THE UNIVERSITY OF WESTERN ONTARIO
DEPARTMENT OF CIVIL AND
ENVIRONMENTAL ENGINEERING

Water Resources Research Report

Tools for Downscaling Climate Variables: A Technical Manual

By:
Sohom Mandal
Patrick A. Breach
Abhishek Gaur
and
Slobodan P. Simonovic

Report No: 097
Date: April 2017

ISSN: (print) 1913-3200; (online) 1913-3219
ISBN: (print) 978-0-7714-3135-7; (online) 978-0-7714-3136-4

Tools for Downscaling Climate Variables:

A Technical Manual

By

Sohom Mandal

Patrick Breach

Abhishek Gaur

and

Slobodon P. Simonovic

Department of Civil and Environmental Engineering

Western University, Canada

April 2017

Executive Summary

Climate impacts on humans, nature and world economics are major research challenges of today. Climate and freshwater resources are integrated with each other in a significant way so any change in one may influence the other. Continuous rise of global temperature has a significant impact on freshwater resources. At present, one of the key objectives of climate change research is to build adequate knowledge regarding future climate impacts and how to connect climate impacts with adaptation actions.

Generally, impacts of climate change on regional water resources are assessed for future climate scenarios obtained from Global Climate Model (GCM) simulations. GCMs represent the state of the art with respect to the simulation of global climate variables in response to emission scenarios of greenhouse gasses. GCMs can satisfactorily model smoothly varying fields such as mean sea level pressure, but often fail to capture non-smooth fields such as precipitation (Hughes and Guttorp, 1994). In addition, the spatial scale of GCM output is very coarse (>100 km^2). Therefore, downscaling of coarsely gridded GCM data is necessary in order to capture the impacts of climate change on hydro-meteorological variables (e.g. temperature, precipitation, soil moisture) at a regional scale.

In this document we discuss the data preparation and computational implementation details for multiple downscaling models which are developed at the Facility for Intelligent Decision Support (FIDS) research lab, the University of Western Ontario. In this manual four downscaling tools are presented in details with the computer code and example applications. They are (i) K-nearest neighbor (K-NN CAD) weather generator; (ii) maximum entropy based (MBEWG) weather generator; (iii) statistical downscaling model based on beta regression (BR); and (iv) physical scaling (SP) method.

The remainder of the manual is organized as follows. Section 1 introduces the spatial interpolation technique used and its user interface. In section 2, a brief background of the change factor methodology and its user interface are provided. Section 3 provides information about the K-NN CAD weather generator model where details about MBEWG are given in section 4. Detailed information regarding BR based statistical downscaling model and physical scaling model are given in sections 5 and 6, respectively.

Table of Contents

List of Tables

List of Figures

1 Introduction

Climate change due to greenhouse gas (GHG) emissions is impacting the global hydrological cycle as well as regional hydrology across the world, and will continue in the future. Precipitation is directly affected due to an increase in global average temperature. Changes in temperature and precipitation are already having significant impacts on ecosystems, agriculture and water resources activities and will magnify in future. Due to significant changes in temperature and precipitation patterns in past decades, climate scientists and engineers are expressing interest in future temperature and precipitation projections under changing climate conditions.

Generally, the outputs of global climate models (GCMs) are used for regional climate change impact assessment. GCMs simulate time series of global climate variables (e.g. sea level pressure, temperature, specific humidity) considering different GHGs emission scenarios. GCM outputs are coarsely gridded (>100 km^2) and often fail to capture non-smooth fields such as precipitation (Hughes and Guttorp, 1994). Spatial downscaling is required for better understanding and assessment of future hydrologic conditions at watershed scales under different climate change scenarios. Spatial downscaling (SD) translates large scale climate variables simulated by GCMs to a regional scale. Downscaling methods are broadly classified as dynamic or statistical. Dynamic downscaling is based on nesting a finer scale regional climate model (RCM) (up to 10 km x 10 km horizontal resolution) within GCMs. Statistical downscaling (SD) methods use parametric/nonparametric and/or linear/nonlinear relationships between predictor and predictand variables (Wilby and Wigley, 1997). SD methods developed so far can be classified into three groups: (a) classification/ weather typing methods (Hay *et al.*, 1991; Hughes and Guttorp, 1994); (b) regression/transfer function methods (Goyal and Ojha, 2010; Kannan and Ghosh, 2010;

Mandal *et al.*, 2016) and (c) weather generators (WG) (Eum and Simonovic, 2012; Srivastav and Simonovic, 2014; King *et al.*, 2015).

King *et al.*, (2015) developed a K-nearest neighbor (K-NN CAD V4) based weather generator where Srivastav and Simonovic (2014) developed a multisite, multivariate weather generator using maximum entropy bootstrap (MBEWG). In addition to these, Mandal *et al.*,(2016) developed a multisite beta regression (BR) based statistical downscaling models to downscale precipitation data. Last but not the least Gaur and Simonovic (2016a) developed a physical scaling model for downscaling climate data. All these models are included in this report. As the GCM outputs have coarse spatial resolutions, the GCM data has to be spatially interpolated for the station coordinates before temporal downscaling. The details about spatial downscaling are given in the following section.

The main objective of this document is to provide technical details (including programming code) of statistical downscaling models developed by the Facility for Intelligent Decision Support (FIDS). Source code for two weather generators (K-nearest neighbor and maximum entropy bootstrap), regression based (beta regression) model and physical scaling downscaling model are provided in the appendices of this report. In addition, digital files, as well as sample input and output data, for the four downscaling tools are available in the “downscaling” folder of the online repository FIDS-UWO/climate.

2 Spatial Interpolation of GCM Data

As the output of different GCMs have different spatial resolutions, the climate data extracted from GCMs are spatially interpolated before downscaling. The inverse distance weighting (IDW) technique is used for spatial downscaling following Shepard (1968). This is a local, deterministic spatial interpolation technique which follows the first law of geography in that observations located nearby to one another are likely to be more similar. This is implemented by using distance-decay function to compute a weighted spatial average of selected points surrounding the interpolated point of interest. The distance-decay function used in this report defines the weighting for a particular station to be inversely proportional to the square of the distance between the station and the nearest set of GCM grid data points. The weight associated (W_j) with four nearest grid points to a particular station (v_i) can be calculated using the following equation:

$$W_j = \frac{\frac{1}{d_j^2}}{\frac{1}{d_1^2}+\frac{1}{d_2^2}+\frac{1}{d_3^2}+\frac{1}{d_4^2}} \qquad \textbf{(2.1)}$$

$$v_i\left(t\right) = \sum_{j=1}^{4} W_j \times v_j\left(t\right) \qquad \textbf{(2.2)}$$

where d_1, d_2, d_3 and d_4 are the distances between the station (v_i) and the four nearest grid points; v_j is climate variable value from the grid points and $v_i(t)$ is the sum of weighted average for a particular time t. Eq (2.1) is used for calculating the weighted average of climate variable for the station v_i. A hypothetical example (Srivastav *et al.*, 2016) of spatial precipitation interpolation using inverse distance method is given below.

In this example, the observation station lies within four grid points (Figure 0.1). In the first step, we calculate the weights using inverse distance method using Eq (2.1).

$$W_1 = \frac{\frac{1}{d_1^2}}{\frac{1}{d_1^2}+\frac{1}{d_2^2}+\frac{1}{d_3^2}+\frac{1}{d_4^2}} = \frac{\frac{1}{8^2}}{\frac{1}{8^2}+\frac{1}{5^2}+\frac{1}{10^2}+\frac{1}{7^2}} = 0.167$$

$$W_2 = \frac{\frac{1}{d_2^2}}{\frac{1}{d_1^2}+\frac{1}{d_2^2}+\frac{1}{d_3^2}+\frac{1}{d_4^2}} = \frac{\frac{1}{5^2}}{\frac{1}{8^2}+\frac{1}{5^2}+\frac{1}{10^2}+\frac{1}{7^2}} = 0.428$$

$$W_3 = \frac{\frac{1}{d_3^2}}{\frac{1}{d_1^2}+\frac{1}{d_2^2}+\frac{1}{d_3^2}+\frac{1}{d_4^2}} = \frac{\frac{1}{10^2}}{\frac{1}{8^2}+\frac{1}{5^2}+\frac{1}{10^2}+\frac{1}{7^2}} = 0.107$$

$$W_4 = \frac{\frac{1}{d_4^2}}{\frac{1}{d_1^2}+\frac{1}{d_2^2}+\frac{1}{d_3^2}+\frac{1}{d_4^2}} = \frac{\frac{1}{7^2}}{\frac{1}{8^2}+\frac{1}{5^2}+\frac{1}{10^2}+\frac{1}{7^2}} = 0.297$$

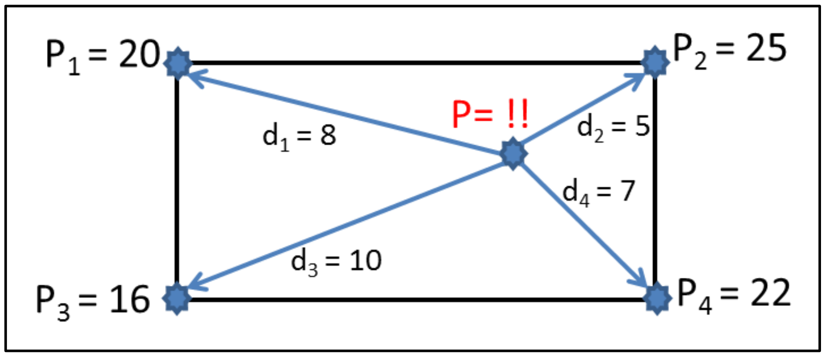


Figure 0.1 Hypothetical example of spatial interpolation

In the final step we calculate the spatially interpolated precipitation using the above weights with Eq (2.2). Using $W_1 \ldots W_4$ in Eq (2.2), we obtain the interpolated value of precipitation P at the desired location.

$$
\begin{aligned}
P &= P_1 w_1 + P_2 w_2 + P_3 w_3 + P_4 w_4 \\
&= (20 \times 0.167) + (25 \times 0.428) + (16 \times 0.107) + (22 \times 0.297) \\
&= 22.30
\end{aligned}
$$

2.1 User Interface and Source Code for IDW

In the previous section, we discussed the mathematical equations behind IDW and provided a hypothetical example. In the present section, we are providing details of a graphical user interface for IDW that was developed using Python. The code and interface are available here: FIDS-UWO/climate/downscaling.

In order to run any of the Python code discussed and included in this report, it is recommended to download a scientific Python distribution with Python 3.6 or greater such as Anaconda or Enthought Canopy. This will include pre-compiled packages for high-performance arrays supporting linear algebra routines, data analysis tools, and netCDF reading and writing capabilities. The 3rd party packages used in this work are listed in Table 0.1.

Table 0.1 3rd party Python packages used to implement the inverse distance interpolation, climate scaling, and KNN-CAD algorithms.

Package	Version	Usage
numpy	1.11.3	High performance arrays, linear algebra, and matrix manipulation
pandas	0.19.2	CSV file reading and data preparation
numba	0.30.1	JIT compilation used to accelerate numpy-based code
libnetcdf	4.3.3.1	C++ library for reading and writing netCDF
netCDF4	1.2.7	Python wrapper for libnetCDF
xarray	0.9.1	High level library to load, manipulate, and extract data from netCDF4 among other file formats. Uses netCDF4 for this work.
qt	5.6.2	C++ library for GUI building based on the Qt framework
pyqt	5.6.0	Python wrapper for qt

If these packages are not included in the scientific Python distribution used, there are different options to go about their installation. On Anaconda Python one can do **$ conda install package** for the package of interest (where $ signifies the start of a command from the terminal on Linux/Mac or command line on Windows), or use the graphical package manager included with Enthought Canopy. If the package is not found one can try **$ pip install package** which uses Python's built-in package manager. There is one more option for installation which is to try Christoph Gohlke's unofficial Windows binaries here, which includes a range of pre-compiled packages for Windows.

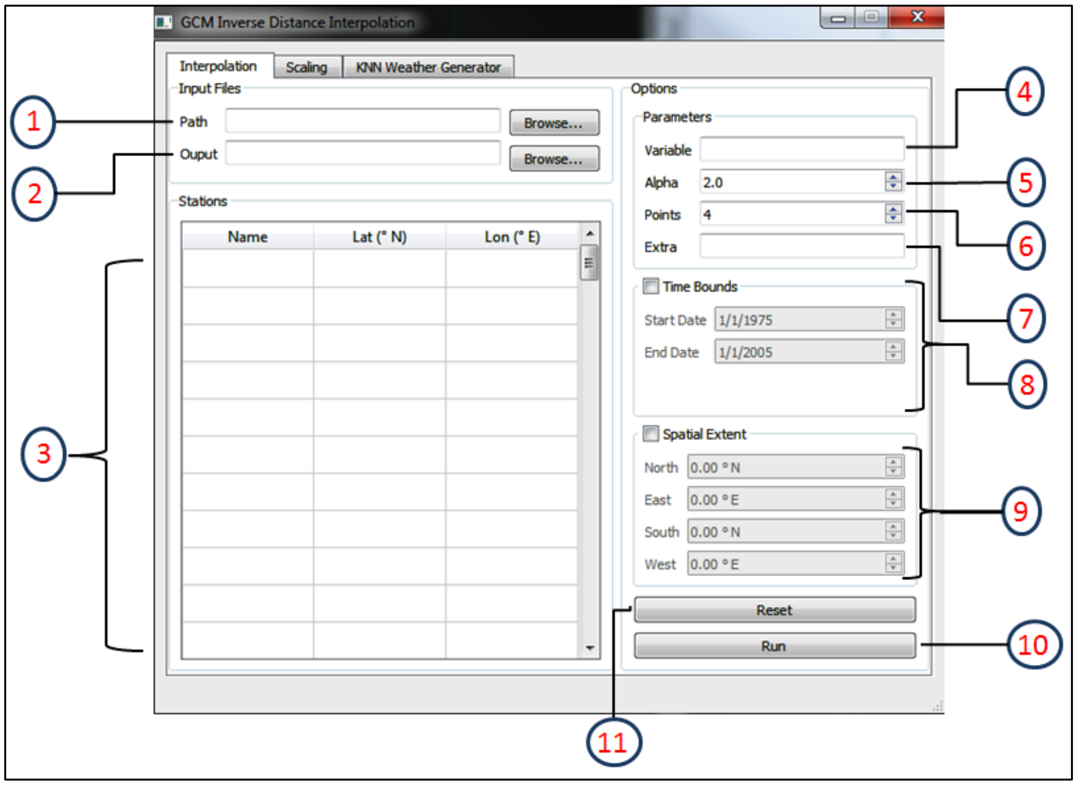
The graphical user-interface was created in Python to help facilitate user input to the inverse-distance weighting, climate scaling, and KNN weather generator algorithms described in this manual. The simplest way to run the interface is from the command line in the directory where "input.py" which is located in the "downscaling/idw" directory the online repository FIDS-UWO/climate. This can be done by using the command **$ python input.py** from the directory containing the script. If the reader prefers to use an Integrated Development Environment (IDE) to run code, there are a number of different Python IDEs available which will not be discussed here. After running "input.py" a window (Figure 0.2) will appear containing tabs for the IDW, change factor methodology (Section 3), and KNN-CAD algorithms (Section 4). In this section, we will demonstrate the IDW technique and details about its interface.

For spatial interpolation using inverse distance weighting (IDW) technique, we developed a user-friendly interface (Figure 0.2). The details of graphical user interface (GUI) element are given below:

(1) **Input**: Select GCM files (.nc file) to be interpolates. Browser option is provided, so the selection can be made anywhere from the computer. Select a folder where .nc files are saved.

(2) **Output**: Select the place where the interpolated files should be stored.

(3) **Station information**: Provide the name of each point to be interpolated to (generally meteorological stations) along with its geographical coordinates. The latitude (Lat) and longitude (Lon) are to be expressed in decimal degree format with latitude in degrees north and longitude in degrees east from Greenwich.



Alpha 2.0
Points 4
Start Date 1/1/1975
End Date 1/1/2005
North 0.00 °N
East 0.00 °E
South 0.00 °N
West 0.00 °E

Figure 0.2 User interface for IDW

(4) **Variable**: Input the name of the variable stored in the netCDF to be interpolated. For example, if the file:

"tasmax_day_CanESM2_rcp26_r1i1p1_20060101-21001231.nc" is selected as input, the corresponding variable should be "tasmax" this information is typically included in the metadata of the netCDF file.

(5) **Alpha**: This option is used to specify the alpha parameter of the distance-decay function used in the inverse distance interpolation. The selection of this parameter requires careful consideration and includes assumptions as to how climate variables vary spatially with distance from known observations. In this work the inverse squared distance value is used which corresponds to alpha value of 2.

(6) **Points**: This option specifies how many nearest grid points are considered for the interpolation. As a default the 4 closest points are used.

(7) **Extra**: Most netCDF files containing GCM output are 3-dimensional, in that the quantity of interest is represented as a surface varying in with latitude and longitude. However, some climate variables (e.g. specific humidity (hus), u-wind, v-wind) are also simulated using different pressure levels. This option can be used to specify the value of additional dimension in order to reduce the netCDF file to one that is 3-dimensional. For example, hus (specific humidity) has eight different pressure levels (100000, 85000, 70000, 50000, 25000, 10000, 5000, 1000 Pa). If user wants to interpolate hus at 5000 Pa pressure level, in option (7) the "plev=5000" value should be entered. Additional filters can be included separated by commas.

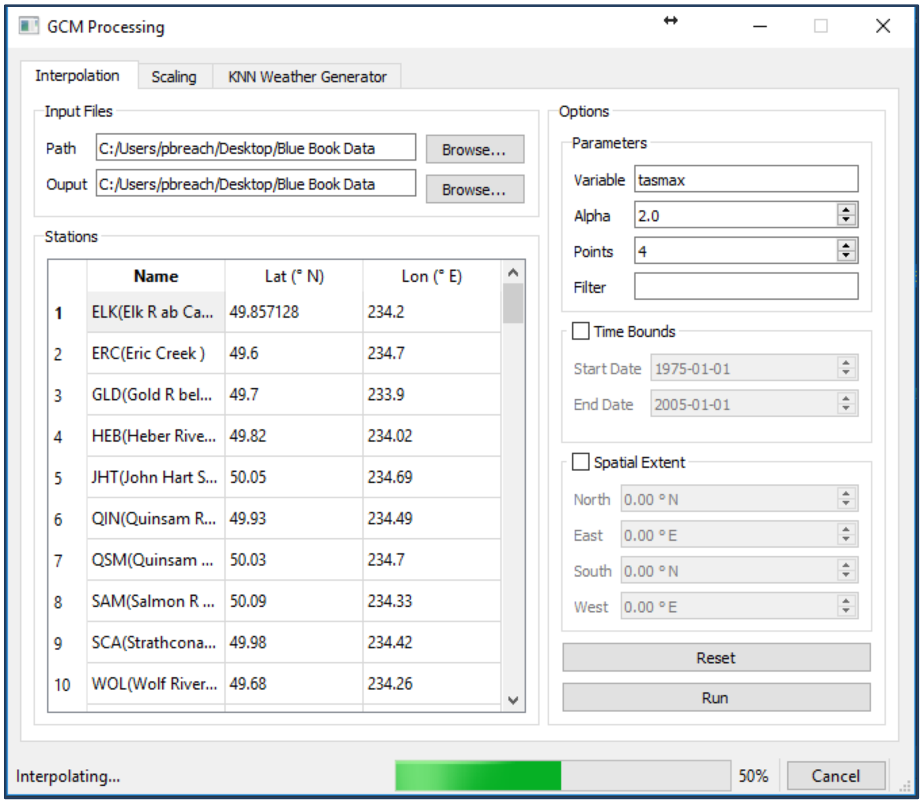
(8) **Time Bound**: This option is used to specify the time range. Time format depends on the local time format of the operating system, but is shown as dd/mm/yyyy here. Users can select any temporal range; they want to interpolate. The data should fall within the time range that is, for example, if the user chooses:

"tasmax_day_CanESM2_rcp26_r1i1p1_20060101-21001231.nc" file for interpolation, any time period between 01/01/2006 and 31/12/2100 can be selected - not earlier than 01/01/2006 or later than 31/12/2100.

(9) **Spatial Extent**: This option is for a specifying the spatial extend of the interpolation. The GCMs provide global simulations. Therefore, this option could be used to clip information for a certain spatial domain. This option along with option 8 will save computer memory and reduce the computational time for interpolation. The coordinate format here is the same format as described in option (3).

(10) **Run**: This button is for executing the process. If successful, the screen will look like Figure 0.3.

(11) **Reset**: This option is for resetting the user input.



Path: C:/Users/pbreach/Desktop/Blue Book Data

Ouput: C:/Users/pbreach/Desktop/Blue Book Data

	Name	Lat (° N)	Lon (° E)
1	ELK(Elk R ab Ca...	49.857128	234.2
2	ERC(Eric Creek)	49.6	234.7
3	GLD(Gold R bel...	49.7	233.9
4	HEB(Heber Rive...	49.82	234.02
5	JHT(John Hart S...	50.05	234.69
6	QIN(Quinsam R...	49.93	234.49
7	QSM(Quinsam ...	50.03	234.7
8	SAM(Salmon R ...	50.09	234.33
9	SCA(Strathcona...	49.98	234.42
10	WOL(Wolf River...	49.68	234.26

Variable: tasmax
Alpha: 2.0
Points: 4
Filter:

Start Date: 1975-01-01
End Date: 2005-01-01

North: 0.00 °N
East: 0.00 °E
South: 0.00 °N
West: 0.00 °E

Interpolating... 50%

Figure 0.3 User interface for IDW during interpolation

A demo output file (“Output_idw_tasmax_day_CanESM2_historical_r1i1p1.csv”) from IDW interface can be found in the downscaling/idw directory. The detailed code of IDW interface is given in

Appendices

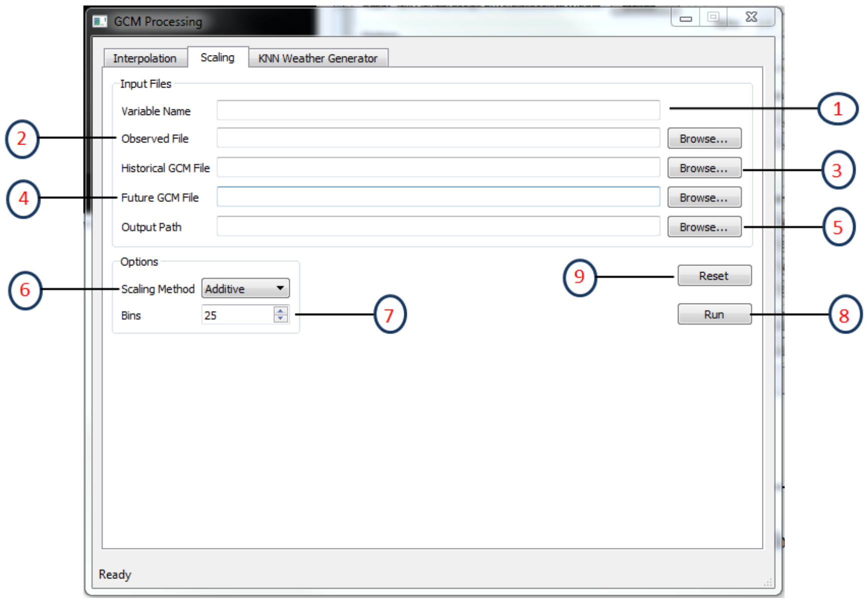
Appendix A.

The following section will provide information about Change Factor (CF) methodology used to incorporate projected changes in climate into observed data.

3 Change Factor Methodology

Change Factor (CF) methodology is used to incorporate projected changes in climate for a future time slice into observed data series for local scale climate change impact assessment. This procedure is also known as the delta change method. Anandhi *et al.*, (2011) found that multiple change factors associated with specific percentiles ranges of a climate variable's distribution were more effective in transferring changes projected by climate models than a single mean change factor. The algorithm used here is based on the work of Anandhi *et al.*, (2011), and was incorporated with a graphical user interface for the multiple change factor methodology for the daily time scale.

To use the interface, the reader is referred to the steps discussed in Section 0. After opening the interface the "Scaling" tab can be used to display the input options for the CF algorithm (Figure 0.1). Before scaling using the CF method, GCM data needs to be spatially interpolated (Section 0). This interface has multiple options which are discussed below.



Scaling Method: Additive
Bins: 25

Figure 0.1 Interface for Change Factor (CF) methodology

(1) **Variable Name**: This field is used to specify the variable to be scaled e.g. “tasmax” for maximum temperature.

(2) **Observed File**: This field is used to specify the file with historical observations. A demo dataset (“CF Observed File.csv”) of historical data can be found here: downscaling/cfm.

(3) **Historical GCM file: S**election of the interpolated historical GCM file (description given in section 0). A sample of the interpolated historical GCM data (input to the CF calculation) is given here: downscaling/cfm.

(4) **Future GCM File:** Spatially interpolated future GCM data set. A demo (“Future GCM File_pr_day_CanESM2_rcp45_r1i1p1.csv”) file of interpolated future GCM data (input to the CF calculation) is given here: downscaling/cfm.

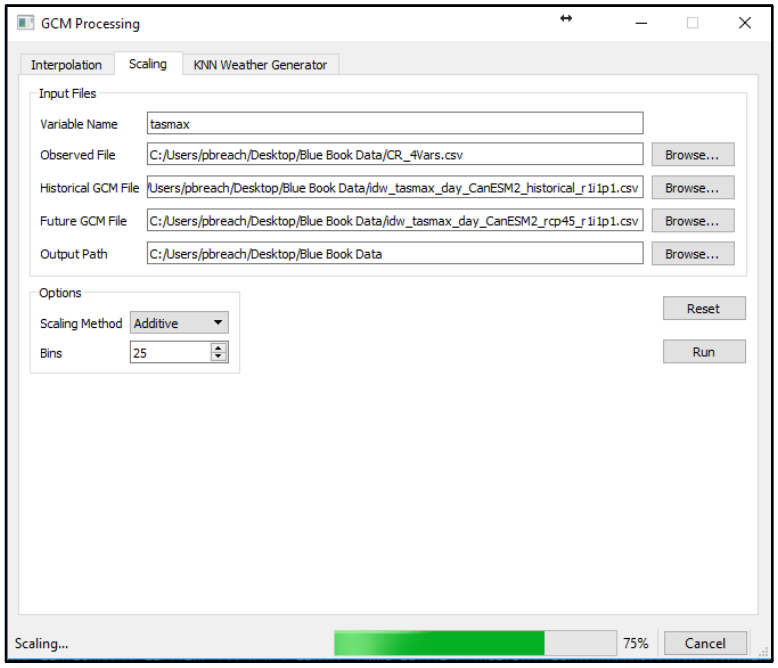
(5) **Output Path: S**election of a folder for saving the output file. A demo output file (“Output _CF_ scaled_pr_day_CanESM2_rcp45_r1i1p1.csv”) is provided here: downscaling/cfm.

(6) **Scaling Method:** Choice between two options - additive and multiplicative following Anandhi *et al.*, (2011). The additive option transfers the change in mean to the observed data, while multiplicative transfers both, the change in mean and variance. It is recommended to use the multiplicative option for precipitation data and additive for temperature data.

(7) **Bins:** Specification of the number of bins for which the change factors will be calculated and applied across the distribution of the GCM input data and observed data respectively. The default number of bins is set to be equal to 25 (Anandhi *et al.*, 2011).

(8) **Run:** To execute the process click on this button. A green progress bar should appear during the process of running the program (Figure 0.2).

(9) **Reset:** Clearing user inputs.


GCM Processing

Input Files

Variable Name: tasmax

Observed File: C:/Users/pbreach/Desktop/Blue Book Data/CR_4Vars.csv

Historical GCM File: Users/pbreach/Desktop/Blue Book Data/idw_tasmax_day_CanESM2_historical_r1i1p1.csv

Future GCM File: C:/Users/pbreach/Desktop/Blue Book Data/idw_tasmax_day_CanESM2_rcp45_r1i1p1.csv

Output Path: C:/Users/pbreach/Desktop/Blue Book Data

Options

Scaling Method: Additive

Bins: 25

Scaling... 75%

Figure 0.2 Interface for Change Factor calculation filled with demo information

Figure 0.2 shows an example of the input information needed to run CF algorithm. After successful execution of this method, an output file will be generated and can be found in the output folder. A demo output ("Output _CF_ scaled_pr_day_CanESM2_rcp45_r1i1p1.csv") is given here: downscaling/cfm. Coding details of CF method and its interface are given in **Error! Reference source not found.**. The following section discusses KNN-CAD weather generator and its interface.

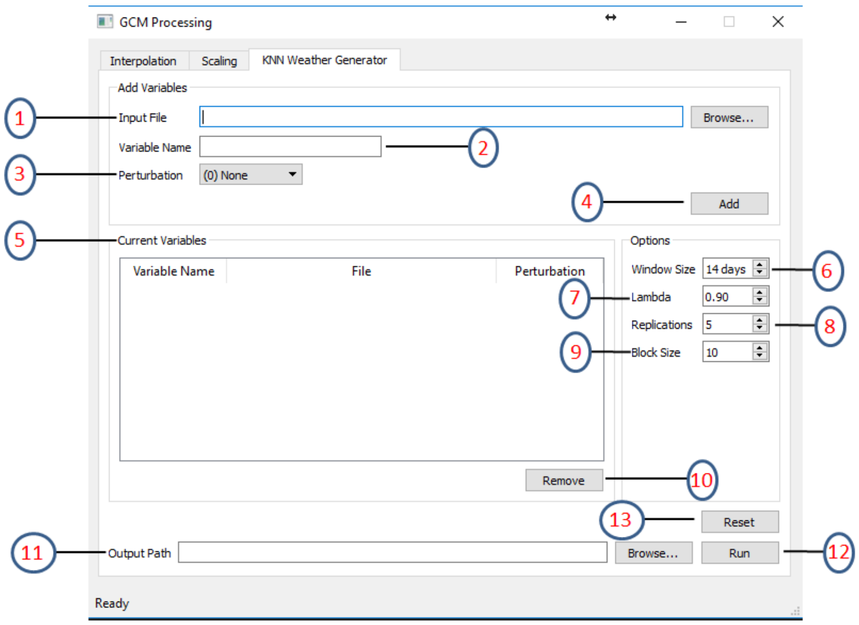
4 KNN-CAD Weather Generator

The KNN-CAD weather generator discussed in this section is based on the modified version of KNN-CADv4, developed by King *et al.,* (2015) for computational efficiency. Additional discussion is available in Appendix C. KNN-CADv4 is a block-bootstrap non-parametric multisite weather generator based on the K-nearest neighbour (K-NN) algorithm for a given day of the year. The algorithm was designed to be able to resample a given input observed weather series to a length suitable for statistical analyses. For hydrological applications, it may be used for generating Intensity-Duration-Frequency (IDF) curves for rainfall, or precipitation and temperature variables for hydrologic modelling with which flow frequency curves could be obtained. In this section, the modified algorithm will be briefly discussed along with the interface developed to run the KNN-CAD algorithm. The Python programming language was used to implement this algorithm, and the computer code can be found in Appendix C. See Section 0 for details regarding the Python installation.

Future climate variables projections using a weather generator are obtained in two steps: (1) scaling of future climate variables; and (2) generation of synthetic future climate time series. The delta change, or change factor, methodology discussed in Section 3 has been used for future-scaling the climate variables. The procedure used here for weather generation of future climate series is performed in three steps:

1. Spatial Interpolation – GCM output files need to be spatially interpolated (Section 2).
2. Scaling – CF methodology applied to incorporate future projected climate changes into observed data (Section 0).
3. Weather Generation - Use of the weather generator to resample the future-scaled climate series.

The first two steps have been discussed in Sections 0 and 0. In the final step illustrated here, it is assumed that the input files have been generated using previously described procedures. The three steps procedure is illustrated in


Perturbation (0) None
Window Size 14 days
Lambda 0.90
Replications 5
Block Size 10
Ready

, while the user interface for the modified KNN-CADv4 algorithm is shown in Figure 0.2.

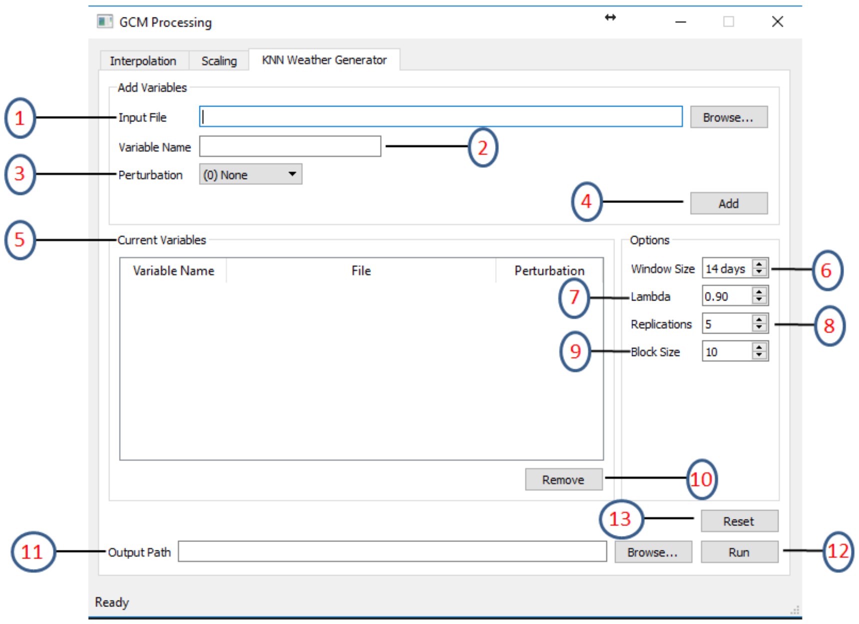

Perturbation (0) None
Window Size 14 days
Lambda 0.90
Replications 5
Block Size 10
Ready

Figure 0.1 User interface for KNN-CAD algorithm

Instructions for running the interface are provided in Section 0. There are multiple inputs and options for the KNN-CAD weather generator, which are detailed here:

(1) **Input File:** Specify the input file for a given climate variable. Only one climate variable per file is specified, and all files should be in the same format. Sample input CSV files with the prescribed formatting are provided in the "downscaling/knncad" directory.

(2) **Variable Name:** Specify the variable corresponding to the input file (primarily for the purpose of assembling the input files together before running the KNN-CAD algorithm).

(3) **Perturbation:** The drop-down menu specifies three different perturbation options to choose from: (i) None, (ii) Normal, and (iii) Log-Normal. The first option does not incorporate any perturbation into the resampled data and will result in a replicated climate series with range of values matching the range of the observed data. The second option uses "Normal" perturbation, for which a random normally-distributed variable with mean corresponding to the climate variable value of the given day and standard deviation equal to that of the K-NN is used. Finally, the third option implements a random two-point log-normally distributed variable to perturb the observed data series. In past climate downscaling experiments, temperature-like variables are used with the "Normal" perturbation option, while precipitation-like variables are used with the "Log-Normal" perturbation option. Perturbation is used to introduce variability in the resampled climate data and allow values to go out of the observed data range.

(4) **Add:** Add the data under the "Add Variables" section to the "Current Variables" list (possible addition of additional variables). For the hydrological analyses performed in the past, daily precipitation, maximum temperature, and minimum temperature have been used together.

(5) **Current Variables:** This table will display the variables along with their file names and selected perturbation option that have been selected to execute the KNN-CAD algorithm. Additional variables can be added as per the instructions for previous step.

(6) **Window Size:** The window size is used to set the number of days around the current day of year to use when defining the L and K nearest neighbors (see King *et al* (2015)). A default value is 14.

(7) **Lambda:** This perturbation parameter is to be applied to the resampled observed climate series. A value of 0 would result in a climate series that has the maximum level of perturbation applied in accordance with the perturbation option used, while a value of 1 implies no perturbation. This value is typically selected to be as large as possible but less than 1. The default value of 0.9 (10% of data perturbed).

(8) **Replications:** The number of replications corresponds to the number of synthetic climate series to be generated of equal length to the observed climate series. This number should be large enough to obtain an adequate synthetic data length for analysis. However, very large numbers should be chosen with caution. The KNN-CAD algorithm used here currently holds all generated data in memory until the calculation process is completed. If the number is too large, there is a risk of consuming computing resources to the point of a crash. For this reason, the number of replication is limited to 20 while the default is 5.

(9) **Block Size:** The KNN-CADv4 algorithm resamples used ‘blocks’ of the observed climate series as opposed to a single day. This allows for resampling to take place with minimal deterioration of the temporal autocorrelation of the observed climate. The default value is set to 10 days.

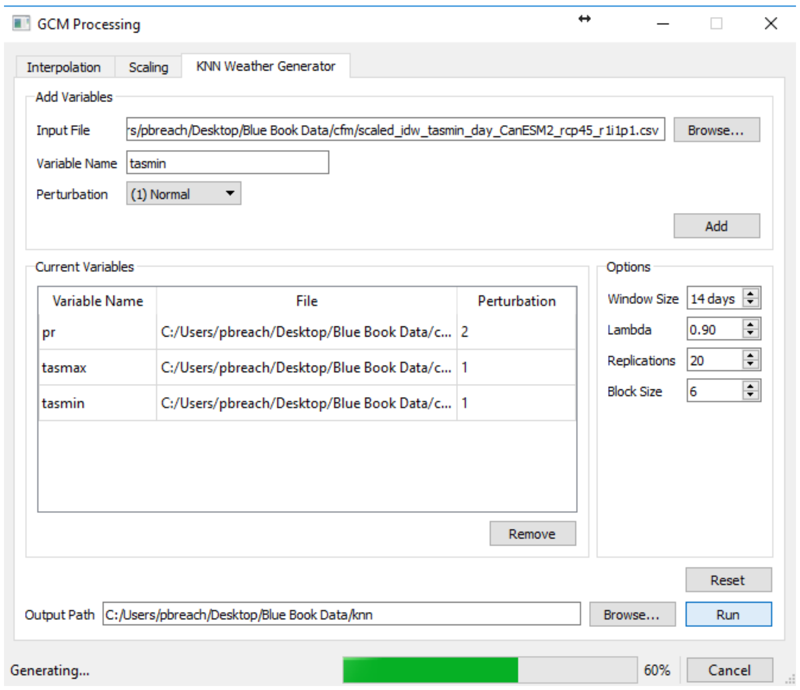
(10) **Remove:** This option removes the currently selected row corresponding to the "Current Variables" table.

(11) **Output Path:** Specifies the path where the generated file will be placed.

(12) **Run:** Execution of the KNN-CAD algorithm using the information provided.

(13) **Reset:** Reset the user input information.

An example of the algorithm running with all provided input is shown in shown in Figure 0.2. Python code used to implement the KNN-CAD algorithm is included in Appendix C.



Input File: s/pbreach/Desktop/Blue Book Data/cfm/scaled_idw_tasmin_day_CanESM2_rcp45_r1i1p1.csv

Variable Name: tasmin

Perturbation: (1) Normal

Variable Name	File	Perturbation
pr	C:/Users/pbreach/Desktop/Blue Book Data/c...	2
tasmax	C:/Users/pbreach/Desktop/Blue Book Data/c...	1
tasmin	C:/Users/pbreach/Desktop/Blue Book Data/c...	1

Window Size: 14 days

Lambda: 0.90

Replications: 20

Block Size: 6

Output Path: C:/Users/pbreach/Desktop/Blue Book Data/knn

Generating... 60%

Figure 0.2 User interface for KNN-CAD algorithm running with all inputs filled in.

The next section discusses about the Maximum Entropy Based Weather Generator (MEBWG).

5 Maximum Entropy Based Weather Generator (MEBWG)

Srivastav and Simonovic (2014) have developed a non-parametric multisite, multivariate maximum entropy based weather generator (MEBWG) for generating synthetic climate series of daily precipitation, minimum and maximum temperature values. There are three main computational steps involved in the implementation of the MEB weather generator: (1) orthogonal transformation of daily climate variables at multiple sites to remove spatial correlation; (2) use of maximum entropy bootstrap (MEB) to generate synthetic replicates of climate variables; and (3) inverse orthogonal transformation of synthetic climate variables to re-established spatial correlation. For more technical details about MEBWG, please refer to Srivastav and Simonovic (2014). In this section, the computer code developed for the implementation of MEBWG is discussed.

Source code for MEBWG is available in Appendix-D. In addition, source code, input and output sample data are available in at downscaling/mbewg. Before using MEBWG, input data must be interpolated (Section 0) and scaled for future climate (Section 0), as was done for the KNN-CAD downscaling procedure (Section 0). Downscaling using MEBWG can be divided into three main steps (Figure 0.1).

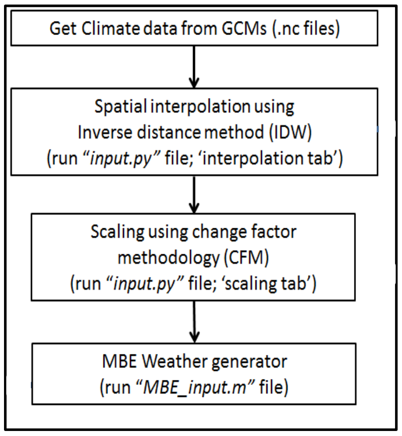

Get Climate data from GCMs (.nc files)
Spatial interpolation using Inverse distance method (IDW) (run "input.py" file; 'interpolation tab')
Scaling using change factor methodology (CFM) (run "input.py" file; 'scaling tab')
MBE Weather generator (run "MBE_input.m" file)

Figure 0.1 Steps involved in downscaling climate variables using MBEWG

In the downscaling/mbewg directory of the FIDS-UWO/climate repository there are three Matlab files described below:

(1) **MBE_input.m**: This file is for input preparation to the MEBWG and includes user prompts for input.

(2) **MBE_RKS.m**: This file contains the functions used to implement the algorithm. User changes to this code are not recommended for running the MBEWG.

(3) **csvwrite_with_headers.m**: This file helps to write output into .csv format with a header.

In previous climate downscaling experiments the model was used with precipitation, maximum and minimum temperature data sets and was shown to reproduce well the observed climate statistics from the input data while generating a synthetic climate series. Illustrative input data sets are provided in the downscaling/mbewg directory. To run this model user needs to have Matlab installed along with the statistics toolbox. For this work Matlab R2016a was used.

5.1 Source Code Execution

To run the MEBWG model, input files should be prepared as is demonstrated by the illustrative input data sets, and should be included in the same directory as the Matlab files. Next, run the "MBE_input.m" file from Matlab and a dialog window should appear (Figure 0.2). The user is then prompted to input the number of replications to make from the observed data which corresponds to the number of synthetic climate series of data length equal to the observed series to be generated. The default value is two.

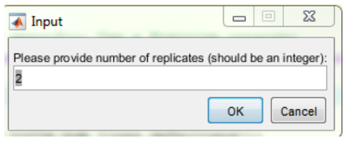
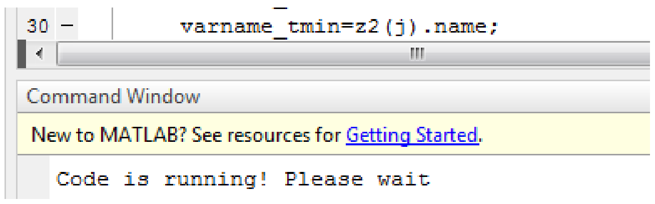

Please provide number of replicates (should be an integer):
2

Figure 0.2 MEBWG: dialogue window

A message should show up in the Matlab console which reads, “Code is running! Please wait” (Figure 0.3). Another dialog box should appear after completion of the generation process (Figure 0.4). If the downscaling task is not complete, “Downscaling is not completed” message will be displayed.



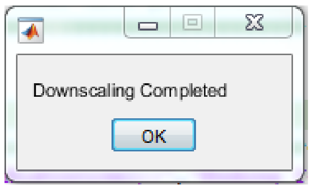
```
varname_tmin=z2(j).name;
```

Command Window

New to MATLAB? See resources for Getting Started.

```
Code is running! Please wait
```

Figure 0.3 MEBWG command window message



Downscaling Completed

Figure 0.4 MEBWG: dialogue window after completion of the downscaling task

An important note, if the GCM name, realization, and RCP (representative concentration pathways) do not match for all input files, it will show an error message, “'Error: GCM files are not matching!!”. In the event that this error occurs, check that the input file names shown in demo data set at downscaling/mbewg or make a change in “MBE_input.m” Line 35-37. Details code of MEBWG provided in Appendix D. The following section will discuss the beta regression based statistical downscaling model.

6 Beta Regression Based Statistical Downscaling Model

Mandal *et al.,*(2016) developed a multisite downscaling method based on the beta regression (BR). The proposed method generates downscaled daily precipitation conditioned on precipitation states. Here the details to run the computer code used to build BR downscaling model are discussed.

The BR model was developed in the Matlab environment. The statistical toolbox of Matlab R2016a academic version is used to develop the model. Before using BR model, the user needs to prepare the input data. The following section describes how to prepare the data to run the BR model in the Matlab.

6.1 Data Preparation

For the BR model to run, the user needs historically observed precipitation data, as well as historical and future GCM output datasets. The BR model is developed based on a statistical relationship between predictor climate variables and a predictand, which in this case is precipitation. For the predictor variables, daily maximum and minimum air surface temperature (Tmax and Tmin), mean sea level pressure (mslp), specific humidity (hus) at 500 hPa, zonal (u-wind) and meridional (v-wind) wind are used. The user needs to extract these variables from GCM output for the historical and future time periods. The following steps should be taken to prepare the data to run the BR model:

(1) Download predictor variables from the Coupled Model Intercomparison Project 5 (CMIP5) GCM output database. http://cmip-pcmdi.llnl.gov/cmip5/data_portal.html

(2) Interpolate all GCM climate variables at downscaling station locations. For spatial interpolation use IDW method (follow Section 0). Combine all the variable data into

a single file for a particular scenario. Illustrative input and output files are provided here: downscaling/br. Temperature data is in units of degrees Kelvin (K).

(3) Prepare observed historical precipitation data as shown here: downscaling/br. “Observed Precipitation.csv”

(4) “Input.m” to run the model.

All the illustrative data sets and source code are provided at: downscaling/br. In addition, the Matlab source code for the BR method is provided in Appendix-E. There are four Matlab files for the BR model:

(1) *Input BR.m*: This file prepares input for and runs the BR model.

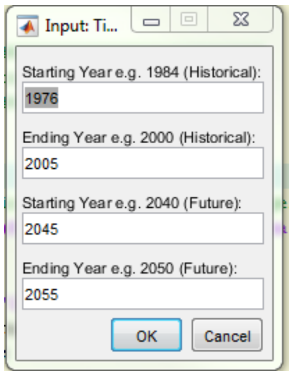
(2) *betareg_main.m*: This file is for regression calculation.

(3) *Beta_Regression.m*: This file is for conditional probability and precipitation states calculation. K-means clustering, classification and regression trees (CART) are included in this file.

(4) *betalik.m*: This file is for link functions in beta regression. Log transformation is used in this model.

6.2 BR Source Code Execution

The user should run “Input BR.m” from Matlab to execute BR model. Multiple input windows will appear in succession. First a window will appear asking for time period of interest. In this window, the user must specify historical and future time slices as shown in Figure 0.1. This interface is suitable for any timeframe but it should not exceed that of the input files.


Input: Ti...

Starting Year e.g. 1984 (Historical): 1976

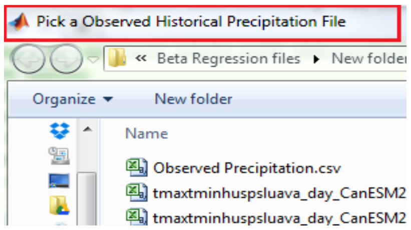
Ending Year e.g. 2000 (Historical): 2005

Starting Year e.g. 2040 (Future): 2045

Ending Year e.g. 2050 (Future): 2055

Figure 0.1 BR: Timeframe input window

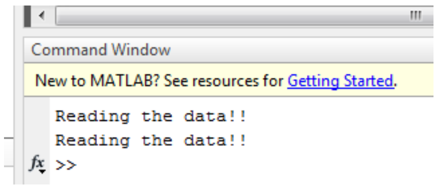
After pressing “OK” another window will appear requesting the historical observed data set (Figure 0.2). Select a historical observed precipitation data set. When the file is being read, a “Reading the data!!” message will appear in the command window (Figure 0.3).


Pick a Observed Historical Precipitation File

Observed Precipitation.csv
tmaxtminhuspsluava_day_CanESM2
tmaxtminhuspsluava_day_CanESM2

Figure 0.2 BR: Input window for historical data set

Next window will prompt the user for historical GCM data (Figure 0.4). Finally, the last input window is for future GCM data which can be selected here (Figure 0.5). If all the inputs are validated a message, “Code is running! Please wait” will be displayed in the Matlab command window (Figure 0.6). After completion a message will appear “Downscaling Completed”.



```
Reading the data!!
Reading the data!!
>>
```

Figure 0.3 BR: Message during input file reading

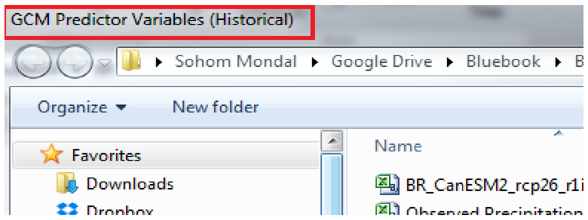


Figure 0.4 BR: Input window for historical GCM data set

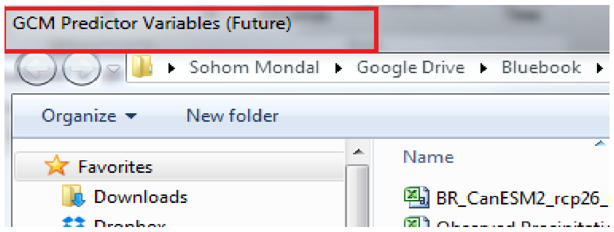
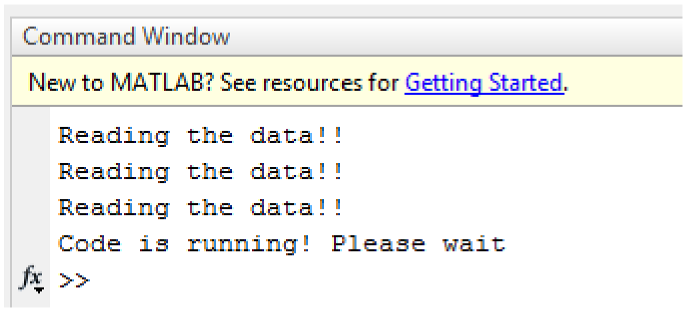


Figure 0.5 BR: Input window for future GCM data set



```
Reading the data!!
Reading the data!!
Reading the data!!
Code is running! Please wait
>>
```

Figure 0.6 BR: Message during compilation

The input data files for this sample run are prepared from the GCMs stored in netCDF format, using the IDW interpolation method discussed in Section 0. There are multiple tools available to read netCDF files such as those found here. It is recommended for the user to inspect the metadata of these files before proceeding with the analysis. This Excel extension for reading netCDF files can be a useful tool for this. Details code of BR downscaling model also provided in Appendix E.

7 Physical Scaling Model for Downscaling Climate Variables

It has been highlighted in previous sections that statistical downscaling methods used currently don't explicitly account for geophysical characteristics of the region of study. This is one of the major drawbacks of statistical downscaling because of which geophysical changes in a region can't be modelled using statistical downscaling methods. In physical scaling (SP) method (Gaur and Simonovic, 2016a, 2016b), this is achieved by including additional geophysical covariates representing land-cover and elevation distribution of the region. The predictand (observed local scale climate) and predictor (large scale climate and geophysical covariate) variables are linked using a Generalized Additive Model (GAM) regression relationship. In GAM regression, the predictand variable is connected to smoothed predictor variables using a link function. The smoothing is generally performed using non-parametric algorithms. The regression function is totally non-parametric in nature and hence is suitable modelling a range of climate variables including precipitation and temperature. A more detailed description of SP model and its extensions is provided below.

7.1 Physical Scaling (SP) Method

SP method approach to downscaling temperature (surface or air) can be mathematically expressed as:

$$g(T_{obs}) = B_0 + f_1(T_{\text{mod}}) + f_2(E_p) + f_3(LC_p) \tag{7.1}$$

Where *T* denotes temperature, *E* denotes elevation (masl) of the reference pixel, *LC* denotes categorical land-cover variable associated with the reference pixel and *B* denotes regression parameters. Subscript obs and mod denote if the climatic data is observed or model based respectively. T_{mod} denotes large scale "background" climate data obtained by bilinear interpolation of GCM temperature data at the reference pixel. Subscript *p* indicates that the data used is a pixel scale data. Variables *g* and *f* represent the link and smoothing functions respectively. In this study, smooth functions are fit using penalized likelihood maximization algorithm. The penalized likelihood maximization algorithm is a variant of maximum likelihood estimation algorithm and applies a tradeoff between model fit wiggliness and goodness of fit by incorporating a penalty function(Wood, 2000).

In the case of precipitation, method involves a two-step process of predicting precipitation occurrence using a logistic regression model and a wet day precipitation amounts model using a GAM regression model. SP method approach to downscale precipitation(Gaur and Simonovic, 2017) can be mathematically expressed as:

$$\ln\left(\frac{P_{obs}}{1-P_{obs}}\right) = B_0 + B_1 P_{\text{mod}} + B_2 E_p + B_3 LC_p \qquad \textbf{(7.2)}$$

$$g(P_{obs,wet}) = B_0 + f_1(P_{\text{mod},wet}) + f_2(E_p) + f_3(LC_p) \qquad \textbf{(7.3)}$$

Where, notations have previously defined meanings. Additionally, subscript *wet* denotes values on wet days only (i.e. days with more than 0.1 mm of precipitation).

7.2 Physical Scaling With Surrounding Pixel Information (SPS) Method

SPS method is an extension to SP method where land-cover and elevation properties of the reference as well as neighborhood pixels are incorporated into the method formulation (Gaur and Simonovic, 2016a). SPS method for downscaling air temperature can be mathematically expressed as:

$$g(T_{obs}) = B_0 + f_1(T_{\text{mod}}) + f_2(E_p) + f_3(LC_p) + f_4(Fr_{W,s}) + ... + f_{17}(Fr_{BSV,s}) + f_{18}(\mathrm{R}_{E,s}) \qquad \textbf{(7.4)}$$

where, symbols have similar meanings as explained above. Predictors $Fr_{W,s}$, ...$Fr_{BSV,s}$ represent the fraction of total area surrounding the reference pixel that is occupied by *Water*,….*Barren and Sparsely Vegetated* land-cover classes respectively. The value of predictors: $Fr_{W,s}$, ...$Fr_{BSV,s}$ is between 0 and 1 and they add up across all neighbourhood land-cover classes to give a value of 1. Neighbourhood elevation information is incorporated by including a predictor $R_{E,s}$ which represents the ratio between reference pixel elevation and mean elevation of pixels surrounding the reference pixel at a specific neighbourhood scale, s.

For precipitation, SPS downscaling method involves two steps of forming a precipitation occurrence and wet day precipitation amounts model (Gaur and Simonovic, 2017a). The two steps involved in SPS method can be mathematically expressed as:

$$\ln\left(\frac{P_{obs}}{1-P_{obs}}\right) = B_0 + B_1 P_{\mathrm{mod}} + B_2 E_p + B_3 LC_p + B_4 Fr_{W,s} + ... + B_{17} Fr_{BSV,s} + B_{18}\,\mathrm{R}_{E,s} \qquad \textbf{(7.5)}$$

$$g(P_{obs,wet}) = B_0 + f_1(P_{\mathrm{mod},wet}) + f_2(E_p) + f_3(LC_p) + f_4(Fr_{W,s}) + ... + f_{17}(Fr_{BSV,s}) + f_{18}(\mathrm{R}_{E,s}) \qquad \textbf{(7.6)}$$

where, symbols have the same meanings as explained before.

The choice of spatial scale within which neighbourhood geophysical characteristics are analysed is critical to SPS method application and performance. Four neighbourhood scales (represented as *s* in equations 4, 5 and 6): 3x3, 5x5, 7x7 and 9x9 have been chosen before to define neighbourhood characteristics (Gaur and Simonovic, 2016a). The selected neighbourhood scales are shown in **Error! Reference source not found.** in darker shades of grey for neighbourhood scales: 3x3, 5x5, 7x7 and 9x9 respectively, while the reference pixel is shown in black. Larger neighbourhood scales are considered to be inclusive of smaller spatial scales for instance neighbourhood scale: 5x5 encompasses pixels corresponding to neighbourhood scale: 3x3 and additional darker shaded pixels specific to 5x5 neighbourhood scale.

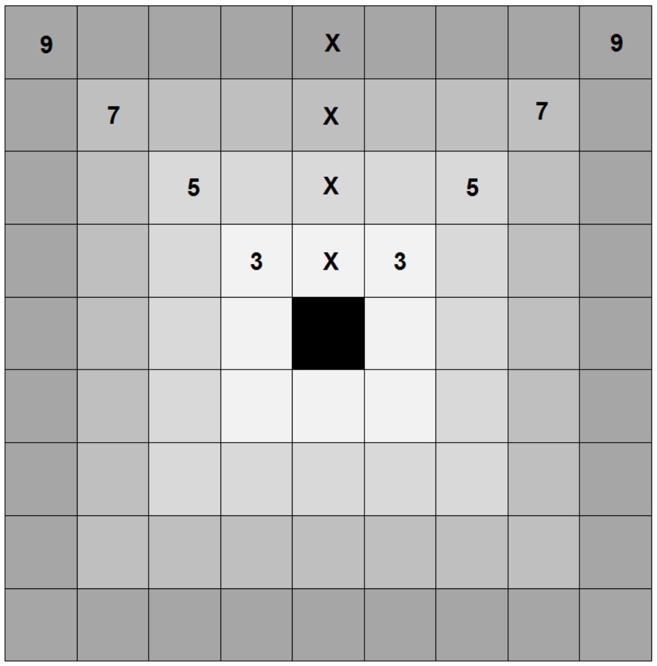


Figure 0.1 Neighbourhood scales considered in this study. Increasing neighbourhood scales are shown in progressively darker shades of grey. The reference pixel is shown in black.

7.3 Working Example of SP and SPS Methods in the R Programming Language

In this section, we demonstrate how SP and SPS methods can be applied to downscale GCM projections in the R programming language. First, we provide a gentle introduction to the R programming language, and then present a step by step demonstration of SP and SPS model downscaling using it.

7.3.1 Basics of R Programming Language

R is a programming language and software environment for statistical analysis, graphics representation, and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team (Team, 2008). R implements many common statistical procedures as well as provides excellent graphics functionalities through its libraries and packages. It is an open source language which means that the users are able to acquire it free of cost as well as can contribute towards its research and development. In this sub-section, basic features of R programming language have been reviewed with an intention that this can help the readers to get started with R.

7.3.2 Downloading and Installing R

R programming language for windows operating system can be downloaded from: https://cran.r-project.org/bin/windows/base/. R for Linux and Mac operating systems can also be downloaded from the R project for statistical computing website: https://www.r-project.org/. It is also common to use R Studio to run and edit R codes as this software provides enhanced code editing, debugging, and visualizing capabilities to the R users. R Studio for Windows, Linux, and Mac operating systems can be downloaded from: https://www.rstudio.com/products/rstudio/download/. R and R Studio software can then be installed on the computer by running the executables obtained from above sources.

7.3.3 Common Data-types in R

Some of the most common data-types used in R programming language are: vector, data frame, and list. A vector can be a sequence of numbers, logical values, or character strings. A vector with three numeric values can be defined as follows in the R Studio command-line. Below the user commands are provided in lines commencing with the symbol: “>” while output from the R (if any) is provided in lines commencing with the symbol: “[1]”.

```
C1 > c(3, 4, 7)
[1] 3 4 7
```

A vector with three logical values can be defined as follows:

```
C2 > c(TRUE, FALSE, FALSE)
[1] TRUE FALSE FALSE
```

A vector with three character values can be defined as follows. The resulting vector has also been saved into a variable named: vec.char.

```
C3 > vec.char = c(“a”, “b”, “c”)
```

A vector element can be extracted using the “[]” brackets with the index of the element to be extracted as shown below.

```
C4 > vec.char[1]
[1] a
```

A data frame is used to store vectors of equal length in the form of tables. Below is the command to create a data frame “df” with three columns and four rows representing the marks obtained by four students named: “A”, “B”, “C”, “D” in two subjects: maths and physics.

```
C5 > df = data.frame (name = c(“A”, “B”, “C”, “D”), maths = c(85, 90, 97, 76), physics = c(88, 66, 76, 98))
```

Data-frame element(s) can be extracted using the “[]” brackets with the row number and column number indices of the element to be extracted. The entire row or column can be extracted by specifying the same in the command. An entire column can also be extracted by specifying the column name together with the “$” operator as shown below.

```
C6 > df[1,2]
[1] 85
C7 > df[1,]
[1] A 85 88
C8 > df[,1]
[1] A B C D
C9 > df$name
```

```
[1] A B C D
```

A list can be used to store vectors of equal or unequal lengths. Below is a R command to create a list with 3 numeric, 4 logical, and 5 character elements stored in the first, second, and third elements. The list is stored as a variable named “lst”.

```
C10 > lst = list (c(1, 5, 7), c(TRUE, TRUE, FALSE, FALSE), c(“a”, “b”, “c”, “d”, “e”))
```

List elements can be extracted by using the “[[]]” brackets along with the list element number that needs to be extracted. Further, sub-elements within a list element can also be accessed using “[]” brackets with the index of the sub-element number as demonstrated below.

```
C11 > lst[[1]]
[1] 1 5 7
C12 > lst[[1]][3]
[1] 7
```

7.3.4 Relevant R Packages

R packages are a collection of R codes, functions, data, and compiled code in a well-defined format. R comes with a standard set of packages. Other packages can be downloaded and installed separately by the users based on their needs. Here we provide a brief introduction of a few R packages (apart from the standard packages) that are very useful in performing downscaling of GCM data by SP and SPS methods in R. These packages are:

- *MODIS and MODISTools*: The intended purpose of these packages is to facilitate acquisition and processing of MODIS data-products. MODIS package contains functions to gain automated access to the global online data archives and processing capabilities such as file conversion, mosaicking, sub-setting, and time-series filtering (Mattiuzzi, 2016). The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages ("MODIS", repos="http://R-Forge.R-project.org"). MODISTools package also allows users to extract MODIS data time-series at one or more than one locations without downloading the image rasters. The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("MODISTools", repos="http://R-Forge.R-project.org").

- *ncdf4*: This package is designed to work with NetCDF libraries version 4 in R, which is the most commonly used NetCDF version currently. Another package: "ncdf" can be used to access NetCDF version 3 libraries. In "ncdf4" package utilities like chunking and compression have also been included. The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("ncdf4", repos="http://R-Forge.R-project.org").
- *raster*: This package is intended to facilitate raster processing in R. Among other functions, the package contains functions that can read and write rasters, perform raster operations such as reprojection, resampling, filtering, merging etc., perform raster calculations, and visualize raster data (Hijmans *et al.*, 2016). The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("raster", repos="http://R-Forge.R-project.org").
- *lubridate*: The lubridate package is intended to facilitate easy handling of date-time data in R. Among others, it contains functions that can be used to extract components of a date-time such as year, month, day, hour, minute, and seconds, and perform algebraic manipulation on the date-time objects (Grolemund *et al.*, 2016). The package can be downloaded from

Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("lubridate", repos="http://R-Forge.R-project.org").

- *reshape2*: This R package is extremely useful to transform data between wide and long formats. A wide format has a column for each variable while a long format has a column for possible variable types and another column for the values of these variables (Wickham, 2016). The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("reshape2", repos="http://R-Forge.R-project.org").
- *ggplot2*: This package is meant for “declaratively” creating graphic by telling ggplot2 how to map variables to aesthetics and what graphical primitives to use. It produces plots following the grammar of graphics (Wilkinson, 2005) where essential building blocks of a graph i.e. data, aesthetic mapping, geometric object, statistical transformations, scales, coordinate system, position adjustments, and faceting are specified by the user (Wickham and Chang, 2016). The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("ggplot2", repos="http://R-Forge.R-project.org").
- *mgcv*: This package is very helpful in performing Generalised Additive Modelling regression. It includes several methods for estimating regression parameters, smoothing functions, and link functions in GAMs in computationally efficient manner (Wood, 2017). The package can be downloaded from Comprehensive R Archive Network (CRAN) by running the following command on the command line: install.packages("ggplot2", repos="http://R-Forge.R-project.org").

7.3.5 Other Relevant R Functions

Apart from the functions available in aforementioned packages, plenty of useful functions are available in R base library. They have been used extensively while downscaling GCM projections using SP and SPS models. A description of some of those functions that have been most extensively used is provided below. A more detailed description of any of these functions (or any other function) can be obtained by running "?"function-name command in R.

- *which()*: This function is used to know the position of elements of a logical vector that are TRUE.
- *subset()*: This function is used to extract section of a data-frame with rows that meet a particular criteria. For instance in a data-frame that stores monthly discharge data time-series, this function can be used to select part of the time-series that correspond to January or have month values equal to 1.
- *sapply/lapply*: The sapply() and other similar functions are an alternate for looping in R. Their usage is recommended as their usage can make the R codes run much faster than when for instance for() loops are used. The "apply" family of functions have many variants like sapply() which stands for simplify and apply, lapply() which stands for list and apply, vapply(), tapply(), etc.
- *rbind/cbind():* Both rbind() and cbind() are used to combine two data-frames or matrices. It is essential that data-frames (or matrices) have the same number of columns when rbind() is used to combine them. Similarly data-frames (or matrices) should have equal number of rows when cbind() is used.

- *do.call():* The do.call() function executes a function call over all list elements passed to it. This function is commonly used in conjunction with lapply() function where "list" output obtained from lapply() are further analysed or manipulated using do.call() function.

7.3.6 Application of SP and SPS Methods Downscaling in R

In this section a demonstration of how R programming language can be used to downscale climate model projections using SP and SPS methods is presented. A discussion on how to extract, organize and prepare remotely sensed and climate model datasets in R is provided first. This is followed by a working example where SP and SPS models are applied to downscale a sample future temperature projection dataset.

Climate model datasets from GCMs or reanalysis products are generally available in Network Common Data from (NetCDF) format. NetCDF is a self-describing, machine independent data format that supports the creation, access, and sharing of array oriented data. By self-describing, it means that information about specifications of the file, the data it stores, and its layout is stored within the file. In R NetCDF format files can be accessed using "ncdf4" package. All packages need to be downloaded and loaded in the R session prior to their usage. Packages can be loaded by using command "library"(package-name) in the R session. Following commands can be used to access and extract data from NetCDF files in R:

```
C13 > file.read = nc_open(file.name)
```

Above function nc_open reads a NetCDF file. The location of .nc file should be specified in the argument file.name of the function. The details of the .nc file are stored in the variable file.read.

```
C14 > file.read$dim$names
```

The variable file.read can be used to extract import file characteristics such as file dimensions, which conveys the layout of data stored in the .nc file. Above command prints out detailed description of all dimensions associated with the file.read variable.

```
C15 > file.read$var$names
```

Variables convey information about the data stored in the .nc files. Above command prints out detailed description of all dimensions associated with the file.read variable.

```
C16 > ncvar_get(file.read, var.name)
C17 > ncvar_get(file.read, varid=var.name, start=c(1,1,1), count=c(dim1,dim2,1))
C18 > sapply(1:10, function(x) ncvar_get(file.read, varid=var.name, start=c(1,1,x), count=c(dim1,dim2,1)))
```

The function “ncvar_get” is used to extract data stored in a particular variable in the .nc file. Command C16 will extract the values of a variable with name “var.name” stored in the .nc file. Sometimes due to data volume it is not possible or desirable to import and save all of the data in one variable in one go. In those cases, “ncvar_get” command can also be specified other attributes so that the data is read in manageable chunks. For instance command C17 provided above reads only the first array element of the variable: “var.name”. Variables “dim1” and “dim2” denote the x and y dimensions of the data array. Command C18 performs the same operation iteratively for array elements 1 to 10 using the “sapply” function. The indices for which the data needs to be extracted is ascertained by examining the time-indices for which data is provided in the .nc file and then finding indices that contain data for the user defined time-period of interest.

Above discussion provides a brief introduction on how R programming language can be used to access GCM and reanalysis climate data that is typically available in the .nc format using the ncdf4 package.

Next, we discuss how MODIS data can be extracted using R programming language. MODIS based climatic and land-cover data are very useful for performing SP and SPS model based downscaling and they can be easily extracted and managed using R programming language. We demonstrate the use of two packages: “MODISTools” and “MODIS” towards downloading and analysing MODIS data in R. The first package MODISTools can be used to download spatio-temporal MODIS data using “MODISSubsets” function provided below:

```
C19 > MODISSubsets(LoadDat, Products, Bands, Size)
```

In “MODISSubsets” function, argument “LoadDat” reads in a data-frame with details about coordinates and IDs of all locations where data needs to be extracted, as well as the start and end dates of the data to be downloaded. The argument “Products” reads in the product code, which can be obtained from the function: GetProducts(). The argument “Bands” is supplied the band names to be downloaded. For a particular product, a list of bands can be obtained using the Getbands() command. The argument “Size” is supplied with the spatial scale at which the data should be extracted. A value of c(0,0) is supplied if only data at the location of interest needs to be extracted. Other values such as c(1,1) provide values spatially averaged over an area of 2km^2 from the location of interest. Following command C20 will extract the day-time and night-time surface temperature data sensed by Terra satellite over the London (Ontario) city (lon = -81.25, lat=42.98) for the year: 2012.

```
C20>MODISSubsets(LoadDat=data.frame(lat=42.98,long=-81.25,start.date=2012,end.date=2012,id=1),
                 Products = “MOD11A1”,

                 Bands = c(“LST_Day_1km”, “LST_Night_1km”),

                 Size = c(0,0))
```

Downloading MODIS data using “MODISTools” is most advantageous when the data needs to be extracted at a limited number of locations. However, if data needs to be extracted over a region or a country, it is beneficial to extract a raster image encompassing the region of interest for analysis. The runGdal function provided in the MODIS package is extremely useful in such cases. Following is a description of the function:

```
C21 > runGdal(product, begin, end, extent)
```

In the “runGdal” function, the argument “product” needs to be supplied with the MODIS product ID, beginning and end dates for data extraction are supplied in the arguments: “begin” and “end” respectively. The area for which the data needs to be extracted is supplied through the “extent” argument. The area to be supplied can be selected interactively, by supplying a shapefile or a raster file, by specifying the country name, or an extent object. Other important arguments such as “outProj” which supplies the output raster projection, “pixelSize” which can be supplied with the output data spatial resolution, and “dataFormat” which can be used to specify the data-format of output raster image. The following command can be used to extract surface temperature data sensed by Terra satellite over Canada for the year: 2012.

```
C22 > runGdal(product="MOD11A1",begin="2012001",end = "2012366",extent="canada")
```

So far the discussion provided in this section dealt with the preparation of geophysical and climatic data required for the application of SP and SPS method downscaling models. Once the climatic and geophysical data are prepared, the downscaling models can be used to downscale GCM projections. We demonstrate this by downscaling future temperature and precipitation projections obtained from a GCM: MRI-CGCM3 under RCP2.6 using SP and SPS models. The data for downscaling model calibration and for making future projections can be downloaded from downscaling/sp. The datasets are provided in .rds format which is a format to save and load R objects as files in the system. File named: “Model calibration data.rds” contains data needed to calibrate SP and SPS models and file named: “Future prediction data.rds” contains data needed for making downscaled future climatic projections.

Once the folder is placed in a directory (termed as “Fakepath” below), the data can be imported into R using following commands:

```
C23 > cal.data=readRDS("Fakepath/Model calibration data.rds")
C24 > pred.data=readRDS("Fakepath/Future prediction data.rds")
```

The function “readRDS” used above reads in any R object with .rds extension. Here model calibration data are imported into an R data-frame object “cal.data” and prediction data are imported into a R object “pred.data”. The datasets can be examined using two very useful R functions: “summary” and “head”/ “tail” as shown below.

```
C25 > summary(cal.data)
C26 > head(cal.data, 50)
```

The former function provides a summary of the data stored in each column of the data-frame. The latter functions: “head” or “tail” show user requested number (which in above case is 50) of first or last few lines in a data-frame respectively.

provides a description of different columns in data-frames: “cal.data” and “val.data”.

SP method calibration can be performed using “gam” function available in the package “mgcv”. This can be performed with either of the following commands:

```
C28 > SP.mod.T = gam(T.obs ~ s(T.NARR) + LC+ s(elev), data = cal.data)

C29 > SP.mod.T.jan = gam(T.obs ~ s(T.NARR) + LC+ s(elev), data = subset(cal.data, month(date) %in% 1))
```

The command C28 fits a GAM function on the observations provided in the “cal.data” object and the resulting model is stored in another R object: “SP.mod”. If the calibration needs to be performed using data belonging to all days in the month of January, this can be done using command C29. The code subsets only the data belonging to January month by using “subset” function and checking which months corresponding to the “dates” column equal to 1. The model can be calibrated for other months in a similar fashion.

SPS model for downscaling temperature can be calibrated in a similar way. Command C30 calibrates SPS3x3 model using entire “cal.data” series and stores the calibrated model in an R object named: “SPS3x3.mod.T”. Appropriate modifications in predictor variables can be made when calibrating SPS models for other neighbourhood scales.

```
C30 > SPS3x3.mod.T = gam(T.obs ~ s(T.NARR) + s(LC) + s(elev) + s(C.3x3) + s(G.3x3) + s(BSV.3x3) +
                     s(OS.3x3) + s(W.3x3) + s(ENF.3x3) + s(UB.3x3) + s(DBF.3x3) + s(DNF.3x3) +
                     s(MF.3x3) + s(S.3x3) + s(WS.3x3) + s(CS.3x3) + s(EBF.3x3) + s(ef.elev.3x3), data =
                     cal.data)
```

Column name	Description
elev	Elevation of the gauging stations
P.obs	Precipitation recorded at the gauging stations
T.obs	Temperature recorded at the gauging stations
C.3x3/C.5x5/C.7x7/C.9x9	Cropland fraction in 3x3/5x5/7x7/9x9 neighborhood scale
G.3x3/G.5x5/G.7x7/G.9x9	Grassland fraction in 3x3/5x5/7x7/9x9 neighborhood scale

BSV.3x3/BSV.5x5/BSV.7x7/BSV.9x9	Barren or Sparsely Vegetated fraction in 3x3/5x5/7x7/9x9 neighborhood scale
OS.3x3/OS.5x5/OS.7x7/OS.9x9	Open Shrublands in 3x3/5x5/7x7/9x9 neighborhood scale
W.3x3/W.5x5/W.7x7/W.9x9	Water fraction in 3x3/5x5/7x7/9x9 neighborhood scale
ENF.3x3/ENF.5x5/ENF.7x7/ENF.9x9	Evergreen Needleleaf Forest fraction in 3x3/5x5/7x7/9x9 neighborhood scale
UB.3x3/UB.5x5/UB.7x7/UB.9x9	Urban fraction in 3x3/5x5/7x7/9x9 neighborhood scale
DBF.3x3/DBF.5x5/DBF.7x7/DBF.9x9	Deciduous Broadleaf Forest fraction in 3x3/5x5/7x7/9x9 neighborhood scale
DNF.3x3/DNF.5x5/DNF.7x7/DNF.9x9	Deciduous Needleleaf Forest fraction in 3x3/5x5/7x7/9x9 neighborhood scale
MF.3x3/MF.5x5/MF.7x7/MF.9x9	Mixed Forest fraction in 3x3/5x5/7x7/9x9 neighborhood scale
S.3x3/S.5x5/S.7x7/S.9x9	Savannas fraction in 3x3/5x5/7x7/9x9 neighborhood scale
WS.3x3/WS.5x5/WS.7x7/WS.9x9	Woody Savannas fraction in 3x3/5x5/7x7/9x9 neighborhood scale
CS.3x3/CS.5x5/CS.7x7/CS.9x9	Closed Shrublands fraction in 3x3/5x5/7x7/9x9 neighborhood scale
EBF.3x3/EBF.5x5/EBF.7x7/EBF.9x9	Evergreen Broadleaf Forest fraction in 3x3/5x5/7x7/9x9 neighborhood scale
ef.elev.3x3/ ef.elev.5x5/ ef.elev.7x7/ ef.elev.9x9	Relative elevation fraction in 3x3/5x5/7x7/9x9 neighborhood scale
T.NARR	NARR temperature interpolated at the gauging station location
P.NARR	NARR precipitation interpolated at the gauging station location
LC	Land-cover at the gauging station
GCM.ppt	GCM precipitation interpolated at the gauging station location
GCM.tas	GCM temperature interpolated at the gauging station location
date	Date associated with the data

Table 0.1 Descriptions of columns present in the calibration and prediction data

Performing downscaling using the calibrated models is also very straightforward and intuitive. For instance, following commands can be used to predict downscaled temperature data using above calibrated SP and SPS3x3 models:

```
C31 > colnames(pred.data)[which(colnames(pred.data) %in% "GCM.tas")]="T.NARR"
```

```
C32 > pred.data$SP.pred.T = predict(SP.mod.T, newdata = pred.data)

C33 > pred.data$SPS3x3.pred.T = predict(SPS3x3.mod.T, newdata = pred.data)
```

For making predictions using the calibrated GAM model, the predictor variables present in the calibration data should also be present in the validation data. Therefore the column "GCM.tas" which contains interpolated GCM data is renamed as "T.NARR" which contained NARR derived model data in the calibration dataset. This is performed by command C31 presented above. Following this, the calibrated models: "SP.mod" and "SPS3x3.mod" are used to downscale prediction dataset: "pred.data" using commands provided in the commands C32 and C33 above.

Precipitation downscaling by SP and SPS models involves two steps as discussed in section 1. First, dry and wet day sequences are predicted using a logistic regression model and then, wet day precipitation intensity is predicted using a GAM model. The calibration of logistic regression model and GAM models can be performed in R using following set of commands:

```
C34 > cal.data$r.switch=0

C35 > cal.data$r.switch[which(cal.data$P.obs>0.1)]=1

C36 > occ.mod.SP.P=glm(r.switch~P.NARR+elev+LC,data=cal.data,family="binomial")

C37 > int.mod.SP.P=gam(P.obs~s(P.NARR)+s(elev)+s(LC),data=subset(cal.data,r.switch==1))

C38 > occ.mod.SPS3x3.P = glm(r.switch ~ P.NARR + elev + LC + C.3x3 + G.3x3 + BSV.3x3 + OS.3x3 + W.3x3
                             + ENF.3x3 + UB.3x3 + DBF.3x3 + DNF.3x3 + MF.3x3 + S.3x3 + WS.3x3 +
                             CS.3x3 + EBF.3x3 + ef.elev.3x3, data=cal.data, family="binomial")

C39 > int.mod.SPS3x3.P = gam(P.obs ~ s(P.NARR) + s(elev) + s(LC) + s(C.3x3) + s(G.3x3) + s(BSV.3x3) +
                             s(OS.3x3) + s(W.3x3) + s(ENF.3x3) + s(UB.3x3) + s(DBF.3x3) + s(DNF.3x3) +
                             s(MF.3x3) + s(S.3x3) + s(WS.3x3) + s(CS.3x3) + s(EBF.3x3) + s(ef.elev.3x3),
                             data=subset(cal.data, r.switch==1))
```

In command C34 and C35 a new column with a rainfall switch defining rainfall (1) or no-rainfall (0) state is added to the calibration data-frame: “cal.data”. All days with daily precipitation magnitude greater than 0.1 mm is allotted a value of 1 in the column “r.switch” whereas all days with precipitation magnitudes less than or equal to 0.1 mm are allotted a value of 0. Command C36 calibrates the logistic regression model for SP model using “r.switch” as the predictant variable and columns: “P.NARR”, “elev” and “LC” columns as predictor variables. In R, logistic regression can be performed within the Generalised Linear Modelling (glm) framework. A glm framework in R can model regression functions of many families including: gaussian, binomial, poisson, gamma, inverse.gaussian, quasi. For defining a logistic regression, we specify the family of the regression function as: “binomial” and link function as: “logit” (not specified above as it is the default link function for binomial family in R). Command C37 calibrates a GAM model on rainfall intensities only using data for days when the value of “r.switch” is equal to 1 (in other words only using data for wet days). Commands C38 and C39 perform similar calibration of logistic regression and GAM models for the SPS3x3 downscaling model. The only difference is that in this case all neighbourhood predictors corresponding to the scale 3x3 are also used in defining the regression models.

Prediction of downscaled precipitation using SP and SPS models involves first predicting dry and wet day sequences and secondly predicting wet precipitation magnitudes. This can be performed in R using passing following set of commands:

```
C40 > colnames(pred.data)[which(colnames(pred.data) %in% "GCM.ppt")]="P.NARR"

C41 > pred.data$r.switch=as.numeric(predict(occ.mod.SP.P,newdata=pred.data,type="response"))

C42 > pred.data$r.switch[which(pred.data$r.switch>=0.5)]=1
```

```
C43 > pred.data$r.switch[which(pred.data$r.switch<0.5)]=0

C44 > pred.data$SP.pred.P=0

C45> pred.data$SP.pred.P[which(pred.data$r.switch>=0.5)] = as.numeric(predict(int.mod.SP.P, newdata =
     subset(pred.data, r.switch> = 0.5)))

C46 > pred.data$r.switch=as.numeric(predict(occ.mod.SPS3x3.P,newdata=pred.data,type="response"))

C47 > pred.data$r.switch[which(pred.data$r.switch>=0.5)]=1

C48 > pred.data$r.switch[which(pred.data$r.switch<0.5)]=0

C49 > pred.data$SPS3x3.pred.P=0

C50> pred.data$SPS3x3.pred.P[which(pred.data$r.switch>=0.5)] = as.numeric(predict(int.mod.SPS3x3.P,
     newdata = subset(pred.data, r.switch> = 0.5)))
```

The command C40 again renames the column “GCM.ppt” to “P.NARR” for prediction to ensure that the predictors chosen to calibrate the occurrence and precipitation intensity models are also present in the prediction dataset “pred.data”. The command C41 predicts the probabilities of a particular day to be rainy given the values of predictors: “P.NARR”, “LC”, and “elev” for each day. Next in lines C42 and C43 we choose a probability threshold that can be used to decide on the predictant state given the predicted probabilities. In Gaur and Simonovic (2017) this has been calibrated for different models. For this demonstration a value of 0.5 is chosen as the probability threshold value above which the predictant i.e. “r.switch” is considered as having a rainy (1) state or else it is considered to have a non-rainy (0) state. In lines C44 and C45 the values of precipitation for days with “r.switch” values equals to zero are taken to be zero. For days with “r.switch” equals to 1 (or for rainy or wet days) the precipitation magnitude is predicted using the model: “int.mod.SP.P”. Commands C46 to C50 perform precipitation downscaling using SPS3x3 model using a similar process but using the logistic regression model: “occ.mod.SPS3x3.P” and wet day precipitation magnitude prediction GAM model: “int.mod.SPS3x3.P”.

Finally, the results can be visualised effectively using several useful functions available in the “ggplot2” package. The function: “melt” in package “reshape2” is also extremely useful for preparing data to be used in “ggplot2” package. A small example is presented here where yearly maximums of GCM and downscaled precipitation data (from models: SP and SPS3x3) stored in the “pred.data” data-frame are plotted using “ggplot2” package.

```
C51 > ymax.P.GCM = sapply(2014:2100, function(x) max(subset(pred.data, year(date) %in% x)$P.NARR,
      na.rm=T))

C52 > ymax.P.SP = sapply(2014:2100, function(x) max(subset(pred.data, year(date) %in% x)$SP.pred.P,
      na.rm=T))

C53 > ymax.P.SPS3x3 = sapply(2014:2100, function(x) max(subset(pred.data, year(date) %in%
      x)$SPS3x3.pred.P, na.rm=T))

C54 > ymax.P=data.frame(year=2014:2100, GCM=ymax.P.GCM, SP=ymax.P.SP, SPS3x3=ymax.P.SPS3x3)

C55 > data.plot = melt(ymax.P, id="year")

C56 > ggplot()+geom_line(data=data.plot,aes(x=year,y=value,group=variable),size=1)+

             geom_point(data=data.plot,aes(x=year,y=value),size=3)+

             facet_wrap(~variable,scales="free")+theme_bw()+

             xlab("Year")+ylab("Precipitation(mm)")+

             theme(legend.title = element_text(size=22,face = "italic"),legend.position="bottom")+

             theme(axis.title.y=element_text(face="bold",size=22),

                   axis.title.x=element_text(face="bold",size=22),

                   text=element_text(size=22))+
                    theme(axis.text.x=element_text(size=18),axis.text.y=element_text(size=18))+
            theme(legend.key.width = unit(3,"cm"))

C57 > ggsave("Fakepath/Sample_plot.png",width=14,height=10)
```

The extraction of yearly maximum precipitation as simulated by GCMs and downscaling models: SP and SPS3x3 is performed in commands C51, C52 and C53 respectively. Next the yearly maximum results are aggregated into one data-frame: “ymax.P”. Thereafter the wide-format data-frame with 4 columns is converted into a long-format data-frame “ymax.P” with 3 columns using a function: “melt” from “reshape2” package. The long-format data-frame “data.plot” has three columns named: “year”, “variable”, and “value”. These three columns store the year values i.e. 2014 to 2100, variable values i.e. GCM, SP, and SPS3x3, and value of precipitation maximums (in mm) corresponding to each combination of year and variable name. The function ggplot() which we are using to plot the graphs needs an input data-frame in the long-format in order to do the plotting.

The data-frame “data.plot” is used to plot yearly maximum precipitation from GCMs, and SP, SPS3x3 downscaling models using the ggplot() function. As explained before, ggplot2 package is built on the grammar of graphics. It can be noted from command C56 that and data, aesthetic mapping, geometric object, statistical transformations, scales, coordinate system, position adjustments, and faceting arguments are passed along with the ggplot() function. The output generated from this command is shown in Figure 2. In the plot, the three panels show annual precipitation maximum values for GCM, SP, and SPS3x3 based precipitation projections. The generated plot can be saved by using the ggsave() function in C57 where among other arguments, the location where file needs to be saved, file-type, plot dimensions etc. are specified to save the plot in the system.

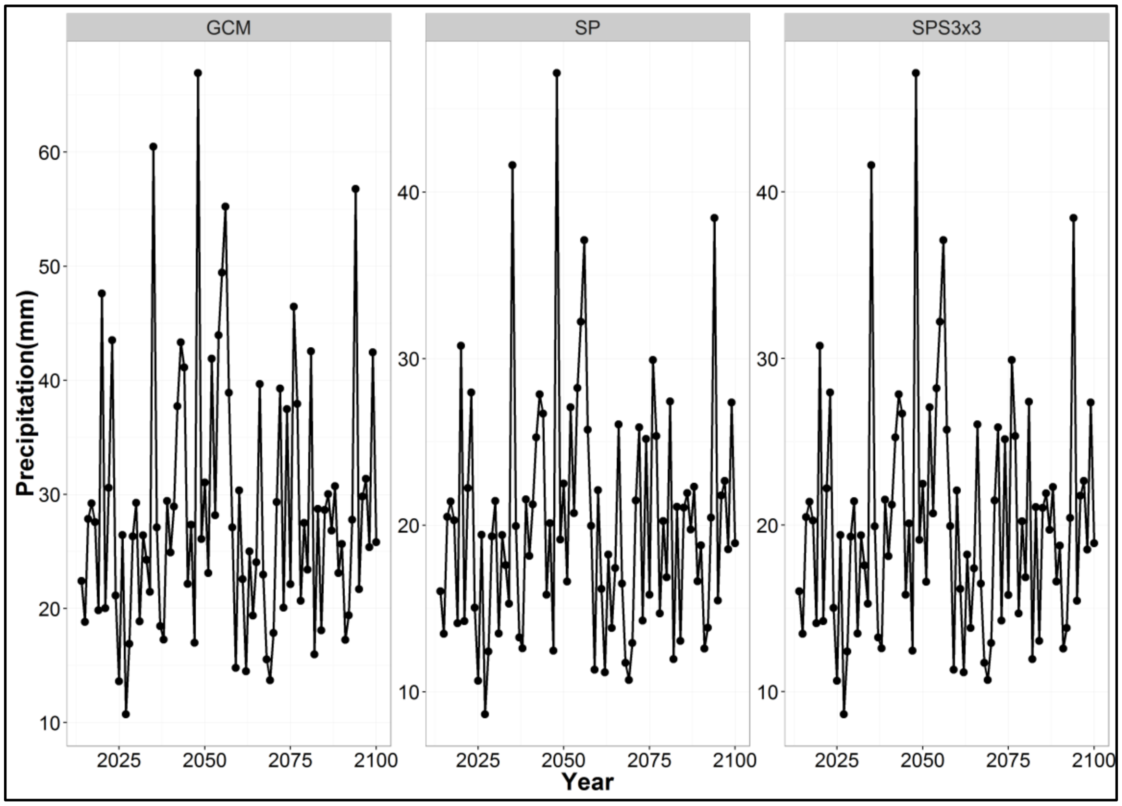


Figure 0.2 Annual precipitation maximum magnitudes (in mm) as plotted by the ggplot() function in R.

8 Summary

In this report we have discussed computational details of downscaling methods developed by the Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, The University of Western Ontario under the supervision of Prof. Slobodan P. Simonovic. In this manual, we have explained the implementation details of each method. The source code is also provided for each method in the Appendices to the manual.

Issues and bug reports should can be filed to our climate repository: https://github.com/FIDS-UWO/climate. Pull requests are also welcome to help improve or update these methods and their implementations. The KNN-CAD, MEBWG and BR model were tested for ten different climate stations in British Columbia, Canada (Table 0.1).

Table 0.1: Downscaling testing locations in the Campbell River basin, BC, Canada

Station	Elevation (m)	Latitude (^{o}N)	Longitude (^{o}W)
Elk R ab Campbell Lk	270	49.85	125.8
Eric Creek	280	49.6	125.3
Gold R below Ucona R	10	49.7	126.1
Heber River near Gold River	215	49.82	125.98
John Hart Substation	15	50.05	125.31
Quinsam R at Argonaut Br	280	49.93	125.51
Quinsam R nr Campbell R	15	50.03	125.3
Salmon R ab Campbell Div	215	50.09	125.67
Strathcona Dam	249	49.98	125.58
Wolf River Upper	1490	49.68	125.74

References

Anandhi A, Frei A, Pierson DC, Schneiderman EM, Zion MS, Lounsbury D, Matonse AH. 2011. Examination of change factor methodologies for climate change impact assessment. *Water Resources Research* **47**: 1–10 DOI: 10.1029/2010WR009104

Eum H-I, Simonovic SP. 2012. Assessment on variability of extreme climate events for the upper thames river basin in Canada. *Hydrological Processes* **26** (4): 485–499 DOI: 10.1002/hyp.8145

Gaur A, Simonovic SP. 2016a. Extension of physical scaling method and its application towards downscaling climate model based near surface air temperature. *International Journal of Climatology* DOI: 10.1002/joc.4921

Gaur A, Simonovic SP. 2016b. Accessing vulnerability of land-cover types to climate change using physical scaling downscaling model. *International Journal of Climatology* DOI: 10.1002/joc.4887

Gaur A, Simonovic SP. 2017. Application of physical scaling towards downscaling climate model precipitation data. *Theoretical and Applied Climatology (In Press)*

Goyal MK, Ojha CSP. 2010. Evaluation of various linear regression methods for downscaling of mean monthly precipitation in arid pichola watershed. *Natural Resources* **1** (1): 11–18 DOI: 10.4236/nr.2010.11002

Grolemund G, Vitalie Spinu, Wickham H, Lyttle I, Constigan I, Law J, Mitarotonda D, Larmarange J, Boiser J, Lee CH. 2016. Package ‘ lubridate ’ Available at: https://github.com/hadley/lubridate

Hay LE, McCabe GJ, Wolock DM, Ayers MA. 1991. Simulation of precipitation by weather type analysis. *Water Resources Research* **27** (4): 493–501 DOI: 10.1029/90WR02650

Hijmans R, van Etten J, Cheng J, Mattiuzzi M, Sumner M, Greenberg JA, Lamigueiro O, Bevan A, Racine E, Shortridge A. 2016. Package ‘ raster ’. CRAN -R.2.5-8 Available at: http://cran.r-project.org/package=raster

Hughes JP, Guttorp P. 1994. A class of stochastic models for relating synoptic atmospheric patterns to regional hydrologic phenomena. *Water Resources Research* **30** (5): 1535–1546 DOI: 10.1029/93WR02983

Kannan S, Ghosh S. 2010. Prediction of daily rainfall state in a river basin using statistical downscaling from GCM output. *Stochastic Environmental Research and Risk Assessment* **25** (4): 457–474 DOI: 10.1007/s00477-010-0415-y

King LM, Mcleod IA, Simonovic SP. 2015. Improved weather generator algorithm for multisite simulation of precipitation and temperature. *Journal of the American Water Resources Association* **7**: 1–16 DOI: 10.1111/1752-1688.12307

Mandal S, Srivastav RK, Simonovic SP. 2016. Use of beta regression for statistical downscaling of precipitation in the Campbell River basin, British Columbia, Canada. *Journal of Hydrology* **538**: 49–62 DOI: 10.1016/j.jhydrol.2016.04.009

Mattiuzzi M. 2016. Package ‘ MODIS ’ Available at: https://github.com/MatMatt/MODIS

Shepard D. 1968. A two-dimensional interpolation function for irregularly-spaced data. In *23rd ACM National Conference*517–524. DOI: 10.1145/800186.810616

Srivastav RK, Simonovic SP. 2014. Multi-site, multivariate weather generator using maximum entropy bootstrap. *Climate Dynamics* DOI: 10.1007/s00382-014-2157-x

Srivastav RK, Schardong A, Simonovic SP. 2016. Computerized tool for the development of Intensity-Duration-Frequency curves under a changing climate: technical manual Available at: http://www.eng.uwo.ca/research/iclr/fids/publications/products/89 v1.4.pdf

Team RDC. 2008. R: A language and environment Computing, Statistical. Available at: http://www.r-project.org

Wickham H. 2016. Package ‘reshape2’ Available at: ttps://github.com/hadley/reshape%0ABugReports

Wickham H, Chang W. 2016. Package ‘ggplot2’ Available at: http://ggplot2.tidyverse.org/

Wilby RL, Wigley TML. 1997. Downscaling general circulation model output: a review of methods and limitations. *Progress in Physical Geography* **21(4)**: 530–548 DOI: 10.1177/030913339702100403

Wood S. 2017. Package ‘mgcv’ DOI: 10.1186/1471-2105-11-11

Wood SN. 2000. Modelling and smoothing parameter estimation with multiple quadratic penalties. *Journal of the Royal Statistical Society (B)* **62** (2): 413–428

Appendices

Appendix A: Inverse Distance Interpolation Source Code

The Python source code for the inverse distance weighting interpolation method used in this manual is given below. It defines all of the relevant functions used to load the netCDF data (load_data), and perform the inverse distance weighting method (inv_dist). These two steps are combined in a single function (idw), which includes options for specifying the spatial extend, time bounds, alpha distance-decay parameter, and number of closest points to use for the interpolation.

```python
from netCDF4 import num2date, date2num
from datetime import datetime
import xarray as xr
import pandas as pd
import numpy as np


def idw(file, varname, stations, extent=None, period=None,
        alpha=2, k=4, **kwargs):
    """
    Extract inverse distance weighting interpolated time series from netcdf
    file for a list of stations.

    Parameters
    ----------
    file : str or list
        file path to netcdf file. A list can be used to load multiple files
        that are to be combined.
    varname : str
        Name of the variable in the netcdf file to be used.
    stations : dict
        A python dictionary containing key : value pairs, where the key is the
        station name, and the value is a tuple containing (lat, lon) , where
        lat is measured in degrees north and lon is measured in 0 to +
        360 degrees from Greenwich.
    extent : list, optional
        A list describing the spatial domain for which the
        interpolation will take place. This should look like
        `[north, east, south, west]`. This can greatly reduce the amount of
        resources required for interpolation.
    period : tuple, optional
        The time period for which the interpolation will take place. For
        example, `[(1950, 1, 1), (1975, 1, 1)]')` will first extract the data
        for this time period. Date formats must be in yyyy-mm-dd. This can also
        greatly reduce interpolation time and RAM usage.
    alpha : float, default 2
        The coefficient with which to calculate the inverse distance of the
        neighboring points of a given station.
    k : int, default 4
        Number of closest data points to use in the interpolation.

    Returns
    -------
    result : pandas.DataFrame
        A pandas data frame containing columns for each station with
        interpolated data along the rows. Index labels are Year, Month, Day.

    Notes
    -----
    1. Ensure that the spatial extent is large enough to encapsulate all
       stations of interest.
    """
    # Load the data
    data, lat, lon, dates = load_data(file, varname, extent, period, **kwargs)

    # Convert points tuple to array
    points = np.array(list(stations.values()))

    # Do the interpolation
    interpolated = inv_dist(data, lat, lon, points, k=k, alpha=alpha)

    # Put together index and columns for output DataFrame
    idx = pd.MultiIndex.from_tuples([(d.year, d.month, d.day) for d in dates])
    col = pd.MultiIndex.from_tuples([(varname, s) for s in stations])

    result = pd.DataFrame(interpolated, index=idx, columns=col)
    result.index.names = ['Y', 'M', 'D']
    result.columns.names = ['Variable', 'Station']

    return result.sort_index(axis=1)


def load_data(file, varname, extent=None, period=None, **kwargs):
    """
    Loads netCDF files and extracts data given a spatial extend and time period
    of interest.
    """
    # Open either single or multi-file data set depending if list of wildcard
    if "*" in file or isinstance(file, list):
        ds = xr.open_mfdataset(file, decode_times=False)
    else:
        ds = xr.open_dataset(file, decode_times=False)

    # Construct condition based on spatial extents
    if extent:
        n, e, s, w = extent
        ds = ds.sel(lat=(ds.lat >= s) & (ds.lat <= n))
        # Account for extent crossing Greenwich
        if w > e:
            ds = ds.sel(lon=(ds.lon >= w) | (ds.lon <= e))
        else:
            ds = ds.sel(lon=(ds.lon >= w) & (ds.lon <= e))

    # Construct condition base on time period
    if period:
        t1 = date2num(datetime(*period[0]), ds.time.units, ds.time.calendar)
        t2 = date2num(datetime(*period[1]), ds.time.units, ds.time.calendar)
        ds = ds.sel(time=(ds.time >= t1) & (ds.time <= t2))

    # Extra keyword arguments to select from additional dimensions (e.g. plev)
    if kwargs:
        ds = ds.sel(**kwargs)

    # Load in the data to a numpy array
    dates = num2date(ds.time, ds.time.units, ds.time.calendar)
    arr = ds[varname].values
    lat = ds.lat.values
    lon = ds.lon.values

    # Convert pr units to mm/day
    if ds[varname].units == 'kg m-2 s-1':
        arr *= 86400
    # Convert tas units to degK
    elif ds[varname].units == 'K':
        arr -= 273.15

    return arr, lat, lon, dates


def inv_dist(data, lat, lon, points, k=4, alpha=2):
    """
    Inverse distance point interpolation function from grid.

    Parameters
    ----------
    data : ndarray
        array of data with shape (n, m, l).
    lat : ndarray
        latitude array of shape (l,).
    lon : ndarray
        longitude array of shape (m,).
    points : ndarray
        array consisting of lon,lat points with shape (q, 2).
    k : int, default 4
        p closest points to use in inverse distance calculation.
    alpha : float, default 2
        coefficient with which to calculate the inverse distance of the
        neighboring points of a given station.

    Returns
    -------
    result : ndarray
        interpolated result of shape (n, q).
    """
    n, m, l = data.shape
    # Pre-allocate memory to resulting array
    result = np.zeros((n, points.shape[0]))

    # Get lon, lat grid
    xx, yy = np.meshgrid(lon, lat)

    for q, (x0, y0) in enumerate(points):
        # Calculate distance of each grid point
        dist = np.sqrt((xx - x0)**2 + (yy - y0)**2)

        # Rank distances for each point
        rank = np.argsort(dist, axis=None).reshape(m, l)

        # Mask for k closest points
        ma = rank > (rank.max() - k)

        # Calculate weighting of each of the grid points
        weights = dist[ma]**-alpha / np.sum(dist[ma]**-alpha)

        # Get the interpolated result
        result[:, q] = data[:, ma] @ weights

    return result
```

Appendix B: Change Factor Methodology Source Code

The Python source code for implementation of the change factor methodology used in this report is given here. It includes a single function to calculate the additive or multiplicative change factors from historical and future GCM output files. These change factors are then applied to the observed data file for a number of evenly spaced bins across the distribution of the given climate variable.

```python
import pandas as pd
import numpy as np


def cfm(his, fut, obs, method, bins=25):
    """
    Apply change factor methodology to scale hist data using fut and
    hist climate models.

    Parameters
    ----------
    his : pandas.DataFrame
        Historical GCM data
    fut : pandas.DataFrame
        Future GCM data
    obs : pandas.DataFrame
        Observed data
    method : int
        A 1d numpy array containing 0's and 1's to indicate the scaling method
        to use.
        * 0 - Apply additive scaling
        * 1 - Apply multiplicative scaling
    bins : int
        The number of bins to apply scaling separately for.

    Returns
    -------
    obs : pandas.DataFrame
       A future-scaled version of the observed dataset.
    """
    # Copy the data to a new DataFrame
    obs = obs.copy()
    # Define the bins
    q = np.linspace(0, 1, bins)
    # Define scaling functions
    scale = [lambda x, d: x + d[x.name],
             lambda x, d: x * d[x.name]]

    his_month = his.index.get_level_values(1)
    fut_month = fut.index.get_level_values(1)
    obs_month = obs.index.get_level_values(1)

    for c in obs.columns:
        for m in range(1, 12 + 1):
            # Find rows corresponding to month "m"
            ai = his_month == m
            bi = fut_month == m
            ci = obs_month == m

            if method == 1:
                # Account for zero precipitation
                ai &= his[c] > 0.01
                bi &= fut[c] > 0.01
                ci &= obs[c] > 0.01

            # Extract values for calendar month "m" and column "c"
            Am = his.loc[ai, c]
            Bm = fut.loc[bi, c]
            Cm = obs.loc[ci, c]

            # Calculate bins for each percentile range
            Ab = pd.cut(Am, Am.quantile(q), labels=False, include_lowest=True)
            Bb = pd.cut(Bm, Bm.quantile(q), labels=False, include_lowest=True)
            Cb = pd.cut(Cm, Cm.quantile(q), labels=False, include_lowest=True)

            # Group data based on bins
            Ag = Am.groupby(Ab)
            Bg = Bm.groupby(Bb)
            Cg = Cm.groupby(Cb)

            # Apply scaling transformation
            if method == 0:
                delta = Bg.mean() - Ag.mean()
            elif method == 1:
                delta = Bg.mean() / Ag.mean()

            obs.loc[ci, c] = Cg.transform(scale[method], delta)

    return obs
```

Appendix C: KNN-CAD Source Code

The Python source code for implementing the modified KNN-CADv4 algorithm is shown here. The only modification to this algorithm is in the calculation of distance for ranking the similarity of the current day to the L nearest neighbors. Previously, the covariance matrix for the L nearest neighbors was calculated, for which the first principal component was taken. The Mahalanobis distance was then used to compute the distance along the first principal components to rank the L nearest neighbors. The motive for this approach was to reduce computation time when several different types of climate variables were used. However, when inspecting the performance of this approach, it was found that using the Euclidean distance was more computationally efficient for any practical number of climate variables used. The synthetic climate series generated using this modified approach were also found to replicate the original climate statistics well with a much lower computational time.

For daily maximum and minimum temperature series this is demonstrated in Figures C.1 and C.2, which show that the monthly variation of mean and extremes are sufficiently well reproduced. In addition, the block sampling routine shows very little reduction in the temporal correlation of the historically observed climate. With regards to precipitation, Figure C.3 shows that the monthly total, daily mean, wet and dry spell lengths, and extremes are adequately reproduced.

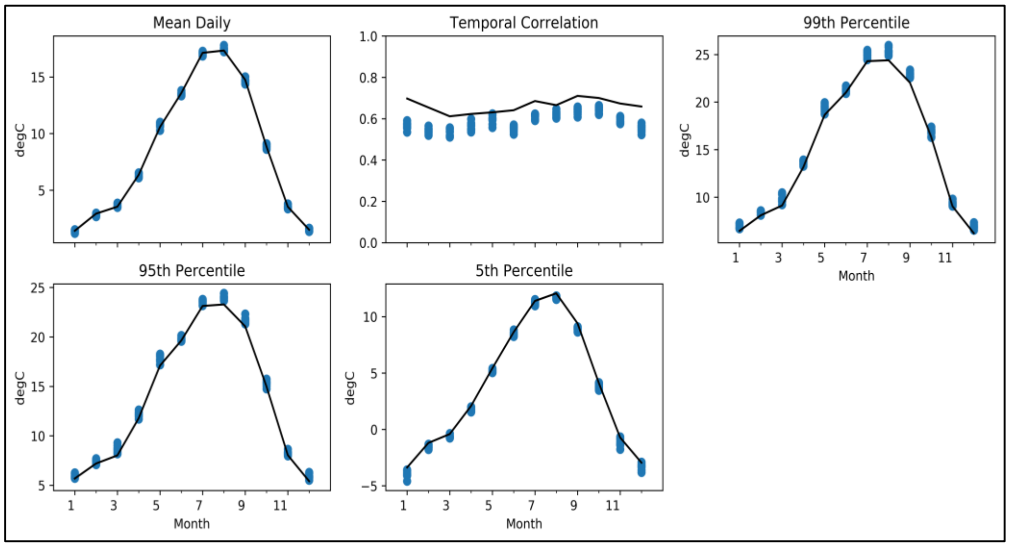


Figure C.1. Maximum temperature monthly climate statistics for 1950-2012 of the ELK meteorological station within the Campbell River Basin, British Columbia. The solid line represents historically observed data, while the points are from 10 synthetic climate replications using KNN-CAD.

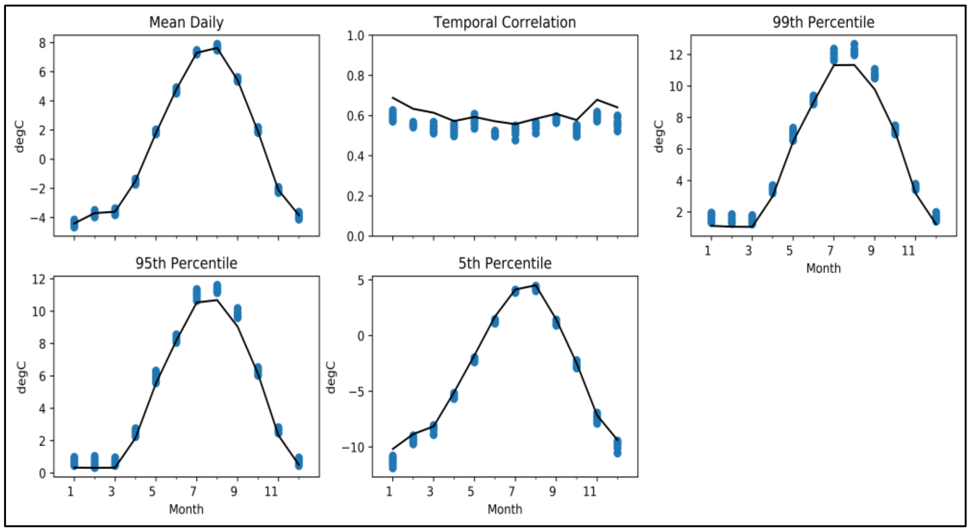


Figure C.2. Minimum temperature monthly climate statistics for 1950-2012 of the ELK meteorological station within the Campbell River Basin, British Columbia. The solid line represents historically observed data, while the points are from 10 synthetic climate replications using KNN-CAD.

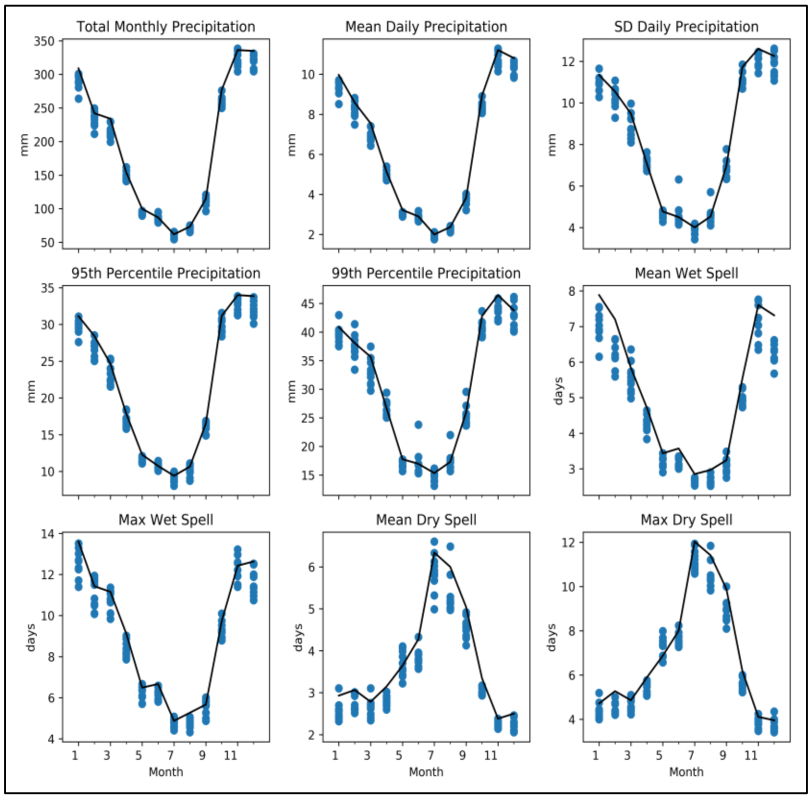


Figure C.3. Precipitation monthly climate statistics for 1950-2012 of the ELK meteorological station within the Campbell River Basin, British Columbia. The solid line represents historically observed data, while the points are from 10 synthetic climate replications using KNN-CAD.

Included below are the main functions used to implement this algorithm. They consist of:

(1) *knn:* This is the main function to run the algorithm, and includes options to specify the window and block sizes, as well as the perturbation type and lambda factor, and the number of replications to perform. These options were discussed in Section 4.

(2) *bootstrap*: This function performs the bootstrapping in order to resample the original climate series and generate the synthetic one.

(3) *perturb:* This function is used to perturb the resampled climate series using the perturbation method and factor specified. It was found previously to be a computational

bottleneck in the code, and has been JIT compiled to machine code to increase performance.

(4) *lnn_algorithm*: This function is used to obtain the L+1 nearest neighbor indices for each day of the year. The current day is remove during the resampling process to obtain the L nearest neighbors.

```python
import pandas as pd
import numpy as np

from numba import njit


def knn(X, P, w=14, B=10, interp=0.9, runs=1):
    """
    Function to run the KNN Weather Generator algorithm. Adapted from
    King et al. (2012) to use euclidean distance for L nearest neighbor
    selection as opposed to mahalanobis distance along 1st principal
    component.

    Parameters
    ----------
    X: pandas.DataFrame
        Input data following prescribed formatting.
    P: 1d array
        Assigns the perturbation type for each column in `X`.\n
        * 0 - No perturbation
        * 1 - Normal perturbation
        * 2 - Log-normal perturbation
    w: int
        Window size for the determination of the L and K nearest neighbors.
    B: int
        Block size for the block boostrap resampling.
    interp: float
        Level of influence of perturbation to be applied. 1 represents full
        perturbation while 0 represents no perturbation.
    runs: int
        The number of runs to be made with length equal to `X`.
    Returns
    -------
    result: pandas.DataFrame
        `result` is returned as a DataFrame with the same number of columns as
        `X` and length equal to `runs` multiplied by that of `X`.

    """
    # Standardize columns of input data
    Xn = (X - X.mean()) / X.std()
    # Group columns of like variables
    Xt = Xn.groupby(level=0, axis=1).mean()
    # Get year, month, and day from index
    year, mon, day = np.array(list(zip(*X.index.values)), dtype=np.uint32)

    if mon[0] != 1 or day[0] != 1 or mon[-1] != 12 or day[-1] != 31:
        raise ValueError("Data must start at Jan 1 and end at Dec 31")

    # Get day of year for all days
    doy = day_of_year(mon, day)
    # Apply block bootstrap to input data for all runs
    result = []
    for r in range(runs):
        # Get the bootstrapped values into a dataframe
        values = bootstrap(X.values, Xt.values, P, doy, w, B, interp)
        # Construct dataframe
        df = pd.DataFrame(values, X.index, Xn.columns)
        # Append dataframe to results list
        result.append(df)

    if len(result) > 1:
        return pd.concat(result,
                         axis=0,
                         keys=list(range(runs)),
                         names=['Run', 'Year', 'Month', 'Day'])
    else:
        return result[0]


def bootstrap(X, Xt, P, doy, w, B, interp):
    n, m = X.shape
    # Pre-allocate results array
    Xsim = np.zeros((n, m))
```

```
# Determine K
N = np.sum(doy == doy[0])
L = N * (w + 1) - 1
K = int(round(np.sqrt(L)))

# Get L nearest neighbors indices for each day of the year
lp1nn_idx = lnn_algorithm(doy, w, L)

# Generate cumulative probability distribution
pn = (np.ones(K) / np.arange(1, K + 1)).cumsum()
pn /= pn.max()

# Random selection of the first day
day1_idx, = np.where(doy == doy[0])

start = np.random.choice(day1_idx)

Xsim[:B] = X[start: start+B]

# Loop through each block of size B
for i in range(B, n, B):
    # Randomly select the day before or after leap day
    if doy[i] < 0:
        t = i + np.random.choice([-1, 1])
    else:
        t = i

    # Get L + 1 nearest neighbors for current day of the year
    lp1nn_t = lp1nn_idx[doy[t]]

    # Remove current day from L + 1 nearest neighbor indices
    lnn_idx = lp1nn_t[lp1nn_t != t]

    # Get euclidean distance of current day from L nearest neighbours
    dist = np.linalg.norm(Xt[lnn_idx] - Xt[t], axis=1)

    # Take the K nearest neighbours
    knn_idx = lnn_idx[dist.argsort()][:K]

    # Randomly draw K nearest neighbor from distribution
    nn = (abs(pn - np.random.rand())).argmin()
```

```
        # Set starting index of sampled block
        j = knn_idx[nn]

        # Adjust block size if at end of input series
        B = n - i if i + B > n else B
        # Adjust selected block if at end of series
        j = n - B if j + B > n else j

        # Obtain K and L nearest neighbors for non-spatially averaged data
        X_knn = X[knn_idx]
        X_lnn = X[lnn_idx]

        # Apply perturbation for each day in block for each variable
        Xsim[i: i+B] = perturb(X[j: j+B].copy(), X_knn, X_lnn, P, interp)

    return Xsim


@njit
def perturb(A, knn, lnn, P, interp):
    n, m = A.shape
    # Go through each day in block
    for i in range(n):
        z = np.random.randn()
        # Go through each variable in day
        for j in range(m):
            # Calculate random variate
            if P[j] == 0:
                continue
            elif P[j] == 1:
                std_knn = np.sqrt(var_i(knn, j))
                z_j = np.random.normal(A[i, j], std_knn)
            else:
                if A[i, j] < 0.001:
                    continue
                else:
                    var_lnn = var_i_nonzero(lnn, j)
                    bm = np.sqrt(np.log10(var_lnn / A[i, j] + 1))
                    am = np.log10(A[i, j]) - 0.5 * bm
                    z_j = np.exp(am + bm * z)
            # Apply perturbation
            A[i, j] = interp * A[i, j] + (1 - interp) * z_j

    return A


def lnn_algorithm(doy, w, L):
    """
    Function to obtain the indices of the L+1 (includes current day) nearest
    neighbors given the day of year, window size, and value of L.

    Parameters
    ----------
    doy : ndarray
        An array consisting of the day of year for all data points. February
        29th should be marked as a negative number as it will not be considered
        for the L nearest neighbors.
    w : int
        Window size for number of days surrounding current day in the L nearest
        neighbors. This should be an even number.
    L : int
        The value of L. Calculated from L = N * (w + 1) - 1 where N is the
        number of years in the historical data.

    Returns
    -------
    lnnp1 : ndarray
        Output of shape (365, L + 1) corresponds to the indices of the L + 1
        nearest neighbors for each day of the year.
    """

    lnnp1 = np.zeros((365, L + 1), dtype=np.uint32)

    for i in range(365):
        # Set left and right bounds for window day of year (DOY)
        w_l = i - w // 2
        w_r = i + w // 2

        # Adjust bounds for DOY less than 0 and greater than 364
        if w_l < 0:
            w_l += 365
        if w_r > 364:
            w_r -= 365
        # Obtain indices of days falling within window for current DOY
        if (w_r - w_l) != w:
            lnnp1[i], = np.where(((w_l <= doy) | (w_r >= doy)) & (doy >= 0))
        else:
            lnnp1[i], = np.where((w_l <= doy) & (w_r >= doy))

    return lnnp1


@njit
def day_of_year(mon, day):
    doy = np.zeros(day.size, dtype=np.int32)
    count = 0
    for i, d in enumerate(day):
        if mon[i] == 2 and day[i] == 29:
            doy[i] = -1
        else:
            doy[i] = count % 365
            count += 1

    return doy


@njit
def var_i(arr, j):
    n = arr.shape[0]
    sum_i = 0
    for i in range(n):
        sum_i += arr[i, j]
    mean_i = sum_i / n
    var = 0
    for i in range(n):
        var += (arr[i, j] - mean_i) ** 2

    return var


@njit
def var_i_nonzero(arr, j):
    n = arr.shape[0]
    sum_i = 0
    for i in range(n):
        if arr[i, j] >= 0.001:
            sum_i += arr[i, j]
    mean_i = sum_i / n
    var = 0
    for i in range(n):
        if arr[i, j] >= 0.001:
            var += (arr[i, j] - mean_i) ** 2

    return var
```

Appendix D : MEBWG Source code

MBE_input.m

```
% PROGRAM TO GENERATE MULTI-VARIATE MULTI-SITE WEATHER DATA
% ROSHAN K SRIVASTAVA
% CREATED ON: JULY 25, 2013
% Modified By: SOHOM MANDAL
% Modification Date: March 1, 2017

clear all
close all
clc
% Number of Replicates: User specific; Use a Positive interger
prompt = {'Please provide number of replicates (should be an integer):'};
dlg_title = 'Input';
num_lines = 1;
defaultans = {'2'};
answer =(inputdlg(prompt,dlg_title,num_lines,defaultans));
numrep = str2num(answer{:});
if isnan(numrep) || fix(numrep) ~= numrep
  disp('Please enter a positive integer ')
end
disp('Code is running! Please wait');
%File name reading
z=dir('*pr*.csv'); % Reading precipitation files name
z1=dir('*tasmax*.csv'); %  Reading maximum temperature files name
z2=dir('*tasmin*.csv'); % Reading minimum temperature files name
% Working through the files
for j=1:length(z)
%     h = waitbar(length(z),'Downsacling');
    varname_pr=z(j).name;
    varname_tmax=z1(j).name;
    varname_tmin=z2(j).name;
    expression = ('\_');
    splitStr_pr=regexp(varname_pr,expression,'split'); % split the Pr files
name using '_' as regular expression
    splitStr_tmax=regexp(varname_tmax,expression,'split'); % split the Tmax
files name using '_' as regular expression
    splitStr_tmin=regexp(varname_tmin,expression,'split'); % split the Tmin
files name using '_' as regular expression

m1=strcmp(strcat(splitStr_pr(5),splitStr_pr(6),splitStr_pr(7)),strcat(splitSt
r_tmax(5),splitStr_tmax(6),splitStr_tmax(7)));

m2=strcmp(strcat(splitStr_tmax(5),splitStr_tmax(6),splitStr_tmax(7)),strcat(s
plitStr_tmin(5),splitStr_tmin(6),splitStr_tmin(7)));
    if m1==1 && m2==1 % Compare files name

% LOAD YOUR INPUT FILES
% Each file contains data for a particular weather variable
% NOTE: Rows and Colums represent time and stations respectively
    [ppt header1]=xlsread(z(j).name); %Read Precipitation data
    tmax=csvread(z1(j).name,1,3); %Read Maximum temperature data
    tmin=csvread(z2(j).name,1,3); %Read Minimum temperature data
    [r,c] = size(ppt);
    % Modify this depending on the number of variables added
    data_all = [ppt(:,4:end)+0.1 tmax+273.15 tmin+273.15];

    % DO NOT MODIFY ANY THING FROM HERE IF YOU ARE NOT SURE
    date=repmat(ppt(:,1:3),numrep,1);
    numobs = length(data_all);
    [m1,n1] = size(data_all);
    [data_all_std,data_all_mu,data_all_sig] = zscore(data_all);
    [loadings, scores, variances, tscores] = princomp(data_all_std);
    rand('state',0);
    cm = 1;
    [sc_fit,debug] = MBE_RKS(scores(:,cm),numrep,[],[],1,1);
    scores_fit = scores;
    loadings_fit = loadings;
    sds = data_all_sig;
    means=data_all_mu;
    c1 =[];
    for i = 1:numrep
        scores_fit(:,cm) = sc_fit(:,i);
        b1 = loadings_fit*scores_fit';
        d1 = (b1' .*  repmat(sds,numobs,1) + repmat(means,numobs,1));
        d1(d1(:,1:22)<0)=0;
        c1 = [c1;d1];
        d1=[];b1=[];
    end
    % WRITE OUTPUT FILE
    newname=strcat('MBE','_',splitStr_pr(3),'_',
splitStr_pr(5),'_',splitStr_pr(6)); % Create output file name; User can
modify
    c1=[date c1(:,1:10) c1(:,11:end)-273.15];
    %Pr= Precipitation; Tasmax=Maximum Temperature;
    %Tasmin=Minimum Temperature; Header will look like:  "Pr_(Station Name)"

mod_header=horzcat(header1(:,1:3),strcat('Pr_',header1(:,4:end)),strcat('Tasm
ax_',header1(:,4:end)),strcat('Tasmin_',header1(:,4:end))); % Change the
header if more than three variables are used
    filename= cell2mat(strcat(newname, '.csv'));
    csvwrite_with_headers(filename,c1, mod_header);% Writes the output file
into csv format
    else
        h=fprintf(2,'Error: GCM files are not matching!!\n'); % Error message
for not matching the GCMs file
    end


end
 if h==36
     msgbox('Downscaling is not Completed'); % message box for not complete
the work
 else
     msgbox('Downscaling Completed'); % message box for not complete the work
 end
```

MBE_RKS.m

```
function [synth,t3]  = MBE_RKS(a1,numrep,dummy1,dummy2,dummy3,dummy4)
% MEB Beta
% ROSHAN
% CREATED ON: JULY 21, 2013

% clear all
% clc
% close all
%
% load APPLE.dat
% data = sum(APPLE,2);
% a = [[1:length(data)]' data];
%a =[1 2 3 4 5; 4 12 36 20 8]';
% rand('state',99)
dummy  =  dummy1;
dummyx =  dummy2;

a = [[1:length(a1)]' a1];
[y,i1] = sort(a);
aa = [i1(:,2) y(:,2)];
t2 =[a aa];
aa1 = (aa(1:end-1,2)+aa(2:end,2))/2;
aa2 = mean(abs(a(2:end,2)-a(1:end-1,2)));
int_mid_points = [aa(1,2)-aa2; aa1; aa(end,2)+aa2];
dmean = zeros(length(a),1);
for i = 1:length(a)
    if i==1, dmean(i,1) = 0.75* aa(i,2) + 0.25*aa(i+1,2);
    elseif i==length(a), dmean(i,1) = 0.25* aa(i-1,2) + 0.75*aa(i,2);
    else dmean(i,1) = 0.25* aa(i-1,2) + 0.5*aa(i,2)+0.25* aa(i+1,2);
    end
end
t3 =[t2 dmean];
% figure
% plot(int_mid_points,'o-')

interval = zeros(length(a),1);
density = zeros(length(a),1);
for i = 1:length(a)
    interval(i,1)= abs(int_mid_points(i+1,1)-int_mid_points(i,1));
    density(i,1) = 1/(abs(int_mid_points(i+1,1)-
int_mid_points(i,1))*length(a));
end
interval_den = [interval density];

denp=[];
for i = 1:length(aa)
    denp =[denp; int_mid_points(i,1) density(i,1);int_mid_points(i+1,1)
density(i,1)];
end
% figure
% plot(denp(:,1),denp(:,2))

cum_den = [0; cumsum(prod(interval_den,2))];
```

```
% figure

imp_points = [cum_den int_mid_points];

% numrep = 999;
synth = [];
for k=1:numrep
    uni_draw = sort(rand(length(a),1));
    %uni_draw = sort([0.12 0.83 0.53 0.59 0.11])';

    inp_data = zeros(length(a),1);
    for i = 1:length(a)
        check = 1;
        j=1;
        while j<=length(a)
            if uni_draw(i,1)>=cum_den(j,1) && uni_draw(i,1)<=cum_den(j+1,1)
                checkp = j;
                check =0;
            else
                check = 1;
            end
            if check == 0, break, end;
            j=j+1;
        end
        aa = int_mid_points(checkp,1);
        bb = int_mid_points(checkp+1,1);
        x1 = cum_den(checkp,1);
        x2 = cum_den(checkp+1,1);

        inp_data(i,1) = aa + ((uni_draw(i,1)-x1)*(bb-aa)/(x2-x1));
    end
    rep = zeros(length(a),1);
    for i=1:length(a)
        rep(i1(i,2),1)= inp_data(i,1);
    end
    synth = [synth rep];
end
```

csvwrite_with_headers.m

```
% This function functions like the build in MATLAB function csvwrite but
% allows a row of headers to be easily inserted
%
% known limitations
%   The same limitation that apply to the data structure that exist with
%   csvwrite apply in this function, notably:
%       m must not be a cell array
%
% Inputs
%
%   filename    - Output filename
%   m           - array of data
%   headers     - a cell array of strings containing the column headers.
%                 The length must be the same as the number of columns in m.
%   r           - row offset of the data (optional parameter)
%   c           - column offset of the data (optional parameter)
%
%
% Outputs
%   None
function csvwrite_with_headers(filename,m,headers,r,c)

%% initial checks on the inputs
if ~ischar(filename)
    error('FILENAME must be a string');
end

% the r and c inputs are optional and need to be filled in if they are
% missing
if nargin < 4
    r = 0;
end
if nargin < 5
    c = 0;
end

if ~iscellstr(headers)
    error('Header must be cell array of strings')
end



if length(headers) ~= size(m,2)
    error('number of header entries must match the number of columns in the
data')
end

%% write the header string to the file

%turn the headers into a single comma seperated string if it is a cell
%array,
header_string = headers{1};
for i = 2:length(headers)
    header_string = [header_string,',',headers{i}];
```

```
end
%if the data has an offset shifting it right then blank commas must
%be inserted to match
if r>0
    for i=1:r
        header_string = [',',header_string];
    end
end

%write the string to a file
fid = fopen(filename,'w');
fprintf(fid,'%s\r\n',header_string);
fclose(fid);

%% write the append the data to the file

%
% Call dlmwrite with a comma as the delimiter
%
dlmwrite(filename, m,'-append','delimiter',',','roffset', r,'coffset',c);
```

Appendix E: Source Code for Beta Regression (BR) Based Downscaling Method

Input BR.m

```
% PROGRAM TO GENERATE MULTI-SITE PRECIPITATION DATA USING BETA REGRESSION
% SOHOM MANDAL
% CREATED ON: MARCH 05, 2017
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run Beta Regression with GCM historical data predictors%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run this file Beta regression
clc
clear
% Number of Replicates: User specific; Use a Positive interger
prompt = {'Starting Year e.g. 1984 (Historical):','Ending Year e.g. 2000
(Historical):','Starting Year e.g. 2040 (Future):', 'Ending Year e.g. 2050
(Future):'};
dlg_title = 'Input: Timeframe';
num_lines = 1;
defaultans = {'1976','2005','2045','2055'};
answer =(inputdlg(prompt,dlg_title,num_lines,defaultans));
Histstart_time = str2num(answer{1});
Histend_time=str2num(answer{2});
Futstart_time = str2num(answer{3});
Futend_time=str2num(answer{4});

Pr_Obs = uigetfile({'*.csv'},'Pick a Observed Historical Precipitation
File');
fprintf(1,'Reading the data!!\n')
y=xlsread(Pr_Obs);
GCM_Hist = uigetfile({'*.csv'},'GCM Predictor Variables (Historical)');
fprintf(1,'Reading the data!!\n')
x=xlsread(GCM_Hist);
GCM_Fut = uigetfile({'*.csv'},'GCM Predictor Variables (Future)');
fprintf(1,'Reading the data!!\n')
z=xlsread(GCM_Fut);
disp('Code is running! Please wait');
expression = ('\_');
splitStr1=regexp(GCM_Hist,expression,'split');
splitStr=regexp(GCM_Fut,expression,'split');
tf = strcmp(splitStr1(3),splitStr(3));
if tf==1; % Compare files name; making sure same GCMs data are using
    y(y(:,2)==2 & y(:,3)==29,:)=[]; % Remove the leap year date
    y=y((Histstart_time<=y(:,1) & y(:,1)<=Histend_time), :); % Slicing the
data set according to input dataframe
    x(x(:,2)==2 & x(:,3)==29,:)=[]; % Remove the leap year date
    x=x((Histstart_time<=x(:,1) & x(:,1)<=Histend_time), :); % Slicing the
data set according to input dataframe
    z(z(:,2)==2 & z(:,3)==29,:)=[]; % Remove the leap year date
    z=z((Futstart_time<=z(:,1) & z(:,1)<=Futend_time), :); % Slicing the data
set according to input dataframe
    Simulated_Precipitation=Beta_Regression(x,y,z); % Calling the beta
regression and running files
```

```
   %## WRITE OUTPUT FILES

filename=num2str(cell2mat(strcat('BR','_',splitStr(3),'_',splitStr(4),'_',spl
itStr(5),'_',splitStr(6))));
 col_header={'Year', 'Month', 'Day',
'ELK','ERC','GLD','HEB','JHT','QIN','QSM','SAM','SCA','WOL'};
 xlswrite(filename,Simulated_Precipitation,'Sheet1','A2');     %Write data
 xlswrite(filename,col_header,'Sheet1','A1');     %Write column header
h = msgbox('Downscaling Completed'); % message box for complete the work
else
    fprintf(2,'Error: GCM files are not matching!!\n')
end
```

Beta_Regression.m

```
function [Simulated_Precipitation] = Beta_Regression(x,y,z)
% THIS CODE IS FOR CART AND PCA
%x=Training period Predictor Variables e.g. tasmax, tasmin,psl,mslp,ua,va
%y=Training period Predictand Variable e.g. Precipitation
%z=Testing period Predictor Variables (data points where regression value
%will be calculated)
% Length of Tranning predictor data and predictand should be same
if length(x)~=length(y);
    disp('Input Matrix length should be same for Predictor and Predictand ')
end

% Matrix dimension should be same for training period and testing period
predictors
if size(x,2)~=size(z,2)
    disp('Training and Testing period predictors dimension is not same')
end

Traning_Predictor=x(:,4:end);
Tranning_Predictand=y(:,4:end);
Testing_Predictor=z(:,4:end);
Testing_Predictor_Date=z(:,1:3);
% Kmeans clustering for rainfall state
rand('state',0);
% Three clusters has taken for clustering, do cluster validation before
% choose the no of clusters
k=3;
[IDX,C,sumd,D]= kmeans(Tranning_Predictand,k); % IDX is the rainfall state
for observed data

% Normaization of the Predictor variable (1960-1990)
[Z,mu,sigma] = zscore(Traning_Predictor);
%PCA
[pc,score1,latent1] = princomp(Z);
Var=(cumsum((latent1)./sum(latent1))*100);
% Find the variance which is less or equal to 98%
Ln_var_explained=length(find(Var<=98));
%Buliding classification Tree
T=classregtree(score1(2:end,1:Ln_var_explained),IDX(1:end-1,:));
%sub_mean_data=bsxfun(@minus, Testing_Predictor, mu); % substraction from
mean
%Testing_Predictor=bsxfun(@rdivide,sub_mean_data,sigma);% %Standarization
future predictor data
Temp_val=zscore(Testing_Predictor)*pc;
Rain_state_Prediction_Traning_Period=T(Temp_val(:,1:Ln_var_explained));

% Vector Space of observed data pr on the basis of rainfall state tranning
% period
Observed_data_pr_Rainfall_state=[Tranning_Predictand IDX];
observed_pr_data_state_1=(Observed_data_pr_Rainfall_state(Observed_data_pr_Ra
infall_state(:,end)==1, 1:end-1));
% Scaling the data in range (0,1)
```

```
Pr_tranning_1=bsxfun(@times,(bsxfun(@minus, observed_pr_data_state_1,
min(observed_pr_data_state_1))), (1./(max(observed_pr_data_state_1)-
min(observed_pr_data_state_1))));
Tranning_Predictand_state1=((Pr_tranning_1*(length(Pr_tranning_1)-
1))+0.5)/length(Pr_tranning_1);
observed_pr_data_state_2=(Observed_data_pr_Rainfall_state(Observed_data_pr_Ra
infall_state(:,end)==2, 1:end-1));
% Scaling the data in range (0,1)
Pr_tranning_2=bsxfun(@times,(bsxfun(@minus, observed_pr_data_state_2,
min(observed_pr_data_state_2))), (1./(max(observed_pr_data_state_2)-
min(observed_pr_data_state_2))));
Tranning_Predictand_state2=((Pr_tranning_2*(length(Pr_tranning_2)-
1))+0.5)/length(Pr_tranning_2);

observed_pr_data_state_3=(Observed_data_pr_Rainfall_state(Observed_data_pr_Ra
infall_state(:,end)==3, 1:end-1));
% Scaling the data in range (0,1)
Pr_tranning_3=bsxfun(@times,(bsxfun(@minus, observed_pr_data_state_3,
min(observed_pr_data_state_3))), (1./(max(observed_pr_data_state_3)-
min(observed_pr_data_state_3))));
Tranning_Predictand_state3=((Pr_tranning_3*(length(Pr_tranning_3)-
1))+0.5)/length(Pr_tranning_3);

%Vector Space of observed data predictor(Temp) on the basis of rainfall
%state tranning period
Observed_data_predictor_Rainfall_state=[score1(:,1:Ln_var_explained) IDX];
Observed_predictor_data_state_1=(Observed_data_predictor_Rainfall_state(Obser
ved_data_predictor_Rainfall_state(:,end)==1, 1:end-1));
Observed_predictor_data_state_2=(Observed_data_predictor_Rainfall_state(Obser
ved_data_predictor_Rainfall_state(:,end)==2, 1:end-1));
Observed_predictor_data_state_3=(Observed_data_predictor_Rainfall_state(Obser
ved_data_predictor_Rainfall_state(:,end)==3, 1:end-1));
%Vector Space of testing data(Predictor:Temp)on the basis of rainfall state
Testdata_predictor_Rainfall_state=[Testing_Predictor_Date
Temp_val(:,1:Ln_var_explained) Rain_state_Prediction_Traning_Period];
Testdata_state_1=
Testdata_predictor_Rainfall_state(Testdata_predictor_Rainfall_state(:,end)==1
, 1:end-1);
Testdata_state_2=
Testdata_predictor_Rainfall_state(1<Testdata_predictor_Rainfall_state(:,end)
& Testdata_predictor_Rainfall_state(:,end)<=2, 1:end-1);
Testdata_state_3=
Testdata_predictor_Rainfall_state(Testdata_predictor_Rainfall_state(:,end)>2,
1:end-1);

for i=1:10
 %Bulid regression for state I
mX1=[ones(length(Observed_predictor_data_state_1),1)
Observed_predictor_data_state_1];
vy1=Tranning_Predictand_state1(:,i);
vP1=betareg_main(vy1,mX1);
%Bulid regression for state II
mX2=[ones(length(Observed_predictor_data_state_2),1)
Observed_predictor_data_state_2];
vy2=Tranning_Predictand_state2(:,i);
vP2=betareg_main(vy2,mX2);
```

```
%Bulid regression for state III
mX3=[ones(length(Observed_predictor_data_state_3),1)
Observed_predictor_data_state_3];
vy3=Tranning_Predictand_state3(:,i);
vP3=betareg_main(vy3,mX3);
% Calculate the precipitation for Testing period or Validation period
Predicted_Rain_State1=[ones(length(Testdata_state_1),1)
Testdata_state_1(:,4:end)]*vP1(2:end);
Predicted_Rain_State2=[ones(length(Testdata_state_2),1)
Testdata_state_2(:,4:end)]*vP2(2:end);
Predicted_Rain_State3=[ones(length(Testdata_state_3),1)
Testdata_state_3(:,4:end)]*vP3(2:end);

Rain(:,i)=[Predicted_Rain_State1;Predicted_Rain_State2;Predicted_Rain_State3]
;
Rain(Rain<0)=0;
end
% Arrange the Date for Validation or Testing period
Date=[datenum(Testdata_state_1(:,1:3));datenum(Testdata_state_2(:,1:3));daten
um(Testdata_state_3(:,1:3))];
%combine the data (simulated precipiation with date)
Predcited_Precipitation=[Date Rain];
%sort the data based date
Precipitation=sortrows(Predcited_Precipitation,1);
% Final bind of simulated precipitation data with time
Simulated_Precipitation=[Testing_Predictor_Date Precipitation(:,2:end)];
end
```

betareg_main.m

```
%IF YOU NOT SURE PLEASE DON'T CHNAGE ANYTHING HERE
function [vP, muhat]= betareg(vy, mX)
format short g;
n = length(vy);
p = size(mX,2);

if(max(vy) >= 1 || min(vy) <= 0)
    error(sprintf('\n\nERROR: DATA OUT OF RANGE (0,1)!\n\n'));
end

if(p >= n)
     error(sprintf('\n\nERROR: NUMBER OF COVARIATES CANNOT EXCEED NUMBER OF
OBSERVATIONS!\n\n'));
end

ynew = log( vy ./ (1-vy) );


if(p > 1)
     betaols = (mX \ ynew);
elseif(p==1)
     betaols = (mean(ynew));
end

olsfittednew = mX*betaols;

olsfitted = exp(olsfittednew) ./ (1 + exp(olsfittednew));
olserrorvar = sum((ynew-olsfittednew).^2)/(n-p);

ybar = mean(vy);
yvar = var(vy);

% starting values
vps = [betaols;(mean(((olsfitted .* (1-olsfitted))./olserrorvar)-1))];
options = optimset('Display','off');
vP = fminsearch(@(vP) betalik(vP, mX, vy), abs(vps),options);
% k_OptimOptions = optimset('Display','off');
etahat = mX*vP(1:p);
muhat = exp(etahat ) ./ (1+exp(etahat));
phihat = vP(p+1);
end
```

betalik.m

```
function y = betalik(vP, mX, vy)
k = length(vP);
eta = mX*vP(1:k-1);
mu = exp(eta) ./ (1+exp(eta));

phi = vP(k);
y = -sum( gammaln(phi) - gammaln(mu*phi)- gammaln(abs(1-mu)*phi) + ((mu*phi-
1) .* log(vy)) + ( (1-mu)*phi-1 ) .* log(1-vy) );
```

Appendix F: List of Previous Reports in the Series

ISSN: (Print) 1913-3200; (online) 1913-3219
In addition to 78 previous reports (No. 01 – No. 78) prior to 2012

Samiran Das and Slobodan P. Simonovic (2012). Assessment of Uncertainty in Flood Flows under Climate Change. Water Resources Research Report no. 079, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 67 pages. ISBN: (print) 978-0-7714-2960-6; (online) 978-0-7714-2961-3.

Rubaiya Sarwar, Sarah E. Irwin, Leanna King and Slobodan P. Simonovic (2012). Assessment of Climatic Vulnerability in the Upper Thames River basin: Downscaling with SDSM. Water Resources Research Report no. 080, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 65 pages. ISBN: (print) 978-0-7714-2962-0; (online) 978-0-7714-2963-7.

Sarah E. Irwin, Rubaiya Sarwar, Leanna King and Slobodan P. Simonovic (2012). Assessment of Climatic Vulnerability in the Upper Thames River basin: Downscaling with LARS-WG. Water Resources Research Report no. 081, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 80 pages. ISBN: (print) 978-0-7714-2964-4; (online) 978-0-7714-2965-1.

Samiran Das and Slobodan P. Simonovic (2012). Guidelines for Flood Frequency Estimation under Climate Change. Water Resources Research Report no. 082, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 44 pages. ISBN: (print) 978-0-7714-2973-6; (online) 978-0-7714-2974-3.

Angela Peck and Slobodan P. Simonovic (2013). Coastal Cities at Risk (CCaR): Generic System Dynamics Simulation Models for Use with City Resilience Simulator. Water Resources Research Report no. 083, Facility for Intelligent Decision Support, Department of Civil and Environmental

Engineering, London, Ontario, Canada, 55 pages. ISBN: (print) 978-0-7714-3024-4; (online) 978-0-7714-3025-1.

Roshan Srivastav and Slobodan P. Simonovic (2014). Generic Framework for Computation of Spatial Dynamic Resilience. Water Resources Research Report no. 085, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 81 pages. ISBN: (print) 978-0-7714-3067-1; (online) 978-0-7714-3068-8.

Angela Peck and Slobodan P. Simonovic (2014). Coupling System Dynamics with Geographic Information Systems: CCaR Project Report. Water Resources Research Report no. 086, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 60 pages. ISBN: (print) 978-0-7714-3069-5; (online) 978-0-7714-3070-1.

Sarah Irwin, Roshan Srivastav and Slobodan P. Simonovic (2014). Instruction for Watershed Delineation in an ArcGIS Environment for Regionalization Studies.Water Resources Research Report no. 087, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 45 pages. ISBN: (print) 978-0-7714-3071-8; (online) 978-0-7714-3072-5.

Andre Schardong, Roshan K. Srivastav and Slobodan P. Simonovic (2014). Computerized Tool for the Development of Intensity-Duration-Frequency Curves under a Changing Climate: Users Manual v.1. Water Resources Research Report no. 088, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 68 pages. ISBN: (print) 978-0-7714-3085-5; (online) 978-0-7714-3086-2.

Roshan K. Srivastav, Andre Schardong and Slobodan P. Simonovic (2014). Computerized Tool for the Development of Intensity-Duration-Frequency Curves under a Changing Climate: Technical Manual v.1. Water Resources Research Report no. 089, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 62 pages. ISBN: (print) 978-0-7714-3087-9; (online) 978-0-7714-3088-6.

Roshan K. Srivastav and Slobodan P. Simonovic (2014). Simulation of Dynamic Resilience: A Railway Case Study. Water Resources Research Report no. 090, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 91 pages. ISBN: (print) 978-0-7714-3089-3; (online) 978-0-7714-3090-9.

Nick Agam and Slobodan P. Simonovic (2015). Development of Inundation Maps for the Vancouver Coastline Incorporating the Effects of Sea Level Rise and Extreme Events. Water Resources Research Report no. 091, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 107 pages. ISBN: (print) 978-0-7714-3092-3; (online) 978-0-7714-3094-7.

Sarah Irwin, Roshan K. Srivastav and Slobodan P. Simonovic (2015). Instructions for Operating the Proposed Regionalization Tool "Cluster-FCM" Using Fuzzy C-Means Clustering and L-Moment Statistics. Water Resources Research Report no. 092, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 54 pages. ISBN: (print) 978-0-7714-3101-2; (online) 978-0-7714-3102-9.

Bogdan Pavlovic and Slobodan P. Simonovic (2016). Automated Control Flaw Generation Procedure: Cheakamus Dam Case Study. Water Resources Research Report no. 093, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 78 pages. ISBN: (print) 978-0-7714-3113-5; (online) 978-0-7714-3114-2.

Sarah Irwin, Slobodan P. Simonovic and Niru Nirupama (2016). Introduction to ResilSIM: A Decision Support Tool for Estimating Disaster Resilience to Hydro-Meteorological Events. Water Resources Research Report no. 094, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 66 pages. ISBN: (print) 978-0-7714-3115-9; (online) 978-0-7714-3116-6.

Tommy Kokas, Slobodan P. Simonovic (2016). Flood Risk Management in Canadian Urban Environments: A Comprehensive Framework for Water Resources Modeling and Decision-Making. Water Resources Research Report no. 095. Facility for Intelligent Decision Support,

Department of Civil and Environmental Engineering, London, Ontario, Canada, 66 pages. ISBN: (print) 978-0-7714-3117-3; (online) 978-0-7714-3118-0.

Jingjing Kong and Slobodan P. Simonovic (2016). Interdependent Infrastructure Network Resilience Model with Joint Restoration Strategy. Water Resources Research Report no. 096, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 83 pages. ISBN: (print) 978-0-7714-3132-6; (online) 978-0-7714-3133-3.