat <= this.x[0]) {
//System.out.print("Interpolation value " + xhat);
//System.out.println(" too small; using " + this.x[0]);
return yhat = this.y[0];
} else
if (xhat >= this.x[n - 1]) {
//System.out.print("Interpolation value " + xhat);
//System.out.println(" too big; using " +this.x[n - 1]);
return yhat = this.y[n - 1];
}

// performs a linear search
int i = n - 1;
while ((this.x[i] > xhat) && (i > 0)) {
i = i - 1;
```

```
}

/*
 *  Performs linear interpolation after
 *  Recktenwald, G. (2000). Numerical Methods with Matlab:
 *  Implementation and Applications. Prentice-Hall, New Jersey.
 */
double L1 = (this.x[i + 1] - xhat) / (this.x[i + 1] - this.x[i]);
double L2 = (xhat - this.x[i]) / (this.x[i + 1] - this.x[i]);
yhat = this.y[i] * L1 + this.y[i + 1] * L2;

return yhat;
}

}
```

D.11   SubBasin.java

```java
/**
 *  This class is a parent class to SoilMoistureAccounting, Clark, and
 * LinearReservoir objects.  It simply groups them together into one unit.
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class SubBasin {

// instance variable declarations
//{{{
// main objects of a SubBasin
private SoilMoistureAccounting loss;
private LinearReservoir baseflow1;
private LinearReservoir baseflow2;
private Clark transform;

// these parameters are shared by more than one object
// (i.e., Clark, SoilMoistureAccounting, LinearReservoir)
private ModelDate currentDate;
private ETZone zone;
private int userTimeStep;

private double ppt;

// SoilMoistureAccounting parameters
private double imperviousness;
private double maxSurfStoreReductionCoeff;
private double maxSoilInfilReductionCoeff;

private double maxCanStore;
private double maxSurfStore;
private double maxSoilStore;
private double maxTensStore;
private double maxGW1Store;
private double maxGW2Store;

private double maxSoilInfil;
private double maxSoilPerc;
private double maxGW1Perc;
private double maxGW2Perc;

private double initCanStore;
private double initSurfStore;
private double initSoilStore;
private double initGW1Store;
private double initGW2Store;

private double routGW1Storage;
private double routGW2Storage;

// LinearReservoir parameters
private double storageCoeffGW1;
private double storageCoeffGW2;

private int numResGW1;
private int numResGW2;

// Clark parameters
private double basinArea;
private double timeOfConcentration;
private double storageCoeffClark;
```

```
// ETZone parameters
private double[] monthlyET;
private double panCoefficient;

// PET reduction coefficient; this reduces the PET
private double PETReductionCoeff;


//}}}
/**
 *  Constructor for the SubBasin object
 *
 * @param  currentDate   Current date
 * @param  userTimeStep  User time step, in [hrs]
 */
public SubBasin(ModelDate currentDate, int userTimeStep) {
this.currentDate = currentDate;
this.userTimeStep = userTimeStep;
this.maxSurfStoreReductionCoeff = 1.0;
this.maxSoilInfilReductionCoeff = 1.0;
}


// set methods are here
//{{{
/**
 *  Sets the physicalProps attribute of the SubBasin object
 *
 * @param  basinArea                 Area of the basin, in [km<SUP>2</SUP>]
 * @param  imperviousness            Impreviousness of the basin, in [%]
 */
public void setPhysicalProps(double basinArea, double imperviousness) {
this.basinArea = basinArea;
this.imperviousness = imperviousness;
}


/**
 *  Sets the maxStores attribute of the SubBasin object; this is where
 * the capacities of the storage buckets are set.
 *
 * @param  maxCanStore   Maximum Canopy Storage, in [mm]
 * @param  maxSurfStore  Maximum Surface Storage, in [mm]
 * @param  maxSoilStore  Maximum Soil Storage, in [mm]
 * @param  maxTensStore  Maximum Tension Storage, in [mm]
 * @param  maxGW1Store   Maximum GW1 Storage, in [mm]
 * @param  maxGW2Store   Maximum GW2 Storage, in [mm]
 */
public void setMaxStores(double maxCanStore, double maxSurfStore,
double maxSoilStore, double maxTensStore, double maxGW1Store,
double maxGW2Store) {

this.maxCanStore = maxCanStore;
this.maxSurfStore = maxSurfStore;
this.maxSoilStore = maxSoilStore;
this.maxTensStore = maxTensStore;
this.maxGW1Store = maxGW1Store;
this.maxGW2Store = maxGW2Store;
}


/**
 *  Sets the maxRates attribute of the SubBasin object; this is where
```

```
 * the max rates of flow between the buckets are set.
 *
 * @param  maxSoilInfil  Maximum Soil Infiltration, in [mm/hr]
 * @param  maxSoilPerc   Maximum Soil Percolation, in [mm/hr]
 * @param  maxGW1Perc    Maximum GW1 Percolation, in [mm/hr]
 * @param  maxGW2Perc    Maximum GW2 Percolation, in [mm/hr]
 */
public void setMaxRates(double maxSoilInfil, double maxSoilPerc,
double maxGW1Perc, double maxGW2Perc) {

this.maxSoilInfil = maxSoilInfil;
this.maxSoilPerc = maxSoilPerc;
this.maxGW1Perc = maxGW1Perc;
this.maxGW2Perc = maxGW2Perc;
}


/**
 *  Sets the initStores attribute of the SubBasin object; this is where
 * initial condition of the model are set.
 *
 * @param  initCanStore   Initial Canopy Storage, in [mm]
 * @param  initSurfStore  Initial Surface Storage, in [mm]
 * @param  initSoilStore  Initial Soil Storage, in [mm]
 * @param  initGW1Store   Initial GW1 Storage, in [mm]
 * @param  initGW2Store   Initial GW2 Storage, in [mm]
 */
public void setInitStores(double initCanStore, double initSurfStore,
double initSoilStore, double initGW1Store, double initGW2Store) {

this.initCanStore = (initCanStore / 100.0) * this.maxCanStore;
this.initSurfStore = (initSurfStore / 100.0) * this.maxSurfStore;
this.initSoilStore = (initSoilStore / 100.0) * this.maxSoilStore;
this.initGW1Store = (initGW1Store / 100.0) * this.maxGW1Store;
this.initGW2Store = (initGW2Store / 100.0) * this.maxGW2Store;
}


/**
 *  Sets the routGWStorage attribute of the SubBasin object; this is
 * where the storage coefficients of two groundwater layers are set.
 * These parameters are ones that route the flow out of
 * SoilMoistureAccounting object
 *
 * @param  routGW1Storage  Storage Coefficient of GW1 layer, in [hrs]
 * @param  routGW2Storage  Storage Coefficient of GW2 layer, in [hrs]
 */
public void setRoutGWStorage(double routGW1Storage,
double routGW2Storage) {
this.routGW1Storage = routGW1Storage;
this.routGW2Storage = routGW2Storage;
}


/**
 *  Sets the baseflowParams attribute of the SubBasin object; this is
 * where the base flow parameters are set.  These parameters correspond to
 * when flow that is received from SoilMoistureAccounting is routed.
 *
 * @param  storageCoeffGW1  Storage Coefficient of GW1 layer, in [hrs]
 * @param  numResGW1        Number of linear reservoirs in layer 1, [-]
 * @param  storageCoeffGW2  Storage Coefficient of GW2 layer, in [hrs]
 * @param  numResGW2        Number of linear reservoirs in layer 2, [-]
 */
```

```
public void setBaseflowParams(double storageCoeffGW1, int numResGW1,
double storageCoeffGW2, int numResGW2) {
this.storageCoeffGW1 = storageCoeffGW1;
this.numResGW1 = numResGW1;
this.storageCoeffGW2 = storageCoeffGW2;
this.numResGW2 = numResGW2;
}


/**
 *  Sets the clarkParams attribute of the SubBasin object; this is where
 * the Clark's method parameters are set.
 *
 * @param  timeOfConcentration  Time of Concentration, in [hrs]
 * @param  storageCoeffClark    Storage Coefficient, in [hrs]
 */
public void setClarkParams(double timeOfConcentration,
double storageCoeffClark) {
this.timeOfConcentration = timeOfConcentration;
this.storageCoeffClark = storageCoeffClark;
}


/**
 *  Sets the ETZoneParams attribute of the SubBasin object
 *
 * @param  monthlyET       Monthly Evapotranspiration, in [mm]
 * @param  panCoefficient  Pan coefficient, in [-]
 */
public void setETZoneParams(double[] monthlyET, double panCoefficient) {
this.monthlyET = monthlyET;
this.panCoefficient = panCoefficient;
}


// these are the feedback set methods
/**
 *  Sets the timeOfConcentration attribute of the SubBasin object
 *
 * @param  timeOfConcentration  The new timeOfConcentration value, in [hrs]
 */
public void setTimeOfConcentration(double timeOfConcentration) {
this.timeOfConcentration = timeOfConcentration;
}


/**
 *  Sets the imperviousness attribute of the SubBasin object
 *
 * @param  imperviousness  The new imperviousness value, in [%]
 */
public void setImperviousness(double imperviousness) {
this.imperviousness = imperviousness;
}


/**
 *  Sets the pETReductionCoeff attribute of the SubBasin object
 *
 * @param  PETReductionCoeff  The new PETReductionCoeff value, in [-]
 */
public void setPETReductionCoeff(double PETReductionCoeff) {
this.PETReductionCoeff = PETReductionCoeff;
}
```

```
/**
 *  Sets the maxSurfStoreReductionCoeff attribute of the SubBasin object
 *
 * @param  maxSurfStoreReductionCoeff  The new maxSurfStoreReductionCoeff
 * value, in [-]
 */
public void setMaxSurfStoreReductionCoeff(double maxSurfStoreReductionCoeff) {
this.maxSurfStoreReductionCoeff = maxSurfStoreReductionCoeff;
}


/**
 *  Sets the maxSoilInfilReductionCoeff attribute of the SubBasin object
 *
 * @param  maxSoilInfilReductionCoeff  The new maxSoilInfilReductionCoeff value
 */
public void setMaxSoilInfilReductionCoeff(double maxSoilInfilReductionCoeff) {
this.maxSoilInfilReductionCoeff = maxSoilInfilReductionCoeff;
}


//}}}

/**
 *  Initializes all parameters for this particular SubBasin object
 */
public void initialize() {

// declares and initializes everything
// ETZone object needs
this.zone = new ETZone(this.currentDate);
this.zone.setETZoneParams(this.monthlyET, this.panCoefficient);
this.zone.setPETReductionCoeff(this.PETReductionCoeff);

// declares and initializes everything
// SoilMoistureAccounting object needs
this.loss = new SoilMoistureAccounting(this.currentDate,
this.zone, this.userTimeStep, this.imperviousness);

this.loss.setMaxSurfStoreReductionCoeff(
this.maxSurfStoreReductionCoeff);
this.loss.setMaxSoilInfilReductionCoeff(
this.maxSoilInfilReductionCoeff);

this.loss.setMaxStores(this.maxCanStore, this.maxSurfStore,
this.maxSoilStore, this.maxTensStore,
this.maxGW1Store, this.maxGW2Store);
this.loss.setMaxRates(this.maxSoilInfil, this.maxSoilPerc,
this.maxGW1Perc, this.maxGW2Perc);
this.loss.setInitStores(this.initCanStore, this.initSurfStore,
this.initSoilStore, this.initGW1Store,
this.initGW2Store);
this.loss.setRoutGWStorage(this.routGW1Storage,
this.routGW2Storage);
this.loss.computeTimeStep();

// declares and initializes everything LinearReservoir
// objects need; these are two LinearReservoir objects for the
// two GW layers
this.baseflow1 = new LinearReservoir(this.storageCoeffGW1,
this.userTimeStep, this.numResGW1);
this.baseflow2 = new LinearReservoir(this.storageCoeffGW2,
```

```
this.userTimeStep, this.numResGW2);
this.baseflow1.initialize(0.0);
this.baseflow2.initialize(0.0);

// declares and initializes everything Clark object needs
this.transform = new Clark(this.basinArea,
this.timeOfConcentration, this.storageCoeffClark,
this.userTimeStep);
this.transform.initialize(0.0);

}


/**
 *  After the SubBasins parameters change, these changes must be
 * registered with the objects SubBasin object consists of, namely
 * SoilMoistureAccounting, LinearReservoir, Clark and ETZone.
 * This method updates the parameters of the SoilMoistureAccounting
 * object, Clark's object, two LinearReservoir objects, and the ETZone.
 * This method is called imediatelly after set methods in HydModel
 * object and is used in cases when the parameters need to be changed
 * during the course of the simulation.
 */
public void updateParams() {

this.loss.setImperviousness(this.imperviousness);
this.zone.setPETReductionCoeff(this.PETReductionCoeff);

this.loss.setMaxSurfStoreReductionCoeff(
this.maxSurfStoreReductionCoeff);
this.loss.setMaxSoilInfilReductionCoeff(
this.maxSoilInfilReductionCoeff);
this.loss.setMaxStores(this.maxCanStore, this.maxSurfStore,
this.maxSoilStore, this.maxTensStore,
this.maxGW1Store, this.maxGW2Store);
this.loss.setMaxRates(this.maxSoilInfil, this.maxSoilPerc,
this.maxGW1Perc, this.maxGW2Perc);
this.loss.setRoutGWStorage(this.routGW1Storage,
this.routGW2Storage);

this.baseflow1.setStorageCoeff(this.storageCoeffGW1);
this.baseflow2.setStorageCoeff(this.storageCoeffGW2);

this.transform.setClarkParams(this.timeOfConcentration,
this.storageCoeffClark);

}


/**
 *  Updates the parameters of the SubBasin object
 *
 * @param  ppt          Precipitation input [mm]
 * @param  currentDate  Current date
 */
public void update(double ppt, ModelDate currentDate) {

this.ppt = ppt;
this.currentDate = currentDate;

// updates the SoilMoistureAccounting object
this.loss.update(this.ppt, this.currentDate);

// checks that the adjTimeStep is the same as userTimeStep
```

```
this.loss.computeTimeStep();

// update the LinearReservoir objects
// note that this.loss.getGW1Outflow() is in [mm/hr]
// but this method needs it to be in [cms]; therefore
// multiply by basinArea and conversion coefficient
this.baseflow1.update(this.loss.getGW1Outflow() *
(1000.0 / 3600.0) * this.basinArea);
this.baseflow2.update(this.loss.getGW2Outflow() *
(1000.0 / 3600.0) * this.basinArea);

// updates the Clark object
// this.loss.getExcess() is in units of [mm] during
// the period of 6 hrs
this.transform.update(this.loss.getExcess());
}


// get methods are here
//{{{
/**
 *  Gets the directFlow attribute of the SubBasin object; this is where
 * the directFlow as calculated by Clark's method is outputted.
 *
 * @return    The direct flow value, in [cms]
 */
public double getDirectFlow() {
return this.transform.getDirectFlow();
}


/**
 *  Gets the initTotalFlow attribute of the SubBasin object; this is
 * where the initial total flow [works for initial time only] is returned
 *
 * @return    The initial total flow value, in [cms]
 */
public double getInitTotalFlow() {
return (this.transform.getDirectFlow() +
this.baseflow1.getInitOutflow() +
this.baseflow2.getInitOutflow());
}


/**
 *  Gets the totalFlow attribute of the SubBasin object
 *
 * @return    The total flow value, in [cms]
 */
public double getTotalFlow() {
return (this.transform.getDirectFlow() +
this.baseflow1.getOutflow() +
this.baseflow2.getOutflow());
}


/**
 *  Gets the initBaseflow attribute of the SubBasin object
 *
 * @return    The initial baseflow value, in [cms]
 */
public double getInitBaseflow() {
return (this.baseflow1.getInitOutflow() +
this.baseflow2.getInitOutflow());
```

```
}


/**
 *  Gets the baseflow attribute of the SubBasin object
 *
 * @return    The baseflow value, in [cms]
 */
public double getBaseflow() {
return (this.baseflow1.getOutflow() +
this.baseflow2.getOutflow());
}


/**
 *  Gets the excess attribute of the SubBasin object
 *
 * @return    The excess value, in [mm]
 */
public double getExcess() {
return this.loss.getExcess();
}


/**
 *  Gets the potEvapTransVol attribute of the SubBasin object
 * during the time period of userTimeStep [hrs]
 *
 * @return    The potEvapTransVol value, in units of [mm]
 */
public double getPotEvapTransVol() {
return this.loss.getPotEvapTransVol();
}


/**
 *  Gets the TTBeyondTensionZone attribute of the
 * SoilMoistureAccounting object during the period of userTimeStep [hrs]
 *
 * @return    The ETBeyondTensionZone value, in units of [mm]
 */
public double getETBeyondTensionZone() {
return this.loss.getETBeyondTensionZone();
}


/**
 *  Gets the GW1Outflow attribute of the SubBasin object
 *
 * @return    The GW1Outflow value, in [mm/hr]
 */
public double getGW1Outflow() {
return this.loss.getGW1Outflow();
}


/**
 *  Gets the GW2Outflow attribute of the SubBasin object
 *
 * @return    The GW2Outflow value, in [mm/hr]
 */
public double getGW2Outflow() {
return this.loss.getGW2Outflow();
}
```

```
/**
 *  Gets the actSoilInfil attribute of the SubBasin object
 *
 * @return    The actSoilInfil value
 */
public double getActSoilInfil() {
return this.loss.getActSoilInfil();
}


/**
 *  Gets the actSoilPerc attribute of the SubBasin object
 *
 * @return    The actSoilPerc value, in [mm]
 */
public double getActSoilPerc() {
return this.loss.getActSoilPerc();
}


/**
 *  Gets the actGW2Perc attribute of the SubBasin object; this method
 * converts the actGW2Perc from [mm] to [cms].
 *
 * @return    The actGW2Perc value, in [m<SUP>3</SUP>/yr]
 */
public double getActGW2Perc() {
return (this.loss.getActGW2Perc() / this.userTimeStep)
 * (1000.0 / 3600.0) * this.basinArea *
3600.0 * 24.0 * currentDate.getDaysInYear();
}


/**
 *  Gets the subBasinArea attribute of the SubBasin object
 *
 * @return    The subBasinArea value, in [km<SUP>2</SUP>]
 */
public double getSubBasinArea() {
return this.basinArea;
}


/**
 *  Gets the timeOfConcentration attribute of the SubBasin object
 *
 * @return    The timeOfConcentration value, in [hrs]
 */
public double getTimeOfConcentration() {
return this.transform.getTimeOfConcentration();
}


/**
 *  Gets the imperviousness attribute of the SubBasin object
 *
 * @return    The imperviousness value, in [%]
 */
public double getImperviousness() {
return this.loss.getImperviousness();
}
```

```
/**
 *  Gets the maxSurfStoreReduction attribute of the SubBasin object
 *
 * @return    The maxSurfStoreReduction value, in [-]
 */
public double getMaxSurfStoreReduction() {
return this.loss.getMaxSurfStoreReduction();
}


/**
 *  Gets the maxSoilInfilReduction attribute of the SubBasin object
 *
 * @return    The maxSoilInfilReduction value, in [-]
 */
public double getMaxSoilInfilReduction() {
return this.loss.getMaxSoilInfilReduction();
}


/**
 *  Gets the PETReductionCoeff attribute of the SubBasin object
 *
 * @return    The PETReductionCoeff value, in [-]
 */
public double getPETReductionCoeff() {
return this.zone.getPETReductionCoeff();
}


/**
 *  Gets the maxSoilInfil attribute of the SubBasin object
 *
 * @return    The maxSoilInfil value, in [mm/hr]
 */
public double getMaxSoilInfil() {
return this.loss.getMaxSoilInfil();
}


/**
 *  Gets the maxSurfStore attribute of the SubBasin object
 *
 * @return    The maxSurfStore value, in [mm]
 */
public double getMaxSurfStore() {
return this.loss.getMaxSurfStore();
}


/**
 *  Gets the canStore attribute of the SubBasin object
 *
 * @return    The canStore value, in [mm]
 */
public double getCanStore() {
return this.loss.getCanStore();
}


/**
 *  Gets the surfStore attribute of the SubBasin object
 *
```

```
 * @return    The surfStore value, in [mm]
 */
public double getSurfStore() {
return this.loss.getSurfStore();
}


/**
 *  Gets the soilStore attribute of the SubBasin object
 *
 * @return    The soilStore value, in [mm]
 */
public double getSoilStore() {
return this.loss.getSoilStore();
}


/**
 *  Gets the gW1Store attribute of the SubBasin object
 *
 * @return    The gW1Store value, in [mm]
 */
public double getGW1Store() {
return this.loss.getGW1Store();
}


/**
 *  Gets the gW2Store attribute of the SubBasin object
 *
 * @return    The gW2Store value, in [mm]
 */
public double getGW2Store() {
return this.loss.getGW2Store();
}
//}}}

}
```

D.12 ModifiedPuls.java

```java
/**
 *  This class represents Modified Puls routing method
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class ModifiedPuls {

// instance variables

// part of storage-outflow function
double[] storage;

// part of storage-outflow function
double[] outflow;

int timeStep;

// these are the internal variables used by the object
double[] indication;
Table io;
Table os;

// actual computed reservoir outflow
double[] resOutflow;

// actual computed reservour storage
// this is S/(dt) + (O/2)
double[] resIndication;


/**
 * Constructor for the ModifiedPuls object
 *
 * @param  storage    Storage array, in [thousands m<SUP>3</SUP>]
 * @param  outflow    Outflow array, in [cms]
 * @param  timeStep   Time step, in [hrs]
 */
public ModifiedPuls(double[] storage, double[] outflow, int timeStep) {

// units of storage are in [thousands m^3]
// units of outflow are in [m^3/sec]
// units of timeStep are in [hrs]
this.storage = storage;
this.outflow = outflow;
this.timeStep = timeStep;

// temporary variables that the constructor initializes
double[] indication = new double[this.storage.length];
Table io;
Table os;

// constructs the indication function
for (int i = 0; i < this.storage.length; i++) {
indication[i] = (this.storage[i] * 1000.0 /
(this.timeStep * 3600.0)) +
(this.outflow[i] / 2.0);
}

// sets the instance variable's indication function to the
// indication function calculated above
this.indication = indication;
```

```
io = new Table(indication, this.outflow);
os = new Table(this.outflow, this.storage);

this.io = io;
this.os = os;

// initializes the internal variables used for output
double[] resOutflow = new double[2];
double[] resIndication = new double[2];

this.resOutflow = resOutflow;
this.resIndication = resIndication;
}


/**
 *  This method initializes reservoir's storage and outflow values
 *
 * @param  inflow  Inflow, in [cms]
 */
public void initialize(double inflow) {
this.resOutflow[0] = inflow;
this.resIndication[0] = (this.os.lookup(this.resOutflow[0]) *
1000.0 / (this.timeStep * 3600.0) +
(this.resOutflow[0] / 2.0));
}


/**
 *  This method updates reservoir's storage and outflow values
 *
 * @param  previousInflow  Previous Inflow, in [cms]
 * @param  inflow          Inflow, in [cms]
 */
public void update(double previousInflow, double inflow) {
// to compute
double avgInflow = (previousInflow + inflow) / 2.0;
this.resIndication[1] = this.resIndication[0] - resOutflow[0] +
avgInflow;
this.resOutflow[1] = this.io.lookup(this.resIndication[1]);

// to reset
this.resIndication[0] = this.resIndication[1];
this.resOutflow[0] = this.resOutflow[1];
}


//
/**
 *  Gets the initOutflow attribute of the ModifiedPuls object
 * This method must be called after initialize
 *
 * @return    The initOutflow value, in [cms]
 */
public double getInitOutflow() {
return this.resOutflow[0];
}


/**
 *  Gets the outflow attribute of the ModifiedPuls object
 *
 * @return    The outflow value, in [cms]
```

```
 */
public double getOutflow() {
return this.resOutflow[1];
}



/**
 *  Gets the storage attribute of the ModifiedPuls object
 *
 * @return     The storage value, in [thousands m<SUP>3</SUP>]
 */
public double getStorage() {

// S/dt + O/2
return (this.os.lookup(this.resOutflow[0]));
}

}
```

D.13 SoilMoistureAccounting.java

```java
/**
 *  This class represents the Soil Moisture Accouting algorithm
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class SoilMoistureAccounting {

// instance variables
//{{{
private ModelDate currentDate;
private ETZone zone;
private int userTimeStep;

// physical properties of the basin, in [%]
private double imperviousness;

// a reduction ratio for the max surface storage, in [-]
private double maxSurfStoreReductionCoeff;

// a reduction ratio for the max soil infiltration, in [-]
private double maxSoilInfilReductionCoeff;

// upper and lower time step bounds [hrs]
private double upperBound;
private double lowerBound;

// internal variables used by the object
// maximum values of states (or stores) in [mm]
private double maxCanStore;
private double maxSurfStore;
private double maxSoilStore;
private double maxTensStore;
private double maxGW1Store;
private double maxGW2Store;

// maximum values of rates in [mm/hr]
private double maxSoilInfil;
private double maxSoilPerc;
private double maxGW1Perc;
private double maxGW2Perc;

// current states in [mm]
private double[] canStore = new double[2];
private double[] surfStore = new double[2];
private double[] soilStore = new double[2];
private double[] GW1Store = new double[2];
private double[] GW2Store = new double[2];

// storage coefficients of GW reservoirs [hrs]
private double routGW1Storage;
private double routGW2Storage;

// variables used to compute the rates in [mm/hr]
private double potEvapTrans;
private double potSoilInfil;
private double potSoilPerc;
private double potGW1Perc;
private double potGW2Perc;

// ppt intensity specified over the adjTimeStep in [mm/hr]
private double precipTimeStep;
```

```
// ppt volume specified over the adjTimeStep, in [mm]
private double precipTimeStepVolume;

// calculated time step [hrs]
private double calcTimeStep;
// min from 8 computed time steps
// adjusted time step [hrs]
private double adjTimeStep;

// temporary variables used to update the states, in [mm]
// ppt that goes beyond canopy
private double precipBeyondCanopy;
// available water for infiltration
private double availWater;
// actual soil infinltration
private double actSoilInfil;
// water available to fill surface storage
private double waterAvailFillSurfStore;
// excess water that leaves surface storage
private double surfExcess;
// water available for soil percolation
private double waterAvailSoilPerc;
// actual soil percolation
private double actSoilPerc;
// water available for GW1 percolation
private double waterAvailGW1Perc;
// actual GW1 percolation
private double actGW1Perc;
// water available for GW2 percolation
private double waterAvailGW2Perc;
// actual GW2 percolation
private double actGW2Perc;
// potEvapTrans as a volume
private double potEvapTransVol;
// evapotranspiration that goes beyond canopy
private double ETBeyondCanopy;
// evapotranspiration that goes beyond surface
private double ETBeyondSurface;
// evapotranspiration that goes beyond upper zone
private double ETBeyondUpperZone;
// evapotranspiration that goes beyond tension
// zone and never really gets satisfied
private double ETBeyondTensionZone;
// volume of water in the upper zone
private double upperZoneVol;
// volume of water in the tension zone
private double tensionZoneVol;
// volume that evaporates from the tension zone
private double actTensionZoneET;

// this is a table function that relates the amount of ET
// that can be drained from the tension zone of the soil
// actEvapSoil is now a fraction of the potEvapTrans
private Table tensionZoneTable;

// this is the excess from imperviousness
// this means if imperviousness is x percent, the excess
// from imperviousness is ppt * x/100
// this is the ppt that contributes to direct runoff
// because it falls on an impervious surface and thus
// is not subject to losses, in [mm]
private double excessFromImperv;

// this is total excess (surfExcess + excessFromImperv), in [mm]
```

```
private double excess;

// precipitation available to be lost
// computed as ppt - excessFromImperv
private double pptAvailToLoss;

// changes in storage volumes, in [mm]
private double dCanStore;
private double dSurfStore;
private double dSoilStore;
private double dGW1Store;
private double dGW2Store;

// GW flow variables, in [mm/hr]
private double[] flowGW1 = new double[2];
private double[] flowGW2 = new double[2];

// GW flow variables as volume, in [mm]
private double[] flowGW1Vol = new double[2];
private double[] flowGW2Vol = new double[2];


//}}}

/**
 *  Constructor for the SoilMoistureAccounting object; makes the
 * assumption that dataTimeStep is the same as the userTimeStep.
 * But the main method takes care of it, so no need to worry about it here.
 *
 * @param   currentDate      Current date
 * @param   zone             ET zone
 * @param   userTimeStep     User time step, in [hrs]
 * @param   imperviousness   Impreviousness of the basin, in [%]
 */
public SoilMoistureAccounting(ModelDate currentDate, ETZone zone,
int userTimeStep, double imperviousness) {

this.currentDate = currentDate;
this.zone = zone;
this.userTimeStep = userTimeStep;
this.imperviousness = imperviousness;

// instantiate the table function that is used to evaporate
// water that is held by the particles in soil's tension zone
double[] x = {0.0, 0.5, 0.6, 1.0};
double[] y = {0.0, 0.5, 1.0, 1.0};
this.tensionZoneTable = new Table(x, y);

// to initialize the flow parameters
// the flowGW1 and flowGW2 will be put through a linear
// reservoir outside the SMA, and excess will be put through
// Clark's method
this.excess = 0.0;
// these are in [mm/hr]
this.flowGW1[0] = 0.0;
this.flowGW2[0] = 0.0;
// these are in [mm]
this.flowGW1Vol[0] = 0.0;
this.flowGW2Vol[0] = 0.0;
}


// set methods
//{{{
```

```
/**
 *  Sets the imperviousness attribute of the SoilMoistureAccounting object
 *
 * @param  imperviousness  The new imperviousness value, in [%]
 */
public void setImperviousness(double imperviousness) {
this.imperviousness = imperviousness;
}


/**
 *  Sets the maxSurfStoreReductionCoeff attribute of the SoilMoistureAccounting object
 *
 * @param  maxSurfStoreReductionCoeff  The new maxSurfStoreReductionCoeff value, in [-]
 */
public void setMaxSurfStoreReductionCoeff(double maxSurfStoreReductionCoeff) {
this.maxSurfStoreReductionCoeff = maxSurfStoreReductionCoeff;
}


/**
 *  Sets the maxSoilInfilReductionCoeff attribute of the SoilMoistureAccounting object
 *
 * @param  maxSoilInfilReductionCoeff  The new maxSoilInfilReductionCoeff value
 */
public void setMaxSoilInfilReductionCoeff(double maxSoilInfilReductionCoeff) {
this.maxSoilInfilReductionCoeff = maxSoilInfilReductionCoeff;
}


/**
 *  Sets the maxStores attribute of the SoilMoistureAccounting object;
 * this is where the capacities of the storage buckets are set.
 *
 * @param  maxCanStore   Maximum Canopy Storage, in [mm]
 * @param  maxSurfStore  Maximum Surface Storage, in [mm]
 * @param  maxSoilStore  Maximum Soil Storage, in [mm]
 * @param  maxTensStore  Maximum Tension Storage, in [mm]
 * @param  maxGW1Store   Maximum GW1 Storage, in [mm]
 * @param  maxGW2Store   Maximum GW2 Storage, in [mm]
 */
public void setMaxStores(double maxCanStore, double maxSurfStore,
double maxSoilStore, double maxTensStore, double maxGW1Store,
double maxGW2Store) {

this.maxCanStore = maxCanStore;
this.maxSurfStore = maxSurfStore;

this.maxSurfStore = this.maxSurfStore *
this.maxSurfStoreReductionCoeff;

this.maxSoilStore = maxSoilStore;
this.maxTensStore = maxTensStore;
this.maxGW1Store = maxGW1Store;
this.maxGW2Store = maxGW2Store;
}


/**
 *  Sets the maxRates attribute of the SoilMoistureAccounting object;
 * this is where the max rates of flow between the buckets are set.
 *
 * @param  maxSoilInfil  Maximum Soil Infiltration, in [mm/hr]
 * @param  maxSoilPerc   Maximum Soil Percolation, in [mm/hr]
```

```
 * @param  maxGW1Perc    Maximum GW1 Percolation, in [mm/hr]
 * @param  maxGW2Perc    Maximum GW2 Percolation, in [mm/hr]
 */
public void setMaxRates(double maxSoilInfil, double maxSoilPerc,
double maxGW1Perc, double maxGW2Perc) {

this.maxSoilInfil = maxSoilInfil;

this.maxSoilInfil = this.maxSoilInfil *
this.maxSoilInfilReductionCoeff;

this.maxSoilPerc = maxSoilPerc;
this.maxGW1Perc = maxGW1Perc;
this.maxGW2Perc = maxGW2Perc;
}


/**
 *  Sets the initStores attribute of the SoilMoistureAccounting object;
 * this is where initial condition of the model are set.
 *
 * @param  initCanStore    Initial Canopy Storage, in [mm]
 * @param  initSurfStore   Initial Surface Storage, in [mm]
 * @param  initSoilStore   Initial Soil Storage, in [mm]
 * @param  initGW1Store    Initial GW1 Storage, in [mm]
 * @param  initGW2Store    Initial GW2 Storage, in [mm]
 */
public void setInitStores(double initCanStore, double initSurfStore,
double initSoilStore, double initGW1Store, double initGW2Store) {

this.canStore[0] = (initCanStore / 100.0) * this.maxCanStore;
this.surfStore[0] = (initSurfStore / 100.0) * this.maxSurfStore;
this.soilStore[0] = (initSoilStore / 100.0) * this.maxSoilStore;
this.GW1Store[0] = (initGW1Store / 100.0) * this.maxGW1Store;
this.GW2Store[0] = (initGW2Store / 100.0) * this.maxGW2Store;
}


/**
 *  Sets the routGWStorage attribute of the SoilMoistureAccounting object;
 * this is where the storage coefficients of two groundwater layers are
 * set.  These parameters are ones that route the flow out of
 * SoilMoistureAccounting object.  To say it another way, the method sets
 * the routing coefficients of GW layers [hrs] these are the routing
 * coefficients that convert excess GW into flow, which are then used
 * as input into baseflow layers
 *
 * @param  routGW1Storage  Storage Coefficient of GW1 layer, in [hrs]
 * @param  routGW2Storage  Storage Coefficient of GW2 layer, in [hrs]
 */
public void setRoutGWStorage(double routGW1Storage,
double routGW2Storage) {
this.routGW1Storage = routGW1Storage;
this.routGW2Storage = routGW2Storage;
}


/**
 *  This method computes the upper and lower bound of the time step.
 * It starts at zeroth time step.
 */
public void computeTimeStep() {

// upper bound is 12 hrs or minimum of one half of entire
```

```
// simulation time period
this.upperBound = 12.0;

// to compute the minimum or lower time bound
// for equations see Bennett (1998) p.16

// note that we are not considering surface routing storage
// coefficient to be one of the limiting factors, because
// Clark's method's time of concentration will impose a limit
// on the time step

double[] a = new double[7];
a[0] = 0.03125 * 9999.99;
// surface routing storage
a[1] = 0.03125 * this.routGW1Storage;
a[2] = 0.03125 * this.routGW2Storage;
a[3] = 0.5 * (this.maxCanStore + this.maxSurfStore
 + this.maxSoilStore) / (this.maxSoilInfil + this.maxSoilPerc
 + this.zone.getMinPotEvapTrans());
a[4] = 0.5 * this.maxSoilStore / this.maxSoilPerc;
a[5] = 0.5 * this.maxGW1Store / this.maxGW1Perc;
a[6] = 0.5 * this.maxGW2Store / this.maxGW2Perc;

// to find the minimum of a
double min = a[0];
for (int i = 1; i < a.length; i++) {
if (a[i] < min) {
min = a[i];
}
}

this.lowerBound = 2.0 * Math.min(min, this.userTimeStep);

// set the calculated time step to be equal to the lower bound
this.calcTimeStep = this.lowerBound;

// since in our study we'll have user time step that is equal
// to the data time step, the adjTimeStep will always work out
// to be the same as userTimeStep (or in our case, 6hrs)

// to calculate the adjusted time step
this.adjTimeStep = this.userTimeStep /
((int) Math.floor(this.userTimeStep /
this.calcTimeStep) + 1);

if (this.adjTimeStep != this.userTimeStep) {
System.out.println("adjTimeStep and userTimeStep are not the same");
System.out.println("Simulation Terminated!");
System.exit(0);
}
}


//}}}

/**
 *  This method computes the rates, and updates the states.  It starts
 * at the first time step.
 *
 * @param  ppt         Precipitation input [mm]
 * @param  currentDate  Current date
 */
public void update(double ppt, ModelDate currentDate) {
//{{{
```

```
// updates the current time, as this is needed for calculation
// of potEvapTrans
this.currentDate = currentDate;

// calculates the ppt that becomes excess immediately and thus
// contributes to direct runoff, as well as the ppt that
// becomes available and subject to losses
this.excessFromImperv = ppt * this.imperviousness / 100.0;
this.pptAvailToLoss = ppt - this.excessFromImperv;

/*
 *  // these are the calculations of the minimum time steps based
 *  // on the state of the storage volumes
 *  double[] timeStep = new double[8];
 *  // initialize this array to a large number
 *  // this will be used to pick minimum numbers
 *  for (int i = 0; i < timeStep.length; i++){
 *  timeStep[i] = 9999.99;
 *  }
 */
// calculates potential evapotranspiration, in [mm/hr]
this.potEvapTrans = this.zone.getPotEvapTrans(this.currentDate);

// calculates the potential soil infiltration, in [mm/hr]
this.potSoilInfil = this.maxSoilInfil -
(this.soilStore[0] / this.maxSoilStore) *
this.maxSoilInfil;

// calculates the potential soil percolation, in [mm/hr]
this.potSoilPerc = this.maxSoilPerc *
(this.soilStore[0] / this.maxSoilStore) *
(1.0 - (this.GW1Store[0] / this.maxGW1Store));

// calculates the potential GW1 percolation, in [mm/hr]
this.potGW1Perc = this.maxGW1Perc *
(this.GW1Store[0] / this.maxGW1Store) *
(1.0 - (this.GW2Store[0] / this.maxGW2Store));

// calculates the potential GW2 percolation, in [mm/hr]
this.potGW2Perc = this.maxGW2Perc *
(this.GW2Store[0] / this.maxGW2Store);

// calculation of ppt intensity during period specified by the
// precipitation data, in [mm/hr]
this.precipTimeStep = this.pptAvailToLoss / this.userTimeStep;

/*
 *  // MINIMUM TIME STEP CALCULATIONS START HERE
 *  // based on canopy interception
 *  if ((this.canStore[0] >= 0.0001) & this.potEvapTrans > 0.0){
 *  timeStep[0] = 0.25 * this.canStore[0] /
 *  this.potEvapTrans;
 *  }
 *  // based on surface interception storage
 *  if (this.surfStore[0] >= 0.0001 & (this.potSoilInfil >0.0 |
 *  this.potEvapTrans > 0.0)){
 *  timeStep[1] = 0.25 * (this.surfStore[0]) /
 *  (this.potSoilInfil + this.potEvapTrans);
 *  }
 *  // based on soil profile storage
 *  if (this.soilStore[0] >= 0.0001 & (this.potSoilPerc >0.0 |
 *  this.potEvapTrans > 0.0)){
 *  timeStep[2] = 0.25 * (this.soilStore[0]) /
```

```
 *  (this.potSoilPerc + this.potEvapTrans);
 *  }
 *  // based on GW1 storage
 *  if (this.GW1Store[0] >= 0.0001 & this.potGW1Perc > 0){
 *  timeStep[3] = 0.25 * this.GW1Store[0] / this.potGW1Perc;
 *  }
 *  if (this.GW1Store[0] / this.routGW1Storage > 0.0001){
 *  timeStep[4] = 0.0625 * this.routGW1Storage;
 *  }
 *  / based on GW2 storage
 *  if (this.GW2Store[0] >= 0.0001 & this.potGW2Perc > 0){
 *  timeStep[5] = 0.25 * (this.GW2Store[0] / this.potGW2Perc);
 *  }
 *  if (this.GW2Store[0] / this.routGW2Storage > 0.0001){
 *  timeStep[6] = 0.0625 * this.routGW2Storage;
 *  }
 *  // based on ppt intensity
 *  timeStep[7] = 0.25 * ( (this.maxCanStore +
 *  this.maxSurfStore + this.maxSoilStore) / this.precipTimeStep );
 *  // now to calculate the minimum of timeStep[]
 *  double min2 = timeStep[0];
 *  for (int i = 1; i < timeStep.length; i++){
 *  if (timeStep[i] < min2){
 *  min2 = timeStep[i];
 *  }
 *  }
 *  this.calcTimeStep = min2;
 *  // check to see if the calculated time step falls within the bounds
 *  if (this.calcTimeStep <= this.lowerBound){
 *  this.calcTimeStep = this.lowerBound;
 *  }
 *  if (this.calcTimeStep >= this.upperBound){
 *  this.calcTimeStep = this.upperBound;
 *  }
 *  // to calculate the adjusted time step
 *  this.adjTimeStep = this.userTimeStep /
 *  ( (int) Math.floor( this.userTimeStep /
 *  this.calcTimeStep ) + 1);
 */
// to calculate ppt to a volume falling during adjTimeStep
// this is just the adjTimeStep times the precipTimeStep
// or the time multiplied by ppt intensity
this.precipTimeStepVolume = this.adjTimeStep *
this.precipTimeStep;

// this is the volume of the potential evapotranspiration, in [mm]
this.potEvapTransVol = this.potEvapTrans * this.adjTimeStep;

// NOW TO UPDATE THE STATES
// must split this based on whether ppt is occuring, or not
// if ppt is not occuring, ET will occur

// if ppt is occuring
if (ppt > 0.0) {

/*
 *  System.out.println("Precipitation is occuring");
 *  System.out.println();
 */
// NOW TO FILL THE CANOPY STORAGE
if (this.canStore[0] <= this.maxCanStore) {
this.dCanStore = this.precipTimeStepVolume;
this.canStore[1] = this.canStore[0] +
this.dCanStore;
```

```
this.precipBeyondCanopy = 0.0;

// this is the calculation that computes what
// goes beyond the canopy (in case it is filled)
if (this.canStore[1] > this.maxCanStore) {
this.canStore[1] = this.maxCanStore;
this.dCanStore = this.canStore[1] -
this.canStore[0];
this.precipBeyondCanopy =
this.precipTimeStepVolume -
this.dCanStore;
}
} else {
this.canStore[1] = this.canStore[0];
this.precipBeyondCanopy =
this.precipTimeStepVolume;
}


/*
 *  System.out.println("ppt: " + this.precipTimeStepVolume);
 *  System.out.println("adjTimeStep: " + this.adjTimeStep);
 *  System.out.println();
 *  System.out.println("maxCanStore: " + this.maxCanStore);
 *  System.out.println("canStore[0]: " + this.canStore[0]);
 *  System.out.println("canStore[1]: " + this.canStore[1]);
 *  System.out.println("precipBeyondCanopy: " + this.precipBeyondCanopy);
 *  System.out.println();
 */
// NOW TO FILL THE SURFACE STORAGE
// amount available for infiltration is computed
this.availWater = this.precipBeyondCanopy +
this.surfStore[0];
// actSoilInfil is in [mm]
this.actSoilInfil = Math.min(
this.availWater, this.potSoilInfil *
this.adjTimeStep);

// if all availWater infiltrates, then surfStore
// doesn't get filled, and no excess runoff occurs
if (this.availWater == this.actSoilInfil) {

// if the surfStore is above zero, drain it
// otherwise, leave it
if (surfStore[0] > 0.0) {
this.surfStore[1] = this.surfStore[0] -
this.actSoilInfil;
// reset to zero in case it goes below
if (this.surfStore[1] < 0.0) {
this.surfStore[1] = 0.0;
}
} else {
this.surfStore[1] = this.surfStore[0];
}
this.surfExcess = 0.0;
}

// availWater can never be less than actSoilInfil,
// but only more
if (this.availWater > this.actSoilInfil) {
this.waterAvailFillSurfStore =
this.availWater -
this.actSoilInfil;
// this is the case where all
// waterAvailFillSurfStore actually fills the
```

```
// surface storage, and nothing becomess excess
this.dSurfStore = this.waterAvailFillSurfStore;
this.surfStore[1] = this.dSurfStore;
this.surfExcess = 0.0;

// in case surfStore gets overfilled, it gets
// reset to maxSurfStore, and computes excess
if (this.surfStore[1] > this.maxSurfStore) {
// changed on 17 Jul 2006, and now
// matches HEC-HMS
this.dSurfStore = this.surfStore[1] -
this.maxSurfStore;

this.surfStore[1] = this.maxSurfStore;

// this was changed on 31 Mar 2006
// originally I had surfExcess =
// availWater - dSurfStore
// this lead to overstimation of excess
this.surfExcess = this.dSurfStore;

}
}
// now to combine the surface excess with excess
// from imperviousness, in [mm]
this.excess = this.surfExcess + this.excessFromImperv;

/*
 *  System.out.println("availWater: " + this.availWater);
 *  System.out.println("potSoilInfil: " + this.potSoilInfil * this.adjTimeStep);
 *  System.out.println("actSoilInfil: " + this.actSoilInfil);
 *  System.out.println("maxSurfStore: " + this.maxSurfStore);
 *  System.out.println("surfStore[0]: " + this.surfStore[0]);
 *  System.out.println("surfStore[1]: " + this.surfStore[1]);
 *  System.out.println("excess: " + this.excess);
 *  System.out.println();
 */
// water available for soil percolation is only the water
// that is above the tension zone
if (this.soilStore[0] >= this.maxTensStore) {
this.waterAvailSoilPerc = this.soilStore[0] +
this.actSoilInfil - this.maxTensStore;
this.actSoilPerc = Math.min(
this.waterAvailSoilPerc,
this.potSoilPerc * this.adjTimeStep);
} else {
this.actSoilPerc = 0.0;
}

// NOW TO FILL THE SOIL STORAGE
// all that infiltrates fills the soil storage
// water available to fill the soil storage comes from
// actual soil infiltration only
if (this.soilStore[0] <= this.maxSoilStore) {
this.soilStore[1] = this.soilStore[0] +
this.actSoilInfil - this.actSoilPerc;

if (this.soilStore[1] > this.maxSoilStore) {
this.soilStore[1] = this.maxSoilStore;
}
} else {
this.soilStore[1] = this.soilStore[0];
}
/*
```

```
 *   System.out.println("actSoilInfil: " + this.actSoilInfil);
 *   System.out.println("maxTensStore: " + this.maxTensStore);
 *   System.out.println("maxSoilStore: " + this.maxSoilStore);
 *   System.out.println("soilStore[0]: " + this.soilStore[0]);
 *   System.out.println("soilStore[1]: " + this.soilStore[1]);
 *   System.out.println("potSoilPerc: " + this.potSoilPerc * this.adjTimeStep);
 *   System.out.println("actSoilPerc: " + this.actSoilPerc);
 *   System.out.println();
 */
// compute the flowGW1 [mm/hr]
this.flowGW1[1] = (this.actSoilPerc + GW1Store[0] -
(this.potGW1Perc * this.adjTimeStep) -
(flowGW1[0] * this.adjTimeStep / 2)) /
(this.routGW1Storage + 0.5 * this.adjTimeStep);

// convert to volume
this.flowGW1Vol[1] = (this.flowGW1[0] +
this.flowGW1[1]) * (this.adjTimeStep / 2.0);

// water available for GW1 percolation is actSoilPerc
// plus GW1 storage, minus flowGW1
this.waterAvailGW1Perc = this.GW1Store[0] +
this.actSoilPerc - this.flowGW1Vol[1];

this.actGW1Perc = Math.min(this.waterAvailGW1Perc,
this.potGW1Perc * this.adjTimeStep);

// NOW TO FILL THE GW1 LAYER STORAGE
// the only input is the actSoilPerc
if (this.GW1Store[0] <= this.maxGW1Store) {
this.GW1Store[1] = this.GW1Store[0] +
this.actSoilPerc - this.flowGW1Vol[1] -
this.actGW1Perc;

if (this.GW1Store[1] > this.maxGW1Store) {
this.GW1Store[1] = this.maxGW1Store;
}
} else {
this.GW1Store[1] = this.GW1Store[0];
}

/*
 *   System.out.println("actSoilPerc: " + this.actSoilPerc);
 *   System.out.println("maxGW1Store: " + this.maxGW1Store);
 *   System.out.println("GW1Store[0]: " + this.GW1Store[0]);
 *   System.out.println("GW1Store[1]: " + this.GW1Store[1]);
 *   System.out.println("flowGW1Vol[1] : " + this.flowGW1Vol[1]);
 *   System.out.println("waterAvailGW1Perc: " + this.waterAvailGW1Perc);
 *   System.out.println("potGW1Perc: " + this.potGW1Perc * this.adjTimeStep);
 *   System.out.println("actGW1Perc: " + this.actGW1Perc);
 *   System.out.println();
 */
// compute the flowGW2 [mm/hr]
this.flowGW2[1] = (this.actGW1Perc + GW2Store[0] -
(this.potGW2Perc * this.adjTimeStep) -
(flowGW2[0] * this.adjTimeStep / 2)) /
(this.routGW2Storage + 0.5 * this.adjTimeStep);
// convert to volume
this.flowGW2Vol[1] = (this.flowGW2[0] +
this.flowGW2[1]) * (this.adjTimeStep / 2.0);
// water available for GW2 percolation is actGW1Perc
// plus GW2 storage, minus flowGW2
this.waterAvailGW2Perc = this.GW2Store[0] +
this.actGW1Perc - this.flowGW2Vol[1];
```

```
this.actGW2Perc = Math.min(this.waterAvailGW2Perc,
this.potGW2Perc * this.adjTimeStep);

// NOW TO FILL THE GW2 LAYER STORAGE
// the only input is the actGW1Perc
if (this.GW2Store[0] <= this.maxGW2Store) {
this.GW2Store[1] = this.GW2Store[0] +
this.actGW1Perc - this.flowGW2Vol[1] -
this.actGW2Perc;

if (this.GW2Store[1] > this.maxGW2Store) {
this.GW2Store[1] = this.maxGW2Store;
}
} else {
this.GW2Store[1] = this.GW2Store[0];
}
/*
 *  System.out.println("actGW1Perc: " + this.actGW1Perc);
 *  System.out.println("maxGW2Store: " + this.maxGW2Store);
 *  System.out.println("GW2Store[0]: " + this.GW2Store[0]);
 *  System.out.println("GW2Store[1]: " + this.GW2Store[1]);
 *  System.out.println("flowGW2Vol[1] : " + this.flowGW2Vol[1]);
 *  System.out.println("waterAvailGW2Perc: " + this.waterAvailGW2Perc);
 *  System.out.println("potGW2Perc: " + this.potGW2Perc * this.adjTimeStep);
 *  System.out.println("actGW2Perc: " + this.actGW2Perc);
 *  System.out.println();
 */
}
// if ppt is not occuring, ET will
// infiltration and percolation rates can occur once the losses
// from ET are satisfied; must have this check
else {
// System.out.println("ET is occuring");
// this is what must evaporate this time step

/*
 *  System.out.println("adjTimeStep: " + this.adjTimeStep);
 *  System.out.println();
 *  System.out.println("potEvapTransVol : " +
 *  this.potEvapTransVol);
 */
// potEvapTrans is satisfied from canopy, surface,
// then soil (first from the upper zone, then from
// the tension zone (tension zone is where water is
// held by surface tension between the soil particles)

// try to satisfy ET from the canopy first
if (this.canStore[0] > 0.0) {
if (this.canStore[0] >= this.potEvapTransVol) {
this.canStore[1] = this.canStore[0] -
this.potEvapTransVol;
this.ETBeyondCanopy = 0.0;
} else {
this.canStore[1] = 0.0;
// calculate ET that goes beyond canopy
this.ETBeyondCanopy =
this.potEvapTransVol -
this.canStore[0];
}
} else {
// if canopy is initially empty
this.canStore[1] = 0.0;
this.ETBeyondCanopy = this.potEvapTransVol;
```

```
}

/*
 *  System.out.println("canStore[0]: " + this.canStore[0]);
 *  System.out.println("canStore[1]: " + this.canStore[1]);
 *  System.out.println("ETBeyondCanopy: " + this.ETBeyondCanopy);
 *  System.out.println();
 */
// try to satisfy ET from surface storage second
// input here is ETBeyondCanopy
if (this.surfStore[0] > 0.0) {
if (this.surfStore[0] >= this.ETBeyondCanopy) {
this.surfStore[1] = this.surfStore[0] -
this.ETBeyondCanopy;
this.ETBeyondSurface = 0.0;
} else {
this.surfStore[1] = 0.0;
this.ETBeyondSurface =
this.ETBeyondCanopy -
this.surfStore[0];
}
} else {
// if surface is initially empty
this.surfStore[1] = 0.0;
this.ETBeyondSurface = this.potEvapTransVol;
}
/*
 *  System.out.println("surfStore[0]: " + this.surfStore[0]);
 *  System.out.println("surfStore[1]: " + this.surfStore[1]);
 *  System.out.println("ETBeyondSurface: " + this.ETBeyondSurface);
 *  System.out.println();
 */
// to satisfy ET from soil storage
// input here is ETBeyondSurface

// calculate how much water is in the upper zone
if (this.soilStore[0] > 0.0) {
if (this.soilStore[0] > this.maxTensStore) {
this.upperZoneVol = this.soilStore[0] -
this.maxTensStore;
this.tensionZoneVol = this.soilStore[0] -
this.upperZoneVol;
} else {
// all water is part of tension zone
this.upperZoneVol = 0.0;
this.tensionZoneVol = this.soilStore[0];
}
}

// to satisfy ET from upper zone first
if (this.upperZoneVol > 0.0) {
if (this.upperZoneVol >= this.ETBeyondSurface) {
this.soilStore[1] = this.soilStore[0] -
this.ETBeyondSurface;
this.ETBeyondUpperZone = 0.0;
} else {
// drains the upper zone
this.soilStore[1] = this.soilStore[0] -
this.upperZoneVol;
this.ETBeyondUpperZone =
this.ETBeyondSurface -
this.upperZoneVol;
}
} else {
```

```
              // if upper zone is initialy empty
              this.soilStore[1] = this.soilStore[0];
              this.ETBeyondUpperZone = this.ETBeyondSurface;
              }
              /*
               *  System.out.println("upperZoneVol: " + this.upperZoneVol);
               *  System.out.println("tensionZoneVol: " + this.tensionZoneVol);
               *  System.out.println("maxTensStore: " + this.maxTensStore);
               *  System.out.println("soilStore[0]: " + this.soilStore[0]);
               *  System.out.println("soilStore[1]: " + this.soilStore[1]);
               *  System.out.println("ETBeyondUpperZone: " + this.ETBeyondUpperZone);
               *  System.out.println();
               */
              // to compute how much of the tensionZoneVol can be
              // evaporated; available is ETBeyondTensionZone
              this.actTensionZoneET = this.ETBeyondUpperZone *
              this.tensionZoneTable.lookup(
              this.soilStore[1] / this.maxTensStore);

              // to satisfy ET from the tension zone last
              // input here is actTensionZoneET
              if (this.tensionZoneVol > 0.0) {
              if (this.tensionZoneVol >= this.actTensionZoneET) {
              // subscript [1] is used on the right hand
              // side because soilStore[1] is previously
              // updated by the upper zone ET calculation
              this.soilStore[1] = this.soilStore[1] -
              this.actTensionZoneET;
              this.ETBeyondTensionZone = 0.0;
              } else {
              // drains the tension zone
              this.soilStore[1] = this.soilStore[1] -
              this.tensionZoneVol;

              this.ETBeyondTensionZone =
              this.actTensionZoneET -
              this.tensionZoneVol;
              }
              } else {
              // if the tension zone is initially empty
              this.soilStore[1] = this.soilStore[1];
              this.ETBeyondTensionZone = this.ETBeyondUpperZone;
              }

              /*
               *  System.out.println("actTensionZoneET: " + this.actTensionZoneET);
               *  System.out.println("tensionZoneVol: " + this.tensionZoneVol);
               *  System.out.println("maxTensStore: " + this.maxTensStore);
               *  System.out.println("soilStore[1]: " + this.soilStore[1]);
               *  System.out.println("ETBeyondTensionZone: " + this.ETBeyondTensionZone);
               *  System.out.println();
               */
              // now to include the infiltration and percolation calcs
              // this happens after ET gets satisfied

              // surface storage can only get drained here, and
              // surface excess can never happen during ET

              // amount available for infiltration is computed
              // the subscripts here are [1] because the states have
              // already been updated once during ET calculations
              this.availWater = this.surfStore[1];
              // actSoilInfil is in [mm]
              this.actSoilInfil = Math.min(
```

```
this.availWater, this.potSoilInfil *
this.adjTimeStep);
// to update (again) the surface profile
if (this.actSoilInfil > 0.0) {
this.surfStore[1] = this.surfStore[1] -
this.actSoilInfil;
}

// during an evapotranspiration period, excess can not
// be occuring
this.surfExcess = 0.0;

// now to combine the surface excess with excess
// from imperviousness, in [mm]
this.excess = this.surfExcess + this.excessFromImperv;

/*
 *  System.out.println("Infiltration and Percolations are computed");
 *  System.out.println();
 *  System.out.println("availWater: " + this.availWater);
 *  System.out.println("potSoilInfil: " + this.potSoilInfil * this.adjTimeStep);
 *  System.out.println("actSoilInfil: " + this.actSoilInfil);
 *  System.out.println("surfStore[1]: " + this.surfStore[1]);
 *  System.out.println();
 */
// water available for soil percolation is only the water
// that is above the tension zone
if (this.soilStore[1] >= this.maxTensStore) {
this.waterAvailSoilPerc = this.soilStore[1] +
this.actSoilInfil - this.maxTensStore;
this.actSoilPerc = Math.min(
this.waterAvailSoilPerc,
this.potSoilPerc * this.adjTimeStep);
} else {
this.actSoilPerc = 0.0;
}

// to update (again) the soil storage
if (this.soilStore[1] <= this.maxSoilStore) {
this.soilStore[1] = this.soilStore[1] +
this.actSoilInfil - this.actSoilPerc;

if (this.soilStore[1] > this.maxSoilStore) {
this.soilStore[1] = this.maxSoilStore;
}
} else {
this.soilStore[1] = this.soilStore[1];
}
/*
 *  System.out.println("potSoilPerc: " + this.potSoilPerc * this.adjTimeStep);
 *  System.out.println("actSoilPerc: " + this.actSoilPerc);
 *  System.out.println("actSoilInfil: " + this.actSoilInfil);
 *  System.out.println("soilStore[1]: " + this.soilStore[1]);
 *  System.out.println();
 */
// compute the flowGW1 [mm/hr]
this.flowGW1[1] = (this.actSoilPerc + GW1Store[0] -
(this.potGW1Perc * this.adjTimeStep) -
(flowGW1[0] * this.adjTimeStep / 2)) /
(this.routGW1Storage + 0.5 * this.adjTimeStep);

// convert to volume
this.flowGW1Vol[1] = (this.flowGW1[0] +
this.flowGW1[1]) * (this.adjTimeStep / 2.0);
```

```
// water available for GW1 percolation is actSoilPerc
// plus GW1 storage, minus flowGW1
this.waterAvailGW1Perc = this.GW1Store[0] +
this.actSoilPerc - this.flowGW1Vol[1];

this.actGW1Perc = Math.min(this.waterAvailGW1Perc,
this.potGW1Perc * this.adjTimeStep);

// NOW TO FILL THE GW1 LAYER STORAGE
// the only input is the actSoilPerc
if (this.GW1Store[0] <= this.maxGW1Store) {
this.GW1Store[1] = this.GW1Store[0] +
this.actSoilPerc - this.flowGW1Vol[1] -
this.actGW1Perc;

if (this.GW1Store[1] > this.maxGW1Store) {
this.GW1Store[1] = this.maxGW1Store;
}
} else {
this.GW1Store[1] = this.GW1Store[0];
}
/*
 *  System.out.println("actSoilPerc: " + this.actSoilPerc);
 *  System.out.println("maxGW1Store: " + this.maxGW1Store);
 *  System.out.println("GW1Store[0]: " + this.GW1Store[0]);
 *  System.out.println("GW1Store[1]: " + this.GW1Store[1]);
 *  System.out.println("flowGW1Vol[1] : " + this.flowGW1Vol[1]);
 *  System.out.println("waterAvailGW1Perc: " + this.waterAvailGW1Perc);
 *  System.out.println("potGW1Perc: " + this.potGW1Perc * this.adjTimeStep);
 *  System.out.println("actGW1Perc: " + this.actGW1Perc);
 *  System.out.println();
 */
// compute the flowGW2 [mm/hr]
this.flowGW2[1] = (this.actGW1Perc + GW2Store[0] -
(this.potGW2Perc * this.adjTimeStep) -
(flowGW2[0] * this.adjTimeStep / 2)) /
(this.routGW2Storage + 0.5 * this.adjTimeStep);

// convert to volume
this.flowGW2Vol[1] = (this.flowGW2[0] +
this.flowGW2[1]) * (this.adjTimeStep / 2.0);

// water available for GW2 percolation is actGW1Perc
// plus GW2 storage, minus flowGW2
this.waterAvailGW2Perc = this.GW2Store[0] +
this.actGW1Perc - this.flowGW2Vol[1];

this.actGW2Perc = Math.min(this.waterAvailGW2Perc,
this.potGW2Perc * this.adjTimeStep);

// NOW TO FILL THE GW2 LAYER STORAGE
// the only input is the actGW1Perc
if (this.GW2Store[0] <= this.maxGW2Store) {
this.GW2Store[1] = this.GW2Store[0] +
this.actGW1Perc - this.flowGW2Vol[1] -
this.actGW2Perc;

if (this.GW2Store[1] > this.maxGW2Store) {
this.GW2Store[1] = this.maxGW2Store;
}
} else {
this.GW2Store[1] = this.GW2Store[0];
}
```

```
/*
 *  System.out.println("actGW1Perc: " + this.actGW1Perc);
 *  System.out.println("maxGW2Store: " + this.maxGW2Store);
 *  System.out.println("GW2Store[0]: " + this.GW2Store[0]);
 *  System.out.println("GW2Store[1]: " + this.GW2Store[1]);
 *  System.out.println("flowGW2Vol[1] : " + this.flowGW2Vol[1]);
 *  System.out.println("waterAvailGW2Perc: " + this.waterAvailGW2Perc);
 *  System.out.println("potGW2Perc: " + this.potGW2Perc * this.adjTimeStep);
 *  System.out.println("actGW2Perc: " + this.actGW2Perc);
 *  System.out.println();
 */
}


// to reset the states
this.canStore[0] = this.canStore[1];
this.surfStore[0] = this.surfStore[1];
this.soilStore[0] = this.soilStore[1];
this.GW1Store[0] = this.GW1Store[1];
this.GW2Store[0] = this.GW2Store[1];

// to reset the GWflow variables
this.flowGW1[0] = this.flowGW1[1];
this.flowGW2[0] = this.flowGW2[1];
this.flowGW1Vol[0] = this.flowGW1Vol[1];
this.flowGW2Vol[0] = this.flowGW2Vol[1];
//}}}

}


// get methods
//{{{

/**
 *  Gets the imperviousness attribute of the SoilMoistureAccounting object
 *
 * @return    The imperviousness value, in [%]
 */
public double getImperviousness() {
return this.imperviousness;
}


/**
 *  Gets the maxSurfStoreReduction attribute of the SoilMoistureAccounting
 * object
 *
 * @return    The maxSurfStoreReduction value, in [-]
 */
public double getMaxSurfStoreReduction() {
return this.maxSurfStoreReductionCoeff;
}


/**
 *  Gets the maxSoilInfilReduction attribute of the SoilMoistureAccounting
 * object
 *
 * @return    The maxSoilInfilReduction value, in [-]
 */
public double getMaxSoilInfilReduction() {
return this.maxSoilInfilReductionCoeff;
}
```

```
/**
 *  Gets the excess attribute of the SoilMoistureAccounting object
 * during the time period of userTimeStep [hrs]
 *
 * @return    The surface excess value in units of [mm]
 */
public double getExcess() {
return this.excess;
}


/**
 *  Gets the potEvapTrans attribute of the SoilMoistureAccounting object
 * during the time period of userTimeStep [hrs]
 *
 * @return    The potEvapTrans value, in units of [mm/hr]
 */
public double getPotEvapTrans() {
return this.potEvapTrans;
}


/**
 *  Gets the potEvapTransVol attribute of the SoilMoistureAccounting object
 * during the time period of userTimeStep [hrs]
 *
 * @return     The potEvapTransVol value, in units of [mm]
 */
public double getPotEvapTransVol() {
return this.potEvapTransVol;
}


/**
 *  Gets the ETBeyondTensionZone attribute of the
 * SoilMoistureAccounting object during the period of userTimeStep [hrs]
 *
 * @return     The ETBeyondTensionZone value, in units of [mm]
 */
public double getETBeyondTensionZone() {
return this.ETBeyondTensionZone;
}


/**
 *  Gets the dateFromSMA attribute of the SoilMoistureAccounting object
 *
 * @return     The dateFromSMA value
 */
public ModelDate getDateFromSMA() {
return this.currentDate;
}


/**
 *  Gets the GW1Outflow attribute of the SoilMoistureAccounting object
 * This is what is to be used as input to linear reservoirs.
 * Subscript [0] is used because this method gets called after update()
 * which resets the current to previous.
 *
 * @return     The GW1Outflow value, in [mm/hr]
 */
public double getGW1Outflow() {
```

```
return this.flowGW1[0];
}


/**
 *  Gets the GW2Outflow attribute of the SoilMoistureAccounting object;
 * this is what is to be used as input to linear reservoirs.
 *
 * @return    The GW2Outflow value, in [mm/hr]
 */
public double getGW2Outflow() {
return this.flowGW2[0];
}


/**
 *  Gets the actSoilInfil attribute of the SoilMoistureAccounting object
 *
 * @return    The actSoilInfil value, in [mm/hr]
 */
public double getActSoilInfil() {
return this.actSoilInfil / this.userTimeStep;
}


/**
 *  Gets the potSoilInfil attribute of the SoilMoistureAccounting object
 *
 * @return    The potSoilInfil value
 */
public double getPotSoilInfil() {
return this.potSoilInfil / this.userTimeStep;
}


/**
 *  Gets the actSoilPerc attribute of the SoilMoistureAccounting object
 *
 * @return    The actSoilPerc value, in [mm/hr]
 */
public double getActSoilPerc() {
return this.actSoilPerc;
}


/**
 *  Gets the actGW2Perc attribute of the SoilMoistureAccounting object;
 * this is what gets sent to the SubBasin object.
 *
 * @return    The actGW2Perc value, in [mm]
 */
public double getActGW2Perc() {
return this.actGW2Perc;
}


/**
 *  Gets the maxSoilInfil attribute of the SoilMoistureAccounting object
 *
 * @return    The maxSoilInfil value, in [mm/hr]
 */
public double getMaxSoilInfil() {
return this.maxSoilInfil;
}
```

```
/**
 *  Gets the maxSurfStore attribute of the SoilMoistureAccounting object
 *
 * @return    The maxSurfStore value, in [mm]
 */
public double getMaxSurfStore() {
return this.maxSurfStore;
}


/**
 *  Gets the canStore attribute of the SoilMoistureAccounting object
 *
 * @return    The canStore value, in [mm]
 */
public double getCanStore() {
return this.canStore[0];
}


/**
 *  Gets the surfStore attribute of the SoilMoistureAccounting object
 *
 * @return    The surfStore value, in [mm]
 */
public double getSurfStore() {
return this.surfStore[0];
}


/**
 *  Gets the soilStore attribute of the SoilMoistureAccounting object
 *
 * @return    The soilStore value, in [mm]
 */
public double getSoilStore() {
return this.soilStore[0];
}


/**
 *  Gets the gW1Store attribute of the SoilMoistureAccounting object
 *
 * @return    The GW1Store value, in [mm]
 */
public double getGW1Store() {
return this.GW1Store[0];
}


/**
 *  Gets the gW2Store attribute of the SoilMoistureAccounting object
 *
 * @return    The GW2Store value, in [mm]
 */
public double getGW2Store() {
return this.GW2Store[0];
}
//}}}
}
```

D.14   LinearReservoir.java

```java
/**
 *  This class represents Linear Reservoir routing method
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class LinearReservoir {
// instance variables
double storageCoefficient;
int timeStep;
int numReservoirs;

// variables that constructor uses

// inflow is 1-d only
// storage and outflow are 2-d because each lr object will have
// current [1] and previous [0] value
double[] inflow;
double[][] storage;
double[][] outflow;


/**
 *  Constructor for the LinearReservoir object.
 *
 * @param  storageCoefficient  Storage Coefficient, in [hrs]
 * @param  timeStep            Time Step, in [hrs]
 * @param  numReservoirs       Number of linear reservoirs, [-]
 */
public LinearReservoir(double storageCoefficient,
int timeStep, int numReservoirs) {

this.storageCoefficient = storageCoefficient;
this.timeStep = timeStep;
this.numReservoirs = numReservoirs;

// to initialize the storage, inflow and outflow arrays
double[] inflow = new double[2];
double[][] storage = new double[this.numReservoirs][2];
double[][] outflow = new double[this.numReservoirs][2];

this.storage = storage;
this.inflow = inflow;
this.outflow = outflow;

}


/**
 *  Initializes the instance variables.
 *
 * @param  inflow  Inflow to the reservoir(s), in [cms].
 */
public void initialize(double inflow) {
this.inflow[0] = inflow;
for (int i = 0; i < this.numReservoirs; i++) {
this.outflow[i][0] = this.inflow[0];
this.storage[i][0] = this.storageCoefficient *
this.outflow[i][0];
}
}
```

```java
/**
 *  Updates the instance variables
 *
 * @param  inflow  Inflow to the reservoir(s), in [cms]
 */
public void update(double inflow) {
this.inflow[1] = inflow;

// for the first linear reservoir
this.outflow[0][1] = ((this.storage[0][0] + this.inflow[0] *
this.timeStep - (this.outflow[0][0] * this.timeStep / 2.0)) /
(this.storageCoefficient + (this.timeStep / 2.0)));
this.storage[0][1] = this.storageCoefficient * this.outflow[0][1];

// for all the other linear reservoirs in the cascade
for (int i = 1; i < this.numReservoirs; i++) {
this.outflow[i][1] = ((this.storage[i][0] + this.outflow[i - 1][0] *
this.timeStep - (this.outflow[i][0] * this.timeStep / 2.0)) /
(this.storageCoefficient + (this.timeStep / 2.0)));
this.storage[i][1] = this.storageCoefficient * this.outflow[i][1];
}

// to reset
this.inflow[0] = this.inflow[1];
for (int i = 0; i < this.numReservoirs; i++) {
this.outflow[i][0] = this.outflow[i][1];
this.storage[i][0] = this.storage[i][1];

}
}


/**
 *  Sets the storageCoeff attribute of the LinearReservoir object.
 * This method is used to change the storage coefficient of the
 * LinearReservoir object during the course of the simulation.
 *
 * @param  storageCoefficient  The new storageCoeff value, in [hrs]
 */
public void setStorageCoeff(double storageCoefficient) {
this.storageCoefficient = storageCoefficient;
}


// call this method before update()
/**
 *  Gets the initOutflow attribute of the LinearReservoir object.  This
 * method must be called before method update().
 *
 * @return    The initOutflow value, in [cms]
 */
public double getInitOutflow() {
return this.outflow[this.numReservoirs - 1][0];
}


/**
 *  Gets the outflow attribute of the LinearReservoir object.
 *
 * @return    The outflow value of the last linear reservoir, in [cms].
 */
public double getOutflow() {
return this.outflow[this.numReservoirs - 1][1];
```

```
    }

    }
```

D.15   ETZone.java

```java
/**
 *  This class represents an evapotranspiration zone
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class ETZone {

// instance variables
double[] monthlyET;
double panCoefficient;
ModelDate currentDate;

// this is a coefficient that varies as a function of vegetative land
double reductionCoeff;


/**
 *  Constructor for the ETZone object
 *
 * @param  currentDate      Current date
 */
public ETZone(ModelDate currentDate) {
this.currentDate = currentDate;
this.reductionCoeff = 1.0;
}


/**
 *  Sets the ETZoneParams attribute of the ETZone object
 *
 * @param  monthlyET       Monthly PET, in [mm]
 * @param  panCoefficient  Pan coefficient, [-]
 */
public void setETZoneParams(double[] monthlyET, double panCoefficient) {
this.monthlyET = monthlyET;
this.panCoefficient = panCoefficient;
}


/**
 *  Sets the PETReductionCoeff attribute of the ETZone object
 *
 * @param  reductionCoeff  The new PETReductionCoeff value
 */
public void setPETReductionCoeff(double reductionCoeff) {
this.reductionCoeff = reductionCoeff;
}


/**
 *  Gets the PETReductionCoeff attribute of the ETZone object
 *
 * @return    The PETReductionCoeff value
 */
public double getPETReductionCoeff() {
return this.reductionCoeff;
}


/**
 *  Gets the minPotEvapTrans attribute of the ETZone object
```

```
 *
 * @return    The minPotEvapTrans value, or a value of a non-zero
 * minimum value of EvapTrans in [mm/hr]
 */
public double getMinPotEvapTrans() {
double minimum = 9999.0;
int daysInMonth = this.currentDate.getDaysInMonth();

for (int i = 0; i < this.monthlyET.length; i++) {
if (this.monthlyET[i] != 0) {
if (this.monthlyET[i] < minimum) {
minimum = this.monthlyET[i];
}
}
}
// 0.1 [mm/month] * 0.7 * (month of Jan /31 days in Jan) * (1 day/ 24 hrs in a day)
return (minimum * this.panCoefficient * this.reductionCoeff /
(daysInMonth * 24));
}


/**
 *  Gets the potEvapTrans attribute of the ETZone object
 *
 * @param  currentDate  Current date
 * @return              The potEvapTrans value,  in [mm/hr]
 */
public double getPotEvapTrans(ModelDate currentDate) {
this.currentDate = currentDate;
int daysInMonth = this.currentDate.getDaysInMonth();
double evapThisMonth = this.monthlyET[this.currentDate.getMonth()];
return (evapThisMonth * this.panCoefficient *
this.reductionCoeff / (daysInMonth * 24.0));
}

}
```

D.16   Clark.java

```java
/**
 *  This class represents Clark's river routing method.  The method is one that
 * convolutes a unit hydrograph.  It takes surface excess, and computes direct
 * runoff.
 *
 * @author      Predrag Prodanovic
 * @created     February 12, 2006
 */
public class Clark {
// instance variables
private double basinArea;
private double timeOfConcentration;
private double storageCoefficient;
private int timeStep;

// these are the internal variables used by the object
private int numPointsUH;
private double[] UH;
private double[] Q;
private double[] Qshifted;

// this is the output variable
private double directFlow;


/**
 * Constructor for the Clark object.  The output of this method is in
 * units of [cms].
 *
 * @param  basinArea            Area of the basin, in [km<SUP>2</SUP>]
 * @param  timeOfConcentration  Time of Concentration, in [hrs]
 * @param  storageCoefficient   Storage Coefficient, in [hrs]
 * @param  timeStep             Time Step, in [hrs]
 */
public Clark(double basinArea, double timeOfConcentration,
double storageCoefficient, int timeStep) {
this.basinArea = basinArea;

this.timeOfConcentration = timeOfConcentration;
this.storageCoefficient = storageCoefficient;
this.timeStep = timeStep;

// assumes that a UH has 100 points each time step
this.numPointsUH = 100;
/*
 *  / have to declare and initialize the internal variables
 *  double[] UH = new double[this.numPointsUH];
 *  double[] Q = new double[this.numPointsUH];
 *  double[] Qshifted = new double[this.numPointsUH];
 *  this.UH = UH;
 *  this.Q = Q;
 *  this.Qshifted = Qshifted;
 *  / this just initialize it; in the program, this value never gets used
 *  this.directFlow = -999.9;
 */
}


/**
 *  Method initializes everthing that is needed by the Clark's object
 *
 * @param  excess  The rainfall exess, in [mm]
```

```
 */
public void initialize(double excess) {
this.UH = getUH();
this.Q = initQ(excess);
this.directFlow = Q[0];
this.Qshifted = shiftQ(Q);
}


/**
 *  Updates the instance variables of the Clark's object
 *
 * @param  excess  The rainfall exess, in [mm]
 */
public void update(double excess) {
// this means that a UH gets computed every time step
this.UH = getUH();
this.Q = getQ(excess, this.Qshifted);
this.directFlow = Q[0];
this.Qshifted = shiftQ(Q);
}


/**
 *  Sets the clarkParams attribute of the Clark object.  This method
 * is used to change the Clark's method parameters during the course
 * of the simulation
 *
 * @param  timeOfConcentration  The new clarkParams value of Time of
 * Concentration, in [hrs]
 * @param  storageCoefficient   The new clarkParams value of the
 * Storage Coefficient, in [hrs]
 */
public void setClarkParams(double timeOfConcentration,
double storageCoefficient) {
this.timeOfConcentration = timeOfConcentration;
this.storageCoefficient = storageCoefficient;
}


/**
 *  Gets the directFlow attribute of the Clark object
 *
 * @return     The directFlow value, in [cms]
 */
public double getDirectFlow() {
return this.directFlow;
}


/**
 *  Gets the UH attribute of the Clark object
 *
 * @return     The UH value, as a vector in [cms]
 */
private double[] getUH() {

// calculates the UH here in [cms]
double c = (2.0 * this.timeStep) /
(2.0 * this.storageCoefficient + this.timeStep);

// initial calculations
double timeToTimeOfConc = 0.0;
double cummulativeArea = 1.414 * Math.pow(timeToTimeOfConc, 1.5);
```

```
double[] cummulativeQ = new double[2];
cummulativeQ[0] = cummulativeArea * this.basinArea / this.timeStep;

double histogramQ = cummulativeQ[0];
double translationalQ = histogramQ * 1.0E6 / (1000.0 * 3600.0);

double[] IUH = new double[2];
//[cms]
IUH[0] = translationalQ;

double[] UH = new double[numPointsUH];
UH[0] = IUH[0];

double time = 0.0;

// all other calculations
for (int i = 1; i < this.numPointsUH; i++) {
time = time + this.timeStep;
if (time / (this.timeOfConcentration) <= 1.0) {
timeToTimeOfConc = time / this.timeOfConcentration;
} else {
timeToTimeOfConc = 0.0;
}
if (timeToTimeOfConc < 0.5) {
cummulativeArea = 1.414 *
Math.pow(timeToTimeOfConc, 1.5);
} else {
cummulativeArea = 1.0 - 1.414 *
Math.pow((1.0 - timeToTimeOfConc), 1.5);
}
cummulativeQ[1] = cummulativeArea * this.basinArea / this.timeStep;
if ((cummulativeQ[1] - cummulativeQ[0]) < 0) {
histogramQ = 0.0;
} else {
histogramQ = cummulativeQ[1] - cummulativeQ[0];
}
translationalQ = histogramQ * 1.0E6 / (1000.0 * 3600.0);
IUH[1] = c * translationalQ + (1.0 - c) * IUH[0];
UH[i] = (IUH[0] + IUH[1]) / 2.0;
// to reset
cummulativeQ[0] = cummulativeQ[1];
IUH[0] = IUH[1];
}
// end for
/*
 *  // this works; the area is really close to unity
 *  double sum = 0.0;
 *  // to calculate the area of the UH
 *  for (int i = 0; i < UH.length; i++){
 *  sum = sum + UH[i];
 *  }
 *  System.out.println("Area of UH [mm]: " + sum *
 *  (3600.0 * 1000.0) /
 *  (this.basinArea * 1000.0 * 1000.0) );
 */
return UH;
}


// this method returns the outflow
// method convolutes a UH
/**
 *  Method returns the initial outflow, after it convolutes the UH
 *  for the initial time step
```

```
 *
 * @param  excess  The rainfall exess, in [mm]
 * @return         Returns the initial flow in [cms]
 */
private double[] initQ(double excess) {
double[] Q = new double[this.numPointsUH];
for (int i = 0; i < this.numPointsUH; i++) {
Q[i] = excess * this.UH[i];
}
return Q;
}


/**
 *  A method that simply shifts the flow values.  This is needed for
 * convoluting the UH
 *
 * @param  Q  Flow values, in [cms]
 * @return    Shifted flow values, in [cms]
 */
private double[] shiftQ(double[] Q) {
double[] Qshifted = new double[this.numPointsUH];
for (int i = 0; i < this.numPointsUH - 1; i++) {
Qshifted[i] = Q[i + 1];
}
Qshifted[this.numPointsUH - 1] = 0.0;
return Qshifted;
}


/**
 *  Method returns the outflow, after it convolutes the UH
 *
 * @param  excess    The rainfall exess, in [mm]
 * @param  Qshifted  Shifted flow values, in [cms]
 * @return           Flow, in [cms]
 */
private double[] getQ(double excess, double[] Qshifted) {
double[] Q = new double[this.numPointsUH];
for (int i = 0; i < numPointsUH; i++) {
Q[i] = excess * this.UH[i] + Qshifted[i];
}
return Q;
}


/**
 *  Gets the timeOfConcentration attribute of the Clark object
 *
 * @return    The timeOfConcentration value, in [hrs]
 */
public double getTimeOfConcentration() {
return this.timeOfConcentration;
}
}
```

D.17 Smooth3.java

```
/**
 *  This is the class that represents third order exponential smoothing.
 * Equations used are those given in Sterman (2000), p. 433.  The Smooth3 object
 * represents an instance of an information delay (i.e., the flows are not
 * conserved).
 *
 * @author     Predrag Prodanovic
 * @created    February 18, 2006
 */
public class Smooth3 {

// instance variables
private double input, delayTime, output, timeStep;

// to declare state (S) and rate (R) variables
private double[] S1, S2, S3;
private double R1, R2, R3;


/**
 * Constructor for the Smooth3 object; it is an empty constructor.
 *
 * @param  timeStep  Time step of the model, same time units as input
 */
public Smooth3(double timeStep) {
this.S1 = new double[2];
this.S2 = new double[2];
this.S3 = new double[2];
this.timeStep = timeStep;
}


/**
 *  Initializes the parameters in the Smooth3 object.
 *
 * @param  input      Input value
 * @param  delayTime  Delay time, in [years]; must be much greater
 * than the time step of the model, probably 15-20 times greater for
 * smooth behaviour.
 */
public void initialize(double input, double delayTime) {
this.input = input;
this.delayTime = delayTime;

// initializes state and rate variables
this.S1[0] = this.input;
this.S2[0] = this.input;
this.S3[0] = this.input;

this.R1 = (this.input - this.S1[0]) / (this.delayTime / 3.0);
this.R2 = (this.S1[0] - this.S2[0]) / (this.delayTime / 3.0);
this.R3 = (this.S2[0] - this.S3[0]) / (this.delayTime / 3.0);

this.output = input;

}


/**
 *  Updates the parameters in the Smooth3 object.
 *
 * @param  input      Input value
```

```
 * @param  delayTime  Delay time, in [years]
 */
public void update(double input, double delayTime) {

this.input = input;
this.delayTime = delayTime;

// to update the state variables
this.S1[1] = this.S1[0] + (this.R1) * this.timeStep;
this.S2[1] = this.S2[0] + (this.R2) * this.timeStep;
this.S3[1] = this.S3[0] + (this.R3) * this.timeStep;

// to update the rate variables
this.R1 = (this.input - this.S1[1]) / (this.delayTime / 3.0);
this.R2 = (this.S1[1] - this.S2[1]) / (this.delayTime / 3.0);
this.R3 = (this.S2[1] - this.S3[1]) / (this.delayTime / 3.0);

this.output = this.S3[1];

// to reset the state variables
this.S1[0] = this.S1[1];
this.S2[0] = this.S2[1];
this.S3[0] = this.S3[1];

}


/**
 *  Gets the output attribute of the Smooth3 object
 *
 * @return     The output value
 */
public double getOutput() {
return this.output;
}

}
```

D.18   Drought.java

```java
/**
 *  This object is part of the SysModel object, and is used soley for the
 * calculations of drought indicators, based on the Ontario Low Water Responce
 * (2003) provintial guidelines and thresholds.
 *
 * @author      Predrag Prodanovic
 * @created     April 22, 2006
 */
public class Drought {
// variable declarations
//{{{
private ModelDate currentDate;

private int currentMonth;
private int numYears;

// these variables it gets from outside
private double monthlyFlow, monthlyPPT;

// these are the current values of one, three and eighteen month ppt
private double oneMonthPPT, threeMonthPPT, eighteenMonthPPT;

// this is the current value of the lowest average summer monthly flow
private double lowAvgSumMonthFlow;

// holds ppt for last three and eighteen months
private double[] pptLastThreeMonths = new double[3];
private double[] pptLastEighteenMonths = new double[18];

// historical averages and their declarations
// for ppt the historical averages are an array of size 12
private double[] histOneMonthPPTAvg = new double[12];
private double[] histOneMonthPPTAvgTotal = new double[12];

private double[] histThreeMonthPPTAvg = new double[12];
private double[] histThreeMonthPPTAvgTotal = new double[12];

private double[] histEighteenMonthPPTAvg = new double[12];
private double[] histEighteenMonthPPTAvgTotal = new double[12];

// for flow the historical average is simply one value
private double histLowAvgSumMonthFlow, histLowAvgSumMonthFlowTotal;

private int droughtLevel;
//}}}

/**
 * Constructor for the Drought object
 *
 * @param  currentDate  The starting date
 * @param  numYears     The number of years of record that historical
 * average ppt and flow were computed
 */
public Drought(ModelDate currentDate, int numYears) {
this.currentDate = currentDate;
this.numYears = numYears;
}


// set methods start here and initialize the historical averages
//{{{
/**
```

```
 *  Sets the histOneMonthPPT attribute of the Drought object
 *
 * @param  histOneMonthPPTAvgTotal  Hist one month ppt totals, in [mm]
 */
public void setHistOneMonthPPT(double[] histOneMonthPPTAvgTotal) {

this.histOneMonthPPTAvgTotal = histOneMonthPPTAvgTotal;

for (int i = 0; i < 12; i++) {
this.histOneMonthPPTAvg[i] =
this.histOneMonthPPTAvgTotal[i] / numYears;
}

}


/**
 *  Sets the histThreeMonthPPT attribute of the Drought object
 *
 * @param  histThreeMonthPPTAvgTotal  Hist three month ppt totals, in [mm]
 */
public void setHistThreeMonthPPT(double[] histThreeMonthPPTAvgTotal) {

this.histThreeMonthPPTAvgTotal = histThreeMonthPPTAvgTotal;

for (int i = 0; i < 12; i++) {
this.histThreeMonthPPTAvg[i] =
this.histThreeMonthPPTAvgTotal[i] / numYears;
}

}


/**
 *  Sets the histEighteenMonthPPT attribute of the Drought object
 *
 * @param  histEighteenMonthPPTAvgTotal  Hist eighteen month ppt totals, in [mm]
 */
public void setHistEighteenMonthPPT(double[] histEighteenMonthPPTAvgTotal) {

this.histEighteenMonthPPTAvgTotal = histEighteenMonthPPTAvgTotal;

for (int i = 0; i < 12; i++) {
this.histEighteenMonthPPTAvg[i] =
this.histEighteenMonthPPTAvgTotal[i] / numYears;
}

}


/**
 *  Sets the histLowAvgSumMonthFlow attribute of the Drought object
 *
 * @param  histLowAvgSumMonthFlowTotal  Hist lowest average summer
 * monthly flow total value, in [mm]
 */
public void setHistLowAvgSumMonthFlow(double histLowAvgSumMonthFlowTotal) {

this.histLowAvgSumMonthFlowTotal = histLowAvgSumMonthFlowTotal;
this.histLowAvgSumMonthFlow = this.histLowAvgSumMonthFlowTotal /
numYears;

// the current value of lowest average summer montly flow will
// be taken as the historic value for the first year
```

```
this.lowAvgSumMonthFlow = this.histLowAvgSumMonthFlow;
}


/**
 *  Sets the PPTLastEighteenMonths attribute of the Drought object
 *
 * @param  pptLastEighteenMonths  The new pPTLastEighteenMonths value, in [mm]
 */
public void setPPTLastEighteenMonths(double[] pptLastEighteenMonths) {
this.pptLastEighteenMonths = pptLastEighteenMonths;

// to initialize the ppt last three months
for (int i = 0; i < 18; i++) {
if (i >= 15) {
this.pptLastThreeMonths[i - 15] =
this.pptLastEighteenMonths[i];
}
}
}


//}}}

/**
 *  Updates all the parameters in the Drought object; must be called only
 * after all of the set methods have initialized the historical properties.
 *
 * @param  currentDate  The current date
 * @param  monthlyPPT   Monthly PPT, in [mm]
 * @param  monthlyFlow  Monthly flow, in [cms]
 */
public void update(ModelDate currentDate, double monthlyPPT,
double monthlyFlow) {
//{{{

// updates the instance variables
this.currentDate = currentDate;
this.monthlyPPT = monthlyPPT;
this.monthlyFlow = monthlyFlow;

// this is between 0 and 11 (i.e., Jan is 0 and Dec is 11)
this.currentMonth = this.currentDate.getMonth();

// updates the number of years in the historical record
if (this.currentMonth == 0) {
this.numYears = this.numYears + 1;
}

// to update the current ppt and flow values
// to update the array that hold the ppt for the last 3 months
for (int i = 0; i < 2; i++) {
this.pptLastThreeMonths[i] =
this.pptLastThreeMonths[i + 1];
}
this.pptLastThreeMonths[2] = this.monthlyPPT;

// to update the array that hold the ppt for the last 18 months
for (int i = 0; i < 17; i++) {
this.pptLastEighteenMonths[i] =
this.pptLastEighteenMonths[i + 1];
}
this.pptLastEighteenMonths[17] = this.monthlyPPT;
```

```
// to calculate three month and eighteen month ppt based on the
// current value of monthly ppt
this.oneMonthPPT = this.monthlyPPT;

double sumThreeMonths = 0.0;
for (int i = 0; i < 3; i++) {
sumThreeMonths = sumThreeMonths +
this.pptLastThreeMonths[i];
}
this.threeMonthPPT = sumThreeMonths / 3.0;

double sumEighteenMonths = 0.0;
for (int i = 0; i < 18; i++) {
sumEighteenMonths = sumEighteenMonths +
this.pptLastEighteenMonths[i];
}
this.eighteenMonthPPT = sumEighteenMonths / 18.0;

// to calculate lowest average summer monthly flow, and update
// the historical lowest average summer monthly flow

if (this.currentMonth == 5) {
this.lowAvgSumMonthFlow = this.monthlyFlow;
// updates the historic min lowest aveg summer monthly flow
this.histLowAvgSumMonthFlowTotal =
this.histLowAvgSumMonthFlowTotal +
this.lowAvgSumMonthFlow;
this.histLowAvgSumMonthFlow =
this.histLowAvgSumMonthFlowTotal / numYears;

}

if ((this.currentMonth) >= 6 && (this.currentMonth <= 7)) {
if (this.monthlyFlow < this.lowAvgSumMonthFlow) {
this.lowAvgSumMonthFlow = this.monthlyFlow;

// updates the hist min low avg summer monthly flow
this.histLowAvgSumMonthFlowTotal =
this.histLowAvgSumMonthFlowTotal +
this.lowAvgSumMonthFlow;
this.histLowAvgSumMonthFlow =
this.histLowAvgSumMonthFlowTotal / numYears;
}
}

// to update the historical average ppt
this.histOneMonthPPTAvgTotal[this.currentMonth] =
this.histOneMonthPPTAvgTotal[this.currentMonth] +
this.oneMonthPPT;
this.histOneMonthPPTAvg[this.currentMonth] =
this.histOneMonthPPTAvgTotal[this.currentMonth] /
this.numYears;

this.histThreeMonthPPTAvgTotal[this.currentMonth] =
this.histThreeMonthPPTAvgTotal[this.currentMonth] +
this.threeMonthPPT;
this.histThreeMonthPPTAvg[this.currentMonth] =
this.histThreeMonthPPTAvgTotal[this.currentMonth] /
this.numYears;

this.histEighteenMonthPPTAvgTotal[this.currentMonth] =
this.histEighteenMonthPPTAvgTotal[this.currentMonth] +
this.eighteenMonthPPT;
this.histEighteenMonthPPTAvg[this.currentMonth] =
```

```
this.histEighteenMonthPPTAvgTotal[this.currentMonth] /
this.numYears;

// to update the level of drought
switch (this.droughtLevel) {

case 0:

if (this.currentMonth > 1 && this.currentMonth < 5) {
// spring
if ((this.monthlyFlow < this.histLowAvgSumMonthFlow) ||
(this.threeMonthPPT < 0.8 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.8 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
this.droughtLevel = 1;
}
} else {
// other
if ((this.monthlyFlow < 0.7 *
this.histLowAvgSumMonthFlow) ||
(this.threeMonthPPT < 0.8 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.8 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
this.droughtLevel = 1;
}
}
break;
case 1:
if (this.currentMonth > 1 && this.currentMonth < 5) {
// spring
if ((this.monthlyFlow < 0.7 *
this.histLowAvgSumMonthFlow) ||
(this.oneMonthPPT < 0.6 *
this.histOneMonthPPTAvg[this.currentMonth]) ||
(this.threeMonthPPT < 0.6 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.6 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {

this.droughtLevel = 2;

} else if ((this.monthlyFlow <
this.histLowAvgSumMonthFlow) ||
(this.threeMonthPPT < 0.8 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.8 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
this.droughtLevel = 1;
} else {
this.droughtLevel = 0;
}
} else {
// other
if ((this.monthlyFlow < 0.5 *
this.histLowAvgSumMonthFlow) ||
(this.oneMonthPPT < 0.6 *
this.histOneMonthPPTAvg[this.currentMonth]) ||
(this.threeMonthPPT < 0.6 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.6 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
```

```
this.droughtLevel = 2;
} else if ((this.monthlyFlow <
this.histLowAvgSumMonthFlow) ||
(this.threeMonthPPT < 0.8 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.8 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
this.droughtLevel = 1;
} else {
this.droughtLevel = 0;
}
}
break;
case 2:

if (this.currentMonth > 1 && this.currentMonth < 5) {
// spring
if ((this.monthlyFlow < 0.5 *
this.histLowAvgSumMonthFlow) ||
(this.oneMonthPPT < 0.4 *
this.histOneMonthPPTAvg[this.currentMonth]) ||
(this.threeMonthPPT < 0.4 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.4 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {

this.droughtLevel = 3;
} else if ((this.monthlyFlow <
this.histLowAvgSumMonthFlow) ||
(this.threeMonthPPT < 0.8 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.8 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
this.droughtLevel = 1;
} else {
this.droughtLevel = 2;
}
} else {
// other

if ((this.monthlyFlow < 0.5 *
this.histLowAvgSumMonthFlow) ||
(this.oneMonthPPT < 0.4 *
this.histOneMonthPPTAvg[this.currentMonth]) ||
(this.threeMonthPPT < 0.4 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.4 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {

this.droughtLevel = 3;
} else if ((this.monthlyFlow <
this.histLowAvgSumMonthFlow) ||
(this.threeMonthPPT < 0.8 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.8 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {
this.droughtLevel = 1;
} else {
this.droughtLevel = 2;
}
}
break;
case 3:
```

```
if (this.currentMonth > 1 && this.currentMonth < 5) {
// spring
if ((this.monthlyFlow < 0.7 *
this.histLowAvgSumMonthFlow) ||
(this.oneMonthPPT < 0.6 *
this.histOneMonthPPTAvg[this.currentMonth]) ||
(this.threeMonthPPT < 0.6 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.6 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {

this.droughtLevel = 2;
}
} else {
// other
if ((this.monthlyFlow < 0.7 *
this.histLowAvgSumMonthFlow) ||
(this.oneMonthPPT < 0.6 *
this.histOneMonthPPTAvg[this.currentMonth]) ||
(this.threeMonthPPT < 0.6 *
this.histThreeMonthPPTAvg[this.currentMonth]) ||
(this.eighteenMonthPPT < 0.6 *
this.histEighteenMonthPPTAvg[this.currentMonth])) {

this.droughtLevel = 2;
}
}
break;
}

//}}}
}


// get methods that output the drought data to an outside object
//{{{
/**
 *  Gets the droughtLevel attribute of the Drought object
 *
 * @return    The droughtLevel value
 */
public int getDroughtLevel() {
return this.droughtLevel;
}


/**
 *  Gets the numYears attribute of the Drought object
 *
 * @return    The numYears value
 */
public double getNumYears() {
return this.numYears;
}


/**
 *  Gets the oneMonthPPT attribute of the Drought object
 *
 * @return    The oneMonthPPT value
 */
public double getOneMonthPPT() {
return this.oneMonthPPT;
}
```

```
/**
 *  Gets the threeMonthPPT attribute of the Drought object
 *
 * @return     The threeMonthPPT value
 */
public double getThreeMonthPPT() {
return this.threeMonthPPT;
}


/**
 *  Gets the eighteenMonthPPT attribute of the Drought object
 *
 * @return     The eighteenMonthPPT value
 */
public double getEighteenMonthPPT() {
return this.eighteenMonthPPT;
}


/**
 *  Gets the histOneMonthPPT attribute of the Drought object
 *
 * @return     The histOneMonthPPT value
 */
public double getHistOneMonthPPT() {
return this.histOneMonthPPTAvg[this.currentMonth];
}


/**
 *  Gets the histThreeMonthPPT attribute of the Drought object
 *
 * @return     The histThreeMonthPPT value
 */
public double getHistThreeMonthPPT() {
return this.histThreeMonthPPTAvg[this.currentMonth];
}


/**
 *  Gets the histEighteenMonthPPT attribute of the Drought object
 *
 * @return     The histEighteenMonthPPT value
 */
public double getHistEighteenMonthPPT() {
return this.histEighteenMonthPPTAvg[this.currentMonth];
}


/**
 *  Gets the oneMonthFlow attribute of the Drought object
 *
 * @return     The oneMonthFlow value
 */
public double getOneMonthFlow() {
return this.monthlyFlow;
}


/**
 *  Gets the lowAvgSumMonthFlow attribute of the Drought object
```

```
 *
 * @return    The lowAvgSumMonthFlow value
 */
public double getLowAvgSumMonthFlow() {
return this.lowAvgSumMonthFlow;
}


/**
 *  Gets the histLowAvgSumMonthFlow attribute of the Drought object
 *
 * @return    The histLowAvgSumMonthFlow value
 */
public double getHistLowAvgSumMonthFlow() {
return this.histLowAvgSumMonthFlow;
}
//}}}


}
```