THE UNIVERSITY OF WESTERN ONTARIO
DEPARTMENT OF CIVIL AND
ENVIRONMENTAL ENGINEERING

Water Resources Research Report

Inverse Drought Risk Modelling 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

Report No: 053
Date: November 2006

ISSN: (print) 1913-3200; (online) 1913-3219;
ISBN: (print) 978-0-7714-2636-0; (online) 978-0-7714-2637-7;

CFCAS Project: Assessment of Risk and
Vulnerability to Changing Climatic Conditions

Inverse Drought Risk Modelling 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

November 3, 2006

Abstract

This report aims to present an alternate approach to climate change impact modelling of water resources. The focus of the project is on the analysis of existing water resources management guidelines specifically targeting critical hydrologic events (extreme droughts in this case). The critical hydrologic events are converted to their corresponding meteorologic conditions via use of an appropriate hydrologic model (continuous based hydrologic model for drought analysis). The local climatic signal is generated by use of a non-parametric weather generator linked to outputs from a global circulation model for three climate scenarios, and their corresponding frequency curves generated. Then, a critical hydrologic event of interest is selected, its corresponding meteorological condition obtained, and its frequency of occurrence for each climate scenario determined. It is noted that all climate change scenarios showed less frequent occurrence of extreme droughts. However, potentially severe droughts are still possible (with a chance of 1 in 10 any given year, sometime less) in the basin; this coupled with the fact that drought damage assessments are non existent in the basin suggests that new or improved drought management guidelines should be investigated.

Based on the analysis presented, recommendations are made for future work to include: (i) drought impact studies (where impacts to agriculture, recreation, wetlands, reservoir operation, ground water withdrawal and streamflow quality are assessed); (ii) definition of local drought triggers (including guidelines on subwatershed scale, as well as monitoring how drought triggers change over time); (iii) water quality management (setting in place practises that enhance water quality over short and long term); (iv) education programs (to bring up to date knowledge in science to all who stand to be adversely impacted by drought).

Keywords: Inverse approach, assessment of climatic change, frequency analysis, weather generator, continuous hydrologic model, water resources management guidelines.

Acknowledgements

The work presented here was developed as part of the project *Assessment of Risk and Vulnerability to Changing Climatic Conditions*, for which *Canadian Foundation for Climate and Atmospheric Sciences* provided funding. Financial assistance from the *Natural Sciences and Engineering Research Council* is also acknowledged. The project is a collaborative effort between The University of Western Ontario, University of Waterloo, and the Upper Thames River Conservation Authority. Our sincerest thanks goes, in no particular order, to Juraj Čunderlík, Shawn Gettler, Mark Helsten, Matt Wood, Linda Mortsch, Donald Burn, and Karen Wey.

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 our 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 via 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.

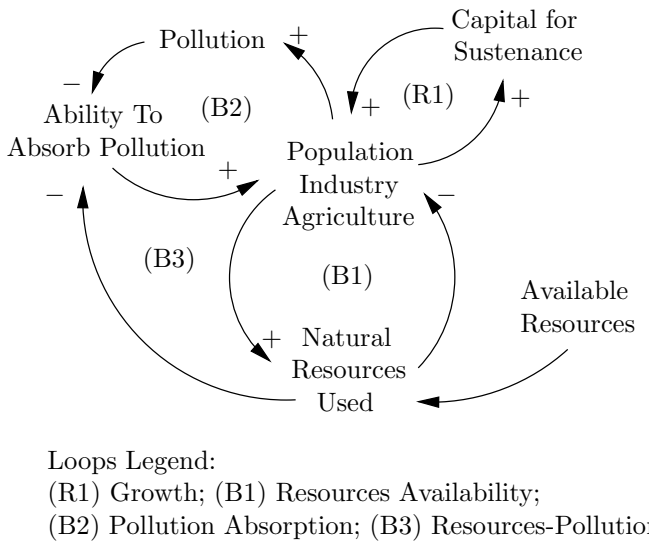

Figure 1: Causal loop of the problem of climate change

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 our 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 our 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 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 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).

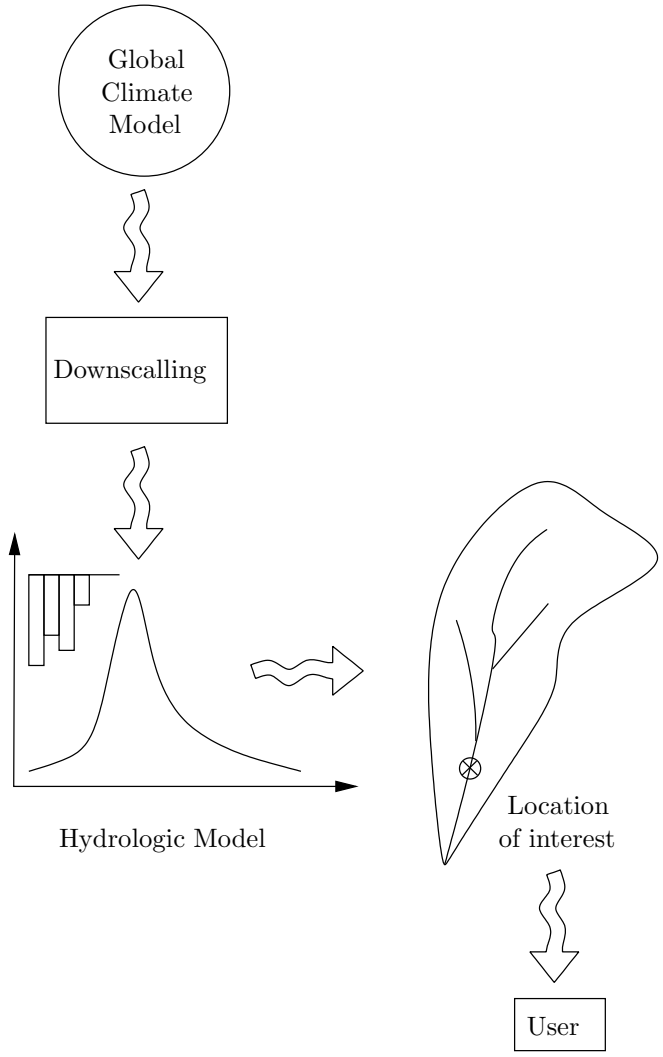

Figure 2: Traditional approach to climate change modelling

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 many 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 watershed scales. In the study by Coulibaly and Dibike (2004) for example, the authors 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. Consequently, the end-users (water management authorities, government policy makers and stakeholders) became sceptical of results of such analysis, and rarely form new guidelines as a result of them.

1.3 Outline of the report

The rest of this report focuses on the application of an inverse approach (alternative to the traditional top-down approach) to the problem of flooding. The application of the inverse approach to the problem of droughts is covered in a separate report. The report is organized as follows: Section 2 outlines the basics of the inverse approach, and outlines its necessary steps. The structure of the event based hydrologic model is outlined, as are the basics of the weather generator used to formulate the climate scenarios. The application of the methodology is illustrated with a case study watershed, described in Section 3. Results of the case study for three different subwatersheds in the basin are given in Section 4, while concluding remarks are given in Section 5.

2 Inverse Approach To Climate Change Modelling

2.1 Methodology

The inverse approach, originally developed by Cunderlik and Simonovic (2004b, 2006) takes an alternate (or bottom-up) route to climate change impact assessment, and thus avoids the many uncertainties of the traditional (downscaling) approach. The main strength of the approach is that it focuses on the end users of water resources systems, and involves them in the process. The inverse approach includes the following steps (modified from Cunderlik and Simonovic, 2004b, 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 me-

teorological conditions for each location of interest are formulated. These are simply precipitation-discharge curves, which capture specific management practises of the end-users (such as flood/water control, environmental and watershed 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 that the model may generate 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 re-

lationship 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 tool 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 option of having an integrated assessment model (i.e., an interface that combines a hydrologic model to a system dynamics model), thus providing the user an option to integrate socio-economic and physical processes. Figure 4 on the other hand, depicts typical plots used in obtaining results from the inverse approach. For example, a critical hydrologic exposure of interest is selected (step 1), and then converted to its corresponding meteorological condition (step 2) in the top most plot of Figure 4. The return periods (or frequencies of occurrence) of various meteorologic exposures (step 3) are shown next in the middle plot of Figure 4 for various scenarios. From these, precipitation return periods can be promptly obtained

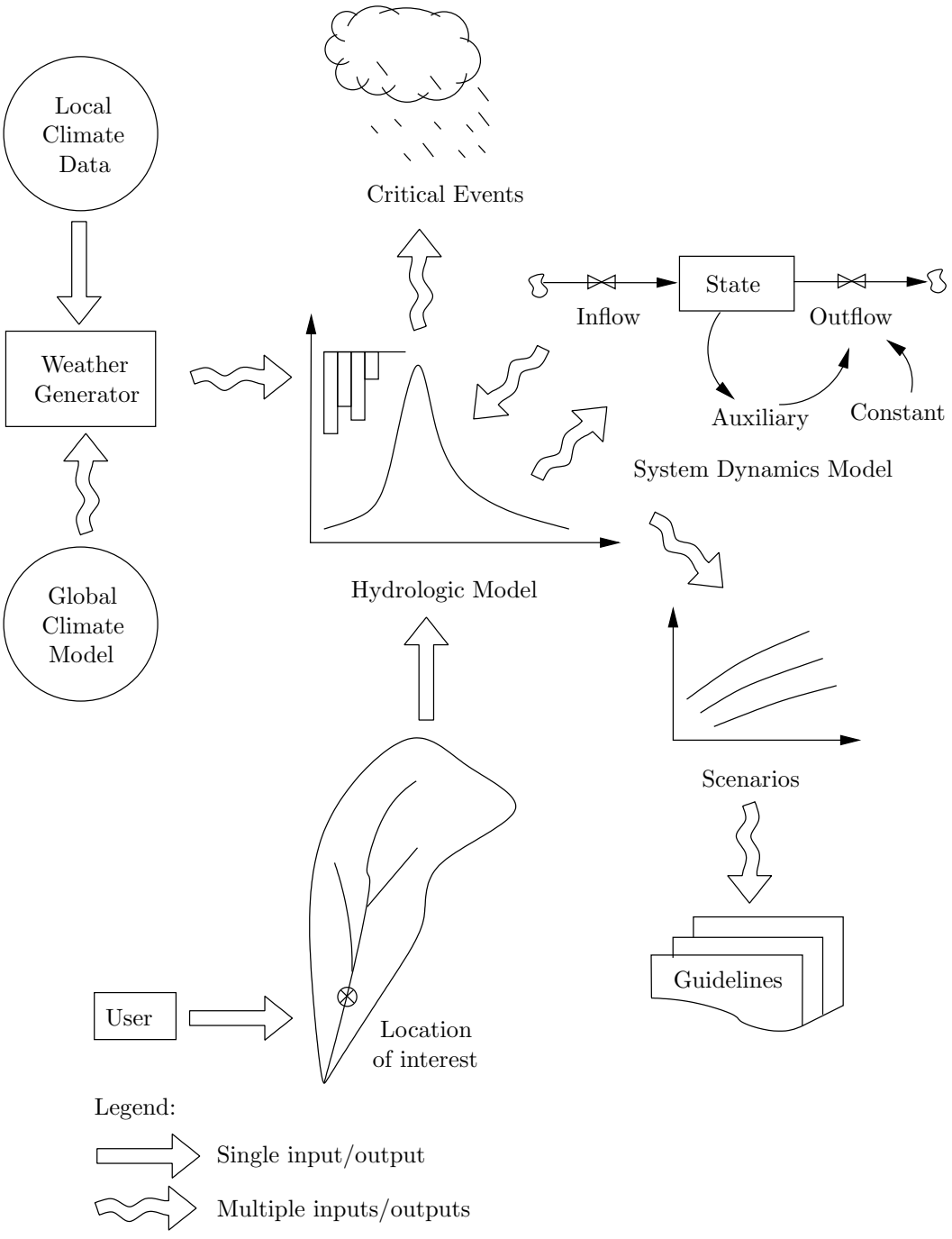

Figure 3: Overall schematic of the inverse approach

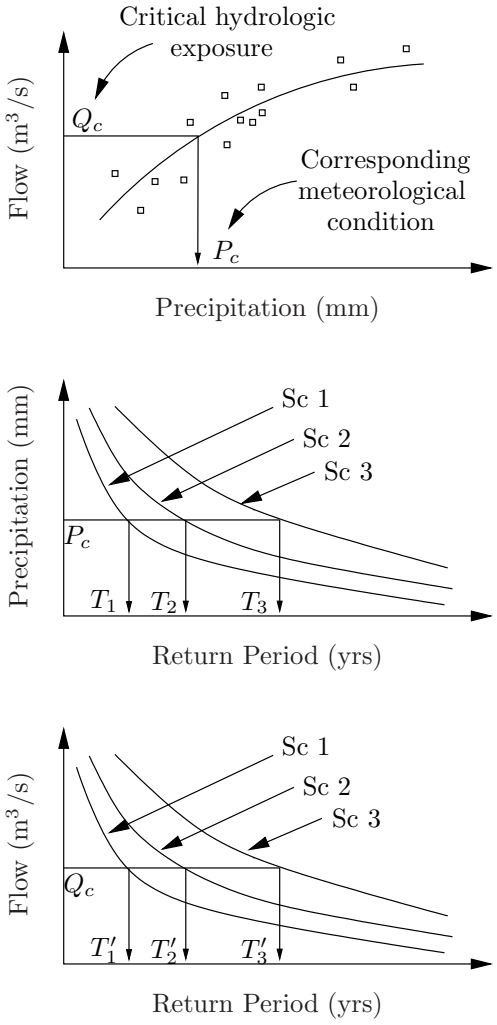

Figure 4: Use of the inverse approach

(step 4). As an additional option, the user is presented with a flow frequency analysis plot (bottom most plot in Figure 4), where return periods of flow can readily be acquired.

2.2 Continuous Hydrologic Model

The hydrologic model used in this work is described by Cunderlik and Simonovic (2004a, 2005), and so details of it will not be presented here. For the purposes of this report, it will suffice to say that the hydrologic model used is a semi-distributed continuous rainfall-runoff model based on the computational engine of HEC-HMS's Soil Moisture Accounting algorithm (Bennett, 1998; USACE, 2000). Continuous hydrologic models are those that 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.

Typically, the long term moisture characteristics are captured by conceptually representing a region with a number of reservoirs, with water flowing between them. The following reservoirs are normally represented in continuous hydrologic models: canopy, surface, soil, and various groundwater storage layers. The flows of water from and to the reservoirs include: precipitation (snow and/or rain), evapotranspiration, soil infiltration, percolation, groundwater flow (interflow and baseflow), as well as groundwater recharge. Figure 5 shows the conceptual structure of the continuous hydrologic model, based on the work of Bennett (1998). Other continuous hydrologic models found in practise include those of Leavesley et al. (1983) (work of Bennett (1998) is based on this model), Bicknell et al. (2001), Huber and Dickinson (1988)

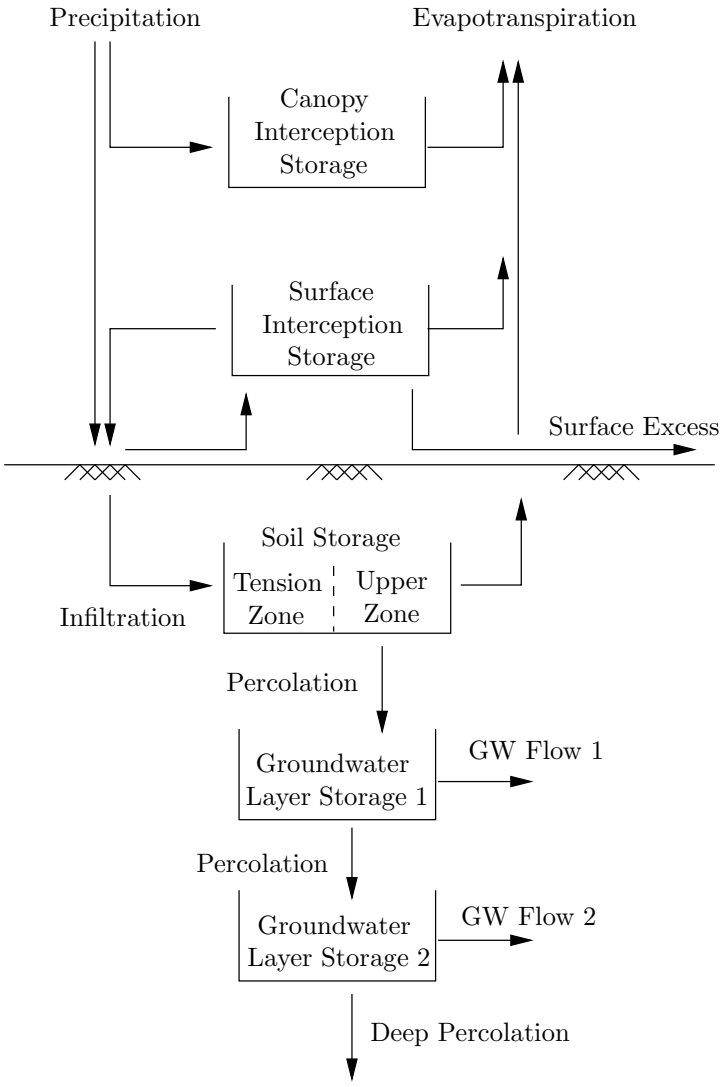

Figure 5: Structure of the Continuous Soil Moisture Accounting model

and others. For a detailed comparison of these and other hydrologic models available in the literature, see the report by Cunderlik and Simonovic (2003).

For completeness, we define here event based hydrologic models too. This category of hydrologic models simplifies the processes and relationships used to calculate the moisture balance (sometimes representing losses by a simple function or a coefficient). They are, therefore, suited best for determination of basin response from single storm events of relatively short duration (in the order of hours, sometimes a few days) (Bedient and Huber, 1988, p. 313). Evaporation and evapotranspiration are usually not included in such models, as the they are deemed negligible during the course of an event. These models are best suited for studies of basin response to short duration high intensity rainfall (or sudden warming causing excessive snowmelt), that typically produce flooding. Continuous models, although more rich in number of captured processes than their event counterparts, 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.

2.3 Weather Generator Model

The weather generator model developed by Sharif and Burn (2004, 2006a,b) is used in this work. In essence, a weather generator is a tool that synthetically creates weather and climate information for an area in question by using both local (historically observed) and global (produced by global models) weather data. 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 latest global climate models, and generates climatic information for an arbitrary long period of record. Sophisticated means are used to shuffle the historical data, and thus generate statistically similar climate for the region in question—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 latest information from global climate models that force it to generate climatic information not necessarily observed in the historical record (such as extreme floods and/or droughts). This means that the synthetically generated data is based upon the historically observed record, as well as information provided by the latest global climate models and their scenarios.

2.4 Inverse Hydro-Climatic Link

The inverse hydro-climatic link is referred to the functional relationship between critical hydrologic exposure and its corresponding meteorological conditions (see item 2 in Section 2.1). The selection of variables in constructing this relationship depends on the problem at hand. For short-term, high-intensity rainfall events that cause flooding, a relationship between daily rainfall and maximum peak flow may be ap-

propriate. For studies of low flows or droughts where long-term basin response is of interest, monthly or seasonal rainfall-runoff relationships may be used.

Note that the relationship between critical hydrologic exposure and its corresponding meteorological conditions is highly dependant on the watershed management practises. For example, a flow in a river is greatly dependant on how a series of water management structures (such as dams and reservoirs) are operated. Furthermore, water supply and irrigation practises are another set of management practises that alter the basin's flow characteristics. Modifying management practises thereby modifies the inverse hydro-climatic relationship as well.

3 Case Study

The inverse 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 on a map of Ontario is shown in Figure 6. This section provides a brief description of the study area and its regional characteristics, together with the necessary information needed to illustrate the application of the inverse approach.

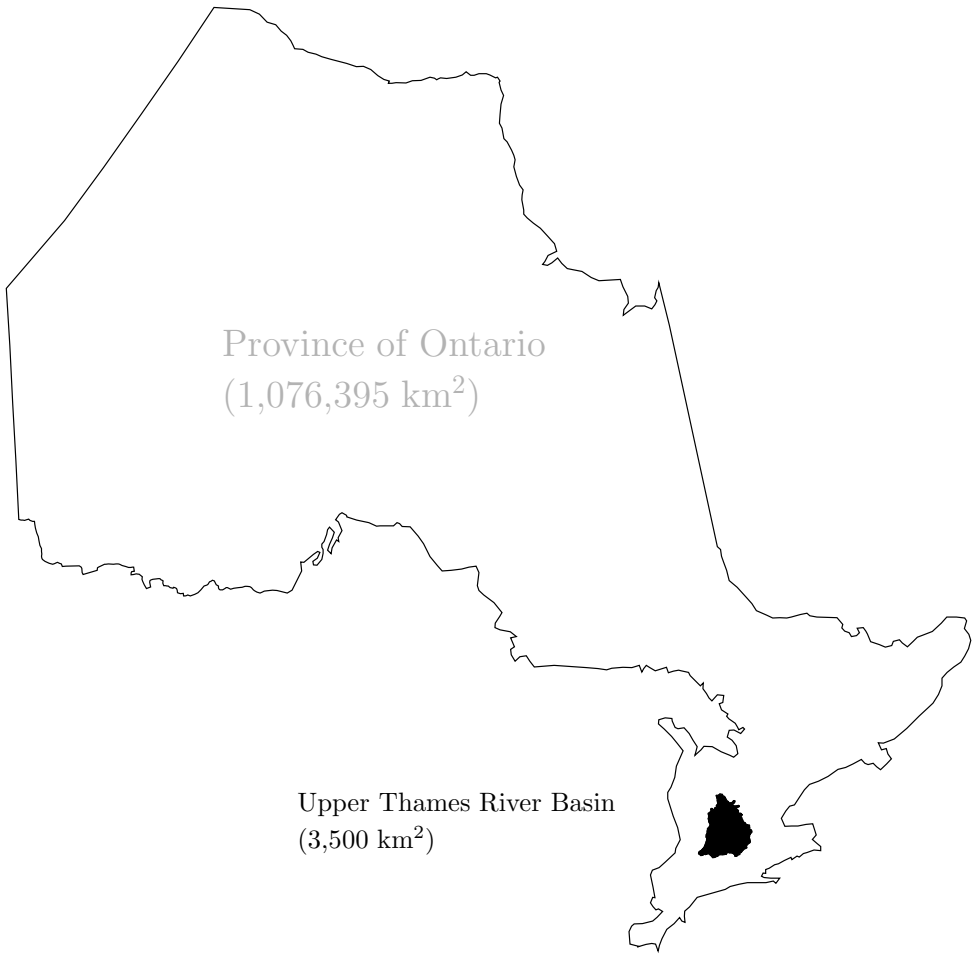

Figure 6: Location of the Upper Thames River basin within Ontario

3.1 Watershed Description

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 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 watershed 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 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 practise

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 450,000, 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.

The Thames river basin consists of two majors tributaries 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 watershed 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 groundwater (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.

The Upper Thames River basin includes three major water management reser-

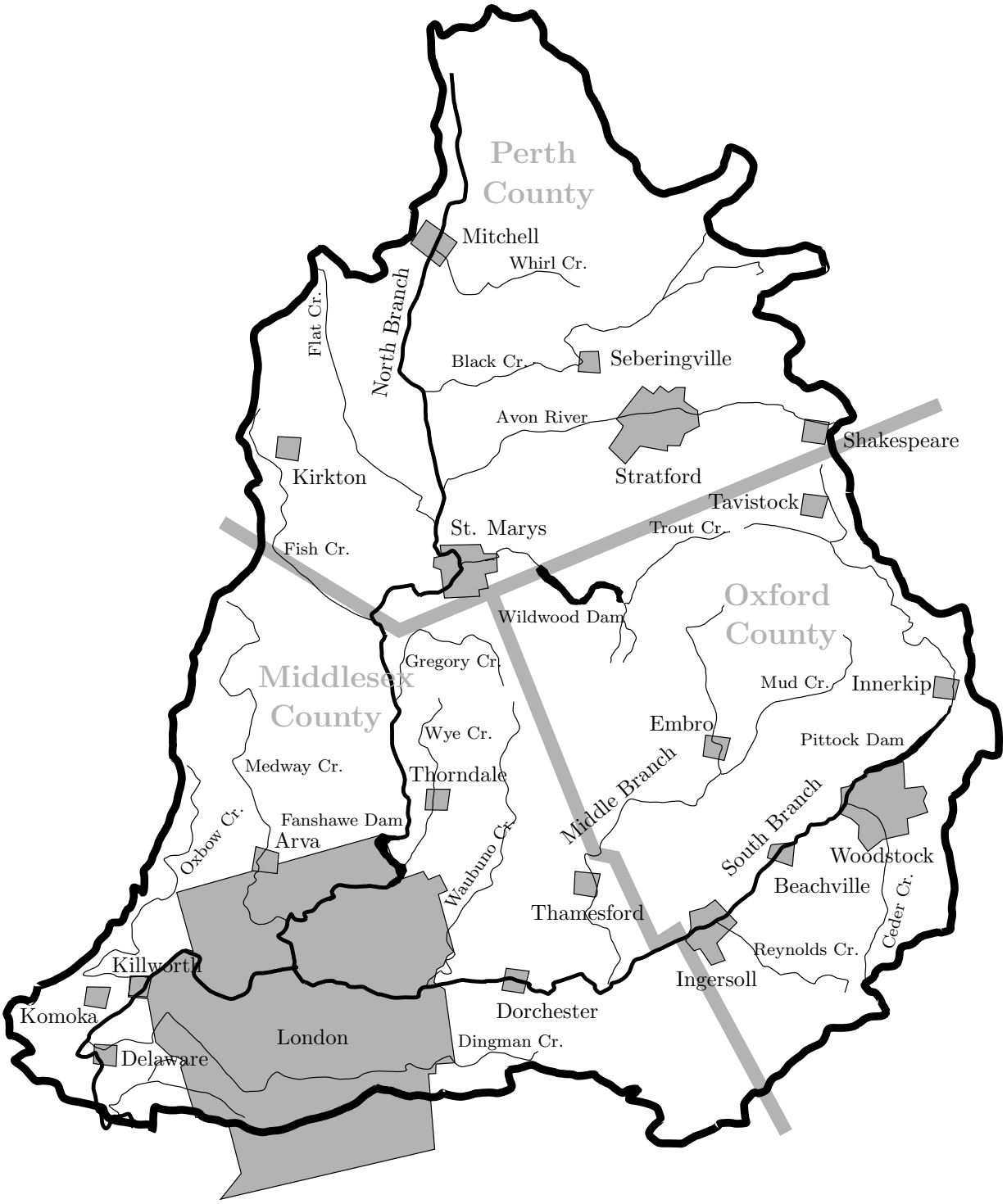

Figure 7: Map of the Upper Thames River basin

voirs: Wildwood, Pittock and Fanshawe near St. Marys, Woodstock and London, respectively. All three have been built in the mid 1960's, with the primary goal of flood management. Since then however, the reservoirs have seen their purposes expand to low flow augmentation, as well as recreational uses.

Floods and droughts represent major hydrologic hazards in the Upper Thames River basin. Flooding most frequently occurs after snowmelt, typically in early March; it also occurs as a result of summer storms usually taking place in July and August. Drought conditions are possible at any time of the year, although they are most frequent between June and September.

3.2 Climate Scenario Modelling

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 model 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 the 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 in the historical data. Two such scenarios are considered in this study, B21 and B11.

The scenarios B21 and B11 are based on IPCC (2001) scenario story lines B2 and B1, reproduced in Appendix A.1 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. In essence,

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 of increasing incidents of flooding, while the B11 scenario to examine increasing incidents of drought conditions. Even though results from both scenarios are shown side by side, the focus of this report is on the scenario B21. By having a historically similar long term record of climate information, together with two widely different scenarios, we can hope to capture a range of possible future conditions for consideration by policy makers, watershed stewards, stakeholders and users.

3.3 Weather Generator Modelling

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, along side with minimum and maximum temperatures. 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.) The names of the stations, along with their coordinate points are shown in Table 1, while a sample output for one station is depicted in Figure 8 for a five year period of record.

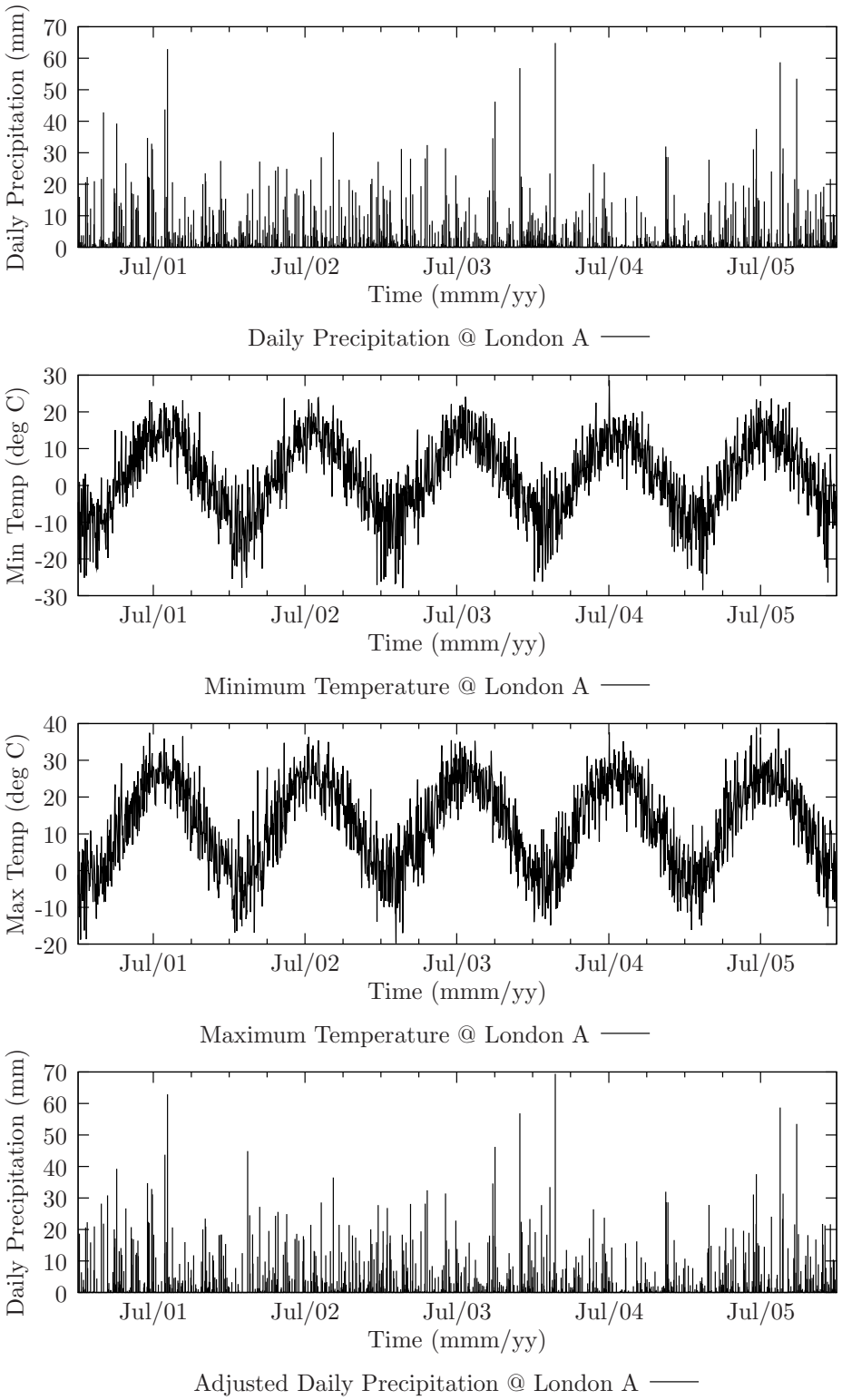

Figure 8: Weather Generator/Snow Model output for the historic scenario

Table 1: Locations of Generated Meteorological Data

Station	Latitude	Longitude
Blythe	43° 43'	-81° 22'
Dorchester	43° 00'	-81° 01'
Embro	43° 15'	-80° 55'
Exeter	43° 21'	-81° 30'
Foldens	43° 01'	-80° 46'
Fullarton	43° 23'	-81° 12'
GlenAllan	43° 40'	-80° 43'
Ilderton	43° 03'	-81° 25'
London	43° 01'	-81° 09'
StThomas	43° 46'	-81° 12'
Stratford	43° 22'	-81° 00'
Tavistock	43° 19'	-80° 49'
Waterloo	43° 28'	-80° 31'
Woodstock	43° 08'	-80° 46'
Wroxeter	43° 52'	-81° 09'

3.4 Snow Accumulation and Melt Modelling

Time series of temperature and precipitation data, before being used by the hydrologic model, are used by the externally built snow accumulation and melt model (Cunderlik and Simonovic, 2004a, p. 65). In essence, the snow model separates solid and liquid forms of precipitation based on the average temperature during each day. The solid form of precipitation is then subject to an accumulation and melt algorithm based on the degree-day method. That part of solid precipitation that is melted is added to the liquid precipitation, thus producing a variable called "adjusted" precipitation. It is this precipitation that is used next as input to the continuous hydrologic model (see bottom most plot in Figure 8).

3.5 Continuous Hydrologic Modelling

The continuous hydrologic model, originally developed by Cunderlik and Simonovic (2004a, 2006), is used in this study. The original model 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 that seasonality be included in the model. Seasonality is a model feature where a different parameter set is used for summer and winter seasons. Furthermore, the model is to be executed for a time period of one hundred years—which would mean that each time a new season is reached, a new parameter set would have to be loaded manually. For a simulation of one hundred years, this would have to be done two hundred times; for three scenarios, six hundred times—clearly a tedious task. Simulating the model for such a long period of record is necessary as one of the final goals of the study is to perform a frequency analysis on various hydro-climatic variables, and thus estimate changes in return periods (or frequencies of occurrence) of precipitation and flow variables.

To cope with this, the computational engine of the continuous version of the HEC-HMS model was built from scratch in the Java programming language. An additional benefit of this is that the model can easily be coupled dynamically with other models (such as a system dynamics socio-economic model), where the results of one model influence, and are influenced by the other. One such version of the combined model has been completed, and is described in a separate report (Prodanovic and Simonovic, 2006a).

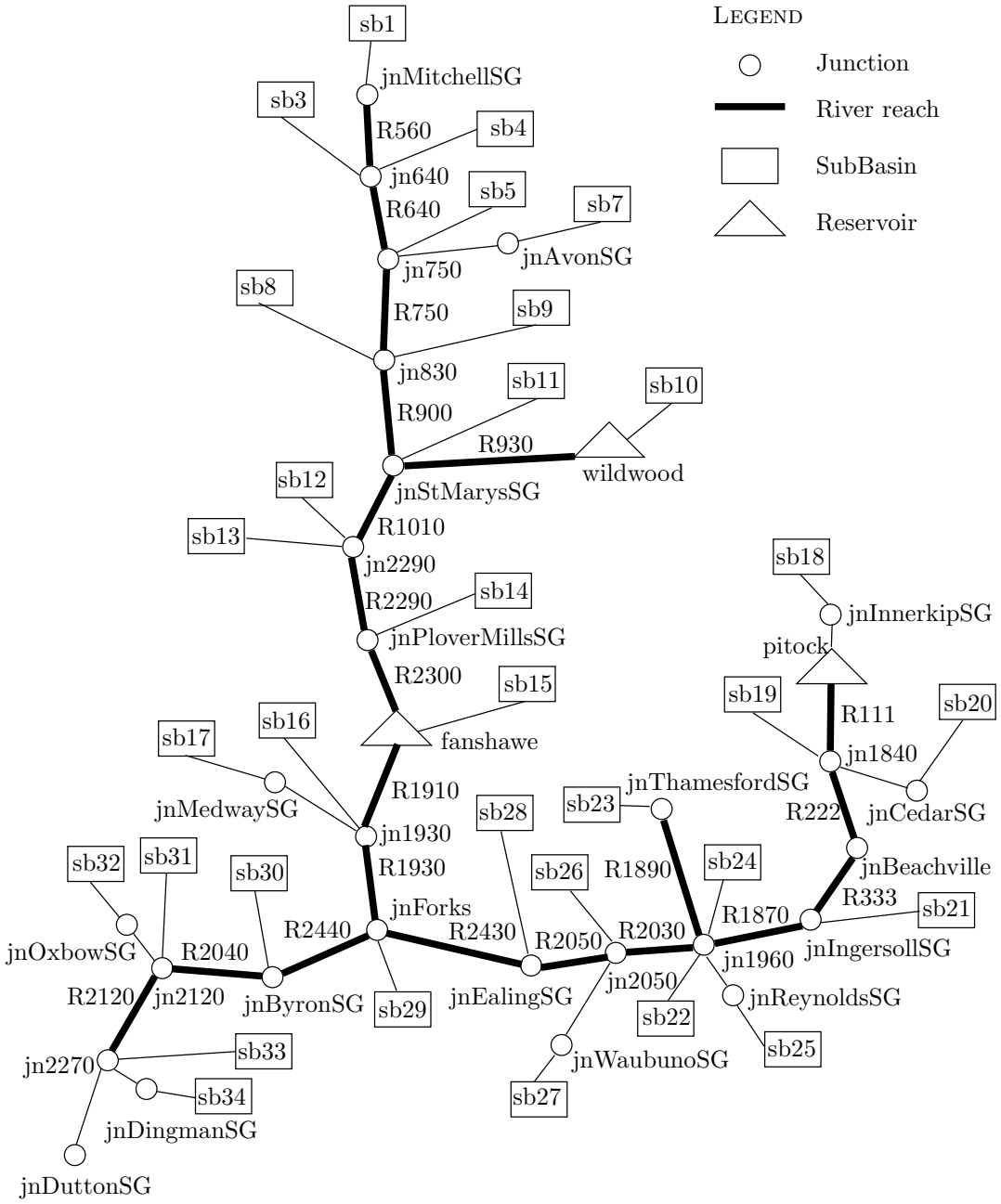

Figure 9: Schematic of the continuous hydrologic model

The continuous hydrologic model used in this study consists of thirty two spacial units (also called subbasins or subcatchments), twenty one river reaches, and three reservoirs. The schematic of the model as applied to the Upper Thames River basin is shown is Figure 9. Each subbasin component contains a further set of sub components representing losses, transform and baseflow methods. The losses in this work are represented via HEC-HMS's Soil Moisture Accounting algorithm, and form the most detailed sub component of the model. The transform method (i.e., where excess rainfall is transformed to direct runoff) applied is one based on Clark's Unit Hydrograph, while the baseflow is represented with a cascade of linear reservoirs for each groundwater layer.

River and flood routing are computed via the Modified Puls method, using storage-discharge curves provided by the staff of Upper Thames River Conservation Authority. The interested reader is referred to USACE (2000), Bennett (1998) and Hoggan (1996) for technical details on these and other methods. The source code (available in Appendix A.4) provides an implementation of said methods to the Upper Thames River basin, together with all relationships and parameters used.

One of the features of the hydrologic model deserves special mention: the flood routing in the reservoirs. At the time of model development, only the Modified Puls method was available for calculating reservoir releases—this was the only option available in version 2.2.2 of HEC-HMS. It must be acknowledged that reservoirs (especially Fanshawe) may not be operate according to the procedures of the Modified Puls method in real life. The calculated releases from the reservoirs thus represent only approximations of management practises actually employed by those responsible for their operations. Since the basin response is extremely sensitive to reservoir operations (especially 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 Appendix A.4), which can easily be adopted to custom modelling of the regional reservoir operating rules, and thus representing hydrologic conditions more accurately.

Before the hydrologic model is executed, a rainfall hyetograph must be specified for each of its thirty two subbasins. However, the rainfall data is provided only for fifteen stations, and sometimes the locations of stations do not correspond to the locations of the subbasins. The climatic signal provided by the weather generator is therefore spatially interpolated in order to be used by the hydrologic model. Of the many available methods, Inverse Distance Weighting Method (USACE, 2000) is used for interpolation, mainly because of its mathematical simplicity and easy implementation. Further arguments in favour of this method is given by Wey (2006). The interpolated rainfall data is then used as input to the hydrologic model, which produces its response in the form of hydrographs. Figures 10 and 11 show the interpolated precipitation for three locations within the basin, as well as its corresponding hydrographs, respectively. The hydrologic model operates on a 6 hour time step, despite the fact that its input data is provided on a daily interval. This is required as some smaller subcatchments in the basin have relatively short response times (i.e., short times of concentration).

3.6 Regional Analysis

In order to illustrate the application of the inverse method, three locations are selected in the Upper Thames River basin—Byron (in Middlesex County), Ingersoll (in Oxford County) and St. Marys (in Perth County). The selected locations represent stream

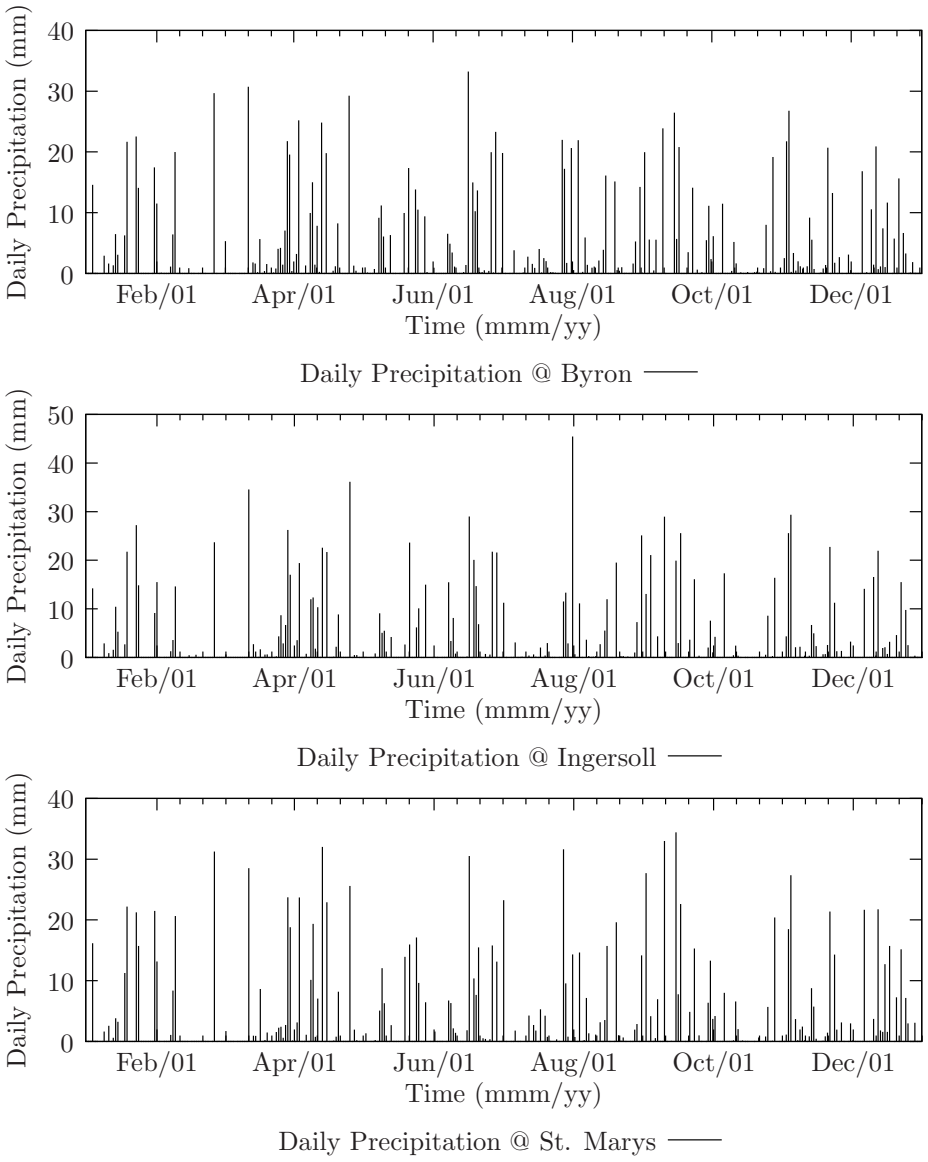

Figure 10: Daily precipitation for historically identical scenario

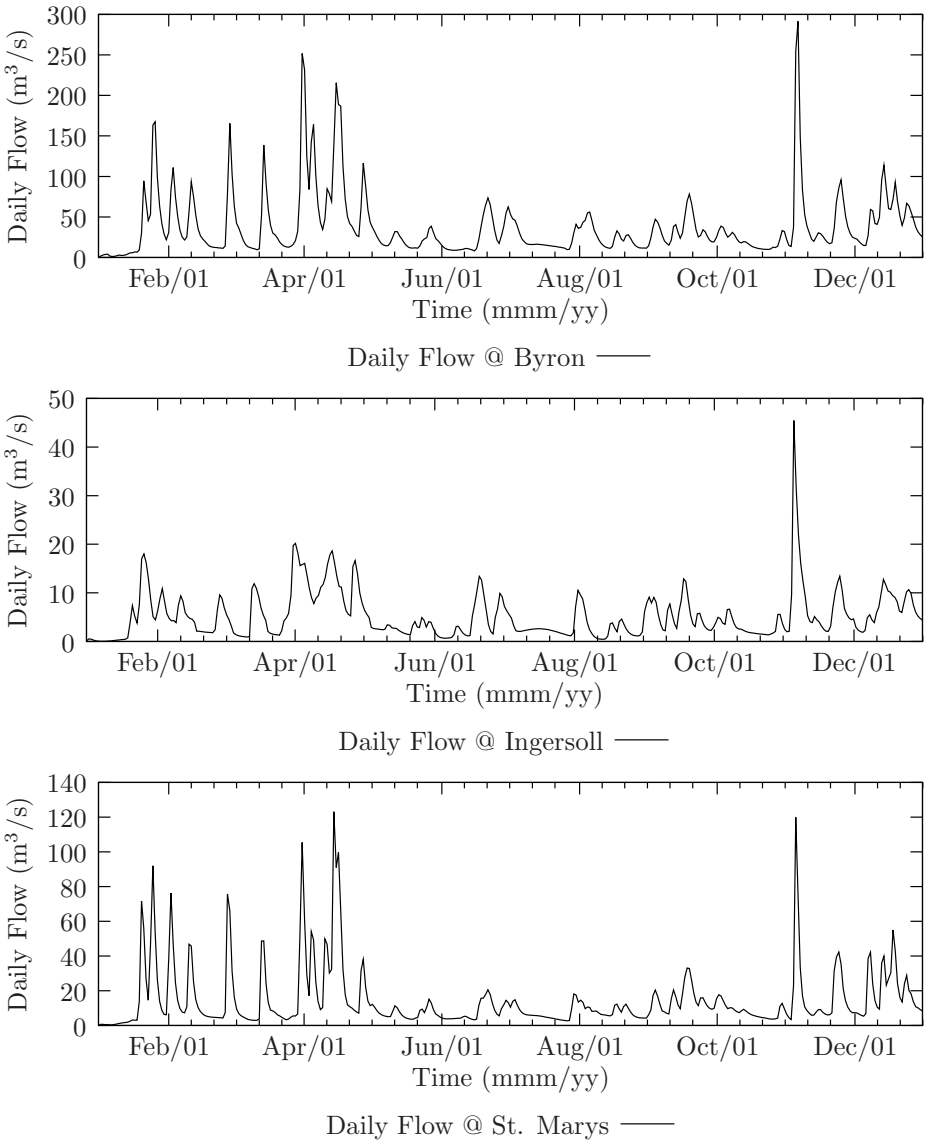

Figure 11: Daily flow for historically identical scenario

gauges in three different branches of the Thames—north, east and west, respectively. Each location has slightly different hydro-climatic characteristics, and our aim is to capture them in the analysis.

In order to apply the inverse approach to the above locations, a number of assumptions are made. For example, the step 2 requires that a monthly precipitation-flow curve be constructed. As part of this analysis, subbasin hyetographs (this is the precipitation that had been previously interpolated) upstream of the stream gauge in question are used to spatially aggregate the precipitation. For example, the St. Marys stream gauge collects runoff from subbasins 1-10 (see Figure 9); the rainfall hyetographs for subbasins 1-10 are aggregated by the weighted area method to arrive at a single rainfall hyetograph corresponding to the St. Marys stream gauge. It worth pointing out that as the upstream area gets larger (as in Byron for example, where almost the entire basin drains to), more spacial aggregation takes place and thus tend to smooth the final output.

After the results of the weather generator and hydrological models are obtained, the frequency analysis of both monthly precipitation and monthly flow is undertaken. Since extreme low flow characteristics are of interest, the Weibull extreme value distribution is used. This distribution has been used for a long time in hydrologic analysis of low flows. For details on the mathematical procedures in hydrologic frequency analysis, see any text on hydrology (Bedient and Huber, 1988; Hoggan, 1996; Linsley and Franzini, 1979); for details on the Weibull extremal distribution, see Benjamin and Cornell (1970). The interested reader is also referred to the code listing of Appendix A.4 for appropriate equations and algorithms of the Weibull and other statistical distributions.

4 Results and Discussion

The results and discussion of the application of the inverse method (Section 2) to the Upper Thames River basin (Section 3) are presented here for three locations—Byron, Ingersoll and St. Marys. This section identifies the hydrologic exposures related to droughts, outlines the procedure used to formulate the inverse relationship, demonstrates the steps followed in the approach, and suggests possible changes to current drought management guidelines.

4.1 Critical Hydrologic Exposures

The first step of the inverse approach (as it applies to droughts) involves identification of drought related critical hydrologic exposures within the basin. In consultation with the staff from the Upper Thames River Conservation Authority, as well as meeting with with various stakeholders in the basin, it was established that critical drought exposures (although acknowledged as real and possibly threatening) do not play a major role in current water resources management practise. One of the reasons for this is that impacts from severe droughts have not been felt in this area for quite some time; studies of agricultural and other drought related damage for this area is virtually non-existent. Drought damage is further masked by widespread use and availability of water for agricultural, industrial, commercial, and residential purposes, especially in the County of Oxford where water use is highest in the region. For reference, it is noted that Oxford and Perth counties are predominant users of groundwater, while those living in the Middlesex County, including the city of London, get their water from Lakes Huron and Erie as well as groundwater sources.

In year 1999 in the province of Ontario a governing body named Ontario Low Water Response was formed to monitor drought conditions in watersheds province

wide. The governing body consists of a number of agencies (provincial, federal and local), all of which are responsible for monitoring, reporting and providing information bulletins to the public regarding droughts. The same governing body produced a document (OLWR, 2003) that outlines specific definitions on how droughts are defined in the province. Based on these drought definitions, information in Table 2 is presented, and serves to identify drought related hydrologic exposures as they relate to less than normal streamflow[4] conditions. It is these exposures that shall be used to illustrate the inverse approach.

4.2 Inverse Drought Risk Relationships

The next step in the application of the inverse approach is the construction of a relationship that relates hydrological extremes (such as low flow) to their corresponding meteorological conditions (such as precipitation). Since no universally accepted long term rainfall-runoff relationships exist, a number of different forms have been tested. Linsley and Franzini (1979) provide some guidance in this task, as they suggest that both annual and seasonal relationships be used (p. 49). Annual curves have been constructed first, but proved to be of little use since droughts are seasonal phenomena. (The annual rainfall-runoff curves show a plot of total annual precipitation versus average annual flow.) With such curves for example, it is possible that heavy winter precipitation can mask less than average summer precipitation (and flow), thus concealing potentially severe drought conditions. For this reason, annual precipitation-flow curve was rejected.

Next, an attempt was made to construct seasonal, and monthly rainfall-runoff relations. Again, guidance from Linsley and Franzini (1979) provided to be useful, as

[4]It should be noted that OLWR (2003) defines droughts based on a combination of streamflow and precipitation records. For the purposes of illustrating the inverse approach, only drought definitions based on streamflow are used.

Table 2: Critical drought exposures in the Upper Thames River basin

Monthly Streamflow	Level[†]	Drought Exposure
Less than 70 % of lowest average summer month	I	Potential water supply problems; Request volontary conservation; Ask for 10 % reduction in water use; Inform media of drought conditions.
Less than 50 % of lowest average summer month	II	Minor supply problems, but potential for major problems; Request volontary conservation of additional 10% and impose restrictions; Contact major water users; Limit new water use permits; Monitor and enforce complience with existing permits; Modify reservoir operations.
Less than 30 % of lowest average summer month	III	Supply fails to meet usual demand; Social and economic impacts; Impose maximum conservation, restriction, and regulation; Reduce water permit levels; Set and institute water use priorities; Enforce water use; Consider hauling water.

[†] Drought levels are based on the Ontario Low Water Response guidelines (OLWR, 2003).

they suggest that:

> It is usually necessary to develop separate relations for each month of the year. The relationships for two or more months may be the same, but distinct seasonal differences will be evident. In some months a storm near the end of the month may produce a large volume of runoff in the subsequent month. Such data must be either adjusted or neglected in developing the rainfall-runoff relation (p. 50).

Precipitation-flow curves have been constructed for the combined summer months (June, July, August and September), plotting monthly total aggregated rainfall versus monthly average flow. Only summer months are used as this is time of the year when drought conditions are most severe. Following this, it was observed that all relationships formulated had such large amount of scatter, that no functional form could be found to fit them. About 2000 commonly used functions were tested with a curve fitting software package, all yielding relatively poor results.

After this, an adjustment referred to by Linsley and Franzini (1979) was attempted on the summer seasonal curves. In order to adjust the data, a time lag was introduced that plots average monthly flow against monthly precipitation lagging back a specified number of days. For example, if the lag time is 21 days, the flow at the end of the current month is plotted against precipitation that is delayed by 21 days, but still of the same four week duration as its corresponding flow (see Figure 12). In essence, this means that monthly precipitation lagging 21 days is responsible for producing the current end of month average flow.

After trial and error (trying different lag times, plotting the results and fitting them to various curves), the following three parameter function was selected as the precipitation-flow relationship:

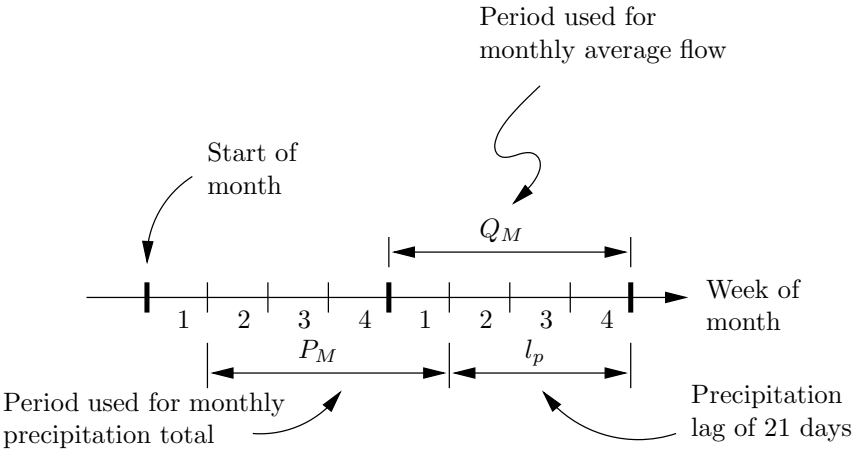

Figure 12: Illustration of the precipitation lag in the inverse relationship

$$Q_M = a + bP_M^c \tag{1}$$

where Q_M is the monthly average flow [m^3/s], P_M is the monthly precipitation lagged by 21 days [mm], and a, b and c are empirical constants. Table 3 shows various lag times that were tested, with their statistical measure of goodness of fit (i.e., r^2 values). By experimenting with different lag times, a value of 21 days was selected as optimal because it yielded the best fit to the data. The goodness of fit of a particular function can range from zero (complete scatter) to unity (perfect fit), with all values in between. The plots in Figure 13 show the summer season monthly rainfall-runoff relationship, fitted to a functional form of equation (1). The precipitation values in Figure 13 have been obtained by using the historically identical weather generator scenario for 100 years, while streamflow was acquired by simulating the hydrologic model.

In the application of the inverse approach, frequency curves (where precipitation and flow are plotted versus their corresponding return period) are used in order to compare return periods under changed climatic conditions. In order to construct the

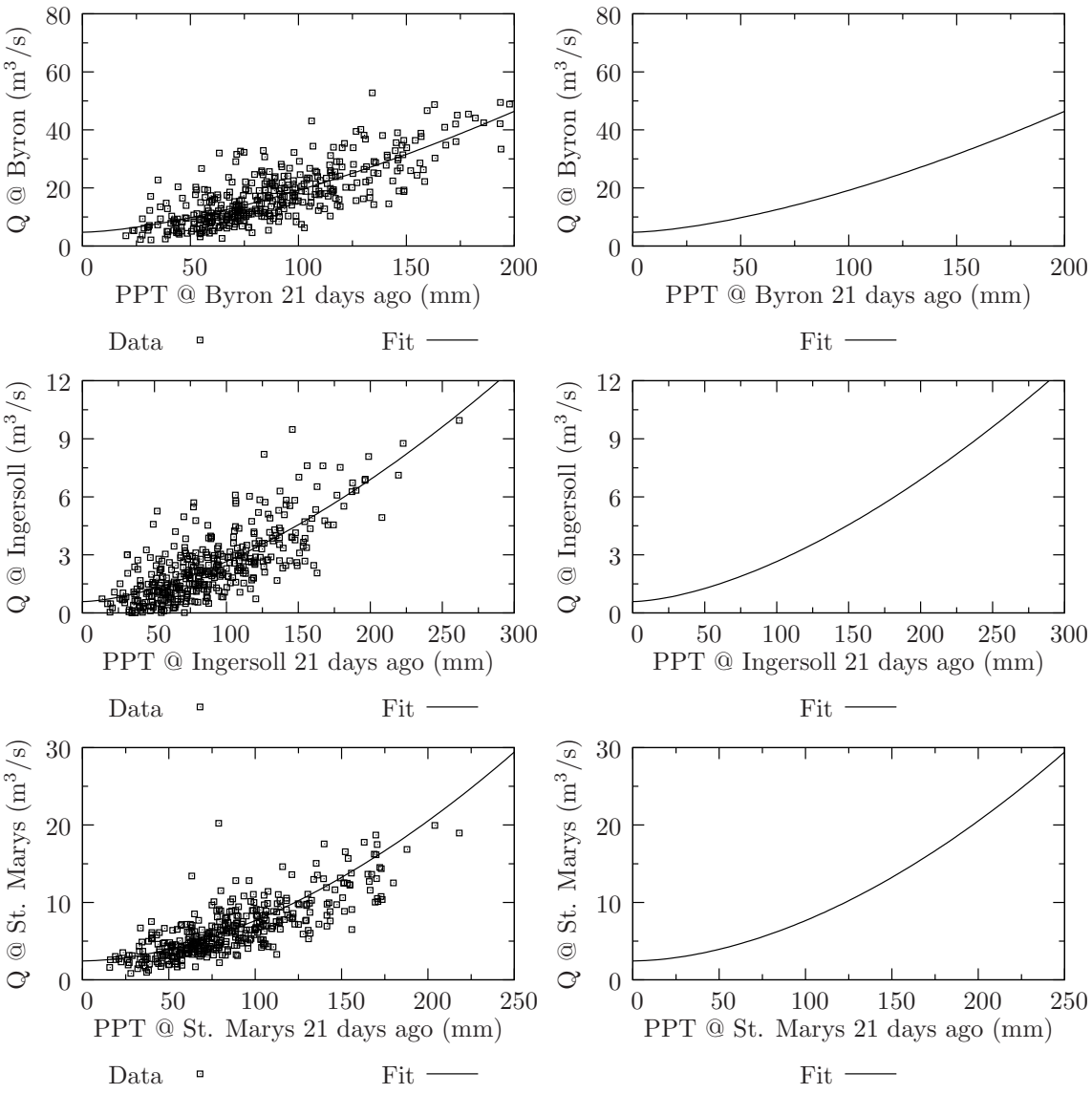

Figure 13: Seasonal (Jun, Jul, Aug, Sep) inverse relationship

Table 3: Goodness of fit (r^2) of monthly precipitation flow relationships

Location	Precipitation lag time				
	7 day	14 day	21 day†	30 day	60 days
Byron	0.30	0.52	0.56	0.42	0.01
Ingersoll	0.28	0.49	0.54	0.49	Poor
St. Marys	0.25	0.44	0.53	0.38	Poor

† Optimal lag time.

frequency curves, annual extremes need to be extracted from the time series data, which are then used to calculate return periods or fit statistical distributions. This type of analysis is therefore performed using seasonal data shown in Figure 13. Annual minimum monthly flows are selected, together with their corresponding precipitation (which is still lagged by 21 days), and new monthly rainfall-runoff curves are constructed. This is done to assure that same data is used in the inverse relationship as in the frequency curves, and thus guarantees a meaningful comparison of the results.

The precipitation-flow curves based on the annual minimum flow (and its corresponding precipitation) are shown in Figures 14-16. The frequency curves prepared for each location show results of a Weibull distribution fit of the annual minimum extremes, for thee climate scenarios (Historic, B11 and B21). The interested reader is referred to Appendix A.2 and A.3, where parameter values for all rainfall-runoff curves are shown, together with Weibull probability paper plots showing goodness of fit of a distribution as it compares to the fitted data.

4.3 Use of the Inverse Approach

In order to illustrate the application of the inverse approach, we start by identifying a critical drought exposure of interest. Suppose that we are interested in studying the frequency of occurrence of level II drought (see Table 2) in the vicinity of the Greenway

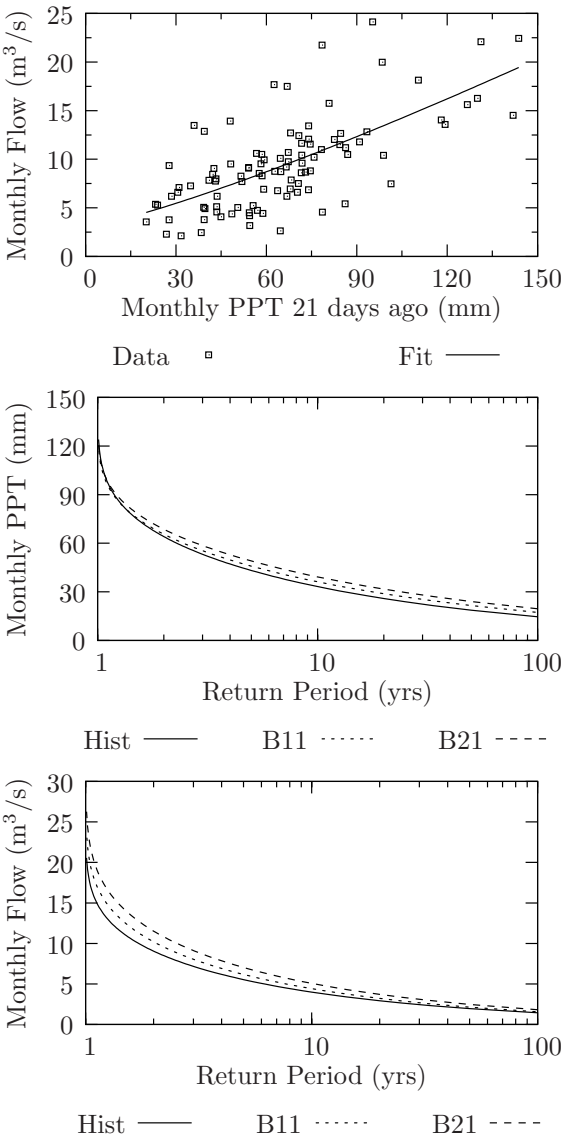

Figure 14: Annual minimum inverse link for Byron

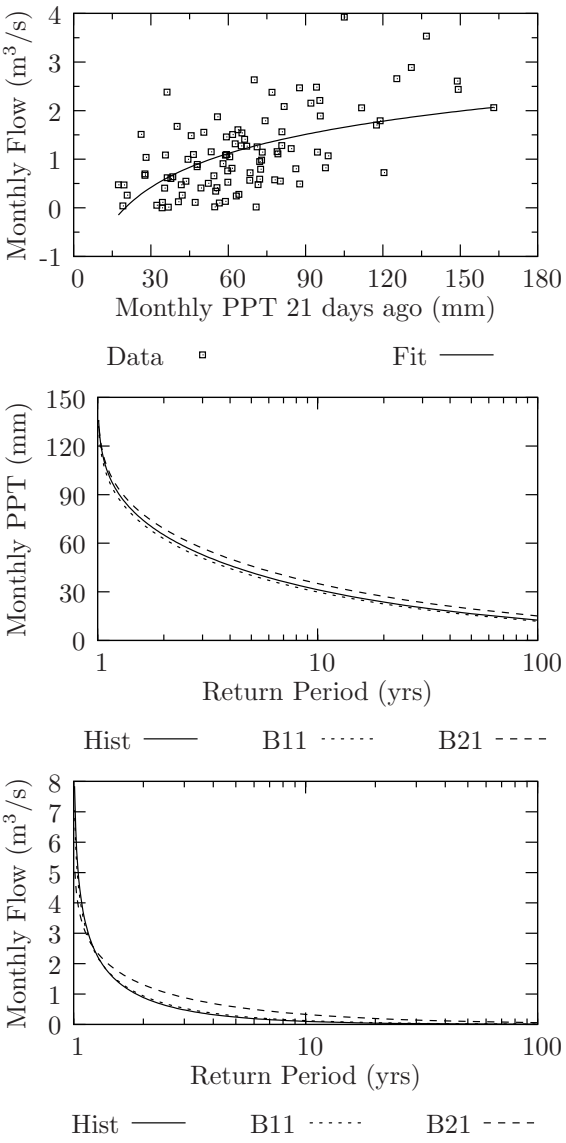

Figure 15: Annual minimum inverse link for Ingersoll

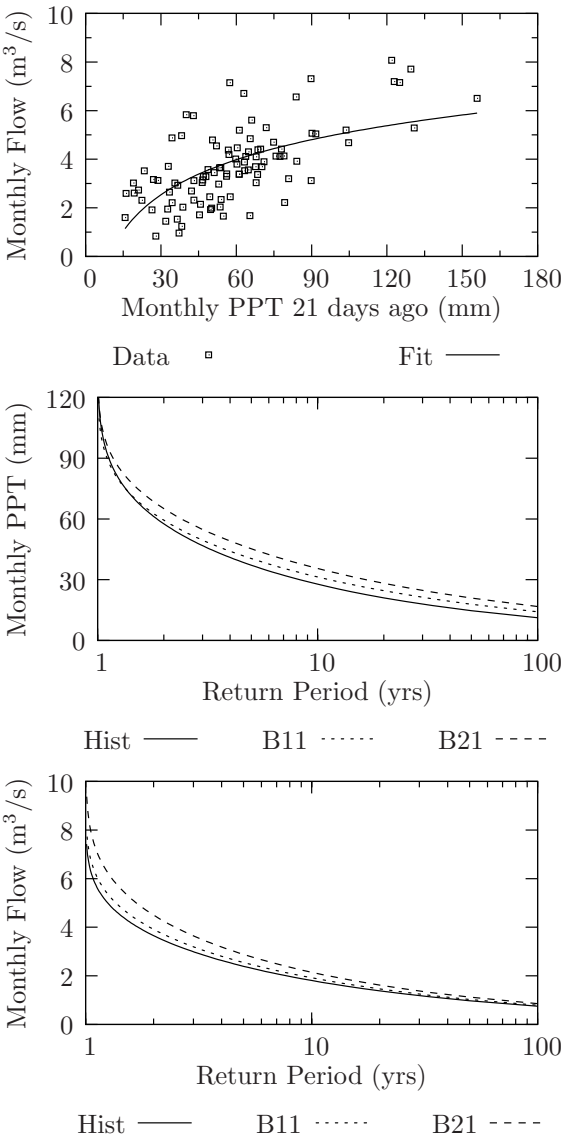

Figure 16: Annual minimum inverse link for St. Marys

Pollution Control Plant in Byron. (Greenway is the largest sewage treatment plant in the basin, having the capacity of 150,000 m^3/day of waste water, and is always running at or near capacity.) By looking at the time series record of simulated flow for the historically identical weather generator scenario, and performing the necessary analysis, we see that the lowest average summer month flow for the Byron stream gauge is 9.5 m^3/s. For reference, the lowest average summer month flow for the Ingersoll and St. Marys stream gauges are 1.0 and 3.8 m^3/s respectively. These numbers are obtained by selecting in each year a lowest average summer month flow, and averaging it over a period of record (100 years in our case).

In order for level II drought to occur, the lowest average summer flow must drop to 50% of its average. For the Byron stream gauge, this translates into a flow of 5 m^3/s. The next step in the inverse approach is to convert the critical hydrologic exposure of 5 m^3/s to a corresponding critical meteorological condition. We do this by consulting the top plot of Figure 14; the precipitation that corresponds to the flow of 5 m^3/s is about 30 mm of monthly rainfall. After this, we look up the *frequencies of occurence of rainfall* in the middle plot of Figure 14. For the historically identical scenario, we see that a return period of such an event is 12.5 years, while for the B11 and B21 scenarios the return period is 20 and 25 years respectively, thus indicating that future drought conditions will occur less frequently than they do today. A return period of T years means that an event equalling to or exceeding the said value has a chance of occurrence of 1 in T, any given year. The definition of the return period is commonly misinterpreted as the number of years between events; this is the incorrect interpretation, and should be avoided. For example, a return period of 12.5 years means that each year the chance of occurence of such an event is 1 in 12.5; under the changed climate signal, it may be either 1 in 20 or 1 in 25.

Alternatively, the same critical hydrologic hazard of 5 m^3/s may be used directly in

conjunction with the monthly flow frequency curve (see bottom most plot in Figure 14), therefore bypassing the precipitation altogether. Using the same value of the monthly flow, we can obtain *flow return periods* for all climate scenarios. (In contrast, the previous paragraph dealt with the return periods of precipitation, not flow.) Under the historic case, the drought level II has a return period of 5.8 years, while under the changed climate it may occur less frequently (with a return periods of 7.14 and 12.5 years for B11 and B21 scenarios respectively).

It is important to mention that differences between the return periods of precipitation and the return periods of monthly flow are present because the rainfall-runoff relationship is fitted to data consisting of a large amount of scatter (see top plots of Figures 14-16). A possible reason for such scatter lies in temporal distribution of precipitation. This is simply another way of saying that not all subcatchments in the basin respond identically to monthly precipitation. Of course, different temporal distribution of rainfall necessarily produces different basin response. But, the inverse hydro-climatic link that uses monthly precipitation in its relationship, may miss this feature as it aggregates daily information into monthly values. It is very easy to think of an example of two months having identical rainfall totals but different temporal distribution, thus producing different basin responses. This may lead to different monthly average flows for the same monthly total precipitation, thus contributing to the scatter in the data. Therefore, depending on whether the user is interested in return periods of precipitation or of flow, slightly different results shall be obtained.

4.4 Recommendations for Revision of Guidelines

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 the Upper Thames River basin are currently quite minor. As was alluded to earlier, drought damage related studies in the basin are altogether lacking, despite the fact that potentially severe droughts, as present analysis shows, can occur with a chance of 1 in 10 any given year.

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 watershed characteristics such as predominant land use patters (urban, rural, agricultural, forest), water use (surface or groundwater), 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 watershed 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) tells us that in the Upper Thames River basin elevated phosphorous concentration results from:

- agricultural runoff and groundwater 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).

SNC-Lavalin and RV-Anderson (2005) tell us 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 practises (particularly in rural areas that drain most of the watershed) 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 groundwater 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 groundwater 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. 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 recommendations 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 practises, 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 subwatershed 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 watershed (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 practises, 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, where:

> The highest leverage lies in clarifying the quality of life we envision for ourselves, and then using that as a guide for creating the systemic structures that will help us achieve that vision (Kim, 1994, p. 5).

5 Conclusions

The main objective of this project is to study risk and vulnerability to changing climatic conditions; in this work, the focus is on situations causing droughts. (Other reports in this series address the problem of flooding as well as socio-economic impacts.) Of the many different accounts, vulnerability in this project is defined through incremental losses occurring due to changes in frequency and magnitude of climatic conditions. Once quantified, the basin wide vulnerability assessments are seen as a starting point towards formulation of new (or improved) management guidelines aimed at reducing risk associated with hydrologic hazards (and in this case, droughts in particular). This information is critical to water resources managers, engineers, planners and other government and private sector officials, as it gives them a glimpse of what the changed climate may bring. In this case, droughts are expected to occur less frequently.

This report outlined a new approach to water resources risk assessment, and developed a number of tools used as aids in quantifying effects of changed climatic conditions on a small (basin wide) scale. The methodology (see Section 2), including use of hydrologic and weather generator models can easily be applied to other regions with minimal adjustments. One of the central points of the approach has been the continual involvement of end users of water resources systems, as they who will have to act and/or formulate new management guidelines in light of this information. Too often, reports and assessments have been made without consultation of end users, and as a result, the recommendations have been rarely implemented.

One of the benefits of the proposed approach is the ease with which it can be employed by an end user of the basin. Although illustrated with one example only (the problem of level II droughts in Byron), interested readers can readily use the

tables and charts provided to perform analysis of their own, and thus estimate risks and vulnerability to climatic change to problems of most interest to them. This is indeed encouraged.

Two of the climate scenarios considered show different possible futures in terms of hydro-meteorologic conditions in the basin. Even though scenarios B11 and B21 show less frequent occurrence of drought conditions (when compared to the historic or base case scenario), they still occur often enough to cause serious concern (with return periods of 10 years, or less). Because of this, the emphasis should be placed on preparing management alternatives and mitigation measures that address a range of possible outcomes. In particular, smaller scale (or local) drought definitions and triggers are altogether lacking, as are studies of potential drought damage (locally and basin wide). It is our recommendation that these studies be undertakes, or at least be included as part of future budgets and planning.

With this information, it is believed, better and more flexible management practises can emerge that should ultimately minimize the regions risk and vulnerability to drought. Based on the analysis presented, recommendations are made for future work which include: (i) drought impact studies (where impacts to agriculture, recreation, wetlands, reservoir operation, ground water withdrawal and streamflow quality are assessed); (ii) definition of local drought triggers (including guidelines on subwatershed scale, as well as monitoring how drought triggers can change over time); (iii) water quality management (setting in place practises that enhance water quality over short and long term); (iv) education programs (to bring up to date knowledge in science to all who stand to be adversely impacted by drought).

The guidelines presented in this report are based on the knowledge and insight obtained by simulating the rainfall-runoff process with a continuous based hydrologic model of the study area. The current modelling framework is able to demonstrate

what the potential impacts are, and what management options should be considered to alleviate possible negative consequences. However, the current model does not have the capability to show how a particular management option should be selected, and what its impact would be over both the socio-economic and physical domains. A different kind of model is needed for this, as water resources management process inevitably involves not only processes that are physical (such as droughts), but those that are social as well (such as land use planning and development, rural and urban business activity, population, etc.). This kind of model is able to show the interaction between the physical and social domains, and is thus able to show both social and physical impacts as they evolve through time. It is also able to test different management strategies (increasing water use, population and business growth, increasing land use and development, etc.) and thus estimate the response of the water resources system as a result of these changes. It is important to note that both social (population, business activity, land use changes) and physical (floods/droughts and its associated damage) information is conveyed by such a model. This kind of model has been developed as part of this study (Prodanovic and Simonovic, 2006a), and should be used in combination with conclusions of this work.

References

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

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.

Bicknell, B., Imhoff, J., Kittle, J., Jobes, T., and Donigian, A. (2001). *Hydrological Simulation Program-FORTRAN (HSPF), User's manual for version 12*. Aqua Terra Consultunts, Mountain View, California.

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

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.

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 Simonovic, S. P. (2003). "Assessment of water resources risk and vulnerability to changing climatic conditions: Hydrologic model selection for the cfcas project." *Report No. I*, The University of Western Ontario, London, Ontario, Canada.

Cunderlik, J. M. and Simonovic, S. P. (2004a). "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*, The University of Western Ontario, London, Ontario, Canada.

Cunderlik, J. M. and Simonovic, S. P. (2004b). "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).

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.

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

Huber, W. C. and Dickinson, R. E. (1988). *Storm Water Management Model (SWMM) User's Manual, version 4.0*. United States Environmental Protection Agency, Athens, Georgia. 595 pp.

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.

Kim, D. (1994). *Systems Archetypes II: Using systems archetypes to take effective action*. Pegasus Communications, Inc., Cambridge, Massachusetts.

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.

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.

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.

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.

Prodanovic, P. and Simonovic, S. P. (2006a). "Assessment of water resources risk and vulnerability to changing climatic conditions: Dynamic feedback coupling of continuous hydrologic and system dynamics models of the Upper Thames River basin (under preparation)." *Report No. X*, 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.

Senge, P. M. (1990). *The Fifth Discipline: The Art & Practise of the 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*, 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.

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.

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.

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

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

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

Wey, K. (2006). *Temporal disaggregation of daily precipitation data in a changing climate (under preparation)*. Masters Thesis, Department of Civil Engineering, University of Waterloo, Waterloo, 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.

A Appendix

A.1 Intergovernmental Panel on Climate Change Scenarios

The following is taken from IPCC (2001) and represent four main families of climate change scenarios.

> 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.

A.2 Inverse Link Parameter Values

Table 4 shows the parameter values of both seasonal and annual minimum inverse relationships. The parameters correspond to a fit of equation (1) to the flow and precipitation data.

Table 4: Parameters of the rainfall-runoff curves of equation (1)

	Seasonal curves (Fig. 13)			Annual min curves (Fig. 14-16)		
Location	a	b	c	a	b	c
Byron	4.74897	0.0126375	1.52882	2.99871	0.0401771	1.21056
Ingersoll	0.574834	0.00132027	1.59371	16.02	-19.5161	-0.0658829
St. Marys	2.10868	0.00556211	1.48463	49.338	-54.5828	-0.0452138

A.3 Probability Paper Plots in Frequency Analysis

This section directs an interested reader to more details in the drought frequency analysis using the Weibull extreme value distribution. For each location of interest that a probability distribution was fitted to the data (both flow and precipitation), probability paper plots are shown. These plots (see Figure 17) show the goodness of fit of the data to the selected statistical distribution; the scales of the plots are selected in such ways so as to show the fitted distribution as a straight line. The primary reason of doing this is to show the goodness of fit to the extreme values.

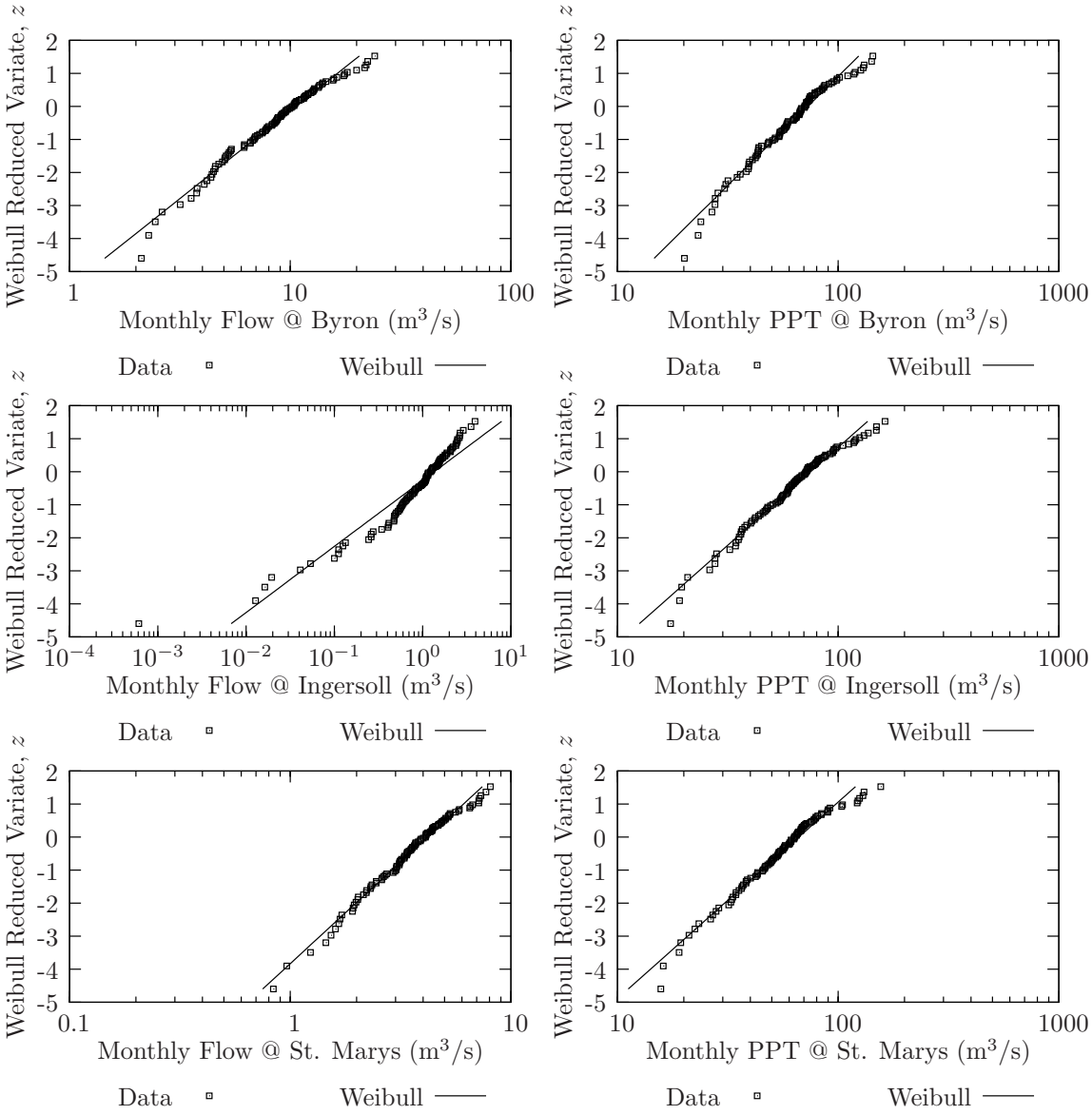

Figure 17: Weibull probability paper plots for historic scenario

A.4 Code Listing

This section presents the complete Java source code of the continuous hydrologic model, together with all algorithms used in the analysis. Anyone wishing to use the model and its components is free to do so. The code is supplied as is, with no guarantees.

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];

// 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();
}

}
```

AnnualSeries2.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();
}

}
```

Clark.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;
}
}
```

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;
```

```
// 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();
//}}}
}

}
```

DataReader.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;
}

}
```

DataWriter.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();
}
}
```

ETZone.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));
}

}
```

HydModel.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;
}
```

```
/**
 *  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]
```

```
 */
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();
}
```

```
/**
 *  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();
}

//}}}

}
```

LinearReservoir.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];
}
}
```

```
/**
 *  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];
```

```
}
}
```

mainCont8Bare.java

```
import java.io.*;

/**
 *  This is a class that executes the combined model
 *
 * @author     Predrag Prodanovic
 * @created    February 12, 2006
 */
public class mainCont8Bare {
/**
 *  The main program for the mainCont8 class; This class is the HydModel
 * by its self, without any links to the SysModel. This version is the
 * bare bones version of the hydrological model
 *
 * @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();

// 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;

// number of 6hr time steps and list of data files used
//{{{
// this is for the historical data between 1964-2002; if using
// historically based ppt, make sure the 29 Feb feature is
// disabled in the HydModel

// when the 1964-2001 data set is used
//int num6hrTimeSteps = 13880 * (24 / userTimeStep);

// when the 1984-2001 data set is used
//int num6hrTimeSteps = 6575 * (24 / userTimeStep);

// the location of the files where the input data is
// this is the actually measured historical data
//String inputDir = "//home//pprodano//code//JavaContinuous//Data//"+
//"//Daily//HistoricalSmall//";
//String outputDir = "//home//pprodano//code//JavaContinuous//Data//"+
//"//Daily//HistoricalSmall//IntermediateFiles//";

// 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 * 5) + 24) * (24 / userTimeStep);

// this is the data for WG Scenarios
String inputDir = "//home//pprodano//code//JavaContinuous//Data//"+
"Daily//WGScenarios//Historic//";
String outputDir = "//home//pprodano//code//JavaContinuous//Data//"+
"Daily//WGScenarios//Historic//IntermediateFiles//";
//}}}

System.out.println("Simulation running ... please wait ...");

// daily output, in units [cms]
//{{{
double jnMitchellSG = 0.0;
double jnMitchellSGBaseflow = 0.0;
double jnAvonSG = 0.0;
double jnDownStreamWildwood = 0.0;
double jnStMarysSG = 0.0;
double jnPloverMillsSG = 0.0;
double jnDownStreamFanshawe = 0.0;
double jnMedwaySG = 0.0;
double jnInnerkipSG = 0.0;
double jnDownStreamPittock = 0.0;
double jnCedarSG = 0.0;
double jnIngersollSG = 0.0;
double jnThamesfordSG = 0.0;
double jnReynoldsSG = 0.0;
double jnWaubunoSG = 0.0;
double jnEalingSG = 0.0;
double jnByronSG = 0.0;
double jnOxbowSG = 0.0;
double jnDingmanSG = 0.0;
//}}}

// number of user time steps in each month
int userTimeStepsInMonth;

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

// DataWriter objects
//{{{
// write the output file header
DataWriter jnOutDaily = new DataWriter(outputDir +
"junctionsDaily.txt");

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

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

// increment the simulated time
currentDate.incrementDateByHours(userTimeStep);

// update the hydrological model
utHyd.update(currentDate);
```

```
// calculates and writes the daily data
//{{{
// this adds up four 6hr intervals
jnMitchellSG = jnMitchellSG + utHyd.getJnMitchellSG();
jnMitchellSGBaseflow = jnMitchellSGBaseflow + utHyd.getJnMitchellSGBaseflow();
jnAvonSG = jnAvonSG + utHyd.getJnAvonSG();
jnDownStreamWildwood = jnDownStreamWildwood + utHyd.getJnDownStreamWildwood();
jnStMarysSG = jnStMarysSG + utHyd.getJnStMarysSG();
jnPloverMillsSG = jnPloverMillsSG + utHyd.getJnPloverMillsSG();
jnDownStreamFanshawe = jnDownStreamFanshawe + utHyd.getJnDownStreamFanshawe();
jnMedwaySG = jnMedwaySG + utHyd.getJnMedwaySG();
jnInnerkipSG = jnInnerkipSG + utHyd.getJnUpStreamPittock();
jnDownStreamPittock = jnDownStreamPittock + utHyd.getJnDownStreamPittock();
jnCedarSG = jnCedarSG + utHyd.getJnCedarSG();
jnIngersollSG = jnIngersollSG + utHyd.getJnIngersollSG();
jnThamesfordSG = jnThamesfordSG + utHyd.getJnThamesfordSG();
jnReynoldsSG = jnReynoldsSG + utHyd.getJnReynoldsSG();
jnWaubunoSG = jnWaubunoSG + utHyd.getJnWaubunoSG();
jnEalingSG = jnEalingSG + utHyd.getJnEalingSG();
jnByronSG = jnByronSG + utHyd.getJnByronSG();
jnOxbowSG = jnOxbowSG + utHyd.getJnOxbowSG();
jnDingmanSG = jnDingmanSG + utHyd.getJnDingmanSG();

if (currentDate.getHour() >= 19) {
// this averages four 6hr intervals into one
// daily value
jnMitchellSG = jnMitchellSG / 4.0;
jnMitchellSGBaseflow = jnMitchellSGBaseflow / 4.0;
jnAvonSG = jnAvonSG / 4.0;
jnDownStreamWildwood = jnDownStreamWildwood / 4.0;
jnStMarysSG = jnStMarysSG / 4.0;
jnPloverMillsSG = jnPloverMillsSG / 4.0;
jnDownStreamFanshawe = jnDownStreamFanshawe / 4.0;
jnMedwaySG = jnMedwaySG / 4.0;
jnInnerkipSG = jnInnerkipSG / 4.0;
jnDownStreamPittock = jnDownStreamPittock / 4.0;
jnCedarSG = jnCedarSG / 4.0;
jnIngersollSG = jnIngersollSG / 4.0;
jnThamesfordSG = jnThamesfordSG / 4.0;
jnReynoldsSG = jnReynoldsSG / 4.0;
jnWaubunoSG = jnWaubunoSG / 4.0;
jnEalingSG = jnEalingSG / 4.0;
jnByronSG = jnByronSG / 4.0;
jnOxbowSG = jnOxbowSG / 4.0;
jnDingmanSG = jnDingmanSG / 4.0;

// outputs the daily value
/*
jnOutDaily.writeData(currentDate.getDate() + ", " +
jnMitchellSG + ", " +
jnMitchellSGBaseflow + ", " +
jnAvonSG + ", " +
jnDownStreamWildwood + ", " +
jnStMarysSG + ", " +
jnPloverMillsSG + ", " +
jnDownStreamFanshawe + ", " +
jnMedwaySG + ", " +
jnInnerkipSG + ", " +
jnDownStreamPittock + ", " +
jnCedarSG + ", " +
jnIngersollSG + ", " +
jnThamesfordSG + ", " +
jnReynoldsSG + ", " +
jnWaubunoSG + ", " +
```

```
jnEalingSG + ", " +
jnByronSG + ", " +
jnOxbowSG + ", " +
jnDingmanSG);
*/

// resets the flow variable
jnMitchellSG = 0.0;
jnMitchellSGBaseflow = 0.0;
jnAvonSG = 0.0;
jnDownStreamWildwood = 0.0;
jnStMarysSG = 0.0;
jnPloverMillsSG = 0.0;
jnDownStreamFanshawe = 0.0;
jnMedwaySG = 0.0;
jnInnerkipSG = 0.0;
jnDownStreamPittock = 0.0;
jnCedarSG = 0.0;
jnIngersollSG = 0.0;
jnThamesfordSG = 0.0;
jnReynoldsSG = 0.0;
jnWaubunoSG = 0.0;
jnEalingSG = 0.0;
jnByronSG = 0.0;
jnOxbowSG = 0.0;
jnDingmanSG = 0.0;

}
//}}}
}

// to close the file
jnOutDaily.closeFile();

// 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);

}
}
```

mainCont8.java

```
import java.io.*;

/**
 *  This is a class that executes the combined model
 *
 * @author     Predrag Prodanovic
 * @created    February 12, 2006
 */
public class mainCont8 {
/**
 *  The main program for the mainCont8 class; This class is the HydModel
 * by its self, without any links to the SysModel. This version contains
 * the classes and methods that analyse the output of the hydrological
 * model.
 *
 * @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();

// 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;

// number of 6hr time steps and list of data files used
//{{{
// this is for the historical data between 1964-2002; if using
// historically based ppt, make sure the 29 Feb feature is
// disabled in the HydModel

// when the 1964-2001 data set is used
//int num6hrTimeSteps = 13880 * (24 / userTimeStep);

// when the 1984-2001 data set is used
//int num6hrTimeSteps = 6575 * (24 / userTimeStep);

// the location of the files where the input data is
// this is the actually measured historical data
//String inputDir = "//home//pprodano//code//JavaContinuous//Data//"+
//"//Daily//HistoricalSmall//";
//String outputDir = "//home//pprodano//code//JavaContinuous//Data//"+
//"//Daily//HistoricalSmall//IntermediateFiles//";

// 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);

// this is the data for WG Scenarios
String inputDir = "//home//pprodano//code//JavaContinuous//Data//"+
"Daily//WGScenarios//Historic//";
String outputDir = "//home//pprodano//code//JavaContinuous//Data//"+
"Daily//WGScenarios//Historic//IntermediateFiles//";
//}}}

System.out.println("Simulation running ... please wait ...");

// daily output, in units [cms]
//{{{
double jnMitchellSG = 0.0;
double jnMitchellSGBaseflow = 0.0;
double jnAvonSG = 0.0;
double jnDownStreamWildwood = 0.0;
double jnStMarysSG = 0.0;
double jnPloverMillsSG = 0.0;
double jnDownStreamFanshawe = 0.0;
double jnMedwaySG = 0.0;
double jnInnerkipSG = 0.0;
double jnDownStreamPittock = 0.0;
double jnCedarSG = 0.0;
double jnIngersollSG = 0.0;
double jnThamesfordSG = 0.0;
double jnReynoldsSG = 0.0;
double jnWaubunoSG = 0.0;
double jnEalingSG = 0.0;
double jnByronSG = 0.0;
double jnOxbowSG = 0.0;
double jnDingmanSG = 0.0;
//}}}

// number of user time steps in each month
int userTimeStepsInMonth;

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

// 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
//{{{
// write the output file header
DataWriter jnOutDaily = new DataWriter(outputDir +
"junctionsDaily.txt");

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

// 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");
//}}}

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

// increment the simulated time
currentDate.incrementDateByHours(userTimeStep);

// update the hydrological model
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);
//}}}

// calculates and writes the daily data
//{{{
// this adds up four 6hr intervals
jnMitchellSG = jnMitchellSG + utHyd.getJnMitchellSG();
jnMitchellSGBaseflow = jnMitchellSGBaseflow + utHyd.getJnMitchellSGBaseflow();
jnAvonSG = jnAvonSG + utHyd.getJnAvonSG();
jnDownStreamWildwood = jnDownStreamWildwood + utHyd.getJnDownStreamWildwood();
jnStMarysSG = jnStMarysSG + utHyd.getJnStMarysSG();
jnPloverMillsSG = jnPloverMillsSG + utHyd.getJnPloverMillsSG();
jnDownStreamFanshawe = jnDownStreamFanshawe + utHyd.getJnDownStreamFanshawe();
jnMedwaySG = jnMedwaySG + utHyd.getJnMedwaySG();
jnInnerkipSG = jnInnerkipSG + utHyd.getJnUpStreamPittock();
jnDownStreamPittock = jnDownStreamPittock + utHyd.getJnDownStreamPittock();
jnCedarSG = jnCedarSG + utHyd.getJnCedarSG();
jnIngersollSG = jnIngersollSG + utHyd.getJnIngersollSG();
jnThamesfordSG = jnThamesfordSG + utHyd.getJnThamesfordSG();
jnReynoldsSG = jnReynoldsSG + utHyd.getJnReynoldsSG();
jnWaubunoSG = jnWaubunoSG + utHyd.getJnWaubunoSG();
jnEalingSG = jnEalingSG + utHyd.getJnEalingSG();
jnByronSG = jnByronSG + utHyd.getJnByronSG();
jnOxbowSG = jnOxbowSG + utHyd.getJnOxbowSG();
jnDingmanSG = jnDingmanSG + utHyd.getJnDingmanSG();

if (currentDate.getHour() >= 19) {
// this averages four 6hr intervals into one
// daily value
jnMitchellSG = jnMitchellSG / 4.0;
jnMitchellSGBaseflow = jnMitchellSGBaseflow / 4.0;
jnAvonSG = jnAvonSG / 4.0;
jnDownStreamWildwood = jnDownStreamWildwood / 4.0;
jnStMarysSG = jnStMarysSG / 4.0;
jnPloverMillsSG = jnPloverMillsSG / 4.0;
jnDownStreamFanshawe = jnDownStreamFanshawe / 4.0;
jnMedwaySG = jnMedwaySG / 4.0;
jnInnerkipSG = jnInnerkipSG / 4.0;
jnDownStreamPittock = jnDownStreamPittock / 4.0;
jnCedarSG = jnCedarSG / 4.0;
jnIngersollSG = jnIngersollSG / 4.0;
jnThamesfordSG = jnThamesfordSG / 4.0;
jnReynoldsSG = jnReynoldsSG / 4.0;
jnWaubunoSG = jnWaubunoSG / 4.0;
jnEalingSG = jnEalingSG / 4.0;
jnByronSG = jnByronSG / 4.0;
jnOxbowSG = jnOxbowSG / 4.0;
jnDingmanSG = jnDingmanSG / 4.0;


// outputs the daily flow values
```

```
// outputs the daily flows for three stations only
flowDailyStMarys.writeData(jnStMarysSG);
flowDailyIngersoll.writeData(jnIngersollSG);
flowDailyByron.writeData(jnByronSG);

// outputs the daily value
/*
jnOutDaily.writeData(currentDate.getDate() + ", " +
jnMitchellSG + ", " +
jnMitchellSGBaseflow + ", " +
jnAvonSG + ", " +
jnDownStreamWildwood + ", " +
jnStMarysSG + ", " +
jnPloverMillsSG + ", " +
jnDownStreamFanshawe + ", " +
jnMedwaySG + ", " +
jnInnerkipSG + ", " +
jnDownStreamPittock + ", " +
jnCedarSG + ", " +
jnIngersollSG + ", " +
jnThamesfordSG + ", " +
jnReynoldsSG + ", " +
jnWaubunoSG + ", " +
jnEalingSG + ", " +
jnByronSG + ", " +
jnOxbowSG + ", " +
jnDingmanSG);
*/

// resets the flow variable
jnMitchellSG = 0.0;
jnMitchellSGBaseflow = 0.0;
jnAvonSG = 0.0;
jnDownStreamWildwood = 0.0;
jnStMarysSG = 0.0;
jnPloverMillsSG = 0.0;
jnDownStreamFanshawe = 0.0;
jnMedwaySG = 0.0;
jnInnerkipSG = 0.0;
jnDownStreamPittock = 0.0;
jnCedarSG = 0.0;
jnIngersollSG = 0.0;
jnThamesfordSG = 0.0;
jnReynoldsSG = 0.0;
jnWaubunoSG = 0.0;
jnEalingSG = 0.0;
jnByronSG = 0.0;
jnOxbowSG = 0.0;
jnDingmanSG = 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
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);

}
}
```

ModifiedPuls.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]));
}

}
```

SoilMoistureAccounting.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];
}
//}}}
}
```

SubBasin.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();
}
//}}}

}
```

Table.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;
}

}
```