THE UNIVERSITY OF WESTERN ONTARIO
DEPARTMENT OF CIVIL AND
ENVIRONMENTAL ENGINEERING

Water Resources Research Report

Coupling System Dynamics with Geographic Information Systems: CCaR Project Report

By:

Angela Peck
Christian Neuwirth
and
Slobodan P. Simonovic

Report No: 086
Date: March 2014

ISSN: (print) 1913-3200; (online) 1913-3219;
ISBN: (print) 978-0-7714-3069-5; (online) 978-0-7714-3070-1;

Coupling System Dynamics with Geographic Information Systems:
CCaR Project Report

Angela Peck

Christian Neuwirth

Slobodan P. Simonovic

Department of Civil and Environmental Engineering

The University of Western Ontario

London, Ontario, Canada

March 2014

EXECUTIVE SUMMARY

This report presents the progress in the development of a City Resilience Simulator (CRS) as part of the Coastal Cities at Risk (CCaR) project. The goal of the CRS is to integrate various impacts and adaptive capacity measures to better assess the dynamics of community resilience in the presence of climate change caused disasters. It combines temporal and spatial simulation to capture real-world dynamic behaviours. In using the CRS to identify systems which contribute most to disaster resilience, a community can take or create opportunities to emerge from a disaster with improved levels of resilience to combat and adapt to future disasters.

The focus of this report is to describe the coupling program (CP) tool used to link together ArcGIS and Vensim software for spatial and temporal resilience simulation. This report presents the technical challenges and solutions for the functionality, development and use of the CP. An application of the CP is presented using the heath care sector (specifically hospital services) as an example within the context of the CCaR project. This report provides the foundation for developing the CP, however further extension of the CP and CRS is required to incorporate all city system sectors (physical, health, social, economic, and organizational). Also, discussion pertaining to incorporation of adaptation policies and scenarios into the CRS is planned for future.

Table of Contents

List of Figures

List of Tables

List of Acronyms

CCaR	Coastal Cities at Risk
CP	Coupling Program
CRS	City Resilience Simulator
DLL	Dynamic Libraries Link
ESRI	Environmental Systems Research Institute
GIS	Geographic Information System
GPL	General Public License
OS	Operating System
SA	Service Area
SD	System Dynamics
SSD	Spatial System Dynamics

0. BACKGROUND

The research presented in this report is part of a larger, inter-disciplinary, multi-institutional, international development research centre (IDRC) funded project “Coastal Cities at Risk” or CCaR. It is a collaborative international research initiative to enhance adaptation to climate change in coastal cities around the globe. The four international project cities include Vancouver, Canada; Manila, Philippines; Lagos, Nigeria; and Bangkok, Thailand. Each of these project cities has their own research team. These four cities were selected for this project, in part, because of their coastal locations, presence of river delta, high density urban populations and diverse economic, physical, social, and political environments.

This report builds on the quantitative framework for the assessment of disaster resilience (Peck and Simonovic, 2013; and Simonovic and Peck, 2013) which moves beyond qualitative conceptualizations and provides a measure that can be used to evaluate various decisions that affect resilience of a city. The quantitative resilience measure introduced by Peck and Simonovic (2013) and Simonovic and Peck (2013) allows for better identification and understanding of those factors which contribute to resilience. In understanding what systems are contributing most to disaster resilience, it will hopefully be possible to take or create opportunities to emerge from a disaster stronger than before, with improved levels of resilience to combat and adapt to future disasters. This research considers a Space-time Dynamic Resilience Measure (ST-DRM), which is designed to capture the complex relationships between the main drivers of resilience. This report introduces the method for linking time and space in the implementation of ST-DRM. Detailed presentation of generic coupling programming concepts designed for use as part of the City Resilience Simulator (CRS) is presented.

The remainder of the report is laid out as follows: Chapter 2 provides an introduction to spatial system dynamics (SSD) approach and current research and methodologies in literature. This chapter also describes the theory behind coupling temporal simulation using Systems Dynamics (SD) and spatial analysis Geographic Information System (GIS) and gives the basic principles and generic approach for coupling different programs using Python programming language. Chapter 3 presents the research methodology. Chapter 4 describes in detail the functionality, development and technical requirements (hardware and software) of the coupling program. This chapter is particularly important for people who will be modifying or using the coupling program. Chapter 5 provides a sample application of the coupling program within the context of CCaR project. Finally, Chapter 6 presents a few key messages and conclusions of the work.

1. INTRODUCTION

There have been recent shifts in the approach to effective disaster management including: (i) a shift from focusing on disaster risk reduction strategies to focusing on building disaster resilience through effective adaptation actions; (ii) from considering static characteristics of vulnerable types of populations to considering the dynamic attributes of resilient communities; and (iii) beginning to recognize the importance of both the spatial and temporal dynamics involved in all phases of disaster planning, response and recovery. The focus on disaster resilience provides a proactive approach to disaster management. The concept of resilience is also unique in that it focuses on the *opportunities* arising from a disaster and creates a positive shift in attitudes related to disaster management.

The theory behind developing a City Resilience Simulator (CRS) tool is built on the fundamental concept that a resilient city is a sustainable network of physical (constructed and natural) systems and human communities (social and institutional) that possess the capacity to survive, cope, recover, learn and transform from a disturbance by: (i) reducing failure probabilities; (ii) reducing consequences; (iii) reducing time to recovery; and (iv) creating opportunity for development and innovation from adverse impacts. The CRS will be used in the assessment process to analyze simulated changes in city resilience over time and space before, during and after the occurrence of a disaster event. Spatial variances in resilience can result in recovery disparities. Therefore, CRS provides for the assessment of system resilience considering both, spatial and temporal dynamics. The ST-DRM integrates various units that characterize the impacts of disasters on an urban community (Peck and Simonovic, 2013; Simonovic and Peck, 2013).

System Dynamics is a logical problem-solving technique which combines traditional management of social systems and feedback theory with computer simulation for the purpose of gaining a better understanding of real-world systems behaviour. The approach is useful in managing complex systems by considering its components, elements, feedbacks and subsystems. It supports the use systems thinking to understand the behaviour of complex city systems. System Dynamics (SD) is an appropriate approach for capturing the temporal dynamics of disaster resilience, but is not originally intended for spatial modeling. Therefore, to overcome this shortcoming, our research proposes coupling SD and GIS together for joint temporal and spatial modeling capabilities.

The focus of this report is a set of technical challenges and solutions of linking together spatial and temporal simulation. The proposed approach achieves this through an independent program which is referred to herein as the "coupling program" (CP). A detailed description of the CRS concept and theory is presented in Peck and Simonovic (2013), thus the following explanation of the CRS will serve as a summary. The CRS encompasses all city resilience modeling and simulation efforts. It includes functionality of three software programs: Vensim for SD temporal simulation (Ventana Systems, 2013); ArcGIS for spatial GIS analyses (ESRI, 2014); and CP designed to provide the bridge between the first two. In this way the CRS is able to simulate the entire set of city resilience components under consideration (physical, economic, social, health and organizational). The CP is merely one of possible technical solutions to link together Vensim and ArcGIS software for spatial-temporal modeling and simulation. Thus, the CP may be considered a tool, much like Vensim and ArcGIS, as part of the overall CRS.

1.1 Coupling SD and GIS

The CRS is dynamically represented in Vensim computer software that is appropriate for modeling and simulating dynamic system behaviour over time. To construct the CRS in Vensim, stocks and flows are used with programs' graphical interface to map the system structure. Then links are created between related model elements which represent real-world relationships and feedbacks. In this way, the model is expanded to include subsystems and interactions most important to the disaster resilience capacity and functioning of a city. To address the spatial modeling shortcomings of Vensim, our research proposes the use of ArcGIS. The CRS allows Vensim to retrieve spatial information from ArcGIS to use in system temporal simulation and then send it back to ArcGIS for spatial analysis.

The needs for coupling SD and GIS arise from:

i) The need to reuse of already available structures in order to derive solution which can be used for more than one specific application. Writing a spatial module for Vensim doesn't solve the problem, since new spatial analysis tools have to be programmed for each new application. This is inefficient and also not possible when it comes to modelling processes which require a comprehensive set of spatial modelling algorithms.

ii) The complexity of GIS software which is too comprehensive to emulate by means of home-made solutions. Apart from spatial analysis tools, GIS also provides sophisticated models for the visualization and management of spatial-data.

iii) The lack of standardized tools for post- and pre-processing of data.

iv) The fact that GIS already has spatial analyst tools and functions available for use, so there's no need to re-program all these pre-packaged features.

v) The fact that coupling of SD with GIS produces a unique computer-based framework for spatiotemporal modelling.

Python is a free, object-oriented programming language which offers many features (Python.org, 2013). The Python interpreter can be embedded into other applications or used as an extension. Extending Python is similar to embedding it, but not quite. The difference is that when you extend Python, the main program of the application is still the Python interpreter, while if you embed Python the main program may have nothing to do with Python - instead, some parts of the application occasionally call the Python interpreter to run some Python code (Python.org, 2013). The proposed approach in this report considers a Python extension, referred to as the CP.

The report by Srivastav and Simonovic (2014) provides another possible technical solution to coupling SD and GIS using MATLAB (MathWorks, 2014) and C# programming language. The report draws directly from the theory and concepts introduced in Peck and Simonovic (2013) and similarly uses the health system as a hypothetical example for demonstrating the mapping of dynamic resilience using the C#/MATLAB coupling tool. The program is able to draw on Vensim's existing online help guide and support tools for C# since they have been specifically tailored to writing external programs in various programming languages. However, the drawbacks of the approach by Srivastav and Simonovc (2014) include: (i) additional proprietary software (MATLAB); (ii) limited ArcGIS support, and (iii) C# geoprocessing tool may require Python wrapping. The intended end-use for the program by Srivastav and Simonovic (2014) differs from the CCaR application, but the report provides a valuable alternative technical solution to modeling the spatial-temporal dynamics.

1.2 Methods

Programming languages originally considered to write the coupling program include: C++, Java, and Python. It was decided that writing the coupling program using a single programming language would be most appropriate for the present application. The decision is made on the basis of:

i) **Simplicity;** no transitions are required between multiple programming languages which can decrease compilation and execution times.

ii) **Reusability;** parts of code can be reused in other parts of the program which avoids unnecessarily having to rewrite code from scratch.

iii) **Consistency;** the coupling program written in a single language simplifies problems associated with debugging the code.

iv) **Transferability;** it requires knowledge of only a single programming language. This helps when the program is redistributed to other program users or software developers. They require an expert in only a single programming language. It will also help other project teams better understand the code, trust the code, and modify the code, as they see fit for their own purposes.

Once the decision is made to write the coupling program in a single programming language, it was decided that the entire stand-alone coupling program should be written in Python programming language. Python is a free, open-source programming language supported by the GNU General Public License (GPL) (Python.org, 2014). It supports object-oriented programming and is selected as the language to write the coupling program since it provides for:

i) **Compatibility;** Python is both Vensim and ArcGIS compatible. One of the leading companies in promoting and supplying GIS software, Environmental Systems Research Institute (ESRI), is promoting Python as the primary language to write GIS scripts. Many of the available tools and functions have already been written in Python, so they are already available for use. In addition, Python scripts are able to call Vensim DLLs.

ii) **Efficiency;** Python was designed as a language which emphasizes simplicity and readability in the code. With a simpler syntax, programmers can perform functions in fewer lines of code than required with other programming languages.

iii) **User friendliness;** the Python language itself is generally considered more user-friendly and intuitive than other programming languages – especially for people new to programming. Python has a detailed standard library which provides tools and examples of programming to accomplish many tasks. There are also many help forums and a solid user-support community to provide assistance when complications arise.

iv) **Planning for the future;** Python is a relatively new programming language, however it looks like the future of programming is headed in a Python direction. This may make it easier for future researchers to continue building on the coupling program or make modifications to it in the future.

v) **Cross platform/OS functionality;** Python can be used on various platforms and is supported by various operating systems.

vi) **Open source language;** other programming languages such as C# and C++ are Microsoft proprietary (although admittedly there has been some progress in open source developments of these languages as well), whereas Python is open source.

Limitations when using Python may include:

i) **Familiarity;** as Python is a relatively new programming language, the learning curve for Python syntax may be a little steeper, but probably a worthwhile investment.

ii) **Backwards incompatibility;** In 2008 Python released a new version of the programming language, Python 3.0; otherwise known as Py3k (Python.org, October 2013). This new release version is incompatible with Python version 2.x releases (Python.org, October 2013).

iii) **Limitations to accessing the internal structure of ArcGIS;** Python is able to access many functions and tools from ArcGIS but doesn't provide complete control over manipulating the data (which is likely not required in the CCaR application anyway). There is potential of replacing ArcGIS with an open source GIS tool in the future.

Although Python version 3.x (where "x" denotes a sub-version within the line of the major release) is described by the Python community as "the present and future of the programming language", the coupling program is written in Python 2.x. Version 2.x offers better library support, documentation and most importantly, at this time Python version 3.x is not supported by ArcGIS packages and libraries. This may sound like trivial information, but it is in fact essential to the proper functioning of the current version of the coupling program. The coupling program requires Python version 2.x only. Presence of Python version 3.x on the same computer may cause unforeseen problems.

2 COUPLING PROGRAM DESCRIPTION

First, in this report the CP is referring to the Python program that is dynamically coupling Vensim and ArcGIS; whereas the CRS encompasses all three tools. The coupling program (CP) is designed to use the functions and tools available in Vensim system dynamics simulation software and ArcGIS geospatial data and analysis system. More specifically, the coupling program uses DLL files (*.dll*), library files (*.lib*), packaged Vensim model files (*.vpm*), and raster-based GIS database files (*.mdb*) to create an environment for dynamically linking Vensim model simulation and GIS spatial analysis tools. In brief, the CP operates in the following way:

Upon successfully loading a SD simulation model, it is desirable to have control over SD model parameters and variables during the course of the entire simulation. This enables piece-wise retrieval of information bit-by-bit as the simulation progresses. Without this feature, Vensim would perform the entire simulation before returning control to the Python program. This means that there would be no opportunity for GIS to interact with the SD model. This control over the simulation is herein referred to as the "game" feature of Vensim. This "game" feature is used by the CP to interact with the SD model in real-time during the course of simulation. This same feature also allows the CP to retrieve the value of specific variables during each time step of the simulation. The CP is able to modify these variable values based on spatial analysis performed in GIS. It is in this way that the CP forms a dynamic link between the two software tools. For a simple visualization of this process, please consider Figure 1.

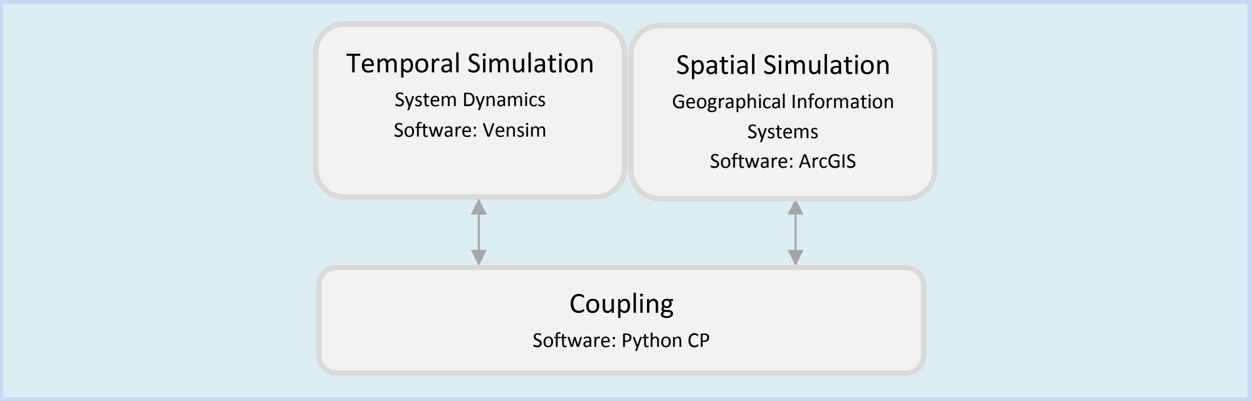


Figure 1. Simple visualization of the coupling technique

Essentially, the coupling program enables communication between a Vensim model simulation and ArcGIS software for spatial data extraction and geospatial analysis tools. The Python code for the entire CP program can be found in Appendix A. The following section describes the technical details of the CP and is intended more for people who will be developing the CP or for people who may be adjusting or modifying the CP for different applications. Potential users of CP may refer to Chapter 3 directly.

2.1 Vensim DLLs

In order to achieve the functionality of Vensim system dynamics simulation outside of the Vensim program, requires the use of files called Dynamic Link Libraries (DLLs). DLLs allow control of Vensim from another program; in this case the “coupling program” (CP) developed in Python. There are two main methods of linking libraries: static linking and dynamic linking. These methods of linking are further described below:

Static linking: when the library can be included within the application itself. Static linking makes deployment easier at the cost of flexibility as different application will each load the same copy of the DLL.

Dynamic linking: shared library of functions collected together so that multiple programs can use them. Thus, programmers are required to write the code for a particular task only once, and it can be used in many other programs to do similar tasks. It functions similar to an executable program, except the library of functions are “exported” and called by the user in other programs. It is “dynamic” because the linking happens when a program is run instead of at link time. Dynamic linking can further be subdivided into load-time dynamic linking and run-time dynamic linking. Advantages of this approach include: saving disk space, easy to update and debug; which involves creating a new DLL which will be dynamically loaded into the program, without having to recompile.

The current version of the CP that has been developed for the CCaR project uses dynamic linking to access the functions available in Vensim and ArcGIS software. In order to achieve full Vensim DLL functionality, the DSS version of Vensim must be installed (Ventana Systems, 2013). To see what other features are available in Vesim DSS, refer to Table B 1 in Appendix B for a comparison of PLE, PLE Plus, Professional, DSS and Model Reader versions of Vensim. Further descriptions of each of the Vensim configurations can be found on the Ventana Systems Vensim website at: http://vensim.com/.

The Vensim DLL has many callable functions available through commands for simulation control, data access, structure access and other miscellaneous functions to control simulation settings. For a more comprehensive list of callable functions, please refer to the Vensim DSS Reference Supplement (Ventana Systems, 2013). These functions are used in the custom-built CP to provide control over Vensim simulation using another programming language.

Important note: the Vensim DLL used by the current version of the CP is 32-bit, so the CP needs to be compiled for x86. When installing Vensim software, leave the DLL where it is automatically installed on your computer or you will face problems compiling the CP. Also, do not rename any *.dll* or *.lib* files or this will create additional complications.

In the CP code (see Appendix A), the Vensim DLL is searched for on the host computer in following line of the code:

```
>lib_path = ctypes.util.find_library(“vendll32.dll”)
```

This is the default name for the 32-bit, single-precision Vensim DSS, *.dll* file. If the file is not found on the system, the CP will return an error message as follows:

```
“Unable to find dll: vendll32.dll”
```

If this message appears, it is likely because the Vensim DLL installation file is either not in the correct location on the computer (in the default installation folder) or the file has been tampered with. For troubleshooting this problem, please refer to Table C 1 in Appendix C.

2.2 Coupling program requirements

The only software required to develop, compile and run the coupling program comes with the installation of ArcGIS; programs listed in Table 1 provide some alternatives which are not required, but are highly recommended for future application development or modifications of the CP. On the other hand, certain software is required in order to effectively compile and run the current version of the CP (see Table 2).

Table 1. Recommended programs for use in further development or modification of the coupling program.

	Recommended Software	Available at	Notes
IDE	IDLE[a]	comes with the installation of Python	Open-source, free, and cross-platform. In many cases, this software comes with the installation of ArcGIS/Python.
Text Editor	Notepad ++	notepad-plus-plus.org	User-friendly, free (under GNU/GPL license), multiple languages supported.

[a] Developed using IDLE version 2.7.2

Table 2. Software required to successfully compile and run the coupling program.

	Required Software	Version	Available at	Notes
Programming Language	Python	2.x[a]	www.python.org	Coupling program not compatible with Python versions 3.x.
Python Libraries	Numpy Arcpy Ctypes Os Sys		www.scipy.org	Numpy and Arcpy should come with ArcGIS installation, however in some instances it may be necessary to separately install these libraries.
GIS Software	ArcGIS	10.x	Proprietary	Install complete Arc suite (ArcInfo, ArcMap, ArcCatalog) if possible.
System Dynamics Software	Vensim	DSS[b]	Proprietary	Install the single-precision version.
Operating System	Windows	XP or later[c]	Proprietary	The coupling program has not been tested on Mac computers, but the assumption is that many modifications

				to the program would be required to successfully compile and run from Mac.

[a] Developed using Python 2.7.2

[b] System dynamics model developed in version 5.10e

[c] Compiled and executed in Windows 7

Abiding by the software recommendations and requirements, should minimize compatibility issues, minimize complications and reduce debugging. Now that the technical details of the coupling program have been addressed, the next chapter will describe how to execute the coupling program and use it as a tool in resilience calculation.

3 USING THE COUPLING PROGRAM

There are some necessary pre-processing steps required for both Vensim and GIS files before they can be used with the current version of the CP. This chapter provides a description of how to prepare a Vensim model and ArcGIS data for simulation using the CP. This description is assuming that the model and data are developed from scratch; for preparing a generic model for the CP program. Thus, major modifications of the CP for health resilience example would actually be required in order to use the CP for another Vensim model. In order to pre-process and running the CP specific to the hospitals example, please refer to Chapter 4.

3.3 Pre-processing

In order to run the CP, you must first create a Vensim model (*.mdl*) file. Build the Vensim model using stocks, flows, variables and constants and fill in equations for each object (Figure 2). Define the Vensim model settings for simulation using the CP (Figure 3).

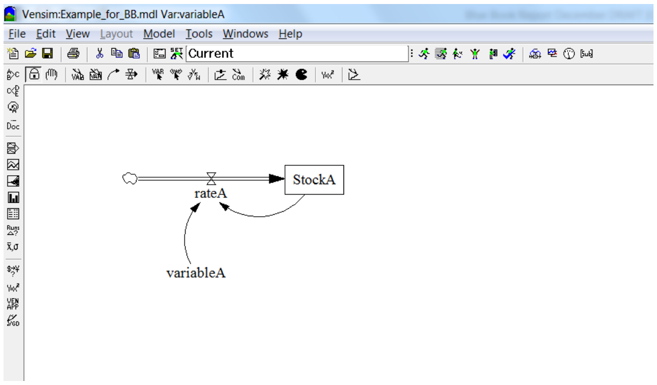
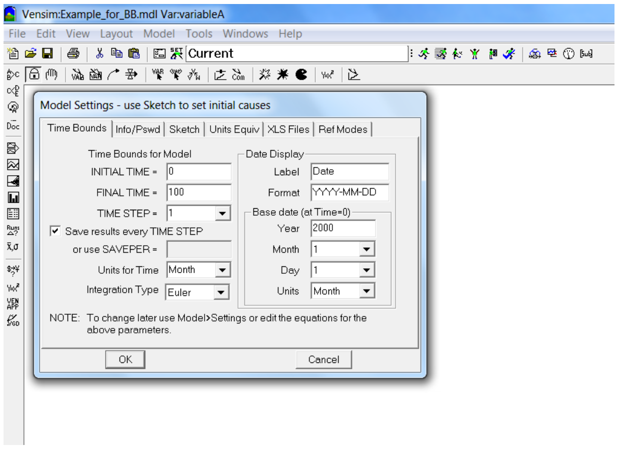

Vensim:Example_for_BB.mdl Var:variableA
File Edit View Layout Model Tools Windows Help
Current
StockA
rateA
variableA

Figure 2. Prepare a model in Vensim



Vensim:Example_for_BB.mdl Var:variableA

Model Settings - use Sketch to set initial causes

File Edit View Layout Model Tools Windows Help
Current
Time Bounds | Info/Pswd | Sketch | Units Equiv | XLS Files | Ref Modes
OK
Cancel

Time Bounds for Model

INITIAL TIME = 0

FINAL TIME = 100

TIME STEP = 1

☑ Save results every TIME STEP

or use SAVEPER =

Units for Time: Month

Integration Type: Euler

Date Display

Label: Date

Format: YYYY-MM-DD

Base date (at Time=0)

Year: 2000

Month: 1

Day: 1

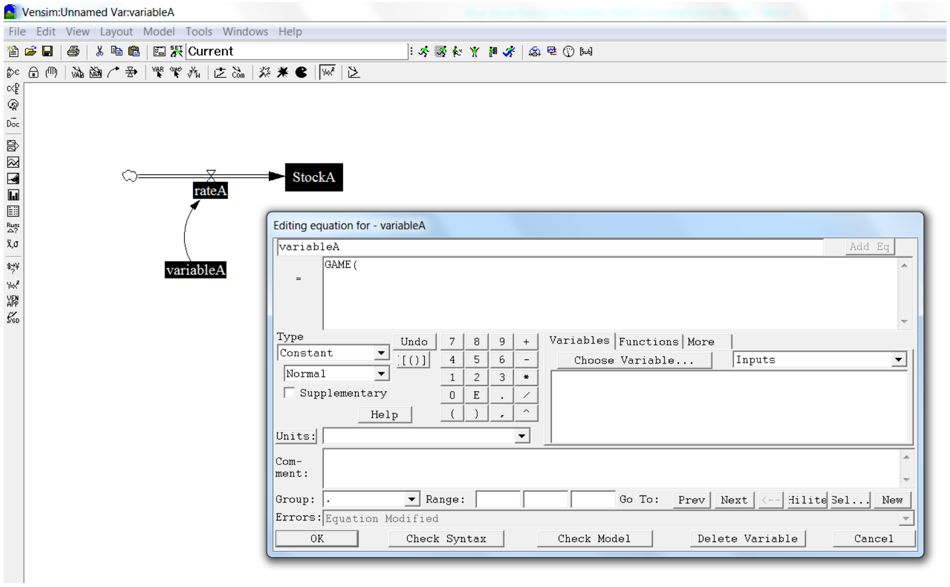
Units: Month

NOTE: To change later use Model>Settings or edit the equations for the above parameters.

Figure 3. Define model settings

Once the Vensim model is prepared, there are a couple additional steps required in order for the model to run with CP:

1) For instance, each of the variables that should be *controlled* and *modified* during a simulation, must be first converted into the "Game" variables. In the Vensim model, open the equation editor for those variables that should be Game variables.
2) In the equation box, next to the "=" sign, type the world "GAME", followed by any regular equation to be used to define that variable (Figure 4). This will convert the model variable into a Game variable and prepare it for use in the CP.



Vensim:Unnamed Var:variableA

Editing equation for - variableA

variableA

= GAME (

Type: Constant

Normal

□ Supplementary

Units:

Errors: Equation Modified

File Edit View Layout Model Tools Windows Help
Current
StockA
rateA
variableA
Add Eq
Undo
7 8 9 +
4 5 6 -
1 2 3 *
0 E . /
( ) , ^
Help
Variables | Functions | More
Choose Variable...
Inputs
Com-ment:
Group: Range: Go To: Prev Next Hilite Sel... New
OK Check Syntax Check Model Delete Variable Cancel

Figure 4. Designating Vensim model variables, "Game" variables

3) The coupling program requires published Vensim model (*.vpm* files) as input. Therefore, once the final version of Vensim model is prepared (and all the controllable variables are converted into Game variables), the "Publish" option should be selected from the "File" menu (Figure 5).

4) After selecting the "Publish" option, a menu will appear (Figure 6) which provides additional options for publishing a Vensim model. In this menu a Name for the published model should be specified (by default, Vensim will assign to the published model the same name as the regular Vensim model file (*.mdl*) name). If different name is required the modification can be done in the box next to "Published model name" with the extension *.vpm*.

5) Once the model has been saved as a published model (*.vpm*) it is ready for use with the CP. Note that password protecting the model could cause problems. Most of the other options in this menu are self-explanatory and do not require modification for the CP to run.

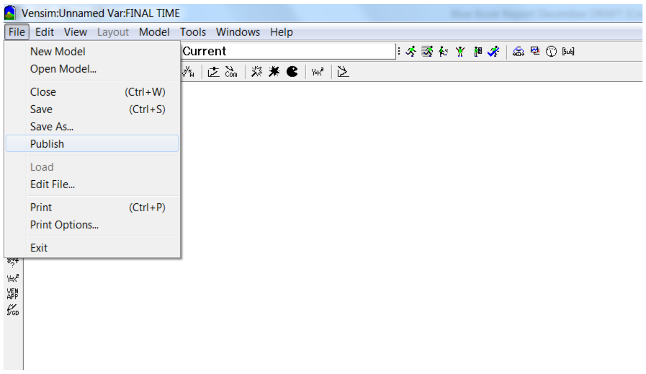
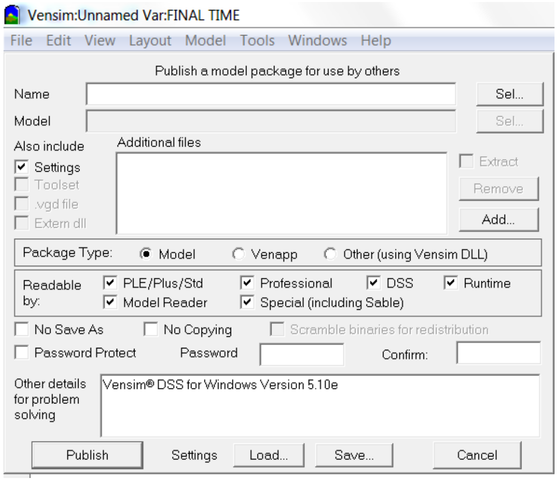


Figure 5. Select the option to publish a Vensim model


Vensim:Unnamed Var:FINAL TIME
File Edit View Layout Model Tools Windows Help

Publish a model package for use by others

Name: ______ Sel...

Model: ______ Sel...

Also include: ☑ Settings ☐ Toolset ☐ .vgd file ☐ Extern dll

Additional files: ______ ☐ Extract Remove Add...

Package Type: ● Model ○ Venapp ○ Other (using Vensim DLL)

Readable by: ☑ PLE/Plus/Std ☑ Professional ☑ DSS ☑ Runtime ☑ Model Reader ☑ Special (including Sable)

☐ No Save As ☐ No Copying ☐ Scramble binaries for redistribution

☐ Password Protect Password ______ Confirm: ______

Other details for problem solving: Vensim® DSS for Windows Version 5.10e

Publish Settings Load... Save... Cancel

Figure 6. The Vensim model "Publish" menu (or similar)

To ensure proper functioning of the CP, the GIS files must be raster-based and have consistent coordinate systems and spatial extents. In order to prepare GIS files for use:

1) Check that all GIS database files are contained in the “GIS_Package”; do not move these files.
2) Ensure that the GIS files prepared for use are raster-based. If one or more map files is in another format (for example, vector-based) then conversion to raster-based map is required before using the data as input into the CP.
3) Ensure that the extent and spatial reference systems for all GIS files are the same. Any extent or spatial reference may be used, as long as they are consistent between all GIS files. To check, select layer properties (Figure 7).

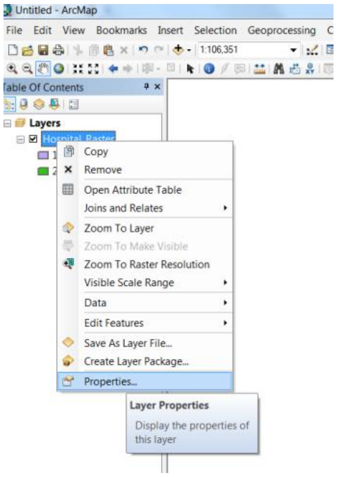
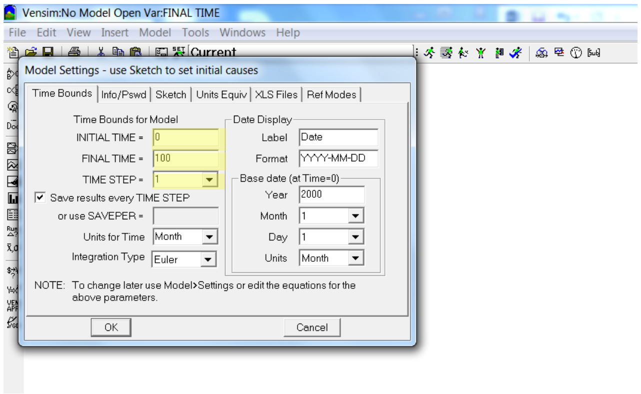


Figure 7. Select layer properties

In the current version of the CP, there are a few modifications which need to be made directly within the CP code to ensure that the program executes properly:

1) First, the number of simulation time steps should be set as desired. This can be done by modifying the value for the variable TIME_STEPS (Figure 8).
2) Check that the variables in the Vensim model and the coupling program match. (i.e. variable names are exactly the same).
3) Check that all the directories in the CP are pointing to the correct location of the files on the host computer (Figure 9). Note: this is one of the first things that should be checked if the program crashes and returns errors.
4) Specify the output directory directly within the CP code. This directory will be where simulation results are saved.

5) Before running the CP, ensure that all other programs are closed, otherwise computational efficiency will decrease and potential errors may arise.

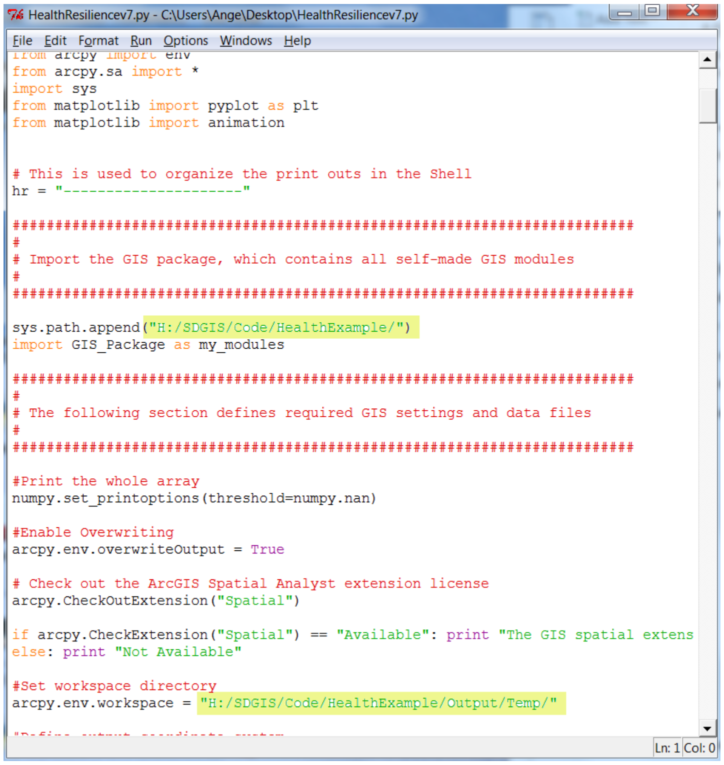

Vensim:No Model Open Var:FINAL TIME
File Edit View Insert Model Tools Windows Help
Current
Model Settings - use Sketch to set initial causes
Time Bounds
Info/Pswd
Sketch
Units Equiv
XLS Files
Ref Modes
OK
Cancel

Time Bounds for Model
INITIAL TIME = 0
FINAL TIME = 100
TIME STEP = 1
☑ Save results every TIME STEP
or use SAVEPER =
Units for Time: Month
Integration Type: Euler

Date Display
Label: Date
Format: YYYY-MM-DD
Base date (at Time=0)
Year: 2000
Month: 1
Day: 1
Units: Month

NOTE: To change later use Model>Settings or edit the equations for the above parameters.

Figure 8. Model settings; define time parameters


HealthResiliencev7.py - C:\Users\Ange\Desktop\HealthResiliencev7.py
File Edit Format Run Options Windows Help
Ln: 1 Col: 0

```
from arcpy import env
from arcpy.sa import *
import sys
from matplotlib import pyplot as plt
from matplotlib import animation


# This is used to organize the print outs in the Shell
hr = "----------------------"

##########################################################################
#
# Import the GIS package, which contains all self-made GIS modules
#
##########################################################################

sys.path.append("H:/SDGIS/Code/HealthExample/")
import GIS_Package as my_modules

##########################################################################
#
# The following section defines required GIS settings and data files
#
##########################################################################

#Print the whole array
numpy.set_printoptions(threshold=numpy.nan)

#Enable Overwriting
arcpy.env.overwriteOutput = True

# Check out the ArcGIS Spatial Analyst extension license
arcpy.CheckOutExtension("Spatial")

if arcpy.CheckExtension("Spatial") == "Available": print "The GIS spatial extens
else: print "Not Available"

#Set workspace directory
arcpy.env.workspace = "H:/SDGIS/Code/HealthExample/Output/Temp/"

#Define output coordinate system
```

Figure 9. Modify the locations to point directory of the files on host computer

Once the Vensim and GIS data have been pre-processed and modifications made to the CP, the package is ready to run a simulation. The following section of the report describes how to run the CP simulation and summarize how the CP operates.

3.4 Running the Program/Technical CP description

Once the pre-processing of the Vensim and GIS files has been completed, it is possible to execute the CP to run a full simulation. Most of the CP has been fairly descriptively commented within the Python script itself (see Appendix A). However, here is a step-by-step description of the main processes executed by the CP.

These are the main functions that the CP performs during the execution:

1) **Open:** The Python executable file (*.py*) in the Python Shell.
2) **Import & define:** The first section of the code is designed to import Python libraries required to run the CP and import necessary ArcGIS Python packages. In this first section of the code, parameters used in formatting outputs to the Python shell are specified and the raster files for use with the CP program are defined.
3) **Load DLL:** This section directs the Python program to the Vensim DLL file and then loads the DLL. The function *find_library* searches along the system search path and returns the full pathname; but since there is no predefined naming scheme, a call like find_library ("c") will fail and return "None". It is important to note that Vensim Single Precision installation will download a different DLL file than the Double Precision version of the software. Therefore, selection of the DLL file that will be used should be based on the current installation of Vensim and the other DLL file should be commented with the code (Vensim single-precision version is recommended)
4) **Load Vensim model:** This section loads a Vensim model. In order for this to work, the model must be a published .vpm file format (see previous section of the report or consult the Vensim Help manual).
5) **Define Vensim GAME simulation parameters:** This section of the CP defines the Vensim GAME controls; start, continue, end. It is in this section of the code that the game interval (i.e. time step) and game time (desired length of the simulation; i.e. final time) should be defined.
6) **Retrieve GAME variables:** This section of the program defines the Vensim get_value function which is required in order to retrieve GAME variable values during a simulation. It is important that the name of the GAME variables in this section of the code match the names of the GAME variables in the published Vensim model. This section also defines the

functions for writing changes that occur during a simulation (similar to saving a log of changes).

7) **Initiate GAME simulation:** begin the Vensim GAME simulation. This initializes all of the model variables to their starting values.
8) **Calculate:** This section is the GIS portion of the code. Data is represented spatially in GIS raster files and the values for each cell require extraction. These extracted values can then be returned to Vensim to complete the next step in simulation. This section of the CP completes the following steps: (i) Generates a random GIS raster file of population (population will vary for each cell between 0 and 5 people); (ii) Calculates the distance from the Hospital to each raster cell in the simulation domain; (iii) Calculates the "travel cost" (at time = 0) of each raster cell for each hospital's service area based on the distance; (iv) Calculates the service population for each hospital by summing up the values of the population in each cell, which are serviced by each hospital.
9) **Return variables:** Returns the calculated values for Vensim variables during the simulation.
10) **Resume GAME simulation:** Continues the simulation to the next time step (as defined by time_interval) in the simulation. Now all the calculations in the Vensim model are executed.
11) **Repeat:** Continues recalculating all the variables, reassigning values, and continuing the simulation to the next time step.
12) **GAME end:** Once the simulation has gone through all of the time step iterations (as defined by vensim_game_time), it is complete and all results have been output to the Python Shell, the Vensim model run, and the specified Output folder.

In order to actually run the simulation, you must open the CP in an IDE (IDLE program is used in this case – see Table 1) and then select the option "Run" and then "Python Shell". This will open up the Python Shell. The Python Shell is where any *print* functions in the code are displayed and indicates the status and the progress of the program execution. This is also the dialog box where errors will be displayed (if any occur).

3.5 Post-processing

Some post-processing is required for best representation of the CP results. Post-processing tasks may include:

Aesthetics; currently the ArcGIS output maps which are generated by the CP do not follow a particular colour scheme and for consistency they should be opened in ArcGIS and modified for better representation (e.g. change water to blue colour). These modifications will also help improve the visualization, understanding and interpretation of CP results. For example, raster cells which represent the flood extent should be blue to reflect the association between blue colour and water; roads may be visualized as black or grey; and raster maps representing the same variable (for example, low level of resilience) over time should remain the same colour.

Animation; linking together all of the raster output files. The ArcGIS raster map files can be simultaneously animated with the temporal output (graphs) from Vensim simulation. The graph features are included as part of the CP, however, to increase computational efficiency, the raster map animation feature has since been removed but may be re-included in the future as an optional feature of the CP. Alternatively, a third-party program may be used to link the GIS raster maps together in an animation.

Now that the pre-processing, post-processing and generic methods for implementing the CP have been presented, the next chapter focuses on a CP implementation case study as part of the CCaR project. This case study builds on the hospital example originally presented in Peck and Simonovic (2013).

4 APPLICATION

The coupling program could, in theory, be used for any number of applications in which time and space have a one or two-way interaction. Current applications include simulating spatial-temporal grassland farm structural changes in Austria; simulating spatial-temporal behaviour of black vs. white daisy growth in a fictitious environment called Daisyworld based on a non-spatial SD model originally proposed by Lovelock and Watson (1983) and reintroduced as a SSD simulation in Neuwirth et al. (*under review*); and in addition, of course, the calculation of resilience to natural disasters as intended for the CCaR project. The following section describes a potential application for use with the resilience calculation in the CCaR project.

4.1 CCaR application: hospital system example

In order to effectively demonstrate the CP, an example has been created for calculation health resilience based on the population access to hospital services. The example is based on the theoretical model originally presented in Peck and Simonovic (2013). In this example two hospitals are represented within the same Vensim model (Figure 10). For ease of CP demonstration, the SD structure behind these two hospitals are the same and equations are comparable. The hypothetical situation is presented below:

*There are two hospitals (herein referred to as HA and HB) which service a city (*Figure 11*). This city area covers 124 x 148 raster cells (18,352 cells*^{2}*). The population within each raster cell is randomly distributed throughout the city (*Figure 12*). The population density within each raster cell varies between [0, 6) people. Each of the hospitals, HA and HB, provides health services to a proportion of the city population: Service Area (SA) of Hospital A (SA–HA) and Service Area of Hospital B (SA–HB). The SA for each hospital is randomly determined before running the CP and subsequent model simulation. Both HA and HB use a road network to access each location (raster cell) within the city (*Figure 13*). A fictitious "flood" is introduced as a "shock" to the health system (*Figure 14*). This disturbance affects the performance of health system in the city. As the road network becomes inundated, the SAs for each hospital are adjusted to reflect the shortest travel distance to service a raster cell. Thus, a raster cell initially serviced by HA may at some point during the simulation become serviced by HB and the population serviced by each hospital at any given time will change depending on the availability of the road network. These SAs are used to determine a "service population" for each hospital. This value, in turn, is passed to the Vensim simulation model and used in system simulation and final calculation of resilience. In this particular case, the GIS is not passing the value of the SAs because the extent of area that each hospital services is not considered to be directly significant for hospital resilience, however the population is. So instead, the value of "population" associated with each of the SAs is passed to the SD model. This population value is then used in one time step of the simulation. This process continues for each time step until the desired number of simulation time steps is completed. The result is a series of semi-continuous maps (the number of maps generated will be determined by the number of simulation time steps). These maps show changes in areas and population served by each hospital. Maps are also generated which show the spatial values of health resilience. Sets of temporal graphs are also generated to show changes in serviceable population, injured patients, and resilience over time.*

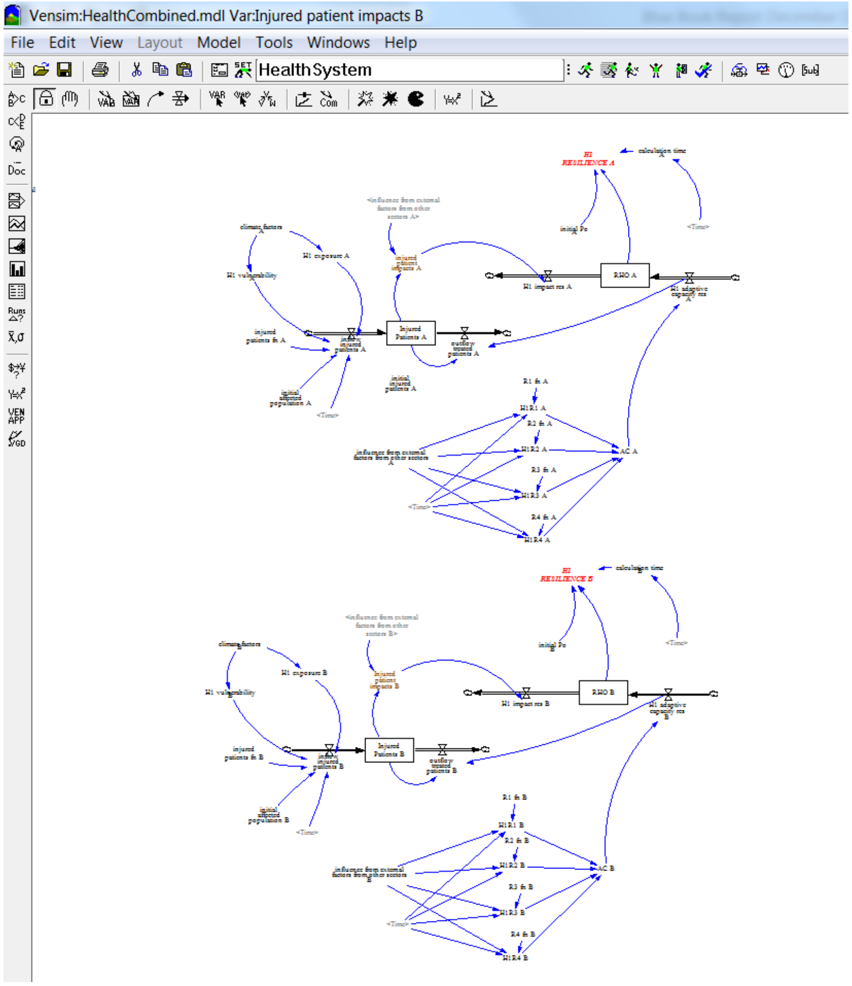

Vensim:HealthCombined.mdl Var:Injured patient impacts B
File
Edit
View
Layout
Model
Tools
Windows
Help
HealthSystem
Doc
Runs
H1 RESILIENCE A
calculation time
initial Po
<Time>
<influence from external factors from other sectors A>
climate factors
H1 exposure A
H1 vulnerability
injured patient impacts A
H1 impact res A
RHO A
H1 adaptive capacity res A
injured patients fn A
inflow injured patients A
Injured Patients A
outflow treated patients A
initial injured patients A
initial affected population A
<Time>
R1 fn A
H1R1 A
R2 fn A
H1R2 A
influence from external factors from other sectors A
AC A
R3 fn A
H1R3 A
<Time>
R4 fn A
H1R4 A
H1 RESILIENCE B
calculation time
<influence from external factors from other sectors B>
initial Po
<Time>
climate factors
H1 exposure B
H1 vulnerability
injured patient impacts B
H1 impact res B
RHO B
H1 adaptive capacity res B
injured patients fn B
inflow injured patients B
Injured Patients B
outflow treated patients B
initial affected population B
<Time>
R1 fn B
H1R1 B
R2 fn B
H1R2 B
influence from external factors from other sectors B
AC B
R3 fn B
H1R3 B
<Time>
R4 fn B
H1R4 B

Figure 10. Combined hospital Vensim SD model

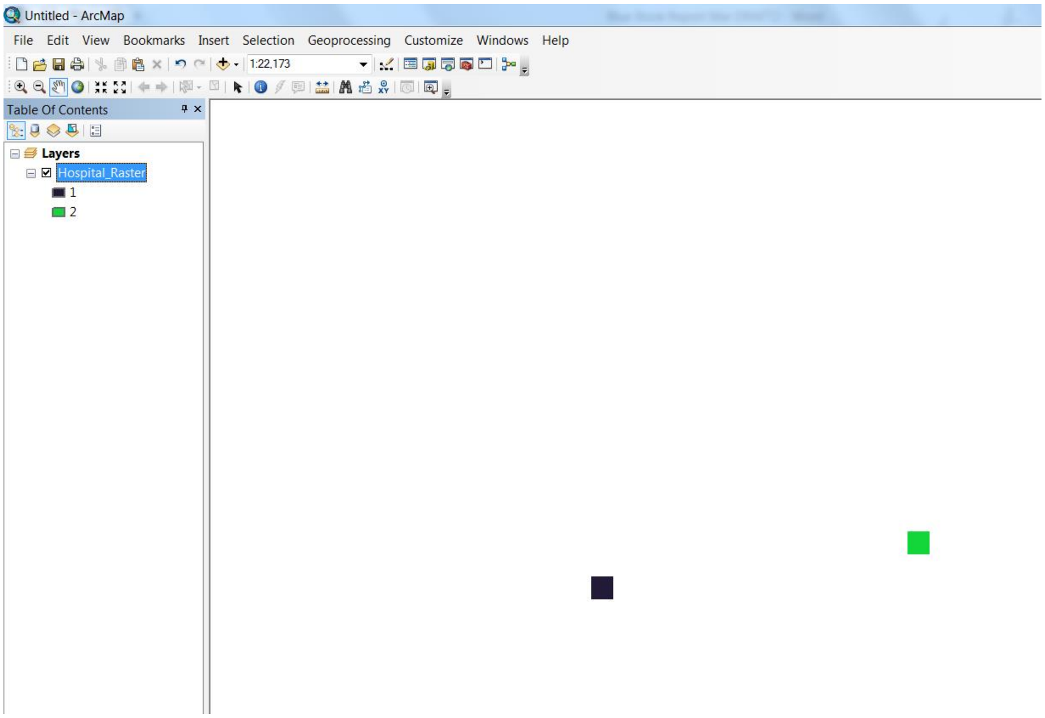

Untitled - ArcMap
File Edit View Bookmarks Insert Selection Geoprocessing Customize Windows Help
1:22,173
Table Of Contents
Layers
Hospital_Raster
■ 1
🟩 2

Figure 11. The two hospitals, A and B

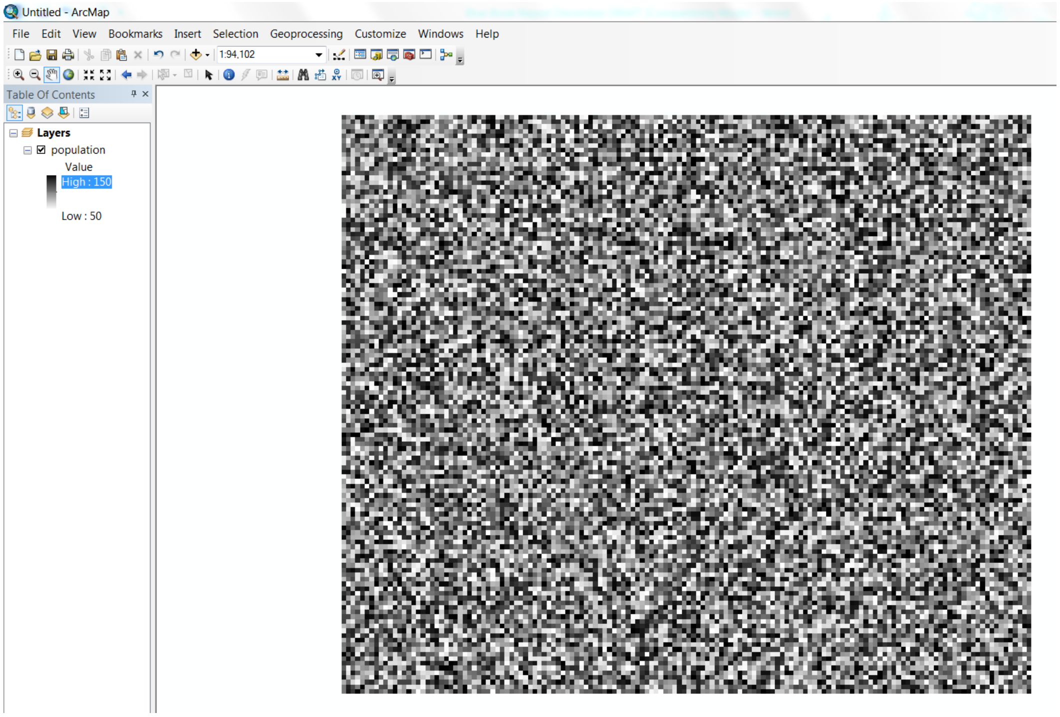


Figure 12. Random population values generated for each raster cell

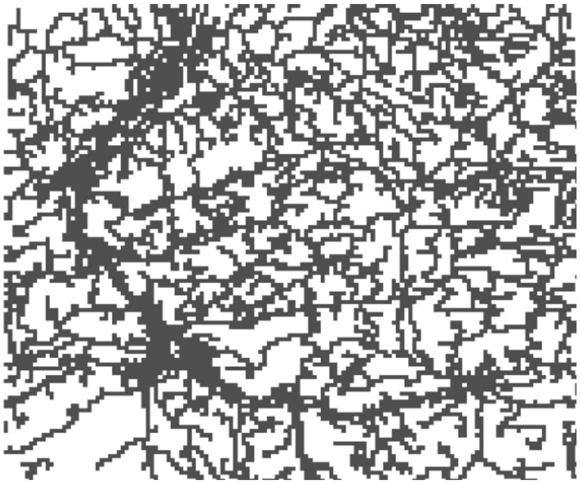


Figure 13. Rasterized road network

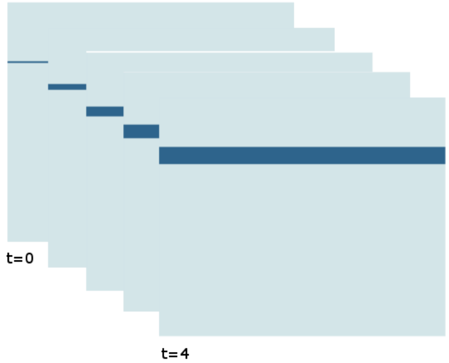


Figure 14. Example of simplified riverine flooding over time

Now that the illustrative example has been introduced, the following is a step-by-step guide that should help compile and run the CP for the hospital example.

1) Start by opening the CP (*.py* file) in IDLE (or equivalent IDE)
2) Edit those sections of the code which point to file locations on the host computer
    a. Typically, these are indicated by the following comment in the code :

       # edit item here; enter destination folder

    b. These locations will be places where a particular file is located on the host computer, if the accompanying hospital example files are saved to the host computer C: drive, these locations will look something like the following:

       "C: folder\subfolder\location"

    c. These lines occur at multiple places in the code. Each one should be commented appropriately.

3) Run the CP by selecting the “Run” option; this should open up another window (the Python Shell) which will execute the CP code.
    a. Some troubleshooting messaged will appear in the Python Shell that may be used to identify potential errors during program execution.
    b. If the Python Shell returns an error, it should provide an error code and/or brief description of where it occurred (line in the code) and what the problem may be. Some common errors and their possible solutions are provided in Table C 1 in Appendix C.
4) Once the CP has finished running, a message should appear that says *“Simulation complete!”*
5) Now the location on host computer that is selected to save the output files (one of the locations specified in Step 2) can be checked to see the raster and SD simulation outputs.
6) After the completion of simulation, it is recommended to undertake post-processing of the outputs for effective presentation and visualization of the results.

The outputs from the model simulation capture both the spatial and temporal dynamic characteristics of resilience. In summary, if no major modifications are made to the current version of the CP (and associated files), the following is expected spatial and temporal output data generated by the CP simulation for the health example:

Temporal Outputs

- Numerical data for each of the GAME variables in the Vensim model (printed to Python shell or can be accessed via the model simulation save file using Vensim software);
  Note: it is possible to retrieve numerical data for all of the variables (including non-GAME variables) in a simulation, however additional code is required.
- Temporal dynamics of important model variables (in this case “patients” and “resilience”) can be visualized as graphs produced by the CP (Figure 15).

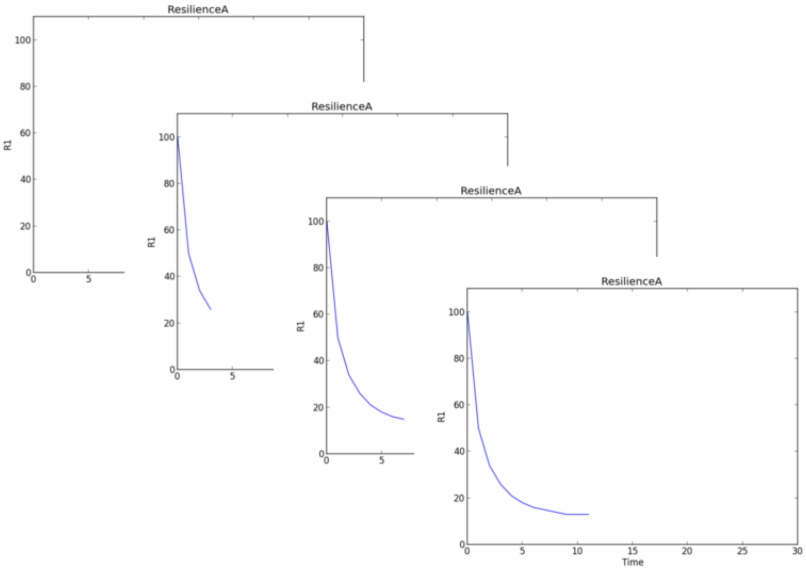


Figure 15. Example of temporal resilience simulation output

Spatial Outputs

- A raster file of size 124 x 148 cells which depicts a random population density value [0, 6) for each cell;
- A series of raster maps (one raster map for each time step) depicting:
  - Modified (flooded) road network;
  - Changing hospital service areas (SA Hospital A and SA Hospital B);
  - Health system resilience (Figure 16).

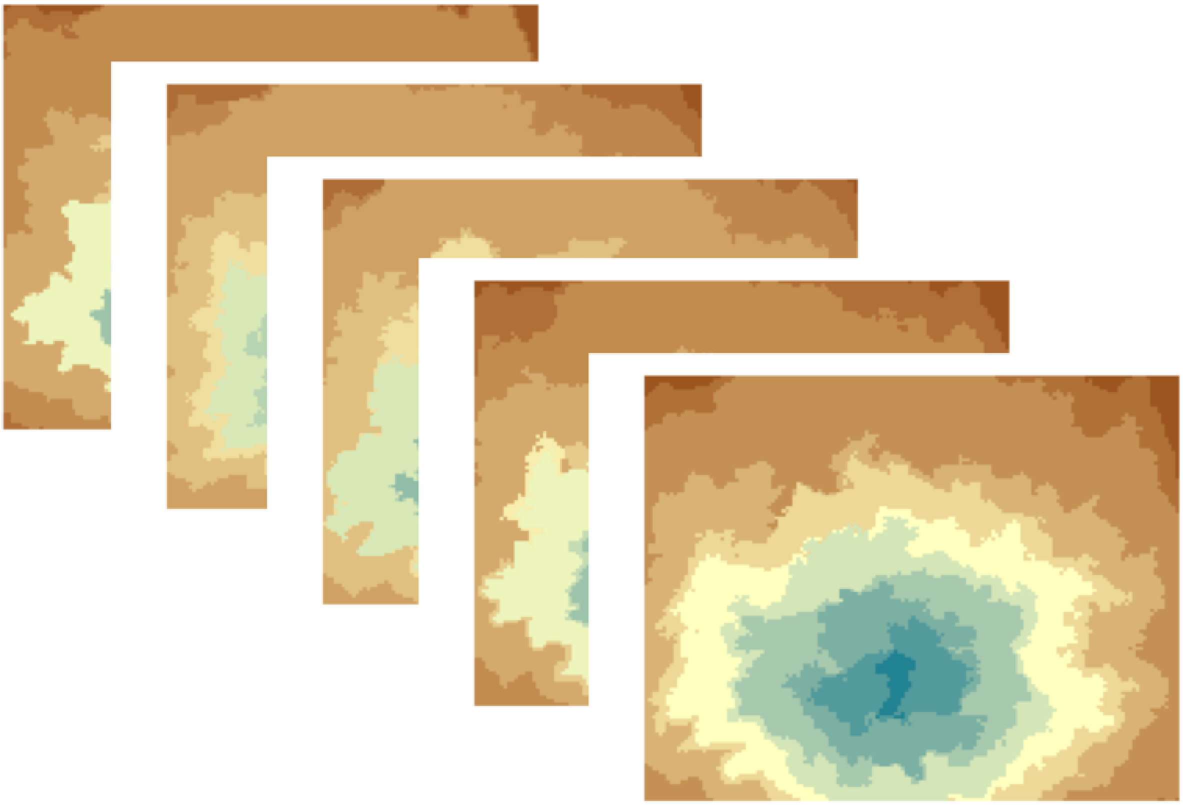


Figure 16. Example of spatial resilience output (blue colour indicates high resilience; red colour indicates low resilience)

The example maps of spatial resilience output (Figure 16) are currently generated considering the “cost distance” value of a particular cell. That is, the further away a cell is from either Hospital A or Hospital B, the lower its resilience value. As the road network becomes flooded, there is reduced access to flooded cells and cells which have been cut off from the transportation network (via flooded roads). These cells are given an additional “cost” which is then factored in to final resilience calculation. The purpose of the maps shown is to provide a visual example of possible representation of spatial resilience. In a more realistic example, spatial health resilience may include many other factors such as: emergency access routes; vehicle accessibility; or in-home patient care.

One of the greatest benefits in using the CRS is its flexibility and possibility to test various adaptation policy scenarios. In the current version of the CP, the easiest way to incorporate these scenarios is by assigning variables values in the CP code or modifying variables directly in the Vensim model before running the CP. Further description and discussion of policy scenarios (commonly known as “what if” scenarios) will be provided in the future.

The health resilience example presented in this report is representative of how SD and GIS will be used to assess system performances and resilience in the CRS, using real data. The example provides a starting point for programming a more complex CRS, which may include systems from other sectors (physical, economic, social, and organizational). When additional systems are added to the Vensim model, it becomes more complex, and the resilience calculations increases in computational effort.

It is important to note that the example presented above is a one-way interaction between Vensim and ArcGIS. This is because the spatial extent of the flood is driving the changes in hospital SAs. If, however, SAs were also influenced by variables in Vensim model, there would need to be a two-way interaction between Vensim and ArcGIS where SAs are continually redefined iteratively through a set of rules. If there is a dependence between a particular Vensim variable and space then the program can be modified to iteratively solve for the Vensim variable value. The current example does not include this two-way interaction. Further details on a more complex two-way interaction between Vensim and ArcGIS will be provided in the future.

The spatial-temporal dynamic resilience measure output from running the CP can effectively represent the impacts of, and system resilience to, natural disasters. This information can be valuable for investigating effective adaptation measures in emergency management, disaster preparedness, recovery action and future policy development.

4.2 Other applications

It’s possible to use the CP for other applications. With minor modifications, the current version of the CP can be extended to not only apply to the health sector as presented in this report, but also to other sectors (physical; social; economic; organizational) of the CCaR project. Each of the Vensim models would require similar pre- and post-processing steps as provided in this report. In future versions of the CP, ideally many of these pre- and post-processing steps will become automated and/or controlled via a more user-friendly graphical user interface.

5 CONCLUSIONS

This report presents the coupling program (CP) used to dynamically link together system dynamics and GIS software for the purposes of simulating both spatial and temporal dynamics of city impacts and resilience to natural disasters.

Although very illustrative, the current version of the CP provides insight into the importance of capturing both spatial and temporal dynamics of resilience behaviour. The current level of implementation is quite basic, but provides a foundation for developing future CPs between system dynamics and GIS software.

The current state of the coupling program is illustrative and does not use the real data. Therefore, the CP requires further refinement and adjustments before being implemented in the final version of the CRS. The CP will require modification for each of the CCaR project cities to best suit their specific Vensim models and GIS data. The modified coupling programs used by each city team will still require pre-processing of the model simulation files (i.e. published *.vpm* Vensim model file(s)) and post-processing of the simulation output (i.e. the complete set of raster files and the temporal graphs may require aesthetic adjustments). However, this coupling program is designed to be a template for future coupling programs. Thus, adjustments can still be made to the CP to address some of the post-processing limitations.

There will be additional issues to take into consideration when developing and running future versions of the CPs. Some of these items include:

- the spatial resolution of GIS data;
- the complexity of the system dynamics simulation model;
- which variables in the system dynamics model are spatially dependent;
- the number of simultaneous GIS analyses required;
- acceptable levels of aggregation;
- spatial boundaries (both physical and political in nature);
- computational effort;
- available resources (human, financial, technological and time); and
- CP developers level of programming experience.

These are all items which will require additional attention for future customization of the CP. Once the CP has been modified and re-developed for each project city, it will be a valuable tool in the assessment of spatial-temporal resilience assessment and be an integral part of the CRS.

Further details on the CCaR project can be found on the official project website at: coastalcitiesatrisk.org/wordpress.

Acknowledgements

The authors would like to acknowledge and appreciate the financial assistance provided by National Sciences and Engineering Council (NSERC) of Canada CGS doctoral scholarship to the first author and by the IDRC to the second author.

REFERENCES

Environmental Systems Research Institute, ESRI. (January 2014). Products: ArcGIS. http://www.esri.com/software/arcgis

MathWorks Inc. (February 2014). MATLAB: The language of technical computing. http://www.mathworks.com/products/matlab/

Neuwirth, C., Peck, A., and Simonovic, S.P. Evolution of non-spatial to spatial system dynamics simulation: A Daisyworld example. Environmental Modeling and Software. *In review.*

Peck, A. and Simonovic, S.P. (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.

Python.org. (October 2013). Python 3.0 Release. http://www.python.org/download/releases/3.0/

Python.org. (January 2014). History and Licence. http://docs.python.org/3.3/license.html#

Simonovic, S.P. and Peck, A. (2013). Dynamic Resilience to Climate Change Caused Natural Disasters in Coastal Megacities: Quantification Framework. British Journal of Environment and Climate Change, 3(3), 378-401.

Srivastav, R. and Simonovic, S.P. (2014). Generic Framework for Computation of Spatially 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) XXX-X-XXXX-XXXX-X; (online) XXX-X-XXXX-XXXX-X.

van Rossum, G. and Drake Jr., F.L. (2013). Python v.2.7.5 Documentation. Python Software Foundation. Retrieved 23.09.2013, from http://docs.python.org/2.7/

Ventana Systems (October 2013). Vensim Documentation. http://vensim.com

Watson, A. J., & Lovelock, J. E. (1983). Biological homeostasis of the global environment: the parable of Daisyworld. Tellus B, 35(4), 284-289.

APPENDIX A – The program (Python code)

The following is the Python code used for the version of CP in this report. Highlighted areas require the user to input the correct directories to files stored on the host computer. Other highlighted sections of the code may be of interest to the user.

```
################################################################################
#
# The Python code to execute Vensim model simulation and data retrieval to
# be used as input into ArcGIS Python code
# Created by: Angela Peck and Christian Neuwirth
# Last modified: March-2014
#
################################################################################

################################################################################
#
# This section of the code imports the necessary Python libraries
# and defines a parameter used in formatting output to the Python shell.
#
################################################################################

import os
import ctypes
import ctypes.util
import numpy
import random
import arcpy
from arcpy import env
from arcpy.sa import *
import sys
from matplotlib import pyplot as plt
from matplotlib import animation


# This is used to organize the print outs in the Shell
hr = "--------------------"

#################################################################################
#
# Import the GIS package, which contains all self-made GIS modules
#
#################################################################################

sys.path.append("C:/")
import GIS_Package as my_modules

#################################################################################
#
# The following section defines required GIS settings and data files
#
#################################################################################

#Print the whole array
numpy.set_printoptions(threshold=numpy.nan)

#Enable Overwriting
arcpy.env.overwriteOutput = True

# Check out the ArcGIS Spatial Analyst extension license
arcpy.CheckOutExtension("Spatial")

if arcpy.CheckExtension("Spatial") == "Available": print "The GIS spatial extension is Available"
else: print "Not Available"
```

```
#Set workspace directory

arcpy.env.workspace = "C:/"


#Define output coordinate system

arcpy.env.outputCoordinateSystem = "C:/"


#Create Raster Object "Hospitals"

Hospitals = arcpy.Raster("C:/")


#Create Raster Object "CostSurface"

CostSurface = arcpy.Raster("C:/")


#Create Raster Object "Ownership"

Ownership = arcpy.Raster("C:/")


#Create Raster Object "Population"

PopRaster = arcpy.Raster("C:/")


#Create Raster Object "Flood"

FloodRaster = arcpy.Raster("C:/")


################################################################################

#

# This section directs the Python program to the Vensim DLL file and then

# loads the DLL.

#

# The function find_library searches along the system search path and

# returns the full pathname, but since there is no predefined naming

# scheme, a call like find_library("c") will fail and return None.

#

# It is important to note that Vensim Single Precision installation will
# download a different DLL file than the Double Precision software.
# Therefore, you must select the DLL file that will be used, based on your
# current installation of Vensim and comment out the other DLL file.
#
############################################################################

lib_name = "vendll32.dll"      # Single precision DLL file
# lib_name = "vdpdll32.dll"    # Double precision DLL file
print("Library: " + lib_name)

# Find the dynamic link library in Windows.
lib_path = ctypes.util.find_library(lib_name)
print(lib_path)

# If the library path cannot be found, the program will return an error
# message and exit
if lib_path == None:
   print("Unable to find dll: "+ lib_name)
   exit()

print(hr)

# Load the dll
my_dll = ctypes.windll.LoadLibrary(lib_path)
print("DLL Load Result: ")
print(my_dll)
print(hr)

# Status
def ven_stat():
   print("Status: ")
   print(my_dll.vensim_check_status())
```

```
    print(hr)

ven_stat()

# A call to vensim_command should avoid a status check return of 6; which
# indicates that Vensim had not freed all its working memory
def ven_cmd(cmd):
    print("Command: "+ cmd)
    #buf = bytes(cmd, 'utf8') # For Python v.3 only
    result = my_dll.vensim_command(cmd)
    print("Result: ")
    print(result)
    ven_stat()

print("Empty Command, then status: ")

#################################################################################
#
# This section loads a Vensim model. In order for this to work, the model
# must be a published .vpm file. To save your Vensim model in this format,
# please refer to this program's Blue Book report or Vensim Help manual.
#
#################################################################################

# Load a Vensim model
model_path = r"C:/.../HealthCombined6_100.vpm"
model_cmd = r"SPECIAL>LOADMODEL|"

print(os.path.exists(model_path))
ven_cmd(model_cmd + model_path)

#################################################################################
```

```
#
# Upon successfully loading a model, I would like to have simulation
# control over that model which would allow me to perform simulations
# bit-by-bit and retrieve information and get feedback on simulation
# progress; otherwise Vensim will perform the entire simulation before
# returning control to the Python program. Once I have control over the
# simulation, I would like to get values for specific game variables.
#
################################################################################
#
# This section of the program will define the Vensim "game" controls:
# start, continue, end. And also defines the game interval and game time.
#
################################################################################

# Start a game
def ven_game_start():
   ven_cmd("MENU>GAME")

# Continue the game by the game interval
def ven_game_continue():
   ven_cmd("GAME>GAMEON")

# End the game
def ven_game_end():
   ven_cmd("GAME>ENDGAME")
   print "The Gaming Simulation is Complete!"

# Set the game interval
def ven_game_interval():
   ven_cmd("GAME>GAMEINTERVAL|1")
```

```
# Set the game time (number of time steps you wish to run)
ven_game_time_range = 30

# Create variables for simulation visualization
game_valT_record = numpy.zeros(ven_game_time_range)
game_valtransB_record = numpy.zeros(ven_game_time_range)
game_valtransA_record = numpy.zeros(ven_game_time_range)

###########################################################################
#
# This section of the program defines the Vensim get value function
# which is required in order to retrieve game variable values during a
# gaming simulation. It also defines the functions for writing changes
# that occur during a game (similar to a log of changes).
#
###########################################################################

# Retrieve the value of a variable during the game
def get_val(name):
   value = ctypes.c_float(0)
   return_val = my_dll.vensim_get_val(name, ctypes.byref(value))
   if return_val ==0:
      raise VensimWarning("variable not found")
   else:
      val_num = numpy.array(value.value)
      val_num_int = numpy.around(val_num)
      return val_num_int

# Write the model changes so far into a file called ginFile
def write_gin():
   ginFile_path = r"C:\...\GinFile.txt"
   ginFile_cmd = r"GAME>WRITEGIN|"
```

```
    ven_cmd(ginFile_cmd + ginFile_path)

# Write all Game variables at all times from vdffilename to the gaming input file
def write_gvars():
    vdfFile_name = r"C:\...\file.vdf|"
    gvarsFile_path = r"C:\...\GvarsFile.txt"
    gvarsFile_cmd = r"GAME>WRITEGVARS|"
    ven_cmd(gvarsFile_cmd + vdfFile_name + gvarsFile_path)

################################################################################
#
# The following section is the GIS portion of the code. It completes the
# steps:
#
# 1. Calculate cost distance from the Hospital to each raster cell in
# the simulation environment
# 2. Calculate the initial shadow price (time = 0) of each raster cell for
# each hospital's service area based on the cost distance.
#
################################################################################

HospitalID = [1,2]

x = 0
for i in HospitalID:
    x+=1
    cost = my_modules.calcostdistance.costsurface(Hospitals, CostSurface, i)
    cost.save("costDistOrig" + str(i))
    vars()["shadowpr" + str(i)] = my_modules.calshadow.shadprice(Ownership, cost, i)

################################################################################
#
```

```
# Get initial service area size of Hospital 1.
# The initial service area size of Hospital 2 is required to compare it to
# the Vensim hospital size. The service area size of Hospital 1 implies the
# service area of Hospital 2, since the entire servicable area remains the
# same.
#
###################################################################################################

# Conversion of Ownership raster to numpy array, zeros are assigned no
# data values

Ownership_Array = arcpy.RasterToNumPyArray(Ownership,"","","",0)

#Calculate current servicable area size in GIS, sum of all elements with value 1

SizeFarm1_ArcGIS = numpy.sum(Ownership_Array == 1)

###################################################################################################
#
# Use a fictive population for each raster cell. These values will be
# randomly generated for this fictive example.
#
###################################################################################################

# Define population range variables

popLow = 0
popHigh = 6

# Define raster properties

rasterHeight = 124
```

```
rasterLength = 148
rasterSize = (rasterHeight, rasterLength)

print rasterSize

# Create a random numpy array in range X to Y with length Z

rdmPopArray = numpy.random.randint(popLow,popHigh,rasterSize)

# Convert to 2D shape

rdmPopArray.shape = (rasterHeight,rasterLength)

# Print lines to check

print "The raster height is "
print rasterHeight
print "The raster length is "
print rasterLength
print "The random Population array is "
print rdmPopArray

# To save the array as a raster file

resolution1 = CostSurface.meanCellHeight
resolution2 = CostSurface.meanCellWidth
lowerleft = CostSurface.extent.lowerLeft
rdmPopRaster = arcpy.NumPyArrayToRaster(rdmPopArray,lowerleft,resolution1,resolution2)
rdmPopRaster.save("population")

##########################################################################
#
```

```
# Use a fictive flood for each raster cell. These values will be
# randomly generated for each time step in this fictive example.
#
##########################################################################

# Define flood range variables

floodLow = 0
floodHigh = 2

# Create a random integer numpy array in range X to Y with length Z (same as
# raster size for the random population layer). Note: that the randint is
# such that randint[low, high) (does not include "high" value in random
# sample).

##########################################################################
#
# Define new cost surface because it will change as the flood changes the
# accessable routes.
#
##########################################################################

# This is the cost surface (rasterized street network)
# The cost surface changes over time, since a flood comes in and temporarily
# blocks streets

costRasterOriginal = CostSurface

# Convert the costSurface to a numpy array

costArrayOriginal = arcpy.RasterToNumPyArray(costRasterOriginal)
```

```
costArray = numpy.zeros(rasterSize)

print(hr)

################################################################################
#
# To generate the serviceable population for each hospital, sum up the
# values of the population in each cell, belonging to each hospital.
#
################################################################################

# Convert the raster object to a numpy array in order to overlay it with the
# fictive population array

costAllocOut_Array = arcpy.RasterToNumPyArray(costRasterOriginal)

# Initialize the population in hospital service area A and hospital service
# area B

service_popA = numpy.array([0])
service_popB = numpy.array([0])

# Initialize the size of hospital service area A and hospital service
# area B

HospitalIdA = 1
HospitalIdB = 2

################################################################################

# Begin the game
# Initiate the game interval, start of the game and continuing of the game
# to the next time step (defined by interval)

ven_game_interval()

ven_game_start()

itnum = 0

##################################################################################
#
# Transportation "costs" are calculated based on the servicable areas and
# transfered back to Vensim
#
##################################################################################

# One step in this "for" loop represents one time step in Vensim
for i in range(ven_game_time_range):

    #Set dynamically changing cost array to original cost array
    costArray[:] = costArrayOriginal[:]

    #Create flood array
    rdmFloodArray = numpy.random.randint(floodLow,floodHigh,rasterSize)
    rdmFloodArray.shape = (rasterHeight,rasterLength)
    rdmFloodRaster = arcpy.NumPyArrayToRaster(rdmFloodArray,lowerleft,resolution1,resolution2)
    rdmFloodRaster.save("flood" + str(i))

    # Check to see which cells are flooded
    for y in range (rasterHeight):
        for z in range (rasterLength):
            if rdmFloodArray[y][z] == 1: #and costArray[y][z] <= 0.002:
                costArray[y][z] = 0.005
            else:
                pass

    #Save dynamically changing cost raster
    costRaster = arcpy.NumPyArrayToRaster(costArray,lowerleft,resolution1,resolution2)
    costRaster.save("cost" + str(i))

    inFeature = Hospitals
    costAllocOut = CostAllocation(inFeature, costRaster, "", "", "", "", "")
    costAllocOut.save("Allocation" + str(i))

    costAllocOut_Arr = arcpy.RasterToNumPyArray(costAllocOut)

    # Need to calculate the cost distance for each new changing cost surface
    # As the streets become flooded, this will affect the cost distances for
    # each hospital. You get a new cost distance raster for each hospital.

    n = 0
    for u in HospitalID:
        n+=1
        costDist = my_modules.calcostdistance.costsurface(Hospitals, costRaster, u)
        costDist.save("costDist" + str(u) + str(i))

    # Reclassify the cost distance raster to be an example of Resilience calc.

        reMap = RemapRange([[0,5,100],[5,10,80],[10,15,60],[15,20,40],[20,30,30],[30,40,20],[40,100,10]])
        rClassCostDist = Reclassify(costDist,"Value",reMap)
        rClassCostDist.save("Reclass" + str(u) + str(i))

    # Sum up the population in each of the service areas
    # This nested loop iterates over the costAllocOut_Array and sums up the
    # population (array defined previously as "rdmPop") in service areas A and B.
```

```
    # This has to be calculated each time the cost surface changes. Raster cells
    # need to be newly allocated and summed up based on the updated allocation.
    for e in range (rasterHeight):
        for f in range (rasterLength):
            if costAllocOut_Arr[e][f] == HospitalIdA:
                service_popA = service_popA + rdmPopArray[e][f]
            elif costAllocOut_Arr[e][f] == HospitalIdB:
                service_popB = service_popB + rdmPopArray[e][f]
            else:
                print "cell is not allocated to one of the hospitals"

    ven_cmd("SIMULATE>SETVAL|initial affected population A=" + str(service_popA))
    ven_cmd("SIMULATE>SETVAL|initial affected population B=" + str(service_popB))

    print "Population serviced by Hospital A is "
    print service_popA
    print "Population serviced by Hosptial B is "
    print service_popB

    #Set dynamically changing allocation array to original allocation array
    costAllocOut_Arr[:] = costAllocOut_Array[:]

    #Set dynamically changing service populations back to zero
    service_popA = 0
    service_popB = 0

    print(hr)

##############################################################################
#
# The serviceable area size in Vensim is transferred to the spatial
# representation in GIS. Spatial changes in ownership are based on the
# shadow price.
#
##############################################################################

    name = "Injured Patients A"
    game_valA = get_val(name)
    print("Thus, game value of " + name)
    print("is = " )
    print(game_valA)

    name = "Injured Patients B"
    game_valB = get_val(name)
    print("Thus, game value of " + name)
    print("is = " )
    print(game_valB)

    name = "RHO A"
    game_valC = get_val(name)
    print("Thus, game value of " + name)
    print("is = " )
    print(game_valC)

    name = "RHO B"
    game_valD = get_val(name)
    print("Thus, game value of " + name)
    print("is = " )
    print(game_valD)

    name = "H1 RESILIENCE A"
    game_valE = get_val(name)
    print("Thus, game value of " + name)
    print("is = " )
    print(game_valE)

    name = "time"
    game_valT = get_val(name)
    game_valT_record[i] = game_valT
    print hr
    print game_valT_record[i]

    name = "Injured Patients A"
    game_valtransB = get_val(name)
    game_valtransB_record[i] = game_valtransB
    print game_valtransB_record[i]

    name = "H1 RESILIENCE A"
    game_valtransA = get_val(name)
    game_valtransA_record[i] = game_valtransA
    print game_valtransA
    print hr

    # To continue the Vensim game to the next time step
    ven_game_continue()

    print(hr)

################################################################################
#
# This section is for creating an example "spatial resilience" layer for
# some other "type" of resilience and is representative ONLY in order to
# demonstrate the overall Total Resilience Calculation.
#
################################################################################
```

```
for i in range (ven_game_time_range):

    inRaster1 = arcpy.Raster("reclass1" + str(i))
    inRaster2 = arcpy.Raster("reclass2" + str(i))

    R_Total = inRaster1 + inRaster2
    R_Total.save("SumRaster" + str(i))

####################################################################################
#
# This section of the code completes the gaming simulation. Once the game
# has gone through all of the time step iterations, we would like to save
# the changes in log files and end the gaming simulation by calling their
# corresponding functions.
#
####################################################################################

write_gvars()

write_gin()

ven_game_end()

####################################################################################
#
# Get time and values of the whole simulation for the visualization
#
####################################################################################

# Print time and resilience of the whole simulation
print "Time succession = "
print game_valT_record
```

```
#print hr
print "Injured Patients A = "
print game_valtransB_record
#print hr
print "Rho A = "
print game_valtransA_record
print hr

# This function sets the value for lineA
def update_line_transA(numA, dataA, lineA):
   lineA.set_data(dataA[...,:numA])
   return lineA,

def update_line_transB(numB, dataB, lineB):
   lineB.set_data(dataB[...,:numB])
   return lineB,

################################################################################
#
# Creating Figure 1
# Define the settings for the plot
#
################################################################################

# Create Figure 1
fig1 = plt.figure()

# Define an empty line and assign the color blue for Figure 1
lineOne, = plt.plot([], [], 'b-')

#Define the dimensions for x and y axis Figure 1
plt.xlim(0, ven_game_time_range)
plt.ylim(0, 110)

#Define axis labels and title of Figure 1
plt.xlabel('Time')
plt.ylabel('R1')
plt.title('ResilienceA')

################################################################################
#
# Creating Figure 2
# Define the settings for the plot
#
################################################################################

#Create Figure 2
fig2 = plt.figure()

#Define an empty line and assign the color red for Figure 2
lineTwo, = plt.plot([], [], 'r-')

#Define the dimensions for x and y axis Figure 2
plt.xlim(0, ven_game_time_range)
plt.ylim(0, 7000)

#Define axis labels and title of Figure 2
plt.xlabel('Time')
plt.ylabel('Injured Patients')
plt.title('Patients')

################################################################################
#
# Create two animated figures to visualize the dynamically changing
# costs. The diagrams are created using matplotlib.
#
#############################################################################################

#############################################################################################
#
# Creating an Animation
# Define the settings for the animation
#
#############################################################################################

#Append time and cost values in data array
dataA = numpy.float64([game_valT_record, game_valtransA_record])
dataB = numpy.float64([game_valT_record, game_valtransB_record])

#Define the number of animation time steps (frames)
vis_step = ven_game_time_range + 1

line_ani_one = animation.FuncAnimation(fig1, update_line_transA, vis_step, fargs=(dataA, lineOne),
interval=200, blit=False)

line_ani_two = animation.FuncAnimation(fig2, update_line_transB, vis_step, fargs=(dataB, lineTwo),
interval=200, blit=False)

plt.show()
```

APPENDIX B – Vensim Features

Table B 1. Features of PLE, PLE Plus, Professional, DSS, and Model Reader (from vensim.com)

Feature	**PLE**	**PLE Plus**	**Pro**	**DSS**	**Read**
Feature	PLE	PLE Plus	Pro	DSS	Read
SyntheSim	X	X	X	X	X
Ability to cut feedback links on the fly in SyntheSim mode	X	X	X	X	
Sketch Editor with Undo/Redo	X	X	X	X	
Causal Loop Diagrams	X	X	X	X	X
Stock and Flow Diagrams	X	X	X	X	X
Tree Diagrams	X	X	X	X	X
Document Tool	X	X	X	X	X
Loop Identification	X	X	X	X	X
Equation Editor	X	X	X	X	
Built-in Functions	X	X	X	X	
Units Check	X	X	X	X	
Causal Tracing®	X	X	X	X	X
Reality Check®	X	X	X	X	X
Simulation	X	X	X	X	X
Graphs	X	X	X	X	X

Feature	PLE	PLE Plus	Pro	DSS	Read
Tabular Output (Tables)	X	X	X	X	X
Run Comparison (between two simulations)	X	X	X	X	X
On-line Help	X	X	X	X	
Multiple Views (pages or sectors of a model)	X	X	X	X	X
Input and Output Sketch Objects	X	X	X	X	X
Gaming		X	X	X	X
Sensitivity Simulations (Monte Carlo)		X	X	X	X
External Data Import/Export (spreadsheet etc.)		X	X	X	X
Live Data Connections		X	X	X	X
Discrete Event Functions			X	X	X
Editable Sketch Toolsets			X	X	
Hide Sketch Elements		X	X	X	X

Feature	PLE	PLE Plus	Pro	DSS	Read
Simulation Control Dialog			X	X	
Partial Model Simulation			X	X	
User-configurable Tools			X	X	
Histograms			X	X	X
Password Protection			X	X	X
Model Calibration (optimization)			X	X	X
Policy Optimization			X	X	X
Kalman Filtering			X	X	
Subscripts (Arrays) - up to 8 dimensions			X	X	X
Bar Graphs			X	X	X
Gantt Charts			X	X	X
Summary Statistics			X	X	X
Text Editor			X	X	
User defined Macros			X	X	X
ODBC Capabilities				X	X
Packaged Applications (Venapps, flight				X	X

Feature	PLE	PLE Plus	Pro	DSS	Read
simulators)					
Graphical Venapp Builder				X	
Dynamic Data Exchange				X	
External Functions				X	X
Compiled Models (C language)				X	X
DLL Configuration				X	X

APPENDIX C – Troubleshooting Common Runtime Errors

Table C 1. Table of common errors and possible methods of troubleshooting

Error	Description	Suggested method of troubleshooting
Import error: module cannot be found	This is because the CP cannot find the name of the file(s) or module that you are attempting to load	Check that all of the places in the CP are referring to the correct file/folder location of the file, module, or package you are attempting to load. (i.e. "C:/Folder/file.extension)
Stop from Vensim: the file name XXXX.vdf is protected	This is because the CP is trying to run a simulation under the same name (e.g. "Current.vdf") as a previous simulation.	This stop will allow you to specify another name for the "run name" of the current simulation, or you may choose to overwrite your previous simulation run by leaving the name unchanged.
System Exit: The program is still running! Do you want to kill it?	This could spawn for any number of reasons, but the most likely reason for this error is that the *.dll* file cannot be found by the CP.	First, check that the right .dll file is being called within the code. There is a separate call for the single precision version of Vensim and the double precision version of Vensim. Be sure to comment out in the code the name of the DLL file that you are not using. If this does not work, double check that the *.dll* file hasn't been renamed or moved since initially installing Vensim DSS. If there is still an error, you may have to reinstall Vensim DSS and associated files. Make sure you select correct options during installation to download the DLL files.

APPENDIX D – List of Previous Reports in the Series

ISSN: (print) 1913-3200; (online) 1913-3219

In addition to the following publications, there are 65 reports (no. 01 – no. 065) prior to 2010.

(1) Leanna King, Tarana Solaiman, and Slobodan P. Simonovic (2010). Assessment of Climatic Vulnerability in the Upper Thames River Basin: Part 2. Water Resources Research Report no. 066, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 72pages. ISBN: (print) 978-0-7714-2834-0; (online) 978-0-7714-2835-7.

(2) Christopher J. Popovich, Slobodan P. Simonovic and Gordon A. McBean (2010). Use of an Integrated System Dynamics Model for Analyzing Behaviour of the Social-Economic-Climatic System in Policy Development. Water Resources Research Report no. 067, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 37 pages. ISBN: (print) 978-0-7714-2838-8; (online) 978-0-7714-2839-5.

(3) Hyung-Il Eum and Slobodan P. Simonovic (2009). City of London: Vulnerability of Infrastructure to Climate Change; Background Report 1 – Climate and Hydrologic Modeling. Water Resources Research Report no. 068, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 102pages. ISBN: (print) 978-0-7714-2844-9; (online) 978-0-7714-2845-6.

(4) Dragan Sredojevic and Slobodan P. Simonovic (2009). City of London: Vulnerability of Infrastructure to Climate Change; Background Report 2 – Hydraulic Modeling and Floodplain Mapping. Water Resources Research Report no. 069, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 147 pages. ISBN: (print) 978-0-7714-2846-3; (online) 978-0-7714-2847-0.

(5) Tarana A. Solaiman and Slobodan P. Simonovic (2011). Quantifying Uncertainties in the Modelled Estimates of Extreme Precipitation Events at the Upper Thames River Basin. Water Resources Research Report no. 070, Facility for Intelligent Decision Support, Department of Civil and Environmental

Engineering, London, Ontario, Canada, 167 pages. ISBN: (print) 978-0-7714-2878-4; (online) 978-0-7714-2880-7.

(6) Tarana A. Solaiman and Slobodan P. Simonovic (2011). Assessment of Global and Regional Reanalyses Data for Hydro-Climatic Impact Studies in the Upper Thames River Basin. Water Resources Research Report no. 071, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 74 pages. ISBN: (print) 978-0-7714-2892-0; (online) 978-0-7714-2899-9.

(7) Tarana A. Solaiman and Slobodan P. Simonovic (2011). Development of Probability Based Intensity-Duration-Frequency Curves under Climate Change. Water Resources Research Report no. 072, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 89 pages. ISBN: (print) 978-0-7714-2893-7; (online) 978-0-7714-2900-2.

(8) Dejan Vucetic and Slobodan P. Simonovic (2011). Water Resources Decision Making Under Uncertainty. Water Resources Research Report no. 073, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 143 pages. ISBN: (print) 978-0-7714-2894-4; (online) 978-0-7714-2901-9.

(9) Angela Peck, Elisabeth Bowering and Slobodan P. Simonovic (2010). City of London: Vulnerability of Infrastructure to Climate Change, Final Report - Risk Assessment. Water Resources Research Report no. 074, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 66 pages. ISBN: (print) 978-0-7714-2895-1; (online) 978-0-7714-2902-6.

(10) Akhtar, M. K., S. P. Simonovic, J. Wibe, J. MacGee, and J. Davies, (2011). An integrated system dynamics model for analyzing behaviour of the social-energy-economic-climatic system: model description. Water Resources Research Report no. 075, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 211 pages. ISBN: (print) 978-0-7714-2896-8; (online) 978-0-7714-2903-3.

(11) Akhtar, M. K., S. P. Simonovic, J. Wibe, J. MacGee, and J. Davies, (2011). An integrated system dynamics model for analyzing behaviour of the social-energy-economic-climatic system: user's manual.

Water Resources Research Report no. 076, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 161 pages. ISBN: (print) 978-0-7714-2897-5; (online) 978-0-7714-2904-0.

(12) Millington, N., S. Das and S. P. Simonovic (2011). The Comparison of GEV, Log-Pearson Type 3 and Gumbel Distributions in the Upper Thames River Watershed under Global Climate Models. Water Resources Research Report no. 077, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 53 pages. ISBN: (print) 978-0-7714-2898-2; (online) 978-0-7714-2905-7.

(13) Andre Schardong and Slobodan P. Simonovic (2012). Multi-objective Evolutionary Algorithms for Water Resources Management. Water Resources Research Report no. 078, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 167 pages. ISBN: (print) 978-0-7714-2907-1; (online) 978-0-7714-2908-8.

(14) Samiran Das and Slobodan P. Simonovic (2012). Assessment of Uncertainty in Flood Flows under Climate Change: the Upper Thames River basin (Ontario, Canada). 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.

(15) Rubaiya Sarwar, Sarah E. Irwin, Leanna M. 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.

(16) 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.

(17) 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.

(18) 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.

(19) Abhishek Gaur and Slobodan P. Simonovic (2013). Climate Change Impact on Flood Hazard in the Grand River Basin, London, Ontario, Canada. Water Resources Research Report no. 084, Facility for Intelligent Decision Support, Department of Civil and Environmental Engineering, London, Ontario, Canada, 92 pages. ISBN: (print) 978-0-7714-3063-3; (online) 978-0-7714-3064-0.

(20) 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.