THE UNIVERSITY OF WESTERN ONTARIO
DEPARTMENT OF CIVIL AND
ENVIRONMENTAL ENGINEERING

Water Resources Research Report

Inverse Flood 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: 052
Date: November 2006

ISSN: (print) 1913-3200; (online) 1913-3219;
ISBN: (print) 978-0-7714-2634-6; (online) 978-0-7714-2635-3;

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

Inverse Flood 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 6, 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 floods in this case). The critical hydrologic events are converted to their corresponding meteorologic conditions via use of an event based hydrologic model. The local climatic signal is generated by use of a non-parametric weather generator linked to outputs from a global climate 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 (one for each climate scenario) determined.

A scenario selected specifically to study the problem of flooding in the basin showed more frequent occurrence of flooding for nearly all magnitudes of floods. Another scenario, selected for studying droughts depicts a lesser tendency of extreme flooding events. Therefore, ranges of estimates of changes of frequency of occurrence of critical hydrologic events are obtained in response to changing climatic conditions. Based on these estimates, recommendations for changing current basin management guidelines are provided. They are categorized into three distinct categories: (i) regulatory (where a review of rules, regulations and operation of current flood management infrastructure are suggested); (ii) budgetary (where investment in new infrastructure, as well as increased maintenance costs of present and future infrastructure, can lead to a need of having higher operating budgets); and (iii) engineering (recommending a review of current design standards of critical infrastructure).

Keywords: Inverse approach, assessment of climatic change, frequency analysis, weather generator, event 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

Abstract **i**

Acknowledgements **ii**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

1.1 The Problem of Climate Change 3

1.2 Climate Change Research 5

1.3 The Flood Management Process 8

1.4 Outline of the report 11

2 Inverse Approach To Climate Change Modelling **12**

2.1 Methodology 12

2.2 Event Hydrologic Model 17

2.3 Weather Generator Model 19

2.4 Inverse Hydro-Climatic Link 20

3 Case Study **22**

3.1 Watershed Description 23

3.2 Climate Scenario Modelling 26

3.3 Weather Generator Modelling 27

3.4 Hydrologic Modelling 30

3.5 Regional Analysis 34

4 Results and Discussion **35**

4.1 Critical Hydrologic Exposures 35

4.2 Inverse Flood Risk Relationships 38

4.3 Use of the Inverse Approach 38

4.4 Summary of Results 43

4.5 Recommendations for Revision of Guidelines 43

5 Conclusions **49**

References **52**

A Appendix **55**

A.1 Intergovernmental Panel on Climate Change Scenarios 55

A.2 Probability Paper Plots in Flood Frequency Analysis 56

A.3 Flood Flow Frequency Analysis in Hydraulic Modelling 59

A.4 Code Listing 66

DSSDateFormat.java 66

EventSubBasin.java 68

FunctionTable.java 73

Junction.java 75

LatLong.java 77

LinkedList.java 79

MetModel.java 82

PrecipDataSource.java 85

Reach.java 87

Reservoir.java 89

testEventModel.java 91

TimeHistoryFileReader.java 102
TimeHistoryFileWriter.java 106
eventUtils.java 109
mainEvent.java 127
testEventModelHistoric.java 131
DataReader.java 146
DataWriter.java 149
ModelDate.java 151

List of Tables

1 Locations of Generated Meteorological Data 28

2 Critical flood exposures at the St. Marys stream gauge 35

3 Critical flood exposures at the Ealing stream gauge 36

4 Critical flood exposures at the Byron stream gauge 37

5 Flood frequency analysis @ Fanshawe 64

6 Flood frequency analysis @ Medway 64

7 Flood frequency analysis @ Ealing 65

8 Flood frequency analysis @ Byron 65

List of Figures

1 Causal loop of the problem of climate change 3

2 Traditional approach to climate change modelling 7

3 Overall schematic of the inverse approach 15

4 Use of the inverse approach 16

5 Location of the Upper Thames River basin within Ontario 22

6 Map of the Upper Thames River basin 25

7 Hyetographs for event 81 of the historically identical scenario 29

8 Schematic of the event based hydrologic model 31

9 Hydrographs for event 81 of the historically identical scenario 33

10 Event model inverse link for St. Marys ($r^2 = 0.87$) 39

11 Event model inverse link for Ealing ($r^2 = 0.67$) 40

12 Event model inverse link for Byron ($r^2 = 0.78$) 41

13 Flow frequency analysis under different climates 44

14 Flow frequency analysis for St. Marys on historic scenario 56
15 Flow frequency analysis for Ealing on historic scenario 57
16 Flow frequency analysis for Byron on historic scenario 58
17 Frequency analysis for the historically identical WG scenario 59
18 Frequency analysis for the B11 WG scenario 60
19 Frequency analysis for the B21 WG scenario 61
20 Comparison of peak flows computed via Gumbel distribution 62
21 Comparison of peak flows computed via LP3 distribution 63

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.

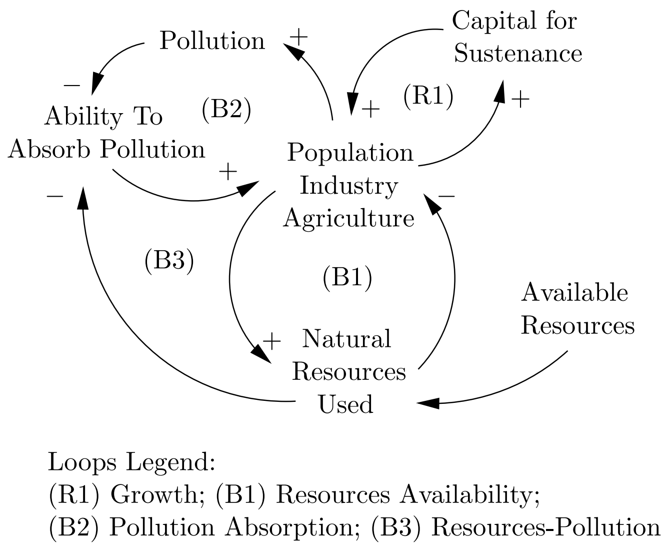
Pollution
Capital for Sustenance
Ability To Absorb Pollution
Population Industry Agriculture
Natural Resources Used
Available Resources
(R1)
(B1)
(B2)
(B3)
+
−
Loops Legend:
(R1) Growth; (B1) Resources Availability;
(B2) Pollution Absorption; (B3) Resources-Pollution

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).

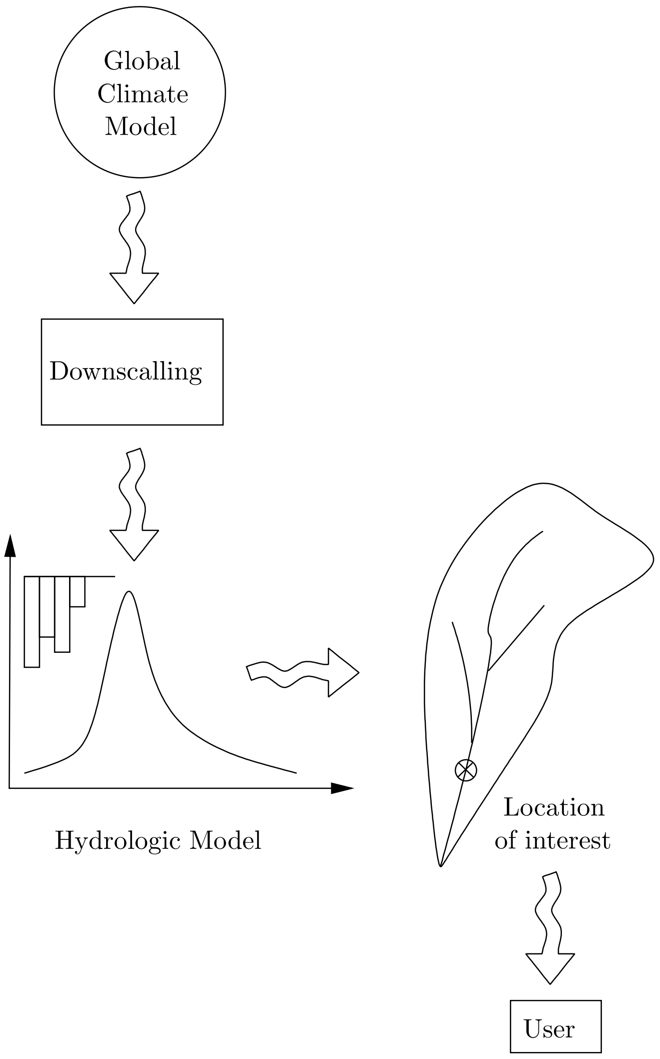

Figure 2: Traditional approach to climate change modelling

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 The Flood Management Process

Changed climatic conditions are expected to bring about changes in hydro-climatic characteristics for many regions of the world. This means that timing, magnitude and frequency of flood flows is expected to change as well, therefore requiring changes in current flood management practises. This section outlines some of the basics of the flood management process.

The process of flood management is defined as a "broad spectrum of water resources activities aimed at reducing potential harmful impact of floods on people, environment and economy of the region" (Simonovic, 2006). In our study area, floods typically occur along rivers and streams; in other areas however, they may occur along lakes, coast lines, alluvial fans, or any other low lying areas.

The flood management process can be divided into the following three phases: pre flood planning, flood emergency management, and post flood recovery (after Simonovic, 2006). The pre flood planning activities are ones where different flood management options are examined in order to reduce harmful effects of floods. Both physical based modelling (such as hydrologic, hydraulic and other engineering analysis) as well as socio-economic analysis (such as population projections, land use planning, etc) are typically employed in this step in order to study long-term impacts of current management practises. The flood emergency management on the other hand, is a set of activities that normally involves flood forecasting, and regular updates of such forecasts. During this phase of flood management,floods are either imminent or are occurring, thereby forcing officials to make urgent decisions regarding changes of operation of flood protection works (such as dams and reservoirs), or to execute emergency evacuation plans. The third phase of flood management, referred to as the post flood recovery, includes such things as flood victim assistance, evaluation of flood damages, and rehabilitation of damaged structures.

Two types of measures are normally employed in the flood management process: structural and non-structural. The former refer to activities that emphasize construction, and include (Simonovic, 2006):

> (i) one or more reservoirs upstream from the protected area to capture the volume of a designed flood and release it at non-damaging rates; (ii) retention (detention) basins to intercept and collect runoff before it reaches the stream channel; (iii) levees or walls to prevent inundation from floods below some specific design flood flow. Additional works may include drainage and pumping facilities for areas that are sealed off from precipitation runoff to the river by the levees; (iv) channel modifications to increase the hydraulic capacity or stability of the river; (v) diversion structures to divert flow during the peak from the protected region; (vi) physical watershed improvement to decrease or delay runoff; and (vii) emergency structural measures for temporary increase of the flood protection capacity.

All such measures impede an excessive amount of flood water from reaching an area needing protection by reducing peak flow, and/or diverting or storing flood water. The latter, on the other hand, concentrate on altering the damage potential of floods with minimal physical intervention through modification of floodplain management practises. Examples of non-structural measures include (Simonovic, 2006):

> (i) zoning (to limit the types of land uses permitted to those which may not be severely damaged by floods); (ii) flood insurance (to recognize the risks of floods and to provide compensation when damages are not avoidable at acceptable cost); (iii) protection of individual properties (waterproofing of the lower floors of existing buildings for example); (iv) flood warning system (to evacuate residents and to move valuables); and (v) disaster preparedness (to prepare the community for effective response to an emergency).

Before assessing suitability (or a need of) a flood management measure a detailed analysis of current physical (hydrologic and hydraulic) conditions normally takes place. This assessment usually concludes with the formulation of discharge-frequency curves, together with stage-discharge and stage-damage curves. These curves describe probability (or risk level) of flood flows, the current channel characteristics, and its associated flood damage, respectively. Although other information is collected as well, these three curves are normally sufficient to initiate the decision making process, and start discussions regarding the viability of different flood management options. For example, successful non-structural measures lower damage amounts for a particular flood level, and thus alter the stage-damage curves, while leaving channel characteristics and frequency of flood flow unchanged. Changed climatic conditions, on the other hand, have the potential to alter hydro-climatic processes within the basin, and therefore alter the frequency of flood flows thus changing discharge-frequency relationships. This implies that safety levels of current flood protection works (such as reservoirs, diversions, dykes, floodwalls, zoning practises, flood insurance guidelines) may become altered as well.

1.4 Outline of the report

The rest of this report focuses on the application of the 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 to the Upper Thames River basin is 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

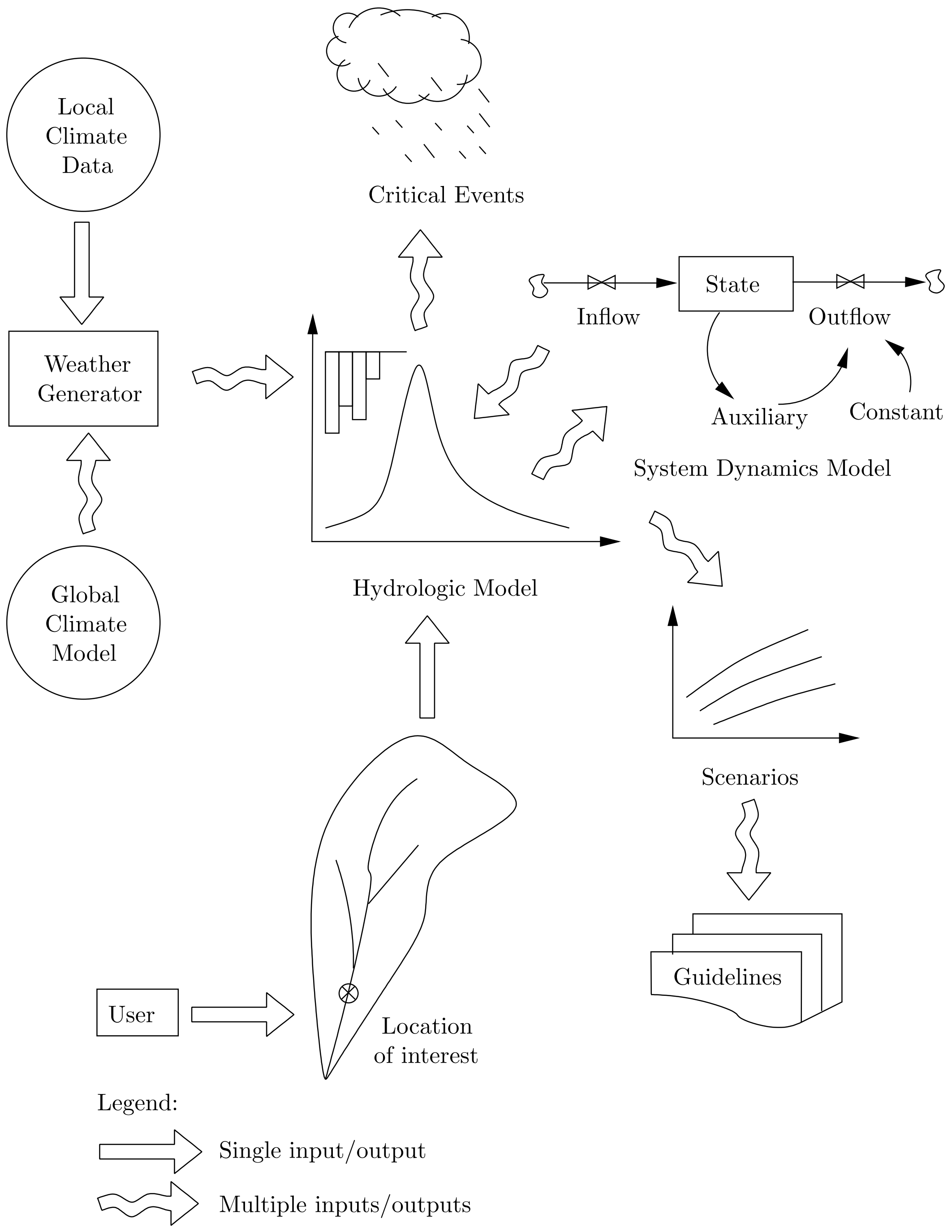

Figure 3: Overall schematic of the inverse approach

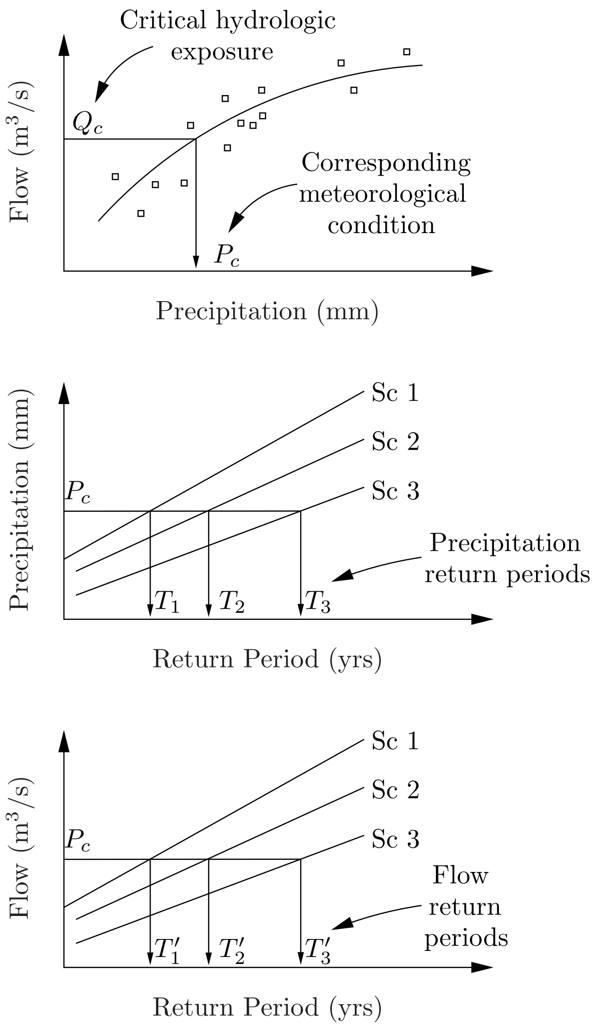

Figure 4: Use of the inverse approach

(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.

When studying the problem of flooding (addressed this report), the following steps are employed: First, the climatic signal is obtained from the weather generator, and then used as an input to an event based hydrologic model. Only those events representing annual maximum daily rainfall are selected, and their corresponding peak flows determined by running the hydrologic model. The frequency analysis is then performed on the peak flows, and its return periods determined. Next, a hydraulic model is used to convert a flood flow into a water elevation for the purpose of floodplain mapping. The area's socio-economic characteristics (such as population density, income distribution, age of housing, etc) are then shown in conjunction with the floodplain map to identify regions most vulnerable to flooding.

2.2 Event Hydrologic Model

The hydrologic model used in this work is described by Cunderlik and Simonovic (2004a, 2006), and so details of it will not be presented here. For the purpose of this report, it will suffice to say that the hydrologic model is a semi-distributed event based rainfall-runoff model based on the computational engine of HEC-HMS (USACE, 2000). As a side note, event based models are those with simplified account of the moisture balance (sometimes representing losses by a simple function or a coefficient), and are therefore suited best for determination of basin response from single storm events (Bedient and Huber, 1988, p. 313). Evapotranspiration is usually not included in the model, as it is deemed negligible during the course of an event (usually no longer than a few days).

The event based hydrologic model used consists of three main components: the basin module, the meteorological module, and the control specification module. The basin module describes the physical characteristics of the watershed (i.e., losses, transformation of excess rainfall to runoff, baseflow recession, flood wave routing and reservoir operation). Appropriate mathematical methods are used to represent each of the above hydrologic processes (see the report by Cunderlik and Simonovic (2004a) for more details). The meteorological module is simply a place for the user to specify the hourly input rainfall (in our case synthetically generated by the weather generator), while the control specification module is used to set the starting (and ending) dates and times for the simulation.

At this point, it is worth mentioning one of the main limitations of all event based hydrologic models—they equate the probability of occurrence of a computed flood flow to the probability of its corresponding input rainfall (Linsley and Franzini, 1979, p. 67). In other words, this means that a T-year return period precipitation corresponds *exactly* to a T-year return period flow, which may not be true at all times. An example can easily be constructed that illustrates this. Suppose that a region receives a large amount of precipitation in a short time interval; also assume that the region was previously very dry, and hence its soils depleted of moisture. Because of this, even though the rainfall is of a large magnitude, the produced flow may not correspond exactly to the return period of the rainfall. In other words, a 100-yr return period precipitation may produce only a 25-yr flood flow.

One way around this limitation is to use a continuous based hydrologic model, and thus take into account a detailed soil moisture balance. Then, a true flood flow frequency analysis can be performed, as the continuous model would produce hydrographs for all days of the year (at a specified time interval). However, continuous hydrologic models are usually not well suited for studies of extreme flood flow. This is because biasing the calibration of a continuous model to extreme flood events lowers its performance in reproducing correctly long term low flows (Bennett, 1998; Cunderlik and Simonovic, 2005). Despite this limitation, event based models are the best tool for simulating flood flows and determining their corresponding return periods.

2.3 Weather Generator Model

The weather generator model developed by Sharif and Burn (2004, 2006a,b), and modified by Wey (2006) is used in this work. A weather generator is a tool that synthetically creates weather and climate information for an area in question. 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). The latter do not make distribution assumptions or 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.

In simple terms, the weather generator employed takes as input historical climate information for a number of weather stations, and generates an arbitrary long period of data synthetically. Sophisticated means are used to shuffle the historical data, and thus generate statistically similar climate information. Of course, the weather generator is not used solely for the replication of historical data; it also contains various perturbation mechanisms that force it to generate climatic information not necessarily observed in the historical record (such as extreme floods and/or droughts). The model is also able to take into account scenarios produced by global climate models, and thus condition the algorithm in that respect. This means that the synthetically generated data is based upon the historically observed record, as well as the information provided by the latest global models.

The original weather generator model developed (Sharif and Burn, 2004, 2006a,b) operates on a daily time step, and produces climatic information for three variables—precipitation, maximum and minimum temperature—for a number of different climate scenarios, for a number of weather stations. For the studies concentrating on high-intensity short-duration events (as is the case for flooding in areas where the basin response is relatively quick), daily time step is too long. To say it another way, for smaller catchments where the basin response is relatively fast, a flood way may come and go in a matter of hours; when shown on a daily time step the hydrographs may not show a significant flood because of temporal averaging. Therefore, for the studies of flooding a temporal disaggregation scheme is developed that takes the synthetically generated daily rainfall, and converts it to hourly intervals (Wey, 2006) by using the locally observed hourly precipitation record and method of fragments. Thus, the consistency of generated results always corresponds to observed conditions.

2.4 Inverse Hydro-Climatic Link

The inverse hydro-climatic link is referred to the functional relationship between critical hydrologic exposures and their 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 appropriate. 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 exposures and their corresponding meteorological conditions is highly dependant on watershed management practises of those responsible for its stewardship. 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 may also alter the basin's flow characteristics. Modifying management practises thereby modifies the inverse hydro-climatic relationship.

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 5. 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.

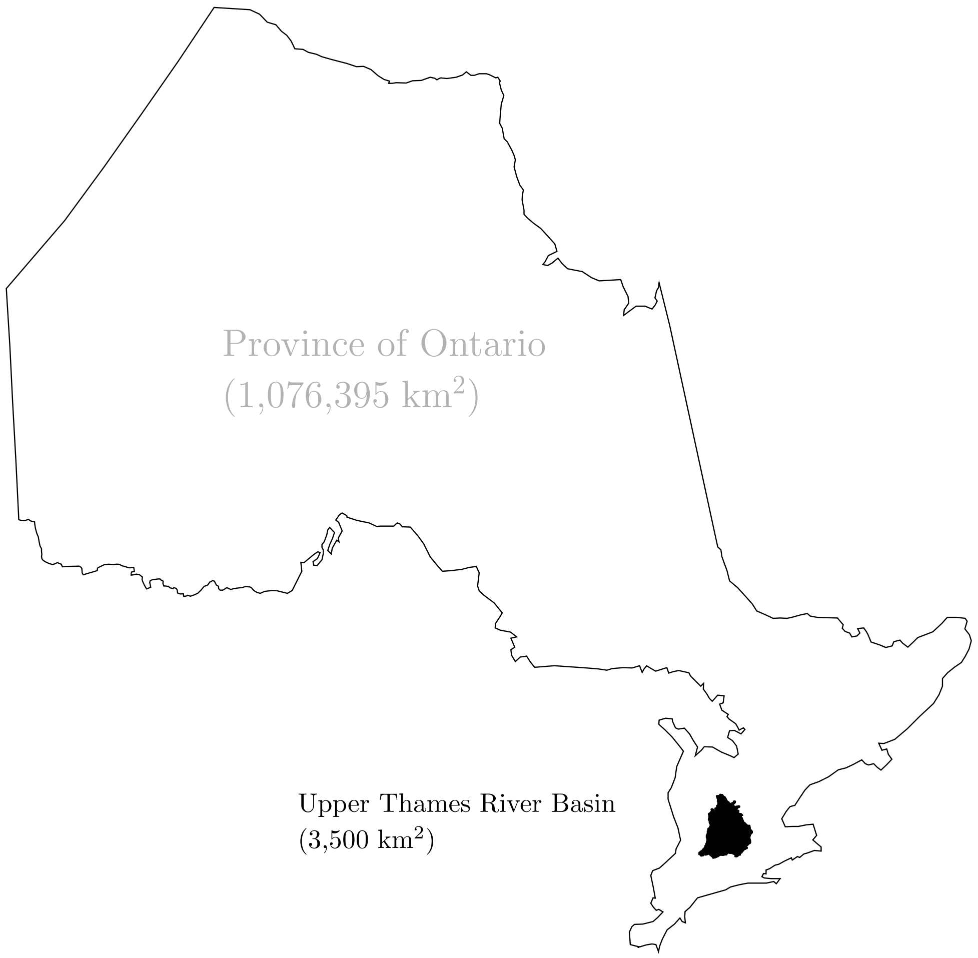

Figure 5: Location of the Upper Thames River basin within Ontario

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 6). 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-

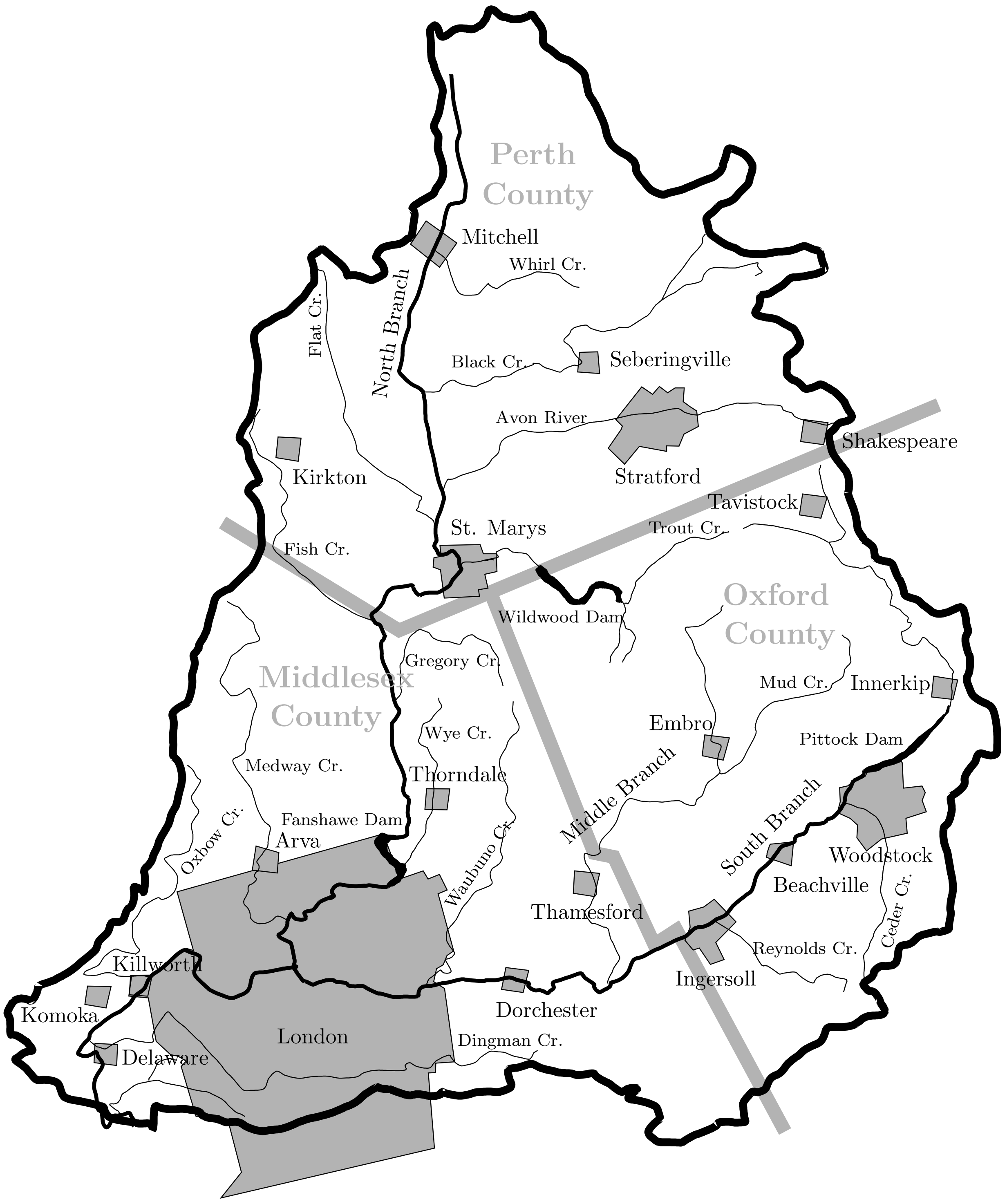

Figure 6: Map of the Upper Thames River basin

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 first version of the weather generator used in this project was developed by Sharif and Burn (2004, 2006a,b). The said weather generator operated on a daily time step, and was thus inadequate for producing short-duration high-intensity storm events that frequently cause summer flooding in the basin. To cope with this problem, a modified version of the weather generator was developed by Wey (2006). This version takes in synthetic daily data generated by the original model, together with historically observed hourly rainfall, and disaggregates it to produce hourly rainfall values. Disaggregated hourly data is used in this work.

The original, as well as the modified versions of the weather generator are able to produce climatic data for a number of locations in the basin. The model can produce synthetic data only for stations that have a long enough historical record to capture essential features of interest (such as extreme storm events, prolonged droughts, etc).

For the Thames River basin, fifteen such stations are selected; their locations and coordinates are shown in Table 1.

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'
Glen Allan	43° 40'	-80° 43'
Ilderton	43° 03'	-81° 25'
London	43° 01'	-81° 09'
St. Thomas	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'

The original version of the weather generator synthetically generates 100 years of data with a daily time step for each climate scenario. This means that each station contains 100 years $\times$ 365 days/year = 36,500 days of data, for each scenario. To disaggregate all 36,500 data points into hourly values would be futile, as a large number of days see little or no precipitation. In the work of Wey (2006), only days which receive more than 25 mm of rainfall are disaggregated into hourly intervals, and are defined as events. Therefore, any given year has a number of events, for which hourly data is available. Rainfall hyetographs for one such event are shown for a number of stations in Figure 7.

Since one of the goals of the study is to perform a frequency analysis, only events producing largest annual rainfall for the entire basin are selected for further analysis.

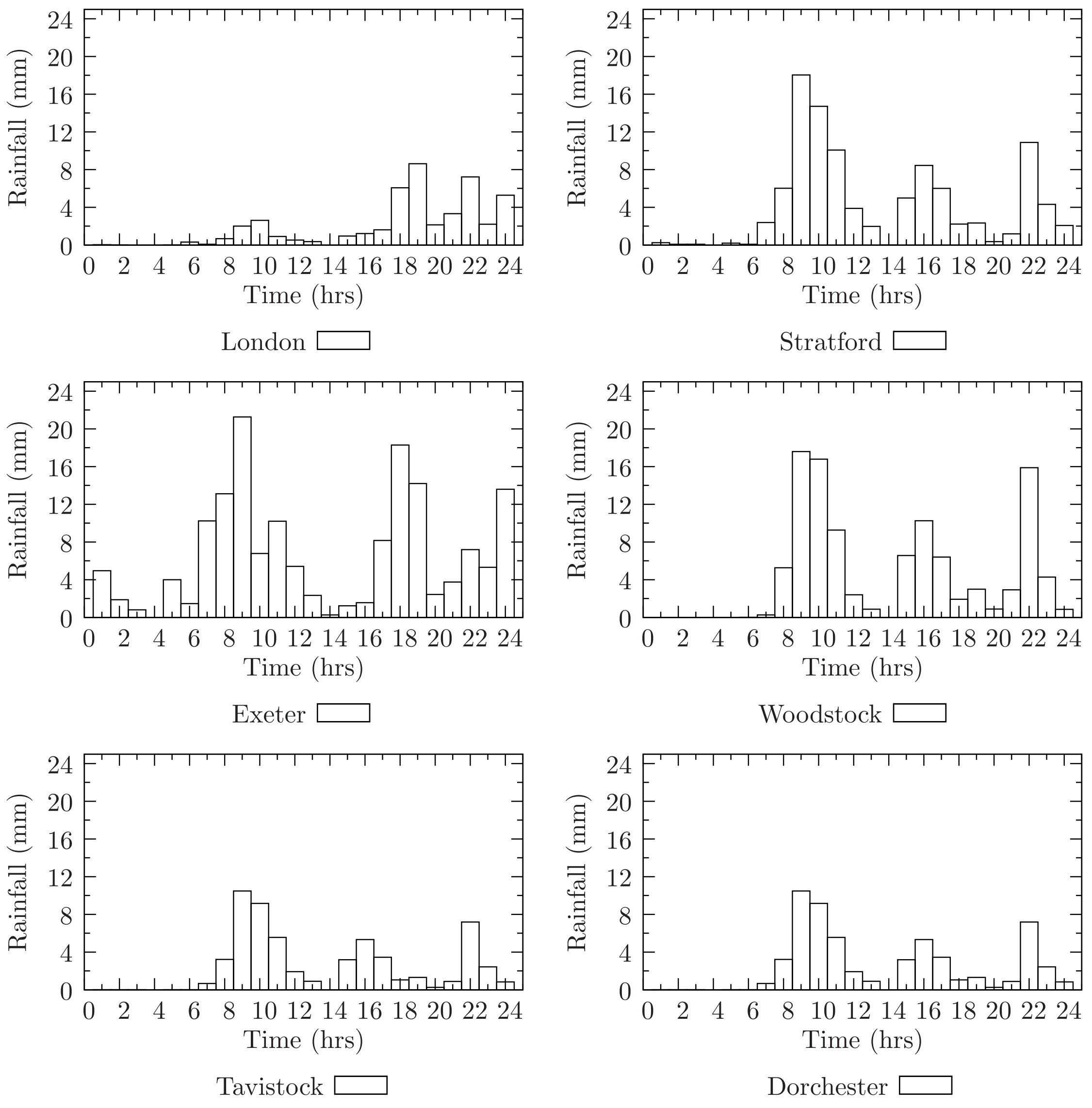

Figure 7: Hyetographs for event 81 of the historically identical scenario

(See Section 2.2 for a discussion of this topic.) Data corresponding to these events are then used as input to the hydrologic model.

3.4 Hydrologic Modelling

An event based hydrologic model (Cunderlik and Simonovic, 2004a, 2006) is used in this report. The original model was created in HEC-HMS version 2.2.2, a program that has since then undergone a major update (USACE, 2005). The requirements of the current project however, demand that a large number of event storms be run with the hydrologic model, so that a flow frequency analysis can be performed. Even if the latest version of the software is used, performing a frequency analysis by running a large number of storms (one hundred for each climate scenario) would be an extremely tedious task. To cope with this, the computational engine of HEC-HMS was reprogrammed in the Java programming language to speed the process of performing a large number of simulations.

The event 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 is shown is Figure 8. The basin representation of the model consists of three components (or methods) describing loss, transform and baseflow components. Since detailed evapotranspiration need not be made in event models, the loss components were modelled via the Initial and Constant method. The transform component is modelled via Clark's Unit Hydrograph, and the baseflow is represented via the Baseflow Recession method. River and reservoir routing were computed via the Modified Puls method, using storage-discharge curves provided by the Upper Thames River Conservation Authority. The interested reader is referred to USACE (2000) and Hoggan (1996) for technical details on these and other methods.

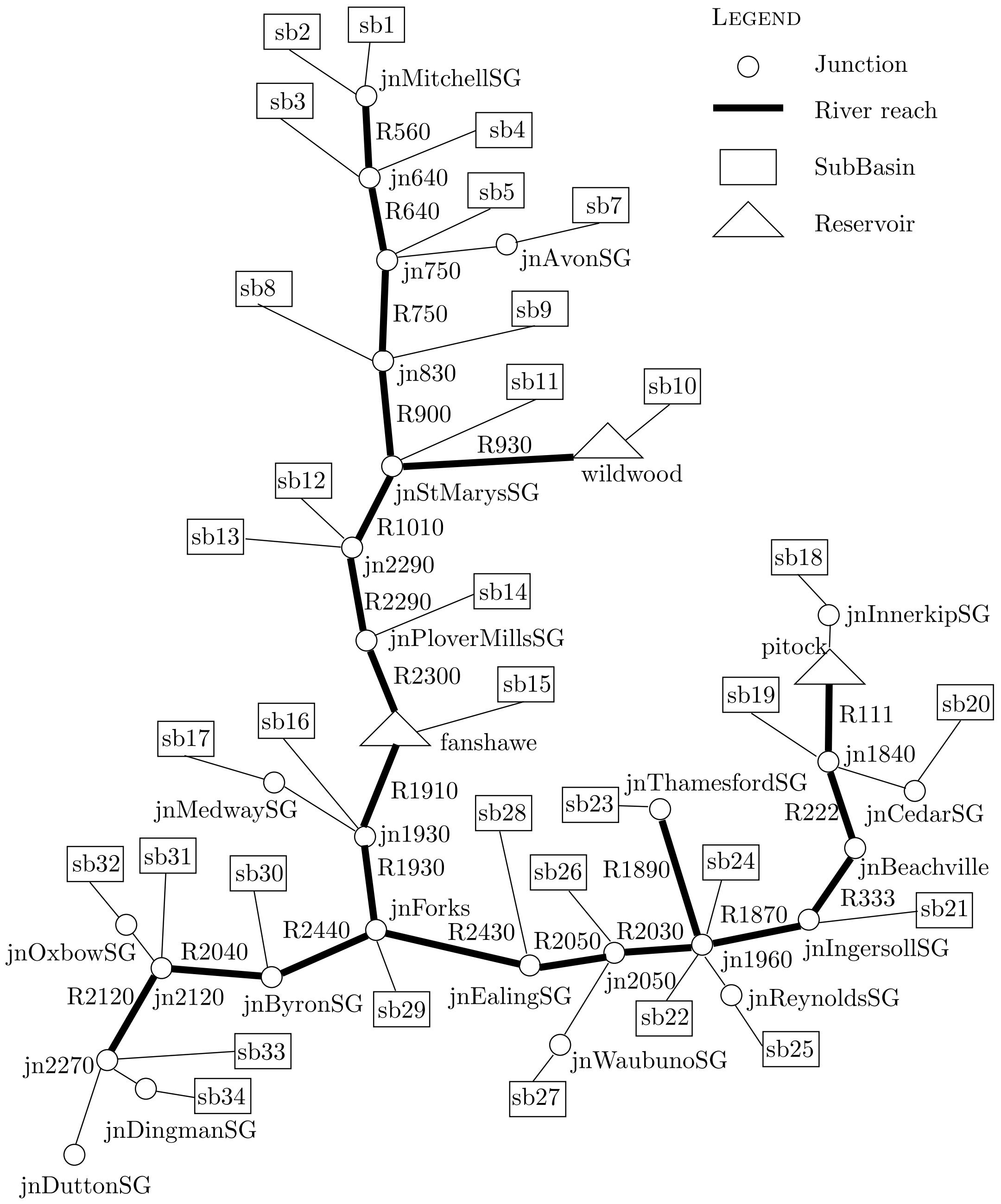

Figure 8: Schematic of the event based hydrologic model

The source code (provided in Section A.4 of the Appendix) implements these methods to the Upper Thames River basin, and provides all relationships and parameters used.

One of the features of the hydrologic model deserves special mention: the reservoir routing. At the time of model development, only the Modified Puls method was available for modelling reservoir routing—this was the only option available in version 2.2.2 of HEC-HMS. It must be acknowledged that in real life reservoirs (especially Fanshawe) may not transform the hydrograph in this way. The releases from the reservoirs thus represent only approximations of management practises actually employed by the officials. 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 reservoir releases. Detailed modelling of 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 and can easily be adopted to modelling customized and 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 the 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. The interpolated rainfall data is then used as input to the hydrologic model, which produces its response in the form of outflow hydrographs. A run of the hydrologic model for one set of inputs is shown in Figure 9, for six locations in the basin.

3.5 Regional Analysis

In order to illustrate the application of the inverse method, three locations are selected in the Upper Thames River basin—St. Marys, Ealing and Byron. The selected locations represent stream gauges in three different branches of the Thames—north, east and west, respectively. Each location has slightly different hydro-climatic characteristics expected to be captured 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 daily precipitation v. peak flow curve be constructed. As part of this analysis, subbasin hyetographs (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 8); the rainfall hyetographs for subbasins 1-10 are aggregated by the weighted area method to arrive at a single rainfall hyetograph corresponding to the stream gauge in question. It is worth pointing out that as the upstream area gets larger (the Byron station 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 hydrological model are obtained, the frequency analysis of both, daily precipitation and peak flow, is undertaken. Since extreme events are of interest, two extreme value statistical distributions are selected: Gumbel and Log Pearson III. These distributions have long been in use in hydrologic analysis. For details on the mathematical procedures, see any text on hydrology (Bedient and

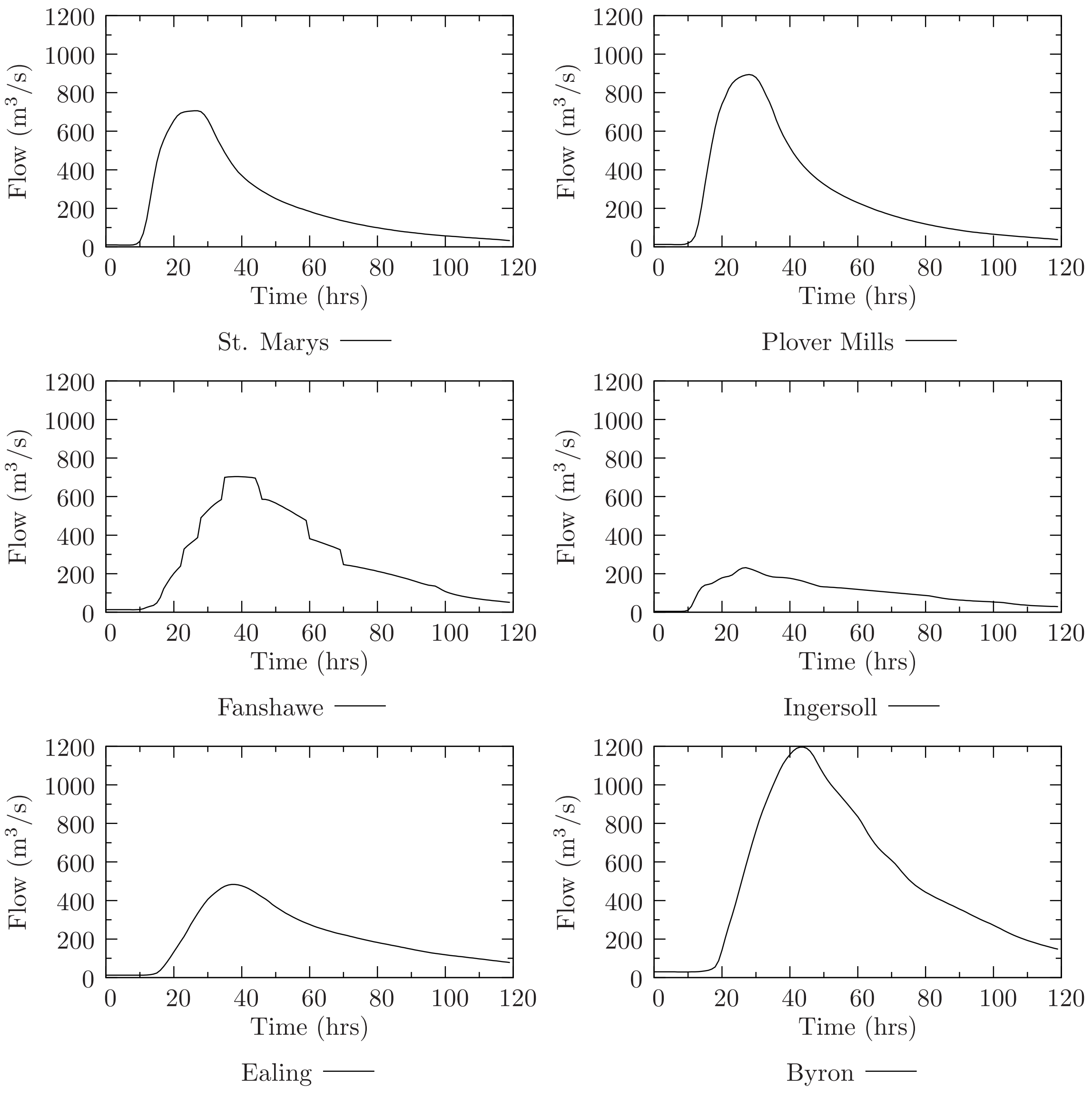

Figure 9: Hydrographs for event 81 of the historically identical scenario

Huber, 1988; Hoggan, 1996; Linsley and Franzini, 1979). The interested reader is also referred to the code listing in Section A.4 of the Appendix for appropriate equations and algorithms.

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 stream gauges in the basin—St. Marys, Ealing and Byron.

4.1 Critical Hydrologic Exposures

The first step in the application of the inverse approach involves identification of the critical flood exposures within the area under consideration. In consultation with the staff from the Upper Thames River Conservation Authority, the information in Tables 2-4 has been assembled. The table entries have been taken from a local flood time operations manual, and show either exposures, or required course of action government (and other) officials need to take. One of the final end goals of this analysis is to show how the frequency of occurrence of such exposures might shift with a changed climatic signal.

Table 2: Critical flood exposures at the St. Marys stream gauge

Peak Flow† (m^3/s)	24-hr Rainfall‡ (mm)	Flood Exposure†
37.5	20	St. Marys Golf Course floods
450	62.9	Top of flood walls; downstream businesses become flooded
500	68.6	Breach of Station Street
630	83.1	South overbank floods near Wellington Street
750	96.6	Property flooding begins on Thames Street downstream of Queen Street
800	102.2	Property flooding begins on St. Maria Street

† Provided by Upper Thames River Conservation Authority staff.
‡ Calculated from the precipitation-discharge curve for St. Marys (see Figure 10).

Table 3: Critical flood exposures at the Ealing stream gauge

Peak Flow[†] (m^3/s)	24-hr Rainfall[‡] (mm)	Flood Exposure[†]
130	37.1	Boating restrictions on river
170	42.9	City on standby to by-pass Vauxhall Pollution Control Plant
225	50.7	City on standby at Chelsea Heights pumping station to remove pumps and disconnect hydro
250	54.3	Low lying green spaces near Adelaide Street flood
280	58.6	City commences patrolling river banks
300	61.5	City on standby to by-pass Pottersburg Pollution Control Plant
340	67.3	City increases river bank patrols; on standby with sandbags to hold down maintenance hole covers and form dykes if necessary
500	90.2	Bankfull stage at York Street
520	93.1	Adelaide Street flooding begins
530	94.5	Front Avenue near Wellington Street inundates
580	101.7	Labatt's Pollution Control Plant sewer surcharges
600	104.6	Property flooding along Keenan Place and southwest of Wellington Street Bridge
660	113.2	Labatt's Plant parking lot floods
700	118.9	Labatt's Plant ammonia room floods
720	121.8	Water overtops Adelaide Street south of bridge
790	131.8	Top of dyke along Nelson Street
900	147.6	Property damage along Thames Street south of York
1000	162.0	Labatt's Brewery main building floods
1100	176.3	Thames Street / York Street intersection inundates

[†] Provided by Upper Thames River Conservation Authority staff.
[‡] Calculated from the precipitation-discharge curve for Ealing (see Figure 11).

Table 4: Critical flood exposures at the Byron stream gauge

Peak Flow[†] (m^3/s)	24-hr Rainfall[‡] (mm)	Flood Exposure[†]
285	33.7	Boating restrictions on the Thames
350	37.8	Flooding of grazing areas at Dellaware Flats
460	44.7	Water overtops banks at the forks
565	51.4	City commences patrolling river banks
600	53.6	Greenway park near Lombardo Bridge begins to flood at shallow depths
840	68.7	Top of dyke—Old Bridge Road, Byron
970	76.9	Greenway park near sewer treatment plant flooded
990	78.2	Top of London's dykes
1080	83.8	Flooding of homes at end of Evergreen and Riverside Avenues near Wharncliffe Bridge
1260	95.2	Pumping station near Springbank park flooded
1300	97.7	Flooding of Wonderland Gardens and adjacent property
1420	105.3	Riverside Drive / Wonderland Road intersection inundated
1640	119.2	Sewer treatment plant building floods
1700	123.0	Property flooding on east side of Wonderland Road north of Riverside

[†] Provided by Upper Thames River Conservation Authority staff.

[‡] Calculated from the precipitation-discharge curve for Byron (see Figure 12).

4.2 Inverse Flood Risk Relationships

The next few steps in the application of the inverse approach are combined here under one section to illustrate the approach in a more efficient manner. The events obtained from the hourly version of the weather generator are used as inputs to the hydrologic model, as well as appropriately aggregated for further analysis (see Section 3.3). The relationship between daily rainfall amount (in mm) and peak flow (in m^3/s) is constructed for each location of interest (top plots in the Figures 10-12), together with appropriate rainfall frequency analysis (middle plots). The outputs of flow frequency analysis are shown in the bottom plots of the figures, for comparison.

For the purpose of this analysis, the probability of daily rainfall and peak flow have been obtained using the Gumbel distribution. This distribution is selected because of its applicability in analysis of extreme events. As a side note, the Log Pearson III distribution is usually employed single handed for the frequency analysis of extreme flows only (see Section A.3 of the Appendix).

4.3 Use of the Inverse Approach

In order to illustrate the application of the inverse approach, we start by identifying critical hydrologic exposures of interest. Suppose that we are interested in studying the frequency of occurrence of rainfall events that have the potential to overtop dykes in the City of London. Consulting Table 4, we can readily look up that if the Thames River at Byron reaches a peak flow of 990 m^3/s, the city's dykes may start to overflow. By looking at the inverse link relationship for this gauge in Figure 12, we can find the corresponding daily amount of precipitation (aggregated over the upstream of the basin) that causes such an event; in this case, it is 78.2 mm. (These daily aggregated rainfall totals have also been placed in the tables together with their corresponding

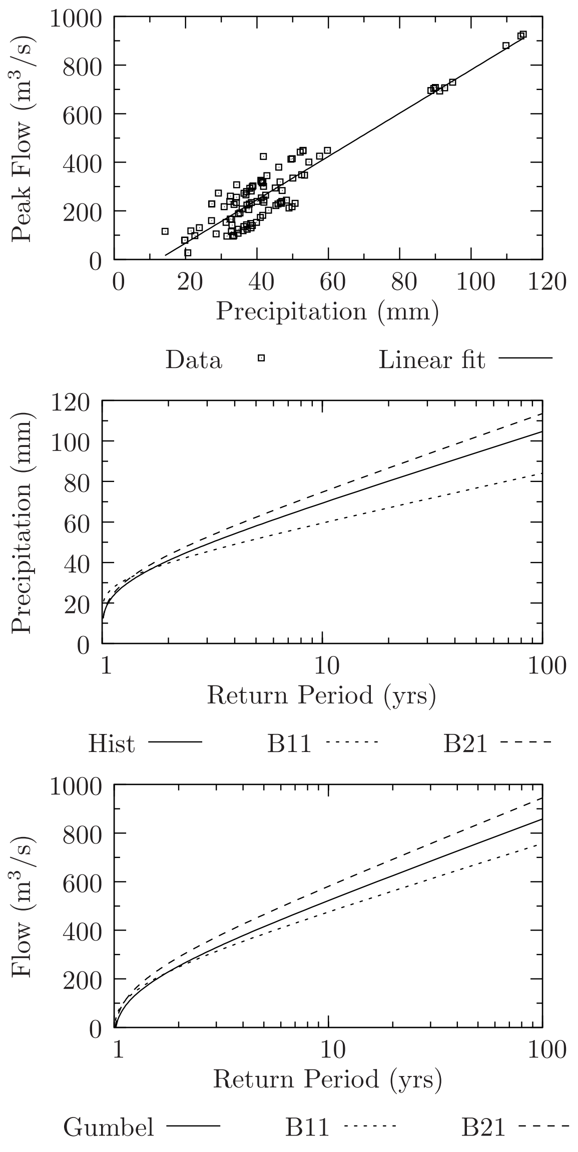

Figure 10: Event model inverse link for St. Marys ($r^2 = 0.87$)

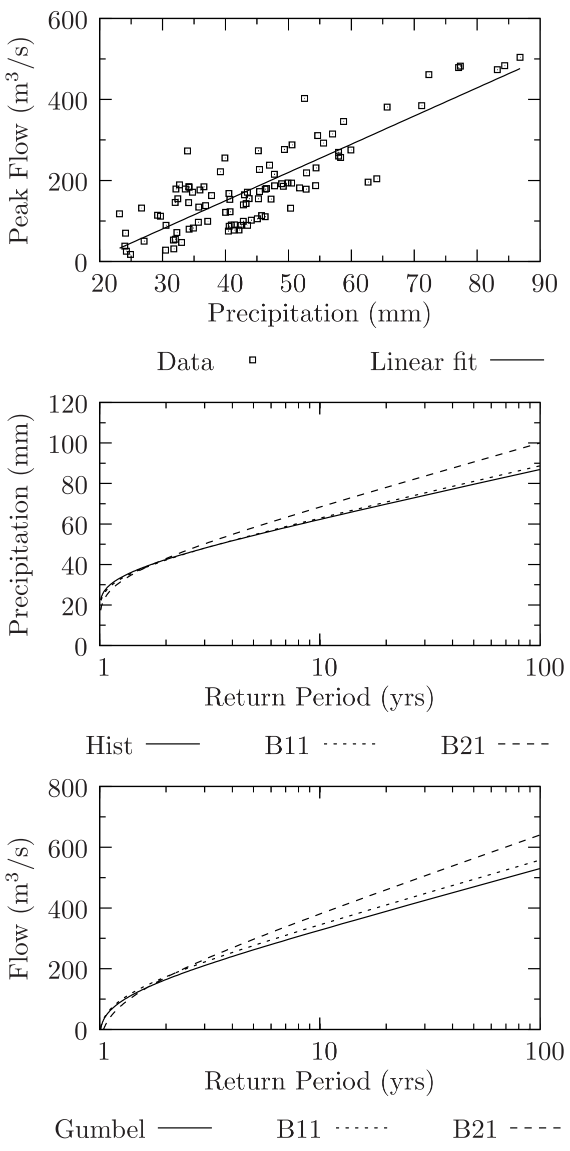

Figure 11: Event model inverse link for Ealing ($r^2 = 0.67$)

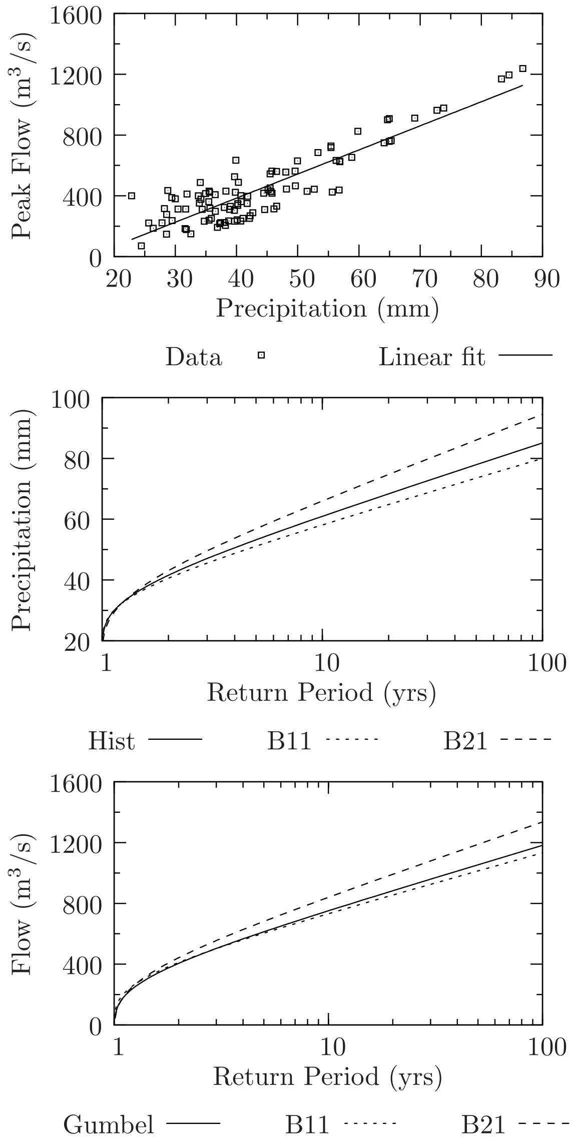

Figure 12: Event model inverse link for Byron ($r^2 = 0.78$)

critical exposure levels, for convenience.)

The next step involves use of the precipitation frequency analysis curves, and looking up the corresponding *return periods of the rainfall*, for each of the three climate scenarios (middle plot in Figure 12). The base case (or the historically identical climate scenario) shows that dykes in the City of London will overtop with a return period of about 50 years; for scenarios B21 and B11 the return periods change to 25 and 100 years, respectively. A return period of T yrs means that an event equalling to or exceeding the said value has a chance of occurrence of 1 in T, any given year. In our example, the flood overtopping rainfall event has (under the base case) a chance of occurrence of 1 in 50 any given year; under the changed climate, this same event may occur more frequently with the B21 scenario (with a chance of 1 in 25 any given year), or less frequently (with a chance of 1 in 100 any given year with the B11 scenario).

As an alternative, the same flood exposure of 990 m^3/s can also be used directly with the flood flow frequency curves (bottom plot in Figure 12). Using the same value of the flood flow, we can obtain *flood return periods* for all climate scenarios. Under the historic case, the flood has a return period of 30 years, while under the changed climate flood may occur more frequently (with a return period of 19 years for a B21 scenario) or less frequently (with a return period of 50 years for the B11 scenario).

It is important to mention that differences between the return periods of rainfall and the return periods of peak flow are present because the precipitation v. peak flow relationship is fitted to data consisting of a large amount of scatter (see top plots of Figures 10-12). A possible reason for such scatter lies in temporal distribution of precipitation events. This is simply another way of saying that not all storms are identically distributed in time (i.e., they all don't peak at the same time). Of course, different temporal distribution of rainfall produces different basin response. But, the inverse hydro-climatic link that uses daily precipitation in its relationship, may miss this feature as it aggregates hourly information into daily values. It is very easy to think of an example of two storms having identical daily totals, but different temporal distribution of rainfall, thus producing different basin responses. This may lead to different peak flows for the same daily total precipitation, thus contributing to the scatter in the data. Therefore, the user is provided with the return periods of rainfall and flood flow, thus having an option to chose which set of results suits him/her best, despite the fact that slightly different results shall be obtained. However, this does not change the consistency in the underlying result: that hazards may become more frequent as a result of the changed climatic signal.

4.4 Summary of Results

In order to better summarize the results of the flood flow frequency analysis for regulatory return periods (those extending up to 500 yrs), under all climates, for two statistical distributions, for all three locations of interest, Figure 13 has been prepared. It is interesting to note that the Log Pearson III statistical distribution (one that nearly every water management agency recommends) tends to consistently estimate higher extreme values than other distributions. (The curious reader is referred to Figures 14-16 of Appendix A.2 for probability paper plots. These plots can give an estimate of goodness of fit of each distribution, particularly for the extreme cases.)

4.5 Recommendations for Revision of Guidelines

For the Upper Thames River basin specifically, altered hydrologic conditions of the basin as a result of climate change imply that flood flow frequency becomes altered

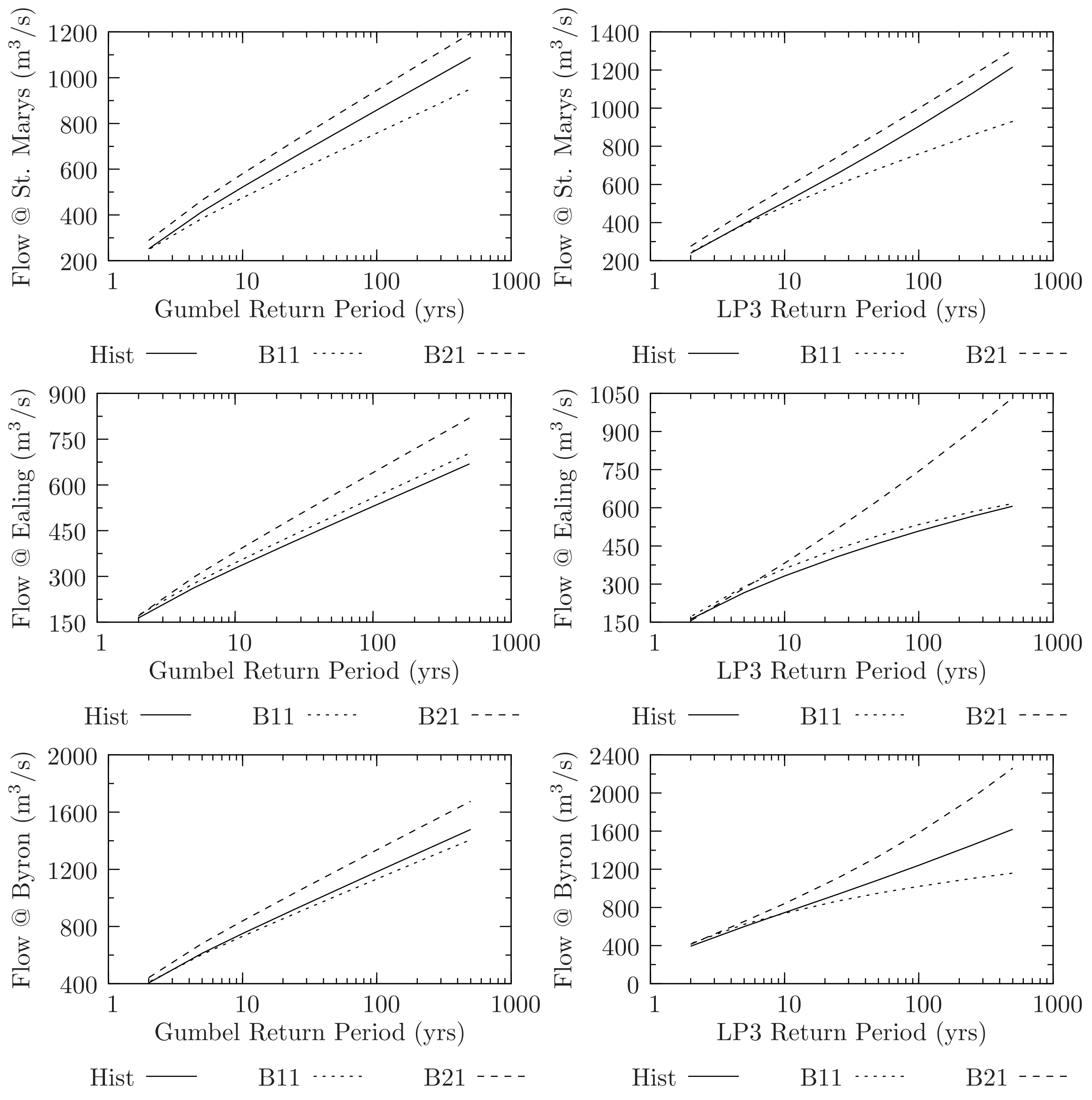

Figure 13: Flow frequency analysis under different climates

as well (see Figure 13), especially for the B21 scenario[4]. This implies that some of the current guidelines and management practises could be revised accordingly. For example, the Upper Thames River Conservation Authority, the government body responsible for establishing and mapping basin's floodplains, may need to revise and update their floodplain maps in light of the changed climatic signal. This work is the subject of a study by Mortsch (2006), which uses flood flow frequency analysis results presented in this report.

Further impacts of climatic change in the basin (especially for the B21 scenario) simply imply that flooding will occur more frequently in the future, regardless of the magnitude of floods. For example, this means that property flooding in the town of St. Marys shall occur more frequently on average than it did in the past. This is a troubling fact, especially since the town currently experiences regular flooding resulting in property damage (see Table 2). The story is no different for the city of London, which if scenario B21 plays out, shall experience flooding more frequently (see Tables 3-4). In particular, those residing near Thames River in London on Adelaide and Nelson Streets, Keenan Place and Evergreen and Riverside Avenues are expected to see increased flood damages over the long term.

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

[4]The discussion in this section focuses on impacts of scenario B21, as this is the scenario that was created specifically for assessing flood related impacts as a result of changed climatic conditions.

management works in the basin, which will undoubtedly result in increased budgets for flood management operations.

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

The following presents a summary of issues regarding possible changes of flood management guidelines as a result of climatic change in the Upper Thames River basin. Some of the issues included in these recommendations are taken from comments and suggestions provided by basin's stakeholders during a stakeholders meeting (held on November 22, 2005) organized by team members of this project. The changes are categorized in the following three categories:

1. *Regulatory.* Changed hydro-climatic characteristics imply that rules and regulations currently employed by the Upper Thames River Conservation Authority may need revision. This recommendation is supported by the analysis showing that floods of all magnitudes will increase in frequency of occurrence (i.e., they

will occur more often). Possible revisions of existing regulation include those pertaining to: (i) use of river and its adjacent park land for recreation (including boating, camping, swimming, fishing, jogging, etc.); (ii) removal of pumping stations near area rivers during the course of a flood (see Tables 3 and 4); (iii) patrolling of river banks during periods of high water levels and monitoring performance of critical infrastructure (such as roads, bridges, culverts, drains, sewer systems); (iv) issuing of permits for floodplain development (including construction of roads, buildings, bridges, culverts, drains, sewer systems, etc); (v) revision of reservoir operating procedures (including smaller dams, weirs, outflows, etc.).

2. *Budgetary.* This set of guidelines look at possible changes in allocation of budgets for safe operation of existing flood management infrastructure, as well as budgetary constraints to be imposed by investment needed for future infrastructure still in the planning stages. If floods occur more frequently (as scenario B21 suggests), so should maintenance, retrofitting and upgrading of existing flood management infrastructure. For example, the overtopping of dykes in London is expected to occur more often than in the past, implying that dykes will hold higher water levels more often. As this happens, development of cracks, crevices and other imperfections will intensify, thus requiring increasing maintenance levels and leading to higher costs. An evaluation of current structural and non-structural measures used to reduce flood damage (such as infrastructure like reservoirs, dykes, floodwalls or implementation of land use zoning practises, flood warning systems, waterproofing, etc.) will need to take place. As flood frequency increases basin wide, so do costs allocated with maintenance of flood protection measures. Next to changed maintenance budgets of

existing infrastructure, a need may arise for investment of additional infrastructure that may need to come on line in the future to address changed climatic conditions. As this new infrastructure comes on line, existing rules that determine budgets typically allocated to maintenance may not be appropriate and may need further consideration.

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

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

5 Conclusions

The main objective of this project is to study risk and vulnerability to changing climatic conditions. In this report we focus on flooding. (Other reports in this series address the problem of droughts 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 hydro-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 critical hydrologic hazards (and in this case, floods in particular). This information is critical to water resources managers, engineers, planners and other government and private sector officials, as it gives them insight of what the changed climate may bring.

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 are ones 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 overtopping of dykes), 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, but emphasis should be placed on scenario B21. This is the scenario that had been selected specifically to study the impact of flooding that a changed climatic conditions may bring. Exactly which climate scenario turns out to the be the case is irrelevant; the emphasis should be on preparing management alternatives and mitigation measures that address a range of possible outcomes. With this information, it is believed, better and more flexible management practises can emerge that should ultimately minimize the regions risk and vulnerability.

In particular, recommendations for either revising existing or setting new flood management guidelines have been outlined. They are categorized into three distinct categories: (i) regulatory (where review of rules, regulations and operation of current flood management infrastructure are suggested); (ii) budgetary (where investment in new infrastructure, as well as increased maintenance costs of present and future infrastructure, can lead to a need of having higher operating budgets); and (iii) engineering (recommending a review of current design standards of critical infrastructure).

The impact of the changed hydro-climatic conditions on current rules and regulations shall be multi-fold. First, the Upper Thames River Conservation Authority should seriously consider reviewing its operating guidelines that are sensitive to changes in flood magnitude and frequency—particularly considering the operation of its three reservoirs, use of river and its surrounding park lands, patrolling and monitoring critical infrastructure during periods of high flow, as well as its guidelines of allowing permits for floodplain development. Secondly, increasing magnitudes and frequency of flood flows are expected to have an impact on operating budgets authorities must secure for safe operation and maintenance of its existing flood management works (including both structural and non-structural measures). Thirdly, a recommendation for a full review of engineering design standards is suggested, as different hydro-climatic conditions will inevitable impose different loading on critical infrastructure (such as roads, bridges, sewer systems, water and sewerage treatment plants, etc.).

The guidelines presented in this report are based on the knowledge and insight obtained by simulating the rainfall-runoff process with an event 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 realms. A different kind of model is needed for this, as water resources management process inevitably involves not only processes that are physical (such as the process of flooding), but those that are socio-economic as well (such as land use planning and development, rural and urban business activity, population, etc.). This kind of model should show the interaction between the physical and socio-economic domains, and thus show both physical and socio-economic impacts as they evolve through time. It should also able to test different management options (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. 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 Jersey.

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

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

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. (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*, Department of Civil and Environmental Engineering, 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.

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, UK.

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

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

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.

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

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, WA.

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." *Report No. X (under preparation)*, Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

Prodanovic, P. and Simonovic, S. P. (2006b). "Systems approach to assessment of climatic change in small river basins." *UNESCO 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, NY.

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

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

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

Simonovic, S. P. (2006). *Natural Disasters: Mitigation, Modelling and Assessment Course Notes.* Department of Civil and Environmental Engineering, The University of Western Ontario, London, Ontario, Canada.

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

USACE (2005). *Hydrologic Modelling System HEC–HMS, User's Manual.* United States Army Corps of Engineers, Davis, CA.

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 Probability Paper Plots in Flood Frequency Analysis

This section directs an interested reader to more details in the flood flow frequency analysis. For each location of interest that a probability distribution was fitted to the data, probability paper plots are shown. These plots 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. The Gumbel (top left) and the Log Pearson III (bottom left) distributions are shown in this manner, together with their corresponding plots of flow v. return period. It is interesting to see how a different distribution has an effect of estimating extreme flows.

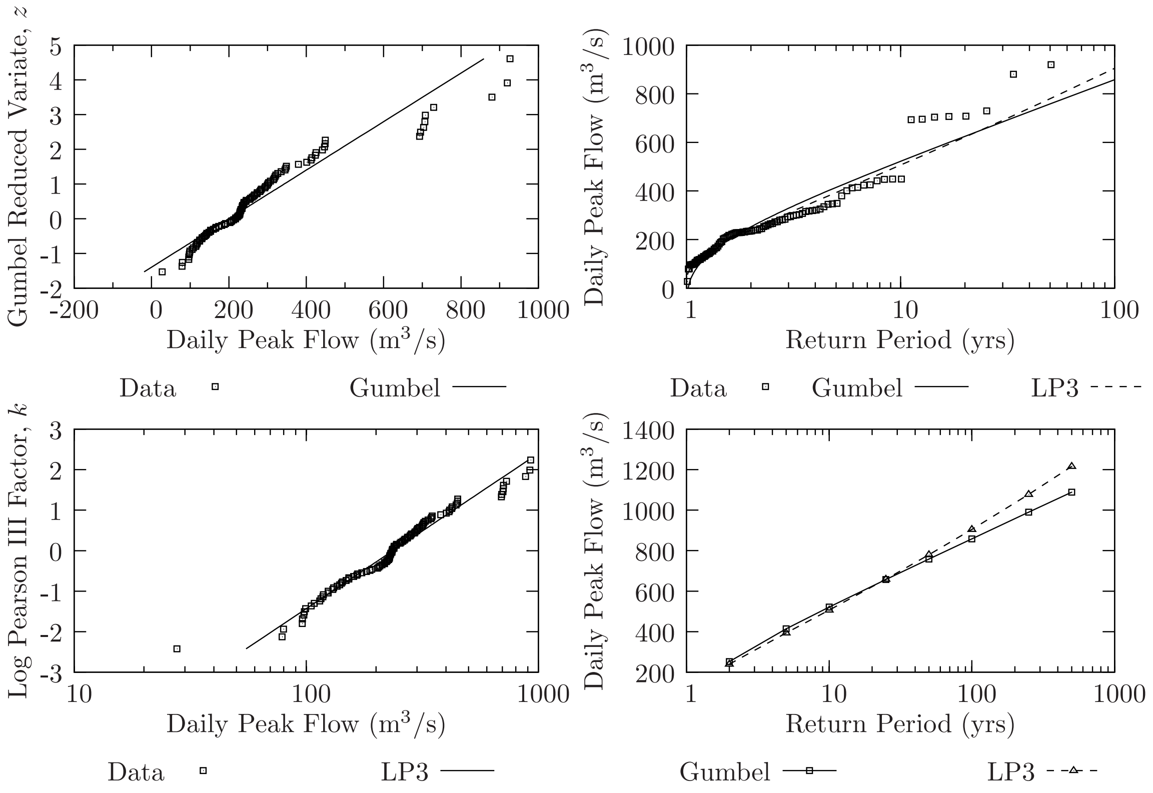

Figure 14: Flow frequency analysis for St. Marys on historic scenario

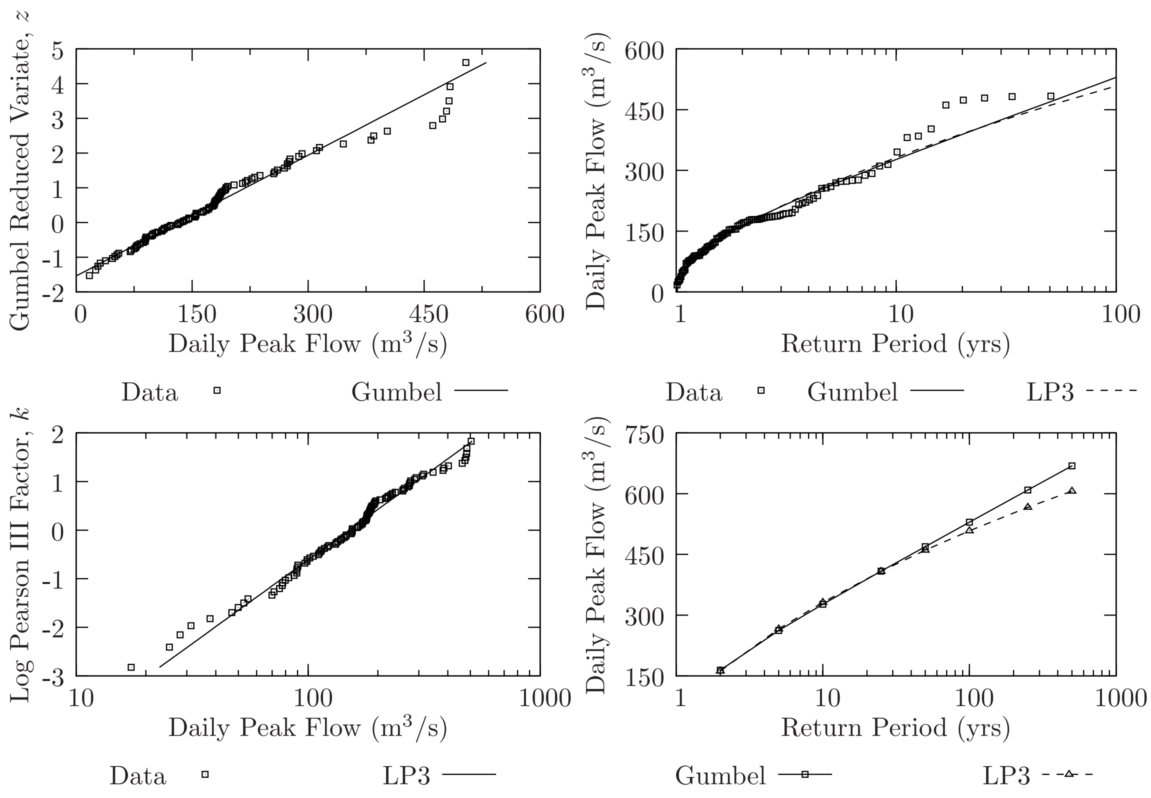

Figure 15: Flow frequency analysis for Ealing on historic scenario

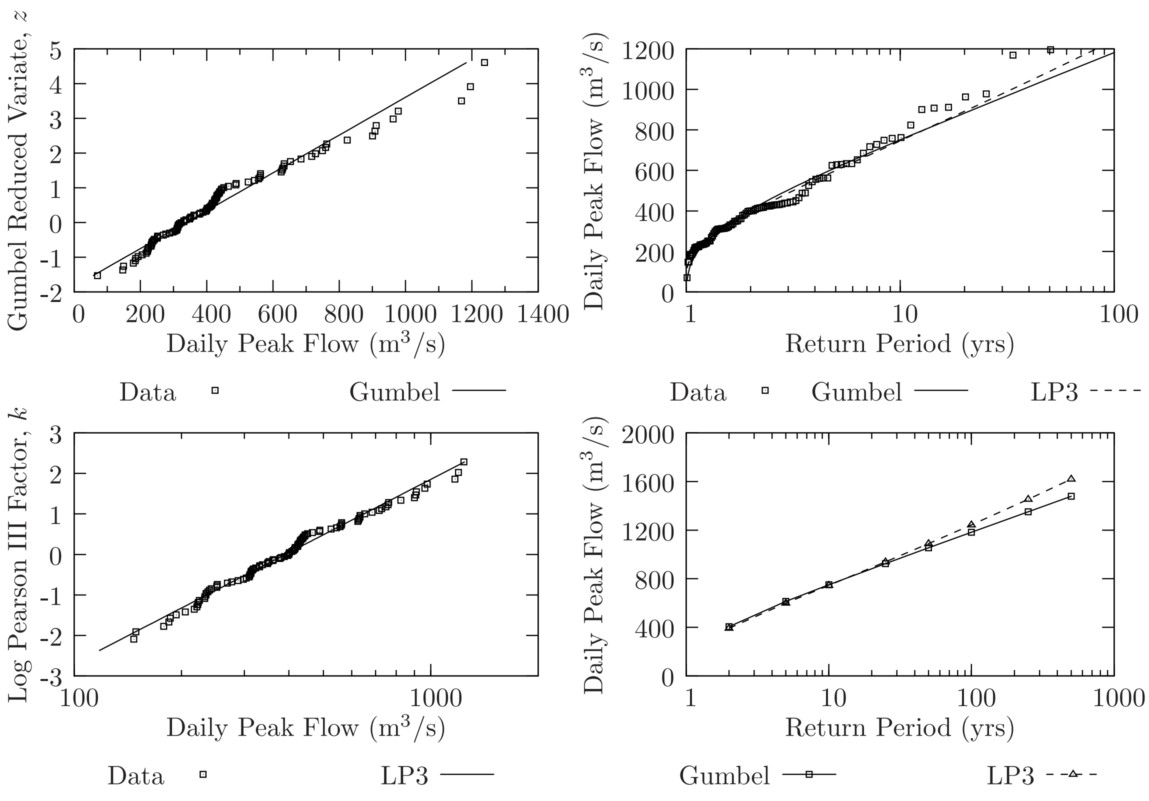

Figure 16: Flow frequency analysis for Byron on historic scenario

A.3 Flood Flow Frequency Analysis in Hydraulic Modelling

This sub section shows the results of the flood flow frequency analysis. This data has been used in hydraulic modelling to determine the extent of the floodplain in the city of London for three climate scenarios. The subsequent plots and graphs show the results for the stream gauges at Fanshawe Dam, Medway Creek, Ealing, and Byron.

All plots show the comparisons of flood flow frequency analysis for three climate scenarios—B11, B21 and Historical. It is interesting to note that large differences are observed simply by altering the extreme value distribution (i.e., Gumbel and Log Pearson III distributions sometime produce markedly different results). This is shown here to emphasize that absolute values produced by the simulation should be regarded with less confidence than differences between the scenarios. In other words, differences between the climate scenarios (for each particular statistical distribution) should be seen to have more importance than the absolute values of any any given scenario.

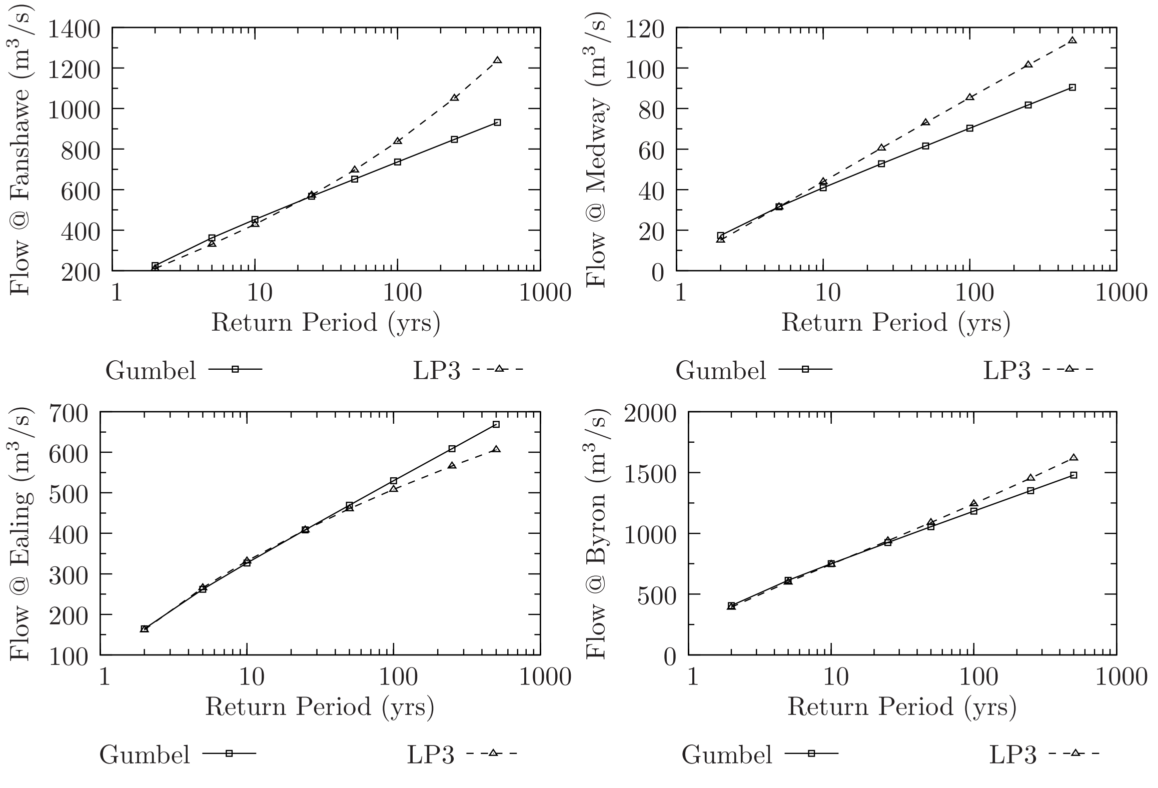

Figure 17: Frequency analysis for the historically identical WG scenario

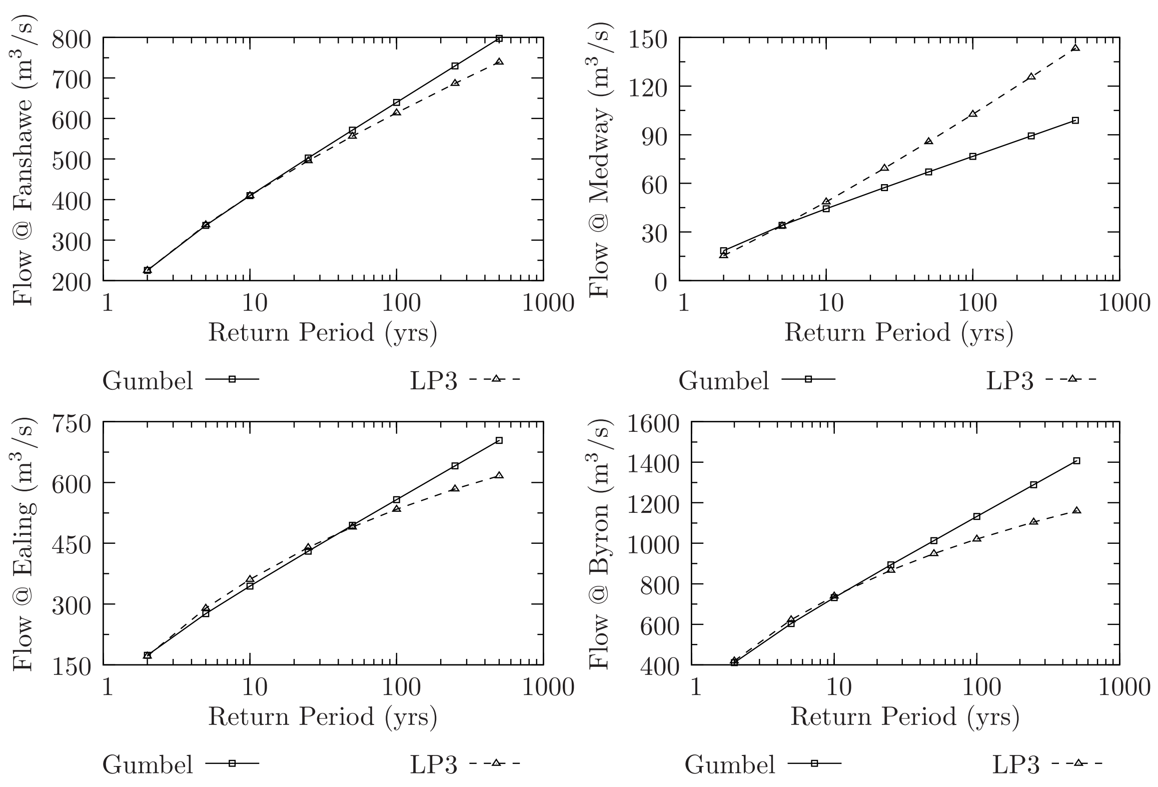

Figure 18: Frequency analysis for the B11 WG scenario

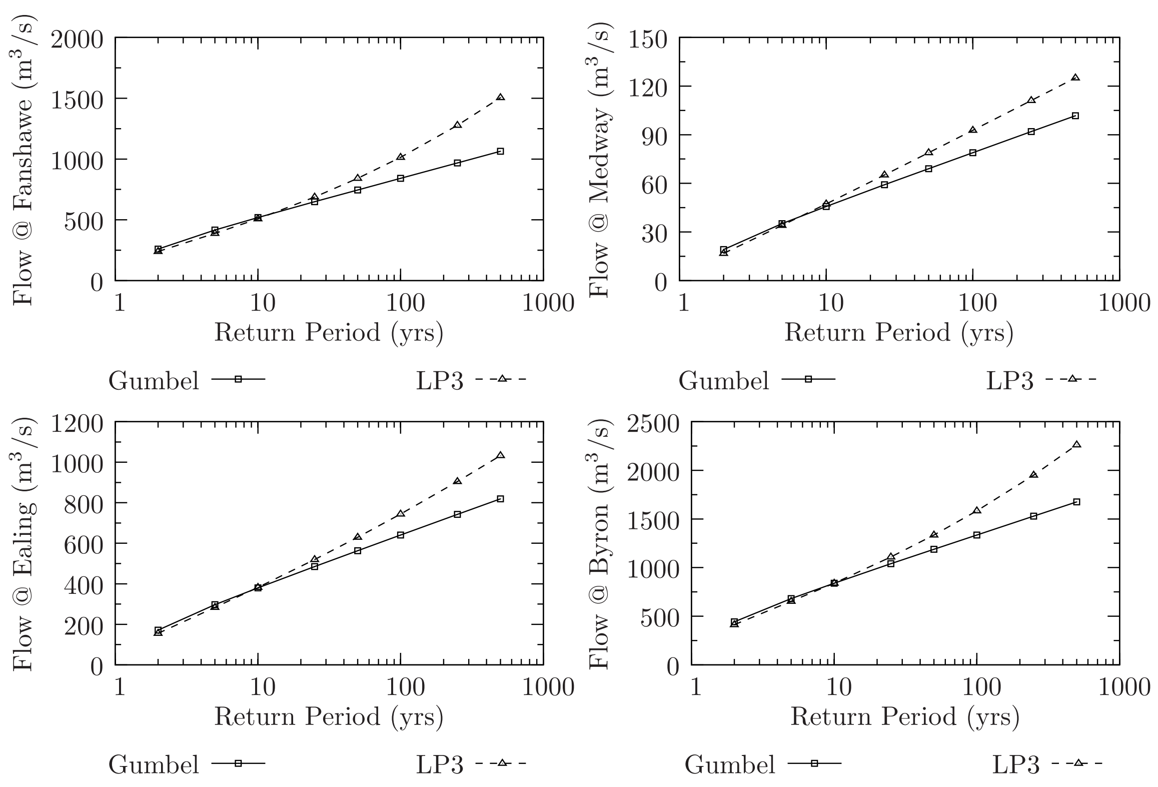

Figure 19: Frequency analysis for the B21 WG scenario

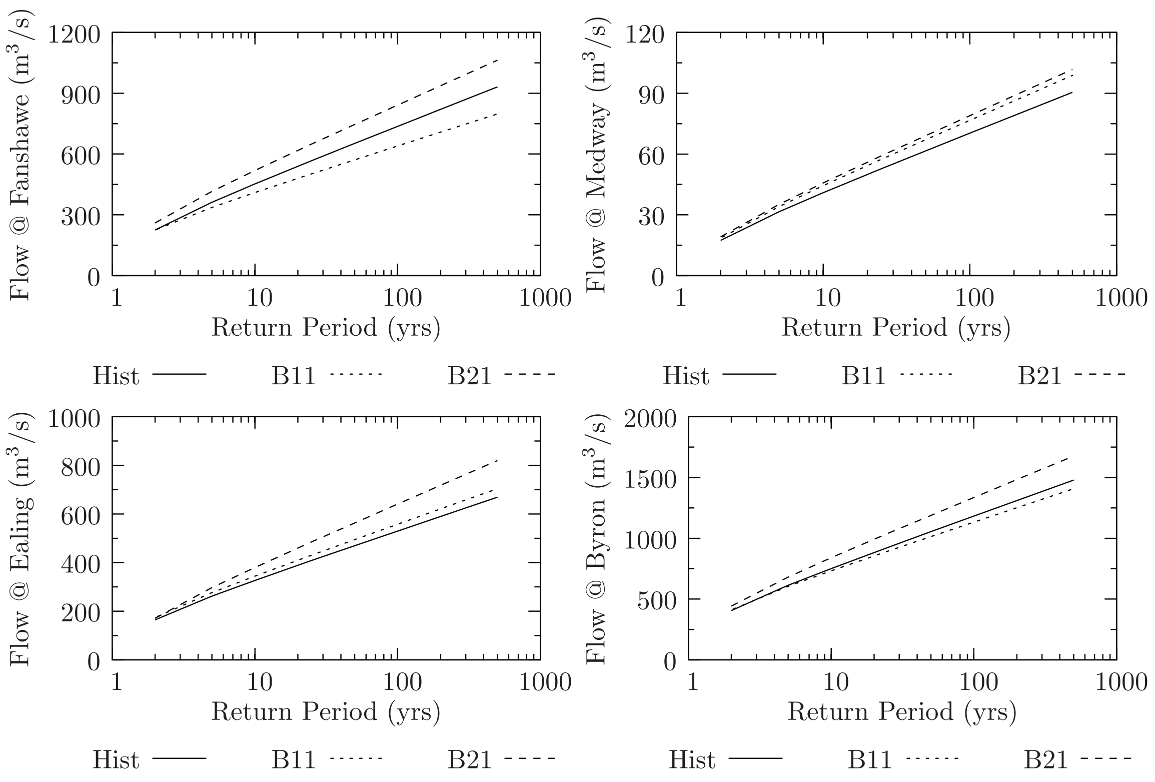

Figure 20: Comparison of peak flows computed via Gumbel distribution

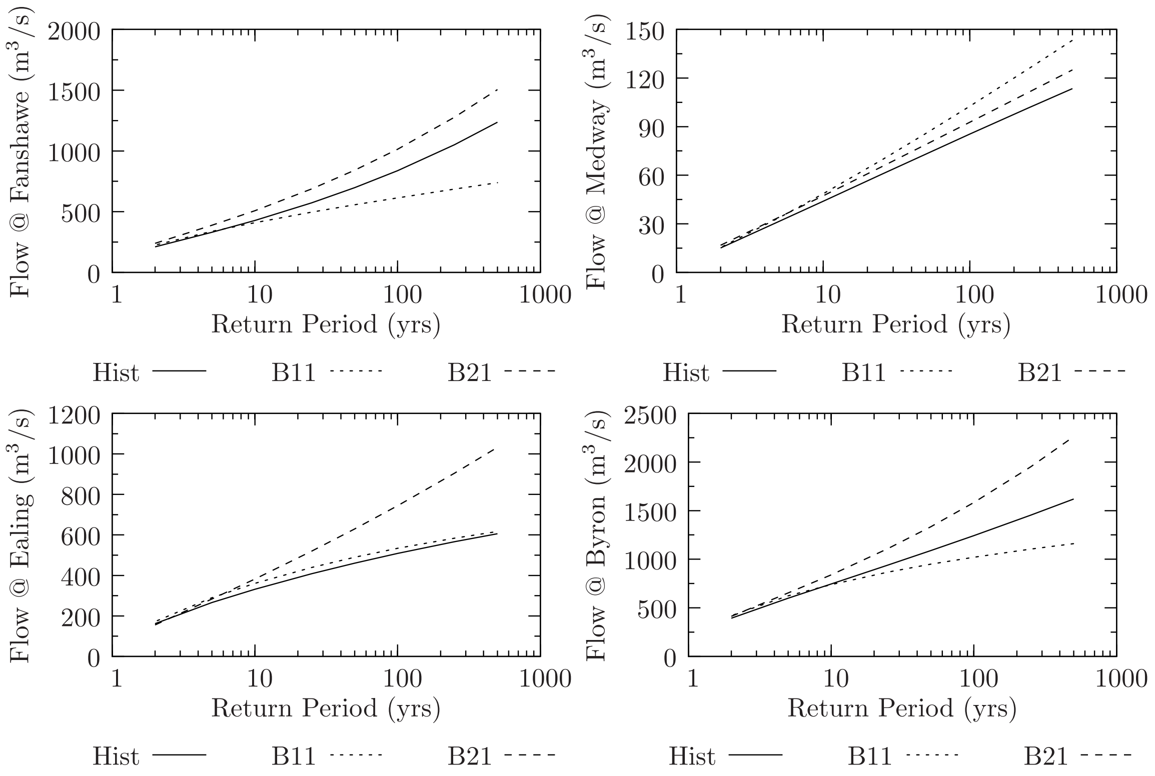

Figure 21: Comparison of peak flows computed via LP3 distribution

Table 5: Flood frequency analysis @ Fanshawe[†]

	Gumbel[‡]			LP3[§]			Current[¶]
T (yrs)	Hist	B11	B21	Hist	B11	B21	
2.0	225.3	225.3	260.0	210.2	225.1	239.6	361.0
5.0	362.2	336.2	415.7	330.7	338.0	387.9	476.9
10.0	452.8	409.7	518.8	427.9	409.8	508.1	534.9
25.0	567.3	502.5	649.1	572.5	495.8	686.9	578.9
50.0	652.2	571.4	745.7	697.2	556.4	841.0	624.0
100.0	736.6	639.7	841.7	837.6	614.1	1014.4	744.7
250.0	847.6	729.7	968.0	1051.1	686.6	1277.4	934.6
500.0	931.4	797.7	1063.4	1236.8	739.1	1505.5	1078.2

† All flows are in m^3/s
‡ Fit based on a Gumbel distribution of one hundred events
§ Fit based on a Log-Pearson III distribution of one hundred events
¶ Current UTRCA values (used in Flood Damage Study, Aug. 2005)

Table 6: Flood frequency analysis @ Medway[†]

	Gumbel[‡]			LP3[§]			Current[¶]
T (yrs)	Hist	B11	B21	Hist	B11	B21	
2.0	17.4	18.5	19.1	15.1	15.4	16.8	N/A
5.0	31.5	34.1	35.1	31.5	33.7	34.1	N/A
10.0	40.9	44.4	45.7	43.9	48.6	47.3	N/A
25.0	52.8	57.4	59.1	60.5	69.3	65.2	N/A
50.0	61.6	67.1	69.1	73.0	85.7	78.9	N/A
100.0	70.3	76.7	78.9	85.4	102.6	92.7	N/A
250.0	81.8	89.3	91.9	101.5	125.6	111.1	N/A
500.0	90.4	98.8	101.7	113.5	143.2	125.0	N/A

† All flows are in m^3/s
‡ Fit based on a Gumbel distribution of one hundred events
§ Fit based on a Log-Pearson III distribution of one hundred events
¶ Current UTRCA values (used in Flood Damage Study, Aug. 2005)

Table 7: Flood frequency analysis @ Ealing[†]

	Gumbel[‡]			LP3[§]			Current[¶]
T (yrs)	Hist	B11	B21	Hist	B11	B21	
2.0	164.3	173.7	171.3	162.1	171.8	155.3	243.0
5.0	262.1	276.4	296.9	266.1	289.7	283.7	369.0
10.0	326.9	344.5	380.1	331.7	360.8	382.5	452.0
25.0	408.7	430.5	485.2	408.4	439.5	519.7	531.9
50.0	469.4	494.3	563.1	460.5	490.0	629.3	635.9
100.0	529.6	557.6	640.5	508.5	534.1	744.4	705.7
250.0	608.9	641.0	742.4	566.3	584.2	904.9	849.5
500.0	668.8	703.9	819.3	606.3	616.7	1032.8	958.3

[†] All flows are in m^3/s
[‡] Fit based on a Gumbel distribution of one hundred events
[§] Fit based on a Log-Pearson III distribution of one hundred events
[¶] Current UTRCA values (used in Flood Damage Study, Aug. 2005)

Table 8: Flood frequency analysis @ Byron[†]

	Gumbel[‡]			LP3[§]			Current[¶]
T (yrs)	Hist	B11	B21	Hist	B11	B21	
2.0	405.0	410.8	442.0	392.9	420.1	413.6	593.0
5.0	613.1	604.0	681.2	599.6	623.1	652.9	843.0
10.0	750.8	731.8	839.6	745.8	739.7	839.8	1070.0
25.0	924.9	893.4	1039.7	939.3	867.1	1109.4	1170.0
50.0	1054.0	1013.2	1188.1	1088.8	948.9	1335.3	1370.0
100.0	1182.2	1132.2	1335.5	1242.6	1021.1	1583.5	1489.0
250.0	1351.0	1288.9	1529.5	1453.7	1104.4	1950.3	1834.0
500.0	1478.4	1407.1	1676.0	1620.0	1159.7	2260.8	2095.0

[†] All flows are in m^3/s
[‡] Fit based on a Gumbel distribution of one hundred events
[§] Fit based on a Log-Pearson III distribution of one hundred events
[¶] Current UTRCA values (used in Flood Damage Study, Aug. 2005)

A.4 Code Listing

This section presents the complete Java source code of the event hydrologic model. Anyone wishing to use the model and its components is free to do so. The code is supplied as is, with no guarantees. We are infinitely indebted to Shawn Gettler for writing the code for this model. All code listings following the file TimeHistory-FileWriter.java are modifications made by P. Prodanović in order to automate the execution of the model for an arbitrary number of events, as well as perform required analysis.

DSSDateFormat.java

```
package cfcas.hydro.util;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;


/**
 * DSSDateFormat encapsulates parsing dates into the format used by HEC-DSS. In
the inexplicable case that a String is formatted incorrectly, DSSDateFormat
returns the current date.
 * @author Shawn Gettler
 * @date Sept 10, 2005
 */
/*
 * CHANGELOG
 * Sept 10, 2005
 *   Initial implementation to clean this try/catch out of the other code.
 */
public abstract class DSSDateFormat {

  protected static final SimpleDateFormat DSS_DATE_FORMAT = new
SimpleDateFormat("dd MMM yyyy HH:mm");


  /**
   * Returns a calendar set to the date parsed from the input String.
   * @param s date in HEC-DSS format
   * @return Calendar representing the parsed date
   */
  public static Calendar parseDateString(String s) {
    Calendar c = Calendar.getInstance();
    try {
      c.setTime(DSS_DATE_FORMAT.parse(s));
    } catch (ParseException exc) {
    }
    return c;
  }


  /**
   * The reverse of parseDateString(), returns a String for the given Calendar.
   * @param c Calendar date
   * @return the date in HEC-DSS format
   */
  public static String getDateString(Calendar c) {
    String s = DSS_DATE_FORMAT.format(c.getTime());
    return s;
  }


}
```

EventSubBasin.java

```
package cfcas.hydro.phys;

import cfcas.hydro.met.MetModel;
import cfcas.hydro.phys.Junction;

/**
 * EventSubbasin is a class that stores all properties and all methods of a
subcatchment of an event based HEC-HMS model.
 * @author Shawn Gettler, Pat Prodanovic
 * @date Sept 16, 2005
 */
/*
 * CHANGELOG
 * Sept 16, 2005 - SG
 *   Implementation of loss model.
 * Sept 22, 2005 - SG
 *   Added transform model and baseflow model.
 * Sept 23, 2005 - SG
 *   Corrected baseflow model. Previously was reporting negative baseflow due to
error in selecting peak flow.
 * Oct 01, 2005 - SG
 *   Corrected the instantaneous vs unit hydrograph. Previously used inst as
unit.
 */
public class EventSubBasin extends Junction {

  // METEOROLOGICAL MODEL
  protected MetModel metModel; // precipitation interpolation model
  protected double[] metWeight; // weight vector for interpolation

  // BASIN PROPERTIES
  protected String basinName; // description of subbasin
  protected double basinLat; // latitude of basin centroid
  protected double basinLon; // longitude of basin centroid
  protected double basinArea; // area of subbasin in km^2

  // LOSS MODEL (Initial and Constant Rate)
  protected double lossInit; // initial loss in mm
  protected double lossRate; // constant loss rate in mm
  protected double lossImp; // basin imperviousness in %
  protected double lossAccum; // accumulated precipitation in mm

  // TRANSFORM MODEL (Clark's Unit Hydrograph)
  protected double clarkTime; // time of concentration in h
  protected double clarkStor; // storage coefficient in h
  protected static final double CLARK_UH_MINAREA = 0.995; // minimum area under
unit hydrograph

  // BASEFLOW MODEL (Recession)
  protected double baseInit; // initial baseflow in m^3/s
  protected double baseRec; // recession coefficient
  protected double baseThr; // total flow recession threshold as ratio-to-peak


  // RESULTANT VALUES
  protected double precipTotal, precipLoss, precipExcess; // precip in mm
  protected double flowDirect, flowBase, flowPeak; // flow in m^3/s
  protected double[] flowFuture; // array of future outputs from known
hydrographs

  /**
   * Creates a new EventSubBasin.
   * @param metModel precipitation interpolation model
   * @param basinName description of subbasin
   * @param basinLat latitude of basin centroid
   * @param basinLon longitude of basin centroid
   * @param basinArea area of subbasin in km<SUP>2</SUP>
   * @param lossInit initial precipitation loss in mm
   * @param lossRate constant precipitation loss rate in mm
   * @param lossImp basin imperviousness in %
   * @param clarkTime time of concentration in h
   * @param clarkStor storage coefficient in h
   * @param baseInit initial baseflow in m<SUP>3</SUP>/km<SUP>2</SUP>s
   * @param baseRec baseflow recession coefficient
   * @param baseThr total flow recession threshold as ratio-to-peak
   */
  public EventSubBasin(MetModel metModel, String basinName, double basinLat,
double basinLon, double basinArea, double lossInit, double lossRate, double
lossImp, double clarkTime, double clarkStor, double baseInit, double baseRec,
double baseThr) {
    /*
     * Initialize basin parameters.
     */
    this.metModel = metModel;
    this.basinName = basinName;
    this.basinLat = basinLat;
    this.basinLon = basinLon;
    this.basinArea = basinArea;
    this.lossInit = lossInit;
    this.lossRate = lossRate;
    this.lossImp = lossImp;
    this.clarkTime = clarkTime;
    this.clarkStor = clarkStor;
    this.baseInit = baseInit * this.basinArea;
    this.baseRec = baseRec;
    this.baseThr = baseThr;

    /*
     * Initialize calculated properties.
     */
    this.metWeight = this.metModel.getWeightVector(this.basinLat,
this.basinLon);
    this.lossAccum = 0;
    this.flowTotal = 0;
    this.flowBase = this.baseInit;
    this.flowPeak = 0;
    this.flowFuture = new double[] {0};

    System.out.println(this.getDescLong());
  }



  public void stepTime(double timestep) {
    /*
     * Calculate input precipitation. This is the dot product of the weight
vector and the precipitation for all gauges.
     */
    this.precipTotal = 0;
    for(int i=0; i < this.metWeight.length; i++) {
      this.precipTotal += this.metWeight[i] *
this.metModel.getPrecipitation()[i];
    }

    /*
     * Calculate excess precipitation after losses.
     * eff* represents effective loss parameters and precipitation considering
imperviousness.
     */
    double effInitial = this.lossInit * (1 - this.lossImp/100);
    double effRate = this.lossRate * (1 - this.lossImp/100);
    double effPrecip = this.precipTotal * (1 - this.lossImp/100);

    this.precipLoss = 0;
    if(this.lossAccum+effPrecip < effInitial) {
      this.precipLoss += effPrecip;
    } else if(this.lossAccum < effInitial && this.lossAccum+effPrecip >
effInitial) {
      double timeToInitial = (effInitial - this.lossAccum) / effPrecip;
      this.precipLoss += effInitial - this.lossAccum;
      this.precipLoss += (1 - timeToInitial) * effRate;
    } else {
      this.precipLoss += effRate;
    }
    if(this.precipLoss > effPrecip) {
      this.precipLoss = effPrecip;
    }
    this.lossAccum += effPrecip;

    this.precipExcess = this.precipTotal - this.precipLoss;

    /*
     * Calculate direct runoff flow:
     * 1) Determine the length of the 0.995 unit hydrograph.
     * A running sum of the area under the unit hydrograph is calculated; the
number of timesteps required for this value to reach 0.995 will be the length of
our "finite" unit hydrograph.
     */
    double clarkC = 2*timestep / (2*this.clarkStor + timestep);

    double uhArea = 0; // cumulative area under uh
    double out = 0; // output at time zero
    int uhCount = 0; // timestep counter
    while(uhArea <= CLARK_UH_MINAREA) {
      double t1 = (uhCount-1)*timestep/this.clarkTime;
      double t2 = uhCount*timestep/this.clarkTime;
      double in = (this.getACRatio(t2) - this.getACRatio(t1));
      out = clarkC*in + (1-clarkC)*out;
      uhArea += out*timestep;
      uhCount++;
    }

    /*
     * 2) Generate the finite unit hydrograph.
     * Same process as above, but we know the domain and can save the values.
     */
    double[] instHG = new double[uhCount];
    double[] unitHG = new double[uhCount];
    instHG[0] = 0;
    unitHG[0] = 0;
    for(int i=1; i < instHG.length; i++) {
      double t1 = (i-1)*timestep/this.clarkTime;
      double t2 = i*timestep/this.clarkTime;
      double in = this.getACRatio(t2) - this.getACRatio(t1);

      in *= this.basinArea * 1000 / (timestep*3600); // convert mm/timestep to
m^3/s

      instHG[i] = clarkC*in + (1-clarkC)*instHG[i-1];
      unitHG[i] = (instHG[i] + instHG[i-1]) / 2;
    }


    /*
     * 3) Add the hydrograph from the current excess precipitation to the
running future flow array.
     * Uses a temporary array to avoid overflow. The future flow array is
shifted one index position for the next timestep.
     */
    double[] flowTemp = new double[Math.max(this.flowFuture.length,
unitHG.length)];
    System.arraycopy(this.flowFuture, 0, flowTemp, 0, this.flowFuture.length);

    for(int i=0; i < unitHG.length; i++) {
      flowTemp[i] += unitHG[i] * this.precipExcess;
    }

    this.flowDirect = flowTemp[0];

    this.flowFuture = new double[flowTemp.length-1];
    for(int i=0; i < this.flowFuture.length; i++) {
      this.flowFuture[i] = flowTemp[i+1];
    }


    /*
     * Calculate baseflow. Total flow curve recession is based on a ratio to
peak flow, so keep track of the largest peak observed.
     */
    if(this.flowDirect==0) {
      this.flowPeak = 0;
    }
    if(this.flowTotal > this.flowPeak) {
      this.flowPeak = this.flowTotal;
    }

    if(this.flowTotal < this.flowPeak*this.baseThr) {
      this.flowTotal *= Math.pow(this.baseRec, timestep/24);
      this.flowBase = this.flowTotal - this.flowDirect;
      if(this.flowBase < 0) {
        this.flowBase = 0;
      }
    } else {
      this.flowBase *= Math.pow(this.baseRec, timestep/24);
      this.flowTotal = this.flowDirect + this.flowBase;
    }

  }


  /*
   * ACCESSOR METHODS
   */
  public String getDescription() { return this.basinName; }
  public double getLatitude() { return this.basinLat; }
  public double getLongitude() { return this.basinLon; }
  public double getArea() { return this.basinArea; }
  public double getInitialLoss() { return this.lossInit; }
  public double getConstantLoss() { return this.lossRate; }
  public double getImperviousness() { return this.lossImp; }
  public double getTimeOfConc() { return this.clarkTime; }
  public double getStorageCoef() { return this.clarkStor; }
  public double getRecessionCoef() { return this.baseRec; }
  public double getRecessionThr() { return this.baseThr; }
  public double getTotalPrecip() { return this.precipTotal; }
  public double getPrecipLosses() { return this.precipLoss; }
  public double getExcessPrecip() { return this.precipExcess; }
  public double getTotalFlow() { return this.flowTotal; }
  public double getBaseFlow() { return this.flowBase; }
  public double getDirectRunoff() { return this.flowDirect; }
  public double getPeakFlow() { return this.flowPeak; }

  public String getDescLong() {
    String s = "  subbasin: " + this.basinName + ", " +
String.valueOf(this.getLatitude()).concat("00000000").substring(0,8) + " " +
String.valueOf(this.getLongitude()).concat("00000000").substring(0,8);
    return s;
  }


  /**
   * Cumulative translation hydrograph function A<SUB>c</SUB>/A:<BR>
   * 1.414(t<SUB>c</SUB>/t)<SUP>1.5</SUP> where t<SUB>c</SUB>/t &lt;= 0.5<BR>
   * 1 - 1.414(1 - t<SUB>c</SUB>/t)<SUP>1.5</SUP> where t<SUB>c</SUB>/t &gt;=
0.5
   * @param t normalized time t<SUB>c</SUB>/t
   * @return A<SUB>c</SUB>/A
   */
  protected double getACRatio(double t) {
    double d = 0;
    if(t >=0 && t <= 0.5) {
      d = 1.414*Math.pow(t, 1.5);
    } else if(t >= 0.5 && t <= 1.0) {
      d = 1 - 1.414*Math.pow(1 - t, 1.5);
    } else if(t >= 1.0) {
      d = 1;
    }
    return d;
  }

}
```

FunctionTable.java

```
package cfcas.hydro.util;

/**
 * FunctionTable is a class that stores paired data (i.e., a series of x and y
values).
 * @author Pat Prodanovic
 * @date Sept 17, 2005
 */
/*
 * CHANGELOG
 * Sept 17, 2005 - PP
 *   Initial implementation.
 */
public class FunctionTable {

  double[] x, y;

  /**
   * Create a new FunctionTable object
   * @param x represents x values of the paired data
   * @param y represents y values of the paired data
   */

  public FunctionTable(double[] x, double[] y) {
    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.
   * @return interpolated y value
   */
  public double getInterpolatedValue(double xhat){
    int n = this.x.length;

    /*
     * 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.println("ordinate outside table range: " + xhat + " < x_0");
      return this.y[0];
    } else if (xhat > this.x[n-1]){
      System.out.println("ordinate outside table range: " + xhat + " > x_n");
      return this.y[n-1];
    }

    /*
     *  performs a linear search
     */
    int i = n - 1;
    while((this.x[i] > xhat) && (i > 0)){
      i--;
    }
    /*
     * 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]);

    return this.y[i]*L1 + this.y[i+1]*L2;
  }


}
```

Junction.java

```java
package cfcas.hydro.phys;

import cfcas.hydro.util.LinkedList;



/**
 * Junction is the simplest component of the physical basin model. Its output is
simply the sum of its inputs.
 * This class is the basis for the rest of the basin components, with its
getOutput() method overwritten.
 * @author Shawn Gettler
 * @date Sept 15, 2005
 */
/*
 * CHANGELOG
 * Sept 15, 2005 - SG
 *   Initial implementation assuming simple addInput()/getOutput() interface.
 * Oct 15, 2005 - SG
 *   Moved summation of input flow to getInputSum() for reuse in Reach and
Reservoir without code duplication.
 */
public class Junction {

  protected LinkedList inputList = new LinkedList();
  protected String junctionName;

  protected double flowTotal;


  public Junction() {
  }

  public Junction(String junctionName) {
    this.junctionName = junctionName;
  }

  /**
   * Adds the given component such that the outflow from the upstream component
is an inflow to this junction.
   * @param input an upstream basin component
   */
  public void addInput(Junction input) {
    this.inputList.add(input);
  }


  /**
   * Outputs the sum of the inputs.
   */
  public void stepTime(double timestep) {
    this.flowTotal = this.getInputSum(timestep);
  }


  public double getTotalFlow() {
    return this.flowTotal;
  }



  public String getDescription() {
    return this.junctionName;
  }



  protected double getInputSum(double timestep) {
    Object[] input = this.inputList.getArray();
    double flow = 0;
    for(int i=0; i < input.length; i++) {
      Junction j = (Junction)input[i];
      j.stepTime(timestep);
      flow += j.getTotalFlow();
    }
    return flow;
  }


}
```

LatLong.java

```
package cfcas.hydro.util;


/**
 * LatLong is a set of utilities relating to latitude and longitude.
 * @author Shawn Gettler
 * @date Sept 04, 2005
 */
/*
 * CHANGELOG
 * Sept 04, 2005 - SG
 *   Initial implementation as functions were needed.
 * Sept 13, 2005 - SG
 *   Added getDMSDegrees()
 */
public abstract class LatLong {

  public static final double MEAN_RADIUS_OF_EARTH = 6371.3; // in km


  /**
   * Returns a decimal equivalent of an angle in degrees, minutes, and seconds.
   * @param dms String listing in the form [-dd]dmmss
   * @return double value of the angle
   */
  public static double getDecimalDegrees(String dms) {
    double deg = Double.valueOf(dms.substring(0, dms.length()-4)).doubleValue();
    double min = Double.valueOf(dms.substring(dms.length()-4,
dms.length()-2)).doubleValue();
    double sec = Double.valueOf(dms.substring(dms.length()-2)).doubleValue();
    if(deg >= 0) {
      return deg + min/60 + sec/3600;
    }
    return deg - min/60 - sec/3600;
  }


  /**
   * Returns a decimal angle as degrees, minutes, and seconds.
   * @param ang double value of the angle
   * @return String listing in the form [-dd]dmmss
   */
  public static String getDMSDegrees(double ang) {
    int deg = (int)Math.floor(Math.abs(ang));
    int min = (int)Math.floor((Math.abs(ang)-deg)*60);
    int sec = (int)Math.floor(((Math.abs(ang)-deg)*60-min)*60);
    if(ang<0) deg *= -1;
    String degs = String.valueOf(deg);
    String mins = String.valueOf(min);
    String secs = String.valueOf(sec);
    String s = degs + "00".concat(mins).substring(mins.length()) +
"00".concat(secs).substring(secs.length());
    return s;
  }


  /**
   * Returns the surface distance between two points, assuming a spherical
Earth.
   * @param lat1 latitude of point 1
   * @param lon1 longitude of point 1
   * @param lat2 latitude of point 2
   * @param lon2 longitude of point 2
   * @return surface distance between points 1 and 2
   */
  /*
   * Function assumes a sphere of radius 1 to compute chord length.
   * dx, dy, and dz compose a vector from point 1 to point 2 in rectangular
coordinates.
   */
  public static double getSurfaceDistance(double lat1, double lon1, double lat2,
double lon2) {
    final double DPR = 180 / Math.PI; // degrees per radian
    double dx = Math.cos(lat2/DPR)*Math.cos(lon2/DPR) -
Math.cos(lat1/DPR)*Math.cos(lon1/DPR);
    double dy = Math.cos(lat2/DPR)*Math.sin(lon2/DPR) -
Math.cos(lat1/DPR)*Math.sin(lon1/DPR);
    double dz = Math.sin(lat2/DPR) - Math.sin(lat1/DPR);
    double chord = Math.sqrt(dx*dx + dy*dy + dz*dz); // pythagorean theorem
    double angc = Math.acos(1 - chord*chord/2); // cosine law, a=1 and b=1
    return angc * LatLong.MEAN_RADIUS_OF_EARTH;
  }


  /**
   * Returns the quadrant in which point 2 falls, relative to point 1.
   * @param lat1 latitude of point 1
   * @param lon1 longitude of point 1
   * @param lat2 latitude of point 2
   * @param lon2 longitude of point 2
   * @return quadrant of point 2 relative to point 1, clockwise from NE=0
   */
  /*
   * Quadrants numbered as follows:
   *
   *   3 | 0
   *   --+--
   *   2 | 1
   */
  public static int getQuadrant(double lat1, double lon1, double lat2, double
lon2) {
    if(lon2 >= lon1) {
      if(lat2 >= lat1) {
        return 0;
      } else {
        return 1;
      }
    } else {
      if(lat2 < lat1) {
        return 2;
      } else {
        return 3;
      }
    }
  }


}
```

LinkedList.java

```
package cfcas.hydro.util;

import java.lang.Object;
import java.util.Iterator;



/**
 * LinkedList is a dynamically-resizing Object list which provides an iterator.
 * @author Shawn Gettler
 * @date Sept 02, 2005
 */
/*
 * CHANGELOG
 * Sept 02, 2005 - SG
 *   Initial implementation from pre-existing code (my own).
 */
public class LinkedList {

  protected LinkedListNode head, tail;
  protected int length;


  /**
   * Creates a new, blank list.
   */
  public LinkedList() {
    this.head = new LinkedListNode(null, null, null);
    this.tail = new LinkedListNode(null, null, null);
    this.clear();
  }


  /**
   * Clears all Objects from the list.
   */
  public void clear() {
    this.head.setNextNode(tail);
    this.tail.setPrevNode(head);
    this.length = 0;
  }


  /**
   * Adds an Object to the end of the list.
   * @param o the Object to add
   */
  public void add(Object o) {
    LinkedListNode newnode = new LinkedListNode(o, this.tail.getPrevNode(),
this.tail);
    newnode.getPrevNode().setNextNode(newnode);
    this.tail.setPrevNode(newnode);
    this.length++;
  }


  /**
   * Generates an array of the contents of the list.
   * @return an array of Objects
   */
  public Object[] getArray() {
    Object[] o = new Object[this.length];
    Iterator t = this.iterator();
    int i = 0;
    while(t.hasNext()) {
      o[i] = t.next();
      i++;
    }
    return o;
  }


  /**
   * Provides an Iterator which includes the remove function.
   * @return the iterator
   */
  public Iterator iterator() {
    return new LinkedListIterator();
  }



  /*
   * "Node" in the list which includes the object stored and a pointer to the
nodes previous and subsequent to it in the list.
   */
  protected class LinkedListNode {
    protected Object o;
    protected LinkedListNode next, prev;
    /*
     * Creates a node with Object o which points to nodes prev and next.
     */
    public LinkedListNode(Object o, LinkedListNode prev, LinkedListNode next) {
      this.o = o;
      this.next = next;
      this.prev = prev;
    }
    /*
     * ACCESSOR METHODS
     */
    public Object getObject() { return this.o; }
    public LinkedListNode getNextNode() { return this.next; }
    public void setNextNode(LinkedListNode next) { this.next = next; }
    public LinkedListNode getPrevNode() { return this.prev; }
    public void setPrevNode(LinkedListNode prev) { this.prev = prev; }
  }



  /*
   * Implementation of an Iterator, including the remove function.
   *
   * REMOVE
   *   before:
   *           [prev]<-->[curr]<-->[next]
   *
   *   after:          ,--------,
   *           [curr]<-' [xxxx] '->[next]
   */
  protected class LinkedListIterator implements Iterator {
    protected LinkedListNode current = head;
    protected boolean nextCalled = false;
    /*
     * Returns true if there are more nodes in the list.
     */
    public boolean hasNext() {
      return (current.getNextNode()!=tail);
    }
    /*
     * Removes the node which corresponds to the last next() call. Will do
nothing if next() has not been called since its last use.
     */
    public void remove() {
      if(this.nextCalled) {
        this.current.getPrevNode().setNextNode(this.current.getNextNode());
        this.current.getNextNode().setPrevNode(this.current.getPrevNode());
        this.current = this.current.getPrevNode();
        length--;
        this.nextCalled = false;
      }
    }
    /*
     * Returns the next Object.
     */
    public Object next() {
      this.current = this.current.getNextNode();
      this.nextCalled = true;
      return this.current.getObject();
    }
  }



}
```

MetModel.java

```
package cfcas.hydro.met;

import cfcas.hydro.util.LinkedList;
import java.util.Calendar;



/**
 * MetModel encapsulates the meteorological model by which recorded or generated
precipitation values for a set of stations is interpolated for given points of
interest (typically, the centroid of a subbasin).
 * The data is supplied as a vector of values as read via GaugeData objects for
each time step. For each point of interest, a weighting vector can be calculated
such that its dot product with the data vector provides a precipitation value
for that point.
 * This approach allows a single weight vector to be calculated for each point
of interest and applied repeatedly.
 * @author Shawn Gettler
 * @date Sept 09, 2005
 */
/*
 * CHANGELOG
 * Sept 04, 2005 - SG
 *   Initial implementation using the weight vector/data vector approach.
 */
public class MetModel {

  protected LinkedList precipDataList = new LinkedList();
  protected double[] currentData;

  protected Calendar tsStart;
  protected int tsLength;
  protected double tsStep;


  /**
   * Though this class must be instantiated, the constructor does nothing.
   */
  public MetModel() {
  }


  /**
   * Adds a source of precipitation data to the meteorologic model. This can be
a direct PrecipDataSource or a SnowModel.
   * @param p source of precipitation data
   */
  public void addPrecipDataSource(PrecipDataSource p) {
    this.tsStart = p.getTimeSeriesStart();
    this.tsLength = p.getTimeSeriesLength();
    this.tsStep = p.getTimeStep();
    this.precipDataList.add(p);
    System.out.println(p.getDescLong());
  }


  /**
   * Returns a weight vector based on the Inverse-Distance-Squared
Method:<br><br>
   * w<sub>i</sub> = d<sub>i</sub><sup>-2</sup> /
(d<sub>0</sub><sup>-2</sup>+d<sub>1</sub><sup>-2</sup>+d<sub>2</sub><sup>-2</sup
>+d<sub>3</sub><sup>-2</sup>)
```

```
   * @param lat latitude of reference point
   * @param lon longitude of reference point
   * @return weight vector for use with the data vector
   */
  public double[] getWeightVector(double lat, double lon) {
    Object[] dataSourceArray = this.precipDataList.getArray();

    /*
     * Find the nearest stations ("nodes") in each of the four quadrants
surrounding the point of interest.
     * Loop through them all, keeping the nearest nodes in an array.
     */
    PrecipDataSource[] nearNode = new PrecipDataSource[4];
    for(int i=0; i < dataSourceArray.length; i++) {
      PrecipDataSource p = (PrecipDataSource)dataSourceArray[i];
      int quadrant = p.getQuadrant(lat,lon);
      if(nearNode[quadrant]==null ||
p.getDistance(lat,lon)<nearNode[quadrant].getDistance(lat,lon)) {
        nearNode[quadrant] = p;
      }
    }

    /*
     * Get the inverse squares of the distances to the nearest nodes. Also, sum
these values for normalization.
     * If no node was found in a given quadrant, give it value 0.
     */
    double[] invsqdist = new double[4];
    double invsqsum = 0;
    for(int i=0; i < 4; i++) {
      if(nearNode[i]!=null) {
        double d = nearNode[i].getDistance(lat,lon);
        invsqdist[i] = 1 / (d*d);
      } else {
        invsqdist[i] = 0;
      }
      invsqsum += invsqdist[i];
    }

    /*
     * Calculate the weights for all data sources.
     * If a precip source was not one of the nearest nodes, give it value 0.
     */
    double[] weight = new double[dataSourceArray.length];
    for(int i=0; i < weight.length; i++) {
      weight[i] = 0;
      for(int j=0; j < 4; j++) {
        if(nearNode[j]==dataSourceArray[i]) {
          weight[i] = invsqdist[j]/invsqsum;
        }
      }
    }

    return weight;
  }


  /**
   * Step the time forward by one unit. This method populates the current data
array and must be called before getPrecipitation().
   * Data is read from source files by the precip sources.
   */
  public void stepTime() {
    Object[] dataArray = this.precipDataList.getArray();
    this.currentData = new double[dataArray.length];
    for(int i=0; i < dataArray.length; i++) {
      PrecipDataSource p = (PrecipDataSource)dataArray[i];
      this.currentData[i] = p.getPrecipitation();
    }
  }


  /**
   * Return the precipitation values for the current time step.
   * @return a vector of values
   */
  public double[] getPrecipitation() {
    return this.currentData;
  }


  /**
   * Return the start date for the time series.
   * @return start date
   */
  public Calendar getTimeSeriesStart() {
    return this.tsStart;
  }


  /**
   * Return the time step.
   * @return step in hours
   */
  public double getTimeStep() {
    return this.tsStep;
  }


  /**
   * Return the number of values in the time series.
   * @return time series length
   */
  public int getTimeSeriesLength() {
    return this.tsLength;
  }


}
```

PrecipDataSource.java

```
package cfcas.hydro.met;

import java.util.Calendar;

import cfcas.hydro.io.TimeHistoryFileReader;



/**
 * PrecipDataSource is essentially a wrapper around the time history file. This
allows the implementation of the SnowModel to be invisible to anything reading
precipitation data.
 * @author Shawn Gettler
 * @date Sept 15, 2005
 */
/*
 * CHANGELOG
 * Sept 15, 2005 - SG
 *   Initial implementation.
 */
public class PrecipDataSource {

  protected TimeHistoryFileReader precipData;


  /**
   * Creates a new PrecipDataSource.
   * @param precipData file from which to read a time series
   */
  public PrecipDataSource(TimeHistoryFileReader precipData) {
    this.precipData = precipData;
  }


  /**
   * Returns the quadrant in which this station falls relative to the given
coordinates.
   * @param lat latitude of the reference point
   * @param lon longitude of the reference point
   * @return quadrant number, clockwise from NE=0
   */public int getQuadrant(double lat, double lon) {
    return this.precipData.getQuadrant(lat, lon);
  }

  /**
   * Returns the distance between this station and the given coordinates.
   * @param lat latitude of the reference point
   * @param lon longitude of the reference point
   * @return distance in km
   */
   public double getDistance(double lat, double lon) {
     return this.precipData.getDistance(lat, lon);
   }


  /**
   * Returns the next value from the time series.
   * @return precipitation in mm
   */
  public double getPrecipitation() {
    return this.precipData.getNextValue();
  }
  /**
   * Returns the start date and time of this time series in a Calendar object.
   * @return start date
   */
  public Calendar getTimeSeriesStart() {
    return this.precipData.getTimeSeriesStart();
  }


  /**
   * Returns the number of values in the time series.
   * @return length of series
   */
  public int getTimeSeriesLength() {
    return this.precipData.getTimeSeriesLength();
  }


  /**
   * Returns the length of the time step for this time series.
   * @return time step in hours
   */
  public double getTimeStep() {
    return this.precipData.getTimeStep();
  }


  /**
   * Returns a string describing this gauge station and time series.
   * @return description
   */
  public String getDescLong() {
    return this.precipData.getDescLong();
  }


  /**
   * Returns only the "name" of the station.
   * @return short description
   */
  public String getDescription() {
    return this.precipData.getDescription();
  }

}
```

Reach.java

```
package cfcas.hydro.phys;

import cfcas.hydro.phys.Junction;
import cfcas.hydro.util.FunctionTable;


/**
 *
 * @author Shawn Gettler, Pat Prodanovic
 * @date Sept 30, 2005
 */
/*
 * CHANGELOG
 * Sept 30, 2005 - SG
 *   Initial implementation based on array-based code from Pat.
 * Oct 15, 2005 - SG
 *   Modified to utilize getInputSum() from Junction.
 */
public class Reach extends Junction {

  protected double[] storage, outflow;
  protected double lastInflow, lastIndicn, lastOutflow;

  public Reach(String reachName, double[] storage, double[] outflow) {
    super(reachName);

    this.storage = storage;
    this.outflow = outflow;

    this.lastInflow = 0;
    this.lastIndicn = 0;
    this.lastOutflow = 0;
  }



  public void stepTime(double timestep) {
    /*
     * Generate indication-outflow curve; create outflow-storage curve;
     */
    double[] indication = new double[this.storage.length];
    for(int i=0; i < this.storage.length; i++) {
      indication[i] = this.storage[i]*1000*(timestep/3600) + this.outflow[i]/2;
    }
    FunctionTable indicn_out = new FunctionTable(indication, this.outflow);

    /*
     * Get current inflow.
     */
    double inflow = this.getInputSum(timestep);


    if(this.lastInflow==0) {
      FunctionTable out_stor = new FunctionTable(this.outflow, this.storage);

      this.lastInflow = inflow;
      this.lastOutflow = inflow;
      this.lastIndicn =
out_stor.getInterpolatedValue(this.lastOutflow)*1000*(timestep/3600) +
this.lastOutflow/2;
    }
    double avgInflow = (inflow + this.lastInflow) / 2;
    double indicn = this.lastIndicn - this.lastOutflow + avgInflow;

    this.flowTotal = indicn_out.getInterpolatedValue(indicn);



    this.lastInflow = inflow;
    this.lastOutflow = this.flowTotal;
    this.lastIndicn = indicn;
  }


}
```

Reservoir.java

```
package cfcas.hydro.phys;

import cfcas.hydro.phys.Junction;
import cfcas.hydro.util.FunctionTable;


/**
 *
 * @author Shawn Gettler, Pat Prodanovic
 * @date Sept 30, 2005
 */
/*
 * CHANGELOG
 * Sept 30, 2005 - SG
 *   Initial implementation based on array-based code from Pat.
 * Oct 15, 2005 - SG
 *   Modified to utilize getInputSum() from Junction.
 */
public class Reservoir extends Junction {

  protected double[] storage, outflow;
  protected double lastInflow, lastIndicn, lastOutflow;

  public Reservoir(String reservoirName, double[] storage, double[] outflow) {
    super(reservoirName);

    this.storage = storage;
    this.outflow = outflow;

    this.lastInflow = 0;
    this.lastIndicn = 0;
    this.lastOutflow = 0;
  }



  public void stepTime(double timestep) {
    /*
     * Generate indication-outflow curve; create outflow-storage curve;
     */
    double[] indication = new double[this.storage.length];
    for(int i=0; i < this.storage.length; i++) {
      indication[i] = this.storage[i]*1000*(timestep/3600) + this.outflow[i]/2;
    }
    FunctionTable indicn_out = new FunctionTable(indication, this.outflow);

    /*
     * Get current inflow.
     */
    double inflow = this.getInputSum(timestep);

    if(this.lastInflow==0) {
      FunctionTable out_stor = new FunctionTable(this.outflow, this.storage);

      this.lastInflow = inflow;
      this.lastOutflow = inflow;
      this.lastIndicn =
out_stor.getInterpolatedValue(this.lastOutflow)*1000*(timestep/3600) +
this.lastOutflow/2;
    }
    double avgInflow = (inflow + this.lastInflow) / 2;
    double indicn = this.lastIndicn - this.lastOutflow + avgInflow;

    this.flowTotal = indicn_out.getInterpolatedValue(indicn);



    this.lastInflow = inflow;
    this.lastOutflow = this.flowTotal;
    this.lastIndicn = indicn;
  }



}
```

testEventModel.java

```
import cfcas.hydro.io.TimeHistoryFileReader;
import cfcas.hydro.io.TimeHistoryFileWriter;
import cfcas.hydro.met.MetModel;
import cfcas.hydro.met.PrecipDataSource;
import cfcas.hydro.phys.EventSubBasin;
import cfcas.hydro.phys.Junction;
import cfcas.hydro.phys.Reach;
import cfcas.hydro.phys.Reservoir;
import cfcas.hydro.util.DSSDateFormat;
import cfcas.hydro.util.LatLong;
import java.util.Calendar;

/**
 *
 * @author Shawn Gettler
 * @date
 */
/*
 * CHANGELOG
 */
public class testEventModel {

  public static void main(String[] arg) {

    System.out.println("UPPER THAMES RIVER BASIN MODEL");
    System.out.println();


    /*
     * Set up meteorologic model.
     */
    MetModel metModel = new MetModel();

    System.out.println("loading July 2000 data...");
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Avon.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Dutton.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Ethel.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Exeter.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\FanshaweDam.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Innerkip.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Listowel.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\MedwayCreek.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Millbank.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Mitchell.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\NewHamburg.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Parkhill.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\PittockDam.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\PloverMills.txt", 0)));
```

```
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Springbank.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\StMarys.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Stratford.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Thamesford.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\WaubunoCreek.txt", 0)));
    metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS2\\jul2000\\Woodstock.txt", 0)));

    System.out.println("done");
    System.out.println();

    Calendar calendar = Calendar.getInstance();
    calendar.setTime(metModel.getTimeSeriesStart().getTime());


    /*
     * Set up physical basin model.
     * 1) subbasins
     */
    System.out.println("initializing event model components...");
    EventSubBasin sbNThmWhirl = new EventSubBasin(  metModel, "[01] North Thames
above Whirl Ck",      LatLong.getDecimalDegrees("433306"),
LatLong.getDecimalDegrees("-810856"), 175.982,  5.0, 1.0,  0.0,  8.0, 10.0,
0.010, 0.4, 0.70, 1.0);
    EventSubBasin sbWhirl = new EventSubBasin(      metModel, "[02] Whirl
Creek",                     LatLong.getDecimalDegrees("432931"),
LatLong.getDecimalDegrees("-810334"), 129.523,  5.0, 1.0,  0.0, 10.0, 12.0,
0.010, 0.4, 0.70, 1.0);
    EventSubBasin sbNThmBlack = new EventSubBasin(  metModel, "[03] North Thames
above Black Ck",      LatLong.getDecimalDegrees("432410"),
LatLong.getDecimalDegrees("-811303"),  47.745,  5.0, 1.1,  0.0, 12.0,  6.0,
0.010, 0.4, 0.40, 1.0);
    EventSubBasin sbBlack = new EventSubBasin(      metModel, "[04] Black
Creek",                     LatLong.getDecimalDegrees("432550"),
LatLong.getDecimalDegrees("-810056"), 151.189,  5.0, 1.0,  0.0, 12.0, 10.0,
0.010, 0.4, 0.50, 1.0);
    EventSubBasin sbNThmAvon = new EventSubBasin(   metModel, "[05] North Thames
above Avon R",        LatLong.getDecimalDegrees("432128"),
LatLong.getDecimalDegrees("-810915"),  76.820,  5.0, 1.1,  0.0,  7.0,  6.0,
0.010, 0.4, 0.40, 1.0);

    // basin #6
    EventSubBasin sbAvon = new EventSubBasin(       metModel, "[07] Avon River",
                      LatLong.getDecimalDegrees("432232"),
LatLong.getDecimalDegrees("-805910"), 144.000,  5.0, 1.0,  2.0,  5.0, 10.0,
0.010, 0.4, 0.10, 1.0);
    EventSubBasin sbFlat = new EventSubBasin(       metModel, "[08] Flat Creek",
                      LatLong.getDecimalDegrees("432228"),
LatLong.getDecimalDegrees("-811522"),  88.355,  5.0, 1.0,  0.0, 11.0,  7.0,
0.010, 0.4, 0.40, 1.0);
    EventSubBasin sbNThmFlat = new EventSubBasin(   metModel, "[09] North Thames
above Flat Ck",       LatLong.getDecimalDegrees("431837"),
LatLong.getDecimalDegrees("-810356"),  78.476,  5.0, 1.1,  0.0,  7.0,  6.0,
0.010, 0.4, 0.40, 1.0);
    EventSubBasin sbTroutWWD = new EventSubBasin(   metModel, "[10] Trout Creek
above Wildwood",       LatLong.getDecimalDegrees("432013"),
LatLong.getDecimalDegrees("-805626"), 141.118,  5.0, 1.1,  0.0, 13.0,  9.0,
0.010, 0.5, 0.40, 1.0);
```

```
    EventSubBasin sbTrout = new EventSubBasin(      metModel, "[11] Trout
Creek",                      LatLong.getDecimalDegrees("431637"),
LatLong.getDecimalDegrees("-810546"),  28.942,  5.0, 1.2,  0.0,  9.0,  5.0,
0.010, 0.3, 0.40, 1.0);
    EventSubBasin sbNThmFish = new EventSubBasin(   metModel, "[12] North Thames
above Fish Ck",       LatLong.getDecimalDegrees("431459"),
LatLong.getDecimalDegrees("-811204"),  35.466,  5.0, 1.3,  0.0, 10.0,  8.0,
0.010, 0.3, 0.40, 1.0);
    EventSubBasin sbFish = new EventSubBasin(       metModel, "[13] Fish Creek",
                      LatLong.getDecimalDegrees("431852"),
LatLong.getDecimalDegrees("-811726"), 153.721,  5.0, 1.0,  0.0, 13.0, 14.0,
0.010, 0.3, 0.40, 1.0);
    EventSubBasin sbNThmPlover = new EventSubBasin( metModel, "[14] North Thames
above Plover Mills",  LatLong.getDecimalDegrees("431215"),
LatLong.getDecimalDegrees("-810913"),  84.539,  5.0, 1.5,  0.0, 14.0, 10.0,
0.010, 0.3, 0.40, 1.0);
    EventSubBasin sbNThmFan = new EventSubBasin(    metModel, "[15] North Thames
above Fanshawe",      LatLong.getDecimalDegrees("431057"),
LatLong.getDecimalDegrees("-810526"),  94.198,  5.0, 2.0,  0.0, 15.0, 20.0,
0.010, 0.3, 0.40, 1.0);

    EventSubBasin sbNThmMedway = new EventSubBasin( metModel, "[16] North Thames
above Medway Ck",     LatLong.getDecimalDegrees("430345"),
LatLong.getDecimalDegrees("-811438"),  75.363,  5.0, 2.0,  5.0, 16.0, 16.0,
0.010, 0.5, 0.30, 1.0);
    EventSubBasin sbMedway = new EventSubBasin(     metModel, "[17] Medway
Creek",                     LatLong.getDecimalDegrees("430837"),
LatLong.getDecimalDegrees("-811659"), 202.478,  5.0, 2.3,  0.0, 24.0, 20.0,
0.010, 0.5, 0.10, 1.0);
    EventSubBasin sbThmInn = new EventSubBasin(     metModel, "[18] Thames River
above Innerkip",      LatLong.getDecimalDegrees("431707"),
LatLong.getDecimalDegrees("-804854"), 148.318,  5.0, 1.0,  0.0, 10.0,  9.0,
0.010, 0.4, 0.80, 1.0);
    EventSubBasin sbThmPit = new EventSubBasin(     metModel, "[19] Thames River
above Pittock Dam",   LatLong.getDecimalDegrees("431140"),
LatLong.getDecimalDegrees("-804529"),  96.840,  5.0, 1.1,  0.0, 15.0,  9.0,
0.010, 0.3, 0.80, 1.0);
    EventSubBasin sbCedar = new EventSubBasin(      metModel, "[20] Cedar
Creek",                      LatLong.getDecimalDegrees("430405"),
LatLong.getDecimalDegrees("-804407"),  97.910,  5.0, 1.0,  0.0,  2.0,  3.0,
0.010, 0.5, 0.10, 1.0);

    EventSubBasin sbThmIng = new EventSubBasin(     metModel, "[21] Thames River
above Ingersoll",     LatLong.getDecimalDegrees("430209"),
LatLong.getDecimalDegrees("-804853"), 170.704,  5.0, 1.3,  0.0, 24.0, 12.0,
0.010, 0.6, 0.50, 1.0);
    EventSubBasin sbThmMThm = new EventSubBasin(    metModel, "[22] Thames River
above Middle Thames", LatLong.getDecimalDegrees("430226"),
LatLong.getDecimalDegrees("-805457"),  42.859,  5.0, 1.3,  0.0, 24.0,  9.0,
0.010, 0.6, 0.50, 1.0);
    EventSubBasin sbThmf = new EventSubBasin(       metModel, "[23] Middle
Thames above Thamesford",   LatLong.getDecimalDegrees("431000"),
LatLong.getDecimalDegrees("-805400"), 291.080,  5.0, 3.5,  0.0,  7.0, 17.0,
0.010, 0.5, 0.20, 1.0);
    EventSubBasin sbMThmThm = new EventSubBasin(    metModel, "[24] Middle
Thames above Thames River", LatLong.getDecimalDegrees("430302"),
LatLong.getDecimalDegrees("-805723"),  35.861,  5.0, 1.4,  0.0, 25.0,  8.0,
0.010, 0.5, 0.20, 1.0);
    EventSubBasin sbReynolds = new EventSubBasin(   metModel, "[25] Reynolds
Creek",                   LatLong.getDecimalDegrees("425718"),
LatLong.getDecimalDegrees("-805245"), 165.973,  5.0, 2.0,  0.0, 25.0, 15.0,
0.010, 0.7, 0.70, 1.0);

    EventSubBasin sbThmWaubuno = new EventSubBasin( metModel, "[26] Thames River
above Waubuno Ck",    LatLong.getDecimalDegrees("430007"),
LatLong.getDecimalDegrees("-810229"), 120.935,  5.0, 1.3,  0.0, 20.0, 10.0,
0.010, 0.6, 0.50, 1.0);
    EventSubBasin sbWaubuno = new EventSubBasin(    metModel, "[27] Waubuno
Creek",                    LatLong.getDecimalDegrees("430613"),
LatLong.getDecimalDegrees("-810447"), 104.945,  5.0, 3.5,  0.0, 15.0, 16.0,
0.010, 0.5, 0.10, 1.0);
    EventSubBasin sbThmEal = new EventSubBasin(     metModel, "[28] Thames River
above Ealing",        LatLong.getDecimalDegrees("430200"),
LatLong.getDecimalDegrees("-810826"),  61.195,  5.0, 1.4,  0.0, 20.0,  8.0,
0.010, 0.6, 0.50, 1.0);
    EventSubBasin sbThmNThm = new EventSubBasin(    metModel, "[29] Thames River
above North Thames",  LatLong.getDecimalDegrees("425745"),
LatLong.getDecimalDegrees("-811302"),  22.556,  5.0, 2.2, 40.0,  4.0,  6.0,
0.010, 0.5, 0.30, 1.0);
    EventSubBasin sbThmByron = new EventSubBasin(   metModel, "[30] Thames River
above Byron",         LatLong.getDecimalDegrees("425911"),
LatLong.getDecimalDegrees("-811814"),  30.002,  5.0, 2.3, 30.0,  7.0, 10.0,
0.010, 0.5, 0.30, 1.0);

    EventSubBasin sbThmOxbow = new EventSubBasin(   metModel, "[31] Thames River
above Oxbow Ck",      LatLong.getDecimalDegrees("425938"),
LatLong.getDecimalDegrees("-812034"),  32.409,  5.0, 2.2,  0.0,  6.0,  6.0,
0.010, 0.5, 0.30, 1.0);
    EventSubBasin sbOxbow = new EventSubBasin(      metModel, "[32] Oxbow
Creek",                     LatLong.getDecimalDegrees("430134"),
LatLong.getDecimalDegrees("-812207"),  88.845,  5.0, 4.0,  0.0, 40.0, 14.0,
0.010, 0.6, 0.40, 1.0);
    EventSubBasin sbThmDing = new EventSubBasin(    metModel, "[33] Thames River
above Dingman Ck",    LatLong.getDecimalDegrees("425635"),
LatLong.getDecimalDegrees("-812737"),  50.486,  5.0, 2.4,  0.0,  8.0,  7.0,
0.010, 0.5, 0.30, 1.0);
    EventSubBasin sbDingman = new EventSubBasin(    metModel, "[34] Dingman
Creek",                    LatLong.getDecimalDegrees("425520"),
LatLong.getDecimalDegrees("-811357"), 168.719,  5.0, 3.0,  2.0, 12.0,  8.0,
0.010, 0.3, 0.05, 1.0);


    /*
     * 2) reaches
     */
    Reach rcPitCedar = new Reach(    "[R111] Thames River, Pittock Dam to Cedar
Ck",       new double[] {0.00, 50.50, 86.16, 196.43, 329.36, 580.92, 673.32,
740.40, 794.03, 1058.11, 1655.93},                new double[] {0.00, 4.40,
8.80, 21.60, 34.00, 74.00, 83.50, 90.00, 93.50, 130.00, 211.00},
1.0);
    Reach rcCedarBch = new Reach(    "[R222] Thames River, Cedar Ck to
Beachville",        new double[] {0.00, 95.13, 156.33, 378.48, 690.30, 1162.93,
1378.07, 1642.92, 1800.79, 2213.88, 3732.20},           new double[] {0.00,
7.00, 14.00, 35.00, 70.00, 107.00, 125.00, 140.00, 156.00, 200.00, 307.20},
    1.0);
    Reach rcBchIng = new Reach(      "[R333] Thames River, Beachville to
Ingersoll",       new double[] {0.00, 107.93, 169.42, 311.18, 513.11, 655.22,
733.83, 809.86, 876.07, 906.99, 1542.14},               new double[] {0.00,
10.80, 21.60, 54.00, 108.00, 154.00, 179.00, 204.00, 223.00, 239.00, 414.00},
    1.0);
    Reach rcMitBlack = new Reach(    "[R560] North Thames, Mitchell to Black
Ck",          new double[] {0.00, 389.67, 657.93, 1421.86, 2607.09, 3508.72,
3860.86, 5109.41, 5650.00, 5984.53, 7505.39},        new double[] {0.00, 20.70,
41.40, 104.00, 207.00, 292.00, 355.00, 445.00, 508.00, 550.00, 768.00},
1.0);
    Reach rcBlackAvon = new Reach(   "[R640] North Thames, Black Ck to Avon R",
          new double[] {0.00, 271.15, 440.87, 924.18, 1663.83, 2256.23, 2499.10,
3106.65, 3440.88, 3621.61, 4737.15},         new double[] {0.00, 25.30, 50.50,
126.00, 253.00, 356.00, 426.00, 534.00, 610.00, 660.00, 937.00},       1.0);
    Reach rcAvonFlat = new Reach(    "[R750] North Thames, Avon R to Flat Ck",
          new double[] {0.00, 104.90, 177.99, 377.73, 668.41, 876.97, 1028.77,
1280.78, 1332.32, 1391.07, 1649.61},           new double[] {0.00, 27.50, 55.00,
138.00, 270.00, 380.00, 461.00, 578.00, 660.00, 715.00, 1021.00},      1.0);
    Reach rcFlatTrout = new Reach(   "[R900] North Thames, Flat Ck to Trout Ck",
          new double[] {0.00, 182.90, 297.24, 603.72, 1058.99, 1369.71, 1578.41,
1866.99, 2093.57, 2184.92, 2782.47},         new double[] {0.00, 31.90, 64.00,
159.00, 345.00, 487.00, 566.00, 657.00, 716.00, 785.00, 1183.00},      1.0);

    Reach rcWWDNThm = new Reach(     "[R930] Trout Creek, Wildwood Dam to North
Thames",   new double[] {0.00, 67.00, 95.00, 115.00, 138.17, 208.56, 319.53,
543.70, 1132.43, 3021.35, 5241.65},               new double[] {0.00, 3.00,
5.00, 7.00, 10.00, 20.00, 30.00, 50.00, 100.00, 151.00, 180.00},
1.0);
    Reach rcTroutFish = new Reach(   "[R1010] North Thames, Trout Ck to Fish
Ck",          new double[] {0.00, 309.23, 503.96, 985.75, 1726.00, 2222.59,
2558.16, 2945.41, 3234.45, 3522.67, 4514.07},         new double[] {0.00, 33.50,
67.00, 168.00, 335.00, 450.00, 530.00, 630.00, 705.00, 784.00, 1057.00},
1.0);
    Reach rcIngMThm = new Reach(     "[R1870] Thames River, Ingersoll to Middle
Thames",   new double[] {0.00, 264.88, 474.80, 1321.19, 1810.16, 3810.08,
4278.23, 4675.72, 5832.38, 6291.38, 8737.16},        new double[] {0.00, 13.20,
26.40, 67.00, 120.00, 214.00, 249.00, 280.00, 320.00, 360.00, 572.00},
1.0);
    Reach rcThmfThm = new Reach(     "[R1890] Middle Thames, Thamesford to
Thames River",  new double[] {0.00, 404.07, 543.70, 958.47, 1558.60, 2047.74,
2327.52, 2642.25, 2881.41, 3097.13, 3846.54},         new double[] {0.00, 9.10,
18.10, 45.30, 90.00, 133.00, 159.00, 190.00, 214.00, 236.00, 314.00},
1.0);
    Reach rcFanMedway = new Reach(   "[R1910] North Thames, Fanshawe Dam to
Medway Ck",    new double[] {0.00, 837.87, 1279.23, 2592.06, 4971.75, 6159.72,
7120.41, 7712.24, 8615.57, 9436.37, 14583.16},      new double[] {0.00, 36.10,
72.20, 181.00, 361.00, 447.00, 535.00, 579.00, 624.00, 744.70, 1121.50},
1.0);
    Reach rcMedwayThm = new Reach(   "[R1930] North Thames, Medway Ck to Thames
River",    new double[] {0.00, 225.94, 363.43, 681.95, 1264.37, 1728.50,
1967.24, 2205.70, 2461.66, 2611.69, 4386.31},         new double[] {0.00, 44.00,
88.00, 220.00, 445.00, 591.00, 669.00, 731.00, 801.00, 815.00, 1367.70},
1.0);
    Reach rcMThmWaubuno = new Reach( "[R2030] Thames River, Middle Thames to
Waubuno Ck",  new double[] {0.00, 585.96, 913.82, 1829.99, 3183.52, 5691.37,
6608.67, 7421.55, 8496.58, 9469.84, 14140.82},       new double[] {0.00, 21.00,
41.00, 104.00, 188.00, 333.00, 387.00, 435.00, 500.00, 560.00, 870.00},
1.0);

    Reach rcByronOxbow = new Reach(  "[R2040] Thames River, Byron to Oxbow Ck",
          new double[] {0.00, 440.04, 658.15, 1007.31, 3191.60, 4271.63,
4961.46, 5716.29, 6300.11, 6857.22, 8951.47},        new double[] {0.00, 33.00,
65.00, 130.00, 651.00, 926.00, 1110.00, 1320.00, 1490.00, 1658.00, 2340.00},
1.0);
    Reach rcWaubunoEal = new Reach(  "[R2050] Thames River, Waubuno Ck to
Ealing",         new double[] {0.00, 265.55, 421.70, 908.26, 1637.43, 2468.77,
2946.06, 3412.74, 3915.06, 4339.66, 6021.55},         new double[] {0.00, 24.00,
48.00, 122.00, 224.00, 391.00, 455.00, 516.00, 580.00, 661.00, 1019.00},
1.0);
    Reach rcOxbowDing = new Reach(   "[R2120] Thames River, Oxbow Ck to Dingman
Ck",       new double[] {0.00, 736.63, 1110.68, 1721.57, 6531.27, 10598.20,
14004.89, 17214.60, 19991.89, 22381.16, 33771.60}, new double[] {0.00, 33.00,
65.00, 130.00, 651.00, 926.00, 1110.00, 1320.00, 1490.00, 1658.00, 2340.00},
1.0);
    Reach rcFishPlover = new Reach(  "[R2290] North Thames, Fish Ck to Plover
Mills",      new double[] {0.00, 230.88, 371.24, 716.78, 1233.15, 1606.14,
1852.44, 2149.65, 2376.79, 2599.42, 3319.45},         new double[] {0.00, 35.50,
71.00, 178.00, 355.00, 477.00, 562.00, 688.00, 748.00, 833.00, 1121.00},
1.0);
    Reach rcPloverFan = new Reach(   "[R2300] North Thames, Plover Mills to
Fanshawe Dam", new double[] {0.00, 522.14, 788.88, 1430.77, 2343.38, 2964.62,
3366.77, 9099.35, 9331.21, 9567.45, 10373.00},       new double[] {0.00, 37.00,
74.00, 184.00, 369.00, 496.00, 584.00, 690.00, 776.00, 864.00, 1164.00},
1.0);
    Reach rcEalNThm = new Reach(     "[R2430] Thames River, Ealing to North
Thames",       new double[] {0.00, 175.00, 344.31, 582.38, 976.73, 1361.44,
1615.79, 1885.80, 2285.38, 2656.99, 4934.06},          new double[] {0.00,
24.00, 48.00, 122.00, 224.00, 391.00, 455.00, 516.00, 580.00, 661.00, 1019.00},
    1.0);
    Reach rcNThmByron = new Reach(   "[R2440] Thames River, North Thames to
Byron",        new double[] {0.00, 691.75, 1024.01, 1820.28, 3114.94, 4105.96,
4766.86, 5373.90, 6136.58, 6776.65, 9264.95},       new double[] {0.00, 59.30,
118.60, 296.50, 593.00, 843.00, 1010.00, 1170.00, 1370.00, 1489.00, 2200.00},
1.0);


    /*
     * 3) reservoirs
     */
    Reservoir rsFanshawe = new Reservoir( "Fanshawe Dam", new double[] {0,
12350, 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}, new double[]
{0.00, 1.00, 3.00, 5.76, 5.76, 29.94, 50.55, 75.30, 103.76, 135.65, 141.80,
155.40, 167.60, 178.80, 199.00, 217.00, 234.00, 248.70, 321.00, 341.00, 365.00,
388.00, 475.00, 502.00, 530.00, 558.00, 586.00, 694.00, 763.00, 836.00, 1335.00,
1453.00}, 1.0);
    Reservoir rsWildwood = new Reservoir( "Wildwood Dam", new double[] {0, 2430,
3050, 3730, 4470, 5310, 6280, 7350, 8520, 9780, 11120, 12580, 14180, 15880,
17730, 18100, 18470, 18840, 19250, 19660, 20470, 21290, 22110, 22930, 23800,
24670},                                                 new double[] {0.00,
0.79, 0.82, 0.86, 0.89, 0.92, 0.95, 0.98, 1.01, 1.03, 1.06, 1.08, 1.10, 3.00,
3.00, 3.00, 4.33, 5.66, 7.37, 18.60, 23.55, 29.35, 35.87, 43.02, 60.66, 68.92},

1.0);
    Reservoir rsPittock = new Reservoir(  "Pittock Dam",  new double[] {0, 100,
100, 260, 470, 680, 890, 890, 1500, 2240, 3070, 4040, 5110, 6340, 7700, 7700,
9250, 9250, 10950, 12880, 14930, 14930, 17160, 17160, 18940},
                                                       new double[] {0.00, 0.40,
2.70, 2.90, 3.00, 3.20, 3.33, 5.90, 7.30, 8.50, 9.50, 10.40, 11.30, 12.10,
14.30, 27.10, 35.00, 48.10, 59.00, 72.00, 86.00, 101.12, 117.00, 180.00,
196.00},
     1.0);


    /*
     * 4) junctions
     */
    Junction jnMitchellSG = new Junction("Mitchell SG");
    Junction jnBlack = new Junction("[JR640] Black Creek");
    Junction jnAvonSG = new Junction("Avon SG");
    Junction jnAvon = new Junction("[JR750] Avon River");
    Junction jnFlat = new Junction("[JR830] Flat Creek");
    Junction jnStMarySG = new Junction("St Marys SG");
    Junction jnFish = new Junction("[JR2290] Fish Creek");
    Junction jnPloverSG = new Junction("Plover Mills SG");
    Junction jnMedwaySG = new Junction("Medway Creek SG");
    Junction jnMedway = new Junction("[JR1930] Medway Creek");

    Junction jnInnerkipSG = new Junction("Innerkip SG");
    Junction jnCedarSG = new Junction("Cedar Creek SG");
    Junction jnCedar = new Junction("[JR1840] Cedar Creek");
    Junction jnBeach = new Junction("Beachville");
    Junction jnIngSG = new Junction("Ingersoll SG");
    Junction jnReynoldsSG = new Junction("Reynolds Creek SG");
    Junction jnThmfSG = new Junction("Thamesford SG");
    Junction jnMThm = new Junction("[JR1960] Middle Thames River");
    Junction jnWaubunoSG = new Junction("Waubuno Creek SG");
    Junction jnWaubuno = new Junction("[JR2050] Waubuno Creek");
    Junction jnEalingSG = new Junction("Ealing SG");

    Junction jnNThm = new Junction("North Thames River");
    Junction jnByronSG = new Junction("Byron SG");
    Junction jnOxbowSG = new Junction("Oxbow Creek SG");
    Junction jnOxbow = new Junction("[JR2120] Oxbow Creek");
    Junction jnDingmanSG = new Junction("Dingman Creek SG");
    Junction jnDingman = new Junction("[JR2270] Dingman Creek");
    Junction jnDuttonSG = new Junction("Dutton SG");


    System.out.println("done");
    System.out.println();


    /*
     * Link model components.
     */
    System.out.println("assembling model...");

    // NORTH BRANCH
    jnMitchellSG.addInput(sbNThmWhirl);
    jnMitchellSG.addInput(sbWhirl);
    rcMitBlack.addInput(jnMitchellSG);

    jnBlack.addInput(sbNThmBlack);
    jnBlack.addInput(sbBlack);
    jnBlack.addInput(rcMitBlack);
    rcBlackAvon.addInput(jnBlack);

    jnAvonSG.addInput(sbAvon);

    jnAvon.addInput(sbNThmAvon);
    jnAvon.addInput(jnAvonSG);
    jnAvon.addInput(rcBlackAvon);
    rcAvonFlat.addInput(jnAvon);

    jnFlat.addInput(sbNThmFlat);
    jnFlat.addInput(sbFlat);
    jnFlat.addInput(rcAvonFlat);
    rcFlatTrout.addInput(jnFlat);

    rsWildwood.addInput(sbTroutWWD);
    rcWWDNThm.addInput(rsWildwood);

    jnStMarySG.addInput(sbTrout);
    jnStMarySG.addInput(rcFlatTrout);
    jnStMarySG.addInput(rcWWDNThm);
    rcTroutFish.addInput(jnStMarySG);

    jnFish.addInput(sbNThmFish);
    jnFish.addInput(sbFish);
    jnFish.addInput(rcTroutFish);
    rcFishPlover.addInput(jnFish);

    jnPloverSG.addInput(sbNThmPlover);
    jnPloverSG.addInput(rcFishPlover);
    rcPloverFan.addInput(jnPloverSG);

    rsFanshawe.addInput(sbNThmFan);
    rsFanshawe.addInput(rcPloverFan);
    rcFanMedway.addInput(rsFanshawe);

    jnMedwaySG.addInput(sbMedway);

    jnMedway.addInput(sbNThmMedway);
    jnMedway.addInput(jnMedwaySG);
    jnMedway.addInput(rcFanMedway);
    rcMedwayThm.addInput(jnMedway);

    // SOUTH BRANCH
    jnInnerkipSG.addInput(sbThmInn);

    rsPittock.addInput(sbThmPit);
    rsPittock.addInput(jnInnerkipSG);
    rcPitCedar.addInput(rsPittock);

    jnCedarSG.addInput(sbCedar);

    jnCedar.addInput(jnCedarSG);
    jnCedar.addInput(rcPitCedar);
    rcCedarBch.addInput(jnCedar);

    jnBeach.addInput(rcCedarBch);
    rcBchIng.addInput(jnBeach);

    jnIngSG.addInput(sbThmIng);
    jnIngSG.addInput(rcBchIng);
    rcIngMThm.addInput(jnIngSG);

    jnThmfSG.addInput(sbThmf);
    rcThmfThm.addInput(jnThmfSG);

    jnReynoldsSG.addInput(sbReynolds);

    jnMThm.addInput(sbMThmThm);
    jnMThm.addInput(sbThmMThm);
    jnMThm.addInput(jnReynoldsSG);
    jnMThm.addInput(rcIngMThm);
    jnMThm.addInput(rcThmfThm);
    rcMThmWaubuno.addInput(jnMThm);

    jnWaubunoSG.addInput(sbWaubuno);

    jnWaubuno.addInput(sbThmWaubuno);
    jnWaubuno.addInput(jnWaubunoSG);
    jnWaubuno.addInput(rcMThmWaubuno);
    rcWaubunoEal.addInput(jnWaubuno);

    jnEalingSG.addInput(sbThmEal);
    jnEalingSG.addInput(rcWaubunoEal);
    rcEalNThm.addInput(jnEalingSG);

    // DOWNSTREAM
    jnNThm.addInput(sbThmNThm);
    jnNThm.addInput(rcMedwayThm);
    jnNThm.addInput(rcEalNThm);
    rcNThmByron.addInput(jnNThm);

    jnByronSG.addInput(sbThmByron);
    jnByronSG.addInput(rcNThmByron);
    rcByronOxbow.addInput(jnByronSG);

    jnOxbowSG.addInput(sbOxbow);

    jnOxbow.addInput(sbThmOxbow);
    jnOxbow.addInput(jnOxbowSG);
    jnOxbow.addInput(rcByronOxbow);
    rcOxbowDing.addInput(jnOxbow);

    jnDingmanSG.addInput(sbDingman);

    jnDingman.addInput(sbThmDing);
    jnDingman.addInput(jnDingmanSG);
    jnDingman.addInput(rcOxbowDing);

    jnDuttonSG.addInput(jnDingman);


    System.out.println("done");
    System.out.println();


    /*
     * Run.
     */
    TimeHistoryFileWriter sub = new TimeHistoryFileWriter("subbasin.txt");
    TimeHistoryFileWriter jct = new TimeHistoryFileWriter("junctions.txt");
    TimeHistoryFileWriter precip = new TimeHistoryFileWriter("precip.txt");
    TimeHistoryFileWriter peak = new TimeHistoryFileWriter("peak.txt");
    System.out.println("running...");

    for(int i=0; i < metModel.getTimeSeriesLength(); i++) {

      metModel.stepTime();
      jnDuttonSG.stepTime(calendar);

      sub.writeValue(String.valueOf(sbWhirl.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmBlack.getTotalFlow()));
      sub.writeValue(String.valueOf(sbBlack.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmAvon.getTotalFlow()));
      sub.writeValue(String.valueOf(sbAvon.getTotalFlow()));
      sub.writeValue(String.valueOf(sbFlat.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmFlat.getTotalFlow()));
      sub.writeValue(String.valueOf(sbTroutWWD.getTotalFlow()));
      sub.writeValue(String.valueOf(sbTrout.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmFish.getTotalFlow()));
      sub.writeValue(String.valueOf(sbFish.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmPlover.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmFan.getTotalFlow()));
      sub.writeValue(String.valueOf(sbNThmMedway.getTotalFlow()));
      sub.writeValue(String.valueOf(sbMedway.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmInn.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmPit.getTotalFlow()));
      sub.writeValue(String.valueOf(sbCedar.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmIng.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmMThm.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmf.getTotalFlow()));
      sub.writeValue(String.valueOf(sbMThmThm.getTotalFlow()));
      sub.writeValue(String.valueOf(sbReynolds.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmWaubuno.getTotalFlow()));
      sub.writeValue(String.valueOf(sbWaubuno.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmEal.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmNThm.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmByron.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmOxbow.getTotalFlow()));
      sub.writeValue(String.valueOf(sbOxbow.getTotalFlow()));
      sub.writeValue(String.valueOf(sbThmDing.getTotalFlow()));
      sub.writeValue(String.valueOf(sbDingman.getTotalFlow()));
      sub.newLine();

      jct.writeValue(String.valueOf(jnMitchellSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnBlack.getTotalFlow()));
      jct.writeValue(String.valueOf(jnAvonSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnAvon.getTotalFlow()));
      jct.writeValue(String.valueOf(jnFlat.getTotalFlow()));
      jct.writeValue(String.valueOf(jnStMarySG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnFish.getTotalFlow()));
      jct.writeValue(String.valueOf(jnPloverSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnMedwaySG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnMedway.getTotalFlow()));
      jct.writeValue(String.valueOf(jnInnerkipSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnCedarSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnCedar.getTotalFlow()));
      jct.writeValue(String.valueOf(jnBeach.getTotalFlow()));
      jct.writeValue(String.valueOf(jnIngSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnReynoldsSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnThmfSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnMThm.getTotalFlow()));
      jct.writeValue(String.valueOf(jnWaubunoSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnWaubuno.getTotalFlow()));
      jct.writeValue(String.valueOf(jnEalingSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnNThm.getTotalFlow()));
      jct.writeValue(String.valueOf(jnByronSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnOxbowSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnOxbow.getTotalFlow()));
      jct.writeValue(String.valueOf(jnDingmanSG.getTotalFlow()));
      jct.writeValue(String.valueOf(jnDingman.getTotalFlow()));
      jct.writeValue(String.valueOf(jnDuttonSG.getTotalFlow()));
      jct.newLine();

      precip.writeValue(String.valueOf(sbNThmWhirl.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbWhirl.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmBlack.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbBlack.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmAvon.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbAvon.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbFlat.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmFlat.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbTroutWWD.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbTrout.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmFish.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbFish.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmPlover.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmFan.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbNThmMedway.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbMedway.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmInn.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmPit.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbCedar.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmIng.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmMThm.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmf.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbMThmThm.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbReynolds.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmWaubuno.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbWaubuno.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmEal.getTotalPrecip()));
```

```
      precip.writeValue(String.valueOf(sbThmNThm.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmByron.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmOxbow.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbOxbow.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbThmDing.getTotalPrecip()));
      precip.writeValue(String.valueOf(sbDingman.getTotalPrecip()));
      precip.newLine();


      calendar.add(Calendar.HOUR, (int)1.0);
    }

    peak.writeValue(String.valueOf(sbNThmWhirl.getPeakFlow()));
    peak.writeValue(String.valueOf(sbWhirl.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmBlack.getPeakFlow()));
    peak.writeValue(String.valueOf(sbBlack.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmAvon.getPeakFlow()));
    peak.writeValue(String.valueOf(sbAvon.getPeakFlow()));
    peak.writeValue(String.valueOf(sbFlat.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmFlat.getPeakFlow()));
    peak.writeValue(String.valueOf(sbTroutWWD.getPeakFlow()));
    peak.writeValue(String.valueOf(sbTrout.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmFish.getPeakFlow()));
    peak.writeValue(String.valueOf(sbFish.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmPlover.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmFan.getPeakFlow()));
    peak.writeValue(String.valueOf(sbNThmMedway.getPeakFlow()));
    peak.writeValue(String.valueOf(sbMedway.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmInn.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmPit.getPeakFlow()));
    peak.writeValue(String.valueOf(sbCedar.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmIng.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmMThm.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmf.getPeakFlow()));
    peak.writeValue(String.valueOf(sbMThmThm.getPeakFlow()));
    peak.writeValue(String.valueOf(sbReynolds.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmWaubuno.getPeakFlow()));
    peak.writeValue(String.valueOf(sbWaubuno.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmEal.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmNThm.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmByron.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmOxbow.getPeakFlow()));
    peak.writeValue(String.valueOf(sbOxbow.getPeakFlow()));
    peak.writeValue(String.valueOf(sbThmDing.getPeakFlow()));
    peak.writeValue(String.valueOf(sbDingman.getPeakFlow()));
    peak.newLine();


    System.out.println("done");

    peak.close();
//    jct.close();
//    precip.close();

  }

}
```

TimeHistoryFileReader.java

```
package cfcas.hydro.io;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.Calendar;

import cfcas.hydro.util.DSSDateFormat;
import cfcas.hydro.util.LatLong;


/**
 * TimeHistoryFileReader reads the header and data from a time series file.
 * Data is read in a stepwise manner.
 * @author Shawn Gettler
 * @date Sept 10, 2005
 */
/*
 * CHANGELOG
 * Sept 10, 2005 - SG
 *   Initial implementation using code originally in the GaugeData class.
 * Sept 13, 2005 - SG
 *   Modified constructor to count the number of values on instantiation.
 * Sept 30, 2005 - SG
 *   Added the column to read feature.
 */
public class TimeHistoryFileReader {

  protected BufferedReader reader;
  protected int col;

  // station data
  protected double lat, lon;
  protected String desc;
  protected Calendar startDate;
  protected double timeStep;

  protected int length = 0;

  /**
   * Creates a new reader from the file given and reads in the header, leaving
the reader ready to start reading data from the time series.
   * @param fileName
   * @param col data column from which to read
   */
  public TimeHistoryFileReader(String fileName, int col) {
    this.col = col;

    try {
      /*
       * Determine length of time series, assuming one line was used as a
header.
       * Creates a temporary reader and counts all non-blank lines.
       */
      BufferedReader tempReader = new BufferedReader(new FileReader(fileName));
      while(tempReader.ready()) {
        String s = tempReader.readLine();
        if(!this.stripComments(s).trim().equals("")) {
          this.length++;
        }
      }
      this.length -= 1; // header line
      /*
       * Set up the reader and read the header.
       * Assumes the first non-blank line is a valid header string.
       */
      this.reader = new BufferedReader(new FileReader(fileName));
      boolean headerRead = false;
      while(!headerRead) {
        String s = this.stripComments(this.reader.readLine());
        if(!s.equals("")) {
          String[] header = s.split(",");
          this.lat = LatLong.getDecimalDegrees(header[0].trim());
          this.lon = LatLong.getDecimalDegrees(header[1].trim());
          this.desc = header[2].trim();
          this.startDate = DSSDateFormat.parseDateString(header[3].trim());
          this.timeStep = Double.valueOf(header[4].trim()).doubleValue();

          headerRead = true;
        }
      }
    } catch (IOException exc) {
      System.out.println("  ERROR loading file " + fileName);
    }
  }


  /**
   * Get the next value from the time series file.
   * @return the next value
   */
  public double getNextValue() {
    try {
      String s = this.stripComments(this.reader.readLine()).trim();
      /*
       * If this line is blank or was commented out, read the next line instead.
       */
      if(!s.equals("")) {
        String[] line = s.split(",");
        return Double.valueOf(line[this.col]).doubleValue();
      } else {
        return this.getNextValue();
      }
    } catch (IOException exc) {
      return -1;
    }
  }


  /**
   * Returns the quadrant in which this station lies relative to the given
coordinates.
   * @param lat latitude of reference point
   * @param lon longitude of reference point
   * @return integer between 0 and 3 such that 0 refers to the NE quadrant and
1-3 follow clockwise
   */
  public int getQuadrant(double lat, double lon) {
    return LatLong.getQuadrant(lat, lon, this.lat, this.lon);
  }


  /**
   * Returns the surface distance between this station and the given
coordinates.
```

```
   * @param lat latitude of reference point
   * @param lon longitude of reference point
   * @return distance in kilometres
   */
  public double getDistance(double lat, double lon) {
    return LatLong.getSurfaceDistance(lat, lon, this.lat, this.lon);
  }


  /**
   * Returns the start date and time of this time series in a Calendar object.
   * @return start date
   */
  public Calendar getTimeSeriesStart() {
    return this.startDate;
  }


  /**
   * Returns the number of values in the time series.
   * @return length of series
   */
  public int getTimeSeriesLength() {
    return this.length;
  }


  /**
   * Returns the length of the time step for this time series.
   * @return time step in hours
   */
  public double getTimeStep() {
    return this.timeStep;
  }


  /**
   * Returns a string describing this gauge station and time series.
   * @return description
   */
  public String getDescLong() {
    String s = "  gauge: " + this.desc + " (dt: " +
String.valueOf(this.timeStep).concat("0000").substring(0,4) + "h), " +
String.valueOf(this.lat).concat("00000000").substring(0,8) + " " +
String.valueOf(this.lon).concat("00000000").substring(0,8) + ", starts " +
DSSDateFormat.getDateString(this.startDate);
    return s;
  }


  /**
   * Returns only the "name" of the station.
   * @return short description
   */
  public String getDescription() {
    return this.desc;
  }


  /*
   * Removes comments (indicated by the # character) from a String.
   */
  protected String stripComments(String s) {
    int i = s.indexOf("#");
    if(i != -1) {
      return s.substring(0, i);
    }
    return s;
  }


}
```

TimeHistoryFileWriter.java

```
package cfcas.hydro.io;

import cfcas.hydro.util.DSSDateFormat;
import cfcas.hydro.util.LatLong;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Calendar;



/**
 * TimeHistoryFileWriter writes a file in the standardized format.
 * A header provides all information about the gauge (if applicable) and about
the data written.
 * @author Shawn Gettler
 * @date Sept 10, 2005
 */
/*
 * CHANGELOG
 * Sept 10, 2005 - SG
 *   Initial implementation.
 * Sept 13, 2005 - SG
 *   Removed comments from the output of writeHeader() to reflect the fact that
any quantity can be written to this time history.
 * Sept 30, 2005 - SG
 *   Added multi-column writing support.
 */
public class TimeHistoryFileWriter {

  protected BufferedWriter writer;


  /**
   * Create a new file to which to write the time series.
   * @param fileName
   */
  public TimeHistoryFileWriter(String fileName) {
    try {
      this.writer = new BufferedWriter(new FileWriter(fileName));
    } catch (IOException exc) {
      System.out.println("ERROR creating file " + fileName);
    }
  }


  /**
   * Writes a header that can be read by a TimeHistoryFileReader.
   * @param lat latitude of the gauge (if applicable)
   * @param lon longitude of the gauge (if applicable)
   * @param desc description of the data
   * @param start start date
   * @param step time step in hours
   */
  public void writeHeader(double lat, double lon, String desc, Calendar start,
double step) {
    String header = LatLong.getDMSDegrees(lat) + "," +
LatLong.getDMSDegrees(lon) + "," + desc + "," +
DSSDateFormat.getDateString(start) + "," + String.valueOf(step);
    try {
      this.writer.write("# HEADER");
      this.writer.newLine();
      this.writer.write(header);
      this.writer.newLine();
      this.writer.write("#");
      this.writer.newLine();
      this.writer.write("# DATA");
      this.writer.newLine();
      this.writer.flush();
    } catch (IOException exc) {
      System.out.println("ERROR writing to file");
    }
  }


  /**
   * Writes a commented string to the file.
   * @param s comment text
   */
  public void writeComment(String s) {
    try {
      this.writer.write("# " + s);
      this.writer.newLine();
      this.writer.flush();
    } catch (IOException exc) {
      System.out.println("ERROR writing to file");
    }
  }



  /**
   * Writes a single value to the time series file. Used with newLine().
   * String is padded to 24 characters in case it is being written to a
multicolumn file.
   * @param d data value to write
   */
  public void writeValue(String s) {
    try {
      this.writer.write(s.concat(",                       ").substring(0,24));
      this.writer.flush();
    } catch (IOException exc) {
      System.out.println("ERROR writing to file");
    }
  }


  /**
   * Writes a line terminator.
   */
  public void newLine() {
    try {
      this.writer.newLine();
      this.writer.flush();
    } catch (IOException exc) {
      System.out.println("ERROR writing to file");
    }
  }


  /**
   * Close the file associated with this writer.
   */
  public void close() {
    try {
      this.writer.close();
    } catch (IOException exc) {
    }
  }


}
```

eventUtils.java

```
import java.io.*;

/**
 *  This class contains utilities that are needed to get the Event model running
 *
 * @author     Predrag Prodanovic
 * @created    May 25, 2006
 */
public class eventUtils {

/**
 *  This method takes in the 15 ppt files that was obtained from Karen
 * Wey and converts them to a file that Shawn Gettler's event model
 * can read.  This had to be done because Karen's previous output
 * joined a number of events together and produced multi peak events.
 * This lead to poor precipitation volume vs. discharge relationships.
 * This method works for her 'original' hourly output, or one that has
 * all events for a single station in one file.  Four columns represent
 * Year, Day of Year, Hour, and Precipitation. Zero means start of the
 * event, and -1 stands for the end of the event.  The method works for
 * extracting any event, no matter what the duration.  However, Shawn's
 * version of the Event model won't work for events that are longer than
 * 24 hrs (i.e., multi-peaked).  So, make sure when using this method with
 * Shawn's Event model, use only those events that are 24 hrs. If the
 * extracted data is to be used within HEC-HMS, any duration can be used.
 *
 * @param  year             The starting year of event to be extracted
 * @param  dayOfYear        The day of year of the event to be extracted
 * @param  inputDir         Input directory of the file to be read
 * @param  outputDir        Output directory of the file to be written
 * @param  duration         Description of the Parameter
 * @exception  IOException  Description of the Exception
 */
public static void formatSinglePeakWGEventInput(int year, int dayOfYear,
int duration, String inputDir, String outputDir) throws IOException {
//{{{

int currentYear;
int currentDayOfYear;
int currentHour;
double currentPPT;

// initializes the currentYear to 1
currentYear = 1;

String[] file = {"Blythe", "Dorchester", "Embro", "Exeter",
"Foldens", "Fullarton", "GlenAllan", "Ilderton",
"London", "StThomas", "Stratford", "Tavistock",
"Waterloo", "Woodstock", "Wroxeter"};

// to create the output files
DataWriter[] out = new DataWriter[15];
for (int i = 0; i < 15; i++) {
out[i] = new DataWriter(outputDir + "Event" + year + "-" +
file[i] + ".txt");
}

// to write the headers of the gauge files
out[0].writeData("434300, -812200, Blythe, 01 Jan 2000 08:00, 1.0");
out[1].writeData("430000, -810100, Dorchester, 01 Jan 2000 08:00, 1.0");
out[2].writeData("431500, -805500, Embro, 01 Jan 2000 08:00, 1.0");
out[3].writeData("432100, -813000, Exeter, 01 Jan 2000 08:00, 1.0");
```

```
out[4].writeData("430100, -804600, Foldens, 01 Jan 2000 08:00, 1.0");
out[5].writeData("432300, -811200, Fullarton, 01 Jan 2000 08:00, 1.0");
out[6].writeData("434000, -804300, GlenAllan, 01 Jan 2000 08:00, 1.0");
out[7].writeData("430300, -812500, Ilderton, 01 Jan 2000 08:00, 1.0");
out[8].writeData("430100, -810900, London, 01 Jan 2000 08:00, 1.0");
out[9].writeData("434600, -811200, StThomas, 01 Jan 2000 08:00, 1.0");
out[10].writeData("432200, -810000, Stratford, 01 Jan 2000 08:00, 1.0");
out[11].writeData("431900, -804900, Tavistock, 01 Jan 2000 08:00, 1.0");
out[12].writeData("432800, -803100, Waterloo, 01 Jan 2000 08:00, 1.0");
out[13].writeData("430800, -804600, Woodstock, 01 Jan 2000 08:00, 1.0");
out[14].writeData("435200, -810900, Wroxeter, 01 Jan 2000 08:00, 1.0");

// to read the input files
DataReader[] in = new DataReader[15];
for (int i = 0; i < 15; i++) {
in[i] = new DataReader(inputDir +
file[i] + ".out");
}

// initializes the line to be read
String[] line = new String[15];
for (int i = 0; i < 15; i++) {
line[i] = null;
}

// number of data points in the file to be read
//int n = in[0].countDataPoints();


// 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;

// just a blank charr array
char[] blank = new char[50];

// the index number of first separation
int indexNum1;

// the loop that reads the files and extracts the event
// counter i is for a number of data points in the file
// to be read, and j is for the number of stations

// reads in the entire line
for (int i = 0; i < 80000; i++) {
for (int j = 0; j < 15; j++) {

if (currentYear <= year) {

line[j] = in[j].readRecord();
indexNum1 = line[j].indexOf("\t");

// this if statements just skips the empty lines
if (indexNum1 > -1) {
// splits the line into four numbers
// line1, line2, line3, line4 are the
// numbers in columns 1-4
```

```
line1 = line[j];
indexNum1 = line1.indexOf("\t");
line1.getChars(0, line1.length(), charArray1, 0);

line1 = new String(charArray1, 0, indexNum1);
line[j] = new String(charArray1, indexNum1 + 1,
charArray1.length - (indexNum1 + 1));

line2 = line[j];
indexNum1 = line2.indexOf("\t");
line2.getChars(0, line2.length(), charArray2, 0);
line2 = new String(charArray2, 0, indexNum1);

line[j] = new String(charArray2, indexNum1 + 1,
charArray2.length - (indexNum1 + 1));

line3 = line[j];
indexNum1 = line3.indexOf("\t");
line3.getChars(0, line3.length(), charArray3, 0);
line3 = new String(charArray3, 0, indexNum1);

line[j] = new String(charArray3, indexNum1 + 1,
charArray3.length - (indexNum1 + 1));

line4 = line[j];

// converts from a String to a number (int or double)
currentYear = Integer.valueOf(line1).intValue();
currentDayOfYear = Integer.valueOf(line2).intValue();
currentHour = Integer.valueOf(line3).intValue();
currentPPT = Double.valueOf(line4).doubleValue();

// resets the char arrays
for (int k = 0; k < 50; k++) {
charArray1[k] = ' ';
charArray2[k] = ' ';
charArray3[k] = ' ';
charArray4[k] = ' ';

}

// dayOfYear + (duration /12) is used to keep track of
// the time when the event should be finished.  If duration
// is 24 hrs, and the event starts at 8:00am on 24th of Jan
// the event will be finished on the 25th of Jan at 7:00am.
// in this case the dayOfYear is 24, and dayOfYear + (duration /12)
// is 26.  The event finished when the currentDayOfYear is less
// than 26 (i.e., when it is 25)
if ((currentYear == year) && (currentDayOfYear >= dayOfYear)
 && (currentDayOfYear < (dayOfYear + (duration / 12)))) {

out[j].writeData(currentPPT);

}

if (currentYear > year) {

// to add 96 zeros to the end of the event
for (int q = 0; q < 96; q++) {
for (int p = 0; p < 15; p++) {
out[p].writeData(0.0);
}
}
}
```

```
}
} else {
out[j].closeFile();
}

}
}

//}}}
}


/**
 * This method interpolates the 15 ppt gauges in the Upper Thames Basin
 * and generates ppt files for each of the 33 SubBasins in the Event model
 * based upon the Inverse Distance method of HEC-HMS. This method is
 * taylored specifically to hourly event model data provided by Karen Wey.
 *
 * @param  inputDir         Directory of input files
 * @param  outputDir        Directory of output files
 * @param  eventNo          Description of the Parameter
 * @exception  IOException  Input Output Exception
 */
public static void interpolateSpacially(String inputDir, String outputDir,
int eventNo) throws IOException {
//{{{

// ppt gauges (input)
String[] inputFiles = new String[15];
inputFiles[0] = "Event" + eventNo + "-Blythe.txt";
inputFiles[1] = "Event" + eventNo + "-Dorchester.txt";
inputFiles[2] = "Event" + eventNo + "-Embro.txt";
inputFiles[3] = "Event" + eventNo + "-Exeter.txt";
inputFiles[4] = "Event" + eventNo + "-Foldens.txt";
inputFiles[5] = "Event" + eventNo + "-Fullarton.txt";
inputFiles[6] = "Event" + eventNo + "-GlenAllan.txt";
inputFiles[7] = "Event" + eventNo + "-Ilderton.txt";
inputFiles[8] = "Event" + eventNo + "-London.txt";
inputFiles[9] = "Event" + eventNo + "-StThomas.txt";
inputFiles[10] = "Event" + eventNo + "-Stratford.txt";
inputFiles[11] = "Event" + eventNo + "-Tavistock.txt";
inputFiles[12] = "Event" + eventNo + "-Waterloo.txt";
inputFiles[13] = "Event" + eventNo + "-Woodstock.txt";
inputFiles[14] = "Event" + eventNo + "-Wroxeter.txt";

// 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 = {"Event" + eventNo + "sb1PPT.csv",
"Event" + eventNo + "sb2PPT.csv",
"Event" + eventNo + "sb3PPT.csv",
"Event" + eventNo + "sb4PPT.csv",
"Event" + eventNo + "sb5PPT.csv",
"Event" + eventNo + "sb7PPT.csv",
"Event" + eventNo + "sb8PPT.csv",
"Event" + eventNo + "sb9PPT.csv",
"Event" + eventNo + "sb10PPT.csv",
"Event" + eventNo + "sb11PPT.csv",
"Event" + eventNo + "sb12PPT.csv",
"Event" + eventNo + "sb13PPT.csv",
"Event" + eventNo + "sb14PPT.csv",
"Event" + eventNo + "sb15PPT.csv",
"Event" + eventNo + "sb16PPT.csv",
"Event" + eventNo + "sb17PPT.csv",
"Event" + eventNo + "sb18PPT.csv",
"Event" + eventNo + "sb19PPT.csv",
"Event" + eventNo + "sb20PPT.csv",
"Event" + eventNo + "sb21PPT.csv",
"Event" + eventNo + "sb22PPT.csv",
"Event" + eventNo + "sb23PPT.csv",
"Event" + eventNo + "sb24PPT.csv",
"Event" + eventNo + "sb25PPT.csv",
"Event" + eventNo + "sb26PPT.csv",
"Event" + eventNo + "sb27PPT.csv",
"Event" + eventNo + "sb28PPT.csv",
"Event" + eventNo + "sb29PPT.csv",
"Event" + eventNo + "sb30PPT.csv",
"Event" + eventNo + "sb31PPT.csv",
"Event" + eventNo + "sb32PPT.csv",
"Event" + eventNo + "sb33PPT.csv",
"Event" + eventNo + "sb34PPT.csv"};

// longitudes of the SubBasins
final double[] lonSubBasins = {-81.138, -81.073, -81.202, -81.019,
-81.154, -81.050, -81.254, -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 = {43.549, 43.473, 43.419, 43.428, 43.358,
43.351, 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;

// a 2-d array that stores the distances
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
// because the first line has a header, read this and store it
// to a String---this never gets used
String header = "";

// 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; one had to be subtracted
// as the header line doesn't count towards the data points
int numDataPoints = in[0].countDataPoints() - 1;

// 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++) {

if (k == 0) {
header = in[j].readRecord();
}

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 takes in interpolated precipitation, and computes an
 * average daily ppt for St. Marys, Ealing and Byron stream gauges. It
 * can also compute the averaged storm total precipitation.
 *
 * @param  dir              The name of the directory where the
 * interpolated ppt files are
 * @param  eventNo          Description of the Parameter
 * @return                  Description of the Return Value
 * @exception  IOException  Input Output Exception
 */
public static double[] computeAvgDailyPPT(String dir, int eventNo) throws IOException {
//{{{

// the sub catchment area of each spacial unit, in km^2
double[] area = {175.982, 129.523, 47.745, 151.189, 76.82, 144,
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, 88.845, 50.486, 168.719};

// hourly ppt files generated by the method interpolateSpacially
// in the class eventUtils
final String[] inputFiles = {"Event" + eventNo + "sb1PPT.csv",
"Event" + eventNo + "sb2PPT.csv",
"Event" + eventNo + "sb3PPT.csv",
"Event" + eventNo + "sb4PPT.csv",
"Event" + eventNo + "sb5PPT.csv",
"Event" + eventNo + "sb7PPT.csv",
"Event" + eventNo + "sb8PPT.csv",
"Event" + eventNo + "sb9PPT.csv",
"Event" + eventNo + "sb10PPT.csv",
"Event" + eventNo + "sb11PPT.csv",
"Event" + eventNo + "sb12PPT.csv",
"Event" + eventNo + "sb13PPT.csv",
"Event" + eventNo + "sb14PPT.csv",
"Event" + eventNo + "sb15PPT.csv",
"Event" + eventNo + "sb16PPT.csv",
"Event" + eventNo + "sb17PPT.csv",
"Event" + eventNo + "sb18PPT.csv",
"Event" + eventNo + "sb19PPT.csv",
"Event" + eventNo + "sb20PPT.csv",
"Event" + eventNo + "sb21PPT.csv",
"Event" + eventNo + "sb22PPT.csv",
"Event" + eventNo + "sb23PPT.csv",
"Event" + eventNo + "sb24PPT.csv",
"Event" + eventNo + "sb25PPT.csv",
"Event" + eventNo + "sb26PPT.csv",
"Event" + eventNo + "sb27PPT.csv",
"Event" + eventNo + "sb28PPT.csv",
"Event" + eventNo + "sb29PPT.csv",
"Event" + eventNo + "sb30PPT.csv",
"Event" + eventNo + "sb31PPT.csv",
"Event" + eventNo + "sb32PPT.csv",
"Event" + eventNo + "sb33PPT.csv",
"Event" + eventNo + "sb34PPT.csv"};

int numSubBasins = area.length;

// DataReader objects
DataReader[] in = new DataReader[numSubBasins];

// instantiate the DataReader objects
for (int i = 0; i < numSubBasins; i++) {
in[i] = new DataReader(dir + inputFiles[i]);
}

// the number of hours in the event
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, 6, 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 Ealing stream gauge
// sb18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28
double ealingArea = area[16] + area[17] + area[18] + area[19] +
area[20] + area[21] + area[22] + area[23] + area[24] +
area[25] + area[26];

// area draining to Byron stream gauge
// all subbasins except 31, 32, 33, 31
double byronArea = totalBasinArea -
(area[32] + area[31] + area[30] + area[29]);

// to create aggregated ppt for our three locations of interest
double[] stMarysPPT = new double[n];
double[] ealingPPT = new double[n];
double[] byronPPT = new double[n];
```

```
// 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 all subbasins
ppt[j] = in[j].readCurrentData();
}
stMarysPPT[i] = (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];

ealingPPT[i] = (area[16] / ealingArea) * ppt[16] +
(area[17] / ealingArea) * ppt[17] +
(area[18] / ealingArea) * ppt[18] +
(area[19] / ealingArea) * ppt[19] +
(area[20] / ealingArea) * ppt[20] +
(area[21] / ealingArea) * ppt[21] +
(area[22] / ealingArea) * ppt[22] +
(area[23] / ealingArea) * ppt[23] +
(area[24] / ealingArea) * ppt[24] +
(area[25] / ealingArea) * ppt[25] +
(area[26] / ealingArea) * ppt[26];

byronPPT[i] = (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];
}

// to compute totalPPT, and avgDailyPPT
```

```
double stMarysTotalPPT = 0.0;
double ealingTotalPPT = 0.0;
double byronTotalPPT = 0.0;

double stMarysAvgDailyPPT = 0.0;
double ealingAvgDailyPPT = 0.0;
double byronAvgDailyPPT = 0.0;

// Karen Wey added 96 extra hours at the end of each event
// these will be ignored in the computation of the avg daily ppt

n = n - 96;

for (int i = 0; i < n; i++) {
stMarysTotalPPT = stMarysTotalPPT + stMarysPPT[i];
ealingTotalPPT = ealingTotalPPT + ealingPPT[i];
byronTotalPPT = byronTotalPPT + byronPPT[i];
}

// the average daily ppt
/*
 *  stMarysAvgDailyPPT = stMarysTotalPPT / (n / 24);
 *  ealingAvgDailyPPT = ealingTotalPPT / (n / 24);
 *  byronAvgDailyPPT = byronTotalPPT / (n / 24);
 */
// the storm total to duration ratio
stMarysAvgDailyPPT = stMarysTotalPPT / n;
ealingAvgDailyPPT = ealingTotalPPT / n;
byronAvgDailyPPT = byronTotalPPT / n;

double[] out = new double[3];
out[0] = stMarysTotalPPT;
out[1] = ealingTotalPPT;
out[2] = byronTotalPPT;

return out;
//}}}
}


/**
 *  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;
//}}}
}


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

```
}


/**
 *  Takes in a Shawn Gettler's event model main file, and replicates it
 * for every event for the scenario.  It also creates the .bat file
 * needed to execute the event model 100 times.
 *
 * @exception  IOException  Description of the Exception
 */
public static void createAllMainFiles(String id) throws IOException {
//{{{
// read the original file and store it in an array of String
DataReader in = new DataReader("testEventModel" + id +".java");
int n = in.countDataPoints();
String[] file = new String[n];

for (int i = 0; i < n; i++) {
file[i] = in.readRecord();
}

// declare the 100 output files
DataWriter[] out = new DataWriter[100];

for (int i = 0; i < 100; i++) {
out[i] = new DataWriter("testEventModel" + id + (i + 1) + ".java");

for (int j = 0; j < n; j++) {

if (j == 22) {
out[i].writeData("public class testEventModel" + id + (i + 1) + " {");
} else if (j == 35) {
out[i].writeData("int eventNo = " + (i + 1) + ";");
} else {

out[i].writeData(file[j]);
}
}
}
for (int i = 0; i < 100; i++) {
out[i].closeFile();
}

// to create the .bat file
DataWriter outBatFile = new DataWriter("run" + id + ".bat");

for (int i = 0; i < 100; i++){
outBatFile.writeData("javac testEventModel" + id + (i+1) + ".java");
outBatFile.writeData("java testEventModel" + id + (i+1));
outBatFile.writeData("");
}

outBatFile.closeFile();

//}}}
}

}
```

mainEvent.java

```
import java.io.*;

/**
 *  Description of the Class
 *
 * @author     Predrag Prodanovic
 * @created    May 28, 2006
 */
public class mainEvent {
/**
 *  The main program for the mainEventFormatting class
 *
 * @param  args             The command line arguments
 * @exception  IOException  Description of the Exception
 */
public static void main(String[] args) throws IOException {

// This is the code that converts Karen's output and makes it
// ready for use in HEC-HMS; No longer needed
//{{{
/*
String masterDir = "c:\\JavaContinuous\\Data\\Hourly\\" +
"WGScenarios\\B21\\";

// and create the files needed to run the event model with HEC-HMS
//{{{
// this is for the B21 Scenario
int[] eventNo = {18, 50, 80, 97, 129, 167, 171, 197, 222, 247,
282, 298, 333, 360, 392, 405, 421, 457, 477, 505,
535, 556, 564, 601, 623, 641, 680, 706, 725, 738,
767, 789, 818, 843, 866, 895, 928, 967, 997, 1026,
1041, 1067, 1083, 1117, 1142, 1167, 1197, 1220, 1233, 1281,
1301, 1330, 1359, 1371, 1397, 1416, 1441, 1479, 1492, 1537,
1564, 1574, 1614, 1629, 1644, 1666, 1712, 1719, 1757, 1787,
1809, 1841, 1871, 1892, 1908, 1942, 1957, 1976, 2021, 2031,
2053, 2067, 2102, 2118, 2150, 2165, 2197, 2213, 2234, 2262,
2295, 2337, 2340, 2385, 2429, 2437, 2480, 2511, 2517, 2549};


int numEvents = eventNo.length;
for (int i = 0; i < numEvents; i++) {
eventUtils.formatWGEventInput(eventNo[i], masterDir);
}
//}}}

// to compute aggregate ppt for each event
//{{{

double[] out = new double[3];

DataWriter fileOut = new DataWriter(masterDir + "stormTotalPPT.txt");
fileOut.writeData("Event, StMarys, Ealing, Byron");

for (int i = 0; i < eventNo.length; i++) {

// to call the interpolate spacially method for the events
String inputDir = "c:\\JavaContinuous\\Data\\Hourly\\" +
"WGScenarios\\B21\\Event" + eventNo[i] + "\\";
String outputDir = "c:\\JavaContinuous\\Data\\Hourly\\" +
"WGScenarios\\B21\\Event" + eventNo[i] + "\\";

//eventUtils.interpolateSpacially(inputDir, outputDir, eventNo[i]);
out = eventUtils.computeAvgDailyPPT(outputDir);

fileOut.writeData(eventNo[i] + ", " + out[0] + ", " + out[1] + ", " + out[2]);

}
fileOut.closeFile();

//}}}

// to compute the statistics of the event flows
//{{{
contUtils.fitStatisticalDistribution(masterDir,
"ByronStorms.txt", "Gumbel");
contUtils.fitStatisticalDistribution(masterDir,
"StMarysStorms.txt", "Gumbel");
contUtils.fitStatisticalDistribution(masterDir,
"EalingStorms.txt", "Gumbel");
//}}}
*/
//}}}

// formatting the event model
//{{{{
/*
// starting day of year of each storm event
int[] dayOfYearHistoric = {213, 250, 334, 54, 173, 122, 135, 212,
227, 270, 68, 203, 192, 100, 194, 343, 250, 299, 208, 267,
110, 265, 266, 278, 332, 269, 316, 315, 164, 90, 258, 99,
38, 116, 144, 159, 64, 185, 271, 328, 107, 351, 192, 268,
167, 202, 257, 266, 130, 275, 104, 281, 180, 133, 347,
142, 243, 272, 185, 260, 276, 304, 287, 136, 95, 168,
260, 242, 256, 210, 173, 163, 255, 229, 357, 317, 204,
238, 205, 98, 187, 51, 104, 270, 103, 32, 199, 79, 302,
253, 282, 206, 57, 106, 282, 271, 310, 215, 311, 205};

int[] dayOfYearB21 = {346,51,92,233,217,247,7,50,33,157,
251,169,268,135,57,106,57,159,141,186,
254,256,104,252,292,206,257,148,112,138,
244,160,177,273,164,112,167,55,247,317,
113,214,207,255,178,106,171,269,220,86,
175,255,107,132,143,338,161,87,111,271,
105,140,250,160,263,134,158,96,258,193,
106,194,49,157,250,178,132,6,245,99,
193,193,109,238,202,181,264,257,185,129,
110,289,139,199,339,308,161,236,98,120};

int[] dayOfYearB11 = {172,189,264,38,246,74,302,145,212,286,
129,318,139,283,141,308,275,139,130,177,
314,179,269,290,248,309,236,262,173,282,
284,345,347,243,323,178,284,135,192,232,
241,220,234,230,233,291,192,184,166,58,
219,196,262,105,342,186,203,233,240,196,
212,148,270,349,220,252,184,143,232,163,
134,231,49,232,171,188,210,296,222,278,
146,262,138,316,143,252,37,324,137,257,
55,216,337,315,278,161,163,285,181,141};

// parameters needed for extracting event data from Karen Wey's
// disaggregated data
int[] year = new int[100];
for (int i = 0; i < 100; i++) {
year[i] = i + 1;
}
```

```
// the duration of the original event
int duration = 24;


String scenario = "B11";
// directories
String inputDir = "c:\\JavaEvent\\Data\\Hourly\\" +
"WGScenarios\\" + scenario + "\\";
String extractedDir = "c:\\JavaEvent\\Data\\Hourly\\" +
"WGScenarios\\" + scenario + "\\ExtractedEvents\\";
String interpolatedDir = "c:\\JavaEvent\\Data\\Hourly\\" +
"WGScenarios\\" + scenario + "\\InterpolatedEvents\\";

// output file of storm totals
DataWriter outFile = new DataWriter(inputDir + "StormTotals.txt");
outFile.writeData("Event, St. Marys, Ealing, Byron");

// to declare an array for outputting the storm totals
double[] out = new double[3];

for (int i = 0; i < 99; i++) {
// extract the event from the file
//eventUtils.formatSinglePeakWGEventInput(year[i],
// dayOfYearB11[i], duration, inputDir, extractedDir);

// to interpolate the gauges
// this calculation is now done within Shawn's file; there is
// no difference between Shawn's interpolation and mine!
//eventUtils.interpolateSpacially(extractedDir,
//interpolatedDir, year[i]);

// to compute aggregated ppt
out = eventUtils.computeAvgDailyPPT(interpolatedDir, year[i]);

// to write the data
outFile.writeData((i+1) + ", " + out[0] + ", " + out[1] + ", " + out[2]);
}

// to create the main and bat files
//eventUtils.createAllMainFiles("B11");

// to compute the statistics of the flows
eventUtils.fitStatisticalDistribution(inputDir, "StMarysPeaksB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(inputDir, "StMarysPeaksB21.txt", "LP3");

eventUtils.fitStatisticalDistribution(inputDir, "FanshawePeaksB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(inputDir, "FanshawePeaksB21.txt", "LP3");

eventUtils.fitStatisticalDistribution(inputDir, "MedwayPeaksB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(inputDir, "MedwayPeaksB21.txt", "LP3");

eventUtils.fitStatisticalDistribution(inputDir, "EalingPeaksB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(inputDir, "EalingPeaksB21.txt", "LP3");

eventUtils.fitStatisticalDistribution(inputDir, "ByronPeaksB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(inputDir, "ByronPeaksB21.txt", "LP3");


// to compute the statistics of the ppt
String dir = "c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\B21\\AnalysisPlots\\";

eventUtils.fitStatisticalDistribution(dir, "StMarysStormTotalsB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(dir, "EalingStormTotalsB21.txt", "Gumbel");
eventUtils.fitStatisticalDistribution(dir, "ByronStormTotalsB21.txt", "Gumbel");
```

```
*/

// to close the file
//outFile.closeFile();
//}}}

DataWriter jt = new DataWriter("combine.bat");

jt.writeDataLn("gs -dBATCH -dNOPAUSE -q -sDEVICE=pdfwrite -sOutputFile=finished.pdf nlReader.pdf ");

for (int i = 1; i <= 200; i++){
if(i < 10){
jt.writeDataLn("00" + i + "nlReader.pdf ");
}else if((i >= 10) && (i <= 99) ){
jt.writeDataLn("0" + i + "nlReader.pdf ");
}else{
jt.writeDataLn(i + "nlReader.pdf ");
}
}

jt.closeFile();



}
}
```

testEventModelHistoric.java

```
import cfcas.hydro.io.TimeHistoryFileReader;
import cfcas.hydro.io.TimeHistoryFileWriter;
import cfcas.hydro.met.MetModel;
import cfcas.hydro.met.PrecipDataSource;
import cfcas.hydro.phys.EventSubBasin;
import cfcas.hydro.phys.Junction;
import cfcas.hydro.phys.Reach;
import cfcas.hydro.phys.Reservoir;
import cfcas.hydro.util.DSSDateFormat;
import cfcas.hydro.util.LatLong;
import java.util.Calendar;

import java.io.*;

/**
 * @author     Shawn Gettler
 * @created    July 11, 2006
 * @date
 */
/*
 *  CHANGELOG
 */
public class testEventModelHistoric {

/**
 *  The main program for the testEventModelHistoric class
 *
 * @param  arg              The command line arguments
 * @exception  IOException  Description of the Exception
 */
public static void main(String[] arg) throws IOException {

System.out.println("UPPER THAMES RIVER BASIN MODEL");
System.out.println();

int eventNo = 1;

// for the calculation of peak flow
double peakStMarys = 0.0;
double peakPloverMills = 0.0;
double peakFanshawe = 0.0;
double peakMedway = 0.0;
double peakIngersoll = 0.0;
double peakEaling = 0.0;
double peakByron = 0.0;

String previousFile;

if (eventNo == 1) {
previousFile = "";
} else {
previousFile = String.valueOf(eventNo - 1);
}

// open the file of flow peaks, and read all of the previous
items in it
DataReader in = new
DataReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\FloodPeaks" +
previousFile + ".txt");

// this is the new peak file
DataWriter peaksFile = new
```

```
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\FloodPeaks" +
eventNo + ".txt");

int numLinesInFile = in.countDataPoints();
for (int i = 0; i < numLinesInFile; i++) {
peaksFile.writeData(in.readRecord());
}

/*
 *  Set up meteorologic model.
 */
MetModel metModel = new MetModel();
System.out.println("PPT data for Event" + eventNo + "...");

metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Blythe.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Dorchester.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Embro.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Exeter.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Foldens.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Fullarton.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-GlenAllan.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Ilderton.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-London.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-StThomas.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Stratford.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Tavistock.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Waterloo.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Woodstock.txt", 0)));
metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
ExtractedEvents\\Event" + eventNo + "-Wroxeter.txt", 0)));

/*
 *  metModel.addPrecipDataSource(new PrecipDataSource(new
TimeHistoryFileReader("c:\\JavaHECHMS\\univ_precip25mm.txt", 0)));
 */
```

```
System.out.println("done");
System.out.println();

Calendar calendar = Calendar.getInstance();
calendar.setTime(metModel.getTimeSeriesStart().getTime());

/*
 *  Set up physical basin model.
 *  1) subbasins
 */
System.out.println("initializing event model components...");
EventSubBasin sbNThmWhirl = new EventSubBasin(metModel, "[01]
North Thames above Whirl Ck", LatLong.getDecimalDegrees("433306"),
LatLong.getDecimalDegrees("-810856"), 175.982, 5.0, 1.0, 0.0, 8.0, 10.0, 0.010,
0.4, 0.70);
EventSubBasin sbWhirl = new EventSubBasin(metModel, "[02] Whirl
Creek", LatLong.getDecimalDegrees("432931"),
LatLong.getDecimalDegrees("-810334"), 129.523, 5.0, 1.0, 0.0, 10.0, 12.0, 0.010,
0.4, 0.70);
EventSubBasin sbNThmBlack = new EventSubBasin(metModel, "[03]
North Thames above Black Ck", LatLong.getDecimalDegrees("432410"),
LatLong.getDecimalDegrees("-811303"), 47.745, 5.0, 1.1, 0.0, 12.0, 6.0, 0.010,
0.4, 0.40);
EventSubBasin sbBlack = new EventSubBasin(metModel, "[04] Black
Creek", LatLong.getDecimalDegrees("432550"),
LatLong.getDecimalDegrees("-810056"), 151.189, 5.0, 1.0, 0.0, 12.0, 10.0, 0.010,
0.4, 0.50);
EventSubBasin sbNThmAvon = new EventSubBasin(metModel, "[05]
North Thames above Avon R", LatLong.getDecimalDegrees("432128"),
LatLong.getDecimalDegrees("-810915"), 76.820, 5.0, 1.1, 0.0, 7.0, 6.0, 0.010,
0.4, 0.40);

// basin #6
EventSubBasin sbAvon = new EventSubBasin(metModel, "[07] Avon
River", LatLong.getDecimalDegrees("432232"),
LatLong.getDecimalDegrees("-805910"), 144.000, 5.0, 1.0, 2.0, 5.0, 10.0, 0.010,
0.4, 0.10);
EventSubBasin sbFlat = new EventSubBasin(metModel, "[08] Flat
Creek", LatLong.getDecimalDegrees("432228"),
LatLong.getDecimalDegrees("-811522"), 88.355, 5.0, 1.0, 0.0, 11.0, 7.0, 0.010,
0.4, 0.40);
EventSubBasin sbNThmFlat = new EventSubBasin(metModel, "[09]
North Thames above Flat Ck", LatLong.getDecimalDegrees("431837"),
LatLong.getDecimalDegrees("-810356"), 78.476, 5.0, 1.1, 0.0, 7.0, 6.0, 0.010,
0.4, 0.40);
EventSubBasin sbTroutWWD = new EventSubBasin(metModel, "[10]
Trout Creek above Wildwood", LatLong.getDecimalDegrees("432013"),
LatLong.getDecimalDegrees("-805626"), 141.118, 5.0, 1.1, 0.0, 13.0, 9.0, 0.010,
0.5, 0.40);

EventSubBasin sbTrout = new EventSubBasin(metModel, "[11] Trout
Creek", LatLong.getDecimalDegrees("431637"),
LatLong.getDecimalDegrees("-810546"), 28.942, 5.0, 1.2, 0.0, 9.0, 5.0, 0.010,
0.3, 0.40);
EventSubBasin sbNThmFish = new EventSubBasin(metModel, "[12]
North Thames above Fish Ck", LatLong.getDecimalDegrees("431459"),
LatLong.getDecimalDegrees("-811204"), 35.466, 5.0, 1.3, 0.0, 10.0, 8.0, 0.010,
0.3, 0.40);
EventSubBasin sbFish = new EventSubBasin(metModel, "[13] Fish
Creek", LatLong.getDecimalDegrees("431852"),
LatLong.getDecimalDegrees("-811726"), 153.721, 5.0, 1.0, 0.0, 13.0, 14.0, 0.010,
0.3, 0.40);
EventSubBasin sbNThmPlover = new EventSubBasin(metModel, "[14]
North Thames above Plover Mills", LatLong.getDecimalDegrees("431215"),
LatLong.getDecimalDegrees("-810913"), 84.539, 5.0, 1.5, 0.0, 14.0, 10.0, 0.010,
0.3, 0.40);
EventSubBasin sbNThmFan = new EventSubBasin(metModel, "[15]
North Thames above Fanshawe", LatLong.getDecimalDegrees("431057"),
LatLong.getDecimalDegrees("-810526"), 94.198, 5.0, 2.0, 0.0, 15.0, 20.0, 0.010,
0.3, 0.40);

EventSubBasin sbNThmMedway = new EventSubBasin(metModel, "[16]
North Thames above Medway Ck", LatLong.getDecimalDegrees("430345"),
LatLong.getDecimalDegrees("-811438"), 75.363, 5.0, 2.0, 5.0, 16.0, 16.0, 0.010,
0.5, 0.30);
EventSubBasin sbMedway = new EventSubBasin(metModel, "[17]
Medway Creek", LatLong.getDecimalDegrees("430837"),
LatLong.getDecimalDegrees("-811659"), 202.478, 5.0, 2.3, 0.0, 24.0, 20.0, 0.010,
0.5, 0.10);
EventSubBasin sbThmInn = new EventSubBasin(metModel, "[18]
Thames River above Innerkip", LatLong.getDecimalDegrees("431707"),
LatLong.getDecimalDegrees("-804854"), 148.318, 5.0, 1.0, 0.0, 10.0, 9.0, 0.010,
0.4, 0.80);
EventSubBasin sbThmPit = new EventSubBasin(metModel, "[19]
Thames River above Pittock Dam", LatLong.getDecimalDegrees("431140"),
LatLong.getDecimalDegrees("-804529"), 96.840, 5.0, 1.1, 0.0, 15.0, 9.0, 0.010,
0.3, 0.80);
EventSubBasin sbCedar = new EventSubBasin(metModel, "[20] Cedar
Creek", LatLong.getDecimalDegrees("430405"),
LatLong.getDecimalDegrees("-804407"), 97.910, 5.0, 1.0, 0.0, 2.0, 3.0, 0.010,
0.5, 0.10);

EventSubBasin sbThmIng = new EventSubBasin(metModel, "[21]
Thames River above Ingersoll", LatLong.getDecimalDegrees("430209"),
LatLong.getDecimalDegrees("-804853"), 170.704, 5.0, 1.3, 0.0, 24.0, 12.0, 0.010,
0.6, 0.50);
EventSubBasin sbThmMThm = new EventSubBasin(metModel, "[22]
Thames River above Middle Thames", LatLong.getDecimalDegrees("430226"),
LatLong.getDecimalDegrees("-805457"), 42.859, 5.0, 1.3, 0.0, 24.0, 9.0, 0.010,
0.6, 0.50);
EventSubBasin sbThmf = new EventSubBasin(metModel, "[23] Middle
Thames above Thamesford", LatLong.getDecimalDegrees("431000"),
LatLong.getDecimalDegrees("-805400"), 291.080, 5.0, 3.5, 0.0, 7.0, 17.0, 0.010,
0.5, 0.20);
EventSubBasin sbMThmThm = new EventSubBasin(metModel, "[24]
Middle Thames above Thames River", LatLong.getDecimalDegrees("430302"),
LatLong.getDecimalDegrees("-805723"), 35.861, 5.0, 1.4, 0.0, 25.0, 8.0, 0.010,
0.5, 0.20);
EventSubBasin sbReynolds = new EventSubBasin(metModel, "[25]
Reynolds Creek", LatLong.getDecimalDegrees("425718"),
LatLong.getDecimalDegrees("-805245"), 165.973, 5.0, 2.0, 0.0, 25.0, 15.0, 0.010,
0.7, 0.70);

EventSubBasin sbThmWaubuno = new EventSubBasin(metModel, "[26]
Thames River above Waubuno Ck", LatLong.getDecimalDegrees("430007"),
LatLong.getDecimalDegrees("-810229"), 120.935, 5.0, 1.3, 0.0, 20.0, 10.0, 0.010,
0.6, 0.50);
EventSubBasin sbWaubuno = new EventSubBasin(metModel, "[27]
Waubuno Creek", LatLong.getDecimalDegrees("430613"),
LatLong.getDecimalDegrees("-810447"), 104.945, 5.0, 3.5, 0.0, 15.0, 16.0, 0.010,
0.5, 0.10);
EventSubBasin sbThmEal = new EventSubBasin(metModel, "[28]
Thames River above Ealing", LatLong.getDecimalDegrees("430200"),
LatLong.getDecimalDegrees("-810826"), 61.195, 5.0, 1.4, 0.0, 20.0, 8.0, 0.010,
0.6, 0.50);
EventSubBasin sbThmNThm = new EventSubBasin(metModel, "[29]
Thames River above North Thames", LatLong.getDecimalDegrees("425745"),
LatLong.getDecimalDegrees("-811302"), 22.556, 5.0, 2.2, 40.0, 4.0, 6.0, 0.010,
0.5, 0.30);
EventSubBasin sbThmByron = new EventSubBasin(metModel, "[30]
Thames River above Byron", LatLong.getDecimalDegrees("425911"),
LatLong.getDecimalDegrees("-811814"), 30.002, 5.0, 2.3, 30.0, 7.0, 10.0, 0.010,
0.5, 0.30);

EventSubBasin sbThmOxbow = new EventSubBasin(metModel, "[31]
Thames River above Oxbow Ck", LatLong.getDecimalDegrees("425938"),
LatLong.getDecimalDegrees("-812034"), 32.409, 5.0, 2.2, 0.0, 6.0, 6.0, 0.010,
0.5, 0.30);
EventSubBasin sbOxbow = new EventSubBasin(metModel, "[32] Oxbow
Creek", LatLong.getDecimalDegrees("430134"),
LatLong.getDecimalDegrees("-812207"), 88.845, 5.0, 4.0, 0.0, 40.0, 14.0, 0.010,
0.6, 0.40);
EventSubBasin sbThmDing = new EventSubBasin(metModel, "[33]
Thames River above Dingman Ck", LatLong.getDecimalDegrees("425635"),
LatLong.getDecimalDegrees("-812737"), 50.486, 5.0, 2.4, 0.0, 8.0, 7.0, 0.010,
0.5, 0.30);
EventSubBasin sbDingman = new EventSubBasin(metModel, "[34]
Dingman Creek", LatLong.getDecimalDegrees("425520"),
LatLong.getDecimalDegrees("-811357"), 168.719, 5.0, 3.0, 2.0, 12.0, 8.0, 0.010,
0.3, 0.05);

/*
 *  2) reaches
 */
Reach rcPitCedar = new Reach("[R111] Thames River, Pittock Dam
to Cedar Ck", new double[]{0.00, 50.50, 86.16, 196.43, 329.36, 580.92, 673.32,
740.40, 794.03, 1058.11, 1655.93}, new double[]{0.00, 4.40, 8.80, 21.60, 34.00,
74.00, 83.50, 90.00, 93.50, 130.00, 211.00});
Reach rcCedarBch = new Reach("[R222] Thames River, Cedar Ck to
Beachville", new double[]{0.00, 95.13, 156.33, 378.48, 690.30, 1162.93, 1378.07,
1642.92, 1800.79, 2213.88, 3732.20}, new double[]{0.00, 7.00, 14.00, 35.00,
70.00, 107.00, 125.00, 140.00, 156.00, 200.00, 307.20});
Reach rcBchIng = new Reach("[R333] Thames River, Beachville to
Ingersoll", new double[]{0.00, 107.93, 169.42, 311.18, 513.11, 655.22, 733.83,
809.86, 876.07, 906.99, 1542.14}, new double[]{0.00, 10.80, 21.60, 54.00,
108.00, 154.00, 179.00, 204.00, 223.00, 239.00, 414.00});
Reach rcMitBlack = new Reach("[R560] North Thames, Mitchell to
Black Ck", new double[]{0.00, 389.67, 657.93, 1421.86, 2607.09, 3508.72,
3860.86, 5109.41, 5650.00, 5984.53, 7505.39}, new double[]{0.00, 20.70, 41.40,
104.00, 207.00, 292.00, 355.00, 445.00, 508.00, 550.00, 768.00});
Reach rcBlackAvon = new Reach("[R640] North Thames, Black Ck to
Avon R", new double[]{0.00, 271.15, 440.87, 924.18, 1663.83, 2256.23, 2499.10,
3106.65, 3440.88, 3621.61, 4737.15}, new double[]{0.00, 25.30, 50.50, 126.00,
253.00, 356.00, 426.00, 534.00, 610.00, 660.00, 937.00});
Reach rcAvonFlat = new Reach("[R750] North Thames, Avon R to
Flat Ck", new double[]{0.00, 104.90, 177.99, 377.73, 668.41, 876.97, 1028.77,
1280.78, 1332.32, 1391.07, 1649.61}, new double[]{0.00, 27.50, 55.00, 138.00,
270.00, 380.00, 461.00, 578.00, 660.00, 715.00, 1021.00});
Reach rcFlatTrout = new Reach("[R900] North Thames, Flat Ck to
Trout Ck", new double[]{0.00, 182.90, 297.24, 603.72, 1058.99, 1369.71, 1578.41,
1866.99, 2093.57, 2184.92, 2782.47}, new double[]{0.00, 31.90, 64.00, 159.00,
345.00, 487.00, 566.00, 657.00, 716.00, 785.00, 1183.00});

Reach rcWWDNThm = new Reach("[R930] Trout Creek, Wildwood Dam to
North Thames", new double[]{0.00, 67.00, 95.00, 115.00, 138.17, 208.56, 319.53,
543.70, 1132.43, 3021.35, 5241.65}, new double[]{0.00, 3.00, 5.00, 7.00, 10.00,
20.00, 30.00, 50.00, 100.00, 151.00, 180.00});
Reach rcTroutFish = new Reach("[R1010] North Thames, Trout Ck to
Fish Ck", new double[]{0.00, 309.23, 503.96, 985.75, 1726.00, 2222.59, 2558.16,
2945.41, 3234.45, 3522.67, 4514.07}, new double[]{0.00, 33.50, 67.00, 168.00,
335.00, 450.00, 530.00, 630.00, 705.00, 784.00, 1057.00});
Reach rcIngMThm = new Reach("[R1870] Thames River, Ingersoll to
Middle Thames", new double[]{0.00, 264.88, 474.80, 1321.19, 1810.16, 3810.08,
4278.23, 4675.72, 5832.38, 6291.38, 8737.16}, new double[]{0.00, 13.20, 26.40,
67.00, 120.00, 214.00, 249.00, 280.00, 320.00, 360.00, 572.00});
Reach rcThmfThm = new Reach("[R1890] Middle Thames, Thamesford
to Thames River", new double[]{0.00, 404.07, 543.70, 958.47, 1558.60, 2047.74,
2327.52, 2642.25, 2881.41, 3097.13, 3846.54}, new double[]{0.00, 9.10, 18.10,
45.30, 90.00, 133.00, 159.00, 190.00, 214.00, 236.00, 314.00});
Reach rcFanMedway = new Reach("[R1910] North Thames, Fanshawe
Dam to Medway Ck", new double[]{0.00, 837.87, 1279.23, 2592.06, 4971.75,
6159.72, 7120.41, 7712.24, 8615.57, 9436.37, 14583.16}, new double[]{0.00,
36.10, 72.20, 181.00, 361.00, 447.00, 535.00, 579.00, 624.00, 744.70, 1121.50});
Reach rcMedwayThm = new Reach("[R1930] North Thames, Medway Ck
to Thames River", new double[]{0.00, 225.94, 363.43, 681.95, 1264.37, 1728.50,
1967.24, 2205.70, 2461.66, 2611.69, 4386.31}, new double[]{0.00, 44.00, 88.00,
220.00, 445.00, 591.00, 669.00, 731.00, 801.00, 815.00, 1367.70});
Reach rcMThmWaubuno = new Reach("[R2030] Thames River, Middle
Thames to Waubuno Ck", new double[]{0.00, 585.96, 913.82, 1829.99, 3183.52,
5691.37, 6608.67, 7421.55, 8496.58, 9469.84, 14140.82}, new double[]{0.00,
21.00, 41.00, 104.00, 188.00, 333.00, 387.00, 435.00, 500.00, 560.00, 870.00});

Reach rcByronOxbow = new Reach("[R2040] Thames River, Byron to
Oxbow Ck", new double[]{0.00, 440.04, 658.15, 1007.31, 3191.60, 4271.63,
4961.46, 5716.29, 6300.11, 6857.22, 8951.47}, new double[]{0.00, 33.00, 65.00,
130.00, 651.00, 926.00, 1110.00, 1320.00, 1490.00, 1658.00, 2340.00});
Reach rcWaubunoEal = new Reach("[R2050] Thames River, Waubuno Ck
to Ealing", new double[]{0.00, 265.55, 421.70, 908.26, 1637.43, 2468.77,
2946.06, 3412.74, 3915.06, 4339.66, 6021.55}, new double[]{0.00, 24.00, 48.00,
122.00, 224.00, 391.00, 455.00, 516.00, 580.00, 661.00, 1019.00});
Reach rcOxbowDing = new Reach("[R2120] Thames River, Oxbow Ck to
Dingman Ck", new double[]{0.00, 736.63, 1110.68, 1721.57, 6531.27, 10598.20,
14004.89, 17214.60, 19991.89, 22381.16, 33771.60}, new double[]{0.00, 33.00,
65.00, 130.00, 651.00, 926.00, 1110.00, 1320.00, 1490.00, 1658.00, 2340.00});
Reach rcFishPlover = new Reach("[R2290] North Thames, Fish Ck to
Plover Mills", new double[]{0.00, 230.88, 371.24, 716.78, 1233.15, 1606.14,
1852.44, 2149.65, 2376.79, 2599.42, 3319.45}, new double[]{0.00, 35.50, 71.00,
178.00, 355.00, 477.00, 562.00, 688.00, 748.00, 833.00, 1121.00});
Reach rcPloverFan = new Reach("[R2300] North Thames, Plover
Mills to Fanshawe Dam", new double[]{0.00, 522.14, 788.88, 1430.77, 2343.38,
2964.62, 3366.77, 9099.35, 9331.21, 9567.45, 10373.00}, new double[]{0.00,
37.00, 74.00, 184.00, 369.00, 496.00, 584.00, 690.00, 776.00, 864.00, 1164.00});
Reach rcEalNThm = new Reach("[R2430] Thames River, Ealing to
North Thames", new double[]{0.00, 175.00, 344.31, 582.38, 976.73, 1361.44,
1615.79, 1885.80, 2285.38, 2656.99, 4934.06}, new double[]{0.00, 24.00, 48.00,
122.00, 224.00, 391.00, 455.00, 516.00, 580.00, 661.00, 1019.00});
Reach rcNThmByron = new Reach("[R2440] Thames River, North
Thames to Byron", new double[]{0.00, 691.75, 1024.01, 1820.28, 3114.94, 4105.96,
4766.86, 5373.90, 6136.58, 6776.65, 9264.95}, new double[]{0.00, 59.30, 118.60,
296.50, 593.00, 843.00, 1010.00, 1170.00, 1370.00, 1489.00, 2200.00});

/*
 *  3) reservoirs
 */
Reservoir rsFanshawe = new Reservoir("Fanshawe Dam", new
double[]{0, 12350, 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},
new double[]{0.00, 1.00, 3.00, 5.76, 5.76, 29.94, 50.55, 75.30, 103.76, 135.65,
141.80, 155.40, 167.60, 178.80, 199.00, 217.00, 234.00, 248.70, 321.00, 341.00,
365.00, 388.00, 475.00, 502.00, 530.00, 558.00, 586.00, 694.00, 763.00, 836.00,
1335.00, 1453.00});
Reservoir rsWildwood = new Reservoir("Wildwood Dam", new
double[]{0, 2430, 3050, 3730, 4470, 5310, 6280, 7350, 8520, 9780, 11120, 12580,
14180, 15880, 17730, 18100, 18470, 18840, 19250, 19660, 20470, 21290, 22110,
22930, 23800, 24670}, new double[]{0.00, 0.79, 0.82, 0.86, 0.89, 0.92, 0.95,
0.98, 1.01, 1.03, 1.06, 1.08, 1.10, 3.00, 3.00, 3.00, 4.33, 5.66, 7.37, 18.60,
23.55, 29.35, 35.87, 43.02, 60.66, 68.92});
Reservoir rsPittock = new Reservoir("Pittock Dam", new
double[]{0, 100, 100, 260, 470, 680, 890, 890, 1500, 2240, 3070, 4040, 5110,
6340, 7700, 7700, 9250, 9250, 10950, 12880, 14930, 14930, 17160, 17160, 18940},
new double[]{0.00, 0.40, 2.70, 2.90, 3.00, 3.20, 3.33, 5.90, 7.30, 8.50, 9.50,
10.40, 11.30, 12.10, 14.30, 27.10, 35.00, 48.10, 59.00, 72.00, 86.00, 101.12,
117.00, 180.00, 196.00});

/*
 *  4) junctions
 */
Junction jnMitchellSG = new Junction("Mitchell SG");
Junction jnBlack = new Junction("[JR640] Black Creek");
Junction jnAvonSG = new Junction("Avon SG");
Junction jnAvon = new Junction("[JR750] Avon River");
Junction jnFlat = new Junction("[JR830] Flat Creek");
Junction jnStMarySG = new Junction("St Marys SG");
Junction jnFish = new Junction("[JR2290] Fish Creek");
Junction jnPloverSG = new Junction("Plover Mills SG");
Junction jnMedwaySG = new Junction("Medway Creek SG");
Junction jnMedway = new Junction("[JR1930] Medway Creek");

Junction jnInnerkipSG = new Junction("Innerkip SG");
Junction jnCedarSG = new Junction("Cedar Creek SG");
Junction jnCedar = new Junction("[JR1840] Cedar Creek");
Junction jnBeach = new Junction("Beachville");
Junction jnIngSG = new Junction("Ingersoll SG");
Junction jnReynoldsSG = new Junction("Reynolds Creek SG");
Junction jnThmfSG = new Junction("Thamesford SG");
Junction jnMThm = new Junction("[JR1960] Middle Thames River");
Junction jnWaubunoSG = new Junction("Waubuno Creek SG");
Junction jnWaubuno = new Junction("[2050] Waubuno Creek");
Junction jnEalingSG = new Junction("Ealing SG");

Junction jnNThm = new Junction("North Thames River");
Junction jnByronSG = new Junction("Byron SG");
Junction jnOxbowSG = new Junction("Oxbow Creek SG");
Junction jnOxbow = new Junction("[2120] Oxbow Creek");
Junction jnDingmanSG = new Junction("Dingman Creek SG");
Junction jnDingman = new Junction("[2270] Dingman Creek");
Junction jnDuttonSG = new Junction("Dutton SG");

System.out.println("done");
System.out.println();

/*
 *  Link model components.
 */
System.out.println("assembling model...");

// NORTH BRANCH
jnMitchellSG.addInput(sbNThmWhirl);
jnMitchellSG.addInput(sbWhirl);
rcMitBlack.addInput(jnMitchellSG);

jnBlack.addInput(sbNThmBlack);
jnBlack.addInput(sbBlack);
jnBlack.addInput(rcMitBlack);
rcBlackAvon.addInput(jnBlack);

jnAvonSG.addInput(sbAvon);

jnAvon.addInput(sbNThmAvon);
jnAvon.addInput(jnAvonSG);
jnAvon.addInput(rcBlackAvon);
rcAvonFlat.addInput(jnAvon);

jnFlat.addInput(sbNThmFlat);
jnFlat.addInput(sbFlat);
jnFlat.addInput(rcAvonFlat);
rcFlatTrout.addInput(jnFlat);

rsWildwood.addInput(sbTroutWWD);
rcWWDNThm.addInput(rsWildwood);

jnStMarySG.addInput(sbTrout);
jnStMarySG.addInput(rcFlatTrout);
jnStMarySG.addInput(rcWWDNThm);
rcTroutFish.addInput(jnStMarySG);

jnFish.addInput(sbNThmFish);
jnFish.addInput(sbFish);
jnFish.addInput(rcTroutFish);
rcFishPlover.addInput(jnFish);

jnPloverSG.addInput(sbNThmPlover);
jnPloverSG.addInput(rcFishPlover);
rcPloverFan.addInput(jnPloverSG);

rsFanshawe.addInput(sbNThmFan);
rsFanshawe.addInput(rcPloverFan);
rcFanMedway.addInput(rsFanshawe);

jnMedwaySG.addInput(sbMedway);

jnMedway.addInput(sbNThmMedway);
jnMedway.addInput(jnMedwaySG);
jnMedway.addInput(rcFanMedway);
rcMedwayThm.addInput(jnMedway);

// SOUTH BRANCH
jnInnerkipSG.addInput(sbThmInn);

rsPittock.addInput(sbThmPit);
rsPittock.addInput(jnInnerkipSG);
rcPitCedar.addInput(rsPittock);

jnCedarSG.addInput(sbCedar);

jnCedar.addInput(jnCedarSG);
jnCedar.addInput(rcPitCedar);
rcCedarBch.addInput(jnCedar);

jnBeach.addInput(rcCedarBch);
rcBchIng.addInput(jnBeach);

jnIngSG.addInput(sbThmIng);
jnIngSG.addInput(rcBchIng);
rcIngMThm.addInput(jnIngSG);

jnThmfSG.addInput(sbThmf);
rcThmfThm.addInput(jnThmfSG);

jnReynoldsSG.addInput(sbReynolds);

jnMThm.addInput(sbMThmThm);
jnMThm.addInput(sbThmMThm);
jnMThm.addInput(jnReynoldsSG);
jnMThm.addInput(rcIngMThm);
jnMThm.addInput(rcThmfThm);
rcMThmWaubuno.addInput(jnMThm);

jnWaubunoSG.addInput(sbWaubuno);

jnWaubuno.addInput(sbThmWaubuno);
jnWaubuno.addInput(jnWaubunoSG);
jnWaubuno.addInput(rcMThmWaubuno);
rcWaubunoEal.addInput(jnWaubuno);

jnEalingSG.addInput(sbThmEal);
jnEalingSG.addInput(rcWaubunoEal);
rcEalNThm.addInput(jnEalingSG);

// DOWNSTREAM
jnNThm.addInput(sbThmNThm);
jnNThm.addInput(rcMedwayThm);
jnNThm.addInput(rcEalNThm);
rcNThmByron.addInput(jnNThm);

jnByronSG.addInput(sbThmByron);
jnByronSG.addInput(rcNThmByron);
rcByronOxbow.addInput(jnByronSG);

jnOxbowSG.addInput(sbOxbow);

jnOxbow.addInput(sbThmOxbow);
jnOxbow.addInput(jnOxbowSG);
jnOxbow.addInput(rcByronOxbow);
rcOxbowDing.addInput(jnOxbow);

jnDingmanSG.addInput(sbDingman);

jnDingman.addInput(sbThmDing);
jnDingman.addInput(jnDingmanSG);
jnDingman.addInput(rcOxbowDing);

jnDuttonSG.addInput(jnDingman);

System.out.println("done");
System.out.println();

/*
 *  Run.
 */
TimeHistoryFileWriter sub = new
TimeHistoryFileWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
Results\\subbasins" + eventNo + ".txt");
TimeHistoryFileWriter jct = new
TimeHistoryFileWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
Results\\junctions" + eventNo + ".txt");
//TimeHistoryFileWriter precip = new
TimeHistoryFileWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
Results\\precip" + eventNo + ".txt");

// Pat's custom output file
TimeHistoryFileWriter out = new
TimeHistoryFileWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
Results\\EventOutput" + eventNo + ".txt");

// interpolated output files
DataWriter sb1PPT = new
```

```
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb1PPT.csv");
DataWriter sb2PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb2PPT.csv");
DataWriter sb3PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb3PPT.csv");
DataWriter sb4PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb4PPT.csv");
DataWriter sb5PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb5PPT.csv");
DataWriter sb7PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb7PPT.csv");
DataWriter sb8PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb8PPT.csv");
DataWriter sb9PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb9PPT.csv");
DataWriter sb10PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb10PPT.csv");
DataWriter sb11PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb11PPT.csv");
DataWriter sb12PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb12PPT.csv");
DataWriter sb13PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb13PPT.csv");
DataWriter sb14PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb14PPT.csv");
DataWriter sb15PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb15PPT.csv");
DataWriter sb16PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb16PPT.csv");
DataWriter sb17PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb17PPT.csv");
DataWriter sb18PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb18PPT.csv");
DataWriter sb19PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb19PPT.csv");
DataWriter sb20PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb20PPT.csv");
DataWriter sb21PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb21PPT.csv");
DataWriter sb22PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb22PPT.csv");
DataWriter sb23PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb23PPT.csv");
DataWriter sb24PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb24PPT.csv");
DataWriter sb25PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb25PPT.csv");
DataWriter sb26PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb26PPT.csv");
DataWriter sb27PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb27PPT.csv");
DataWriter sb28PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb28PPT.csv");
DataWriter sb29PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb29PPT.csv");
DataWriter sb30PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb30PPT.csv");
DataWriter sb31PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb31PPT.csv");
DataWriter sb32PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb32PPT.csv");
DataWriter sb33PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb33PPT.csv");
DataWriter sb34PPT = new
DataWriter("c:\\JavaEvent\\Data\\Hourly\\WGScenarios\\Historic\\
InterpolatedEvents\\Event" + eventNo + "sb34PPT.csv");

// to write the headers
out.writeValue("Time");
out.writeValue("StMarysSG");
out.writeValue("PloverMillsSG");
out.writeValue("Fanshawe");
out.writeValue("MedwaySG");
out.writeValue("IngersollSG");
out.writeValue("EalingSG");
out.writeValue("ByronSG");
out.newLine();

System.out.println("running...");
for (int i = 0; i < metModel.getTimeSeriesLength(); i++) {

metModel.stepTime();
jnDuttonSG.stepTime(1.0);

sub.writeValue(String.valueOf(i));

sub.writeValue(String.valueOf(sbNThmWhirl.getTotalFlow()));
sub.writeValue(String.valueOf(sbWhirl.getTotalFlow()));

sub.writeValue(String.valueOf(sbNThmBlack.getTotalFlow()));
sub.writeValue(String.valueOf(sbBlack.getTotalFlow()));

sub.writeValue(String.valueOf(sbNThmAvon.getTotalFlow()));
sub.writeValue(String.valueOf(sbAvon.getTotalFlow()));
sub.writeValue(String.valueOf(sbFlat.getTotalFlow()));
sub.writeValue(String.valueOf(sbNThmFlat.getTotalFlow()));

sub.writeValue(String.valueOf(sbTroutWWD.getTotalFlow()));
sub.writeValue(String.valueOf(sbTrout.getTotalFlow()));

sub.writeValue(String.valueOf(sbNThmFish.getTotalFlow()));
sub.writeValue(String.valueOf(sbFish.getTotalFlow()));

sub.writeValue(String.valueOf(sbNThmPlover.getTotalFlow()));

sub.writeValue(String.valueOf(sbNThmFan.getTotalFlow()));

sub.writeValue(String.valueOf(sbNThmMedway.getTotalFlow()));
sub.writeValue(String.valueOf(sbMedway.getTotalFlow()));
sub.writeValue(String.valueOf(sbThmInn.getTotalFlow()));
sub.writeValue(String.valueOf(sbThmPit.getTotalFlow()));
sub.writeValue(String.valueOf(sbCedar.getTotalFlow()));
sub.writeValue(String.valueOf(sbThmIng.getTotalFlow()));

sub.writeValue(String.valueOf(sbThmMThm.getTotalFlow()));
sub.writeValue(String.valueOf(sbThmf.getTotalFlow()));

sub.writeValue(String.valueOf(sbMThmThm.getTotalFlow()));

sub.writeValue(String.valueOf(sbReynolds.getTotalFlow()));

sub.writeValue(String.valueOf(sbThmWaubuno.getTotalFlow()));

sub.writeValue(String.valueOf(sbWaubuno.getTotalFlow()));
sub.writeValue(String.valueOf(sbThmEal.getTotalFlow()));

sub.writeValue(String.valueOf(sbThmNThm.getTotalFlow()));

sub.writeValue(String.valueOf(sbThmByron.getTotalFlow()));

sub.writeValue(String.valueOf(sbThmOxbow.getTotalFlow()));
sub.writeValue(String.valueOf(sbOxbow.getTotalFlow()));

sub.writeValue(String.valueOf(sbThmDing.getTotalFlow()));

sub.writeValue(String.valueOf(sbDingman.getTotalFlow()));
sub.newLine();

jct.writeValue(String.valueOf(i));

jct.writeValue(String.valueOf(jnMitchellSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnBlack.getTotalFlow()));
jct.writeValue(String.valueOf(jnAvonSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnAvon.getTotalFlow()));
jct.writeValue(String.valueOf(jnFlat.getTotalFlow()));

jct.writeValue(String.valueOf(jnStMarySG.getTotalFlow()));
jct.writeValue(String.valueOf(jnFish.getTotalFlow()));

jct.writeValue(String.valueOf(jnPloverSG.getTotalFlow()));

jct.writeValue(String.valueOf(jnMedwaySG.getTotalFlow()));
jct.writeValue(String.valueOf(jnMedway.getTotalFlow()));

jct.writeValue(String.valueOf(jnInnerkipSG.getTotalFlow()));

jct.writeValue(String.valueOf(jnCedarSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnCedar.getTotalFlow()));
jct.writeValue(String.valueOf(jnBeach.getTotalFlow()));
jct.writeValue(String.valueOf(jnIngSG.getTotalFlow()));

jct.writeValue(String.valueOf(jnReynoldsSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnThmfSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnMThm.getTotalFlow()));

jct.writeValue(String.valueOf(jnWaubunoSG.getTotalFlow()));

jct.writeValue(String.valueOf(jnWaubuno.getTotalFlow()));

jct.writeValue(String.valueOf(jnEalingSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnNThm.getTotalFlow()));

jct.writeValue(String.valueOf(jnByronSG.getTotalFlow()));

jct.writeValue(String.valueOf(jnOxbowSG.getTotalFlow()));
jct.writeValue(String.valueOf(jnOxbow.getTotalFlow()));

jct.writeValue(String.valueOf(jnDingmanSG.getTotalFlow()));

jct.writeValue(String.valueOf(jnDingman.getTotalFlow()));

jct.writeValue(String.valueOf(jnDuttonSG.getTotalFlow()));
jct.newLine();

// PP custom output is here
out.writeValue(String.valueOf(i));

out.writeValue(String.valueOf(jnStMarySG.getTotalFlow()));

out.writeValue(String.valueOf(jnPloverSG.getTotalFlow()));

out.writeValue(String.valueOf(rsFanshawe.getTotalFlow()));

out.writeValue(String.valueOf(jnMedwaySG.getTotalFlow()));
out.writeValue(String.valueOf(jnIngSG.getTotalFlow()));

out.writeValue(String.valueOf(jnEalingSG.getTotalFlow()));

out.writeValue(String.valueOf(jnByronSG.getTotalFlow()));
out.newLine();

// to output interpolated ppt
sb1PPT.writeData(String.valueOf(sbNThmWhirl.getTotalPrecip()));
sb2PPT.writeData(String.valueOf(sbWhirl.getTotalPrecip()));
sb3PPT.writeData(String.valueOf(sbNThmBlack.getTotalPrecip()));
sb4PPT.writeData(String.valueOf(sbBlack.getTotalPrecip()));
sb5PPT.writeData(String.valueOf(sbNThmAvon.getTotalPrecip()));
sb7PPT.writeData(String.valueOf(sbAvon.getTotalPrecip()));
sb8PPT.writeData(String.valueOf(sbFlat.getTotalPrecip()));
sb9PPT.writeData(String.valueOf(sbNThmFlat.getTotalPrecip()));
sb10PPT.writeData(String.valueOf(sbTroutWWD.getTotalPrecip()));
sb11PPT.writeData(String.valueOf(sbTrout.getTotalPrecip()));
sb12PPT.writeData(String.valueOf(sbNThmFish.getTotalPrecip()));
sb13PPT.writeData(String.valueOf(sbFish.getTotalPrecip()));
sb14PPT.writeData(String.valueOf(sbNThmPlover.getTotalPrecip()));
sb15PPT.writeData(String.valueOf(sbNThmFan.getTotalPrecip()));
sb16PPT.writeData(String.valueOf(sbNThmMedway.getTotalPrecip()));
sb17PPT.writeData(String.valueOf(sbMedway.getTotalPrecip()));
sb18PPT.writeData(String.valueOf(sbThmInn.getTotalPrecip()));
sb19PPT.writeData(String.valueOf(sbThmPit.getTotalPrecip()));
sb20PPT.writeData(String.valueOf(sbCedar.getTotalPrecip()));
sb21PPT.writeData(String.valueOf(sbThmIng.getTotalPrecip()));
sb22PPT.writeData(String.valueOf(sbThmMThm.getTotalPrecip()));
```

```
sb23PPT.writeData(String.valueOf(sbThmf.getTotalPrecip()));
sb24PPT.writeData(String.valueOf(sbMThmThm.getTotalPrecip()));
sb25PPT.writeData(String.valueOf(sbReynolds.getTotalPrecip()));
sb26PPT.writeData(String.valueOf(sbThmWaubuno.getTotalPrecip()));
sb27PPT.writeData(String.valueOf(sbWaubuno.getTotalPrecip()));
sb28PPT.writeData(String.valueOf(sbThmEal.getTotalPrecip()));
sb29PPT.writeData(String.valueOf(sbThmNThm.getTotalPrecip()));
sb30PPT.writeData(String.valueOf(sbThmByron.getTotalPrecip()));
sb31PPT.writeData(String.valueOf(sbThmOxbow.getTotalPrecip()));
sb32PPT.writeData(String.valueOf(sbOxbow.getTotalPrecip()));
sb33PPT.writeData(String.valueOf(sbThmDing.getTotalPrecip()));
sb34PPT.writeData(String.valueOf(sbDingman.getTotalPrecip()));

calendar.add(Calendar.HOUR, (int) 1.0);

// to compute the peak flows
if (jnStMarySG.getTotalFlow() > peakStMarys) {
peakStMarys = jnStMarySG.getTotalFlow();
}
if (jnPloverSG.getTotalFlow() > peakPloverMills) {
peakPloverMills = jnPloverSG.getTotalFlow();
}
if (rsFanshawe.getTotalFlow() > peakFanshawe) {
peakFanshawe = rsFanshawe.getTotalFlow();
}
if (jnMedwaySG.getTotalFlow() > peakMedway) {
peakMedway = jnMedwaySG.getTotalFlow();
}
if (jnIngSG.getTotalFlow() > peakIngersoll) {
peakIngersoll = jnIngSG.getTotalFlow();
}
if (jnEalingSG.getTotalFlow() > peakEaling) {
peakEaling = jnEalingSG.getTotalFlow();
}
if (jnByronSG.getTotalFlow() > peakByron) {
peakByron = jnByronSG.getTotalFlow();
}
}
System.out.println("done");

// write all the current peaks
peaksFile.writeData(peakStMarys + ", " + peakPloverMills + ", "
+
peakFanshawe + ", " + peakMedway + ", " + peakIngersoll
+ ", " +
peakEaling + ", " + peakByron);

// to close the files
sub.close();
jct.close();
//precip.close();
out.close();
sb1PPT.closeFile();
sb2PPT.closeFile();
sb3PPT.closeFile();
sb4PPT.closeFile();
sb5PPT.closeFile();
sb7PPT.closeFile();
sb8PPT.closeFile();
sb9PPT.closeFile();
sb10PPT.closeFile();
sb11PPT.closeFile();
sb12PPT.closeFile();
sb13PPT.closeFile();
```

```
sb14PPT.closeFile();
sb15PPT.closeFile();
sb16PPT.closeFile();
sb17PPT.closeFile();
sb18PPT.closeFile();
sb19PPT.closeFile();
sb20PPT.closeFile();
sb21PPT.closeFile();
sb22PPT.closeFile();
sb23PPT.closeFile();
sb24PPT.closeFile();
sb25PPT.closeFile();
sb26PPT.closeFile();
sb27PPT.closeFile();
sb28PPT.closeFile();
sb29PPT.closeFile();
sb30PPT.closeFile();
sb31PPT.closeFile();
sb32PPT.closeFile();
sb33PPT.closeFile();
sb34PPT.closeFile();

// this is a DataReader object
peaksFile.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);
}


/**
 *  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 writeDataLn(String dataPoint) {
this.FileOut.print(dataPoint);
}
/**
 *  A method to simply close the file
 */
public void closeFile() {
(this.FileOut).close();
}
}
```

ModelDate.java

```
// all these are needed for the date
import java.text.SimpleDateFormat;
import java.text.ParseException;
import java.util.Date;
import java.text.DateFormat;
import java.util.GregorianCalendar;

/**
 *  A class that keeps track of simulation time
 *
 * @author     Predrag Prodanovic
 * @created    February 12, 2006
 */
public class ModelDate {

// instance variables
private int year;
private int month;
private int day;
private int hour;
private int minute;

// these get computed by the constructor
private SimpleDateFormat dateFormat;
private GregorianCalendar cal;
private Date currentDate;


/**
 *Constructor for the ModelDate object
 *
 * @param  year    Starting year
 * @param  month   Starting month
 * @param  day     Starting day
 * @param  hour    Starting hour
 * @param  minute  Starting minute
 */
public ModelDate(int year, int month, int day,
int hour, int minute) {

this.year = year;
this.month = month;
this.day = day;
this.hour = hour;
this.minute = minute;

this.dateFormat = new SimpleDateFormat("dd MMM yyyy HH:mm");
this.cal = new GregorianCalendar(year, month, day, hour, minute);
this.currentDate = cal.getTime();
}


/**
 *  Gets the year attribute of the ModelDate object
 *
 * @return    The year value
 */
public int getYear() {
return this.cal.get(GregorianCalendar.YEAR);
}
```

```
/**
 *  Gets the month attribute of the ModelDate object
 *
 * @return    The month value
 */
public int getMonth() {
return this.cal.get(GregorianCalendar.MONTH);
}


/**
 *  Gets the day attribute of the ModelDate object
 *
 * @return    The day value
 */
public int getDay() {
return this.cal.get(GregorianCalendar.DAY_OF_MONTH);
}


/**
 *  Gets the dayOfYear attribute of the ModelDate object
 *
 * @return    The dayOfYear value
 */
public int getDayOfYear() {
return this.cal.get(GregorianCalendar.DAY_OF_YEAR);
}


/**
 *  Gets the dayOfWeek attribute of the ModelDate object
 *
 * @return    The dayOfWeek value
 */
public int getDayOfWeek() {
return this.cal.get(GregorianCalendar.DAY_OF_WEEK);
}


/**
 *  Gets the weekOfYear attribute of the ModelDate object
 *
 * @return    The weekOfYear value
 */
public int getWeekOfYear() {
return this.cal.get(GregorianCalendar.WEEK_OF_YEAR);
}



/**
 *  Gets the hour attribute of the ModelDate object
 *
 * @return    The hour value
 */
public int getHour() {
return this.cal.get(GregorianCalendar.HOUR_OF_DAY);
}


/**
 *  Gets the daysInMonth attribute of the ModelDate object.
 * Note that this method calls the method getMonth() from above
 *
 * @return    The daysInMonth value
 */
public int getDaysInMonth() {

int[] daysInMonths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
if (this.cal.isLeapYear(getYear())) {
daysInMonths[1] = 29;
}
// no need to put this.getMonth() because getMonth() already uses this
return (daysInMonths[getMonth()]);
}


/**
 *  Gets the daysInYear attribute of the ModelDate object
 *
 * @return    The daysInYear value
 */
public int getDaysInYear() {

int daysInYear;
if (this.cal.isLeapYear(getYear())) {
daysInYear = 366;
} else {
daysInYear = 365;
}
return daysInYear;
}


/**
 *  Gets the date attribute of the ModelDate object
 *
 * @return    The date value
 */
public String getDate() {
return this.dateFormat.format(this.currentDate);
}


/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByMonths(int val) {
this.cal.add(GregorianCalendar.MONTH, val);
this.currentDate = this.cal.getTime();
}


/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByDays(int val) {
this.cal.add(GregorianCalendar.DATE, val);
this.currentDate = this.cal.getTime();

}
```

```
/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByHours(int val) {
this.cal.add(GregorianCalendar.HOUR, val);
this.currentDate = this.cal.getTime();
}


/**
 *  Description of the Method
 *
 * @param  val  Description of the Parameter
 */
public void incrementDateByMinutes(int val) {
this.cal.add(GregorianCalendar.MINUTE, val);
this.currentDate = this.cal.getTime();
}

}
```