Model Name: LAVegMod

Functional Area: Louisiana Coastal Zone Vegetation

Model Proponents: Coastal Protection and Restoration Authority

Model Developer(s):

Dr. Jenneke M. Visser, School of Geoscience and Institute for Coastal Ecology and Engineering, University of Louisiana at Lafayette
Dr. Scott Duke-Sylvester, Department of Biology, University of Louisiana at Lafayette
Dr. Whitney Broussard, Institute for Coastal Ecology and Engineering, University of Louisiana at Lafayette
Dr. Jacoby Carter, U.S. Geological Survey - National Wetlands Research Center

Table of Contents

1. Background 1

a. Purpose of Model 1

b. Model Description and Depiction 1

c. Contribution to Planning Effort 3

d. Description of Input Data 3

e. Description of Output Data 11

f. Statement on the capabilities and limitations of the model 12

g. Description of model development process including documentation on testing conducted (Alpha and Beta tests) 13

2. Technical Quality 18

a. Theory 18

b. Analytical requirements 21

c. Assumptions 21

d. Identification of formulas used in the model and proof that the computations are appropriate and done correctly 21

3. System Quality 23

a. Description and rationale for selection of supporting software tool/programming language and hardware platform 23

b. Proof that the programming was done correctly 23

c. Availability of software and hardware required by model 23

d. Description of process used to test and validate model 24

e. Discussion of the ability to import data into other software analysis tools (interoperability issue) 25

4. Usability 26

a. Availability of input data necessary to support the model 26

b. Formatting of output in an understandable manner 26

c. Usefulness of results to support project analysis 26

d. Ability to export results into project reports 26

e. Training availability 27

f. Users documentation availability and whether it is user friendly and complete 27

g. Technical support availability 27

h. Software/hardware platform availability to all or most users 27

i. Accessibility of the model 27

j. Transparency of model and how it allows for easy verification of calculations and outputs 28

5. Sources of model uncertainty 28

6. Suggested model improvements **28**
7. Quality review **28**
8. Uncertainty analysis **29**
9. References **29**
Attachment 1 ..... **32**
Attachment 2 ..... **52**
Attachment 3 ..... **54**

1. *Background*

a. Purpose of Model

Vegetation models are an integral part of any ecosystem modeling project (DeAngelis 1998, Davis and Ogden 1994). Vegetation composition is an important input parameter for habitat suitability models for animals and hydrodynamic models (Figure 1). The 2012 Coastal Master Plan has as one of its goals to maintain a diverse array of fish and wildlife habitats. To determine if a project contributes to this goal, information on vegetation composition is essential.

The vegetation model (LAVegMod) described in this appendix is the next generation of a similar model (habitat switching module) initially developed as part of the Louisiana Coastal Area study (Visser et al. 2003). LAVegMod divides the original 5-habitat model for the Louisiana coast into 19 vegetation types. LAVegMod provides longer estimates of interannual variation in aboveground biomass; however, biomass varies with vegetation type.

The model is intended as a tool for protection and restoration planning that provides repeatable estimates of the changes in species composition based on changes in hydrology and wetland area. This tool was developed for use by a trained modeler and is dependent on other models that forecast changes in hydrology and wetland area.

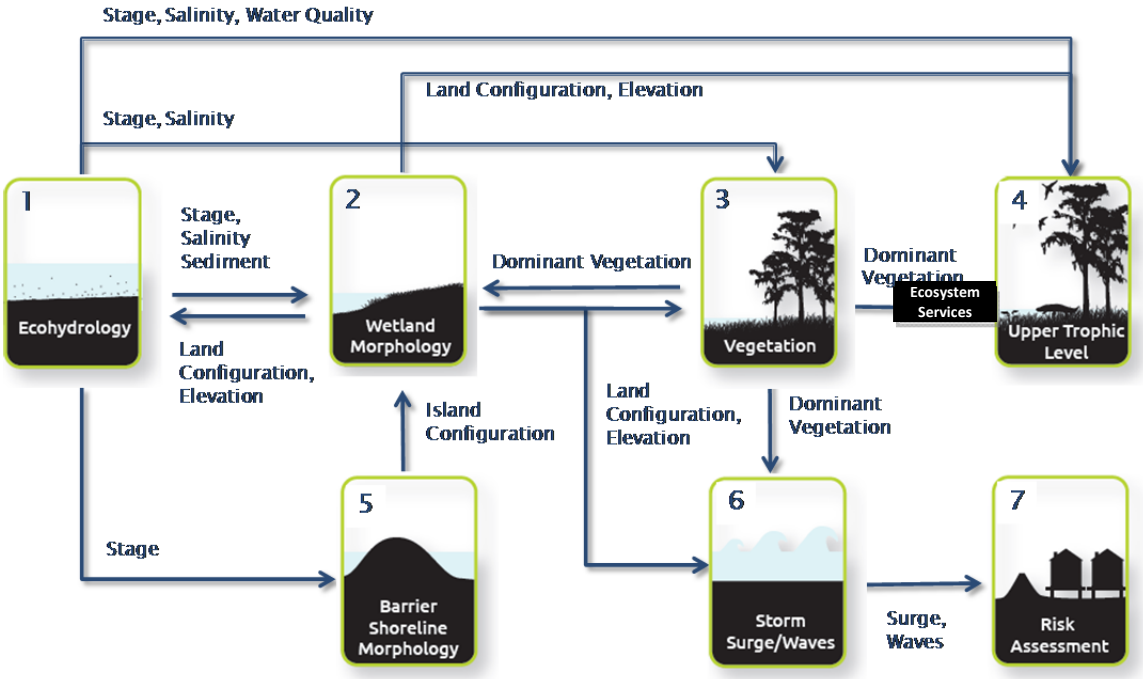

Figure 1. Interaction of models developed for the 2012 Coastal Master Plan. The Vegetation model (3) in this diagram is LAVegMod. The diagram shows how output from LAVegMod is used as input to Wetland Morphology (2), Ecosystem Services (4), and Storm Surge and Waves (6).

b. Model Description and Depiction

LAVegMod projects the annual net change in vegetation composition over the entire Louisiana coastline at a 500x500 meter resolution. LAVegMod is currently used to project changes over the 50-year time horizon (2010 to 2060) defined for the 2012 Coastal Master Plan analysis.

LaVegMod is composed of two components. One component handles the change in submerged aquatic vegetation (SAV) cover while the other handles the change in cover for 19 emergent vegetation types. The separation of SAV from the other vegetation types reflects the availability of a unique data set for SAV that allowed us to develop a model of SAV response based on a multiple linear regression relating SAV cover to water depth, salinity and water temperature. For the other 19 vegetation types, the dynamics of vegetation cover are based on the assumption that each plant species occupies a unique environmental niche that is defined by the range of abiotic conditions the species can tolerate. Changes in the composition of the plant community result as environmental conditions move beyond the niche of one species and into the environmental niche of another. The environmental niche for each species is defined by the range of salinity conditions and water-level variation that each species can tolerate. The choice of these two environmental categories is based on our analysis of the Coast wide Reference Monitoring System (CRMS) hydrology data (Folse et al. 2008) which showed that annual mean salinity and annual water-level variation produced the best distinction between species (Gregg Snedden USGS personnal communication), within the limit of the hydrology data (hourly observation of stage, salinity and temperature) that is currently available from CRMS. The hydrology data currently used by LAVegMod as input data consists of daily stage, average monthly salinity, and water temperature.

Figure 2 illustrates our approach to simulating vegetation change based on niche and changes in environmental conditions. At year 1, a 500x500 meter cell starts with a plant community defined by a particular community structure, shown in Figure 2 as two vegetation types A and B in addition to a portion of the cell covered by open water, a portion of which is predicted to be occupied by SAV based on the hydrologic conditions (stage, salinity, and temperature). Hydrology and salinity input into LAVegMod can result in a shift in environmental conditions out of the niche for the species that characterize vegetation types A and B resulting in an increased mortality rate for these species and an associated reduction in cover for the vegetation type. The same change in hydrology and salinity can bring local environmental conditions within the niche breadth of a new species that can establish and occupy the area vacated by species A and B changing it to a new vegetation type. If the environmental conditions remain in the optimal range for the species present, no mortality occurs and the vegetation type occupies the same area. LAVegMod can also respond to changes in the relative amount of open water present in a cell. In the example in Figure 2, input to LAVegMod uses an increase in the relative amount of open water obtained from the wetland morphology model. LAVegMod takes this information and reduces the proportionate representation of non-aquatic species. The change in the relative amount of open water can also permit an increase in the area covered by SAV. Currently, changes in area of open water and wetland occur at a 25-year interval.

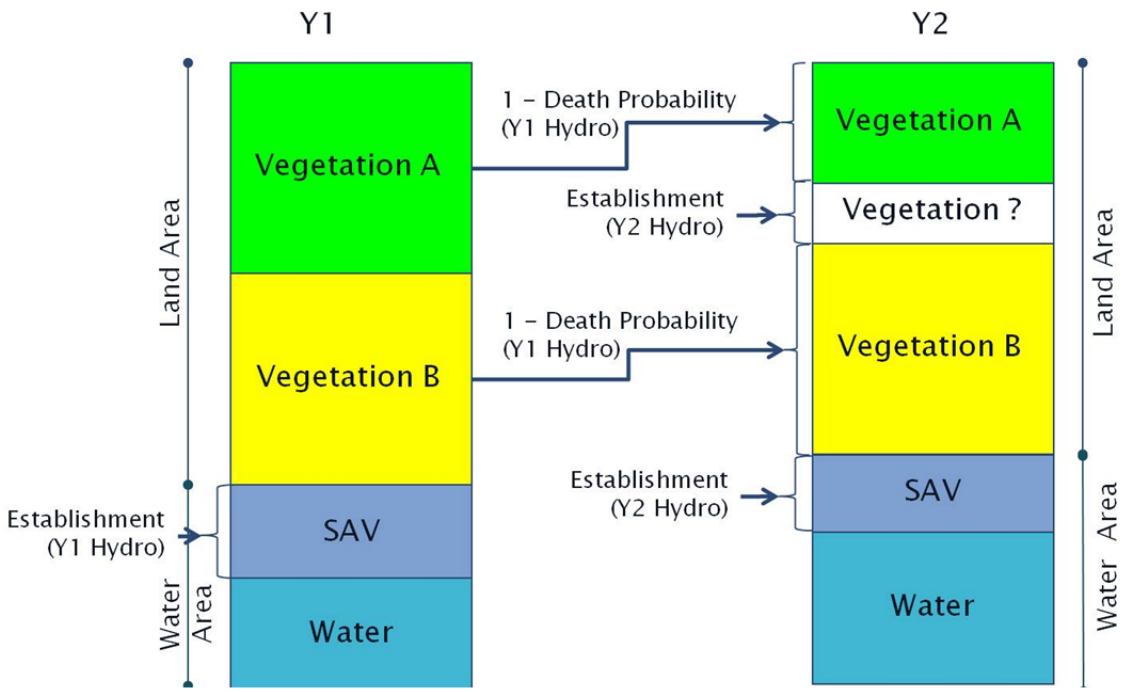

Figure 2: A summary of one time step in one cell of the vegetation model. Y1 is the first year representing either the initial condition or the result from a previous time step. Establishment of SAV in any year is based on average annual water depth of the cell, area of water in the cell, and salinity and temperature in the water that year in the eco-hydrobox in which the cell is located. To forecast emergent vegetation in the next year, the model determines the percentage of a vegetation class present in the cell that dies based on the water-level variability and average salinity in the previous year. Space that becomes available due to death of the previous year's vegetation can be occupied by a new vegetation class. Establishment of this new vegetation class is determined by the hydrologic conditions

The year to year processes illustrated by Figure 2 are repeated for each 500x500 meter cell in the model. Each 500x500 meter cell receives different inputs based on the spatial distribution of hydrology, salinity and water temperature provided as outputs from the Eco-Hydrology (Appendix D-1) and Wetland Morphology (Appendix D-2) models. Once all 500x500 meter cells have been updated, the model records the new spatial distribution of plant to a file and proceeds to the next year where the entire update process is applied again. The process continues until the model has been run for all 50 years.

c. Contribution to Planning Effort

LAVegMod provides an intermediate step in the overall collection of models (Figure 1). It provides the necessary input used by other models to forecast project effects on a suite of ecosystem services and also to set 'roughness' in the Storm Surge and Waves model (ADCIRC).

d. Description of Input Data

LAVegMod uses several distinct categories of input including those that characterize the initial conditions, data that represent the physical environmental processes along the LA coastline and inputs that govern how different plant species respond to changes in the environment. The

input variables for the model are Initial Vegetation Distribution (km^2), Land Area Distribution (km^2), Elevation (m), Water level or Stage Height (cm), Salinity (as parts per thousand, or ppt), and Temperature (°C). A short description of the source of these input data is provided below. In addition, LAVegMod requires several files that are used to define the spatial domain of the model and to uniquely identify each 500x500 meter cell. The latter is important to assure that different model inputs and outputs are coordinated within LAVegMod and also coordinated with the Eco-Hydrology and Wetland Morphology models as well as the other models used in the 2012 Coastal Master Plan analysis (Figure 1).

Initial conditions for the model are based on two data files that describe the spatial distribution of plant species at the start of a simulation and the distribution of land and open water (Figure 3). LAVegMod is expected to receive the initial spatial distribution of plant species in ASCII Grid Plus format. The ASCII Grid Plus format is described below. The distribution of land and open water is expected to be passed to the model as an ERDAS imagine file.

LAVegMod takes several input files that describe how hydrology, salinity and water temperature change over time. The hydrology, salinity and water temperature data are to be stored in separate files each following the Eco-Hydrology model data format standard. A brief description of this data format is provided below. The hydrology, salinity and water temperature data files are expected to be joined by ESRI shapefiles that provide GIS information defining the Eco-Hydrology boxes used by the Eco-Hydrology model.

The LAVegMod calculates the parameters needed to forecast mortality and/or establishment of the vegetation types. It uses the daily stage data to compute the annual standard deviation in stage height as an estimate of water level variability (WLV) for each 500x500 meter cell in the model. Hydrology data are required on a daily basis with values provided in meters (NAVD 88). The LAVegMod also uses the hydrology data in conjunction with the elevation data to compute the mean summer water depth for each 500x500 meter cell in the model. Summer means are computed based on hydrology from May until August of each year based on the requirements for the SAV submodel. Salinity data are required on a monthly basis with values provided in ppt. Temperature data are required on a monthly basis with values provided in degrees Celsius.

Finally, the LAVegMod requires data tables that describe how plant species respond to changes in the environmental conditions. A single table is currently used to describe the combination of conditions that lead to the establishment of each of the 19 vegetation types in response to the annual WLV and annual mean salinity (Figure 4). The probability that a plant species will senesce under the current combination of WLV and salinity is provided by a second set of tables. One table is provided for each of the 19 vegetation types (Attachment 1). One example of a mortality probability table is provided in Figure 5.

The mortality tables describe a three-dimensional surface where x- and y-axes define the WLV and the salinity and the z-axis gives the probability of a plant species dying. That is, the surface can be thought of as being defined as a function of two arguments $F(w,s)$ where w is the WLV, s is the salinity conditions and the value of $F(w,s)$ is the mortality rate of a plant under the conditions w and s.

The mortality functions, *F(w,s)* are defined by a table stored in comma separated value (CSV) format. The table defines the value of *F(w,s)* at a set of specific values of w and s. Values for *F(w,s)* are defined by linear interpolation between the defined values of *w* and *s.* A simplified example of the mortality function table is given in Figure 5.

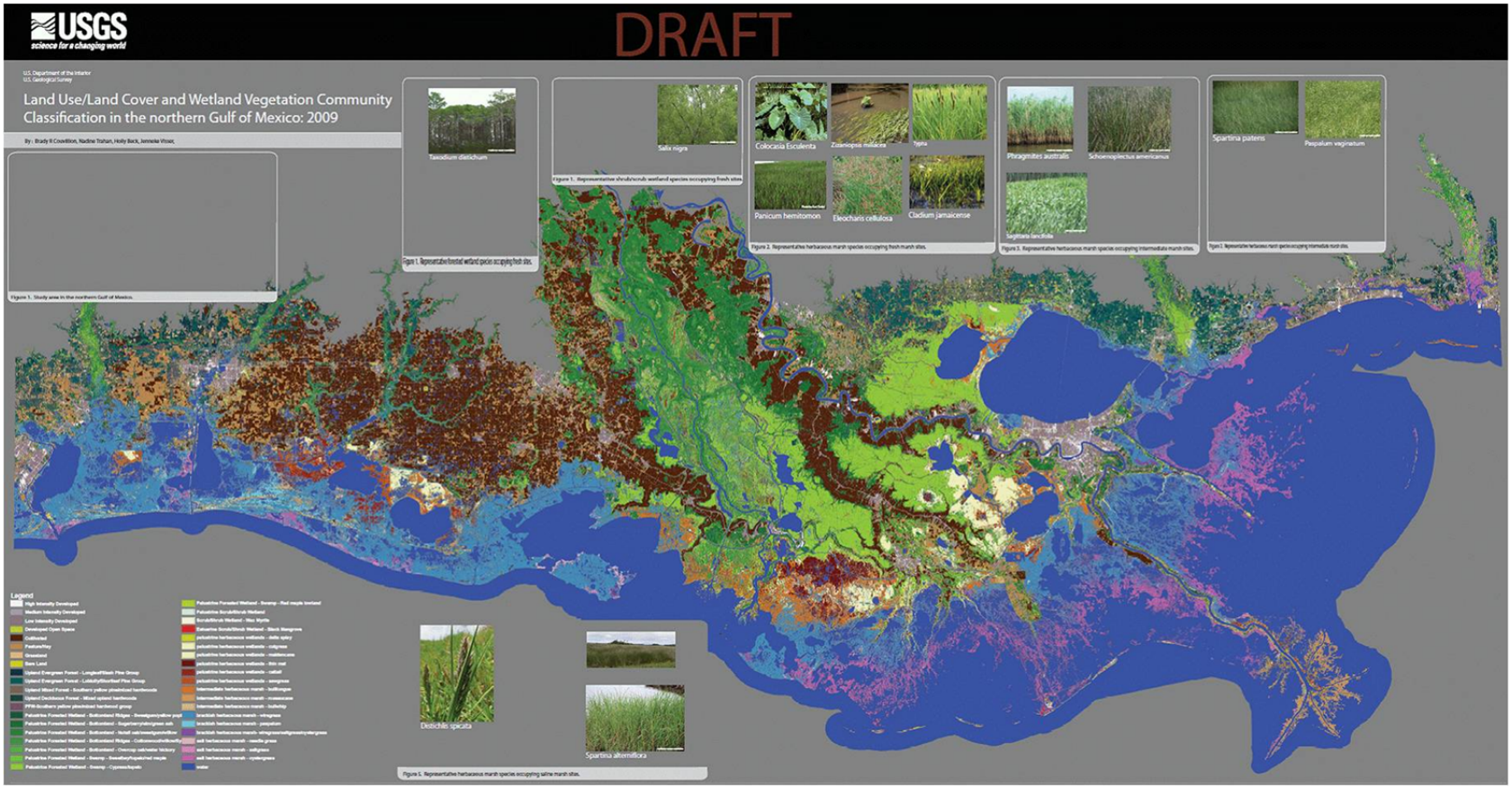

Figure 3. Vegetation map of coastal Louisiana in 2010 showing initial vegetation type distribution in the coastal zone.

Water Level Variability [Standard Deviation of Water Elevation (m)]

Average Annual Salinity (ppt)	0	0.03	0.06	0.09	0.12	0.15	0.16	0.17	0.175	0.18	0.191	0.2	0.206	0.21	0.24	0.27	0.3	0.34	0.37	0.4	0.43	0.46	0.49	0.52	0.55	0.58	0.61
0	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
0.2	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	WAXM	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
0.4	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	THIN	SWMP	SPLAY	CTGR	CTGR	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
0.6	THIN	THIN	THIN	THIN	THIN	THIN	THIN	CAT	CAT	CAT	SPLAY	CTGR	CTGR	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
0.8	THIN	THIN	CAT	CAT	CAT	CAT	CAT	CAT	CAT	CAT	CAT	CTGR	CTGR	SAWG	SAWG	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
1	CAT	CAT	CAT	CAT	CAT	CAT	CAT	CAT	CAT	CAT	BULL	BULL	BULL	SAWG	SAWG	SAWG	SAWG	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
1.2	CAT	CAT	CAT	CAT	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	SAWG	SAWG	SAWG	SAWG	SAWG	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
1.4	CAT	BULL	BULL	BULL	BULL	BULL	BULL	BULL	ROSE	ROSE	ROSE	BULL	BULL	SAWG	SAWG	SAWG	SAWG	SAWG	SAWG	MAID	MAID	MAID	MAID	MAID	MAID	MAID	MAID
1.6	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	ROSE	ROSE	ROSE	BULL	BULL	SAWG	SAWG	SAWG	SAWG	SAWG	SAWG	SAWG	MAID	MAID	MAID	MAID	MAID	MAID	MAID
1.8	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	ROSE	ROSE	ROSE	BULL	BULL	BULL	BULL	BULL	SAWG	SAWG	SAWG	SAWG	SAWG	MAID	MAID	MAID	MAID	MAID	MAID
2	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	WHIP	WHIP	WHIP	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL
3	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	WHIP	WHIP	WHIP	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL
4	BULL	BULL	BULL	BULL	BULL	BULL	BULL	BULL	WHIP	SCRB	WHIP	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE
5	PASP	PASP	PASP	PASP	PASP	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE
6	PASP	PASP	PASP	PASP	PASP	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE
7	PASP	PASP	PASP	PASP	PASP	WIRE	WIRE	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK
8	PASP	PASP	WIRE	WIRE	WIRE	WIRE	WIRE	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK
9	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	WIRE	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK	BRAK
10	SALT	SALT	SALT	SALT	SALT	SALT	SALT	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL	NEDL
12	SALT	SALT	SALT	SALT	SALT	SALT	SALT	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
14	SALT	SALT	SALT	SALT	SALT	SALT	SALT	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
16	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
18	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
20	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
22	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
24	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST	OYST
26	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR
28	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR
30	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR	MNGR

Figure 4: Establishment matrix for the different emergent vegetation types. Vegetation type abbreviations show the mean of the distribution for that vegetation type. Outlined boxes show the location of the median of that vegetation type's distribution. For details seesection 1.g.2. Abbreviations provided in Table 1.

Water Variability [Standard Deviation of Water Elevation (m)]

Average Annual Salinity (ppt)

THIN	0.00	0.03	0.06	0.09	0.12	0.15	0.18	0.21	0.24	0.27	0.30	0.34	0.37	0.40	0.43	0.46	0.49	0.52	0.55	0.58	0.61
0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
0.2	0.1	0.1	0.1	0.1	0.1	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
0.4	0.1	0.1	0.1	0.1	0.1	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
0.6	0.1	0.1	0.1	0.1	0.1	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
0.8	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
1	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
1.2	0.3	0.3	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
1.4	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
1.6	0.5	0.5	0.5	0.5	0.5	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1
1.8	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
2	0.6	0.6	0.6	0.6	0.6	0.5	0.5	0.5	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
3	0.7	0.7	0.7	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.8	0.9	1	1	1	1	1	1	1	1	1
4	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	0.7	0.9	0.9	1	1	1	1	1	1	1	1	1
5	0.9	0.9	0.9	0.9	0.9	0.8	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1
6	1	1	1	1	1	0.9	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 5: Mortality matrix for Thinmat (THIN) Vegetation Group. Orange cell shows the median condition of the vegetation type from the CRMS data analysis (see section 1.g.2), Yellow cells, show values from the 0.25th quantile to the 0.75th quantile of the vegetation type, and green cells show values of 0.05th quantile to the 0.95th quantile. These values were used to guide the mortality proportions depicted in the table. Tables for the other vegetation types are provided in Attachment 1.

Table 1. Emergent wetland vegetation types used in the LAVegMod .

Habitat Type	Vegetation Type	Characteristic Species	Abbreviation in Program	Abbreviation in Figure 2
Saline	1. Mangrove	*Avicennia germinans* (L.) L.	MANGR	MNGR
Saline	2. Oystergrass	*Spartina alterniflora* Loisel.	OYST	OYST
Saline	3. Saltgrass	*Distichlis spicata* (L.) Greene	SALT	SALT
Saline	4. Needlerush	*Juncus roemerianus* Scheele	NEEDL	NEDL
Brackish	5. Brackish Mixture	Mixture of: *Spartina patens* (Ait.) Muhl., *Distichlis spicata* (L.) Greene, *Spartina alterniflora* Loisel.	BRACK	BRAK
Brackish	6. Wiregrass	*Spartina patens* (Ait.) Muhl.	WIRE	WIRE
Brackish	7. Paspalum	*Paspalum vaginatum* Sw.	PASP	PASP
Intermediate	8. Bullwhip	*Schoenoplectus californicus* (C.A. Mey.) Palla	WHIP	WHIP
Intermediate	9. Roseaucane	*Phragmites australis* (Cav.) Trin. ex Steud.	ROSEAU	ROSE
Intermediate	10. Scrub-shrub	Mixture of :*Iva frutescens* L., *Baccharis halimifolia* L.	SCRUB	SCRB
Intermediate	11. Bulltongue	*Sagittaria lancifolia* L.	BULL	BULL
Swamp	12. Swamp Forest	*Taxodium distichum* (L.) L.C. Rich. and *Nyssa aquatica* L.	SWAMP	SWMP
Fresh	13. Delta Splay	Mixture of: *Sagittaria latifolia* Willd. *Schoenoplectus deltarum* (Schuyler) Sojak, *Colocasia esculenta* (L.) Schott	SPLAY	SPLAY
Fresh	14. Thinmat	Mixture of: *Eleocharis baldwinii* (Torr.) Chapm., *Hydrocotyle ranunculoides* L. f. *Bidens laevis* (L.) B.S.P.	THIN	THIN
Fresh	15. Maidencane	*Panicum hemitomon* J.A. Schultes	MAID	MAID
Fresh	16. Sawgrass	*Cladium mariscus* (L.) Pohl	SAWGR	SAWG
Fresh	17. Cattail	*Typha domingensis* Pers	CAT	CAT
Fresh	18. Cutgrass	*Zizaniopsis miliacea* (Michx.) Doell & Aschers.	CUTG	CTGR
Fresh	19. Waxmyrtle	*Morella cerifera* (L.) Small	WAXM	WAXM

d.1. *ASCII Grid Plus file format*

The ASCII Grid Plus formatted data files contains two blocks of data. The first block is a standard ASCII grid data structure containing a header of six metadata fields followed by a grid (matrix) of data values. The header provides the number of row and columns in the data (nrows, ncols), the geographic coordinates of the lower left corner of the data grid (xllcorner, yllcorner), the cell size of each block (cellsize) and the "no data value" (NODATA_value). Each row of the grid is stored on a separate line terminated with a new line character. Within each row values for each column are separated by one or more white spaces. The values in each cell are either the "no data value" (typically -

9999) or are the unique cell identification (ID) for the location represented by the cell. Cell IDs follow the Coastal Louisiana Ecosystem Assessment and Restoration (CLEAR) Grid cell ID standard (Twilley and Barras 2004).

The second block in the ASCII Grid Plus format contains data stored in comma separated values (CSV) format. This block starts on a new line immediately below the associated ASCII Grid block with no intervening characters. The first row in the CSV block contains the headers for the data stored in the table. Each subsequent row contains the vegetation cover data for a single grid cell. The first column contains the cell id and refers to the location with the matching cell id in the ASCII Grid. The following 22 columns represent the fraction of the 500x500m cell covered by various vegetation types. The vegetation types included in the model are shown in Table 1. An example of this data structure is included in Attachment 2.

Typically an ASCII Grid Plus file contains a single year's worth of data. However, to simplify file exchange an extended file format was implemented that allows several years of data to be stored in a single file. In multi-year ASCII Grid Plus files, each year's worth of data is separated by the string "# Year = <value>" where "<value>" is replaced by a number. Each separator string appears on a line by itself.

d.2. *Eco-Hydrology model file format*

The Eco-Hydrology model Team developed a file format for delivering spatial data where the spatial structure is defined by an ESRI shapefile. The format stores data as plain text and arranges data as tab-delimited values. The first row if the file contains the header names for the table. The first name is either "DAY" or "MONTH" indicating that the data are provided on a daily or monthly time step. The LAVegMod expects that hydrology is provided at daily intervals while salinity and water temperature are provided at monthly intervals. The balance of the header identifies which Eco-Hydrology box is represented by each column of data. The Eco-Hydrology box labels are composed to a two letter prefix followed by a three digit number. The two letter prefix takes one of the values AA, CP or PB, indicating that the column represents a Eco-Hydrology box for the Atchafalaya Area (AA), the Chenier Plain (CP) or the Pontchartrain -Barataria (BP) basin. Files from the Eco-Hydrology model do not mix hydrology regions, so in any single file all prefixes will be of one type (e.g. one of AA, CP or PB). The three digit code refers to one of the Eco-Hydrology model boxes. For box ids that are less than 100, the number is left padded with zeros. So box id 1 from the Atchafalaya area would have the label "AA001". The shapefile that is associated with each set of Eco-Hydrology model output is expected to have a field labeled "BoxID". Each element in the header row is separated by a single tab.

Each row of data in an Eco-Hydrology file represents data for a single day or month (depending on the file type). The first entry in each row is the day or month number. Each subsequent entry represents model predictions from the Eco-Hydrology model for each Eco-Hydrology box. The entries in a row are tab-separated.

d.3. *Basemap*

LAVegMod first input is an initial vegetation distribution map. This map was created by the U.S. Geological Survey – National Wetlands Research Center (Coastal Restoration

Assessment Branch) (Couvillion et al. in prep) as follows. A 2010 baseline land-cover dataset for the study area was created using 2007 coast wide vegetation survey (Sasser et al. 2008) as the training data. Survey points were first categorized into dominant vegetation types following methods similar to Visser et al. (2003), but expanded to the 18 herbaceous vegetation types used in this model (excluding swamp forest, which is not included in the coast wide survey). Remotely sensed imagery from 2007 and 2010 was then utilized, and change-vector analysis was employed to eliminate any training data points which appeared to have changed from 2007 to 2010. The 2010 imagery was then utilized to conduct a vegetation classification. Nationally recognized classification methodologies, such as those utilized in the U.S. Environmetal Protection Agency's National Land Cover Data (NLCD) and the National Oceanic and Atmospheric Administration Coastal Change Analysis Program (C-CAP), were altered slightly to provide increased value, yet retain consistency with these programs. Herbaceous wetland classes were dissected into 18 dominant vegetation types. Similarly, forested wetlands were classified into seven categories not included in previous datasets, including the swamp forest vegetation type used in the model. The Vegetation and Wetland Morphology modeling teams are currently working on a journal article that describes the creation of this map in more detail (Couvillion et al. in prep.).

Land area and elevation in each 500x500m cell are obtained from the Wetland and Barrier Shoreline Morphology models. Data from the two models are stitched together before being used by the Vegetation model. Morphology changes due to projects and scenarios are described in more detail in the Wetland Morphology appendix (D-2) and the Barrier Shoreline Morhology appendix (D-3).

e. Description of Output Data

The output from the LAVegMod projects changes in the distribution of each of the 19 vegetation types and SAV. An example of the output for one vegetation type in one output year is provided in Figure 6. The model provides estimates of change for the entire Louisiana coastline at a 500x500 meter resolution. For each 500x500 meter cell the output files store the fraction of the cell covered by each of the 19 vegetation types and SAV.

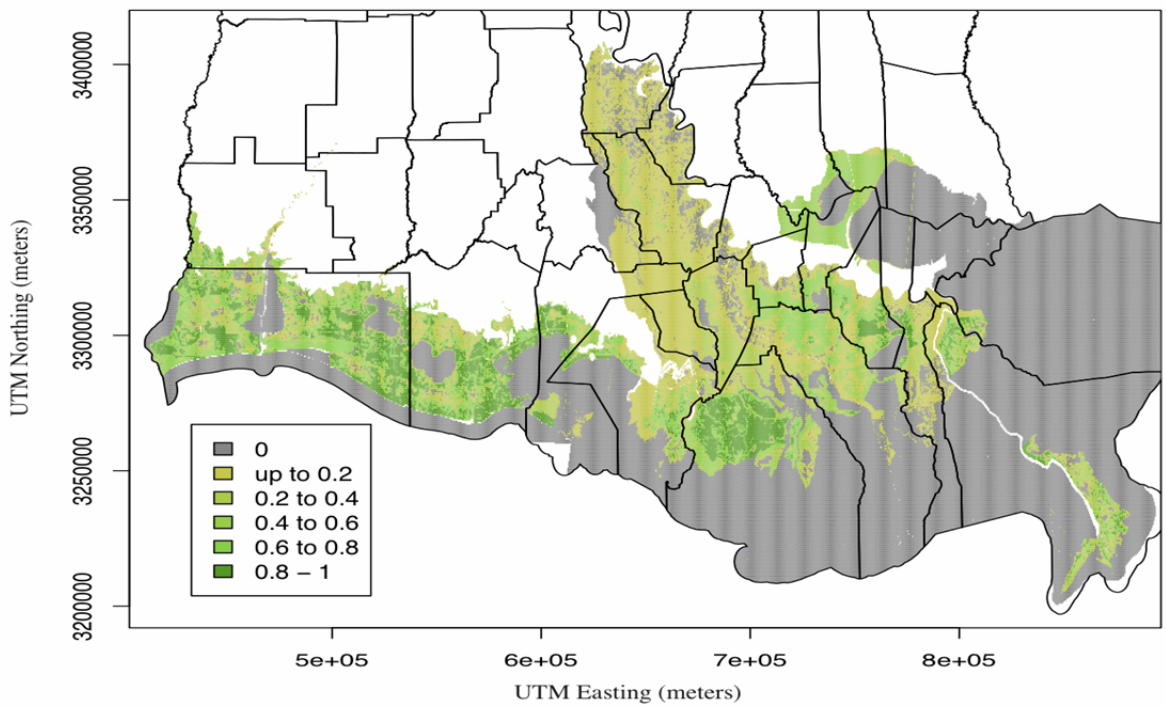

Figure 6. Distribution of the thin mat vegetation type after a 50 year simulation for the entire Louisiana coastline. Each pixel in the image represents a 0.25 km^2 plot. Each pixel is colored according to fraction of the area covered by the thin mat vegetation type. Cover is classified into six classes. Areas with no cover by thin mat are colored grey. The remaining 5 fractional cover classes are: 0 to 0.2, 0.2 to 0.4, 0.4 to 0.6, 0.6 to 0.8 and 0.8 to 1.0. Each interval excludes its lower bound but includes its upper bound. Colors ramp from yellow to green and correspond to cover classes as shown in the legend. Louisiana parish boundaries are shown in black. Spatial coordinates along the x- and y-axis are are Universal Transect Mercator (UTM) coordiantes for UTM zone 15 north.

The model produces output both annually and at five year intervals. Annual data are produced for each of the three individual Eco-Hydrology modeling regions (Atchafalaya - AA, Chenier Plain - CP and Pontchartrain Basin - PB). For each hydrology region, output is stored in two files. The first file contains output for the first 25 years of the simulation while the second file contains output for the second 25 years of the simulation. These data are stored in the file using the ASCII Grid Plus file format with multiple years of data stored in a single file.

Data created at five year intervals are produced for the entire Louisiana coast line by merging model output from the three Eco-Hydrology modeling regions. The data files for each five year interval contains the fraction of each 500x500 meter cell occupied by each of the 19 vegetation types included in the model.

f. Statement on the capabilities and limitations of the model

LAVegMod is capable of projecting changes in 19 wetland vegetation types and SAV cover in response to changes in average annual salinity (emergent vegetation), annual water-level variability (emergent vegetation), average salinity during the summer months (SAV), average summer water depth (SAV), and average summer temperature (SAV) along the Louisiana coast. The model currently does not include bottomland hardwood forest, dune vegetation, and

different upland land use or land cover, because these are outside the domain of the modeling team and not included in the CRMS sites. The model domain is limited to the Louisiana coast because all the data used to generate the parameters in the model were collected within this area. In addition, the hydrology and salinity input data are currently only available for the coastal region. The model could be expanded to other regions of the gulf coast with similar wetland vegetation if the necessary information to parameterize the model became available and hydrology and salinity inputs were created.

There are some limitations in the ability of the model to capture important processes, such as grazing, turbidity, and soil type. These limitations are the result of a lack of data needed to parameterize the response of plant species to these factors. Some processes are the same with or without projects such as hurricanes, frost, environmental disaster, and lightning strike fire. Some of the effects of hurricanes are captured in the Eco-Hydrology model and influence salinity and water level which in turn influence the change in vegetation cover. In addition, hurricane effects on land loss and barrier island configuration are captured in the Wetland and Barrier Shoreline Morphology models. Water temperature is forecasted by the Eco-Hydrology model and could be used as a proxy for frost; e.g., which would have an effect on the mangrove vegetation type.

Some vegetation classes are not resolved to the species level because they reflect species assemblages without a clear dominant. This increases the variability of the assemblage response to the driving factors. Future versions of the LAVegMod may include dividing these assemblages into their constituent species and simulating the individualistic response of these species. Currently the model does not include the effect of establishment based on the dispersal of propagules (seeds and fragments) from nearby cells. As a result, the model may lead to simulated replacement series that are not observed in the natural system, as well as establishment of vegetation types outside their current range (for example thin-mat marsh in the Chenier Plain region). Soil type and especially the floating nature of maidencane and thin-mat make it unlikely that they could be replaced by more salt tolerant species; the Vegetation model has currently no limitation on replacement of these vegetation types. However, the Wetland Morphology model includes land loss due to rapid salinity change in fresh and intermediate marshes.

g. Description of model development process including documentation on testing conducted (Alpha and Beta tests)

g.1. *Overview of Model Development Process*

The first stage of model development included the formulation of the overall goals of LAVegMod, a review of past and current approaches to modeling plant community dynamics and an assessment of the availability and quality of data to parameterize and calibrate LAVegMod.

The goal of LAVegMod is to provide an estimate of the changes in vegetation cover in response to changing environmental conditions in order to assess the potential effects of different restoration and protection projects and to provide input for higher trophic level models and storm surge models as well as provide feedback to the Wetland Morphology and Eco-Hydrology models used in the master plan modeling effort. To provide the most detailed estimate of vegetation change, we chose to model plants at

the level of individual species. This decision reflects the predominant and well supported theory of individualistic response by plant species to changes in their environment (Gleason 1926). This approach also provides the greatest degree of flexibility in the use of LAVegMod output by other models. In addition, most of the coastal vegetation communities in coastal Louisiana are dominated by one species (Table 1). We decided to simulate the dynamics at a 500x500 meter resolution as a compromise between providing the most detailed estimates of plant community change possible and the time it would take to run the model.

The next step was to develop the mathematical equations upon which the local plant community dynamics would be based. We adapted the equation developed by Pacala and Tilman (1994) to include the effects of establishment and senescence due to changes in the environmental conditions. The equation for the change in vegetation cover within a single 500x500 meter cell is:

$$\Delta C_i = -d_i(W_t, S_t)C_{i,t} + \left[\left(1 - \sum_j C_{j,t}\right) + \sum_j dC_{j,t}\right] P_i(W_t, S_t)$$

where species is indexed by *i* and *j* and *t* is time. C_i is the cover of species *i* within the cell and ΔC_i is the change in the cover of species *i* in response to changes in the environmental conditions. The environmental conditions are represented by W_t, the annual water level variability, and S_t the mean annual salinity. $d_i(W_t,S_t)$ is the fraction of cover lost by species *i* as a function of the environmental conditions. $P_i(W_t,S_t)$ is the probability that species *i* can become established in any unoccupied area as a function of the current water level variability and salinity. The first term on the right hand side of the equation computes the amount of area lost by species in the current time interval. The second term is the amount of area gained by species *i* during the current time interval. The gain in area is computed as the product of the total area that is unoccupied and can be occupied by species *i*. The area that is unoccupied is computed as the total fraction of the cell that is unoccupied at the start of the time interval plus any additional area that becomes available because of the mortality of the plant species.

Next, we designed a conceptual framework to guide the development of the model into a computer program, In our conceptual design we identified the modeling elements that made up our model and we designed C++ classes to represent these elements. The top level modeling elements include a modeling class (class Model) that is responsible for coordinating the actions of all the other model components as well as updating the model dynamics from one time step to the next and cycling over the individual spatial cells in the model during each time step. The other top level class (class Param) handles all of the input of all the model parameter values and data. This class makes use of a number of subordinate classes to process input files and store structured data, such as spatial data, internally.

As each component of the model code was created, it was tested. The code was tested to make sure it performed the required task and that there were no memory leaks.

While we were developing the model code, we inserted code comments that described the sequence of steps taken by each function or class. That is, we have inserted prose

into the model source code files that describe what the code is designed to do in terms that do not require the reader to be familiar with the programming language. These comments are ignored by the computer and have no effect on the performance of the model. The commenting is currently limited to the portions of the model code directly involved in simulating the change in vegetation cover from one time step to the next.

The model source code also perform many common "house-keeping" tasks needed by any computer program. These house-keeping tasks include actions such as allocating and deallocating computer memory (e.g. RAM) use by the model while computing the changes in vegetation types or reading data from and writing data to files located on the computer's hard drive. Such tasks are not directly involved in the simulation performed by the model, but they are necessary to make the model run quickly and correctly and to record model output after the simulation is finished. Because of time constraints, much of the house keeping code in not current thoroughly documented using in-code comments. Future version of the model code will provide in-code documentation using comments to describe the house-keeping operations.

The decision to use the open source modeling language R (R Development Core Team 2010) was made to allow for rapid code development in the file format conversion steps of the model to accommodate the emerging file formats used by the models that provide input data for the LAVegMod.

g.2. *Data Analysis in Support of Emergent Vegetation Algorithm*

Preparation of hydrologic data

Raw hydrologic data were downloaded from the CRMS database in August 2010. We collected all available hourly salinity and hourly water level data for every CRMS station in Water Years 2007, 2008, and 2009. Data were available for 2006, but not all stations were online at that time and the dataset for that year was incomplete. A water year includes the dates between October 1 and the following September 30 (i.e. WY2007 = 10/1/2006 – 9/30/2007).

We then computed the annual mean, 90th percentile, and standard deviation for each year of interest at each available CRMS location using the following hourly data records: 1. salinity (ppt), 2. water level relative to average marsh elevation (feet). The standard deviation of water level relative to the marsh surface is an estimate of water level variability and thus a proxy for the volume of water exchanged between the interior marsh and adjacent water bodies. We use the term *Water Variability Index* throughout this report to refer to the standard deviation of water level relative to the marsh surface.

Preparation of vegetation survey data

Annual vegetation survey data for every available CRMS station were downloaded from the CRMS database for vegetation sampling years 2007, 2008, and 2009. These survey records included the annual estimates of percent cover for each species present at a station (the percentage of total area occupied by a species' canopy). Percent cover here is an ocular estimate of vegetation cover by species conducted in 10 or more (9 in swamps) 2m x 2m or $4m^2$ plots per site. For a detailed description of CRMS vegetation

sampling techniques, see Folse et al. (2008). Vegetation surveys are regularly conducted between July and October for a given year.

The plot-level percent cover data for each species were then averaged by site. In the CRMS dataset, however, only observed species and cover values were recorded. A total of 5-10 plots per site were sampled, and the total area of all sampled plots must be taken into account when averaging the data by site. So, the estimated percent cover value for each species present at a site was computed as follows:

$$PercentCover_{ab} = \frac{\sum_{i=1}^{z} PercentCover_i}{z}$$

where:

a = individual species

b = individual station

i = plot number

Z = number of plots per site

A subset of the data table was then produced that included only the species of interest for this study. We created *Dominant Vegetation Groups* based on the vegetation types used in the final model (Table 1). Most dominant vegetation groups consist of only one species. The total percent cover value for a dominant vegetation group with more than one species is equal to the summation of the percent cover values of the individual species in that group. Data records with no vegetation cover in the Dominant Vegetation Groups were discarded.

The final dataset of vegetation included the percent cover values for the dominant vegetation groups at each available CRMS station in the survey years 2007, 2008, and 2009.

Merging the hydrologic and vegetation records

Finally, the hydrologic and vegetation records were merged by station. The resulting table included the following parameters for each available CRMS station in the years 2007, 2008, and 2009: mean annual salinity, standard deviation of water level relative to marsh surface, percent cover for each present Dominant Vegetation Group. Only those records with Vegetation and Salinity and Water Level data were retained. Vegetation survey data collected between July and October of a given year are therefore concatenated with hydrology records from October of the prior year through October of the given year. This method ensures that the hydrology of a growing season is identified with the cover estimates at the end of the growing season for analysis.

Calculation of summary statistics

Summary statistics generated for use in model development include grand median, P5, P25, P75, and P90 values for annually-averaged salinity and annually-averaged Water Variability Index [i.e., standard deviation of water level (relative to marsh surface)] for each Dominant Vegetation Group. These summary statistics for each Dominant

Vegetation Group were weighted by the percent cover values at the station of record. That is to say that the salinity and water level records at a station where the percent cover is high for a given Dominant Vegetation Group, will have more influence on the final statistic. The resulting mean and standard error for each vegetation type are provided in Figure 7.

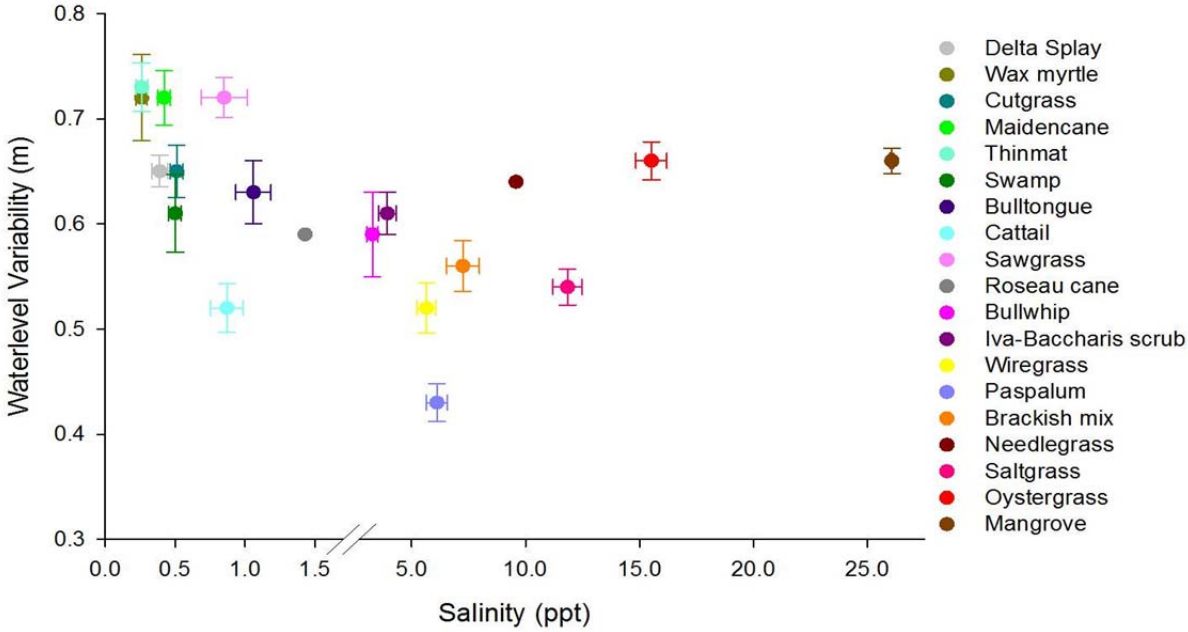

Figure 7. Distribution of occurrence of vegetation types relative to average annual salinity and water level variability. Bars represent one standard error. This data was used to inform the tables in Attachment 1 and Figure 2.

g.3. *Data Analysis in Support of Submerged Aquatic Vegetation Algorithm*

The goal of the submerged aquatic vegetation (SAV) submodel is to predict the annual localized presence of SAV in response to environmental forcing variables. To develop this submodel, we used Jacoby Carter's unpublished data collected from November 1999 through October 2002 from Sabine National Wildlife Refuge. Over the three-year study 498 sites were sampled, 234 sites with SAV and 264 without SAV. This data represents 498 samples with salinity ranging from 0 to 39 ppt. Data collected included: notation of SAV presence/absence, name of various SAV species (if present), date, time, latitude and longitude, temperature, salinity, conductivity, dissolved oxygen (concentration and percent saturation), water depth, pH, turbidity, and substrate characterization. Data were analyzed using logistic multiple linear regression (SAS V9.2).

It was determined through this three-year study of SAV at the Sabine National Wildlife Refuge that presence of SAV was seasonal and seasonal peaks varied somewhat from year to year. In general, however, there was a large peak in occurrence in spring

through summer, followed by a drop in the fall, and then by a smaller peak in December. Since LAVegMod predicts the presence of SAV on an annual basis, we focused the analysis on the period with the highest likelyhood of SAV occurrence during the summer months. In order to model the annual presence of all SAV in any given location in response to environmental conditions we used environmental data from no-SAV plots during the peak SAV months (April to August) to estimate the annual probability of finding SAV (Table 5). The inverse of probability of finding no-SAV in a given location based on environmental data gives us the probability of finding SAV. Not all variables that were strongly correlated with SAV presence were used. For example, dissolved oxygen and pH showed a strong correlation with the presence of SAV but were not used as predictive variables since both are strongly affected by photosynthesis and thus are a consequence not a cause of SAV presence. Turbidity significantly improved the fit of the logistic regression model. However, the eco-hydrology model does not forecast turbidity. We researched the possibility to use the suspended sediment and chlorophyll a forecasts to derive turbidity, but could not find a satisfactory method to do so. Therefore, the logistic regression equation without turbidity was used to estimate SAV presence.

Table 5. Predicting No-SAV (absence) with and without turbidity.
Turbidity-T, Salinity – S, Depth – D, Temperature-C

With Turbidity					**Without Turbidity**				
Prob(SAV) = 1- (-0.503T +0.0252***D** +0.0969***S** +0.0227***C** -0.6772)					**Prob(SAV)** = 1-(0.026***D** +0.0781***S** -0.0375***C** - 0.8789)				
Variable		**Estimate**		**Chi Square**	**Variable**		**Estimate**		**Chi Square**
Intercept		-0.6772		0.3305	Intercept		-0.8789		0.0734
Temp		0.0227		0.3592	Temp		-0.0375		0.0204
Salinity		0.0969		0.0002	Salinity		0.0781		<0.0001
Depth		0.0252		0.0056	Depth		0.0260		<0.0001
Turbidity		-0.5037		<0.0001					
Correct		**Incorrect**			**Correct**		**Incorrect**		
Event	**Non-Event**	**Event**	**Non-Event**	**%Correct**	**Event**	**Non-Event**	**Event**	**Non-Event**	**%Correct**
155	149	25	7	90.5	90	132	42	72	66.1

2. Technical Quality

a. Theory

LAVegMod is based on the niche theory of the ecology of species and the individualistic theory of plant community succession. The niche is a well-founded theory in ecology and provides a useful framework for understanding the distribution and abundance of species, how they coexist and how their distribution is expected to change in response to changes in the environment. The theory considers that the environment of a species or community is composed of a number of distinct factors that can be represented as independent axes in a multidimensional space. Each species that occupies a partial location in the environmental space based on the range of conditions where it is able to grow and reproduce. The coexistence and relative abundance of multiple species in an area can be understood as the set of species whose

niches overlap with the current state of the environment. Likewise, changes in the identity of species that make up a community can be understood as changes that result in a decrease in the overlap between the niche of some species and the environment and an increase in the overlap for others.

The individualistic theory of plant community succession was first introduced by Gleason (1926). This theory holds that the distribution of each species responds to its environment independent of the other species in the community. In particular the Gleasonian view holds that plants of different species do not cooperate to produce or preserve a particular community type, as was advocated by Frederic Clements (Clements 1936). Instead the communities are the consequence of each species responding to environmental changes to maximize its own fitness without reference to the fitness of other species. The Gleasonian view is by far the best supported of the two theories of succession on both ecological and evolutionary grounds. Our model embraces the Gleasonian view by modeling each species individually based on the environmental conditions that lead to their establishment, persistence and senescence.

Description of system being represented by the model

The system being represented by LAVegMod is the wetland vegetation and submerged aquatic vegetation in the Louisiana coastal zone. The emergent wetlands have been traditionally classified into five habitat types (Chabreck et al. 1968, Chabreck 1972): swamp, fresh marsh, intermediate marsh, brackish marsh, saline marsh. However, many different vegetation types that have different ability to support wildlife and different roughness (biomass and height) with respect to storm surge occur within these marsh zones (Table 6). Most of the herbaceous vegetation types have been described in detail by Visser et al. (1998, 2000, 2002). Vegetation types for coastal Louisiana not described by Visser et al. (1998, 2000, 2002) include Cattail, Sawgrass, Needlerush, Waxmyrtle, and Scrub-shrub. Each of the aforementioned vegetation types cover more than 1% of the coastal area (Figure 8). Shrub vegetation (Waxmyrtle or Scrub-shrub) was excluded from the previous analyses of the herbaceous wetlands, but always has been part of the Louisiana coastal vegetation. Sawgrass dominated areas have been described as dominating large areas of the Chenier Plain by O'Neill (1949). Needlerush marshes occur more extensively in the Breton Sound area, which was not included in previous analyses. Cattail marshes seem to be expanding in the Louisiana coast (not described for the 1968 through 2001 period), although Penfound and Hathaway (1938) describe fresh marshes dominated by cattails in Barataria Basin. Submerged aquatic vegetation in coastal Louisiana consists of a variety of native and introduced species (Merino et al 2009).

Table 6. Difference in height and biomass among the different emergent vegetation types

Vegetation Type	Vegetation Height[1] (cm)	Standing Crop (g/m^2)	Source for Standing Crop data
1. Mangrove	123	1500	Mitsch and Gosselink 2000
2. Oystergrass	91	739	DeLaune et al. 1979
3. Saltgrass	78	700	Hopkinson et al. 1978
4. Needlerush	132	883	Schmalzer et al. 1991
5. Brackish Mixture	93	1000	Burdick et al. 1989
6. Wiregrass	109	1100	Hopkinson et al. 1978
7. Paspalum	197	440	Garcia et al. 1993
8. Bullwhip	197	662	Pratolongo et al. 2008
9. Roseaucane	365	950	Hopkinson et al. 1978
10. Scrub-shrub	163	2000	Elsey-Quirk et al. 2011
11. Bulltongue	105	600	Hopkinson et al. 1978
12. Swamp Forest	916	37500	Mitsch et al. 2009
13. Delta Splay	118	432	Mitsch et al. 2009
14. Thinmat	70	130	Mitsch et al. 2009
15. Maidencane	121	1160	Mitsch et al. 2009
16. Sawgrass	201	517	Mitsch et al. 2009
17. Cattail	173	1215	Mitsch et al. 2009
18. Cutgrass	192	1039	Mitsch et al. 2009
19. Waxmyrtle	265	800	Sah et al. 2004

[1]Calculated from 2006 to 2009 CRMS data.

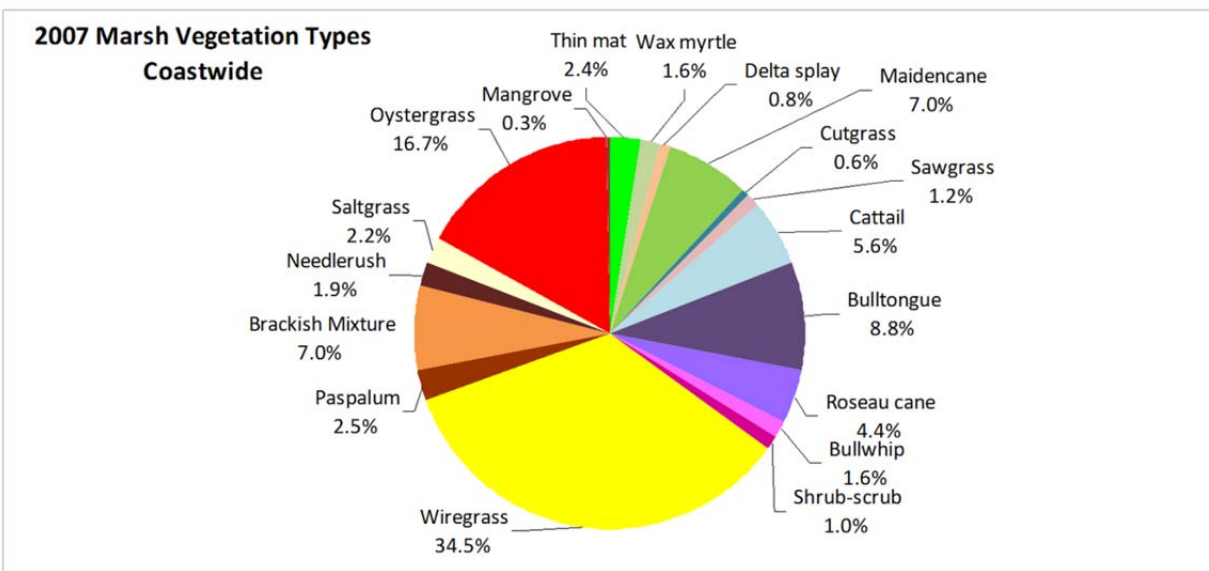

Figure 8. Percentage of stations surveyed in the 2007 coast wide marsh survey that were classified into each vegetation type. This survey did not include the forested wetlands in the coastal zone.

Vegetation change can occur rapidly along the coast (Visser et al. 1999, 2002). Although there are many factors that affect vegetation change in coastal wetlands only a few are generally affected by coastal restoration projects and are therefore the focus of this vegetation model. The few factors that are included in this model are water salinity and water level fluctuation. Grazing is a factor that is affected by management but not included in the current model due to the time constraints in the development phase. For example, the Coastwide Nutria Control Program has significantly reduced eat-out areas along the coast (Jordan and Mouton 2011). Nutrient input and eutrophication are known to affect species composition in wetlands (McJannet et al. 1995, Childers et al. 2003), however we have no available data that can be used to develop a relationship between vegetation change with respect to nutrients in coastal Louisiana. The recent apparent increase Cattail marsh may be an early indicator of increased nutrient loading, but also could be related to the higher salinity tolerance of *Typha domingensis* relative to the fresh marsh species it is replacing. Fire may affect species composition but is not currently used for marsh restoration.

b. **Analytical requirements**

Death and establishment parameters for the different vegetation types were derived from the hydrologic conditions under which these vegetation types occur in coastal Louisiana. Three years (October 1, 2006-September 30, 2009) of Coastwide Reference Monitoring System data on vegetation and hydrology were analyzed to elucidate the relationships (Figure 4). Mortality of a given vegetation class was assumed to be zero if hydrologic conditions were between the 25th and 75th percentile of the observed values, and increased to 50% from the 25th to the 5th as well as from the 75th to the 95th percentile, grading to 100% based on the slope outside of the known distribution of the vegetation type. For establishment probability a matrix of possible salinity (0 to 30 ppt) and water level variability (0-2) was populated with the different vegetation types. The median of the known distribution of the vegetation types was assigned to establishment of that vegetation type. Empty cells were proportionally assigned by the relative frequency at which the vegetation class occurred in 2007 (e.g. Maidencane occupied approximately 36% of the fresh areas of the coast and was assigned approximately 36% of the matrix below 2ppt) around the position of median of its distribution (Figure 7).

c. Assumptions

- Distribution of vegetation types reflects hydrologic variables:
  - Vegetation type dies when hydrology is outside its range.
  - Vegetation type establishes when hydrology is within the establishment range and land is available

d. Identification of formulas used in the model and proof that the computations are appropriate and done correctly

The LAVegMod dynamics are based on two equations. The first is for projecting the cover of SAV, the second is used to project the cover of the remaining 19 emergent wetland vegetation types included in the model. The SAV equation is based on a linear regression and is:

$$Cover = C_0 + b_{temp}*T + b_{salinity}*S + b_{depth} * D$$

where *Cover* is the fraction of the area in open water covered by SAV, C_0 is the intercept of the regression, b_{temp} is the rate of change in cover associated with a change mean summer water temperature, T, $b_{salinty}$ is the rate of change in cover with a change in mean summer salinity, S

and b_{depth} is the rate of change in cover with a change in the mean summer water depth, *D*. This equation is appropriate because it is currently the best empirical estimate for SAV cover in response to changing environmental conditions (see section 1.g.3)

This equation was incorporated into the LAVegMod as a distinct class (class SAV). We produced an instantiation of the class and examined the results of the computation over a biologically reasonable range of the input variables: temperatures, salinity values and water depths. We recorded the model responses in a CSV file and analyzed the relationship between the model generated values of *Cover* and the input variables using linear regression in the the statistical package R. We compared the values for b_{temp}, $b_{salinity}$ and b_{depth} parameters used in our model to the slopes obtained from the linear regression of cover on temperature, salinity, and water depth. We performed this test for several different values of each of the model parameters. The comparisons showed that the parameters estimated from the linear regression were the same as the parameter values used in our model for all combinations of parameters values.

The equation used for the other vegetation types used in our model is:

$$\Delta C_i = -d_i(W_t, S_t)C_{i,t} + \left[\left(1 - \sum_j C_{j,t}\right) + \sum_j dC_{j,t}\right] P_i(W_t, S_t)$$

where species is indexed by *i* and *j* and *t* is time. C_i is the cover of species *i* within the cell and ΔC_i is the change in the cover of species *i* in response to changes in the environmental conditions. The environmental conditions are represented by W_t, the annual water-level variation and S_t the mean annual salinity. $d_i(W_t,S_t)$ is the fraction of cover lost by species *i* as a function of the environmental conditions. $P_i(W_t,S_t)$ is the probability that species *i* can become established in any unoccupied area as a function of the current water-level variation and salinity.

This equation is appropriate because it includes two of the major processes involved in plant community succession, establishment and senescence, in response to environmental conditions that were revealed as being important in our analysis of the CRMS data.

We added this equation to our model by first incorporating it into a C++ class, class Patch in the function Patch::step(). Each instantiation of Patch represents a single 500x500 meter cell on the landscape. To test that the equation was correctly translated into C++, we ran obtained output from the class over a biologically realistic range of inputs of water level variation and salinity. The procedure we followed was to initialize the patch with a random configuration of the 19 species types. We then simulated the dynamics for a 20 year period using fixed levels of water level variation and salinity that were chosen to result in the eventual dominance of one species. That is we used water level variation and salinity levels that were appropriate for only a single species. We they plotted the change in cover of all the plants over time to assure that the change in each species from one time step to the next was consistent with the intended function of the model. We repeated this processes, setting the environmental conditions to a level that would result in the eventual dominance of each species. In all cases the model produced the predicted outcome.

3. System Quality

a. Description and rationale for selection of supporting software tool/programming language and hardware platform

- The Vegetation Model uses C++. The reason for using this programming language is simply because it is what we know and is sufficient and fast enough for what we are doing.
- We also use the programming language R because it allows for rapid prototyping. At times when people from other aspects of the model make rapid changes on us R helps us to adapt and accommodate the whole system. R can reformat data from varying programming languages into something that the original model can use.
- Because of scheduling issues the model actually consists of 3 different languages including C++, R, and SH.
- It is important to note that the model can be run without the shell script (and technically the R script). It is just somewhat difficult.

b. Proof that the programming was done correctly

The algorithms were extensively checked by generating maps of the vegetation type changes at 5 year intervals. These maps were checked for obvious errors in vegetation type distributions. For more details see section 2.f. above.

c. Availability of software and hardware required by model

LAVegMod is programmed using three languages, C++, R and SH. The core model that actually performs the simulation is coded in C++. The code was developed on Apple OSX 10.6 using the GNU C++ compiler version 4.2.1. The C++ code requires that the Boost C++ library version 1.40.0 and the GSL library 1.13. There is a makefile included with the program source code that coordinates the compilation of the model. The makefile should be modified to reflect point to the installation directory of the Boost and GSL libraries.

The code is also known to compile and run correctly under Linux assuming the appropriate compiler version and the appropriate library versions have been installed.

The C++ model is called by R using an R script named lavegmod.R. This script handles the automatic downloading of hydrology, salinity, elevation and land/water data from the master plan modeling website. This script also handles the conversion of the model input data from the various file formats used by the other models (i.e. eco-hydrology and wetland morphology models) into the format expected by the LAVegMod (i.e. the ASCII Grid Plus format). Once the data files have been reformatted the R script runs the LAVegMod.

In order for the R script to be invoked the R program must be installed. The script was designed to run on R version 2.11.1. However, any subsequent version of R should also work. In addition, the model comes with a library of R functions written to provide the data format conversion services needed by the model. These are located in the file myGISlib.R and are stored with the other code elements. The following R libraries also need to me installed: *shapefiles* version 0.6, *sp* version 0.9-65, *rgdal* version 0.6-26. These R libraries are required to read, write and manipulate the various file formats used by other models in the master plan modeling effort including ESRI shapefiles, ERDAS raster files and D-Base files.

Finally, the R script is called from a shell script named lavegmod.sh. This script is provided to make calling the R script more convenient since there are a number of arguments that need to be passed to the R script.

Summary of software requirements

- Apple OSX or BSD Unix of Linux
- GNU C++, version 4.2.1
- Boost C++ library version 1.40.0
- GSL library version 1.13
- GEOS library version 3.3.1-CAPI-1.7.1
- R version 2.11.1
- R library shapefile version 0.6
- R library sp version 0.9-65
- R library rgdal version 0.6-26
- R library rgeos version 0.1-15

The model is currently run on an Apple Mac Pro with 12 Gigabytes of ram and 2 4-core CPUs.

d. Description of process used to test and validate model

LAVegMod has been tested based on an evaluation of model predictions by our internal review team, which is composed of Rebecca Howard, Charles Sasser, Ken Krauss, and Mark Hester. The members of our review team are all active scientists with extensive experience studying the particulars of the wetland plant communities of coastal Louisiana. The review was performed in two phases. In the first phase we presented the model structure, assumptions, capabilities and limitations to our review team members for comment. They provided us with input as to the suitability of our niche-based approach for modeling plant response to changing environmental conditions and the calibration of the model using the CRMS.

At the first review the team members requested an additional analysis of the CRMS data to check for the relationship between hydrology and the presence or absence of different plant species. We carried out this reanalysis, but we ran into difficulties. The first is that there are considerable gaps in the record of water depths for many of the CRMS stations. As a result we were not able to obtain hydroperiod estimates for most of the stations. Second, different plant species were not distinguishable on the basis of hydroperiod.

In the second phase of our model validation, we presented the changes in plant communities predicted by our model produced under hydrology and salinity data from the eco-hydrology model (Figure 9). We chose the output from the eco-hydrology model that represents the future hydrology and salinity based on the assumption that no additions would be made to the existing water control structures in south Louisiana and the management processes will follow currently established guidelines. The review team evaluated output from our model based on the dynamics from 9 locations, three located in the Chenier Plan, three in the Atchafalaya River and three from the Lake Pontchartrain area. In each area, we chose a EcoHydro Box from the upper estuary, middle estuary, and lower estuary. In this way the review team was able to evaluate the model's performance in the three major regions that make up the Louisiana coastline representing a wide variety of vegetation types that characterize Louisiana's coastal wetlands.

One of the results from this review is that the waterlevel variability forecasted was significantly lower than the measured variability, and the eco-hydrology model was adjusted. An example of the type of visualization of the data used for this type of caibration is provided in Figure 9.

Although the team made comments on ways to improve the model, these recommendations were made under the assumption that the eco-hydrology models would be unchanged. However, during the workgroup leader call following this review meeting the eco-hydrology modelers announced that the underestimation of WLV was due to using one monthly value of water level as a boundary condition, which was being corrected by having daily fluctuating boundary conditions. Calibration of LAVegMod is scheduled after eco-hydrology has been changed, but may require contract extension and additional funding.

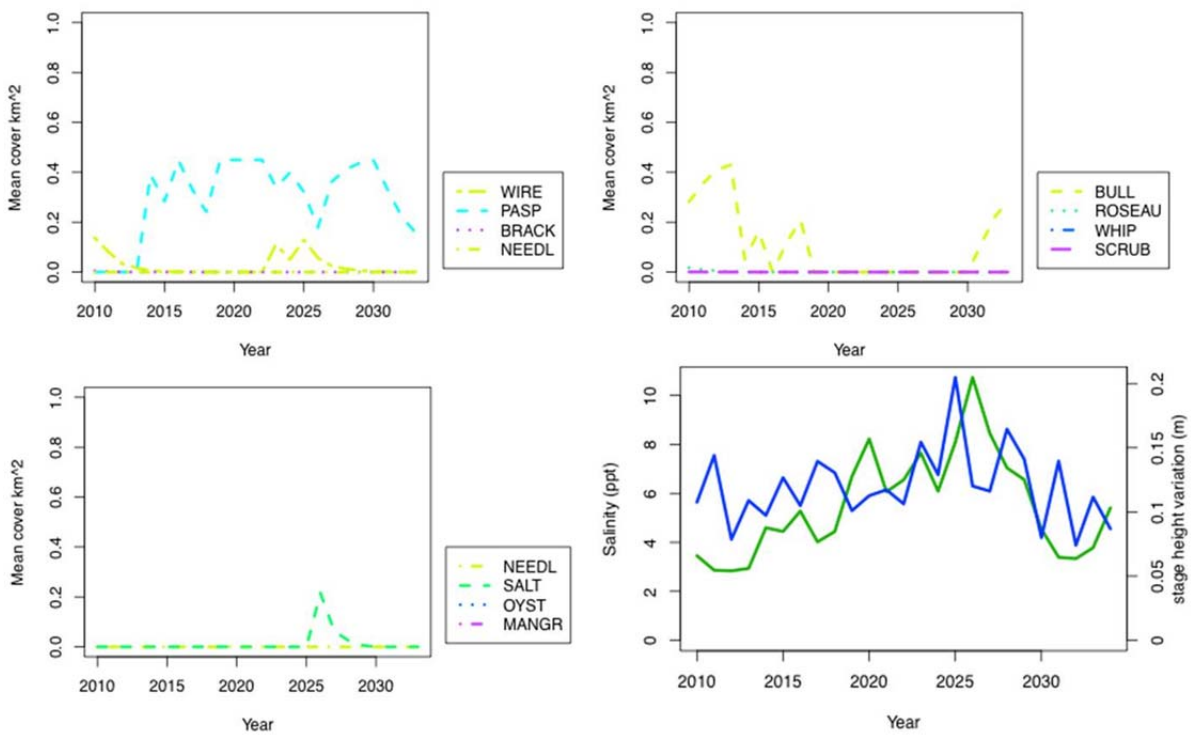

Figure 9. Changes in vegetation in a middle estuary box, using the future without action and moderate climate change scenario. The lower right hand panel shows the changes in WLV in blue and salinity in green. The upper left box shows changes in brackish vegetation types, the upper right box shows changes in intermediate vegetation types, and the lower left box represents changes in saline vegetation types.

e. Discussion of the ability to import data into other software analysis tools (interoperability issue)

The output format produced by the LAVegMod can be read into R using the functions provided in myGISlib.R. There are a suite of functions provided in myGISlib.R that can convert LAVegMod output into standard ASCII Grid files that can be read into any GIS application. There are further options using additional R libraries.

4. Usability

a. Availability of input data necessary to support the model

This model is designed to work in concert with the eco-hydrology and wetland and barrier shoreline morphodynamic models, which provide all of the input data other than the initial vegetation distribution. However, other models that generate forecasts of changes in the input parameters could be used as long as input data are provide in the format used by the model.

***b.* Formatting of output in an understandable manner**

The output of the model is all stored as standard ASCII text files. As a result, these datafiles can be opened and read in any text program and a user can quickly become sufficiently familiar with the data to manipulate it. While the data storage format used by the LAVegMod is novel, its structure is composed of two well known file fomats: ASCII Grid and comma separated values (csv).

c. Usefulness of results to support project analysis

Our vegetation model supports project analysis in two ways. First, the model provides a direct assessment of the changes in plant community structure in response to the restoration and projection projects proposed as part of the 2012 Coastal Master Plan. The distribution of plant species define the habitats that give Louisiana's coastal wetland ecosystem their unique character as well as providing the habitat that is critical for the survival of the animals that make these wetlands their home.

The proposed restoration and projection projects involve an alteration of freshwater flows across Louisiana's wetland ecosystem. The modified hydrology may affect spatial and temporal patterns of salinity across the landscape. Salinity gradients are one of the primary determinates of plant species distribution along the Louisiana coastline. Our vegetation model provides a spatially explicit, whole landscape estimate of how altered salinity patterns will change the distribution of plants. Plant communities may also respond to modification of the cyclical pattern of water depths and periods of inundation associated with restoration efforts. Our vegetation model includes the effects of water level variability on the distribution of plant species as well as the interaction of these effects with those associated with changes in salinity.

Second, the model provides vegetation change projections as input to the other models, such as the eco-hydrology model and the Wetland and Barrier Shoreline Morphology Models. The integration of vegetation model outputs into the other models allows the overall modeling effort to reflect the connections between abiotic, processes such as hydrology and soil morphology dynamics, and the ecology of Louisiana's coast wetland ecosystems.

d. Ability to export results into project reports

Our vegetation model provides a number of outputs that can be exported into project reports for the 2012 Coastal Master Plan. These outputs include whole coastline maps of the distribution and abundance of the 19 plant species and communities included in our model (Table 6). An example map is shown in Figure 6. This vegetation model can also be used to create time series graphs that show the change in plant species abundance (Figure 9). Because each cell can contain multiple vegetation types, we can also present changes as stacked bars graphs that summarize the extend of each vegetation type in the project area with and without the project implementation under multiple scenarios. An example of such a graph is provided in Figure 10.

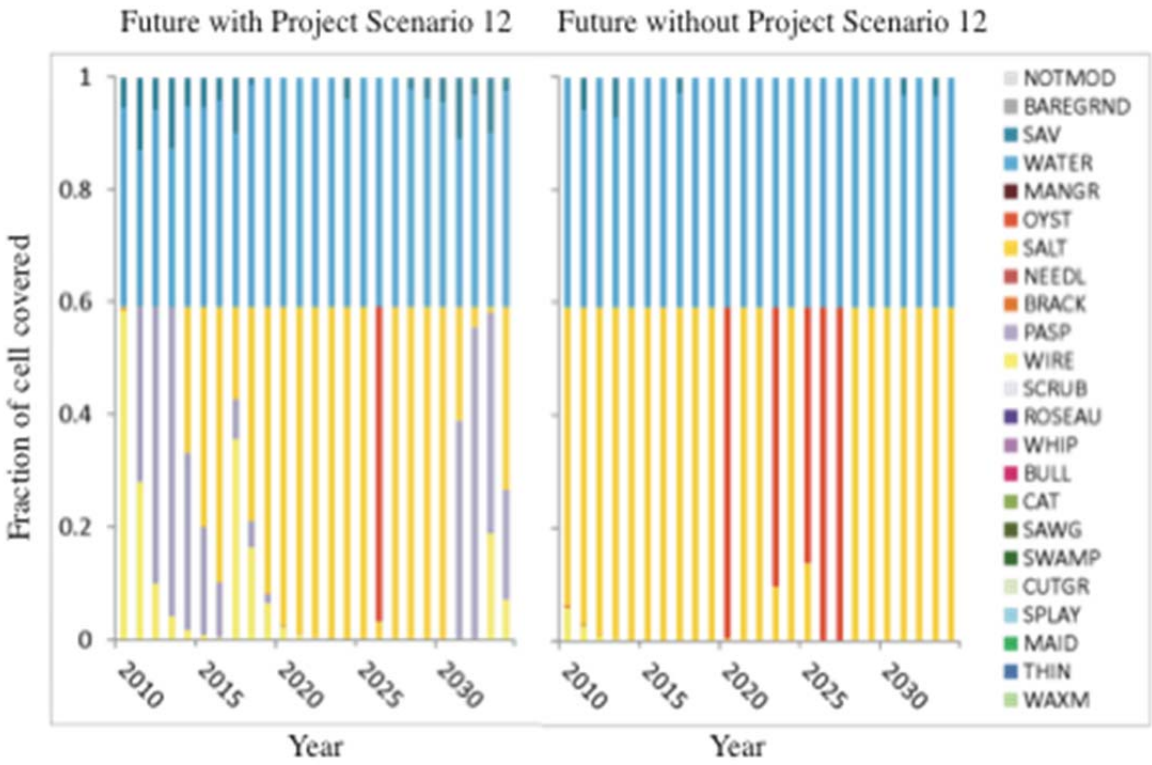

Figure 10. Example of the extraction of data used in a project report.

e. Training availability

It is not part of the current contract with the Vegetation model team to train others on the use of this model. If funding is available, the Vegetation team can train experienced coders that have working knowledge of UNIX to operate the model.

f. Users documentation availability and whether it is user friendly and complete

It is not part of the current contract with the Vegetation model team to make the model user friendly and to develop a user's manual. If funding is available, the vegetation team can do this.

g. Technical support availability

It is not part of the current contract with the Vegetation model team to provide technical surrport for future uses. If funding is available, the vegetation team can do this.

***h.* Software/hardware platform availability to all or most users**

The programming languages used to develop this model, C++, R and SH are freely available to anyone. The primary development platform, Apple OSX is a commonly available system. The model is also known to run under Linux, which is freely available. The model can also be moved to the Microsoft Windows operating system with the use of Cygwin (http://www.cygwin.com/).

i. Accessibility of the model

A copy of the code is included in Attachment 3. Please make note of this when printing this report.

j. **Transparency of model and how it allows for easy verification of calculations and outputs**
In developing the C++ code, we applied a minimum of the more advanced features available in C++. For example, we made limited use of class and functional templates and limited use of class inheritance. This was done for several reasons. First, many of the more advanced features are not necessary to produce the most efficient and functional model code. Second, by minimizing the number of advanced features the code is simpler, easier to maintain and easier to modify and verify. We also made use of in code comments to describe the purpose and function of model components.

5. Sources of model uncertainty

The greatest model uncertainty stems from using the growing conditions under which the vegetation types are currently found to generate the death and establishment tables. As more data becomes available (more years in the CRMS database) , the conditions under which sites change in vegetation composition and the conditions under which vegetation composition remains stable can inform future versions of the model. Although the relationship between most species distribution with respect to salinity have been extensively studied, the relationship with water level variability is less well known. Therefore, water level variability will be considered in the Model Uncertainty Analyses (see Section 8 and Appendix D-27 for additional information regardingthis analysis).

6. Suggested model improvements

- Include nutrient relationship with vegetation change.
- Investigate in more detail the relationships between vegetation change and water level and salinity.
- Include nutria grazing in the model by predicting nutria population responses to changes in hydrology and vegetation composition.
- Include turbidity as a variable predicted by the Eco-Hydrology model, so that it can be used for SAV presence prediction.
- Include additional sources of SAV data widening the spatial distribution and number of species represented.

7. Quality review

Specific Quality Review (QR) procedures for the Vegetation Model to support the 2012 Coastal Master Plan included the following:

- Assembly of an internal QR team at the onset of model development consisting of Dr. Rebecca Howard (USGS), Dr. Ken Krauss (USGS), Dr. Charles Sasser (LSU), and Dr. Mark Hester (ULL).
- Addition of Dr. Jacoby Carter (USGS) to the modeling team for submerged aquatic vegetation (SAV) expertise based on internal QR team recommendations.
- Internal QR team review of modeling strategy and agreement on the general model form.

- Internal QR team review of the SAV algorithm and the establishment and death tables that drive of the Vegetation model.
- Revision of the SWAMP mortality and establishment tables based on internal QR feedback. Which will be incorporated in the analysis of alternatives. Additional revisions have been postponed because of revisions in the eco-hydrology model (see section 3d).
- Internal QR team review of model outputs for FWOA and project group runs; output data QR was performed by visual inspection of the patterns of individual species in the model.
- The previous draft of this report was reviewed by the QR team and their comments have been incorporated in this version of the report.

8. Uncertainty analysis

The propagation of uncertainty through the various modelsis presented in a separate (See Appendix D-27). The largest uncertainty in LAVegMod is the mortality and establishment due to water level variability. These matrices (Attachment 1) were shifted one or two columns to the left (putting the center of the survival at higher levels of water variability) or to the right (putting the center of survival at lower levels of water level variability). Results from the Model Uncertainty Analysis will show how sensitive LAVegMod is to the uncertainties in the eco-hydrology and wetland morphology models as well as how sensitive the output from LAVegMod is to slight changes in the establishment and mortality matrices. To date, the Model Uncertainty Analysis required nine additional runs of the Vegetation model to fulfill the objective established by the team.

9. References

Burdick, D. M., I. A. Mendelssohn and K. L., Mckee. Live Standing Crop and Metabolism of the Marsh Grass Spartina Patens as Related to Edaphic Factors in a Brackish, Mixed Marsh Community in Louisiana. Estuaries 12v 1989:195-204.

Chabreck, R. H. 1972. Vegetation, water, and soil characteristics of the Louisiana coastal region. Louisiana State University Agricultural Experime'nt Station Bulletin No. 664. Louisiana State University, Baton Rouge, Louisiana.

Chabreck, R. H., J. T. Joanen and A. W. Palmisano. 1968. Vegetative type map of the Louisiana coastal marshes. Louisiana Wildlife and Fisheries Commission, New Orleans, Louisiana.

Childers, D. L., Doren R. F, Jones R., Noe G. B., Rugge M. and Scinto L.J. 2003. Decadal Change in Vegetation and Soil Phosphorus Pattern across the Everglades Landscape. Journal of Environmental Quality 32:344-362.

Clements, F. E. 1936. Nature and Structure of the Climax. The Journal of Ecology 24:252-284.

Davis, S. M. and Ogden, J. C. 1994 Everglades: The ecosystem and its restoration, St. Lucie Press.

Deangelis, D. L., L. J. Gross, M. A.Huston, W. F. Wolff, D. M.Fleming, E. J. Comiskey and S. M. Sylvester. Landscape modeling for everglades ecosystem restoration. Ecosystems 1: 64-75.

Delaune, R. D., R. J. Buresh, W. H. Patrick. "Relationship of Soil Properties to Standing Crop Biomass of Spartina Alterniflora in a Louisiana Marsh". Estuarine and Coastal Marine Science 8v 1979:477-487.

Elsey-Quirk, T., D. M. Seliskar, C. K. Sommerfield, and J. L. Gallagher. 2011. "Salt Marsh Plant Species in Coastal Louisiana". Ecology 59:760-769.

Folse, T. M., J. L. West, M. K. Hymel, J. P. Troutman, L. Sharp, D. Weifenbach, T. McGinnis, and L. B. Rodrigue. 2008. A Standard Operating Procedures Manual for the Coast-wide Reference Monitoring System-Wetlands: Methods for Site Establishment, Data Collection, and Quality Assurance/Quality Control. Louisiana Coastal Protection and Restoration Authority, Office of Coastal Protection and Restoration. Baton Rouge, LA. 191 pp.

Garcia L. V., T. Marañon, A. Moreno and L. Clemente. 1993. Above-ground biomass and species richness in a Mediterranean salt marsh. Journal of Vegetation Science 4:417-424.

Gleason, H. A. 1926. The Individualistic Concept of the Plant Association. Bulletin of the Torrey Botanical Club. 53:7-26.

Hopkinson, C. S., J. G. Gosselink, and R. T. Parrando. 1978. Aboveground Production of Seven Marsh Plant Species in Coastal Louisiana. Ecology 59:760-769.

Jordan, J., and E. Mouton. 2011. Nutria Harvest and Distribution 2010-2011 and A Survey of Nutria Herbivory Damage in Coastal Louisiana in 2011.

McJannet, C. L., P. A. Keddy and F. R. Pick. 1995. Nitrogen and phosphorus tissue concentrations in 41 wetland plants: a comparison accross habitats and functional groups. Functional Ecology 9:231-238.

Merino, J. H., J. Carter and S. L. Merino. 2009. Mesohaline submerged aquatic vegetation survey along the U.S. Gulf of Mexico coast, 2001 and 2002: A salinity gradient approach. Gulf of Mexico Science 2009: 9-20.

Mitsch, W. J. and J. G. Gosselink. Wetlands. 3rd ed. New York: John Wiley & Sons, Inc, 2000.

Mitsch, W., J. G. Gosselink, C. J. Anderson and L. Zhang. Wetland Ecosystems. New Jersey: John Wiley & Sons, Inc., 2009.

Pacala, S. and D. Tilman. 1994. Limiting Similarity in Mechanistic and Spatial Models of Plant Competition in Heterogeneous Environments. Am. Nat. 143:222-257.

Penfound, W. Y. and E.S. Hathaway. 1938. Plant communities in the marshland of southeastern Louisiana. Ecological Monographs 8:1-56.

Pratolongo, P., P. Kandus and M. M. Brinson. 2008. Net Aboveground Primary Production and Biomass Dynamics of Schoenoplectus Californicus (Cyperaceae) Marshes Growing Under Different Hydrological Conditions. Darwiniana 46(2): 258-269.

Sasser, C. E., J. M. Visser, E. Mouton, and J. Linscombe. 2008. Vegetation Types in Coastal Louisiana in 2007. USGS Open-File Report 2008-1224. http://pubs.usgs.gov/of/2008/1224/

R Development Core Team. 2010. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Sah, J. P., Ross M. S., Koptur S., Snyder J. R. 2004. Estimating aboveground biomass of broadleaved woody plants in the understory of Florida Keys pine forests. Forest Ecology and Management 203: 319-329.

Schmalzer, P. A., C. R. Hinkle and J.L. Mailander. Changes in Community Composition and Biomass in Juncus-Roemerianus Scheele and Spartina-Bakeri Merr Marshes One Year After a Fire, Wetlands. 11v, 77p, 1991.

Twilley, Robert R. and J. Barras. 2004. Formulation of the LCA Ecosystem Model. Louisiana Coastal Area (LCA), Louisiana Ecosystem Restoration Study.

Visser, J. M., Sasser C. E., Chabreck R. H. and Linscombe R. G. 1998. Marsh vegetation types of the Mississippi River Deltaic Plain, USA. Estuaries 21:818–828.

Visser, J. M., Sasser C. E., Chabreck R. H. and Linscombe R. G. 1999. Long-term vegetation change in Louisiana tidal marshes, 1968–92. Wetlands 19(1):168–175.

Visser, J. M., Sasser C. E., Chabreck R. H. and Linscombe R. G. 2000. Marsh vegetation types of the Chenier Plain, USA. Estuaries. 23-3: 318-327.

Visser, J. M., Sasser C. E., Chabreck R. H. and Linscombe R. G. 2002. The Impact of a Severe Drought on the Vegetation of a Subtropical Estuary. Estuaries. 25-6A: 1184-1195.

Visser, J. M., G. D. Steyer, G. P. Shaffer, S. S. Höppner, M. W. Hester, E. Reyes, P. Keddy, I. A. Mendelssohn, C. E. Sasser and C. Swarzenski. 2003. Habitat Switching Module, Chapter 9. In, R.R. Twilley (ed.), Coastal Louisiana Ecosystem Assessment and Restoration (CLEAR) Model of Louisiana Coastal Area (LCA) Comprehensive Ecosystem Restoration Plan. Volume I: Tasks 1-8. Final Report to Department of Natural Resources, Coastal Restoration Division, Baton Rouge, LA. Contract No. 2511-02-24. 319 pp.

Attachment 1

Mortality Probability Tables for the 19 Vegetation Types used for Project Evaluation

Vegetation Type	Characteristic Species	Abbreviation
1. Mangrove	*Avicennia germinans* (L.) L.	MANGR
2. Oystergrass	*Spartina alterniflora* Loisel.	OYST
3. Saltgrass	*Distichlis spicata* (L.) Greene	SALT
4. Needlerush	*Juncus roemerianus* Scheele	NEEDL
5. Brackish Mixture	Mixture of: *Spartina patens* (Ait.) Muhl., *Distichlis spicata* (L.) Greene, *Spartina alterniflora* Loisel.	BRACK
6. Wiregrass	*Spartina patens* (Ait.) Muhl.	WIRE
7. Paspalum	*Paspalum vaginatum* Sw.	PASP
8. Bullwhip	*Schoenoplectus californicus* (C.A. Mey.) Palla	WHIP
9. Roseaucane	*Phragmites australis* (Cav.) Trin. ex Steud.	ROSEAU
10. Scrub-shrub	Mixture of: *Iva frutescens* L., *Baccharis halimifolia* L.	SCRUB
11. Bulltongue	*Sagittaria lancifolia* L.	BULL
12. Swamp Forest	*Taxodium distichum* (L.) L.C. Rich. and *Nyssa aquatica* L.	SWAMP
13. Delta Splay	Mixture of: *Sagittaria latifolia* Willd. *Schoenoplectus deltarum* (Schuyler) Sojak, *Colocasia esculenta* (L.) Schott	SPLAY
14. Thinmat	Mixture of: *Eleocharis baldwinii* (Torr.) Chapm., *Hydrocotyle ranunculoides* L. f. *Bidens laevis* (L.) B.S.P.	THIN
15. Maidencane	*Panicum hemitomon* J.A. Schultes	MAID
16. Sawgrass	*Cladium mariscus* (L.) Pohl	SAWGR
17. Cattail	*Typha domingensis* Pers	CAT
18. Cutgrass	*Zizaniopsis miliacea* (Michx.) Doell & Aschers.	CUTG
19. Waxmyrtle	*Morella cerifera* (L.) Small	WAXM

1. Mortality Table for Mangrove

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	MANGR	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	0	1	0.95	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	0.75	0.6	0.6	0.6	0.7	0.9	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	0.8	0.55	0.4	0.4	0.4	0.5	0.7	0.9	1	1	1	1	1	1	1	1	1	1
	20	1	1	0.95	0.7	0.45	0.3	0.3	0.3	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	22	1	1	0.85	0.6	0.35	0.2	0.2	0.2	0.3	0.5	0.7	0.9	1	1	1	1	1	1	1	1	1
	24	1	1	0.85	0.6	0.35	0.1	0.1	0.1	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1
	26	1	1	0.85	0.6	0.35	0.1	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1
	28	1	1	0.85	0.6	0.35	0.1	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1
	30	1	1	0.85	0.6	0.35	0.1	0.1	0.1	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1

2. Mortality Table for Oystergrass

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	OYST	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	0.9	0.75	0.75	0.9	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	0.95	0.75	0.6	0.6	0.75	0.9	1	0	0	0	1	1	1	1	1	1	1
	8	1	1	1	1	0.8	0.6	0.45	0.45	0.6	0.75	0.9	0	0	0	1	1	1	1	1	1	1
	9	1	1	1	0.9	0.65	0.45	0.3	0.3	0.45	0.6	0.75	0.9	0	0	1	1	1	1	1	1	1
	10	1	1	1	0.75	0.5	0.25	0.15	0.15	0.3	0.45	0.6	0.75	0.9	0	1	1	1	1	1	1	1
	12	1	1	1	0.75	0.5	0.25	0	0	0.15	0.3	0.45	0.6	0.75	0.9	1	1	1	1	1	1	1
	14	1	1	1	0.75	0.5	0.25	0	0	0.15	0.3	0.45	0.6	0.75	0.9	1	1	1	1	1	1	1
	16	1	1	1	0.75	0.5	0.25	0	0	0.15	0.3	0.45	0.6	0.75	0.9	1	1	1	1	1	1	1
	18	1	1	1	0.75	0.5	0.25	0	0	0.15	0.3	0.45	0.6	0.75	0.9	1	1	1	1	1	1	1
	20	1	1	1	0.75	0.5	0.25	0.15	0.15	0.25	0.4	0.55	0.7	0.9	1	1	1	1	1	1	1	1
	22	1	1	1	0.9	0.65	0.5	0.3	0.3	0.5	0.65	0.7	0.9	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	0.85	0.65	0.6	0.6	0.8	0.95	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	0.85	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

3. Mortality Table for Saltgrass

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																			
	SALT	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	1.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	0.75	0.75	0.75	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	0.75	0.5	0.5	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	0.75	0.5	0.25	0.25	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	0.75	0.5	0.25	0	0	0	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1
	8	1	0.75	0.5	0.25	0	0	0	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1
	9	1	0.75	0.5	0.25	0	0	0	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1
	10	1	0.75	0.5	0.25	0	0	0	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1
	12	1	0.75	0.5	0.25	0	0	0	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1
	14	1	0.75	0.5	0.25	0	0	0	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	0.75	0.5	0.25	0.25	0.25	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	0.75	0.5	0.5	0.5	0.75	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	0.75	0.75	0.75	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

4. Mortality Table for Needlerush

Annual Mean Salinity (ppt)	Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
NEEDL	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
0.8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1.8	1	1	1	1	1	0.75	0.75	0.75	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
2	1	1	1	1	0.75	0.5	0.5	0.5	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1
3	1	1	1	0.75	0.5	0.3	0.3	0.3	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
4	1	1	0.75	0.5	0.25	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
5	1	1	0.75	0.5	0.25	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
6	1	1	0.75	0.5	0.25	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
7	1	1	0.75	0.5	0.25	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
8	1	1	0.75	0.5	0.25	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
9	1	1	0.75	0.5	0.25	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
10	1	1	0.75	0.5	0.25	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
12	1	1	0.75	0.5	0.25	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
14	1	1	0.75	0.5	0.25	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
16	1	1	0.75	0.5	0.35	0.2	0.2	0.2	0.3	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
18	1	1	0.75	0.5	0.45	0.3	0.3	0.3	0.4	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
20	1	1	1	0.75	0.6	0.5	0.5	0.5	0.5	0.8	0.9	1	1	1	1	1	1	1	1	1	1
22	1	1	1	1	0.75	0.75	0.75	0.75	0.75	0.9	1	1	1	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

5. Mortality Table for Brackish Mixture

	Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
Annual Mean Salinity (ppt)	BRACK	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.6	1	1	1	1	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.8	1	1	1	0.9	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1	1
	1	1	1	1	0.8	0.7	0.7	0.7	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1
	1.2	1	1	0.9	0.7	0.6	0.6	0.6	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1	1
	1.4	1	1	0.85	0.6	0.5	0.5	0.5	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1
	1.6	1	1	0.75	0.5	0.4	0.4	0.4	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
	1.8	1	0.9	0.65	0.4	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
	2	1	0.8	0.55	0.3	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1
	3	1	0.7	0.45	0.2	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	4	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	5	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	6	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	7	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	8	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	9	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	10	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	12	1	0.75	0.5	0.25	0	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	14	1	0.75	0.5	0.25	0.15	0.15	0.15	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	16	1	0.95	0.65	0.4	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
	18	1	1	0.8	0.55	0.45	0.45	0.45	0.45	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
	20	1	1	0.95	0.7	0.6	0.6	0.6	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	0.85	0.75	0.75	0.75	0.75	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

6. Mortality Table for Wiregrass

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	WIRE	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	1	1	1	0.9	0.9	0.9	0.9	0	0	0	0	1	1	1	1	1	1	1	1	1	1
	0.2	1	1	1	0.8	0.8	0.8	0.8	0	0	0	0	1	1	1	1	1	1	1	1	1	1
	0.4	1	1	0.9	0.7	0.7	0.7	0.7	0	0	0	0	1	1	1	1	1	1	1	1	1	1
	0.6	1	1	0.8	0.6	0.6	0.6	0.6	0	0	0	0	1	1	1	1	1	1	1	1	1	1
	0.8	1	0.9	0.7	0.5	0.5	0.5	0.5	0	0	0	0	1	1	1	1	1	1	1	1	1	1
	1	1	0.8	0.6	0.4	0.4	0.4	0.4	0	0	0	0	1	1	1	1	1	1	1	1	1	1
	1.2	1	0.75	0.55	0.35	0.3	0.3	0.3	0.25	0.25	0	0	1	1	1	1	1	1	1	1	1	1
	1.4	1	0.7	0.5	0.3	0.25	0.25	0.25	0.25	0.25	0	0	1	1	1	1	1	1	1	1	1	1
	1.6	1	0.65	0.45	0.25	0.2	0.2	0.2	0.25	0.25	0	0	1	1	1	1	1	1	1	1	1	1
	1.8	1	0.6	0.4	0.2	0.15	0.15	0.15	0.25	0.25	0	0	1	1	1	1	1	1	1	1	1	1
	2	0.95	0.55	0.35	0.15	0.1	0.1	0.1	0.25	0.25	0	0	1	1	1	1	1	1	1	1	1	1
	3	0.9	0.5	0.3	0.1	0.05	0.05	0.05	0.25	0.25	0	0	1	1	1	1	1	1	1	1	1	1
	4	0.8	0.6	0.4	0.2	0	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	5	0.8	0.6	0.4	0.2	0	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	6	0.8	0.6	0.4	0.2	0	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	7	0.8	0.6	0.4	0.2	0	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	8	0.8	0.6	0.4	0.2	0	0	0	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	9	0.8	0.6	0.4	0.2	0.1	0.1	0.1	0.2	0.4	0.6	0.8	1	1	1	1	1	1	1	1	1	1
	10	0.95	0.75	0.55	0.35	0.25	0.25	0.25	0.35	0.55	0.8	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	0.8	0.6	0.5	0.5	0.5	0.6	0.8	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	0.85	0.75	0.75	0.75	0.85	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

7. Mortality Table for Paspalum

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	PASP	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.4	1	1	1	0.9	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1	1	1
	0.6	1	1	0.9	0.8	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1	1
	0.8	1	0.9	0.8	0.7	0.7	0.7	0.7	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1
	1	0.9	0.8	0.7	0.6	0.6	0.6	0.6	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1	1
	1.2	0.8	0.7	0.6	0.5	0.5	0.5	0.5	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1
	1.4	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
	1.6	0.6	0.5	0.4	0.3	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
	1.8	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1
	2	0.4	0.3	0.2	0.15	0.15	0.15	0.15	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	3	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	4	0.4	0.3	0.2	0.1	0	0	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	5	0.4	0.3	0.2	0.1	0	0	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	6	0.4	0.3	0.2	0.1	0	0	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	7	0.4	0.3	0.2	0.1	0	0	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	8	0.4	0.3	0.2	0.1	0	0	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	9	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	10	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	12	0.6	0.6	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.55	0.65	0.8	0.85	0.9	1	1	1	1	1	1
	14	0.8	0.8	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.65	0.7	0.8	1	1	1	1	1	1	1	1	1
	16	1	1	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

8. Mortality Table for Bullwhip

Annual Mean Salinity (ppt)	Water Level Variability (standard deviation of water level relative to marsh surface elevation)																			
WHIP	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
0	1	0.9	0.8	0.7	0.6	0.6	0.6	0.6	0.6	0.65	0.75	0.85	0.95	1	1	1	1	1	1	1	1
0.2	0.9	0.8	0.7	0.6	0.5	0.5	0.5	0.5	0.5	0.55	0.65	0.75	0.85	0.95	1	1	1	1	1	1	1
0.4	0.8	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1	1	1	1	1
0.6	0.7	0.6	0.5	0.4	0.35	0.35	0.35	0.35	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1	1	1	1
0.8	0.6	0.5	0.4	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1	1	1
1	0.5	0.4	0.35	0.3	0.25	0.25	0.25	0.25	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1	1
1.2	0.4	0.35	0.3	0.25	0.2	0.2	0.2	0.2	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1
1.4	0.4	0.3	0.25	0.2	0.15	0.15	0.15	0.15	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1
1.6	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95
1.8	0.4	0.3	0.2	0.1	0.05	0.05	0.05	0.05	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85
2	0.4	0.3	0.2	0.1	0	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85
3	0.4	0.3	0.2	0.1	0	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85
4	0.4	0.3	0.2	0.1	0	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.75	0.85	0.95
5	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.85	0.95	1
6	0.5	0.4	0.35	0.25	0.25	0.25	0.25	0.25	0.25	0.3	0.35	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1	1
7	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.45	0.55	0.65	0.75	0.85	0.95	1	1	1	1	1	1
8	1	0.8	0.65	0.55	0.55	0.55	0.55	0.55	0.55	0.65	0.75	0.85	0.95	1	1	1	1	1	1	1	1
9	1	1	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.85	0.95	1	1	1	1	1	1	1	1	1	1
10	1	1	1	0.85	0.85	0.85	0.85	0.85	0.85	0.95	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

9. Mortality Table for Roseaucane

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	ROSEAU	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.9	0.8	0.7	0.6	0.5	0.5	0.5	0.5	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
	0.2	0.8	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.5	0.6	0.7	0.8	0.9	0.9	1	1	1	1	1	1
	0.4	0.7	0.6	0.5	0.4	0.3	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.8	0.9	0.9	1	1	1	1
	0.6	0.6	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.7	0.8	0.8	0.9	0.9	1	1
	0.8	0.5	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.6	0.7	0.7	0.8	0.8	0.9	0.9
	1	0.5	0.4	0.3	0.2	0.1	0	0	0	0.05	0.1	0.2	0.3	0.4	0.5	0.5	0.6	0.6	0.7	0.7	0.8	0.8
	1.2	0.5	0.4	0.3	0.2	0.1	0	0	0	0.05	0.1	0.1	0.2	0.3	0.4	0.4	0.5	0.5	0.6	0.6	0.7	0.7
	1.4	0.5	0.4	0.3	0.2	0.1	0	0	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1.6	0.5	0.4	0.3	0.2	0.1	0	0	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1.8	0.5	0.4	0.3	0.2	0.1	0	0	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	2	0.5	0.4	0.3	0.2	0.1	0	0	0	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	3	0.6	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85
	4	0.8	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.5	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1	1
	5	1	0.9	0.8	0.7	0.6	0.6	0.6	0.6	0.75	0.8	0.85	0.9	0.95	1	1	1	1	1	1	1	1
	6	1	1	1	0.9	0.8	0.8	0.8	0.8	0.85	0.9	0.95	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

10. Mortality Table for Scrub-shrub

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	SCRUB	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.85	0.8	0.75	0.7	0.55	0.55	0.55	0.55	0.55	0.55	0.6	0.65	0.8	0.9	1	1	1	1	1	1	1
	0.2	0.8	0.75	0.7	0.65	0.5	0.5	0.5	0.5	0.5	0.5	0.55	0.6	0.75	0.85	0.95	1	1	1	1	1	1
	0.4	0.75	0.7	0.65	0.6	0.45	0.45	0.45	0.45	0.45	0.45	0.5	0.55	0.7	0.8	0.9	1	1	1	1	1	1
	0.6	0.7	0.65	0.6	0.55	0.4	0.4	0.4	0.4	0.4	0.4	0.45	0.5	0.65	0.75	0.85	0.95	1	1	1	1	1
	0.8	0.65	0.6	0.55	0.45	0.35	0.35	0.35	0.35	0.35	0.35	0.4	0.45	0.6	0.7	0.8	0.9	1	1	1	1	1
	1	0.6	0.55	0.45	0.35	0.3	0.3	0.3	0.3	0.3	0.3	0.35	0.4	0.55	0.65	0.75	0.85	0.95	1	1	1	1
	1.2	0.55	0.45	0.35	0.3	0.25	0.25	0.25	0.25	0.25	0.25	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	1.4	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	1.6	0.5	0.4	0.3	0.2	0.15	0.15	0.15	0.15	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	1.8	0.5	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	2	0.5	0.4	0.3	0.2	0.1	0.05	0.05	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	3	0.5	0.4	0.3	0.2	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	4	0.5	0.4	0.3	0.2	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	5	0.5	0.4	0.3	0.2	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	6	0.5	0.4	0.3	0.2	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	7	0.5	0.4	0.3	0.2	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	8	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	9	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1
	10	0.9	0.8	0.7	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
	12	1	1	0.9	0.8	0.8	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

11. Mortality Table for Bulltongue

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	BULL	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.5	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.2	0.5	0.4	0.3	0.2	0.1	0.05	0.05	0.05	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.4	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.6	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.8	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1.2	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1.4	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1.6	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1.8	0.5	0.4	0.3	0.2	0.1	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	2	0.5	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	3	0.6	0.5	0.4	0.35	0.3	0.3	0.3	0.3	0.3	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.75	0.8	0.85	0.9
	4	0.8	0.7	0.6	0.55	0.5	0.5	0.5	0.5	0.5	0.5	0.55	0.6	0.65	0.75	0.8	0.85	0.9	0.95	1	1	1
	5	1	1	0.9	0.8	0.75	0.75	0.75	0.75	0.75	0.8	0.85	0.9	0.95	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

12. Mortality Table for Swamp Forest

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	SWAMP	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0.2	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0.4	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0.6	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0.8	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.2	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.4	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.6	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	1.8	0.3	0.2	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	3	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	4	0.4	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	5	0.5	0.4	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	6	0.6	0.5	0.4	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	7	0.7	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	8	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	9	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	10	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	12	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	14	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	16	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	18	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	20	0.7	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	22	0.8	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	24	0.9	0.8	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	26	1	0.9	0.9	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	28	1	1	1	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

13. Mortality Table for Delta Splay

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	SPLAY	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	1	0.8	0.6	0.4	0.2	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	0.2	1	0.8	0.6	0.4	0.2	0.1	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	0.4	1	0.8	0.6	0.4	0.2	0.1	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	0.6	1	0.8	0.6	0.4	0.2	0.1	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	0.8	1	0.8	0.6	0.4	0.2	0.1	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	1	1	0.8	0.6	0.4	0.2	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	1.2	1	0.8	0.6	0.4	0.25	0.2	0.2	0.2	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8
	1.4	1	0.85	0.65	0.45	0.45	0.4	0.4	0.4	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1
	1.6	1	0.95	0.8	0.75	0.7	0.6	0.6	0.6	0.6	0.65	0.7	0.75	0.8	0.85	0.9	0.95	1	1	1	1	1
	1.8	1	1	0.95	0.9	0.9	0.8	0.8	0.8	0.8	0.85	0.9	0.95	1	1	1	1	1	1	1	1	1
	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	4	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

14. Mortality Table for Thinmat

	Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
Annual Mean Salinity (ppt)	THIN	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
	0.4	0.1	0.1	0.1	0.1	0.1	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
	0.6	0.1	0.1	0.1	0.1	0.1	0.1	0	0	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
	0.8	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
	1	0.2	0.2	0.2	0.2	0.2	0.2	0.15	0.15	0.15	0.15	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1
	1.2	0.25	0.25	0.25	0.25	0.25	0.2	0.2	0.2	0.2	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1
	1.4	0.3	0.3	0.3	0.3	0.3	0.25	0.25	0.25	0.25	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1
	1.6	0.45	0.45	0.45	0.45	0.45	0.3	0.3	0.3	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1
	1.8	0.5	0.5	0.5	0.5	0.5	0.45	0.45	0.45	0.4	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1
	2	0.6	0.6	0.6	0.6	0.6	0.5	0.5	0.5	0.5	0.6	0.7	0.8	0.9	1	1	1	1	1	1	1	1
	3	0.7	0.7	0.7	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.8	0.9	1	1	1	1	1	1	1	1	1
	4	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	0.7	0.85	0.9	1	1	1	1	1	1	1	1	1
	5	0.9	0.9	0.9	0.9	0.9	0.8	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	0.9	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

15. Mortality Table for Maidencane

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	MAID																					
Annual Mean Salinity (ppt)	0	0.2	0.2	0.2	0.2	0.15	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.2	0.2	0.2	0.2	0.2	0.15	0.1	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.4	0.2	0.2	0.2	0.2	0.15	0.1	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.6	0.2	0.2	0.2	0.2	0.15	0.1	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	0.8	0.2	0.2	0.2	0.2	0.15	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65
	1	0.2	0.2	0.2	0.2	0.2	0.15	0.15	0.15	0.15	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7
	1.2	0.3	0.3	0.3	0.3	0.25	0.2	0.2	0.2	0.2	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75
	1.4	0.35	0.35	0.35	0.35	0.3	0.25	0.25	0.25	0.25	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8
	1.6	0.4	0.4	0.4	0.4	0.35	0.3	0.3	0.3	0.3	0.3	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.9
	1.8	0.5	0.5	0.5	0.5	0.4	0.35	0.35	0.35	0.35	0.35	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.9	1
	2	0.6	0.6	0.6	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.45	0.5	0.55	0.6	0.65	0.7	0.75	0.8	0.9	1	1
	3	0.7	0.7	0.7	0.7	0.6	0.5	0.5	0.5	0.5	0.5	0.6	0.65	0.7	0.75	0.8	0.9	1	1	1	1	1
	4	0.8	0.8	0.8	0.8	0.7	0.6	0.6	0.6	0.6	0.6	0.7	0.75	0.8	0.9	1	1	1	1	1	1	1
	5	0.8	0.8	0.8	0.8	0.8	0.7	0.7	0.7	0.7	0.7	0.8	0.9	1	1	1	1	1	1	1	1	1
	6	0.9	0.9	0.9	0.9	0.9	0.8	0.8	0.8	0.8	0.8	0.85	0.95	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	0.9	0.9	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

16. Mortality Table for Sawgrass

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	SAWG	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.45	0.4	0.35	0.3	0.25	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	0.2	0.45	0.4	0.35	0.3	0.25	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
	0.4	0.45	0.4	0.35	0.3	0.25	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.6	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.8	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.2	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.4	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.6	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.8	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	2	0.45	0.4	0.35	0.3	0.25	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	3	0.5	0.5	0.4	0.4	0.3	0.1	0.1	0.1	0.1	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15
	4	0.6	0.6	0.5	0.5	0.4	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25
	5	0.7	0.7	0.6	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	6	0.8	0.8	0.7	0.7	0.6	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
	7	0.9	0.9	0.8	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	8	1	1	0.9	0.9	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85	0.85
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

17. Mortality Table for Cattail

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
	CAT	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.3	0.3	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.2	0.3	0.3	0.3	0.2	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.4	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.6	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	0.8	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.2	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.4	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.6	0.3	0.3	0.3	0.2	0.1	0	0	0	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
	1.8	0.3	0.3	0.3	0.3	0.2	0.1	0.1	0.1	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	2	0.4	0.4	0.4	0.4	0.3	0.2	0.2	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	3	0.5	0.5	0.5	0.5	0.4	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	4	0.6	0.6	0.6	0.6	0.5	0.4	0.4	0.4	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	5	0.7	0.7	0.7	0.7	0.6	0.5	0.5	0.5	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	6	0.8	0.8	0.8	0.8	0.7	0.6	0.6	0.6	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	7	0.9	0.9	0.9	0.9	0.8	0.7	0.7	0.7	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	8	1	1	1	1	0.9	0.85	0.85	0.85	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

18. Mortality Table for Cutgrass

		Water Level Variability (standard deviation of water level relative to marsh surface elevation)																			
	CUTGR	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
Annual Mean Salinity (ppt)	0	0.2	0.2	0.2	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	0.2	0.2	0.2	0.2	0.2	0.2	0.1	0	0	0	0.1	0.15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	0.4	0.2	0.2	0.2	0.2	0.2	0.1	0	0	0	0.1	0.15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	0.6	0.2	0.2	0.2	0.2	0.2	0.1	0	0	0	0.1	0.15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	0.8	0.2	0.2	0.2	0.2	0.2	0.1	0	0	0	0.1	0.15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	1	0.2	0.2	0.2	0.2	0.2	0.15	0.1	0.1	0.1	0.15	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
	1.2	0.3	0.3	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
	1.4	0.4	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
	1.6	0.5	0.5	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
	1.8	0.6	0.6	0.6	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6
	2	0.7	0.7	0.7	0.7	0.7	0.6	0.6	0.6	0.6	0.6	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7
	3	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	4	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9
	5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	6	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

19. Mortality Table for Waxmyrtle

Annual Mean Salinity (ppt)	Water Level Variability (standard deviation of water level relative to marsh surface elevation)																				
WAXM	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9	2
0	0.25	0.2	0.15	0.1	0.05	0.05	0.05	0.05	0.05	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6
0.2	0.25	0.2	0.15	0.1	0.05	0	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6
0.4	0.25	0.2	0.15	0.1	0.05	0	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6
0.6	0.25	0.2	0.15	0.1	0.05	0	0	0	0	0	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.45	0.5	0.55	0.6
0.8	0.25	0.2	0.15	0.1	0.05	0.05	0.05	0.05	0.05	0.05	0.1	0.15	0.2	0.25	0.3	0.35	0.4	0.5	0.6	0.6	0.7
1	0.25	0.2	0.15	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.15	0.2	0.25	0.35	0.4	0.5	0.6	0.7	0.7	0.8
1.2	0.3	0.25	0.2	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.15	0.2	0.25	0.3	0.4	0.5	0.6	0.7	0.8	0.8	0.9
1.4	0.35	0.3	0.25	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.25	0.3	0.35	0.5	0.6	0.7	0.8	0.9	0.9	1
1.6	0.4	0.35	0.3	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.25	0.3	0.35	0.4	0.6	0.7	0.8	0.9	1	1	1
1.8	0.5	0.4	0.35	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.35	0.4	0.5	0.7	0.8	0.9	1	1	1	1
2	0.6	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.6	0.8	0.9	1	1	1	1	1
3	0.7	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.6	0.7	0.9	1	1	1	1	1	1
4	0.8	0.7	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.6	0.7	0.8	1	1	1	1	1	1	1
5	0.9	0.8	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.7	0.8	0.9	1	1	1	1	1	1	1
6	1	0.9	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.8	0.9	1	1	1	1	1	1	1	1
7	1	1	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	0.9	1	1	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
12	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
14	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
22	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
24	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
26	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
28	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
30	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Attachment 2

A simplified example of the ASCII Grid Plus file format used to store output from the LAVegMod model. This example does not show an actual output file from the LAVegMod model. LAVegMod output files provide information for thousands of cells and could not be displayed here.

```
ncols 9
nrows 8
xllcorner 403000
xllcorner 3260000
cellsize 500
NODATA_value -9999
-9999 -9999 -9999   32 -9999 -9999 -9999 -9999 -9999
-9999 -9999    23   31    39 -9999 -9999 -9999 -9999
-9999    14    22   30    38    46 -9999 -9999 -9999
-9999    13    21   29    37    45    53    61 -9999
-9999    12    20   28    36    44    52    60    68
-9999 -9999    19   27    35    43    51 -9999 -9999
-9999 -9999    18   26    34    42 -9999 -9999 -9999
-9999 -9999 -9999   25    33 -9999 -9999 -9999 -9999
CELLID,BAREGRND,SPLAY,WAXM,CUTGR,MAID,THIN,SWAMP,CAT,SAWG,BULL,ROSEAU,WHIP,SCRUB,WIRE,PASP,BRACK,NEEDL,SALT,OYST,MANGR,WAT
ER,SAV,NOTMOD
12, 0, 1.06852e-05, 0, 0, 0.0157225, 0.0950541, 0.0363687, 0, 0, 0.0003545, 8.9485e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02231, 0.07248, 0.7576
13, 0, 3.2055e-06, 0, 0, 0.013154, 0.055629, 0.072737, 0.00126533, 0, 0, 9.72663e-06, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.00789858, 0.0257014, 0.8236
14, 0, 4.2740e-06, 0, 0, 0.016896, 0.12059, 0.038389, 0.0012653, 0, 0.00035454, 9.532e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.01723, 0.05596, 0.7492
18, 0, 9.6166e-06, 0, 0, 0.018670, 0.0978712, 0.0686964, 0.00189799, 0, 0, 5.44691e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0536386, 0.174361, 0.5848
19, 0, 3.098e-05, 0, 0, 0.0293424, 0.122804, 0.0606144, 0.00316332, 0, 0, 4.47425e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0150172, 0.0485828, 0.7204
20, 0, 3.2055e-06, 0, 0, 0.0225206, 0.104329, 0.0397361, 0.00295243, 0, 0, 5.83598e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0133089, 0.0430911, 0.774
21, 0, 6.41e-06, 0, 0, 0.03266, 0.1064, 0.07947, 0.004850, 0, 0.0004727, 3.501e-05, 3.89e-07, 0, 0, 0, 0, 0, 0, 0, 0, 0.0240, 0.0779158, 0.674
22, 0, 6.41e-06, 0, 0, 0.032, 0.243, 0.049, 0.0031, 0, 0.0007, 0.000132, 1.7542e-06, 1.8781e-06, 0, 0, 0, 0, 0, 0, 0, 0.02723, 0.0879635, 0.55
23, 0, 9.616e-06, 0, 0, 0.0452, 0.2836, 0.1373, 0.00717, 0, 0.0003545, 0.0001809, 5.847e-07, 0, 0, 0, 0, 0, 0, 0, 0, 0.04234, 0.136458, 0.3472
25, 0, 0, 0, 0, 0.05335, 0.2336, 0.0585, 0.01434, 0, 0.00106364, 0.000161462, 2.14406e-06, 0, 0, 0, 0, 0, 0, 0, 0, 0.0411707, 0.132829, 0.4648
26, 0, 0, 0, 0, 0.02999, 0.1076, 0.1778, 0.004217, 0, 0.000709, 4.08518e-05, 5.84744e-07, 0, 0, 0, 0, 0, 0, 0, 0, 0.0260675, 0.0843325, 0.5692
27, 0, 0, 0, 0.01336, 0.02909, 0.160496, 0.0434404, 0.000632663, 0, 0.0013, 6.80864e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.0504459, 0.163154, 0.538
28, 0, 1.28e-05, 0, 0.0023, 0.0482, 0.322, 0.0676, 0.0048, 0, 0.00236, 4.86e-05, 1.94e-06, 2.815e-06, 0, 0, 0, 0, 0, 0, 0, 0.0423, 0.136, 0.37
```

29, 0, 7.479e-06, 0, 0.00511, 0.0331, 0.213, 0.0363, 0.000632, 0, 0.00165, 9.337e-05, 1.36e-06, 0, 0, 0, 0, 0, 0, 0, 0, 0.0778, 0.25, 0.3812
30, 0, 0, 0, 0, 0.004196, 0.2888, 0.245, 0.0444, 0, 0.00034, 0.00277, 8.007e-05, 0, 3.61015e-09, 0, 0, 0, 0, 0, 0, 0.0537691, 0.135831, 0.2244
31, 0, 3.526e-05, 0, 0.01926, 0.05704, 0.363, 0.04243, 0.0137, 0, 0.000827, 4.863e-05, 0, 5.634e-06, 0, 0, 0, 0, 0, 0, 0, 0.0617, 0.199, 0.242
32, 0, 8.548e-06, 0, 0.00432, 0.0386, 0.239, 0.00909, 0.00948, 0, 0.0013, 0.00012, 5.84744e-07, 0, 0, 0, 0, 0, 0, 0, 0, 0.142, 0.46, 0.0948
33, 0, 0, 0, 0.00471, 0.0166, 0.121, 0.0212151, 0.000632663, 0, 0.000354546, 3.30705e-05, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.169211, 0.545189, 0.12
34, 0, 2.733e-06, 0, 0, 0.00472, 0.176, 0.258, 0.0264, 0.000309, 0, 0.000992, 0, 0, 2.18113e-09, 0, 0, 0, 0, 0, 0, 0.0510972, 0.128903, 0.3528
35, 0, 1.6e-05, 0, 0.0131, 0.056, 0.38, 0.042, 0.01033, 0, 0.000709, 0.0001089, 5.847e-07, 2.817e-06, 0, 0, 0, 0, 0, 0, 0, 0.06027, 0.19, 0.23
36, 0, 1.389e-05, 0, 0.0051, 0.041, 0.276, 0.006, 0.0130, 0, 0.000118, 0.000114, 1.75e-06, 2.81e-06, 0, 0, 0, 0, 0, 0, 0, 0.131, 0.425, 0.1008
37, 0, 2.671e-05, 0, 0.0047, 0.0426, 0.269, 0.0495, 0.0101, 0, 0.00106, 0.000105, 3.8e-07, 0, 0, 0, 0, 0, 0, 0, 0, 0.0948, 0.3055, 0.2216
38, 0, 9.616e-06, 0, 0.01022, 0.1074, 0.4798, 0.01212, 0.0256, 0, 0.00697, 5.44e-05, 0, 2.81e-06, 0, 0, 0, 0, 0, 0, 0, 0.0642, 0.2069, 0.0864
39, 0, 3.205e-06, 0, 0.00707, 0.0382, 0.30, 0.003, 0.000632, 0, 0.000354, 8.559e-05, 5.847e-07, 0, 0, 0, 0, 0, 0, 0, 0, 0.093, 0.299, 0.2484
42, 0, 0, 0, 0.00595, 0.00137, 0.1454, 0, 0.1279, 0, 0, 0.0033, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.207764, 0.475836, 0.0324
43, 0, 0, 0, 0.0063771, 0, 0.0684338, 0, 0.0638596, 0, 0, 0.000529452, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.124074, 0.286326, 0.4504
44, 0, 0, 0, 0, 0, 0.14938, 0.415195, 0.00302526, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.4324
45, 0, 0, 0, 0, 0.0036, 0.184828, 0.386907, 0.00786568, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.4168
46, 0, 3.680e-06, 0, 0.00223198, 0.00275996, 0.0850889, 0, 0.080986, 0, 0, 0.000529452, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.127559, 0.292441, 0.4084
51, 0, 0, 0, 0, 0.0036, 0.155786, 0.138398, 0.00181516, 0, 0, 7.62215e-07, 1.48981e-10, 0, 0, 0, 0, 0, 0, 0, 0, 0.00700914, 0.0193909, 0.674
52, 0, 0, 0, 0, 0, 0.11384, 0.317555, 0.000605052, 0, 0, 0, 1.48981e-10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.568
53, 0, 0, 0, 0, 0, 0.268337, 0.298697, 0.00665557, 0, 0.000311136, 0, 1.14219e-09, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.426
50, 0, 0, 0, 0, 0, 0.197569, 0.226, 0.00907578, 0, 0.000155568, 0, 3.47622e-10, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.5672
61, 0, 0, 0, 0.0063771, 0, 0.0684338, 0, 0.0638596, 0, 0, 0.000529452, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.124074, 0.286326, 0.4504
68, 0, 0, 0, 0, 0, 0.14938, 0.415195, 0.00302526, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.4324

Attachment 3

This attachment contains the source code for the LAVegMod model.

I. Source code for Makefile

```
CPP=g++

FLAGS=
FLAGS+=-w
#FLAGS+=-g
FLAGS+=-O
#FLAGS+=-O2
#FLAGS+=-O3

#FLAGS+=-Ddebug_CTmemory

GSL_PATH=/Users/sdukesy/local/gsl
BOOST_PATH=/Users/sdukesy/local/boost

INC=
INC+=-I${GSL_PATH}/include
#INC+=-I${CGAL_PATH}/include
INC+=-I${BOOST_PATH}/include
#INC+=-I

LINK_PATH=
LINK_PATH+=-L${GSL_PATH}/lib
#LINK_PATH+=-L${CGAL_PATH}/lib
LINK_PATH+=-L${BOOST_PATH}/lib

LINK_LIB=
LINK_LIB+=-lgsl
LINK_LIB+=-lgslcblas
```

```
#LINK_LIB+=-lCGAL
LINK_LIB+=-lboost_regex
LINK_LIB+=-lm

LINK=${LINK_PATH} ${LINK_LIB}

.cc.o :
	${CPP} -c ${FLAGS} $< ${INC}

lavegmod : model.o newconfig.o
	${CPP} ${FLAGS} -o lavegmod model.o newconfig.o ${LINK}

clean:
	rm -f *.o
	rm -rf lavegmod
```

II. Source code for model.cc

```
// System includes
#include <sys/stat.h>
#include <time.h>
#include <unistd.h>

// C++/STL includes
#include <algorithm>
#include <cmath>
#include <cctype>
#include <fstream>
#include <ios>
#include <iostream>
#include <list>
#include <limits>
#include <locale>
```

```
#include <map>
#include <numeric>
#include <string>
#include <utility>
#include <vector>

// Third-party includes
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/matrix_proxy.hpp>
#include <boost/numeric/ublas/io.hpp>

// Model includes
#include "newconfig.h"
#include "systimes.h"


////////////////////////////////////////////////////////////////////////
// Utility/Auxiliary stuff.
//
////////////////////////////////////////////////////////////////////////
typedef double                                  Float;
typedef unsigned int                            LocationIndex;
typedef unsigned int                            SpIndex;
typedef boost::numeric::ublas::matrix<Float>            MovementMatrix;
typedef boost::numeric::ublas::matrix_row<MovementMatrix > MatrixRow;
typedef std::map<int, std::vector<Float> >              AGPTable;

class IOS
{
    public:
        virtual std::ostream& write(std::ostream& ) const =0;
        virtual IOS& operator=(IOS& orig)
        {
std::cout << "IOS::operator=" << std::endl;
```

```
            return *this;
        }
};

std::ostream& operator<<(std::ostream& ostr, const IOS& obj)
{
    return obj.write(ostr);
}

template<class Type>
std::ostream& operator<<(std::ostream& ostr, const std::vector<Type>& obj)
{
    for ( typename std::vector<Type>::const_iterator lC_iter = obj.begin(); lC_iter != obj.end(); ++lC_iter ) {
        ostr << (*lC_iter);
        if ( (lC_iter+1) != obj.end() ) { ostr << ", ";}
    }
    return ostr;
}

template<class First, class Second>
std::ostream& operator<<(std::ostream& ostr, const std::map<First, Second>& obj)
{
    typename std::map<First, Second>::const_iterator lC_iter = obj.begin();
    typename std::map<First, Second>::const_iterator lC_next = ++(obj.begin());
    for ( ;lC_iter != obj.end(); ++lC_iter, ++lC_next ) {
        ostr << "[ " << lC_iter->first << ", " << lC_iter->second << "]";
        if ( lC_next != obj.end() ) { ostr << ", ";}
    }
    return ostr;
}

void bprintf(FILE* aF_strm, const char* const acp_val, size_t ai_size )
{
        // This print the bits from left to right, least to most significant digit, which is backward
```

```
        // for how we normally write numbers.
    char lc_mask = 1;
    for ( int li_count = 0; li_count < ai_size; ++li_count ) {
        for ( int li_bit = 0; li_bit <= 7; ++li_bit )
            fprintf(aF_strm, "%d", (acp_val[li_count]&(lc_mask<<li_bit))? 1 :0 );
        // fprintf(aF_strm, " ");
    }
}

template<typename Type>
int CSVRowToVec(const std::string& str, std::vector<Type>& vec)
{
        boost::regex  csvrow( Config::myNUMBER + std::string("( *, *") + Config::myNUMBER + std::string(")*"), boost::regex::extended);
        boost::smatch results;

        if ( !regex_search(str, results, csvrow, boost::regex_constants::match_any) ) {
          std::cout << "CSVRowToVeg(...) : Error : unknown row content : " << str << std::endl;
          return 1;
        }

        vec.clear();

        boost::regex                elt(Config::myNUMBER, boost::regex::extended);
        boost::sregex_token_iterator i(str.begin(), str.end(), elt, 1);
        boost::sregex_token_iterator j;
        std::stringstream           sstream;
        Type value;
        while ( i != j ) {
          sstream.clear();
          sstream.str( std::string(*i) );
          sstream >> value;
          vec.push_back(value);
          ++i;
        }
```

```
        return 0;
}

int CSVRowToVec(const std::string& str, std::vector<std::string>& vec)
{
        boost::regex  csvrow( Config::myVARNAME + std::string("( *, *") + Config::myVARNAME + std::string(")*"), boost::regex::extended);
        boost::smatch results;

      std::string strcpy = boost::regex_replace( str, boost::regex(std::string("\"")), std::string("") );

        if ( !regex_search(strcpy, results, csvrow, boost::regex_constants::match_any) ) {
          std::cout << "CSVRowToVeg(...) : Error : unknown row content : " << strcpy << std::endl;
          return 1;
        }

        vec.clear();
        boost::regex                elt(Config::myVARNAME, boost::regex::extended);
        boost::sregex_token_iterator i(strcpy.begin(), strcpy.end(), elt, 1);
        boost::sregex_token_iterator j;
        std::stringstream           sstream;
        while ( i != j ) {
          vec.push_back(std::string(*i));
          ++i;
        }

    return 0;
}

template<typename Type>
class op : public IOS
{
// class op
// This is a simple class to store ordered pair information.
```

```
// The primary function of this class is the storage of geographic coordinates.
// The model combines this class with a Landscape class to store the
// geographic coordinates of each location included in the simulation.
// However, the model only uses this information to coordinate output.
// The coordinates are not used in any actual computation.
// The physical arrangement of the locations is completely defined by the movement
// matricies.

    public:
        Type x,y;

    public:
        op() : x(), y() {}
        op(const Type& a_x, const Type& a_y) : x(a_x), y(a_y) {}
        op(const op<Type>& aC_orig ) : x(aC_orig.x), y(aC_orig.y) {}
        ~op() {}

        op& operator=(const op<Type>& aC_orig) {
          if ( this != &aC_orig ) {
            x = aC_orig.x;
            y = aC_orig.y;
          }
          return *this;
        }

        std::ostream& write(std::ostream& aC_O ) const
        {
          aC_O << x << ", " << y;
          return aC_O;
        }

};

//////////////////////////////////////////////////////////////////////////////
```

```
// Actual Model stuff.
//
////////////////////////////////////////////////////////////////////////////
class Params;

template<class Type>
class Landscape : public std::vector<Type>, public IOS
{
    public:
        typedef        std::vector<Type>              BaseClass;
        typedef        std::vector<Type*>             OffsetClass;
        typedef        Landscape<Type>                ThisClass;
        typedef typename std::vector<Type>::iterator      iterator;
        typedef typename std::vector<Type>::const_iterator const_iterator;

        typedef int   ROW;
        typedef int   COL;
        typedef float NORTHING;
        typedef float EASTING;
        typedef float CELLSIZE;

    protected:
        ROW      m_nrows;
        COL      m_ncols;
        NORTHING m_yllcorner;
        EASTING  m_xllcorner;
        CELLSIZE m_cellsize;
        Type     m_nodatavalue;

        OffsetClass m_offset;

    public:
        inline ROW      nrows()      const { return m_nrows;}
        inline COL      ncols()      const { return m_ncols;}
```

```
    inline NORTHING yllcorner()   const { return m_yllcorner;}
    inline EASTING  xllcorner()   const { return m_xllcorner;}
    inline CELLSIZE cellsize()    const { return m_cellsize;}
    inline Type     nodatavalue() const { return m_nodatavalue;}

    inline int nlocs() const { return m_ncols*m_nrows;}

    Type& operator()(ROW r, COL c)       { return ( *(m_offset[r] + c) ); }
    Type  operator()(ROW r, COL c) const { return ( *(m_offset[r] + c) ); }
    Type& at(ROW r, COL c)               { if ( r < m_nrows && c < m_ncols ) {return( *(m_offset[r]+c));} else {throw std::out_of_range();} }
    Type  at(ROW r, COL c)        const { if ( r < m_nrows && c < m_ncols ) {return( *(m_offset[r]+c));} else {throw std::out_of_range();} }

private:
    void build_offset()
    {
      m_offset.resize(m_nrows);
      for ( int r = 0; r < m_nrows; ++r) m_offset.operator[](r) = &BaseClass::operator[](r*m_ncols);
  }

public:
    Landscape () :
      m_nrows(0),
      m_ncols(0),
      m_yllcorner(0),
      m_xllcorner(0),
      m_cellsize(0),
      m_offset(),
      BaseClass()
    {}

    Landscape(const Landscape<Type>& a_orig ) :
      m_nrows(a_orig.m_nrows),
      m_ncols(a_orig.m_ncols),
      m_yllcorner(a_orig.m_yllcorner),
```

```
    m_xllcorner(a_orig.m_xllcorner),
    m_cellsize(a_orig.m_cellsize),
    m_nodatavalue(a_orig.m_nodatavalue),
    m_offset(),
    BaseClass( BaseClass(a_orig) )
  {
    build_offset();
}

  ~Landscape() {}

  Landscape& operator=(const Landscape<Type>& a_orig)
  {
    if ( this != a_orig ) {
      m_nrows      = a_orig.m_nrows;
      m_ncols      = a_orig.m_ncols;
      m_yllcorner   = a_orig.m_yllcorner;
      m_xllcorner   = a_orig.m_xllcorner;
      m_cellsize    = a_orig.m_cellsize;
      m_nodatavalue = a_orig.m_nodatavalue;
      BaseClass::operator=(a_orig);

      build_offset();
    }
    return *this;
  }

  Landscape(ROW a_nrows, COL a_ncols,  NORTHING a_yllcorner, EASTING a_xllcorner, CELLSIZE a_cellsize, Type a_nodatavalue) :
    m_nrows(a_nrows),
    m_ncols(a_ncols),
    m_yllcorner(a_yllcorner),
    m_xllcorner(a_xllcorner),
    m_cellsize(a_cellsize),
    m_nodatavalue(a_nodatavalue)
```

```
{
  BaseClass::resize( m_ncols*m_nrows);
  build_offset();
}

void resize()
{
  m_nrows = 0;
  m_ncols = 0;
  m_yllcorner = 0;
  m_xllcorner = 0;
  m_cellsize = 0;
  BaseClass::resize( 0);
  m_offset.resize(0);
}

void resize(ROW a_nrows, COL a_ncols, NORTHING a_yllcorner, EASTING a_xllcorner, CELLSIZE a_cellsize, Type a_nodatavalue)
{
  m_nrows = a_nrows;
  m_ncols = a_ncols;
  m_yllcorner = a_yllcorner;
  m_xllcorner = a_xllcorner;
  m_cellsize = a_cellsize;
  m_nodatavalue = a_nodatavalue;
  BaseClass::clear();
  BaseClass::resize( m_ncols*m_nrows);
  build_offset();
}

template<typename AltType>
void resize(const Landscape<AltType>& orig)
{
  m_nrows = orig.nrows();
  m_ncols = orig.ncols();
```

```
        m_yllcorner = orig.yllcorner();
        m_xllcorner = orig.xllcorner();
        m_cellsize  = orig.cellsize();
        m_nodatavalue = orig.nodatavalue();
        BaseClass::clear();
        BaseClass::resize(m_ncols*m_nrows);
        build_offset();
      }

  public:
      std::ostream& write(std::ostream& strm) const
      {
        strm << "nrows "       << m_nrows << std::endl;
        strm << "ncols "       << m_ncols << std::endl;
        strm << "yllcorner "    << m_yllcorner << std::endl;
        strm << "xllcorner "    << m_xllcorner << std::endl;
        strm << "cellsize "     << m_cellsize << std::endl;
        strm << "nodata_value " << m_nodatavalue << std::endl;
        strm << "i, row, col, value" << std::endl;
        for ( int i = 0; i < BaseClass::size(); ++i ) {
          int r = i/m_ncols;
          int c = i%m_ncols;
          strm << i << ", " << r << ", " << c << ", " << BaseClass::operator[](i) << std::endl;
        }
        return strm;
      }
  protected:
      // void resize( std::size_type, Type ) {}


};

template<typename Type>
Type max(const Landscape<Type>& obj)
```

```
{
    Type ret = -std::numeric_limits<Type>::max();
    for ( typename Landscape<Type>::const_iterator iter = obj.begin(); iter != obj.end(); ++iter)
          ret = *iter != obj.nodatavalue() & *iter > ret ? *iter : ret;
    return ret;
}

class AsciiGridReader
{
    public:
        template<typename Type>
        int operator()(std::istream& strm, Landscape<Type>& obj) const
        {
          int li_err = 0;
          if ( !strm.good() ) {
            std::cout << "AsciiGridReader: Error: File stream is broken before anything was done." << std::endl;
            return 1;
          }

          boost::regex lC_comment     (std::string("#.*")                          );
          boost::regex lC_ncols       (std::string("NCOLS *"       + Config::myINT)    );
          boost::regex lC_nrows       (std::string("NROWS *"       + Config::myINT)    );
          boost::regex lC_xllcorner   (std::string("XLLCORNER *"    + Config::myNUMBER) );
          boost::regex lC_yllcorner   (std::string("YLLCORNER *"    + Config::myNUMBER) );
          boost::regex lC_cellsize    (std::string("CELLSIZE *"    + Config::myNUMBER) );
          boost::regex lC_nodata_value(std::string("NODATA_VALUE *" + Config::myNUMBER) );
          boost::regex asciigrid_row( Config::myNUMBER + std::string(" *") + std::string("( *") + Config::myNUMBER + std::string(")*"),
          boost::regex::extended);

          int li_nrows = 0;
          int li_ncols = 0;
          float lf_xllcorner = 0;
          float lf_yllcorner = 0;
          float lf_cellsize = 0;
```

```
Type li_nodata_value = 0;

std::string lS_line;
boost::smatch lC_results;
std::stringstream lC_sstream;

int li_count = 0;
while ( li_count < 6 ) {
  getline(strm, lS_line);
  if ( strm.eof() ) { std::cout << "AsciiGridReader: Error: eof encountered while reading header info." << std::endl; return 1; }
  if ( strm.bad() ) { std::cout << "AsciiGridReader: Error: bad bit encountered while reading header info. I'm hosed." << std::endl; return 1; }
  if ( strm.fail() ) { std::cout << "AsciiGridReader: Error: fail bit encountered while reading header info. You're hosed." << std::endl; return 1; }


  lS_line = boost::regex_replace( lS_line, lC_comment, std::string(" ") ); // Strip out comments, "#" is the comment character
  if (    regex_match(lS_line, boost::regex(std::string(" *"))  )    ) { continue; } // Skip blank lines

  ++li_count;

  std::transform(lS_line.begin(), lS_line.end(), lS_line.begin(), (int(*)(int)) std::toupper);


  if ( regex_search(lS_line, lC_results, lC_ncols) ) {
      lC_sstream.clear(); lC_sstream.str( std::string(lC_results[1].first, lC_results[1].second) ); lC_sstream >> li_ncols;
  } else if ( regex_search(lS_line, lC_results, lC_nrows) ) {
      lC_sstream.clear(); lC_sstream.str( std::string(lC_results[1].first, lC_results[1].second) ); lC_sstream >> li_nrows;
  } else if ( regex_search(lS_line, lC_results, lC_xllcorner) ) {
      lC_sstream.clear(); lC_sstream.str( std::string(lC_results[1].first, lC_results[1].second) ); lC_sstream >> lf_xllcorner;
  } else if ( regex_search(lS_line, lC_results, lC_yllcorner) ) {
      lC_sstream.clear(); lC_sstream.str( std::string(lC_results[1].first, lC_results[1].second) ); lC_sstream >> lf_yllcorner;
  } else if ( regex_search(lS_line, lC_results, lC_cellsize) ) {
      lC_sstream.clear(); lC_sstream.str( std::string(lC_results[1].first, lC_results[1].second) ); lC_sstream >> lf_cellsize;
  } else if ( regex_search(lS_line, lC_results, lC_nodata_value) ) {
      lC_sstream.clear(); lC_sstream.str( std::string(lC_results[1].first, lC_results[1].second) ); lC_sstream >> li_nodata_value;
```

```
  } else {
      std::cout << "AsciiGridReader::operator(...) : Error : unknown field " << lS_line << std::endl;
      ++li_err;
  }
}

// std::cout << "ncols = " << li_ncols << std::endl;
// std::cout << "nrows = " << li_nrows << std::endl;
// std::cout << "xllcorner = " << lf_xllcorner << std::endl;
// std::cout << "yllcorner = " << lf_yllcorner << std::endl;
// std::cout << "nodata_value = " << li_nodata_value << std::endl;

if ( li_err ) {
  std::cout << "AsciiGridReader::operator(...) : Error : errors encountered while reading data, bailing out." << std::endl;
  return li_err;
}

obj.resize(typename Landscape<Type>::ROW(li_nrows),
           typename Landscape<Type>::COL(li_ncols),
           typename Landscape<Type>::NORTHING(lf_yllcorner),
           typename Landscape<Type>::EASTING(lf_xllcorner),
           typename Landscape<Type>::CELLSIZE(lf_cellsize),
           Type(li_nodata_value) );

for ( int li_r = 0; li_r < li_nrows; ++li_r) {
  getline(strm, lS_line);
  if ( strm.eof() ) { std::cout << "AsciiGridReader: Error: eof encountered while reading data. Bummer." << std::endl; return 1; }
  if ( strm.bad() ) { std::cout << "AsciiGridReader: Error: bad bit encountered while reading data. That blows." << std::endl; return 1; }
  if ( strm.fail() ) { std::cout << "AsciiGridReader: Error: fail bit encountered while reading data. Life sucks." << std::endl; return 1; }

  if ( !regex_search(lS_line, lC_results, asciigrid_row, boost::regex_constants::match_any) ) {
      std::cout << "AsciiGridReader::operator(...) : Error : unknown row content " << lS_line << std::endl;
      ++li_err;
      continue;
```

```
            }

            boost::regex lC_elt(Config::myNUMBER, boost::regex::extended);
            boost::sregex_token_iterator i(lS_line.begin(), lS_line.end(), lC_elt, 1);
            boost::sregex_token_iterator j;
            int li_c = 0;
            while ( i != j && li_c < li_ncols ) {

                lC_sstream.clear();
                lC_sstream.str( std::string( *i ) );
                lC_sstream >> obj[li_r * li_ncols + li_c];

                ++i; ++li_c;
            }

            if ( i != j ) {
                std::cout << "AsciiGridReader::operator(...) : Error : wrong number of columns in line " << lS_line << std::endl;
                ++li_err;
            }

          }
          return li_err;
        }
};

class AsciiGridWriter
{
    public:
        template<typename Type>
        int operator()(std::ostream& strm, Landscape<Type>& obj) const
        {
          int li_err = 0;

          strm << "NROWS " << obj.nrows() << std::endl;
```

```
        strm << "NCOLS " << obj.ncols() << std::endl;
        strm << "XLLCORNER " << obj.xllcorner() << std::endl;
        strm << "YLLCORNER " << obj.yllcorner() << std::endl;
        strm << "CELLSIZE " << obj.cellsize() << std::endl;
        strm << "NODATA_VALUE " << obj.nodatavalue() << std::endl;

        for ( int r = 0; r < obj.nrows(); ++r) {
          strm << obj[r,0];
          for ( int c = 1; c < obj.ncols(); ++c) {
              strm << " " << obj(r,c);
          }
          strm << std::endl;
        }

        return li_err;
      }
};

class EcoHydroReader
{
    protected:
      int read_raw_data(std::ifstream& strm, std::map<int, float>& data) const
      {
        int li_err = 0;
        boost::regex ecohydro_row( Config::myNUMBER + std::string("\t*") + std::string("(\t*") + Config::myNUMBER + std::string(")*"),
        boost::regex::extended);

        std::string header;
        std::string line;

        // Step 1: Read in the header
        {
        int strmPos = strm.tellg();
        strm.seekg(0, std::ios_base::beg);
```

```
getline(strm, header);
std::transform(header.begin(), header.end(), header.begin(), (int(*)(int)) std::toupper);
if ( strmPos != 0 ) strm.seekg(strmPos, std::ios_base::beg); // If strmPos == 0, then we were at the head of the file where the header info is
located
}

// Step 1.5: Read the next line of data
getline(strm, line);

// Step 2: Make sure the header and the line have the correct format.
// STG|TMP|SAL
boost::smatch results;
// if ( regex_search(header, results, boost::regex("(Day|DAY|Month|MONTH|Time|TIME)(\t*[AA|CP|PB|STG|TMP|SAL][0-
9]+)*",boost::regex::extended), boost::regex_constants::match_any) ) {
//        std::cout << "EcoHydroReader::read_raw_data(...) : Error : unknown header content " << header << std::endl;
//    ++li_err;
// }

if ( !( regex_search(header, results, boost::regex("(MONTH|DAY)(\t*AA[0-9]+)*",boost::regex::extended), boost::regex_constants::match_any)
||
      regex_search(header, results, boost::regex("(MONTH|DAY)(\t*CP[0-9]+)*",boost::regex::extended), boost::regex_constants::match_any) ||
      regex_search(header, results, boost::regex("(MONTH|DAY)(\t*PB[0-9]+)*",boost::regex::extended), boost::regex_constants::match_any)    )
) {
  std::cout << "EcoHydroReader::read_raw_data(...) : Error : unknown header content " << header << std::endl;
  ++li_err;
}

if ( !regex_search(line, results, ecohydro_row, boost::regex_constants::match_any) ) {
  std::cout << "EcoHydroReader::read_raw_data(...) : Error : unknown row content " << line << std::endl;
std::cout << line << std::endl;
  ++li_err;
}

// Step 3: If errors have occured at this point, thing are hosed and there is nothing more we can do. Bail out.
```

```
if ( li_err ) return li_err;

// Step 4: Strip off the prefix label on each column label and the "TIME" label
// and transform the tab seperated values into column seperated values.
// header = boost::regex_replace(  header, boost::regex(std::string("Day\t*")), std::string("")  );
header = boost::regex_replace(  header, boost::regex(std::string("DAY\t*")), std::string("")  );
// header = boost::regex_replace(  header, boost::regex(std::string("Month\t*")), std::string("")  );
header = boost::regex_replace(  header, boost::regex(std::string("MONTH\t*")), std::string("")  );
// header = boost::regex_replace(  header, boost::regex(std::string("Time\t*")), std::string("")  );
// header = boost::regex_replace(  header, boost::regex(std::string("TIME\t*")), std::string("")  );
header = boost::regex_replace(  header, boost::regex(std::string("AA")), std::string("")  );
header = boost::regex_replace(  header, boost::regex(std::string("CP")), std::string("")  );
header = boost::regex_replace(  header, boost::regex(std::string("PB")), std::string(""), boost::match_default | boost::format_all  );
// header = boost::regex_replace(  header, boost::regex(std::string("STG")), std::string(""), boost::match_default | boost::format_all  );
// header = boost::regex_replace(  header, boost::regex(std::string("TMP")), std::string(""), boost::match_default | boost::format_all  );
// header = boost::regex_replace(  header, boost::regex(std::string("SAL")), std::string(""), boost::match_default | boost::format_all  );
header = boost::regex_replace(  header, boost::regex(std::string("\t")), std::string(",")  );

line   = boost::regex_replace( line, boost::regex(std::string("\t")), std::string(",") );

// Step 4.1: Push the data from the header into a vector
std::vector<int> header_index;
li_err += CSVRowToVec(header, header_index);

// Step 5: Push the data from the line into a vector
std::vector<Float> values;
li_err += CSVRowToVec(line, values);

// Step 6: Push everything into the map object
for ( int i = 0; i < header_index.size(); ++i) {
  data[ header_index[i] ] = values[i+1]; // Need to use i+1, because the zeroth element of the values array contains the time.
}

return li_err;
```

```
    }

public:
    int operator()(std::ifstream& strm, Landscape<std::vector<Float> >& obj, const Landscape<int>& BoxID, int N) const
    {

      // Step 1: Make sure the input object "obj" has the correct size and shape.
      std::vector<Float> eltinit;
      if ( obj.nlocs() != BoxID.nlocs() ) {
        obj.resize( BoxID.nrows(), BoxID.ncols(), BoxID.yllcorner(), BoxID.xllcorner(), BoxID.cellsize(), eltinit);
        eltinit.resize(N);
        for ( int i = 0; i < obj.nlocs(); ++i ) { obj[i] = eltinit; }
      }

      // Step 2: Read the data from the ecohydro file and pack it into the input object.
      std::map<int, float> data;
      for ( int d = 0; d < N; ++d ) {

        // Step 2.1: Slurp up the data from the datafile into a map object
        data.clear();
        if ( read_raw_data(strm, data) ) {
            std::cout << "EcoHydroReader::operator()(...) : Error : Could not read raw data from input file." << std::endl;
            return 1;
        }

        // Step 2.2: Pack the data into the input object.
        // How long is obj?
        for ( int i = 0; i < obj.size(); ++i) {
            if ( BoxID[i] != BoxID.nodatavalue() ) {
                obj[i][d] = data[ BoxID[i] ];
            }
        }
      }
```

```
        return 0;


};

class Function
{
    protected:
        template<typename type>
        void toIndex(Float x, const std::vector<type>& vec, int& b, Float& scale) const
        {
          if ( x < *vec.begin() || *vec.rbegin() < x ) {
            std::stringstream strm;

            strm << "Function::toIndex(...): Error: x out of range" << std::endl;
            strm << "x           = " << x << std::endl;
            strm << "vec.begin()  = " << *vec.begin() << std::endl;
            strm << "vec.rbegin() = " << *vec.rbegin() << std::endl;
            throw std::out_of_range(strm.str());
          }

             b      = 0;
          int   t      = vec.size()-1;
          int   m      = ceil((b+t)/2.0);
          // float epsilon = vec[b+1] - vec[b];

          if ( x == vec[t] ) {
            b     = t-1;
            scale = 1.0;
            return;
          }

          while( vec[b+1] <= x ) {
                #ifdef Verbose_Function__toIndex
```

```
            std::cout << "b      = " << b << std::endl;
            std::cout << "m      = " << m << std::endl;
            std::cout << "t      = " << t << std::endl;
            std::cout << "vec[b]  = " << vec[b] << std::endl;
            std::cout << "vec[m]  = " << vec[m] << std::endl;
            std::cout << "vec[t]  = " << vec[t] << std::endl;
            // std::cout << "epsilon = " << epsilon << std::endl;
            #endif

        if (vec[b] <= x && x < vec[m] ) t = m; else b = m;
        m = ceil((b+t)/2.0);
        // epsilon = vec[b+1] - vec[b];

      }

      #ifdef Verbose_Function__toIndex
      std::cout << "b      = " << b << std::endl;
      std::cout << "m      = " << m << std::endl;
      std::cout << "t      = " << t << std::endl;
      std::cout << "vec[b]  = " << vec[b] << std::endl;
      std::cout << "vec[m]  = " << vec[m] << std::endl;
      std::cout << "vec[t]  = " << vec[t] << std::endl;
      // std::cout << "epsilon = " << epsilon << std::endl;
      #endif

      scale = (x - vec[b])/(vec[b+1] - vec[b]);

    }

public:
    Function() {}
    Function(const Function& orig) {}
    ~Function() {}
```

```
        virtual Function& operator=(const Function& orig )
        {
std::cout << "Function::operator=" << std::endl;
          return *this;
        }

};

class PWLFunc_2args : public Function, public IOS
{
    protected:
        std::vector<Float> m_X;
        std::vector<Float> m_Y;
        std::vector< std::vector<Float> > m_data;

    public:
        int config(const std::vector<Float>& a_X, const std::vector<Float>& a_Y, const std::vector< std::vector<Float> >& a_data)
        {
          m_X    = a_X;
          m_Y    = a_Y;
          m_data = a_data;
          return 0;
        }

        Float operator()(Float x, Float y) const
        {
          int i        = 0;
          int j        = 0;
          Float i_scale = 0;
          Float j_scale = 0;
          try { Function::toIndex(x, m_X, i, i_scale); }
          catch ( std::exception& exc ) {
            std::stringstream strm;
            strm << "PWLFunc_2args::operator()(Float, Float): Error: out of range error caught on x" << std::endl;
```

```
  strm << "x = " << x << std::endl;
  strm << std::string( exc.what() ) << std::endl;
  throw std::out_of_range(strm.str());
}
try { Function::toIndex(y, m_Y, j, j_scale); }
catch ( std::exception& exc ) {
  std::stringstream strm;
  strm << "PWLFunc_2args::operator()(Float, Float): Error: out of range error caught on y" << std::endl;
  strm << "y          = " << y << std::endl;
  strm << "m_Y.begin() = " << *m_Y.begin() << std::endl;
  strm << "m_Y.end()   = " << *m_Y.rbegin() << std::endl;
  strm << std::string( exc.what() ) << std::endl;
  throw std::out_of_range(strm.str());
}


Float x0 = m_data[j  ][i]*(1.0 - i_scale) + m_data[j  ][i+1]*(i_scale);
Float x1 = m_data[j+1][i]*(1.0 - i_scale) + m_data[j+1][i+1]*(i_scale);

Float ret = x0*(1.0 - j_scale) + x1*(j_scale);

// #define Verbose_PWLFunc_2args
#ifdef Verbose_PWLFunc_2args
std::cout << "x      = " << x << std::endl;
std::cout << "y      = " << y << std::endl;
std::cout << "m_X    = " << m_X << std::endl;
std::cout << "m_Y    = " << m_Y << std::endl;
std::cout << "i      = " << i << std::endl;
std::cout << "j      = " << j << std::endl;
std::cout << "i_scale = " << i_scale << std::endl;
std::cout << "j_scale = " << j_scale << std::endl;
std::cout << "x0     = " << x0 << std::endl;
std::cout << "x1     = " << x1 << std::endl;
std::cout << "ret    = " << ret << std::endl;
```

```
    #endif

    return ret;
  }

public:
  PWLFunc_2args() : m_X(), m_Y(), m_data() {}
  PWLFunc_2args(const PWLFunc_2args& orig) :
    m_X(orig.m_X),
    m_Y(orig.m_Y),
    m_data(orig.m_data)
  {}
  ~PWLFunc_2args()
  {
    m_X.clear();
    m_Y.clear();
    m_data.clear();
  }

  virtual PWLFunc_2args& operator=(const PWLFunc_2args& orig )
  {
    if ( this != &orig ) {
      m_X    = orig.m_X;
      m_Y    = orig.m_Y;
      m_data = orig.m_data;
    }
    return *this;
  }

public:
  std::ostream& write(std::ostream& ostrm) const
  {
    ostrm << "m_X = " << m_X << std::endl;
    ostrm << "m_Y = " << m_Y << std::endl;
```

```
        ostrm << "m_data size = " << m_data.size() << ", " << m_data[0].size() << std::endl;
        return ostrm;
      }
};

class SpeciesInfo : public IOS
{

    public:
            std::string name;
            SpIndex     index;
           std::string abr;
      static PWLFunc_2args P;
           PWLFunc_2args D;

      static SpIndex spi_water;
      static SpIndex spi_sav;

    public:
      SpeciesInfo() :
        name("None"),
        index(0),
        abr("None"),
        D()
      {}

      SpeciesInfo(const SpeciesInfo& aC_orig) :
        name (aC_orig.name),
        index(aC_orig.index),
        abr  (aC_orig.abr),
        D(aC_orig.D)
      {}

      ~SpeciesInfo()
```

```
        {
          name  = "None";
          index = 0;
          abr   = "None";
        }

        SpeciesInfo& operator=(const SpeciesInfo& aC_orig)
        {
          if ( this != &aC_orig ) {
            name   = aC_orig.name;
            index  = aC_orig.index;
            abr    = aC_orig.abr;
            D      = aC_orig.D;
          }
          return *this;
        }

        std::ostream& write(std::ostream& strm) const
        {
          strm << "name  = " << name << std::endl;
          strm << "index = " << index << std::endl;
          strm << "abr   = " << abr << std::endl;
          strm << "D     = \n" << D << std::endl;
          return strm;
        }
};

PWLFunc_2args SpeciesInfo::P; // This is a common establishment function for all the species in SpeciesLUT
SpIndex SpeciesInfo::spi_water;
SpIndex SpeciesInfo::spi_sav;


class SpeciesD_config
{
```

```
public:
    int config(const std::string& filename, PWLFunc_2args& obj) const
    {
      int err = 0;


      std::string   headerPrefix("^ *[a-zA-Z]+ *, *");
      boost::regex  headerRegex( headerPrefix + Config::myNUMBER + std::string("( *, *") + Config::myNUMBER + std::string(")*"),
      boost::regex::extended);
      boost::regex  prefixMatch( headerPrefix );
      boost::regex  numberMatch( std::string(" *") + Config::myNUMBER + std::string(" *, *"));
      boost::regex  rowRegex( Config::myNUMBER + std::string("( *, *") + Config::myNUMBER + std::string(")*"), boost::regex::extended);
      boost::smatch results;

      std::string       line;
      // ***********************************
      //
      // ***********************************
      std::ifstream     fstrm;
      int               count = 0;
      fstrm.clear();
      std::cout << "Reading Species persistance data from " << filename << std::endl;
      fstrm.open(filename.c_str()); // Open the file
      if ( !fstrm.is_open() ) {
        std::cout << "class SpeciesD : Error : Could not open input file " << filename << std::endl;
        return 1;
      }

      // ******************************************************
      // Read off the header
      // Erase the headerPrefix, since we don't need it.
      // Parse the rest of the header into a vector.
      //
      // ******************************************************
```

```
  // std::cout << "class SpeciesD::config(...) : Reading header" << std::endl;
  getline(fstrm, line); if ( !regex_search(line, results, headerRegex, boost::regex_constants::match_any) ) { std::cout << "class SpeciesD::config(...) :
  Error : unknown header " << line << std::endl; return 1; }

  line = (boost::regex_replace(line, prefixMatch, std::string(""), boost::match_default | boost::format_all));

  std::vector<Float> X; // values in X are wave amplitudes.
  // std::cout << "class SpeciesD::config(...) : Parsing header" << std::endl;
  err += CSVRowToVec(line, X); if ( err ) { std::cout << "class SpeciesD::config(...) : Error : CSVRowToVec failed" << std::endl;; return err; }
  // std::cout << "class SpeciesD::config(...) : header = " << X << " (" << line  << "}" << std::endl;
for ( std::vector<Float>::iterator iter = X.begin(); iter != X.end(); ++iter ) { (*iter) *= 0.3048; }

  // ******************************************************
  // Pack all the remaining data into some vectors.
  //
  // ******************************************************
  std::vector<Float>              Y;
  std::vector< std::vector<Float> > values;

  // std::cout << "class SpeciesD_config::config(...) : Packing data into vectors" << std::endl;
  while ( getline(fstrm, line) ) {
    // Make sure the line has the correct form.
    if ( !regex_search(line, results, rowRegex, boost::regex_constants::match_any) ){
        std::cout << "class SpeciesD::config(...) : Error : unknown row "<< line << std::endl;
        ++err;
        continue;
    }

    // Pack the data from elt 1 to N-1 into values.
    std::vector<Float> temp;
    err += CSVRowToVec(line, temp); if ( err ) std::cout << "class SpeciesD::config(...) : Error : CSVRowToVec failed" << std::endl;

    values.push_back( std::vector<Float>(0) );
    values.back().assign(temp.begin()+1,temp.end());
```

```
            // Pack the data from element 0 into Y. Y values are the mean salinities.
            Y.push_back(*temp.begin());
          }

          // ****************************************************
          // Finally, configure the data elements inside SpeciesP
          //
          // ****************************************************

          obj.config(X,Y,values);


          return err;
        }

};

class SpeciesP_config
{
    public:
        int config(const std::string& filename, PWLFunc_2args& obj, const std::map<SpIndex,SpeciesInfo>& LUT ) const
        {
          int err = 0;


          std::string   headerPrefix(" *ALL *, *");
          boost::regex  headerRegex( headerPrefix + Config::myNUMBER + std::string("( *, *") + Config::myNUMBER + std::string(")*"),
          boost::regex::extended);
          boost::regex  prefixMatch( headerPrefix );
          boost::regex  numberMatch( std::string(" *") + Config::myNUMBER + std::string(" *, *"));
          boost::regex  rowRegex( Config::myNUMBER + std::string("( *, *") + Config::myNUMBER + std::string(")*"), boost::regex::extended);
          boost::smatch results;
```

```
// ***********************************
// Slurp up all the data from the file.
// Search and replace species abreviations w/ indicies
//
// ***********************************
std::stringstream  strstrm;
std::string       line;
std::ifstream     fstrm;
int               count = 0;
fstrm.clear();
std::cout << "class SpeciesP_config::config(...) : Reading SpeciesP data from " << filename << std::endl;
fstrm.open(filename.c_str()); // Open the file
if ( !fstrm.is_open() ) {
  std::cout << "class SpeciesP_config::config(...): Error: Could not open input file " << filename << std::endl;
  return 1;
}

while ( getline(fstrm, line) ) { // Slurping
  strstrm << line << std::endl;
  ++count;
}
fstrm.close(); // Close file

line.clear();
line = strstrm.str();
for ( std::map<SpIndex,SpeciesInfo>::const_iterator lC_iter = LUT.begin(); lC_iter != LUT.end(); ++lC_iter ) { // Search and replace
  std::stringstream index;
  index << (lC_iter->second).index;
  line = boost::regex_replace(line, boost::regex( (lC_iter->second).abr), index.str(), boost::match_default | boost::format_all);
}
strstrm.clear();
strstrm.str(line);
// ****************************************************
// Read off the header
```

```
  // Erase the headerPrefix, since we don't need it.
  // Parse the rest of the header into a vector.
  //
  // ****************************************************
  // std::cout << "class SpeciesP::config(...) : Reading header" << std::endl;
  getline(strstrm, line); if ( !regex_search(line, results, headerRegex, boost::regex_constants::match_any) ) { std::cout << "class SpeciesP::config(...)
  : Error : unknown header " << line << std::endl; return 1; }

  line = (boost::regex_replace(line, prefixMatch, std::string(""), boost::match_default | boost::format_all));

  std::vector<Float> X; // values in X are salinity values.
  // std::cout << "class SpeciesP::config(...) : Parsing header" << std::endl;
  err += CSVRowToVec(line, X); if ( err ) { std::cout << "class SpeciesP::config(...) : Error : CSVRowToVec failed" << std::endl;; return err; }
  // std::cout << "class SpeciesP::config(...) : header = " << X << " (" << line  << ")" << std::endl;
for ( std::vector<Float>::iterator iter = X.begin(); iter != X.end(); ++iter ) { (*iter) *= 0.3048; }


  // ****************************************************
  // Pack all the remaining data into some vectors.
  //
  // ****************************************************
  std::vector<Float>              Y;
  std::vector< std::vector<Float> > values;

  // std::cout << "class SpeciesP_config::config(...) : Packing data into vectors" << std::endl;
  while ( getline(strstrm, line) ) {
    // Make sure the line has the correct form.
    if ( !regex_search(line, results, rowRegex, boost::regex_constants::match_any) ){
        std::cout << "class SpeciesP::config(...) : Error : unknown row "<< line << std::endl;
        ++err;
        continue;
    }

    // Pack the data from elt 1 to N-1 into temp.
```

```
  std::vector<Float> temp;
  err += CSVRowToVec(line, temp);
  values.push_back( std::vector<Float>(0) );
  Y.push_back(0);
  if ( err ) {
      std::cout << "class SpeciesP::config(...) : Error : CSVRowToVec failed" << std::endl;
  } else {
      values.back().assign(temp.begin()+1,temp.end());
      Y.back() = *temp.begin();
  }

}


// ****************************************************
// The establishment function (P) is a little odd.
// It needs to return single values over discrete ranges
// instead of returning linearly interpolated values.
// I can do this w/ the current data structure, but
// the configuration passed to the object needs to
// be built from the input data, rather than just passing
// the input straight into the function.
//
// ****************************************************

std::vector<Float>          XX    (X.size()*2,0);
std::vector<Float>          YY    (Y.size()*2,0);
std::vector< std::vector<Float> > vvalues(2*values.size(), std::vector<Float>(X.size()*2,0) );

XX[0] = X[0];
for ( int i = 1; i < X.size(); ++i ) { XX[2*i-1] = XX[2*i] = (X[i-1] + X[i])/2; }
XX.back() = X.back();
YY[0] = Y[0];
for ( int i = 1; i < Y.size(); ++i ) { YY[2*i-1] = YY[2*i] = (Y[i-1] + Y[i])/2; }
```

```
        YY.back() = Y.back();


        for ( int i = 0; i < values.size(); ++i ) {
          for ( int j = 0; j < values[i].size(); ++j ) {
               vvalues[2*i][2*j] = vvalues[2*i][2*j+1] = vvalues[2*i+1][2*j] = vvalues[2*i+1][2*j+1] = values[i][j];
          }
        }

        // ****************************************************
        // Finally, configure the data elements inside SpeciesP
        //
        // ****************************************************

        obj.config(XX,YY,vvalues);

        return err;
      }

};

class SAV
{
    public:
        Float intercept;
        Float temp;
        Float sal;
        Float depth;

    int config(const Config& config)
    {
        int li_err = 0;
        std::string interceptName("SpeciesSAV.Intercept");
        std::string tempName     ("SpeciesSAV.Temp");
```

```
    std::string salName      ("SpeciesSAV.Sal");
    std::string depthName    ("SpeciesSAV.Depth");

    if ( config.set_var(interceptName, intercept) ) {
      std::cout << "SAV: Error: Parameter " << interceptName << "not defined" << std::endl;
      ++li_err;
    } else {
      std::cout << "SAV: intercept = " << intercept << " (" << config.find(interceptName) << ")" << std::endl;
    }

    if ( config.set_var(tempName,      temp) ) {
      std::cout << "SAV: Error: Parameter " << tempName << "not defined" << std::endl;
      ++li_err;
    } else {
      std::cout << "SAV: temp = " << temp << " (" << config.find(tempName) << ")" << std::endl;
    }

    if ( config.set_var(salName,       sal) ) {
      std::cout << "SAV: Error: Parameter " << salName << "not defined" << std::endl;
      ++li_err;
    } else {
      std::cout << "SAV: sal = " << sal << " (" << config.find(salName) << ")" << std::endl;
    }

    if ( config.set_var(depthName,     depth) ) {
      std::cout << "SAV: Error: Parameter " << depthName << "not defined" << std::endl;
      ++li_err;
    } else {
      std::cout << "SAV: depth = " << depth << " (" << config.find(depthName) << ")" << std::endl;
    }

    return li_err;
}
```

```
    inline Float cover_proportion(Float t, Float s, Float d) const
    { return( std::min(1.0,std::max(0.0,intercept + temp*t + sal*s + depth*d)) ); }

    inline Float operator()(Float t, Float s, Float d) const
    { return(cover_proportion(t,s,d));      }
};

class SpeciesLUT : public std::map<SpIndex, SpeciesInfo>, public IOS
{
    public:
        typedef std::map<SpIndex, SpeciesInfo>                BaseClass;
        typedef std::map<SpIndex, SpeciesInfo>::iterator       iterator;
        typedef std::map<SpIndex, SpeciesInfo>::const_iterator  const_iterator;

        SAV sav;

    // ########################################
    // Code involved in modeling
    //
    // ########################################
    public:
        int config(const Config& aC_config)
        {

          std::cout << "SpeciesLUT::config(...) : begin" << std::endl << std::flush;
          int li_err = 0;

          std::string lS_temp_name("Species<index>.Name");
          std::string lS_temp_index("Species<index>.Index");
          std::string lS_temp_abr("Species<index>.Abr");
          std::string lS_temp_D("Species<index>.D");
          boost::regex lC_key("<index>");
          boost::smatch results;
```

```
// First figure out how many species definitions there are.
boost::regex lC_regex_name(std::string("^Species[[:word:][:digit:]]+\\.Name"));
Config lC_names = aC_config.subset(lC_regex_name);
int li_N = lC_names.size();

// Now, get the symbols used to label each species.
std::vector<std::string> lV_sym((size_type)li_N);
int li_i = 0;
for ( Config::const_interator lC_iter = lC_names.begin(); lC_iter != lC_names.end(); ++lC_iter, ++li_i) {
  lV_sym[li_i] = lC_iter->first;
  lV_sym[li_i] =  boost::regex_replace( lV_sym[li_i], boost::regex(std::string("Species")), std::string(""));
  lV_sym[li_i] =  boost::regex_replace( lV_sym[li_i], boost::regex(std::string("\\.Name")),  std::string(""));
}

// Now, cycle over the species by symbol.
// For each species add an entry into the SpeciesLUT.
for ( int li_count = 0; li_count < li_N; ++li_count ) {
  std::cout << "Configuring species " << lV_sym[li_count] << std::endl;
  std::stringstream lC_sstream;
  lC_sstream << li_count;
  std::string lS_tag_name    = boost::regex_replace(lS_temp_name,    lC_key, lV_sym[li_count]);
  std::string lS_tag_index   = boost::regex_replace(lS_temp_index,   lC_key, lV_sym[li_count]);
  std::string lS_tag_abr     = boost::regex_replace(lS_temp_abr,     lC_key, lV_sym[li_count]);
  std::string lS_tag_D       = boost::regex_replace(lS_temp_D,       lC_key, lV_sym[li_count]);

  SpeciesInfo lC_sp;

  if ( aC_config.set_var(lS_tag_name, lC_sp.name) ) {
      std::cout << "SpeciesLUT::Config(..) : Error : species parameter name " << lS_tag_name << " is not defined\n";
      ++li_err;
  }
  if ( aC_config.set_var(lS_tag_index, lC_sp.index) ) {
      std::cout << "SpeciesLUT::Config(..) : Error : species parameter name " << lS_tag_index << " is not defined\n";
      ++li_err;
```

```
        }

        if ( aC_config.set_var(lS_tag_abr, lC_sp.abr) ) {
            std::cout << "SpeciesLUT::Config(..) : Error : species parameter name " << lS_tag_abr << " is not defined\n";
            ++li_err;
        }

        SpeciesD_config SDC; SDC.config(aC_config.find(lS_tag_D), lC_sp.D);

        BaseClass::operator[](lC_sp.index) = lC_sp;

        if ( regex_search(lC_sp.abr, results, boost::regex("SAV"))       ) { lC_sp.spi_sav   = lC_sp.index; }
        if ( regex_search(lC_sp.abr, results, boost::regex("WATER"))  ) { lC_sp.spi_water = lC_sp.index; }
      }

      // Finally, read the data in from the common establishment file.
      SpeciesP_config SPC;
      if ( SPC.config(aC_config.find("EstablishmentFile"), SpeciesInfo::P ,(BaseClass)*this) ) {
        ++li_err;
      }

      sav.config(aC_config);

      std::cout << "SpeciesLUT::config(...) : done" << std::endl;
      return li_err;
    }

// ########################################
// Basic class mechanics code
//
// ########################################
public:
    SpeciesLUT() : BaseClass() {}
    SpeciesLUT(const SpeciesLUT& aC_orig) : BaseClass( (BaseClass)aC_orig) {}
```

```
    ~SpeciesLUT() { BaseClass::clear(); }


public:
    const SpeciesInfo& operator[](SpIndex i) const
    {
      return BaseClass::at(i);
        // This is a hack. For some reason stl:map's do not
        // have a const version of the operator[]. This is
        // very odd, because they do have a const version of map::at().
        // So I'm using map::at() hear. map::at() can throw
        // an exception. It is possible that this will slow
        // the program. If so, I will fix it. Otherwise
        // I'm going to keep it this way because I think
        // this is safer.
    }

    SpeciesInfo& operator[](SpIndex i) { return BaseClass::operator[](i); }

protected:
    SpeciesLUT& operator=(const SpeciesLUT& aC_orig)
    {
      if ( this != &aC_orig ) {
        BaseClass::operator=( (BaseClass)aC_orig );
      }
      return *this;
    }

public:
    std::ostream& write(std::ostream& aC_O) const
    {
      for ( const_iterator lC_iter = BaseClass::begin(); lC_iter != BaseClass::end(); ++lC_iter ) {
        aC_O << "Species" << (lC_iter->first) << ".Name  = " << (lC_iter->second).name << std::endl;
        aC_O << "Species" << (lC_iter->first) << ".Index  = " << (lC_iter->second).index << std::endl;
```

```
            aC_O << "Species" << (lC_iter->first) << ".abr  = " << (lC_iter->second).abr << std::endl;
            aC_O << "Species" << (lC_iter->first) << ".D  = \n" << (lC_iter->second).D << std::endl;
            aC_O << std::endl;
          }

          aC_O << std::endl << std::flush;
          // aC_O << "got here\n";

          // for ( BaseClass::const_iterator lC_iter = this->begin(); lC_iter != this->end(); ++lC_iter) {
            // aC_O << "Species" << (lC_iter->first) << ".Name  = " << (lC_iter->second).name << std::endl;
            // aC_O << "Species" << (lC_iter->first) << ".Index = " << (lC_iter->second).index << std::endl;
            // aC_O << "Species" << (lC_iter->first) << ".WD    = " << (lC_iter->second).WD << std::endl;
            // aC_O << "Species" << (lC_iter->first) << ".S     = " << (lC_iter->second).S << std::endl;
            // aC_O << "Species" << (lC_iter->first) << ".N     = " << (lC_iter->second).N << std::endl;
          // }
        }
};

class Patch : public std::map<SpIndex,Float>, public IOS
{
    friend class Params;

    public:
        typedef std::map<SpIndex, Float>                BaseClass;
        typedef std::map<SpIndex, Float>::iterator       iterator;
        typedef std::map<SpIndex, Float>::const_iterator const_iterator;

    protected:
        static const Params& mC_params;

        unsigned int mi_myloc;
              int mi_CellID; // This is the cellID used in the CLEAR grid.

    public:
```

```
    inline unsigned int loc()    const { return mi_myloc;}
    inline         int CellID() const { return mi_CellID;}


public:
    int config(const std::map<SpIndex, Float>& aC_map, int ai_myloc, int ai_CellID)
    {
      BaseClass::clear();
      BaseClass::insert(aC_map.begin(), aC_map.end());
      mi_myloc = ai_myloc;
      mi_CellID = ai_CellID;
    }
    int step();

public:
    Patch() :
      mi_myloc(0),
      mi_CellID(-9999),
      BaseClass()
    {}

    Patch(const Patch& aC_orig) :
      mi_myloc(aC_orig.mi_myloc),
      mi_CellID(aC_orig.mi_CellID),
      BaseClass(aC_orig)
    {}

    ~Patch() {}

    Patch& operator=(const Patch& aC_orig)
    {
      if ( this != &aC_orig ) {
        mi_myloc = aC_orig.mi_myloc;
        mi_CellID = aC_orig.mi_CellID;
```

```
      BaseClass::operator=( (const BaseClass)aC_orig);
    }
    return *this;
  }


  std::ostream& write_csv(std::ostream& aC_O) const
  {
    aC_O << "SpIndex, CoverFraction" << std::endl;
    for ( const_iterator lC_iter = begin(); lC_iter != end(); ++lC_iter) {
      aC_O << lC_iter->first << ", " << lC_iter->second << std::endl;
    }
    return aC_O;
  }

  std::ostream& write(std::ostream& aC_O) const
  {
    aC_O << mi_CellID;
    return aC_O;

    /*
    aC_O << "[" << BaseClass::size() << "]";
    for ( const_iterator lC_iter = begin(); lC_iter != end();) {
      aC_O << " (" << lC_iter->first << ", " << lC_iter->second << ")";
      ++lC_iter;
      if ( lC_iter != end() ) aC_O << ",";
    }
    return aC_O;
    */
  }

};

class AsciiGridPlusReader
```

```
{
    public:
        int read_asc(std::istream& strm, Landscape<int>& LocID)
        {
          AsciiGridReader reader;
          reader(strm, LocID);
          return 0;
        }
        int read_csv(std::istream& strm, AGPTable& table)
        {
          std::vector<std::string> header;
          std::string line;
          getline(strm, line);
          CSVRowToVec(line, header); // Read off the header
          std::vector<Float> blank();

          std::vector<Float> row;
          int             CellID; // CellID in the CellID_map
          int             count  = 0;
          while ( getline(strm, line) ) {
            CSVRowToVec(line, row);
            CellID = row[0];
            table[CellID].assign( ++row.begin(), row.end() );
            if ( !((count++)%10000) ) { std::cout << count << std::endl; }
          }

          return 0;
        }
        int build_agp(Landscape<int>& CellID_map, AGPTable& table, Landscape< std::vector<Float> >& obj)
        {
          std::vector<Float> nodatavalue;
          obj.resize( CellID_map.nrows(),  CellID_map.ncols(), CellID_map.yllcorner(), CellID_map.xllcorner(), CellID_map.cellsize(), nodatavalue);
          int cellid = 0;
          for ( int r = 0; r < obj.nrows(); ++r ) {
```

```
    for ( int c = 0; c < obj.ncols(); ++c ) {
        cellid = CellID_map(r,c);
        if ( cellid != CellID_map.nodatavalue() ) {
            obj(r,c) = table[cellid];
        }
    }
  }
  return 0;
}

int operator()(std::string asc_file, std::string csv_file, Landscape< std::vector<Float> >& obj)
{
  int li_err = 0;

  std::ifstream asc_strm;
  std::ifstream csv_strm;

  asc_strm.open(asc_file.c_str());
  if ( !asc_strm.is_open() ) {
    std::cout << "AsciiGridReader: Error: Could not open asc file " << asc_file << std::endl;
    ++li_err;
  }

  csv_strm.open(csv_file.c_str());
  if ( !csv_strm.is_open() ) {
    std::cout << "AsciiGridReader: Error: Could not open csv file " << csv_file << std::endl;
    ++li_err;
  }

  if ( li_err ) return li_err;

  Landscape<int> CellID_map;
  AGPTable      table;
  read_asc(asc_strm, CellID_map);
```

```
        read_csv(csv_strm, table);

        build_agp(CellID_map, table, obj);
      }

      int operator()(std::istream& strm, Landscape< std::vector<Float> >& obj)
      {
        Landscape<int> CellID_map;
        AGPTable      table;
        read_asc(strm, CellID_map);
        read_csv(strm, table);

        build_agp(CellID_map, table, obj);
      }


  public:
      explicit AsciiGridPlusReader() {}
  protected:
      explicit AsciiGridPlusReader(const AsciiGridPlusReader& ) {}
      AsciiGridPlusReader& operator=(const AsciiGridPlusReader& ) {}
};

class InitCondReader // public AsciiGridPlusReader
{
  public:
      int operator()(std::istream& strm, Landscape< Patch >& obj, const SpeciesLUT& SpLUT)
      {
        int err = 0;
        std::cout << "InitCondReader : begin" << std::endl;

        // This function reads the initialization data from an asciigrid+ file.
        // The asciigrid+ file contains 2 parts. The first part of the file
        // is a standard asciigrid formatted data set. However, the values
```

```
// in the asciigrid data are cell reference numbers. The second
// half of the file is a standard asciigrid formatted data set.
// This data containes the actual values for the cells in the asciigrid data.
// There is one row of data in the CSV file for each index in the asciigrid.
// The first column of each row is the grid cell reference number.
// The rest of the row is the data for the cell.
//
// To read the file as quickly as possible, I need to build a couple of
// look up tables (LUTs). The first converts from the cell reference number
// to the index of each cell in the Landscape object that will hold the
// data. This makes cross referencing from the asciigrid data to the
// Landscape object fast. The second table converts from the column
// names in the CSV data to the internal species ID used by the model.
// Current, the column names are the species abreviation listed in the
// SpeciesInfo structure read from the configuation file.
//
// Once the LUTs are built, I read a row of data from the CSV, get the
// Landscape index associated with the cell reference number, then
// Pack the data into a stl map and push that into the appropriate location
// in the Landscape object.

// These are the only local variable that persist across steps.
int                     li_err = 0;
std::map<std::string, int> SpName2SpIndex;
std::vector<int>          locLUT;
std::vector<std::string>   header;

// Step 1: Build SpName to SpIndex Look-up table (LUT).
std::cout << "InitCondReader : building Name to Index LUT" << std::endl;
for ( SpeciesLUT::const_iterator iter = SpLUT.begin(); iter != SpLUT.end(); ++iter ) {
  SpName2SpIndex[ (iter->second).abr ] = iter->first;
}
std::cout << "InitCondReader : results = " << std::endl;
std::cout << SpName2SpIndex << std::endl;
```

```
// Step 2: Read in the asciigrid data and build the location LUT, locLUT.
{
std::cout << "InitCondReader : Reading Ascigrid data" << std::endl;
Landscape<int>          CellID_map;
AsciiGridReader         agreader;
agreader(strm, CellID_map);

std::cout << "InitCondReader : Building location LUT." << std::endl;
// I need an object to convert from a gridID to the "address" in the Landscape object.
// gridID's are integers, so I can use a vec.
int gridID_max = max(CellID_map);
locLUT.resize(gridID_max+1,-1); // +1 because C++ vectors count from zero. But the gridID's count from 1.
for ( int i = 0; i < CellID_map.size(); ++i ) {
  if ( CellID_map[i] != CellID_map.nodatavalue() ) {
      locLUT[ CellID_map[i] ] = i;
  }
}

// Step 3: Resize the target Landscape<Patch> object.
std::cout << "InitCondReader : Resizing Landscape<Patch> object" << std::endl;
Patch nodatavalue;
obj.resize( CellID_map.nrows(),  CellID_map.ncols(), CellID_map.yllcorner(), CellID_map.xllcorner(), CellID_map.cellsize(), nodatavalue);
}

// Step 4: Read the CSV header and make sure it is correct.
{
std::cout << "InitCondReader : Reading CSV header" << std::endl;
std::string line;
getline(strm, line);
CSVRowToVec(line, header);

std::cout << "InitCondReader : Comparing Header to SpeciesLUT info." << std::endl;
for ( int i = 1; i < header.size(); ++i ) {
```

```
  if ( SpName2SpIndex.find(header[i]) == SpName2SpIndex.end() ) {
      std::cout << "InitCondReader : Column name " << header[i] << " is not in the SpeciesLUT." << std::endl;
      ++err;
  } else { std::cout << "InitCondReader : Column name " << header[i] << " found." << std::endl;}
}
}

// Step 5: Read the CSV data and push it into the Landscape<Patch> object.

{
std::cout << "InitCondReader : Reading CSV data" << std::endl;
std::string          line;
std::vector<Float>      row;
std::map<SpIndex, Float> temp;
int                  CellID; // CellID in the CellID_map
int                  loc; // Landscape location
while ( getline(strm, line) ) {
  CSVRowToVec(line, row);
  CellID = row[0];
  if ( locLUT[CellID] == - 1 ) {
      std::cout << "InitCondReader : Cell " << CellID << " defined in table but not in grid" << std::endl;
      ++err;
      continue;
  }
  loc    = locLUT[CellID]; // row[0] contains the gridID for this row of data.
  obj[loc].clear();
  for ( int i = 1; i < row.size(); ++i) {
      // std::cout << "row[0] = " << row[0] << std::endl;
      // std::cout << "row[i] = " << row[i] << std::endl;
      // std::cout << "header[i] = " << header[i] << std::endl;
      // std::cout << "SpName2SpIndex[header[i]] = " << SpName2SpIndex[header[i]] << std::endl;
      temp[ SpName2SpIndex[header[i]] ] = row[i];
  }
  obj[loc].config(temp,loc,CellID);
```

```
            // std::cout << "obj[loc] = " << obj[loc] << std::endl;
          }
          }

          std::cout << "InitCondReader : done" << std::endl;
          // std::cout << obj << std::endl;
          return err;
        }

    public:
        explicit InitCondReader() {}
    protected:
        explicit InitCondReader(const InitCondReader& ) {}
        InitCondReader& operator=(const InitCondReader& ) {}
};

class AsciiGridPlusWriter
{
    public:
    int write_asciigrid(std::ostream& strm, Landscape<Patch>& obj, const SpeciesLUT& SpLUT)
    {
        //##############################################
        // Write the asciigrid data
        //
        //##############################################
        strm << "ncols " << obj.ncols() << std::endl;
        strm << "nrows " << obj.nrows() << std::endl;
        strm << "xllcorner " << obj.xllcorner() << std::endl;
        strm << "yllcorner " << obj.yllcorner() << std::endl;
        strm << "cellsize  " << obj.cellsize()  << std::endl;
        strm << "NODATA_value " << -9999 << std::endl;
        for ( int li_row = 0; li_row < obj.nrows(); ++li_row ) {
          for ( int li_col = 0; li_col < obj.ncols(); ++li_col ) {
            strm << obj(li_row, li_col).CellID();
```

```
        if ( (li_col + 1) < obj.ncols() ) { strm << " "; }
      }
      strm << std::endl;
    }

    return 0;
}
int write_csv(std::ostream& strm, Landscape<Patch>& obj, const SpeciesLUT& SpLUT)
{
    //##############################################
    // Write the header for the CSV data
    //
    //##############################################
    strm << "CELLID";
    for ( SpeciesLUT::const_iterator lC_iter = SpLUT.begin(); lC_iter != SpLUT.end(); ++lC_iter ) {
      strm << "," << (lC_iter->second).abr;
    }
    strm << std::endl;

    //##############################################
    // Write the CSV data
    //
    //##############################################
  for( Landscape<Patch>::iterator l_liter = obj.begin(); l_liter != obj.end(); ++l_liter ) {
    if ( l_liter->CellID() != -9999 ) {
      strm << l_liter->CellID();
      for ( Patch::iterator l_piter = l_liter->begin(); l_piter != l_liter->end(); ++l_piter) {
        strm << ", " << l_piter->second;
      }
      strm << std::endl;
    }
  }

    return 0;
```

```
    }
    int operator()(std::ostream& strm, Landscape<Patch>& obj, const SpeciesLUT& SpLUT)
    {
        int err = 0;
        err += write_asciigrid(strm, obj, SpLUT);
        err += write_csv(strm, obj, SpLUT);
        return err;
    }
};

// PreProc is defined per post-processesing file.
class PreProc
{
    private:
        enum TestState {OK = 0, OK_GO = 1, ERROR = 2};

        static Landscape<int> BoxIDs;
        static Landscape<int> CellIDs;

    public:
        typedef std::map<std::string, std::string> Map;

    public:
        Map ftpParamNames;
        Map preParamNames;
        Map postParamNames;

   protected:

        int do_preProc(const Config& config) const
        {
          TestState ret = preProcTest(config);
          if ( ret == ERROR ) { return 1;}
          if ( ret == OK )    { return 0;}
```

```
    if ( ret == OK_GO)  { return this->preProc(config);}
    return 1;
  }

  int do_ftp(const Config& config) const
  {
    TestState ret = ftpTest(config);
    if ( ret == ERROR ) { return 1;}
    if ( ret == OK    ) { return 0;}
    if ( ret == OK_GO ) { return ftp(config);}
    return 1;
  }


int preProcTest_openFiles(const Config& config, const Map& map, std::ios_base::openmode mode, std::stringstream& errmsg) const
{
  int err = 0;
  for ( Map::const_iterator iter = map.begin(); iter != map.end(); ++iter ) {
    std::string filename;
    std::fstream strm;
    config.set_var( iter->second, filename);
    strm.open(filename.c_str(), mode);
    if ( !strm.is_open() ) {
      errmsg << "PreProc: Error: Could not open file: filename = " << filename << std::endl;
      ++err;
    }
    strm.close();
  }
  return err;
}

  time_t get_filemtime(std::string& filename) const
  {
    struct stat attrib;
```

```
    stat(filename.c_str(), &attrib);
    return attrib.st_mtime;
  }

int preProcTest_getTimes(const Config& config, const Map& map, std::map<std::string, time_t>& times) const
{
  std::string filename;
  config.set_var( map.begin()->second, filename);
  times[std::string("newest")] = get_filemtime(filename);
  times[std::string("oldest")] = get_filemtime(filename);
  for ( Map::const_iterator iter = map.begin(); iter != map.end(); ++iter ) {
     config.set_var( iter->second, filename);
     times[ iter->first ] = get_filemtime(filename);
     times[std::string("newest")] = std::max( times[std::string("newest")], times[iter->first] );
     times[std::string("oldest")] = std::min( times[std::string("oldest")], times[iter->first] );
  }
  return 0;
}

  TestState preProcTest(const Config& config) const
  {
    // Pre-processesing Test
    // Step 1: Can the pre-processing files be openned for reading?
  std::cout << "\tChecking preprocess files" << std::endl;
  std::stringstream errmsg;
  if ( preProcTest_openFiles(config, preParamNames, std::ios_base::in, errmsg) ){
       // If no, return ERROR.
     std::cout << errmsg.str();
     return ERROR;
  }
  std::cout << "\tPreprocess file exist and are readable" << std::endl;

    // Step 2: Get the creation date of the pre-processing files.
  std::map<std::string, time_t> preTimes;
```

```
preProcTest_getTimes(config, preParamNames, preTimes);

  // Step 3: Do the post-processing files exist?
std::cout << "\tChecking postprocess files" << std::endl;
errmsg.clear();
if ( !preProcTest_openFiles(config, postParamNames, std::ios_base::in, errmsg) ) {
      // If yes, get the creation date
    std::cout << "\tPostprocess files exist and are readable" << std::endl;
    std::map<std::string, time_t> postTimes;
    preProcTest_getTimes(config, postParamNames, postTimes);

      // Step 4: Are the pre-processing files newer?
    std::cout << "\tComparing modification times" << std::endl;
    if( preTimes["newest"] > postTimes["oldest"] ) {
          // Yes -> OK_GO, make post-processing files.
        std::cout << "\tPreprocess files are newer, will attempt to recreate postprocess file" << std::endl;
        return OK_GO; // Here it goes
                    // here it goes
                    // here it goes again...

    } else {
          // No  -> return OK. everything is ok, no need to run the pre-processor.
        std::cout << "\tPostprocess files are new, skipping preprocessing process" << std::endl;
        return OK;
    }

} else {
      // If no (step 3), can it be openned for writting.
    std::cout << "\tPostprocess files do not exist." << std::endl;
    errmsg.clear();
    if ( preProcTest_openFiles(config, postParamNames, std::ios_base::out, errmsg) ) {
          // no -> return ERROR.
        std::cout << errmsg.str();
        std::cout << "PreProc: Error: postprocess files could not be openned for writting." << std::endl;
```

```
        return ERROR;

    } else {
          // yes -> OK_GO, proceed to make post-processing files.
        std::cout << "\tAttempting to create postprocess files" << std::endl;
        return OK_GO; // So you were born
                    // in an electrical storm,
                    // took a bite out the sun
                    // and saw your future in
                    // a machine built for two.
    }
}

std::cout << "PreProc: Error: Something untoward occured, in PreProc::preProcTest(...). Please consult your local code monkey." << std::endl;
  return ERROR;
}


TestState ftpTest(const Config& config) const
{
  // Download Test
  // Step 1: Does the FTP file exist?
  //    No ->  return ERROR
  //    Yes -> Get the date of the FTP file.
  // Step 2: Does the Pre-processing file exist?
  //    If no, can it be openned for writting?
  //       No  -> return ERROR.
  //       Yes -> OK_GO, go ahead and download the files.
  //    If yes, get the creation date
  // Step 3: Is the FTP file newer than the pre-processing file?
  //    No  -> return OK, Everything is OK, no need to download
  //    Yes -> return OK_GO, go ahead and download the files.

  return OK;
```

```
    }

    int ftp(const Config& config) const
    {
      return 0;
    }


    virtual int preProc(const Config& config) const =0;

    // Make post-processing files.
    // Make annual wave amp file
    //    Input: EcoHydroBoxID, EcoHydroStage, compute yearly avg from daily data.
    // Make annual salinity file
    //    Input: EcoHydroBoxID, EcoHydroSalinity, compute yearly avg from monthly data.
    // Make summer avg salinity file
    //    Input: EcoHydroBoxID, EcoHydroSalinity, compute 4-month summer avg from monthly data.
    // Make summer avg temp file
    //    Input: EcoHydroBoxID, EcoHydroTemp, compute 4-month summer avg from monthly data.
    // Make summer avg depth file
    //    Input: EcoHydroBoxID, EcoHydroStg, SoilMorphCellID, SoilMorphElev, compute 4-month summer avg from daily data.
    // Make Land/Water file
    //    Input: SoilMorphCellID, SoilMorphLandWater, compute landwater


public:
    int go(const Config& config) const
    {
      // Step 1: Make sure the post-process file parameters have been defined.
      int li_err = 0;
      for ( Map::const_iterator lC_iter = postParamNames.begin(); lC_iter != postParamNames.end(); ++lC_iter ) {
        if ( !config.isDefined( lC_iter->second ) ) {
            std::cout << "PreProc: Error: file parameter " << lC_iter->second << " not defined in config file" << std::endl;
            ++li_err;
```

```
  }
}
if ( li_err ) return li_err;

// Step 2: See how many FTP files have been defined.
int li_nftp = 0;
for ( Map::const_iterator lC_iter = ftpParamNames.begin(); lC_iter != ftpParamNames.end(); ++lC_iter ) {
  if ( config.isDefined( lC_iter->second ) ) {
      ++li_nftp;
  }
}

// Step 3: See how many pre-processing files have been defined.
int li_npre = 0;
for ( Map::const_iterator lC_iter = preParamNames.begin(); lC_iter != preParamNames.end(); ++lC_iter ) {
  if ( config.isDefined( lC_iter->second ) ) {
      ++li_npre;
  }
}

// Step 4: If there are no ftp and no pre-processing files, we're done here.
// return without error. It's all up to Params now.
if ( !li_nftp && !li_npre ) {
  return 0;
}

// Step 5: If there are no ftp files and all the preproc files are defined, then go ahead with the
// pre_proc step.
if ( li_nftp == 0 && li_npre == preParamNames.size() ) {
  return do_preProc(config);
}

// Step 6: If all the ftp files have been defined and all the preproc files are defined,
// then do the ftp step followed by the preproc step.
```

```
  if ( li_nftp == ftpParamNames.size() && li_npre == preParamNames.size() ) {
    if ( do_ftp(config) ) { return 1;}
    return do_preProc(config);
  }

  // Step 7: If some but not all of the pre-processing files have been defined
  // this is an error. Report and return error.
  if ( li_npre != preParamNames.size() ) {
    for ( Map::const_iterator lC_iter = preParamNames.begin(); lC_iter != preParamNames.end(); ++lC_iter )
        if ( !config.isDefined( lC_iter->second ) ) {
            std::cout << "PreProc: Error: file parameter " << lC_iter->second << " not defined in config file while relate pre-processing parameters are
  defined." << std::endl;
            ++li_err;
        }
  }

  // Step 8: if we have reached this point, something has gone wrong.
  // Report some errors and return.
  if ( li_nftp != ftpParamNames.size() ) {
    for ( Map::const_iterator iter = ftpParamNames.begin(); iter != ftpParamNames.end(); ++iter )
        if ( !config.isDefined( iter->second ) ) {
            std::cout << "PreProc: Error: file parameter " << iter->second << " not defined in config file while ftp relate parameters are defined." <<
  std::endl;
            ++li_err;
        }
  }

  return li_err;
}

int report_filenames(const Config& config, const Map& map) const
{
  for ( Map::const_iterator iter = map.begin(); iter != map.end(); ++iter ) {
    std::cout << "\t  " << iter->second << " = " << config.find(iter->second) << std::endl;
```

```
        }
        return 0;
      }

      int report_filenames(const Config& config) const
      {
        std::cout << "\tPreProcess files are:" << std::endl;
        report_filenames(config, preParamNames);
        std::cout << "\tPostProcess files are:" << std::endl;
        report_filenames(config, postParamNames);
        return 0;
      }

};

class preProcWaveAmp : public PreProc
{
   public:
     preProcWaveAmp()
     {
       ftpParamNames[std::string("FTP.EcoHydroStgHeightFile")]    = std::string("FTP.EcoHydroStgHeightFile");

       preParamNames[std::string("PreProc.EcoHydroStgHeightFile")] = std::string("PreProc.EcoHydroStgHeightFile");
       preParamNames[std::string("PreProc.EcoHydroBoxIDFile")]     = std::string("PreProc.EcoHydroBoxIDFile");

       postParamNames[std::string("WaveAmplitudeFile")] = std::string("WaveAmplitudeFile");
     }
   protected:
       int preProc(const Config& config) const
       {
         int li_err = 0;
       std::cout << "\tRunning wave amp file preprocessor" << std::endl;
       std::string filename;
       std::string filename_bi;
```

```
std::string filename_eh;

std::ifstream istrm_bi;
std::ifstream istrm_eh;

config.set_var( postParamNames.find(std::string("WaveAmplitudeFile"))->second, filename);
config.set_var( preParamNames.find(std::string("PreProc.EcoHydroStgHeightFile"))->second, filename_eh);
config.set_var( preParamNames.find(std::string("PreProc.EcoHydroBoxIDFile"))->second, filename_bi);

istrm_bi.open( filename_bi.c_str());
  if ( !istrm_bi.is_open() ) {
    std::cout << "PreProcWaveAmp::preProc(): Error: Could not open EcoHydro BoxID file " << filename_bi << ". This is odd." << std::endl;
    ++li_err;
  }
istrm_eh.open( filename_eh.c_str());
  if ( !istrm_bi.is_open() ) {
    std::cout << "PreProcWaveAmp::preProc(): Error: Could not open EcoHydro stage height file " << filename_bi << ". This is odd." << std::endl;
    ++li_err;
  }

  std::ofstream ofstrm;
  ofstrm.open(filename.c_str());
  if ( !ofstrm.is_open() ) {
    std::cout << "Params::preproc_mkWaveAmp() : Error : Could not open wave amplitude file for writting:  " << filename << std::endl;
    ++li_err;
  }

  if ( li_err ) {
    istrm_bi.close();
    istrm_eh.close();
    ofstrm.close();
    return li_err;
  }
```

```
  istrm_bi.seekg(0, std::ios_base::beg);
  Landscape<int> BoxID;
  AsciiGridReader agreader;
  if ( agreader(istrm_bi, BoxID) ) {
    std::cout << "Params::preproc_mkWaveAmp(): Error: Could not read BoxID." << std::endl;
    ++li_err;
  }
  istrm_bi.close();

  if ( li_err ) {
    istrm_eh.close();
    ofstrm.close();
    return li_err;
  }

  report_filenames(config);

  EcoHydroReader                                    ehreader;
int                    NumYears;
  AsciiGridWriter              agwriter;
  Landscape< Float>            l_waveAmplitude;
  Landscape< std::vector<Float> > Hydro;



config.set_var("NumYears", NumYears);

  for ( int year = 0; year < NumYears; ++year ) {
    li_err += ehreader(istrm_eh, Hydro , BoxID, 365 );
    l_waveAmplitude.resize(Hydro.nrows(), Hydro.ncols(), Hydro.yllcorner(), Hydro.xllcorner(), Hydro.cellsize(), Float(-9999));
    for ( int i = 0; i < Hydro.size(); ++i) {
      if ( BoxID[i] != BoxID.nodatavalue() ) {
          Float meanWD = 0;
```

```
                Float WDsq   = 0;
                for ( int tick = 0; tick < 365; ++tick ) {
                    meanWD = meanWD + Hydro[i][tick];
                    WDsq   = WDsq   + Hydro[i][tick]*Hydro[i][tick];
                }
                meanWD /= 365.0;
                WDsq   /= 365.0;
                l_waveAmplitude[i] = sqrt(WDsq - meanWD*meanWD);
            } else {
               l_waveAmplitude[i] = BoxID.nodatavalue();
            }
          }

          ofstrm << "# Year = " << year << std::endl;
          li_err += agwriter(ofstrm, l_waveAmplitude);
        }

        istrm_eh.close();
        ofstrm.close();

        return li_err;
      }

};

class preProcMeanSalinity : public PreProc
{
   public:
      preProcMeanSalinity()
      {
         ftpParamNames[std::string("FTP.EcoHydroSalinityFile")]     = std::string("FTP.EcoHydroSalinityFile");

         preParamNames[std::string("PreProc.EcoHydroSalinityFile")] = std::string("PreProc.EcoHydroSalinityFile");
         preParamNames[std::string("PreProc.EcoHydroBoxIDFile")]    = std::string("PreProc.EcoHydroBoxIDFile");
```

```
      postParamNames[std::string("MeanSalinityFile")]          = std::string("MeanSalinityFile");
   }

protected:
   int preProc(const Config& config) const
    {
       int li_err = 0;
      std::string filename;
      std::string filename_bi;
      std::string filename_es;

      std::ifstream istrm_bi;
      std::ifstream istrm_es;

      config.set_var( postParamNames.find(std::string("MeanSalinityFile"))->second, filename);
      config.set_var( preParamNames.find(std::string("PreProc.EcoHydroSalinityFile"))->second, filename_es);
      config.set_var( preParamNames.find(std::string("PreProc.EcoHydroBoxIDFile"))->second, filename_bi);

      istrm_es.open(filename_es.c_str());
        if ( !istrm_es.is_open() ) {
          std::cout << "PreProcMeanSalinity::preProc(): Error: Could not open EcoHydro salinity file " << filename_es << ". This is odd." << std::endl;
          ++li_err;
        }
      istrm_bi.open(filename_bi.c_str());
        if ( !istrm_bi.is_open() ) {
          std::cout << "PreProcMeanSalinity::preProc(): Error: Could not open EcoHydro BoxID file " << filename_bi << ". This is odd." << std::endl;
          ++li_err;
        }

        std::ofstream ofstrm;
        ofstrm.open(filename.c_str());
        if ( !ofstrm.is_open() ) {
          std::cout << "PreProcMeanSalinity::preProc(): Error : Could not open mean salinity file for writting:  " << filename << std::endl;
```

```
    ++li_err;
  }

  if ( li_err ) {
    istrm_es.close();
    istrm_bi.close();
    ofstrm.close();
    return li_err;
  }

  EcoHydroReader  ehreader;
  AsciiGridWriter agwriter;

  istrm_bi.seekg(0, std::ios_base::beg);
  Landscape<int> CellID;
  AsciiGridReader agreader;
  if ( agreader(istrm_bi, CellID) ) {
    std::cout << "PreProcMeanSalinity::preProc(): Error: Could not read BoxID data." << std::endl;
    ++li_err;
  }
  istrm_bi.close();
  if ( li_err ) {
    istrm_es.close();
    ofstrm.close();
    return li_err;
  }


  report_filenames(config);

  Landscape< std::vector<Float> > S;
  Landscape< Float > l_meanSalinity;
int NumYears;
config.set_var("NumYears", NumYears);
```

```
        for ( int year = 0; year < NumYears; ++year ) {
          ehreader(istrm_es, S, CellID, 12);
          l_meanSalinity.resize(S.nrows(), S.ncols(), S.yllcorner(), S.xllcorner(), S.cellsize(), Float(-9999));
          for ( int i = 0; i < S.size(); ++i) {
            if ( CellID[i] != CellID.nodatavalue() ) {
                  Float meanS = 0;
                  for ( int tick = 0; tick < 12; ++tick ) {
                       meanS = meanS + S[i][tick];
                  }
                  meanS /= 12.0;
                  l_meanSalinity[i] = meanS;
            } else {
               l_meanSalinity[i] = CellID.nodatavalue();
            }
          }
          ofstrm << "# Year = " << year << std::endl;
          li_err += agwriter(ofstrm, l_meanSalinity);
        }

        istrm_es.close();
        ofstrm.close();
        return li_err;
      }

};

class preProcLandWater : public PreProc
{
   public:
      preProcLandWater()
      {
         ftpParamNames[std::string("FTP.WetlandMorphLandWaterFile")]    = std::string("FTP.WetlandMorphLandWaterFile");

         preParamNames[std::string("PreProc.WetlandMorphCellIDFile")]    = std::string("PreProc.WetlandMorphCellIDFile");
```

```
    preParamNames[std::string("PreProc.WetlandMorphLandWaterFile")]     = std::string("PreProc.WetlandMorphLandWaterFile");

    postParamNames[std::string("LandWaterFile")] = std::string("LandWaterFile");
  }

protected:
  int preProc(const Config& config) const
    {
      int li_err = 0;
    std::string   filename;
    std::string   filename_lw;
    std::string   filename_ci;
    std::ifstream istrm_asc;
    std::ifstream istrm_csv;
      std::ofstream ofstrm;

    config.set_var( postParamNames.find(std::string("LandWaterFile"))->second,filename);
    config.set_var( preParamNames.find(std::string("PreProc.WetlandMorphCellIDFile"))->second,filename_ci);
    config.set_var( preParamNames.find(std::string("PreProc.WetlandMorphLandWaterFile"))->second,filename_lw);

    istrm_asc.open(filename_ci.c_str());
      if ( !istrm_asc.is_open() ) {
        std::cout << "PreProcLandWater::preProc(): Error: Could not open Wetland Morph Cell ID file " << filename_ci << ". This is odd." << std::endl;
        ++li_err;
      }

    istrm_csv.open(filename_lw.c_str());
      if ( !istrm_asc.is_open() ) {
        std::cout << "PreProcLandWater::preProc(): Error: Could not open Wetland Morph Land/Water file " << filename_lw << ". This is odd." <<
      std::endl;
        ++li_err;
      }

      ofstrm.open(filename.c_str());
```

```
if ( !ofstrm.is_open() ) {
  std::cout << "PreProcLandWater::preProc(): Error : Could not open Veg Model land/water file for writting:  " << filename << std::endl;
  ++li_err;
}

if ( li_err ) {
  istrm_asc.close();
  istrm_csv.close();
  ofstrm.close();
  return li_err;
}

Landscape<int>     CellID_map;
AGPTable           table;
AsciiGridPlusReader agpr;

istrm_asc.seekg(0, std::ios_base::beg);
istrm_csv.seekg(0, std::ios_base::beg);
if ( agpr.read_asc(istrm_asc, CellID_map) ) {
  std::cout << "PreProcLandWater::preProc(): Error: Could not read Wetland Morph Cell ID data." << std::endl;
  ++li_err;
}
std::cout << "Here" << std::endl;
if ( agpr.read_csv(istrm_csv, table) ) {
  std::cout << "PreProcLandWater::preProc(): Error: Could not read Wetland Morph land/water data." << std::endl;
  ++li_err;
}
istrm_asc.close();
istrm_csv.close();

if ( li_err ) {
  ofstrm.close();
  return li_err;
}
```

```
        report_filenames(config);

        std::cout << "Here" << std::endl;
        Landscape<Float> l_landWater;
        l_landWater.resize( CellID_map.nrows(),  CellID_map.ncols(), CellID_map.yllcorner(), CellID_map.xllcorner(), CellID_map.cellsize(), -9999);
        int cellid = 0;
        for ( int i = 0; i < l_landWater.size(); ++i ) {
            cellid = CellID_map[i];
            if ( cellid != CellID_map.nodatavalue() ) {
                l_landWater[i] = table[cellid][2] ? table[cellid][1]/table[cellid][2] : 0;
            } else {
                l_landWater[i] = CellID_map.nodatavalue();
            }
        }

        AsciiGridWriter agw;
        li_err += agw(ofstrm, l_landWater);
        ofstrm.close();

        return li_err;
      }

};


class preProcElevation : public PreProc
{
   public:
      preProcElevation()
      {
         ftpParamNames[std::string("FTP.WetlandMorphElevationFile")]    = std::string("FTP.WetlandMorphElevationFile");

         preParamNames[std::string("PreProc.WetlandMorphCellIDFile")]     = std::string("PreProc.WetlandMorphCellIDFile");
```

```
    preParamNames[std::string("PreProc.WetlandMorphElevationFile")]     = std::string("PreProc.WetlandMorphElevationFile");

    postParamNames[std::string("ElevationFile")] = std::string("ElevationFile");
  }

protected:
  int preProc(const Config& config) const
    {
    std::string filename;
    std::string filename_el;
    std::string filename_ci;
    std::ifstream istrm_asc;
    std::ifstream istrm_csv;

    config.set_var( postParamNames.find(std::string("ElevationFile"))->second,filename);
    config.set_var( preParamNames.find(std::string("PreProc.WetlandMorphCellIDFile"))->second,filename_ci);
    config.set_var( preParamNames.find(std::string("PreProc.WetlandMorphElevationFile"))->second,filename_el);

      int li_err = 0;
    istrm_asc.open(filename_ci.c_str());
      if ( !istrm_asc.is_open() ) {
        std::cout << "PreProcElevation::preProc(): Error: Could not open Wetland Morph CellID file " << filename_ci << ". This is odd." << std::endl;
        ++li_err;
      }
    istrm_csv.open(filename_el.c_str());
      if ( !istrm_asc.is_open() ) {
        std::cout << "PreProcElevation::preProc(): Error: Could not open Wetland Morph elevation file " << filename_el << ". This is odd." << std::endl;
        ++li_err;
      }
      std::ofstream ofstrm;
      ofstrm.open(filename.c_str());
      if ( !ofstrm.is_open() ) {
        std::cout << "PreProcElevation::preProc(): Error : Could not open elevation file for writting:  " << filename << std::endl;
        ++li_err;
```

```
}


if ( li_err ) {
  istrm_asc.close();
  istrm_csv.close();
  ofstrm.close();
  return li_err;
}

AsciiGridPlusReader agpr;
Landscape<int> CellID_map;
AGPTable  table;

istrm_asc.seekg(0, std::ios_base::beg);
istrm_csv.seekg(0, std::ios_base::beg);
if ( agpr.read_asc(istrm_asc, CellID_map) ) {
  std::cout << "PreProcElevation::preProc(): Error: Could not read CellID data." << std::endl;
  ++li_err;
}
if ( agpr.read_csv(istrm_csv, table) ) {
  std::cout << "PreProcElevation::preProc(): Error: Could not read elevation data." << std::endl;
  ++li_err;
}
istrm_asc.close();
istrm_csv.close();

if ( li_err ) {
  ofstrm.close();
  return li_err;
}

report_filenames(config);
```

```
        Landscape<Float> l_elv;
        l_elv.resize( CellID_map.nrows(),  CellID_map.ncols(), CellID_map.yllcorner(), CellID_map.xllcorner(), CellID_map.cellsize(), -9999);
        int cellid = 0;
        for ( int i = 0; i < l_elv.size(); ++i ) {
          cellid = CellID_map[i];
          if ( cellid != CellID_map.nodatavalue() ) {
               l_elv[i] = table[cellid][0];
          } else {
               l_elv[i] = CellID_map.nodatavalue();
          }
        }

        AsciiGridWriter agw;
        li_err += agw(ofstrm, l_elv);
        ofstrm.close();

        return li_err;
      }

};

class preProcSummerMeanDepth: public PreProc
{
   public:
      preProcSummerMeanDepth()
      {
        ftpParamNames.clear();

        preParamNames["PreProc.EcoHydroBoxIDFile"]     = std::string("PreProc.EcoHydroBoxIDFile");
        preParamNames["PreProc.EcoHydroStgHeightFile"] = std::string("PreProc.EcoHydroStgHeightFile");
        preParamNames["ElevationFile"]                 = std::string("ElevationFile");

        postParamNames["SummerMeanWaterDepthFile"]     = std::string("SummerMeanWaterDepthFile");
      }
```

```
protected:
  int preProc(const Config& config) const
  {
      int li_err = 0;
      std::string filename_out;
      std::string filename_bi;
      std::string filename_eh;
      std::string filename_el;
      std::ofstream strm_out;
      std::ifstream strm_eh;
      std::ifstream strm_bi;
      std::ifstream strm_el;

      config.set_var( postParamNames.find("SummerMeanWaterDepthFile")->second, filename_out);
      config.set_var( preParamNames.find("PreProc.EcoHydroBoxIDFile")->second, filename_bi);
      config.set_var( preParamNames.find("PreProc.EcoHydroStgHeightFile")->second, filename_eh);
      config.set_var( preParamNames.find("ElevationFile")->second, filename_el);

      strm_out.open(filename_out.c_str());
      if ( !strm_out.is_open() ) {
        std::cout << "PreProcSummerMeanDepth::preProc(): Error: Could not open summer mean depth file " << filename_out << std::endl;
        ++li_err;
      }

      strm_eh.open(filename_eh.c_str() );
      if ( !strm_eh.is_open() ) {
        std::cout << "PreProcSummerMeanDepth::preProc(): Error: Could not open EcoHydro stage height file " << filename_eh << std::endl;
        ++li_err;
      }

      strm_bi.open(filename_bi.c_str() );
      if ( !strm_bi.is_open() ) {
        std::cout << "PreProcSummerMeanDepth::preProc(): Error: Could not open EcoHydro box ID file " << filename_bi << std::endl;
```

```
  ++li_err;
}

strm_el.open(filename_el.c_str() );
if ( !strm_el.is_open() ) {
  std::cout << "PreProcSummerMeanDepth::preProc(): Error: Could not open elevation file " << filename_el << std::endl;
  ++li_err;
}

if ( li_err ) {
  strm_out.close();
  strm_eh.close();
  strm_bi.close();
  strm_el.close();
}

strm_bi.seekg(0, std::ios_base::beg);
Landscape<int> BoxID;
AsciiGridReader agreader;
if ( agreader(strm_bi, BoxID) ) {
  std::cout << "PreProcMeanSummerWaterDepth::preProc(): Error: Could not read boxID data." << std::endl;
  ++li_err;
}
strm_bi.close();

Landscape<Float> Elv;
if ( agreader(strm_el, Elv) ) {
  std::cout << "PreProcMeanSummerWaterDepth::preProc(): Error: Could not read elevation data." << std::endl;
  ++li_err;
}
strm_el.close();

if ( li_err ) {
  strm_eh.close();
```

```
    strm_out.close();
    return li_err;
  }

  report_filenames(config);

  EcoHydroReader                                    ehreader;
int                    NumYears;
  AsciiGridWriter              agwriter;
  Landscape< Float>            depth;
  Landscape< std::vector<Float> > Hydro;

config.set_var("NumYears", NumYears);

  for ( int year = 0; year < NumYears; ++year ) {
    ehreader(strm_eh, Hydro , BoxID, 365 );
    depth.resize(Hydro.nrows(), Hydro.ncols(), Hydro.yllcorner(), Hydro.xllcorner(), Hydro.cellsize(), Float(-9999));
    for ( int i = 0; i < Hydro.size(); ++i) {
      if ( BoxID[i] != BoxID.nodatavalue() ) {
            Float meanWD = 0;
            for ( int day = 120; day < 244; ++day ) {
                meanWD = meanWD + Hydro[i][day];
            }
            meanWD /= 124;
            depth[i] = meanWD - Elv[i];
      } else {
         depth[i] = BoxID.nodatavalue();
      }
    }

    strm_out << "# Year = " << year << std::endl;
    li_err += agwriter(strm_out, depth);
  }
```

```
            strm_eh.close();
            strm_out.close();

            return li_err;

        }
};

class preProcSummerMeanSalinity: public PreProc
{
    public:
        preProcSummerMeanSalinity()
        {
            ftpParamNames.clear();

            preParamNames["PreProc.EcoHydroBoxIDFile"]     = std::string("PreProc.EcoHydroBoxIDFile");
            preParamNames["PreProc.EcoHydroSalinityFile"] = std::string("PreProc.EcoHydroSalinityFile");

            postParamNames["SummerMeanSalinityFile"]     = std::string("SummerMeanSalinityFile");

        }

    protected:
        int preProc(const Config& config) const
          {
            int li_err = 0;
            std::string filename_out;
            std::string filename_bi;
            std::string filename_es;

            std::ofstream strm_out;
            std::ifstream strm_es;
            std::ifstream strm_bi;
```

```
config.set_var( postParamNames.find("SummerMeanSalinityFile")->second, filename_out);
config.set_var( preParamNames.find("PreProc.EcoHydroBoxIDFile")->second, filename_bi);
config.set_var( preParamNames.find("PreProc.EcoHydroSalinityFile")->second, filename_es);

strm_out.open(filename_out.c_str());
if ( !strm_out.is_open() ) {
  std::cout << "PreProcSummerMeanSalinity::preProc(): Error: Could not open summer mean salinity file " << filename_out << std::endl;
  ++li_err;
}

strm_es.open(filename_es.c_str() );
if ( !strm_es.is_open() ) {
  std::cout << "PreProcSummerMeanSalinity::preProc(): Error: Could not open EcoHydro salinity file " << filename_es << std::endl;
  ++li_err;
}

strm_bi.open(filename_bi.c_str() );
if ( !strm_bi.is_open() ) {
  std::cout << "PreProcSummerMeanSalinity::preProc(): Error: Could not open EcoHydro box ID file " << filename_bi << std::endl;
  ++li_err;
}

if ( li_err ) {
  strm_out.close();
  strm_es.close();
  strm_bi.close();
}

strm_bi.seekg(0, std::ios_base::beg);
Landscape<int> BoxID;
AsciiGridReader agreader;
if ( agreader(strm_bi, BoxID) ) {
  std::cout << "PreProcMeanSummerSalinity::preProc(): Error: Could not read boxID data." << std::endl;
  ++li_err;
```

```
  }
  strm_bi.close();

  if ( li_err ) {
    strm_es.close();
    strm_out.close();
    return li_err;
  }

  report_filenames(config);

  EcoHydroReader                                    ehreader;
int                    NumYears;
  AsciiGridWriter              agwriter;
  Landscape< Float>            msal;
  Landscape< std::vector<Float> > Sal;

config.set_var("NumYears", NumYears);

  for ( int year = 0; year < NumYears; ++year ) {
    ehreader(strm_es, Sal , BoxID, 12 );
    msal.resize(Sal.nrows(), Sal.ncols(), Sal.yllcorner(), Sal.xllcorner(), Sal.cellsize(), Float(-9999));
    for ( int i = 0; i < Sal.size(); ++i) {
      if ( BoxID[i] != BoxID.nodatavalue() ) {
            Float meanSA = 0;
            for ( int month = 4; month < 8; ++month ) {
                meanSA = meanSA + Sal[i][month];
            }
            msal[i] = meanSA/4;
      } else {
         msal[i] = BoxID.nodatavalue();
      }
    }
```

```
            strm_out << "# Year = " << year << std::endl;
            li_err += agwriter(strm_out, msal);
          }

          strm_es.close();
          strm_out.close();

          return li_err;

      }

};

class preProcSummerMeanTemp: public PreProc
{
   public:
      preProcSummerMeanTemp()
      {
          ftpParamNames.clear();

          preParamNames["PreProc.EcoHydroBoxIDFile"]     = std::string("PreProc.EcoHydroBoxIDFile");
          preParamNames["PreProc.EcoHydroTempFile"] = std::string("PreProc.EcoHydroTempFile");

          postParamNames["SummerMeanTempFile"]     = std::string("SummerMeanTempFile");
      }

   protected:
      int preProc(const Config& config) const
        {
          int li_err = 0;
          std::string filename_out;
          std::string filename_bi;
          std::string filename_et;
```

```
std::ofstream strm_out;
std::ifstream strm_bi;
std::ifstream strm_et;

config.set_var( postParamNames.find("SummerMeanTempFile")->second, filename_out);
config.set_var( preParamNames.find("PreProc.EcoHydroBoxIDFile")->second, filename_bi);
config.set_var( preParamNames.find("PreProc.EcoHydroTempFile")->second, filename_et);

strm_out.open(filename_out.c_str());
if ( !strm_out.is_open() ) {
  std::cout << "PreProcSummerMeanSalinity::preProc(): Error: Could not open summer mean temp file " << filename_out << std::endl;
  ++li_err;
}

strm_bi.open(filename_bi.c_str() );
if ( !strm_bi.is_open() ) {
  std::cout << "PreProcSummerMeanSalinity::preProc(): Error: Could not open EcoHydro box ID file " << filename_bi << std::endl;
  ++li_err;
}

strm_et.open(filename_et.c_str() );
if ( !strm_et.is_open() ) {
  std::cout << "PreProcSummerMeanSalinity::preProc(): Error: Could not open EcoHydro temp file " << filename_et << std::endl;
  ++li_err;
}

if ( li_err ) {
  strm_out.close();
  strm_bi.close();
  strm_et.close();
}

strm_bi.seekg(0, std::ios_base::beg);
Landscape<int> BoxID;
```

```
  AsciiGridReader agreader;
  if ( agreader(strm_bi, BoxID) ) {
    std::cout << "PreProcMeanSummerSalinity::preProc(): Error: Could not read boxID data." << std::endl;
    ++li_err;
  }
  strm_bi.close();

  if ( li_err ) {
    strm_out.close();
    strm_et.close();
    return li_err;
  }

  report_filenames(config);

  EcoHydroReader                                    ehreader;
int                    NumYears;
  AsciiGridWriter              agwriter;
  Landscape< Float>            met;
  Landscape< std::vector<Float> > Temp;

config.set_var("NumYears", NumYears);

  for ( int year = 0; year < NumYears; ++year ) {
    ehreader(strm_et, Temp , BoxID, 12 );
    met.resize(Temp.nrows(), Temp.ncols(), Temp.yllcorner(), Temp.xllcorner(), Temp.cellsize(), Float(-9999));
    for ( int i = 0; i < Temp.size(); ++i) {
      if ( BoxID[i] != BoxID.nodatavalue() ) {
          Float meanT = 0;
          for ( int month = 4; month < 8; ++month ) {
              meanT = meanT + Temp[i][month];
          }
          met[i] = meanT/4;
      } else {
```

```
                met[i] = BoxID.nodatavalue();
              }
            }

            strm_out << "# Year = " << year << std::endl;
            li_err += agwriter(strm_out, met);
          }

          strm_et.close();
          strm_out.close();

          return li_err;

      }

};

class Params : public IOS
{
// class Params
// This is a large class that used to read data in and store data provided
// by the user. It is the "authoritative" source of model parameters.
// The job of this class is to hold all of the model parameters,
// hold all of the support components (file names, file streams, etc ...),
// make sure all of these components are configured correctly and keep them up to date
// as the model runs.


        // ***************************************************************
        // Model support components
        //
        // ***************************************************************
    protected:
        Config                  mC_config;
```

```
public:
    enum RunType {ConfigOnly=1, ConfigAndReport, Test, Run, ReadConfigOnly} ThisRunType;

    // Input streams for environmental variable that will
    // change over time.
    mutable std::ifstream istrm_waveAmp;
    mutable std::ifstream istrm_meanSal;
    // mutable std::ifstream istrm_landWater;
    mutable std::ifstream istrm_smDepth;
    mutable std::ifstream istrm_smSal;
    mutable std::ifstream istrm_smTemp;

    // Output streams
    mutable std::ofstream  ostrm_dyn;
    std::string outputTemplate_dyn;

    // ***************************************************************
    // Model parameters. These are the objects actually involved
    // in running the model.
    //
    // ***************************************************************
    Float                   NumYears;
    Float                   StartYear;
    Float                   JumpToYear;
    SpeciesLUT                  SpLUT;
    Landscape<Patch>            C0; // Initial conditions.
    Landscape<Float>            waveAmplitude;
    Landscape<Float>            meanSalinity;
    // Landscape<Float>            landWater;
    Landscape<Float>            smDepth;
    Landscape<Float>            smSal;
    Landscape<Float>            smTemp;
```

```
    std::vector<int>            outputYears;
    int                         inputYear;

public:
    Params() :
      mC_config(),
      ThisRunType(Run),
      istrm_waveAmp(),
      istrm_meanSal(),
      ostrm_dyn(),
      NumYears(0),
      SpLUT(),
      C0(),
      waveAmplitude(),
      meanSalinity ()
    {}

    ~Params()
    {
      istrm_waveAmp.close();
      istrm_meanSal.close();
      ostrm_dyn.close();
    }

private:
    int config_openfile(const char * const ac_paramName, std::ifstream& aC_stream) const
    {
      if ( !mC_config.isDefined(ac_paramName) ) {
        std::cout << "class Params::config_open(...) : Parameter " << ac_paramName << " is not defined." << std::endl;
        return 1;
      }

      std::string lS_filename;
      mC_config.set_var(ac_paramName, lS_filename);
```

```
lS_filename = boost::regex_replace( lS_filename, boost::regex(std::string("^ *")), std::string("") );
lS_filename = boost::regex_replace( lS_filename, boost::regex(std::string(" *$")), std::string("") );
std::cout << "class Params: Opening file for reading = >>" << lS_filename << "<<" << std::endl;
  aC_stream.open(lS_filename.c_str());
  if ( !aC_stream.is_open() ) {
    std::cout << "class Params::config_open(...) : Error : Could not open input file " << lS_filename << " (" << mC_config.find(ac_paramName) << ")"
  << std::endl;
    return 1;
  }

  std::cout << "class Params::config_open(...) : Opened file " << lS_filename << " (" << mC_config.find(ac_paramName) << ") for reading" <<
  std::endl;

  return 0;
}

int config_openfile(const std::string& a_filename, std::ifstream& a_stream) const
{ return config_openfile(a_filename.c_str(), a_stream); }

int config_openfile(const char * const ac_paramName, std::ofstream& aC_stream ) const
{
  if ( !mC_config.isDefined(ac_paramName) ) {
    std::cout << "class Params::config_open(...) : Parameter " << ac_paramName << " is not defined." << std::endl;
    return 1;
  }

  std::string lS_filename;
  mC_config.set_var(ac_paramName, lS_filename);
lS_filename = boost::regex_replace( lS_filename, boost::regex(std::string("^ *")), std::string("") );
lS_filename = boost::regex_replace( lS_filename, boost::regex(std::string(" *$")), std::string("") );
std::cout << "class Params: Opening file for reading = >>" << lS_filename << "<<" << std::endl;
  aC_stream.open(lS_filename.c_str());
  if ( !aC_stream.is_open() ) {
    std::cout << "class Params : Error : Could not open output file " << lS_filename << " (" << mC_config.find(ac_paramName) << ")" << std::endl;
```

```
      return 1;
    }

    std::cout << "class Params : Opened file " << lS_filename << " (" << mC_config.find(ac_paramName) << ") for writting" << std::endl;
    return 0;
  }

  int config_openfile(const std::string& a_filename, std::ofstream& a_stream) const
  { return config_openfile(a_filename.c_str(), a_stream); }

int preproc()
{
  int err = 0;

  std::cout << "Preprocessing" << std::endl;

  preProcWaveAmp          preProcWA;
  preProcMeanSalinity     preProcMS;
  preProcLandWater        preProcLW;
  preProcElevation        preProcEL;
  preProcSummerMeanDepth    preProcSMD;
  preProcSummerMeanSalinity preProcSMS;
  preProcSummerMeanTemp     preProcSMT;

  std::cout << "Preprocessing wave amp. files" << std::endl;
  err += preProcWA.go(mC_config);
  std::cout << "Preprocessing mean salinity files" << std::endl;
  err += preProcMS.go(mC_config);
  // std::cout << "Preprocessing land/water files" << std::endl;
  // err += preProcLW.go(mC_config);
  // std::cout << "Preprocessing elevation files" << std::endl;
  // err += preProcEL.go(mC_config);
  std::cout << "Preprocessing summer mean water depth" << std::endl;
  err += preProcSMD.go(mC_config);
```

```
    std::cout << "Preprocessing summer mean salinity" << std::endl;
    err += preProcSMS.go(mC_config);
    std::cout << "Preprocessing summer mean temp" << std::endl;
    err += preProcSMT.go(mC_config);
    std::cout << "Done preprocessing" << std::endl;

    return err;

  }

public:
    int config(const char * const filename)
    {
      // ***************************************************************
      // Step 1: Load the data from the configuration file.
      //
      // ***************************************************************
      if ( mC_config.read_config(filename) ) {
        std::cout << "class Params : Error : Config::read_config(...) failed" << std::endl;
        return 1;
      }

      int li_err = 0;

      // ***************************************************************
      // Step 2: Configure the support components of the model.
      //
      // ***************************************************************

      // Step 2.1: Set the run type.
      ThisRunType = Run;
      std::string lS_RunType;
      if ( mC_config.isDefined("RunType") ) mC_config.set_var("RunType", lS_RunType);
          if ( lS_RunType == std::string("ConfigOnly")      ) { ThisRunType = ConfigOnly;}
```

```
  else if ( lS_RunType == std::string("ConfigAndReport") ) { ThisRunType = ConfigAndReport;}
  else if ( lS_RunType == std::string("Test")          ) { ThisRunType = Test;}
else if ( lS_RunType == std::string("ReadConfigOnly")  ) { ThisRunType = ReadConfigOnly; return 0; }

  // Step 2.2: Configure NumYears, because the preprocessor may need this value.
  if ( mC_config.set_var("NumYears", NumYears) ) {
    std::cout << "Params::config(): Error: NumYears not defined." << std::endl;
    ++li_err;
  }
  if ( mC_config.isDefined("StartYear") && mC_config.isDefined("JumpToYear") ) {
    mC_config.set_var("StartYear", StartYear);
    mC_config.set_var("JumpToYear", JumpToYear);
    if ( JumpToYear < StartYear ) {
        std::cout << "Params::config(): Error: JumpToYear < StartYear :" << JumpToYear << " < " << StartYear << std::endl;
        ++li_err;
    }
  } else if ( mC_config.isDefined("StartYear"  ) ) {
    mC_config.set_var("StartYear", StartYear);
    JumpToYear = StartYear;
  } else if ( mC_config.isDefined("JumpToYear" ) ) {
    std::cout << "Params::config(): Error: JumpToYear is defined but StartYear is not." << std::endl;
    ++li_err;
  } else {
    StartYear = 0;
    JumpToYear = 0;
  }

  // Step 2.3: Open the wave amplitude file, the mean salinity and the land/water file.
  //         This is a complex set of steps that involve possibly
  //         running a preprocessor.
  li_err += preproc();


  // Step 2.4: Open the input/output files.
```

```
li_err += this->config_openfile("OutputFile", ostrm_dyn);

if ( mC_config.isDefined("OutputTemplate") ) { mC_config.set_var("OutputTemplate", outputTemplate_dyn); }

if ( mC_config.isDefined("OutputYears") ) {
  std::vector<int> years;
  mC_config.set_var("OutputYears", years);

  int maxyear = *std::max_element( years.begin(), years.end() );
  if ( NumYears < maxyear ) { std::cout << "Params: Warning: max output year greater than number of years" << std::endl; }

  outputYears.resize(NumYears+1,0);
  for ( int i = 0; i < years.size(); ++i ) {
      outputYears[ std::min(years[i],int(NumYears+1) ) ] = 1;
  }
} else {
  outputYears.resize(NumYears+1,1);
}

mC_config.set_var("Year", inputYear);



// ##################################################################
// Step 3: Conigure the modeling parameters.
//
// ##################################################################

// Step 3.1: Read the veg table information in to the SpeciesLUT object
std::cout << "Configuring species look up table" << std::endl;
li_err += SpLUT.config(mC_config);
if ( mC_config.isDefined("Verbose") ) { std::cout << SpLUT << std::endl; }
```

```
  // Step 3.2: Read the initial conditions.
  std::ifstream istrm;
  if ( config_openfile( "InitialConditionFile", istrm) ) {
    ++li_err;
  } else {
    InitCondReader icreader;
    li_err += icreader(istrm, C0, SpLUT);
  }

  if ( li_err ) { return li_err;}

  // Step 3.3: Give the waveAmplitude, meanSalinity and landWater objects their correct size
if ( config_openfile( "WaveAmplitudeFile", istrm_waveAmp) ){
   ++li_err;
} else {
     AsciiGridReader agreader;
     if ( agreader(istrm_waveAmp, waveAmplitude) ) {
       std::cout << "Params::config(...): Error: Could not read data from the wave amplitude file. This is odd." << std::endl;
       ++li_err;
     }
     istrm_waveAmp.seekg(0, std::ios_base::beg);
}

if ( config_openfile( "MeanSalinityFile", istrm_meanSal) ) {
   ++li_err;
} else {
     AsciiGridReader agreader;
     if ( agreader(istrm_meanSal, meanSalinity) ) {
       std::cout << "Params::config(...): Error: Could not read data from mean salinity file. This is odd." << std::endl;
       ++li_err;
     }
     istrm_meanSal.seekg(0, std::ios_base::beg);
}
```

```
/*
if( config_openfile("LandWaterFile", istrm_landWater) ) {
   ++li_err;
} else {
     AsciiGridReader agreader;
     if ( agreader(istrm_landWater, landWater) ) {
       std::cout << "Params::config(...): Error: Could not read data from mean salinity file. This is odd." << std::endl;
       ++li_err;
     }
     istrm_landWater.seekg(0, std::ios_base::beg);
}
*/

  if( config_openfile("SummerMeanWaterDepthFile", istrm_smDepth) ) {
   ++li_err;
} else {
     AsciiGridReader agreader;
     if ( agreader(istrm_smDepth, smDepth) ) {
       std::cout << "Params::config(...): Error: Could not read data from summer mean depth file. This is odd." << std::endl;
       ++li_err;
     }
     istrm_smDepth.seekg(0, std::ios_base::beg);
}

  if( config_openfile("SummerMeanSalinityFile", istrm_smSal) ) {
   ++li_err;
} else {
     AsciiGridReader agreader;
     if ( agreader(istrm_smSal, smSal) ) {
       std::cout << "Params::config(...): Error: Could not read data from summer mean salinity file. This is odd." << std::endl;
       ++li_err;
     }
     istrm_smSal.seekg(0, std::ios_base::beg);
}
```

```
  if( config_openfile("SummerMeanTempFile", istrm_smTemp) ) {
   ++li_err;
} else {
     AsciiGridReader agreader;
     if ( agreader(istrm_smTemp, smTemp) ) {
       std::cout << "Params::config(...): Error: Could not read data from summer mean temp file. This is odd." << std::endl;
       ++li_err;
     }
     istrm_smTemp.seekg(0, std::ios_base::beg);
}

  // Position the file pointers to the proper place in each file.
  Float diff = JumpToYear - StartYear;
  AsciiGridReader agreader;
  for ( int i = 0; i < diff; ++i) {
    agreader(istrm_waveAmp, waveAmplitude);
    agreader(istrm_meanSal, meanSalinity);
    agreader(istrm_smDepth, smDepth);
    agreader(istrm_smSal,   smSal);
    agreader(istrm_smTemp,  smTemp);
  }

  return li_err;
}

int update_all()
{
  int li_err = 0;
  AsciiGridReader agreader;

  li_err += agreader(istrm_waveAmp, waveAmplitude);
  li_err += agreader(istrm_meanSal, meanSalinity);
  li_err += agreader(istrm_smDepth, smDepth);
```

```
        li_err += agreader(istrm_smSal,   smSal);
        li_err += agreader(istrm_smTemp,  smTemp);

        return li_err;
      }

      std::ostream& write(std::ostream& ostr) const
      {
        ostr << "Run Time                    : NumYears  = " << NumYears     << " (" << mC_config.find("NumYears") << ") years \n";

        return ostr;
      }

      int config_report() const
      {
        std::cout << "Config::config_report(): " << std::endl;
        return 0;
      }


};


std::ostream& operator<<(std::ostream& aC_O, const Landscape<Patch>& aC_obj)
{
    int count = 0;
    for ( Landscape<Patch>::const_iterator iter = aC_obj.begin(); iter != aC_obj.end(); ++iter, ++count ) {
        for ( Patch::const_iterator miter = iter->begin(); miter != iter->end(); ++miter ) {
          aC_O << count << ", " << miter->first << ", " << miter->second << std::endl;
        }
    }
    return aC_O;
}
```

```
int Patch::step()
{
    // Step advances one model cell by one time step.
    // Currently, a model cell is a 500x500 patch, and the time step is 1 year.

    Float  waveAmplitude = mC_params.waveAmplitude[mi_myloc]; // Hydrologic amplitude (stdev) for the year in this patch.wave amplitude
    Float  meanSalinity  = mC_params.meanSalinity[mi_myloc];
    Float  d             = mC_params.smDepth[mi_myloc];
    Float  s             = mC_params.smSal[mi_myloc];
    Float  t             = mC_params.smTemp[mi_myloc];

    // C_lost, C_occupied and the individual species cover values are fractions of total cell that is covered by each species.

    Float C_lost = 0;
    Float C_occupied = 0;
    for ( BaseClass::iterator iter = BaseClass::begin(); iter != BaseClass::end(); ++iter ) {
        int spi          = iter->first;
        Float cover      = iter->second;

        SpeciesInfo spinfo = mC_params.SpLUT[spi];

        Float p_death     = spinfo.D(waveAmplitude, meanSalinity); // The call to D should pass wave amplitude first and mean salinity second.

        C_occupied       += cover;
        C_lost          += p_death*cover;
        cover           -= p_death*cover;
        iter->second      = cover;
    }
    Float C_unoccupied = std::max(0.0, std::min( (1.0 - C_occupied) + C_lost, 1.0));

    int spi = mC_params.SpLUT[0].P(waveAmplitude, meanSalinity);

    BaseClass::operator[](spi) += C_unoccupied;
```

```
    // Work out the changes in SAV and WATER
    Float totalWater              = (BaseClass::operator[](SpeciesInfo::spi_water) + BaseClass::operator[](SpeciesInfo::spi_sav));
    BaseClass::operator[](SpeciesInfo::spi_sav)   = mC_params.SpLUT.sav(t,s,d) * totalWater;
    BaseClass::operator[](SpeciesInfo::spi_water) = totalWater - BaseClass::operator[](SpeciesInfo::spi_sav);

    return 0;
}

class Model : public IOS
{

// class Model
// This is the core of the model, were all the elements are brought together.

    public:
        static Params  params;


    public:
        Model() {}

    protected:
        int config(int ai_argc, char* acp_argv[] )
        {
          // This function configures the model from the data held in the Params class.
          if ( ai_argc != 2 ) {
            std::cout << "class Model : Error : No config file specified." << std::endl;
            std::cout << "class Model : Error : Call format is : lavegmod <configfile>" << std::endl;
            return 1;
          }

          if ( params.config(acp_argv[1]) ) {
            std::cout << "class Model : Error : Params::config(...) failed." << std::endl;
            return 1;
```

```
      }

      std::cout << params << std::endl;

      return 0;
    }

public:
    int run(int ai_argc, char* acp_argv[])
    {
      // Step 1: Read in the data from the configuration file.
      std::cout << "Model::run(...) : Reading configuration file" << std::endl;
      if ( config(ai_argc, acp_argv) ) {
        std::cout << "class Model : Model::config(...) failed." << std::endl;
        return 1;
      }

      if ( params.ThisRunType == Params::ConfigOnly )      {                    return 0; }
      if ( params.ThisRunType == Params::ConfigAndReport ) { params.config_report(); return 0;}
      if ( params.ThisRunType == Params::Test )           { test();              return 0;}
    if ( params.ThisRunType == Params::ReadConfigOnly )  { return 0; }

      // Step 1 : Create the model and set it up for running.
      Landscape<Patch> lC_model(params.C0);


      // Step 3 : Run the model
      for ( int time = 1; time <= params.NumYears; ++time) {
        std::cout << "##########################" << std::endl;
        std::cout << "# run time = " << time << std::endl;
        std::cout << "##########################" << std::endl;

        // Step 3.1: Tell params to update everything for the next time step.
        if ( params.update_all() ) {
```

```
        return 1;
    }

    // Step 3.2: Update the model
    std::cout << "Updating vegetation distribution" << std::endl;
    for ( Landscape<Patch>::iterator lC_iter = lC_model.begin(); lC_iter != lC_model.end(); ++lC_iter ) {
      if ( lC_iter->CellID() != -9999 ) {
            try { lC_iter->step(); }
             catch( std::exception& exc ) {
                        std::cout << exc.what() << std::endl;
                        std::cout << "myloc = " << lC_iter->loc() << std::endl;
                        std::cout << "CellID = " << lC_iter->CellID() << std::endl;
             }
      }
    }

    // Step 3.3: Write data to the output file.
    if ( params.outputTemplate_dyn.size() != 0 && params.outputYears[time] ) {
        AsciiGridPlusWriter lC_agpw;
        std::string        filename;
        std::stringstream   year;
        year << params.inputYear + time;
        filename = boost::regex_replace( params.outputTemplate_dyn, boost::regex(std::string("<YEAR>")), year.str() );
        std::ofstream strm;
        strm.open(filename.c_str());
        lC_agpw(strm, lC_model, params.SpLUT);
        strm.close();
    }
    {
    AsciiGridPlusWriter agpw;
    params.ostrm_dyn << "# Year = " << time << std::endl;
    agpw(params.ostrm_dyn, lC_model, params.SpLUT);
    }
  }
```

```
        // Step 4 : Write output

        return 0;
      }

      int test()
      {
      }

      int write_output()
      {
        return 0;
      }

      std::ostream& write(std::ostream& ostr) const
      {
        return ostr;
      }
};


Params Model::params;
const Params& Patch::mC_params = Model::params;


int main(int argc, char* argv[])
{
    Times lC_timer;
    lC_timer.start();
    Model model;

    if ( model.run(argc, argv) ) {
        std::cout << "main : Error : Model::run(...) failed." << std::endl;
```

```
        return 1;
    }
    lC_timer.end();
    std::cout << lC_timer << std::endl;

    return 0;
}
```

III. Source code for newconfig.h

```
#ifndef newconfig_h
#define newconfig_h

// C++/STL includes
#include <fstream>
#include <iostream>
#include <string>
#include <sstream>
#include <map>
#include <algorithm>

// Third party includes
#include <boost/regex.hpp>
#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/io.hpp>

class Config : public std::map<std::string, std::string>
{
    public:
        typedef std::map<std::string, std::string> BaseClass;
        typedef std::string                VarNameClass;
        typedef std::string                ValueClass;
        typedef std::string                FirstClass;
```

```
    typedef std::string                 SecondClass;
    typedef BaseClass::iterator             iterator;
    typedef BaseClass::const_iterator       const_interator;

public:
    Config();
    Config(const Config& );
    ~Config();

    Config& operator=(const Config& );

public:
    int read_config(std::string aS_filename);
    int read_config(const char * const acp_filename);

    bool isDefined(const std::string& aS_varname)  const;
    bool isDefined(const char * const acp_varname) const;

    std::string  find(const std::string& aS_varname)   const;
    std::string& find(const std::string& aS_varname);
    std::string  find(const char * const acp_varname ) const;
    std::string& find(const char * const acp_varname );

    Config      subset(const std::string& aS_varname) const;
    Config      subset(const char * const acp_varname) const;
    Config      subset(boost::regex& aC_varname) const;

protected:
    template <class fundtype>
    int set_var_fundtype(const std::string& aS_varname, fundtype& aC_value) const;

public:
    int set_var(const std::string& aS_varname, char&          a_value ) const;
    int set_var(const std::string& aS_varname, unsigned char&  a_value ) const;
```

```
    int set_var(const std::string& aS_varname, short&         a_value ) const;
    int set_var(const std::string& aS_varname, unsigned short& a_value ) const;
    int set_var(const std::string& aS_varname, int&           a_value ) const;
    int set_var(const std::string& aS_varname, unsigned int&   a_value ) const;
    int set_var(const std::string& aS_varname, long&          a_value ) const;
    int set_var(const std::string& aS_varname, long long&      a_value ) const;
    int set_var(const std::string& aS_varname, float&         a_value ) const;
    int set_var(const std::string& aS_varname, double&        a_value ) const;
    int set_var(const std::string& aS_varname, long double&    a_value ) const;
    int set_var(const std::string& aS_varname, std::string& aS_string ) const;

    int set_var(const char * const acp_varname, char&           a_value )  const;
    int set_var(const char * const acp_varname, unsigned char&   a_value )  const;
    int set_var(const char * const acp_varname, short&          a_value )  const;
    int set_var(const char * const acp_varname, unsigned short&  a_value )  const;
    int set_var(const char * const acp_varname, int&            a_value )  const;
    int set_var(const char * const acp_varname, unsigned int&    a_value )  const;
    int set_var(const char * const acp_varname, long&           a_value )  const;
    int set_var(const char * const acp_varname, long long&       a_value )  const;
    int set_var(const char * const acp_varname, float&          a_value )  const;
    int set_var(const char * const acp_varname, double&         a_value )  const;
    int set_var(const char * const acp_varname, long double&     a_value )  const;

protected:
    template<class fundtype>
    int set_var_vector_fundtype(const std::string& aS_varname, std::vector<fundtype>&         a_value ) const;
    template<class fundtype>
    int parse_vector_fundtype(const std::string& aS_line,      std::vector<fundtype>&         a_value ) const;

public:
    int set_var(const std::string& aS_varname, std::vector<char>&          a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<unsigned char>&  a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<short>&         a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<unsigned short>& a_value ) const;
```

```
    int set_var(const std::string& aS_varname, std::vector<int>&          a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<unsigned int>&   a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<long>&          a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<long long>&      a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<float>&         a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<double>&        a_value ) const;
    int set_var(const std::string& aS_varname, std::vector<long double>&    a_value ) const;

    int set_var(const char * const acp_varname, std::vector<char>&          a_value )  const;
    int set_var(const char * const acp_varname, std::vector<unsigned char>&   a_value )  const;
    int set_var(const char * const acp_varname, std::vector<short>&          a_value )  const;
    int set_var(const char * const acp_varname, std::vector<unsigned short>&  a_value )  const;
    int set_var(const char * const acp_varname, std::vector<int>&           a_value )  const;
    int set_var(const char * const acp_varname, std::vector<unsigned int>&    a_value )  const;
    int set_var(const char * const acp_varname, std::vector<long>&          a_value )  const;
    int set_var(const char * const acp_varname, std::vector<long long>&      a_value )  const;
    int set_var(const char * const acp_varname, std::vector<float>&         a_value )  const;
    int set_var(const char * const acp_varname, std::vector<double>&        a_value )  const;
    int set_var(const char * const acp_varname, std::vector<long double>&    a_value )  const;

protected:
    template<class fundtype>
    int set_var_matrix_fundtype(const std::string& aS_varname, boost::numeric::ublas::matrix<fundtype>&       a_value ) const;

public:
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<char>&          a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<unsigned char>&  a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<short>&         a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<unsigned short>& a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<int>&           a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<unsigned int>&   a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<long>&          a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<long long>&      a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<float>&         a_value ) const;
```

```
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<double>&        a_value ) const;
    int set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<long double>&    a_value ) const;

    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<char>&          a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<unsigned char>&  a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<short>&         a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<unsigned short>& a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<int>&           a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<unsigned int>&   a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<long>&          a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<long long>&      a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<float>&         a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<double>&        a_value )  const;
    int set_var(const char * const acp_varname, boost::numeric::ublas::matrix<long double>&    a_value )  const;


public:

static const std::string myINT;
static const std::string myNUMBER;
static const std::string myRANGE;
static const std::string myRANGEDef;
static const std::string myRANGESET;
static const std::string myRANGESETS;
static const std::string myVARIABLE;
static const std::string myVARNAME;
static const std::string mySimpVec;

static const std::string myROWCOLSET; // matrix data format
static const std::string myROWCOLSETS;
static const std::string mySMFRow;
static const std::string mySMF; // Simple matrix format

static const std::string re_variable;
```

```
    static const std::string re_rangeset;
    static const std::string re_line;

    static const std::string myFILENAME;
    static const std::string myPATHSTRING;


};

std::ostream& operator<<(std::ostream& aC_O, const Config& aC_obj);

std::ostream& operator<<(std::ostream& aC_O, const std::vector<float>& aC_obj);


#endif
```

IV. Source code for newconfig.cc

```
#include "newconfig.h"

Config::Config() : BaseClass() {}
Config::Config(const Config& aC_orig ) : BaseClass(aC_orig) {}
Config::~Config() {BaseClass::clear();}

Config& Config::operator=(const Config& aC_orig ) {
    if ( this != &aC_orig ) {
        BaseClass::operator=(aC_orig);
    }
    return *this;
}

int Config::read_config(const char * const acp_filename)
{
    std::string lS_filename(acp_filename);
```

```
    return read_config(lS_filename);
}
int Config::read_config(std::string aS_filename)
{
    std::string lS_space(" *");
    std::string lS_any("(.*)");
    // std::string lS_paramName("[[:word:]][[:word:],[:digit:]]*");
    std::string lS_paramName("([[:word:]][[:word:][:digit:]\\.]*)");
    std::string lS_filename(lS_any);

    // boost::regex  lC_assign( lS_space + lS_paramName + lS_space + std::string("=") + lS_space + lS_any);
    boost::regex  lC_assign( lS_space + lS_paramName + lS_space + std::string("=") + lS_space + lS_any);
    boost::regex  lC_assignAlt( lS_space + lS_paramName + lS_space + std::string(":") + lS_any + std::string(":") + lS_space + lS_any );
    boost::regex  lC_comment( std::string("//.*"));
    boost::regex  lC_include( std::string("#include") + lS_space + std::string("<") + lS_space + lS_filename + lS_space + std::string(">") + lS_space);
    boost::regex  lC_blank( lS_space );
    boost::regex  lC_paramCall( std::string("\\$\\{ *") + lS_paramName + std::string(" *\\}"));
    boost::regex  lC_cc( std::string("\\\\$") );
    boost::regex  lC_tab( std::string("\t") );

    boost::smatch lC_results;

    std::ifstream lC_infile(aS_filename.c_str());
    std::string   lS_line;
    int          li_lineno = 0;

    if ( !lC_infile.is_open() ) {
        std::cout << "class Config : Error : Could not open file " << aS_filename << std::endl;
        return 1;
    }

    for ( getline( lC_infile, lS_line); !lC_infile.eof(); getline(lC_infile, lS_line) ) {
        li_lineno++;
```

```
// Find any comments and remove them from the line
lS_line = boost::regex_replace( lS_line, lC_comment, std::string("") );

// Find the continuation character (CC) if it is present, add concatination lines until we don't read a CC any longer.
while ( regex_search(lS_line, lC_results, lC_cc) ) {
  std::string lS_contLine;
  getline(lC_infile, lS_contLine);
  lS_line = boost::regex_replace(lS_line, lC_cc, lS_contLine);
}

// Find tabs and replace them with spaces
lS_line = boost::regex_replace( lS_line, lC_tab, std::string(" ") );

// If after all this pre-processes the line is blank, skip it.
if ( regex_match( lS_line, lC_blank ) ) {
  continue;
}

if ( regex_search( lS_line, lC_results, lC_assign ) ) {
  std::string lS_varname( lC_results[1].first, lC_results[1].second );
  std::string lS_value  ( lC_results[2].first, lC_results[2].second );

  while ( regex_search ( lS_value, lC_results, lC_paramCall ) ) {
    std::string lS_keyname ( lC_results[1].first, lC_results[1].second );
    boost::regex lC_keyCall( std::string("\\$\\{ *") + lS_keyname + std::string(" *\\}"));
    lS_value = boost::regex_replace( lS_value, lC_keyCall, find(lS_keyname));
  }

  operator[]( lS_varname ) = lS_value;
  continue;
}

if ( regex_search( lS_line, lC_results, lC_assignAlt ) ) {
  std::string lS_varname( lC_results[1].first, lC_results[1].second );
```

```
        std::string lS_value  ( lC_results[3].first, lC_results[3].second );

        while ( regex_search ( lS_value, lC_results, lC_paramCall ) ) {
          std::string lS_keyname ( lC_results[1].first, lC_results[1].second );
          boost::regex lC_keyCall( std::string("\\$\\{ *") + lS_keyname + std::string(" *\\}"));
          lS_value = boost::regex_replace( lS_value, lC_keyCall, find(lS_keyname));
        }

        operator[]( lS_varname ) = lS_value;
        continue;
      }

      if ( regex_search( lS_line, lC_results, lC_include ) ) {
        std::string lS_filename( lC_results[1].first, lC_results[1].second );
        continue;
      }

      std::cout << "Unrecognized text on line " << li_lineno << " of " << aS_filename << " : " << lS_line << std::endl;
    }

    return 0;
}

      bool Config::isDefined(const std::string& aS_varname)  const { return BaseClass::find(aS_varname) != BaseClass::end(); }
      bool Config::isDefined(const char * const acp_varname) const { return BaseClass::find(std::string(acp_varname) ) != BaseClass::end(); }

      std::string  Config::find(const std::string& aS_varname)   const { return BaseClass::find(aS_varname)->second; }
      std::string& Config::find(const std::string& aS_varname)        { return BaseClass::find(aS_varname)->second; }
      std::string  Config::find(const char * const acp_varname ) const { return BaseClass::find( std::string(acp_varname))->second; }
      std::string& Config::find(const char * const acp_varname )       { return BaseClass::find( std::string(acp_varname))->second; }

      Config Config::subset(boost::regex& aC_varname) const
      {
        Config lC_ret;
```

```
  boost::smatch lC_results;

  for (BaseClass::const_iterator iter = BaseClass::begin(); iter != BaseClass::end(); ++iter ) {
    if ( regex_search(iter->first, lC_results, aC_varname ) ) {
        lC_ret[(iter->first)] = iter->second;
    }
  }
  return lC_ret;
}



template <class fundtype>
int Config::set_var_fundtype(const std::string& aS_varname, fundtype& aC_value) const
{
  if ( !isDefined( aS_varname ) ) return 1;

  boost::regex lC_re(myNUMBER);
  if ( !boost::regex_match( find(aS_varname), lC_re) ) return 1;

  std::stringstream lC_strm( find(aS_varname) );
  lC_strm >> aC_value;
  return 0;
}


int Config::set_var(const std::string& aS_varname, char&          a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, unsigned char&  a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, short&         a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, unsigned short& a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, int&           a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, unsigned int&   a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, long&          a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, long long&      a_value ) const { return set_var_fundtype(aS_varname, a_value); }
```

```
int Config::set_var(const std::string& aS_varname, float&         a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, double&        a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, long double&    a_value ) const { return set_var_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, std::string&  aS_string ) const
{
  if ( !isDefined( aS_varname ) ) return 1;
  aS_string = find(aS_varname);
  return 0;
}

int Config::set_var(const char * const acp_varname, char&           a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, unsigned char&   a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, short&          a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, unsigned short&  a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, int&            a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, unsigned int&    a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, long&           a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, long long&       a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, float&          a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, double&         a_value )  const { return set_var( std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, long double&     a_value )  const { return set_var( std::string(acp_varname), a_value ); }

template<class fundtype>
int Config::set_var_vector_fundtype(const std::string& aS_varname, std::vector<fundtype>&         a_value ) const
{
  if ( !isDefined( aS_varname) ) return 1;
  std::string lS_line = find(aS_varname);
  return parse_vector_fundtype(lS_line , a_value );
}

template<class fundtype>
int Config::parse_vector_fundtype(const std::string& aS_line, std::vector<fundtype>& a_value ) const
{
  boost::regex lC_cv(myRANGESETS, boost::regex::extended);
```

```
boost::regex lC_sv(mySimpVec,   boost::regex::extended);

if ( boost::regex_match(aS_line, lC_cv) ) {

  boost::regex rangeset(re_rangeset, boost::regex::extended);
  boost::sregex_token_iterator i(aS_line.begin(), aS_line.end(), rangeset, 1);
  boost::sregex_token_iterator j;
  int count=0;
  while (i != j) {
      std::string lC_range = *i;
      boost::smatch lC_mr;
      boost::regex_match( lC_range, lC_mr, rangeset);

      int li_start    = 0;
      int li_stop     = 0;
      double ld_value = 0;
      std::istringstream lC_strm;

      if ( lC_mr[3].length() != 0 ) {
          lC_strm.clear();
          lC_strm.str( std::string(lC_mr[3]) );
          lC_strm >> li_start;
          lC_strm.clear();
          lC_strm.str( std::string(lC_mr[4]) );
          lC_strm >> li_stop;
      } else {
          lC_strm.clear();
          lC_strm.str( std::string(lC_mr[5]) );
          lC_strm >> li_start;
          li_stop = li_start;
      }

      lC_strm.clear();
      lC_strm.str( std::string( lC_mr[6] ) );
```

```
        lC_strm >> ld_value;

        if ( a_value.size() < li_stop+1 ) a_value.resize(li_stop+1);
        for ( int li_count = li_start; li_count <= li_stop; ++li_count ) a_value[li_count] = ld_value;

        ++count;
        ++i;
    }
    if (count <= 0) return(1);

    return 0;
  }

  if ( boost::regex_match(aS_line, lC_sv, boost::regex_constants::match_any) ) {
    boost::regex lC_elt(myNUMBER, boost::regex::extended);
    boost::sregex_token_iterator i(aS_line.begin(), aS_line.end(), lC_elt, 1);
    boost::sregex_token_iterator j;
    int count = 0;
    a_value.clear();
    std::stringstream lC_strm;
    while ( i != j ) {

        fundtype l_value;
        lC_strm.clear();
        lC_strm.str( std::string( *i ) );
        lC_strm >> l_value;



        a_value.push_back(l_value);

        ++count;
        ++i;
    }
```

```
    if ( count <= 0 ) return 1;

    return 0;
  }

  return 1;

}

int Config::set_var(const std::string& aS_varname, std::vector<char>&          a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<unsigned char>&  a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<short>&         a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<unsigned short>& a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<int>&           a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<unsigned int>&   a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<long>&          a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<long long>&      a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<float>&         a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<double>&        a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
int Config::set_var(const std::string& aS_varname, std::vector<long double>&    a_value ) const { return set_var_vector_fundtype(aS_varname,
  a_value);}
```

```
int Config::set_var(const char * const acp_varname, std::vector<char>&          a_value )  const { return set_var( std::string(acp_varname), a_value
  ); }
int Config::set_var(const char * const acp_varname, std::vector<unsigned char>&   a_value )  const { return set_var( std::string(acp_varname),
  a_value ); }
int Config::set_var(const char * const acp_varname, std::vector<short>&         a_value )  const { return set_var( std::string(acp_varname), a_value
  ); }
int Config::set_var(const char * const acp_varname, std::vector<unsigned short>&  a_value )  const { return set_var( std::string(acp_varname),
  a_value ); }
int Config::set_var(const char * const acp_varname, std::vector<int>&           a_value )  const { return set_var( std::string(acp_varname), a_value );
  }
int Config::set_var(const char * const acp_varname, std::vector<unsigned int>&    a_value )  const { return set_var( std::string(acp_varname),
  a_value ); }
int Config::set_var(const char * const acp_varname, std::vector<long>&          a_value )  const { return set_var( std::string(acp_varname), a_value
  ); }
int Config::set_var(const char * const acp_varname, std::vector<long long>&       a_value )  const { return set_var( std::string(acp_varname),
  a_value ); }
int Config::set_var(const char * const acp_varname, std::vector<float>&         a_value )  const { return set_var( std::string(acp_varname), a_value
  ); }
int Config::set_var(const char * const acp_varname, std::vector<double>&         a_value )  const { return set_var( std::string(acp_varname),
  a_value ); }
int Config::set_var(const char * const acp_varname, std::vector<long double>&     a_value )  const { return set_var( std::string(acp_varname),
  a_value ); }


template<class fundtype>
int Config::set_var_matrix_fundtype(const std::string& aS_varname, boost::numeric::ublas::matrix<fundtype>&         a_value ) const
{
  a_value.clear();

  boost::regex lC_mdf(myROWCOLSETS, boost::regex::extended); // matrix data format
  boost::regex lC_smf(mySMF,       boost::regex::extended); // simple matrix format

  if ( !isDefined( aS_varname) ) return 1;
```

```
std::string lS_line = find(aS_varname);
if ( boost::regex_match(lS_line, lC_mdf) ) {

  // This is the matrix data format. This is a special shorthand format for specifying a matrix.
  // The format allows you to specify the value for sub-matricies within the overall matrix.
  // The basic format for the matrix data format is :
  //     [ <start row>-<end row>, <start col>-<end column> ] : <value>
  // This structure can be repeated any many times as nessisary to define your matrix
  // with columns seperating each entry. So for example you can define a 5x5 matrix
  // with 1s in the lower right hand corner and zeros else were as:
  // [ 0-4,0-4 ] : 0, [ 2-4,2-4 ] : 1
  //
  //
  //
  boost::regex rangeset(myROWCOLSET, boost::regex::extended);
  boost::sregex_token_iterator i(lS_line.begin(), lS_line.end(), rangeset, 1);
  boost::sregex_token_iterator j;
  int count=0;
  while (i != j) {
      std::string lC_range = *i;
      boost::smatch lC_mr;
      boost::regex_match( lC_range, lC_mr, rangeset);

      int li_rowstart = 0;
      int li_rowstop  = 0;
      int li_colstart = 0;
      int li_colstop  = 0;
      double ld_value = 0;
      std::istringstream lC_strm;

      // Get the row info
      if ( lC_mr[5].length() != 0 ) {
          // then the row entry is just a single number, which is stored in entry 5.
```

```
        // Note that the number and order of entries in lC_mr depends on the number and order of unique subexpressions that make up
myROWCOLSET
        lC_strm.clear();
        lC_strm.str( std::string(lC_mr[5]) );
        lC_strm >> li_rowstart;
        li_rowstop = li_rowstart;
    } else {
        lC_strm.clear();
        lC_strm.str( std::string(lC_mr[3]) );
        lC_strm >> li_rowstart;
        lC_strm.clear();
        lC_strm.str( std::string(lC_mr[4]) );
        lC_strm >> li_rowstop;
    }

    // Get the col info
    if ( lC_mr[9].length() != 0 ) {
        // then the col entry is just a single number, which is stored in entry 9.
        lC_strm.clear();
        lC_strm.str( std::string(lC_mr[9]) );
        lC_strm >> li_colstart;
        li_colstop = li_colstart;
    } else {
        lC_strm.clear();
        lC_strm.str( std::string(lC_mr[7]) );
        lC_strm >> li_colstart;
        lC_strm.clear();
        lC_strm.str( std::string(lC_mr[8]) );
        lC_strm >> li_colstop;
    }

    // Get the value
    lC_strm.clear();
    lC_strm.str( std::string(lC_mr[10]) ); // entry 10 contains the value, a myNUMBER
```

```
        lC_strm >> ld_value;

        if ( a_value.size1() <= li_rowstop || a_value.size2() <= li_colstop )
            a_value.resize(std::max((int)a_value.size1(),li_rowstop+1),std::max((int)a_value.size2(),li_colstop+1),true);

        for ( int r = li_rowstart; r <= li_rowstop; ++r)
            for ( int c = li_colstart; c <= li_colstop; ++c)
                    a_value(r,c) = ld_value;

        ++count;
        ++i;
    }
    if (count <= 0) return(1);

    return 0;
  } else if ( boost::regex_match(lS_line, lC_smf) ) {
    std::cout << "Recongnized the simple matrix format" << std::endl;
    std::cout << "lS_line = " << lS_line << std::endl;

    boost::regex rowset(mySMFRow, boost::regex::extended);
    boost::sregex_token_iterator i(lS_line.begin(), lS_line.end(), rowset, 1);
    boost::sregex_token_iterator j;
    while (i != j ) {
        std::string   lC_row = *i;
        boost::smatch lC_mr;
        boost::regex_match( lC_row, lC_mr, rowset);

            // Now get the subexpression containing the elements for the current row.
            // The storage of the subexpression in element 2 of the lC_mr object
            // is related to the way subexpressions are nested and defined in the
            // mySMFRow regular expression
        std::vector<fundtype> lC_vec;
        if ( parse_vector_fundtype(lC_mr.str(2), lC_vec) ) {
            std::cout << "class Config : Error : Error while parsing line = " << lS_line << std::endl;
```

```
            std::cout << "class Config : Error : line read from configuation file" << std::endl;
            return 1;
        }

        //    .resize( rows, cols )
        a_value.resize( a_value.size1() + 1, std::max( a_value.size2(), lC_vec.size() ), true);
        for ( int li_k = 0; li_k < lC_vec.size(); ++li_k ) a_value(a_value.size1()-1, li_k) = lC_vec[li_k];

        ++i;
    }

    return 0;
  } else {

    std::ifstream lC_infile(lS_line.c_str());
    if ( !lC_infile.is_open() ) {
        std::cout << "class Config : Error : Unsuccessfully tried to open file to configure matrix, filename = " << lS_line << std::endl;
        return 1;
    }

    std::string lS_rowline;
    for ( getline ( lC_infile, lS_rowline); !lC_infile.eof(); getline(lC_infile, lS_rowline) ){
        std::vector<fundtype> lC_vec;
        if ( parse_vector_fundtype(lS_rowline, lC_vec) ) {
            std::cout << "class Config : Error : Error while parsing line = " << lS_rowline << std::endl;
            std::cout << "class Config : Error : read line from filename = " << lS_line << std::endl;
            return 1;
        }

        //    .resize( rows, cols )
        a_value.resize( a_value.size1() + 1, std::max( a_value.size2(), lC_vec.size() ), true);
        for ( int li_k = 0; li_k < lC_vec.size(); ++li_k ) a_value(a_value.size1()-1, li_k) = lC_vec[li_k];
    }
    return 0;
```

```
  }

  return 1;

}

int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<char>&          a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<unsigned char>&   a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<short>&          a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<unsigned short>&  a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<int>&           a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<unsigned int>&    a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<long>&          a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<long long>&       a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<float>&          a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<double>&         a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }
int Config::set_var(const std::string& aS_varname, boost::numeric::ublas::matrix<long double>&     a_value ) const { return
  set_var_matrix_fundtype(aS_varname, a_value); }

int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<char>&          a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<unsigned char>&  a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
```

```
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<short>&         a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<unsigned short>& a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<int>&           a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<unsigned int>&   a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<long>&          a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<long long>&      a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<float>&         a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<double>&        a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }
int Config::set_var(const char * const acp_varname, boost::numeric::ublas::matrix<long double>&    a_value ) const { return set_var(
  std::string(acp_varname), a_value ); }

const std::string Config::myINT       ("([0-9][0-9]*)");
const std::string Config::myNUMBER    (std::string("([+-]{0,1}") + Config::myINT + std::string("(\\.") + Config::myINT + std::string("){0,1}") +
        std::string("([eE][+-]{0,1}") + Config::myINT + std::string("){0,1}") + std::string(" *)"));
const std::string Config::myRANGE     (std::string("(") +  Config::myINT + std::string(" *: *") + Config::myINT + std::string("|") +  Config::myINT +
        std::string(")"));
const std::string Config::myRANGESET  (std::string("(\\[ *") +  Config::myRANGE + std::string(" *, *") + Config::myNUMBER + std::string(" *\\])") );
const std::string Config::myRANGESETS (Config::myRANGESET + std::string(" *") + std::string("(, *") +  Config::myRANGESET + std::string(" *)*"));
const std::string Config::myVARNAME   ("([a-zA-Z][a-zA-Z0-9_\\.]*)");
const std::string Config::myVARIABLE  ("([a-zA-Z][a-zA-Z0-9_\\.]*) * = *");
const std::string Config::mySimpVec   (Config::myNUMBER + std::string(" *") + std::string("(, *") + Config::myNUMBER + std::string(" *)*"));

const std::string Config::myROWCOLSET (std::string("(\\[ *") + Config::myRANGE + std::string(" *, *") + Config::myRANGE + std::string(" *\\] *: *") +
        Config::myNUMBER + std::string(" *)"));
const std::string Config::myROWCOLSETS (Config::myROWCOLSET + std::string("(, *") + Config::myROWCOLSET + std::string(" *)*"));
```

```
const std::string Config::mySMFRow    (std::string("(\\[ *(") + Config::mySimpVec + std::string(") *\\])") );
const std::string Config::mySMF       (std::string(" *\\[ *") + mySMFRow + std::string(" *") + std::string("(, *") + mySMFRow + std::string(" *)*") +
            std::string(" *\\]"));

const std::string Config::re_variable (myVARIABLE);
const std::string Config::re_rangeset (myRANGESET);
const std::string Config::re_line     (std::string("^") +  myVARIABLE + myRANGESETS);




std::ostream& operator<<(std::ostream& aC_O, const Config& aC_obj)
{
    for ( Config::const_iterator lC_iter = aC_obj.begin(); lC_iter != aC_obj.end(); ++lC_iter )
        aC_O << lC_iter->first << " = " << lC_iter->second << std::endl;
    return aC_O;
}

std::ostream& operator<<(std::ostream& aC_O, const std::vector<float>& aC_obj)
{
    for ( std::vector<float>::const_iterator lC_iter = aC_obj.begin(); lC_iter != aC_obj.end(); ++lC_iter ) {
        aC_O << (*lC_iter);
        if ( (lC_iter + 1) != aC_obj.end() ) {aC_O << ", "; }
    }
    return aC_O;
}
```

V. Source code for systime.h

```
#ifndef sms_systimes_h
#define sms_systimes_h
// __COPYRIGHT_START__
```

```
// Package Name :
// File Name : systimes.h
// Copyright 2003 Scott M. Duke-Sylvester and the ATLSS
// project.
//
//
// All rights reserved. No part of this source code maybe reproduced, stored
// in a retrieval system or transmitted in any form  or by any means, electrontic
// mechanical photocopying recording or otherwise without prior written
// permission of the above mentioned copyright holders.
// __COPYRIGHT_END__
// Unix system includes
#include <sys/times.h>

// C++ includes
#include <iostream>


class Times
{
    public:
        Times()
        {
          initialize();
        }

        ~Times() {}

        Times(const Times& aC_orig)
        {
          copy(aC_orig);
        }

        Times& operator=(const Times& aC_orig)
```

```
    {
      if ( this != &aC_orig )
        copy(aC_orig);
      return *this;
    }

    void start()
    {
      rstart = times(&tmsstart);
    }

    void end()
    {
      rend   = times(&tmsend);
      compute_diff();
    }

    void reset()
    {
      initialize();
    }

    std::ostream& write(std::ostream& aC_O) const
    {
      long clktck = sysconf(_SC_CLK_TCK);
      aC_O << "real      : " << rdiff/double(clktck)            << std::endl;
      aC_O << "user      : " << tmsdiff.tms_utime/double(clktck)  << std::endl;
      aC_O << "sys       : " << tmsdiff.tms_stime/double(clktck)  << std::endl;
      aC_O << "child user : " << tmsdiff.tms_cutime/double(clktck) << std::endl;
      aC_O << "child sys  : " << tmsdiff.tms_cstime/double(clktck) << std::endl;
      return aC_O;
    }

protected:
```

```
void initialize()
{
  tmsstart.tms_utime  = 0;
  tmsstart.tms_stime  = 0;
  tmsstart.tms_cutime = 0;
  tmsstart.tms_cstime = 0;

  tmsend  .tms_utime  = 0;
  tmsend  .tms_stime  = 0;
  tmsend  .tms_cutime = 0;
  tmsend  .tms_cstime = 0;

  tmsdiff .tms_utime  = 0;
  tmsdiff .tms_stime  = 0;
  tmsdiff .tms_cutime = 0;
  tmsdiff .tms_cstime = 0;

  rstart            = 0;
  rend              = 0;
  rdiff             = 0;
}

void copy(const Times& aC_orig)
{
  tmsstart.tms_utime  = aC_orig.tmsstart.tms_utime;
  tmsstart.tms_stime  = aC_orig.tmsstart.tms_stime;
  tmsstart.tms_cutime = aC_orig.tmsstart.tms_cutime;
  tmsstart.tms_cstime = aC_orig.tmsstart.tms_cstime;

  tmsend  .tms_utime  = aC_orig.tmsend  .tms_utime;
  tmsend  .tms_stime  = aC_orig.tmsend  .tms_stime;
  tmsend  .tms_cutime = aC_orig.tmsend  .tms_cutime;
  tmsend  .tms_cstime = aC_orig.tmsend  .tms_cstime;
```

```
        tmsdiff .tms_utime  = aC_orig.tmsdiff .tms_utime;
        tmsdiff .tms_stime  = aC_orig.tmsdiff .tms_stime;
        tmsdiff .tms_cutime = aC_orig.tmsdiff .tms_cutime;
        tmsdiff .tms_cstime = aC_orig.tmsdiff .tms_cstime;

        rstart          = aC_orig       .rstart;
        rend            = aC_orig       .rend;
        rdiff           = aC_orig       .rdiff;

      }

      void compute_diff()
      {
        tmsdiff.tms_utime  += tmsend.tms_utime  - tmsstart.tms_utime;
        tmsdiff.tms_stime  += tmsend.tms_stime  - tmsstart.tms_stime;
        tmsdiff.tms_cutime += tmsend.tms_cutime - tmsstart.tms_cutime;
        tmsdiff.tms_cstime += tmsend.tms_cstime - tmsstart.tms_cstime;
        rdiff           += rend           - rstart;
      }

      struct tms tmsstart, tmsend, tmsdiff;
      clock_t rstart, rend, rdiff;
};

std::ostream& operator<<(std::ostream& aC_O, Times& aC_times)
{
    return aC_times.write(aC_O);
}
#endif
```

VI. Source code for lavegmod.sh

```
#!/bin/sh
```

```
R --no-save --slave --args $1 < /Users/sdukesy/LAVegMod-Devel/LAVegMod/lavegmod.R
```

VII. Source code for lavegmod.R

```
source('~/LAVegMod-Devel/LAVegMod/myGISlib.R');

Scenario        = 'SXX';
Group           = 'GXX';
Variation       = 'VXX';
Year            = 00;
VegRealization   = 'RXX';
Realization      = 'RXX';
regionList       = c('PB', 'AA', 'CP');
tablePath        = '/Users/sdukesy/LAVegMod-Devel/Data/VegMod_CRMSHydroVeg_Analysis_WB';

#config = function()
#{
   #args = commandArgs(trailingOnly=TRUE);
#
   #filenameConfig = args[1];
   #source(filenameConfig,local=TRUE);
   #assign('Scenario'   , Scenario   , envir=globalenv() );
   #assign('Group'      , Group      , envir=globalenv() );
   #assign('Variation'  , Variation  , envir=globalenv() );
   #assign('Year'       , Year       , envir=globalenv() );
   #assign('regionList' , regionList , envir=globalenv() );
   #assign('Realization', Realization, envir=globalenv() );
   #if ( sum(ls()%in%'TablePath')  ) { assign ('TablePath', TablePath, envir=globalenv() ); };
#}

args = commandArgs(trailingOnly=TRUE);
filenameConfig = args[1];
source(filenameConfig)
```

```
#source('lavegmod.config');

if ( VegRealization == 'RXX' ) { VegRealization = Realization }


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

templateFilename = '<SCENARIO>_<GROUP>_<VARIATION>_<MODEL><REGION><FILE><YEAR><REALIZATION><STITCHED_SPECIAL><EXT>';
make.filename = function(S=Scenario, G=Group, V=Variation, M='DELAY', Reg='DELAY', F='DELAY', Y=Year, Real=Realization, E='DELAY', Sp='',
        temp=templateFilename)
{
   ret = temp;
   if ( toupper(S)     != 'DELAY' ) { ret = sub('<SCENARIO>',    S,    ret); }
   if ( toupper(G)     != 'DELAY' ) { ret = sub('<GROUP>',       G,    ret); }
   if ( toupper(V)     != 'DELAY' ) { ret = sub('<VARIATION>',   V,    ret); }
   if ( toupper(M)     != 'DELAY' ) { ret = sub('<MODEL>',       M,    ret); }
   if ( toupper(Reg)   != 'DELAY' ) { ret = sub('<REGION>',      Reg,  ret); }
   if ( toupper(F)     != 'DELAY' ) { ret = sub('<FILE>',        F,    ret); }
   if ( toupper(Y)     != 'DELAY' ) { ret = sub('<YEAR>',        Y,    ret); }
   if ( toupper(Real)  != 'DELAY' ) { if ( Real != '' ) { ret = sub('<REALIZATION>'     , '_<REALIZATION>'     , ret); }; ret = sub('<REALIZATION>'     , Real, ret); }
   if ( toupper(Sp)    != 'DELAY' ) { if ( Sp   != '' ) { ret = sub('<STITCHED_SPECIAL>', '_<STITCHED_SPECIAL>', ret); }; ret = sub('<STITCHED_SPECIAL>', Sp  ,
        ret); }
   if ( toupper(E)     != 'DELAY' ) { if ( E    != '' ) { ret = sub('<EXT>'            , '.<EXT>'            , ret); }; ret = sub('<EXT>'            , E   , ret); }

   return(ret);
}


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

```
fullRegionList = c('PB', 'AA', 'CP');

ftpHost='ftp.dnr.state.la.us/Master-Plan-Modeling/ModelRuns';
ftpUser='MPM';
ftpPasswd='dnrftp.2010';


prefix     = paste(Scenario,'_',Group,'_',Variation,'_',sep='');
suffix     = paste('_',Realization,sep='');


Year = as.integer(Year);

filenameSteps              = paste(prefix,'Vsteps',Year,'.txt',sep='');

templateEcoHydroSAL        = make.filename(G='DELAY', M='H', Reg='DELAY', F='SAL', Y='DELAY', Real = 'DELAY', E='txt');
templateEcoHydroSTG        = make.filename(G='DELAY', M='H', Reg='DELAY', F='STG', Y='DELAY', Real = 'DELAY', E='txt');
templateEcoHydroTMP        = make.filename(G='DELAY', M='H', Reg='DELAY', F='TMP', Y='DELAY', Real = 'DELAY', E='txt');
templateShapefile          = make.filename(G='DELAY', M='H', Reg='DELAY', F='SHP', Y='DELAY', Real = 'DELAY', E='DELAY');
templateProjectFile        = make.filename(G='DELAY', M='H', Reg='DELAY', F='SHP', Y='DELAY', Real = 'DELAY', E='DELAY');
extList                    = c('dbf','prj','sbn','sbx','shp','shx');



filenameLandwater          = '~/LAVegMod-Devel/Data/Land_Water/S00_G00_V00_WLAPCL00_500m-2010.img'
filenameElevation          = '~/LAVegMod-Devel/Data/Land_Water/S00_G00_V00_WLAELV00_500m-2010.img'
filenameCLEARGrid          = '~/LAVegMod-Devel/Data/CLEARGRidCellID/Clear_Domain_500m_Grid.asc';
filenameShapeLA            = '~/LAVegMod-Devel/GIS/lacountiesp/la_countiesp';

templateLandwaterXLSX_NSTD   = paste(prefix,'WLALNW__YEAR___500m-__YEARRANGE__.xlsx',sep='');
templateElevationXLSX_NSTD   = paste(prefix,'WLAELV__YEAR___500m-__YEARRANGE__.xlsx',sep='');

templateElevationIMG       = make.filename(M='D', Reg='LA', F='ELV', Y='DELAY', E='img');
templateLandwaterIMG       = make.filename(M='D', Reg='LA', F='PCL', Y='DELAY', E='img');
```

```
templateElevationASC        = make.filename(M='D', Reg='DELAY', F='elv', Y='DELAY', E='asc');
templatePCLandwaterASC      = make.filename(M='D', Reg='DELAY', F='percentland', Y='DELAY', E='asc');

templateElevationCSV        = make.filename(M='D', Reg='DELAY', F='ELV', Y='DELAY', E='csv');
templateLandwaterCSV        = make.filename(M='D', Reg='DELAY', F='LNW', Y='DELAY', E='csv');

templateInitCond            = make.filename(         M='V', Reg='DELAY', F='InitCond', Y='DELAY', E='asc+');

templateClearGrid           = make.filename(M='V', Reg='DELAY', F='ClearGrid', Y='DELAY', E='asc');
templateBoxID               = make.filename(M='V', Reg='DELAY', F='BoxID'    , Y='DELAY', E='asc');

templateWaveAmp             = make.filename(M='H', Reg='DELAY', F='waveamp', E='asc')
templateMeanSal             = make.filename(M='H', Reg='DELAY', F='meansal', E='asc')
templateSumMeanDepth        = make.filename(M='H', Reg='DELAY', F='sumMeanDepth', E='asc');
templateSumMeanSal          = make.filename(M='H', Reg='DELAY',   F='sumMeanSal', E='asc');
templateSumMeanTemp         = make.filename(M='H', Reg='DELAY',   F='sumMeanTemp', E='asc');

filenameConfigtemplate      = '~/LAVegMod-Devel/LAVegMod/lavegmod_template.config';
templateConfig              = paste('lavegmod_',make.filename(M='V', Reg='DELAY', Y='DELAY', F="config", E='config'),sep='');

templateVG                  = make.filename(G='DELAY', M='V', Reg='DELAY', F='VG'      , Y='DELAY', Real='DELAY', E='asc+');
templateVEG                 = make.filename(         M='V', Reg='DELAY', F='VEG'     , Y='DELAY', Real='DELAY', E='asc+');

templatePDF                 = make.filename(M='V', Reg='DELAY', F='PDF', Y='DELAY', Real=VegRealization, E='pdf');

##############################################################################
##############################################################################
# Utility functions
#
##############################################################################

read.steps = function(filename)
{
```

```
   ret = list();
   localNames = c('localNames', ls());


   source(filename, local=TRUE);
   for ( var in ls()[ !(ls()%in%localNames) ] ) {
      ret[[var]] = get(var, pos=-1);
   }
   return(ret);
}

write.steps = function(steps, filename)
{
   cat('', file=filename,sep='');
   stepOrder = as.integer(sub(    'Step', '', sub('_.*$', '', names(steps))     ));
   index = sort.int(stepOrder, index.return=TRUE);
   for ( var in names(steps)[index$ix] ) {
      cat(var,' = ',steps[[var]],'\n',sep='',append=TRUE,file=filename);
   }
}
file.exists = function(filename)
{
   con = try( file(filename, open='r'), silent=TRUE );
   if ( class(con)[1] == 'try-error' ) {
      return(FALSE);
   } else {
      close(con);
   }
   return(TRUE);
}

link_byregion = function(f_template, t_template)
{
   err = 0;
```

```
  for ( region in regionList ) {
    f = f_template;
    f = sub('__REGION__',region, f);
    f = sub('__YEAR__',  Year,   f);

    t = t_template;
    t = sub('__REGION__',region, t);
    t = sub('__YEAR__',  Year,   t);

    if ( !file.exists(t) ) {
      cmd = paste('ln -s ',f,' ',t,sep='');
      cat(cmd,'\n');
      if ( system(cmd, intern=FALSE) ) {
        cat('Error: link(): command failed: ',cmd,'\n');
        err = err + 1;
      }
    }

  }

  return(err);

}

check_shapefiles = function()
{
  err = 0;
  for ( region in regionList ) {
    for ( ext in extList ) {
      f   = templateShapefile;
      f   = sub('__REGION__',region,f);
      f   = sub('__YEAR__',  Year,  f);
      f   = sub('__EXT__',   ext,   f);
      if ( !file.exists(f) ) {
```

```
            cat('Error: check_shapefiles(): Could not find (or open) ',f,'\n');
            err = err + 1;
         }
      }
   }

   return(err);
}

ftp_file = function(f)
{
   err = 0;
   errList = list();
   cmd       = paste('ftp ftp://',ftpUser,':',ftpPasswd,'@',ftpHost,'/',Scenario,'/',Scenario,'_',Group,'/',f,sep='');
   system_err = system(cmd, intern=FALSE);
   if ( system_err ) {
      err = err + 1;
      errList[err] = paste('Error: ftp_file(): Could not download file ',f);
   }

   attr(err, 'errList') = errList;
   return(err);
}

link_file = function(file, link)
{
   err = 0;
   errList = list();

   cmd = paste('ln -s ',file,' ',link,sep='');

   system_err = system(cmd, intern=FALSE);
   if ( system_err ) {
      err = err + 1;
```

```
      errList[err] = paste('Error: link_file(): Could not link file:',file,' to link:',link,sep='');
   }

   attr(err, 'errList') = errList;
   return(err);

}

##############################################################################
##############################################################################
# FTP functions
#
##############################################################################

ftp_shapefiles = function()
{

   err = 0;
   for ( region in fullRegionList ) {
      for ( ext in extList ) {
         # Step 1: If the current configuration is for Year == 10, the R00 realization and
         # the file we want is in the regionList, then go get it from the FTP site.
         if ( Year == 10 && Realization == 'R00' && sum( regionList%in%region ) ) {
            f = make.filename(Reg=region, E=ext, temp=templateShapefile);
            ret = ftp_file(f);

            if ( ret ) {
               err = err + 1;
               attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
            }
            next;
         }

         # Step 2: One of the conditions above failed. That means that the file we want
```

```
        # does not exist on the MPM ftp cite. However, the information that we want
        # is in an existing file, so we will create a link to the existing file with the
        # filename that we want, so that everything after this point can proceed as normal.
        #
        # There are two important catagories at this point. Either the file we want
        # is associated with a region in the regionList, or is not.
        # If it is not, then we link to the data is ../Sss_G01_R00/Sss_G01_V00_H<REGION>SHP10.<EXT>
        # If the file is in the region list, then we want to link to ../Sss_Ggg_R00/Sss_Ggg_V00_H<REGION>SHP10.<EXT>
        # were ss and gg represent the current scenario and group we are processesing.
        file = '';
        if ( sum( regionList%in%region ) ) {
           file = make.filename(Reg=region, Y=10, Real='R00', E=ext, temp=templateShapefile);
           file = paste('../',Scenario,'_',Group,'_R00/',file,sep='');

        } else {
           file = make.filename(G='G01', Reg=region, Y=10, Real='R00', E=ext, temp=templateShapefile);
           file = paste('../',Scenario,'_G01_R00/',file,sep='');

        }

        link = make.filename(M='H', Reg=region, F='SHP', Y=Year, E=ext, temp=templateShapefile);

        ret = link_file(file,link);
        if ( ret ) {
           err = err + 1;
           attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
        }

      }

   }

   return(err);
}
```

```
ftp_projectfiles = function()
{
templateProjectFile        = '~/LAVegMod-Devel/GIS/GIS_DATA_TEST_ALT_Nov2011/GIS_LA_Polys';
}

ftp_ecohydrofiles = function()
{
   err = 0;
   for ( region in fullRegionList ) {
      for ( fileType in c(templateEcoHydroSAL, templateEcoHydroSTG, templateEcoHydroTMP) ) {
         if ( sum(regionList%in%region) ) {
            file = make.filename(M='H', Reg=region, E='txt', temp=fileType );
            ret = ftp_file(file);
            if ( ret ) {
               err = err + 1;
               attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
            }
         } else {
            file = make.filename(G='G01', M='H', Reg=reg, Real='R00', E='txt', temp=fileType);
            file = paste('../',Scenario,'_G01_R00',file,sep='');

            link = make.filename(M='H', Reg=region, E='txt', temp=fileType);
            ret = link_file(file, link);
            if ( ret ) {
               err = err + 1;
               attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
            }
         }
      }
   }
   return(err);
}
```

```
ftp_stitchedfiles = function()
{

   err = 0;
   if ( Year == 10 ) {
      link = make.filename(temp = templateLandwaterIMG);
cat('link = ',link,'\n');
      ret = link_file(filenameLandwater, link);
      if ( ret ) {
         err = err + 1;
         attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
      }

      link = make.filename(temp = templateElevationIMG);
cat('link = ',link,'\n');
      ret = link_file(filenameElevation, link)
      if ( ret ) {
         err = err + 1;
         attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
      }

      return(ret);
   }


   errList = list();
   attr(err, 'errList') = errList;

   # Download stitched files.
   file = make.filename(temp = templateLandwaterIMG);
   ret = ftp_file(file);
   if ( ret ) {
      err = err + 1;
      attr(err, 'errList')[err] = attr(ret, 'errList')[[1]];
```

```
  }

  file = make.filename(temp = templateElevationIMG);
  ret = ftp_file(file);
  if ( ret ) {
    err = err + 1;
    errList[err] = attr(ret, 'errList')[[1]];
  }

  return(err);
}

################################################################################
################################################################################
# Create functions
#
################################################################################

create_cleargrid = function()
{
  # Check
  err = 0;
  ag_grid500 = read.asciigrid(filenameCLEARGrid);
  shapeLA    = read.shapefile(filenameShapeLA);
  shapeLA    = transform.shapefile(shapeLA, '+proj=longlat', '+proj=utm +zone=15 +north');

  ehShapeList = list();

  if ( Year == 10 ) {
    for ( region in fullRegionList ) {
      if ( sum(regionList%in%region) ) {

        cat('Building ',region,' clear grid file\n');
        sf = make.filename(Reg=region, Y=Year, E='', temp=templateShapefile);
```

```
        cg = make.filename(Reg=region, Y=Year, temp=templateClearGrid);

        cat('Read shapefile = ',sf,'\n',sep='');
        shapeEH = read.shapefile(sf);
        ehShapeList[[region]] = shapeEH;

        ag_grid500_clip = clip.grid2shape(ag_grid500,     shapeEH);
        ag_grid500_clip = clip.grid2shape(ag_grid500_clip, shapeLA);
        header         = attr(ag_grid500_clip, 'header');
        ag_grid500_clip = matrix( as.integer(ag_grid500_clip), nrow=nrow(ag_grid500_clip), ncol=ncol(ag_grid500_clip) );
        attr(ag_grid500_clip, 'header') = header;
        write.asciigrid(ag_grid500_clip, filename=cg);
        remove(ag_grid500_clip);
      } else {
        sf = make.filename(Reg=region, Y=Year, E='');
        shapeEH = read.shapefile(sf);
        ehShapeList[[region]] = shapeEH;
      }
    }

    cat('Building LA clear grid file\n');
    cg = make.filename(Reg='LA', Y=Year, temp = templateClearGrid);

    ag_grid500_clip = clip.grid2shapelist(ag_grid500, ehShapeList);
    ag_grid500_clip = clip.grid2shape(ag_grid500_clip, shapeLA);
    header         = attr(ag_grid500_clip, 'header');
    ag_grid500_clip = matrix( as.integer(ag_grid500_clip), nrow=nrow(ag_grid500_clip), ncol=ncol(ag_grid500_clip) );
    attr(ag_grid500_clip, 'header') = header;
    write.asciigrid(ag_grid500_clip, filename=cg);

    return(0);
  }

  for ( region in c('LA', fullRegionList) ) {
```

```
      file = make.filename(Reg = region, Y=10,   temp = templateClearGrid);

      link = make.filename(Reg = region, Y=Year, temp = templateClearGrid);
      link_file(file, link);
   }

   return(err);
}


create_boxid = function()
{
   # Check
   shapeLA    = read.shapefile(filenameShapeLA);
   shapeLA    = transform.shapefile(shapeLA, '+proj=longlat', '+proj=utm +zone=15 +north');

   if ( Year == 10 ) {
      for ( region in regionList ) {
         f = make.filename(Reg = region, Y = Year, temp = templateClearGrid);
         ag_grid500_clip = read.asciigrid(f);

         s = make.filename(Reg = region, Y = Year, E='', temp = templateShapefile);
         shape = read.shapefile(s);

         filenameBoxID = templateBoxID;
         filenameBoxID = make.filename(Reg = region, Y = Year, temp = templateBoxID);

         header = attr(ag_grid500_clip,'header');

         index = 1:(header$nrows * header$ncols) - 1;
         row   = floor( index/header$ncols ) + 1;
         col   = index%%header$ncols + 1;

         xcent = header$xllcorner + header$cellsize*(col - 0.5);
```

```
        ycent = header$yllcorner + header$cellsize*(row - 0.5);

        index = point.in.shapefile(shape, coords=data.frame(x=xcent, y = ycent), shapeIDname='BoxID' );

        row = row[index];
        col = col[index];
        BoxID = as.integer( sub(region,'', attr(index,'shape_ID')[index])  );

        for ( i in 1:length(index) ) {
           r = row[i];
           c = col[i];
           b = BoxID[i];
           ag_grid500_clip[r,c] = b;
        }

        ag_grid500_clip = clip.grid2shape(ag_grid500_clip, shapeLA);

        write.asciigrid(ag_grid500_clip,filename=filenameBoxID);
     }
     return(0);
  }

  for ( region in regionList ) {

     file = make.filename(Reg = region, Y = 10,   temp = templateBoxID);

     link = make.filename(Reg = region, Y = Year, temp = templateBoxID);
     link_file(file, link);
  }

  return(0);

}
```

```
create_landwaterfileASC = function()
{
   err = 0;

   f = make.filename(Y = Year, temp = templateLandwaterIMG);
   data = read.erdas(f);

   shapeLA    = read.shapefile(filenameShapeLA);
   shapeLA    = transform.shapefile(shapeLA, '+proj=longlat', '+proj=utm +zone=15 +north');

   for ( region in regionList ) {
      cat('Building ',region,' landwater file\n');
      sf = make.filename(Reg = region, Y = Year, E='', temp = templateShapefile);
      shapeEH = read.shapefile(sf);

      f = make.filename(Reg = region, Y = Year, temp = templatePCLandwaterASC);


      final = clip.grid2shape(data, shapeEH);
      final = clip.grid2shape(final,shapeLA);
      write.asciigrid(final, filename=f);

      f = make.filename( Reg = region, Y = Year, temp = templateClearGrid);

      cg = read.asciigrid(f);

      index = which ( !is.na(cg), arr.ind=TRUE);

      final_df = data.frame(CELL_ID=cg[index], LAND_SQKM=0.25*(final[index]/100), WATER_SQKM=0.25*(1.0-(final[index]/100)) );

      f = make.filename( Reg = region, Y = Year, temp = templateLandwaterCSV);
      write.csv(final_df, f, row.names=FALSE);

   }
```

```
   return(err);

}

create_elevationfileASC = function()
{
   err = 0;

   f = make.filename(Y = Year, temp = templateElevationIMG);
   data = read.erdas(f);

   shapeLA    = read.shapefile(filenameShapeLA);
   shapeLA    = transform.shapefile(shapeLA, '+proj=longlat', '+proj=utm +zone=15 +north');

   for ( region in regionList ) {
      cat('Building ',region,' elevation file\n');
      sf = make.filename( Reg = region, Y = Year, E='', temp = templateShapefile);

      f = make.filename( Reg = region, Y = Year, temp = templateElevationASC);

      shapeEH = read.shapefile(sf);

      final = clip.grid2shape(data, shapeEH);
      final = clip.grid2shape(final,shapeLA);
      write.asciigrid(final, filename=f);

   }
   return(err);
}

#create_wetlandmorphfilesCSV = function()
#{
#    err = 0;
#    if ( Year == 10 ) {
```

```
#       err = err + link_byregion(filenameLandwater, templateLandwaterCSV);
#       err = err + link_byregion(filenameElevation, templateElevationCSV);
#       return(err);
#   }
#
#   yearrange = paste( (2000 + Year)-5,'-',2000+Year,sep='');
#
#   f = templateLandwaterXLSX_NSTD;
#   f = sub('__YEAR__',      Year,      f);
#   f = sub('__YEARRANGE__', yearrange, f)
#   t = sub('xlsx$','csv', f);
#   cmd = paste('xlsx2csv.py ',f,' ',t,sep='');
#   if ( system(cmd, intern=FALSE ) ) {
#       cat('Error: link_welandmortphfiles(): Could not convert xlsx to csv with command:',cmd,'\n');
#       return(1);
#   }
#   f = t;
#   err = err + link_byregion(f, templateLandwaterCSV);
#
#   f = templateElevationXLSX_NSTD;
#   f = sub('__YEAR__',      Year,      f);
#   f = sub('__YEARRANGE__', yearrange, f)
#   t = sub('xlsx$','csv', f);
#   cmd = paste('xlsx2csv.py ',f,' ',t,sep='');
#   if ( system(cmd, intern=FALSE ) ) {
#       cat('Error: link_welandmortphfiles(): Could not convert xlsx to csv with command:',cmd,'\n');
#       return(1);
#   }
#   f = t;
#   err = err + link_byregion(f, templateElevationCSV);
#
#
#   return(err);
#}
```

```
create_initialconditions = function()
{

   # This function creates the initial conditions for the LAVegMod.
   # The initial conditions are stored in an asciigridplus file.


   err = 0;

   # Load the Louisiana shapefile. This is used in clipping the initial conditions to the
   # correct shape.
   shapeLA = read.shapefile(filenameShapeLA);
   shapeLA = transform.shapefile(shapeLA, '+proj=longlat', '+proj=utm +zone=15 +north');

   # Load the projects shapefile. This data is used to set certain areas to a particular vegetation type
   # based on the construction of various projects.
   shapeProj = read.shapefile(templateProjectFile);

   # Load the list of vegetation type names used in the initial conditions data file.
   # This list also contains the list of vegetation types that will actuallly be used in the model
   # and maps between the long names and the abbreviations used by the veg model.
   filename_habtable = '~/LAVegMod-Devel/GIS/Grids/Vegetation classes.csv';
     habtable         = read.csv(filename_habtable);
     habtable$VegABR   = defactorize(habtable$VegABR);

   if ( Year == 10 ) {
      # If Year == 10, then we are running the first 25 years of the simulation. In this case
      # we want to load the base, year + 1 day data.
      # This data is stored in a csv that describes the area, in km^2, of each CLEAR grid cell that is occupied by
      # each of the vegetation types.
      # This information is coordinated with the CLEAR grid to produce the appropriate spatial
      # data layer used to initialize the model.
      filename_habtypes = '~/LAVegMod-Devel/GIS/Grids/Master_Plan_500m_grid_base_yr_2010_lulc_area_summ_021311_final.csv';
        habtypes         = read.csv(filename_habtypes);
```

```
} else {
   # If Year != 10, then we are running the second 25 years of the simulation. In this case
   # we want to load the output from the last step of the model run from the first 25 year simulation.
   # This information is stored in an asciigrid plus file. All we really need is the table of
   # cover values stored in the asciigrid plus file.
   # The data is stored as a fraction of each CLEAR grid covered by each type.
   # This is converted to area in km^2 before moving on.
   filename_habtypes = make.filename(Reg = 'LA', Y = Year, Real = VegRealization, temp = templateVEG);
   habtypes = read.asciigridplus(filename_habtypes);
   habtypes = habtypes$table;
   for ( name in names(habtypes) ) {
      if ( name != 'CELLID' ) {
         habtypes[[name]] = habtypes[[name]]*0.25;
      }
   }
}

# For each region that we are going to run, we need to create an initial condition file.
for ( region in regionList ) {

   # Load the shapefile for the region
   filenameSHP     = make.filename(Reg = region, Y = Year, E='', temp = templateShapefile);
   shape           = read.shapefile(filenameSHP);

   # Load the clear grid for the region
   filenameCG      = make.filename(Reg = region, Y = Year,      temp = templateClearGrid);
      landwater       = read.csv(filenameLW);

   # Load the land/water file for the region
   filenameLW      = make.filename(Reg = region, Y = Year,      temp = templateLandwaterCSV);
   ag_grid500_clip = read.asciigrid(filenameCG);
```

```
# Make a list of all the cell IDs
cat('Getting cellids\n');
#cellids          = sort(ag_grid500_clip[ag_grid500_clip != -9999]);
cellids          = sort(ag_grid500_clip[!is.na(ag_grid500_clip) ]);

# Make a list indicies of all the vegetation types that we want to model.
vegtypeindex      = habtable$VegABR != '';

cat('getting habtypes\n');
if ( Year == 10 ) {
   habtypes_sub       = habtypes[vegtypeindex];
   names(habtypes_sub) = habtable$VegABR[vegtypeindex];
   habtypes_sub       = subset(habtypes_sub, CELLID%in%cellids);
   habtypes_sub$SAV    = rep(0,length(habtypes_sub$CELLID));
   habtypes_sub$NOTMOD = rep(0,length(habtypes_sub$CELLID));

   cat('setting notmod\n');
   cat('step 1\n');
   notmodindex        = !vegtypeindex;
   cat('step 2\n');
   notmodindex[1]      = TRUE; # Because the first element is the CELLID and we need that.
   cat('step 3\n');
   notmodindex[length(notmodindex)] = FALSE; # Because the last element is the total, which we don't want.
   cat('step 4\n');
   habtypes_notmod     = habtypes[notmodindex];
   cat('step 5\n');
   names(habtypes_notmod)[[1]] = 'CELLID';
   cat('step 6\n');
   habtypes_notmod     = subset(habtypes_notmod, CELLID%in%cellids);
   cat('step 7\n');
   #habtypes_sub$NOTMOD = ddply(habtypes_notmod, .(CELLID), sum)$V1 - habtypes_notmod$CELLID;
   habtypes_notmod     = as.matrix(habtypes_notmod);
   habtypes_notmod[,1] = 0;
   habtypes_sub$NOTMOD = apply(habtypes_notmod, 1, sum);
```

```
} else {
   habtypes_sub       = habtypes;
   habtypes_sub       = subset(habtypes_sub, CELLID%in%cellids);
}


habtypes_sub       = as.matrix(habtypes_sub);

cat('subsetting landwater\n');
IDColName     = names(landwater)[1];
landwater_sub = subset(landwater, IDColName%in%cellids);


# If there is a conflict between the veg data and the land/water data,
# we will use the values in the land/water data.
index     = which( (habtypes_sub[,'WATER']+habtypes_sub[,'SAV']) != landwater_sub[[3]], arr.ind=TRUE);
col_land  = !(   attr(habtypes_sub,'dimnames')[[2]]%in%c('CELLID','WATER','SAV')   );

cat('adjusting land/water\n');
cat('length(index) = ',length(index),'\n');
for ( i in index ) {
   # If the area covered by water in the land/water data is
   # greater than the area in the habitat type data, then
   # set the water area in the veg data equal to the value
   # in the land/water data, and scale back all the other
   # types by the amount of area in land.
   # Otherwise, expand the cover of the veg types
   # by the marginal increase in land.

   # If land has been lost, then decrease the cover of each of the terrestrial types
   # by the fraction of land lost.
   # In this case, water has been added. Increase the cover of water and SAV
```

```
    curWater = habtypes_sub[i, 'WATER'] + habtypes_sub[i,'SAV'];
    curLand  = sum( habtypes_sub[i, col_land ] );

    newWater = landwater_sub[[3]][i];
    newLand  = landwater_sub[[2]][i];

    if ( curLand < newLand ) {
       deltaLand = newLand - curLand;
       habtypes_sub[i,'BAREGRND'] = deltaLand;

       # curWater > newWater
       scaleWater            = newWater/curWater;
       habtypes_sub[i,'WATER'] = habtypes_sub[i,'WATER'] * scaleWater;
       habtypes_sub[i,  'SAV'] = habtypes_sub[i,  'SAV'] * scaleWater;
    } else {
       scaleLand               = newLand/curLand;
       habtypes_sub[i, col_land] = habtypes_sub[i, col_land] * scaleLand;

       deltaWater              = newWater - curWater;
       habtypes_sub[i, 'WATER']  = habtypes_sub[i,'WATER'] + deltaWater;
    }
}

# This converts everything from area in km^2, to a fractional occupancy of the cell.
# Don't change columns 1 & 2
# Column 1 is the cell id
# Column 2 is bare ground, this will be fixed in the next step.
cat('converting to fraction as int\n');
habtypes_sub[,3:ncol(habtypes_sub)] = habtypes_sub[,3:ncol(habtypes_sub)]/0.25 * 1E6;

# If any area in the cell is not classified as a veg type or water
# then it is bare ground.
cat('adjusting bare ground\n');
#for ( i in 1:nrow(habtypes_sub) ) {
```

```
        #habtypes_sub[i,2] = (1E6 - as.integer(   sum(habtypes_sub[i,3:ncol(habtypes_sub)])   ));
      #}
      #temp = habtypes_sub;
      #temp[,1:2] = 0;
      #s = apply(temp, 1, sum);
      #habtypes_sub[,2] = 1E6 - as.integer(s);


      cat('converting faction as float\n');
      habtypes_sub[,2:ncol(habtypes_sub)] = habtypes_sub[,2:ncol(habtypes_sub)]*1.0E-6;

      cat('formatting for output\n');
      habtypes_formatted     = signif(habtypes_sub, digits=6);
      habtypes_formatted[,1] = habtypes_sub[,1];

      cat('writing output\n');
      filename_ascp = make.filename(Reg = region, Y = Year, temp = templateInitCond);

      header = attr(ag_grid500_clip,'header');
      ag_grid500_clip = matrix( as.integer(ag_grid500_clip), nrow=nrow(ag_grid500_clip), ncol=ncol(ag_grid500_clip) );
      attr(ag_grid500_clip, 'header') = header;
      write.asciigrid(ag_grid500_clip, filename_ascp);
      write.table(habtypes_formatted,filename_ascp,row.names=FALSE, sep=',',append=TRUE );

    }

      return(err);

}


create_configfile = function()
{
    for ( region in regionList ) {
```

```
i = file(filenameConfigtemplate, open='r');

f = make.filename(Reg = region, Y = Year, temp = templateConfig);

out = file(f, open='w');

line = readLines(i);
while ( length(line) ) {
   line = sub('__SCENARIO__'   , Scenario            , line);
   line = sub('__GROUP__'      , Group               , line);
   line = sub('__YEAR__'       , Year                , line);
   line = sub('__REGION__'     , region              , line);
   line = sub('__VARIATION__'  , Variation           , line);
   line = sub('__REALIZATION__', Realization         , line);
   line = sub('__STARTYEAR__'  , paste(2000+Year,sep=''), line);
   line = sub('__TABLEPATH__'  , tablePath           , line);

   line = sub('__EcoHydroBoxIDFile__'       , make.filename(Reg = region, temp = templateBoxID         ), line);
   line = sub('__EcoHydroStgHeightFile__'   , make.filename(Reg = region, temp = templateEcoHydroSTG    ), line);
   line = sub('__EcoHydroSalinityFile__'    , make.filename(Reg = region, temp = templateEcoHydroSAL    ), line);
   line = sub('__EcoHydroTempFile__'        , make.filename(Reg = region, temp = templateEcoHydroTMP    ), line);
   line = sub('__WetlandMorphCellIDFile__'  , make.filename(Reg = region, temp = templateClearGrid      ), line);
   line = sub('__WetlandMorphLandWaterFile__', make.filename(Reg = region, temp = templateLandwaterCSV   ), line);
   line = sub('__WetlandMorphElevationFile__', make.filename(Reg = region, temp = templateElevationCSV   ), line);
   line = sub('__InitialConditionFile__'    , make.filename(Reg = region, temp = templateInitCond       ), line);
   line = sub('__WaveAmplitudeFile__'       , make.filename(Reg = region, temp = templateWaveAmp        ), line);
   line = sub('__MeanSalinityFile__'        , make.filename(Reg = region, temp = templateMeanSal        ), line);
   line = sub('__ElevationFile__'           , make.filename(Reg = region, temp = templateElevationASC   ), line);
   line = sub('__SummerMeanWaterDepthFile__' , make.filename(Reg = region, temp = templateSumMeanDepth   ), line);
   line = sub('__SummerMeanSalinityFile__'  , make.filename(Reg = region, temp = templateSumMeanSal     ), line);
   line = sub('__SummerMeanTempFile__'      , make.filename(Reg = region, temp = templateSumMeanTemp    ), line);

   line = sub('__OutputFile__'              , make.filename(Reg = region,          Real = VegRealization, temp = templateVEG  ), line); # VEG file
   line = sub('__OutputTemplate__'          , make.filename(Reg = region, Y='DELAY', Real = VegRealization, temp = templateVG   ), line); #VG file
```

```
        writeLines(line, out);
        line = readLines(i);
      }

      close(i);
      close(out);
    }
}

run_model = function()
{
    err = 0;
    for ( region in regionList ) {
      f = make.filename(Reg = region, temp = templateConfig);
      cmd = paste('~/LAVegMod-Devel/LAVegMod/lavegmod ',f,sep='');
      cat('Running model with : ',cmd,'\n\n');
      if ( system(cmd, intern=FALSE) ) {
        cat('Error: run_model(): Could not run model for region ',region,'\n');
        err = err + 1;
      }
    }

    iYear = as.integer(Year);

    for ( region in fullRegionList ) {

      if ( !( region%in%regionList ) ) {
        for ( year in seq(iYear+5,iYear+25,by=5) ) {
          s = make.filename(G='G01', Reg = reg, Y = year, Real = VegRealization, temp = templateVG);

          t = make.filename(Reg = region, T = year, Real = VegRealization, temp = templateVG);

          cmd = paste('ln -s ../G01/',s,' ',t,sep='');
```

```
            if ( system(cmd, intern=FALSE) ) {
               cat('Error: run_model(): command filed: ',cmd,'\n');
               err = err + 1;
            }
         }

         s = make.filename(G = 'G01', Reg = region, Y = Year, Real = VegRealization, temp = templateVEG);

         t = make.filename(Reg = region, Y = Year, Real = VegRealization, temp = templateVEG);

         cmd = paste('ln -s ../G01/',s,' ',t,sep='');
         if ( system(cmd, intern=FALSE) ) {
            cat('Error: run_model(): command filed: ',cmd,'\n');
            err = err + 1;
         }
      }
   }


   return(err);
}

create_wholecoastlinefiles = function(YearList)
{


   for ( year in YearList ) {
      fList = list( PB=make.filename(Reg='PB', Real = VegRealization, temp = templateVG, Y=year),
                    AA=make.filename(Reg='AA', Real = VegRealization, temp = templateVG, Y=year),
                    CP=make.filename(Reg='CP', Real = VegRealization, temp = templateVG, Y=year),
                    LA=make.filename(Reg='LA', Real = VegRealization, temp = templateVEG, Y=year) );


      cat('Reading PB data from : ',fList$PB,'\n');
```

```
      PB = read.asciigridplus(fList$PB);
      cat('Reading AA data from : ',fList$AA,'\n');
      AA = read.asciigridplus(fList$AA);
      cat('Reading CP data from : ',fList$CP,'\n');
      CP = read.asciigridplus(fList$CP);

      cg = make.filename(Reg='LA', Y=Year, temp = templateClearGrid);
      CLEARGrid = read.asciigrid(cg);

      cellID = AA$table$CELLID;

      m = rbind(subset( PB$table, !CELLID%in%cellID), AA$table, subset(CP$table, !CELLID%in%cellID) );

      header = attr(CLEARGrid, 'header');
      CLEARGrid = matrix( as.integer(CLEARGrid), nrow=nrow(CLEARGrid), ncol=ncol(CLEARGrid) );
      attr(CLEARGrid, 'header') = header;
      write.asciigrid(CLEARGrid, fList$LA);
      write.table(m, fList$LA, append=TRUE, row.names=FALSE, sep=',');
   }
}

create_PDF = function(template, region)
{
   i = make.filename(Reg = region, Y = Year + 25,  temp = template);

   o = make.filename(Reg = region, Y = Year + 25, Real=VegRealization, temp = templatePDF);

   LA = read.shapefile(filenameShapeLA);
   LA = transform.shapefile(LA,'+proj=longlat','+proj=utm +zone=15 +north');

   colors = list( colors_lut = c('#878787',   '#C6C73CFF', '#A0CA3AFF', '#7ACD37FF', '#53CF35FF', '#009900FF'),
                  breaks = c( -2.225074e-308,   0.000000e+00,   0.2, 0.4, 0.6, 0.8, 1.0),
                  labels = c( '0', 'up to 0.2', '0.2 to 0.4', '0.4 to 0.6', '0.6 to 0.8', '0.8 - 1') );
```

```
layer=c( 'BAREGRND',
         'MANGR',
         'PASP',
         'SWAMP',
         'SCRUB',
         'WAXM',
         'SPLAY',
         'CUTGR',
         'MAID',
         'THIN',
         'CAT',
         'SAWG',
         'BULL',
         'ROSEAU',
         'WHIP',
         'WIRE',
         'BRACK',
         'NEEDL',
         'SALT',
         'OYST',
         'WATER',
         'SAV',
         'NOTMOD');

vs = read.asciigridplus(i);

pdf(file=o, height=7.5, width=11, paper='USr', pagecentre=TRUE, onefile=TRUE);
for ( l in layer ) {
   par(mar=c(5,5,3,15));
   plot.asciigridplus(vs,colors_lut = colors, layer=l, main=l);
   plot.shape(LA,add=TRUE);
   legend(905000,3400000,
      legend=colors$labels,
      fill=colors$colors_lut,
```

```
        xpd=NA
        );
    }
    dev.off();
}


get_shapefiles = function()
{
    err = 0;
    err = ftp_shapefiles();

    if ( err ) {
        for ( error in attr(err, 'errList') ) {
            cat(error,'\n');
        }
    }

    return(err);
}

get_ecohydrofiles = function()
{
    err = 0;
    err = ftp_ecohydrofiles();
    if ( err ) {
        for ( error in attr(err, 'errList') ) {
            cat(error,'\n');
        }
    }
    return(err);
}

#get_stichedfiles = function()
```

```
#{
   #err = 0;
   #if ( Year == 10 ) {
      #t = templateLandwaterIMG;
      #t = sub('__YEAR__', Year, t);
      #cmd = paste('ln -s ',filenameLandwater,' ',t,sep='');
      #system(cmd, intern=TRUE);
#
      #t = templateElevationIMG;
      #t = sub('__YEAR__', Year, t);
      #cmd = paste('ln -s ',filenameElevation,' ',t,sep='');
      #system(cmd, intern=TRUE);
#
   #} else {
      #ftp_err = ftp_stitchedfiles();
      #if ( ftp_err ) {
         #for ( error in attr(ftp_err, 'errList') ) {
            #cat('main(): Error: Could not download ', error,' from the ftp site\n');
            #err = err + 1;
         #}
      #}
   #}
   #return(err);
#}

main = function()
{
   cat('Scenario   = ',Scenario   , '\n');
   cat('Group      = ',Group      , '\n');
   cat('Variation  = ',Variation  , '\n');
   cat('Year       = ',Year       , '\n');
   cat('Realization= ',Realization, '\n');
   cat('regionList = ',regionList , '\n');
```

```
steps = list();
if ( file.exists(filenameSteps) ) {
   steps = read.steps(filenameSteps);
} else {
   steps = list( Step1_GetShapeFiles=0,
                 Step2_GetEcoHydroFiles=0,
                 Step3_GetStitchedFiles=0,
                 Step4_CreateClearGrid=0,
                 Step5_CreateBoxID=0,
                 Step6_CreateElevation=0,
                 Step7_CreateLandwater=0,
                 Step8_CreateInitialConditions=0,
                 Step9_CreateConfig=0,
                 Step10_RunModel=0,
                 Step11_CreateWholeCoastline=0,
                 Step12_CreatePDF=0);
   write.steps(steps, filenameSteps);
}

err = 0;
# Step 0: Get shapefils
cat('Getting shapefiles setup\n');
if ( !steps$Step1_GetShapeFiles ) {
   err = err + get_shapefiles();
   if ( err ) {
      return(err);
   }

   steps$Step1_GetShapeFiles = 1;
   write.steps(steps, filenameSteps);
} else {
   cat('Already done, skipping\n');
}
```

```
# Step 0.5: Get ecohydrofiles()
cat('Getting ecohydro files setup\n');
if ( !steps$Step2_GetEcoHydroFiles ) {
   err = err + get_ecohydrofiles();
   if ( err ) {
      return(err);
   }

   steps$Step2_GetEcoHydroFiles = 1;
   write.steps(steps, filenameSteps);
} else {
   cat('Already done, skipping\n');
}

# Step 1: Get stitched files
cat('Getting elevation and landwater files setup\n');
if ( !steps$Step3_GetStitchedFiles ) {
   ret = ftp_stitchedfiles();
   #if ( ret ) {
      #for ( msg in attr(ret, 'errList') ) {
         #cat(msg,'\n');
      #}
      #err = err + ret;
      #return(ret);
   #}

   steps$Step3_GetStitchedFiles = 1;
   write.steps(steps, filenameSteps);
}  else {
   cat('Already done, skipping\n');
}

# Step 3: Create CLEAR Grid files
```

```
cat('Getting clear grid setup\n');
if ( !steps$Step4_CreateClearGrid ) {
   create_cleargrid();
   steps$Step4_CreateClearGrid = 1;
   write.steps(steps, filenameSteps);
} else {
   cat('Already done, skipping\n');
}

# Step 4: Create BoxID files
cat('Getting BoxID setup\n');
if ( !steps$Step5_CreateBoxID ) {
   create_boxid();
   steps$Step5_CreateBoxID = 1;
   write.steps(steps, filenameSteps);
} else {
   cat('Already done, skipping\n');
}

# Step 5: Convert wetlandmorph files.
# create_wetlandmorphfilesCSV();
cat('Getting elevation data setup\n');
if ( !steps$Step6_CreateElevation ) {
   create_elevationfileASC();
   steps$Step6_CreateElevation = 1;
   write.steps(steps, filenameSteps);
} else {
   cat('Already done, skipping\n');
}

cat('Getting landwater data setup\n');
if (!steps$Step7_CreateLandwater ) {
   create_landwaterfileASC();
   steps$Step7_CreateLandwater = 1;
```

```
    write.steps(steps, filenameSteps);
} else {
    cat('Already done, skipping\n');
}

# Step 5: Create Initial conditions files
cat('Getting initial conditions setup\n');
if ( !steps$Step8_CreateInitialConditions ) {
    create_initialconditions();
    steps$Step8_CreateInitialConditions = 1;
    write.steps(steps, filenameSteps);
} else {
    cat('Already done, skipping\n');
}

# Step 6: Run the models
cat('Getting configuration files setup\n');
if ( !steps$Step9_CreateConfig ) {
    create_configfile();
    steps$Step9_CreateConfig = 1;
    write.steps(steps, filenameSteps);
} else {
    cat('Already done, skipping\n');
}

cat('Running the model\n');
if ( !steps$Step10_RunModel ) {
    err = 0;
    err = run_model();
    if ( !err ) {
        steps$Step10_RunModel = 1;
        write.steps(steps, filenameSteps);
    } else {
        steps$Step10_RunModel = 0;
```

```
        write.steps(steps, filenameSteps);
        return(err);
      }
    } else {
      cat('Already done, skipping\n');
    }

    # Step 7: Merge results.
    cat('Mergin data\n');
    if ( !steps$Step11_CreateWholeCoastline ) {
      create_wholecoastlinefiles(Year + c(5, 10, 15, 20, 25));
      steps$Step11_CreateWholeCoastline = 1;
      write.steps(steps, filenameSteps);
    } else {
      cat('Already done, skipping\n');
    }

    # Step 8: Visualize the output
    cat('Creating PDF files\n');
    if ( !steps$Step12_CreatePDF ) {
      create_PDF(make.filename(Reg = 'DELAY', Y = 'DELAY', Real = VegRealization, temp = templateVG ),'CP');
      create_PDF(make.filename(Reg = 'DELAY', Y = 'DELAY', Real = VegRealization, temp = templateVG ),'AA');
      create_PDF(make.filename(Reg = 'DELAY', Y = 'DELAY', Real = VegRealization, temp = templateVG ),'PB');
      create_PDF(make.filename(Reg = 'DELAY', Y = 'DELAY', Real = VegRealization, temp = templateVEG),'LA');
      steps$Step12_CreatePDF = 1;
      write.steps(steps, filenameSteps);
    } else {
      cat('Already done, skipping\n');
    }
}

main()
#quit(save='no')
```

VIII. Source code for myGISlib.R

```
#######################################################
# Load 3rd party libraries
#
#######################################################
require("shapefiles");
require("sp");
#require("spatstat");
require("rgdal");

#######################################################
# Some general utility tools
#
#######################################################
defactorize = function(data)
{
    if ( class(data) == "factor") {
        return(levels(data)[data]);
    } else {
        return(data);
    }
}


#######################################################
# Define functions to create a raster data set
# after the style of the ascigrid format
# used below to read grids
#######################################################
create.asciigrid=function(ncols, nrows, xllcorner, yllcorner, cellsize, nodata_value, init_value = 0)
{
    if ( ncols == Inf || ncols == -Inf ) {
        cat("create.asciigrid(...): Error: ncols == Inf or -Inf, returning empty grid\n");
```

```
    return( create.asciigrid(0,0,0,0,1,0,0) );
}
if ( is.na(ncols) ) {
    cat("create.asciigrid(...): Error: ncols == NA, returning empty grid\n");
    return( create.asciigrid(0,0,0,0,1,0,0) );
}

if ( nrows == Inf || nrows == -Inf ) {
    cat("create.asciigrid(...): Error: nrows == Inf or -Inf, returning empty grid\n");
    return( create.asciigrid(0,0,0,0,1,0,0) );
}
if ( is.na(nrows) ) {
    cat("create.asciigrid(...): Error: nrows == NA, returning empty grid\n");
    return( create.asciigrid(0,0,0,0,1,0,0) );
}

if ( xllcorner == Inf || xllcorner == -Inf ) {
    cat("create.asciigrid(...): Error: xllcorner == Inf or -Inf, returning empty grid\n");
    aeturn( create.asciigrid(0,0,0,0,1,0,0) );
}
if ( is.na(xllcorner) ) {
    cat("create.asciigrid(...): Error: xllcorner == NA, returning empty grid\n");
    return( create.asciigrid(0,0,0,0,1,0,0) );
}

if ( yllcorner == Inf || yllcorner == -Inf ) {
    cat("create.asciigrid(...): Error: yllcorner == Inf or -Inf, returning empty grid\n");
    return( create.asciigrid(0,0,0,0,1,0,0) );
}
if ( is.na(yllcorner) ) {
    cat("create.asciigrid(...): Error: yllcorner == NA, returning empty grid\n");
    return( create.asciigrid(0,0,0,0,1,0,0) );
}
```

```
    if ( cellsize == 0 ) {
        cat("create.asciigrid(...): Error: cellsize == 0, returning empty grid\n");
        return( create.asciigrid(0,0,0,0,1,0,0) );
    }
    if ( cellsize == Inf || cellsize == -Inf ) {
        cat("create.asciigrid(...): Error: cellsize == Inf or -Inf, returning empty grid\n");
        return( create.asciigrid(0,0,0,0,1,0,0) );
    }
    if ( is.na(cellsize) ) {
        cat("create.asciigrid(...): Error: cellsize == NA, returning empty grid\n");
        return( create.asciigrid(0,0,0,0,1,0,0) );
    }

    header = list(nrows=nrows, ncols=ncols, xllcorner=xllcorner, yllcorner=yllcorner, cellsize=cellsize, NODATA_value=nodata_value);
    data   = matrix(init_value,nrow=nrows, ncol=ncols);
    attr(data,"header") = header;
    return(data);
}

create.asciigrid.fromshape=function(shape, cellsize)
{
    box = bbox.shapefile(shape);

    xllcorner = box$xmin;
    yllcorner = box$ymin;

    nrows = ceiling( box$ymax - box$ymin)/cellsize;
    ncols = ceiling( box$xmax - box$xmin)/cellsize;

    ret = create.asciigrid(ncols, nrows, xllcorner, yllcorner, cellsize, -9999, -9999 );

    index = 1:(nrows*ncols) - 1;

    r = floor(index/ncols) + 1;
```

```
    c = index%%ncols + 1;

    xcent = xllcorner + cellsize*(c-0.5);
    ycent = yllcorner + cellsize*(r-0.5);

    index_points = point.in.shapefile(shape, coords=data.frame(x=xcent,y=ycent) );

    index = matrix(   data.frame(r=r[index_points], c=c[index_points])   );

    ret[index] = 0;
    return(ret);
}

copy.asciigrid = function(grid)
{
    ret = grid;
    header = attr(grid,"header");
    attr(ret,"header") = header;
    return(ret);
}

#######################################################
# Define functions to read/write EcoHydro data files
#
#######################################################
read.ecohydro = function(eh_file, day=1)
{
    eh_con = file(eh_file, open="r");

    if ( !isOpen(eh_con) ) {
        cat("read.ecohydro(...): Error : could not open file for reading\n");
        return(1);
    }
```

```
header_line = readLines(eh_con, n = 1);


offset = gregexpr("PB[0-9]+",header_line)[[1]];
header = rep("",length(offset));
for ( o in offset ) {
    header[offset == o] = substr(header_line,o,o+attr(offset, "match.length")[offset == o]-1);
}

ret = data.frame( BoxID=header);

n_skip = min(day) - 1;
if ( n_skip != 0 ) {
    lines = readLines(eh_con, n=n_skip, ok=TRUE);
    if ( length(lines)  != n_skip ) {
      cat("read.ecohydro(...): Error: day = ",day," larger than the number of lines in the data file = ",length(lines),"\n");
      return(1);
    }
}

for ( d in day ) {
    values_line = readLines(eh_con, n = 1);
    if ( length(values_line) != 1 ) {
      cat("read.ecohydro(...): Error: could not read data for day = ",day,". Not enough data\n");
      return(1);
    }


    offset = gregexpr("[0-9]+\\.[0-9]+",values_line)[[1]];
    values = rep(0,length(offset));
    for ( o in offset ) {
      values[offset == o] = substr(values_line,o,o+attr(offset, "match.length")[offset == o]-1);
    }
```

```
        values = as.numeric(values);
        ret[paste("EcoHydroDay",d,sep="")] = signif(values,digits=6);
    }
    close(eh_con);

    return(ret);
}

#######################################################
# Define functions to read/write raster data.
#
#######################################################
read.gridfloat = function(filename, rescale = 1)
{
    cat("reading ",filename,"\n");
    filename_hdr = paste(filename,".hdr",sep="");
    filename_flt = paste(filename,".flt",sep="");

    re_int         = "(-?[0-9]+)";
    re_float       = "(-?[0-9]+\\.[0-9]+)";
    re_ncols       = "^ncols";
    re_nrows       = "^nrows";
    re_xllcorner   = "^xllcorner";
    re_yllcorner   = "^yllcorner";
    re_cellsize    = "^cellsize";
    re_NODATA_value = "^NODATA_value";
    re_byteorder   = "^byteorder";

    ncols       = 0;
    nrows       = 0;
    xllcorner   = 0;
    yllcorner   = 0;
    cellsize    = 0;
    NODATA_value = 0;
```

```
byteorder    = "unknown";

fcon = file(filename_hdr,"r");
line = readLines(fcon,n=1);
regexpr(re_ncols,line);
while( length(line) != 0 ) {

    if ( length(grep(re_ncols, line)) != 0 ) {
      start = regexpr(re_int,line);
      stop = start + attr(start,"match.length");
      ncols = as.integer(substr(line,start,stop));
    }

    if ( length(grep(re_nrows, line)) != 0 ) {
      start = regexpr(re_int,line);
      stop = start + attr(start,"match.length");
      nrows = as.integer(substr(line,start,stop));
    }

    if ( length(grep(re_xllcorner, line)) != 0 ) {
      start = regexpr(re_float,line);
      stop = start + attr(start,"match.length");
      xllcorner = as.numeric(substr(line,start,stop));
    }

    if ( length(grep(re_yllcorner, line)) != 0 ) {
      start = regexpr(re_float,line);
      stop = start + attr(start,"match.length");
      yllcorner = as.numeric(substr(line,start,stop));
    }

    if ( length(grep(re_cellsize, line)) != 0 ) {
      start = regexpr(re_float,line);
      stop = start + attr(start,"match.length");
```

```
      cellsize = as.numeric(substr(line,start,stop));
    }

    if ( length(grep(re_NODATA_value, line)) != 0 ) {
      start = regexpr(re_int,line);
      stop = start + attr(start,"match.length");
      NODATA_value = as.integer(substr(line,start,stop));
    }

    if( length(grep(re_byteorder,line)) != 0 ) {
      line = sub(re_byteorder,"",line);
      byteorder = gsub(" ","",line);
    }


    line = readLines(fcon,n=1);
}
close(fcon);


# Read the data into the matrix
fcon = file(filename_flt,"rb");
values=readBin(fcon,"numeric",nrows*ncols,size=4,endian="little");
close(fcon);

data = matrix(values,nrow=nrows,ncol=ncols,byrow=TRUE);
data = data[nrows:1,1:ncols];

header = list(nrows=nrows,ncols=ncols,xllcorner=xllcorner,llcorner=yllcorner,cellsize=cellsize,NODATA_value=NODATA_value,byteorder=byteorder);
attr(data,"header") = header;
remove(values);
gc();

if ( rescale != 1 ) {
```

```
        temp = rescale.gridfloat(data, ncols_new = as.integer(ncol(data)/rescale), nrows_new = as.integer(nrow(data)/rescale) );
        remove(data);
        gc();
        data=temp;
        return(data);
    } else {
        return(data);
    }

}

write.gridfloat = function(grid, filename, path=".")
{
    dir.create(paste(path,"/",filename,sep=""),showWarnings=TRUE);
    filename_flt = paste(path,"/",filename,"/",filename, ".flt", sep="");
    filename_hdr = paste(path,"/",filename,"/",filename, ".hdr", sep="");

    header = attr(grid,"header");

    cat("ncols ",       header$ncols,       "\n", file=filename_hdr, sep="");
    cat("nrows ",       header$nrows,       "\n", file=filename_hdr, sep="", append=TRUE);
    cat("xllcorner ",    header$xllcorner,    "\n", file=filename_hdr, sep="", append=TRUE);
    cat("yllcorner ",    header$yllcorner,    "\n", file=filename_hdr, sep="", append=TRUE);
    cat("cellsize ",     header$cellsize,     "\n", file=filename_hdr, sep="", append=TRUE);
    cat("NODATA_value ", header$NODATA_value, "\n", file=filename_hdr, sep="", append=TRUE);
    cat("byteorder ",    header$byteorder,    "\n", file=filename_hdr, sep="", append=TRUE);

    fcon = file(filename_flt, "wb");
    if ( is.na(header$NODATA_value) ) {
        value_min = min(grid, na.rm=TRUE);
        NODATA_value = value_min - 1;
    } else {
        NODATA_value = header$NODATA_value;
    }
```

```
    grid[is.na(grid)] = NODATA_value;
    grid = grid[header$nrows:1,1:header$ncols];
    writeBin(as.vector(grid), fcon);
    close(fcon);
}

write.asciigrid = function(grid, filename)
{
   base_type = class( grid[1,1] );

    header = attr(grid,"header");

    if ( is.na(header$NODATA_value) ) {
         value_min = min(grid, na.rm=TRUE);
         if ( value_min < -9999 ) { NODATA_value = floor(value_min - 1); } else { NODATA_value = -9999;}
    } else {
         NODATA_value = header$NODATA_value;
    }

   if ( base_type == "integer" ) {
      NODATA_value = as.integer(NODATA_value);
   }

   grid[is.na(grid)] = NODATA_value;

    cat("ncols ",       header$ncols,       "\n", file=filename, sep="");
    cat("nrows ",       header$nrows,       "\n", file=filename, sep="", append=TRUE);
    cat("xllcorner ",    header$xllcorner,    "\n", file=filename, sep="", append=TRUE);
    cat("yllcorner ",    header$yllcorner,    "\n", file=filename, sep="", append=TRUE);
    cat("cellsize ",     header$cellsize,     "\n", file=filename, sep="", append=TRUE);
    cat("NODATA_value ", NODATA_value,       "\n", file=filename, sep="", append=TRUE);

    grid = grid[header$nrows:1,1:header$ncols];
```

```
    write.table(grid, file=filename, append=TRUE, col.names=FALSE, row.names=FALSE, sep=" ");
}

read.asciigrid = function(filename="", fcon=-1, rescale = 1)
{

    re_int         = "(-?[0-9]+)";
    re_float       = "(-?[0-9]+(\\.[0-9]+)?([eE][+-]?[0-9]+)?)";
    # re_float       = "(-?[0-9]+(\\.[0-9]+)?([eE][+-]?[0-9]+)?)";
    re_ncols       = "^NCOLS";
    re_nrows       = "^NROWS";
    re_xllcorner   = "^XLLCORNER";
    re_yllcorner   = "^YLLCORNER";
    re_cellsize    = "^CELLSIZE";
    re_NODATA_value = "^NODATA_VALUE";

    ncols       = NA;
    nrows       = NA;
    xllcorner   = NA;
    yllcorner   = NA;
    cellsize    = NA;
    NODATA_value = NA;

   if ( filename !=  "" )  {
       cat("reading ",filename,"\n");
       fcon = file(filename,"r");
   }
    #while(  (is.na(ncols) ||
           #is.na(nrows) ||
           #is.na(xllcorner) ||
           #is.na(yllcorner) ||
           #is.na(cellsize) ||
           #is.na(NODATA_value)) && length(line) != 0 )
    for ( i in 1:6 ) {
```

```
line = ""
while ( nchar(line) == 0 ) {
  line = readLines(fcon,n=1);
  line = sub("#.*","",line);
}

line = toupper(line);


if ( length(grep(re_ncols, line)) != 0 ) {
  start = regexpr(re_int,line);
  stop = start + attr(start,"match.length");
  ncols = as.integer(substr(line,start,stop));
}

if ( length(grep(re_nrows, line)) != 0 ) {
  start = regexpr(re_int,line);
  stop = start + attr(start,"match.length");
  nrows = as.integer(substr(line,start,stop));
}

if ( length(grep(re_xllcorner, line)) != 0 ) {
  start = regexpr(re_float,line);
  stop = start + attr(start,"match.length");
  xllcorner = as.numeric(substr(line,start,stop));
}

if ( length(grep(re_yllcorner, line)) != 0 ) {
  start = regexpr(re_float,line);
  stop = start + attr(start,"match.length");
  yllcorner = as.numeric(substr(line,start,stop));
}

if ( length(grep(re_cellsize, line)) != 0 ) {
```

```
        start = regexpr(re_float,line);
        stop = start + attr(start,"match.length");
        cellsize = as.numeric(substr(line,start,stop));
      }

      if ( length(grep(re_NODATA_value, line)) != 0 ) {
        start = regexpr(re_int,line);
        stop = start + attr(start,"match.length");
        NODATA_value = as.integer(substr(line,start,stop));
      }

  }

  values = scan(fcon, what=numeric(0), nmax=nrows*ncols, sep=" ");
if ( filename != "" ) {
     close(fcon);
}

  data = matrix(values[!is.na(values)],nrow=nrows,ncol=ncols,byrow=TRUE);
  data = data[nrows:1,1:ncols];
  data[data == NODATA_value] = NA;

  header = list(nrows=nrows,ncols=ncols,xllcorner=xllcorner,yllcorner=yllcorner,cellsize=cellsize,NODATA_value=NODATA_value);
  attr(data,"header") = header;
  remove(values);
  gc();

  if ( rescale != 1 ) {
      temp = rescale.gridfloat(data, ncols_new = as.integer(ncol(data)/rescale), nrows_new = as.integer(nrow(data)/rescale) );
      remove(data);
      gc();
      data=temp;
      return(data);
  } else {
```

```
        return(data);
    }

}


get.offset = function(filename, string)
{

   # Step 1: Get the offsets for the requested string
   cmd = paste("grep -b \"",string,"\" ",filename,sep="");
   ret = system(cmd, intern=TRUE);

   rret = sub("^[0-9]*:","", ret);
   ret = ret[ rret%in%string ];

   offset = ret[1];
   offset = sub(paste(":",string,sep=""), "", offset);
   offset = as.integer(offset);
   return(offset);
}

read.asciigrid.year = function(filename, year)
{
   # Step 1: Get the offset to the particular year.
   offset = get.offset(filename, paste("# Year = ",year,sep=""));

   # Step 2: Open the connection and move to the offset
   con    = file(filename, open="r");
   seek(con, offset);

   # Step 3: Read off the year marker.
   line = readLines(con, n=1); # read off the year marker.
```

```
  # Step 4: Read off the asciigrid data.
  grid = read.asciigrid(fcon = con);

  # Step 5: Close the connection and return the grid
  close(con);
  return(grid);
}

read.asciigridplus.year = function(filename, year)
{
  # Step 1: Get the offsets for the start of the requested year.
  cmd = paste("grep -b \"Year = \" ",filename,sep="");
  ret = system(cmd, intern=TRUE);

  numYears = length(ret);

  offset = ret[year];
  offset = sub(paste(":# Year = ",year,sep=""), "", offset);
  offset = as.integer(offset);

  # Step 2: Get the line number for the requeste year.
  cmd = paste("grep -n \"Year = \" ",filename,sep="");
  retLnum = system(cmd, intern=TRUE);

  s = retLnum[year];
  s = sub(paste(":# Year = ",year,sep=""), "", s);
  s = as.integer(s);

  m = 0;
  # Step 2.1: Find out where to stop reading data for this year.
  if ( year == numYears ) {
    cmd = paste('wc -l ',filename, ' | awk \'{ print $1 }\'', sep='');
    retLength = system(cmd, intern=TRUE);
    m = retLength[1];
```

```
    m = as.integer(m);
  } else {
    m = retLnum[year+1];
    m = sub(paste(":# Year = ",as.integer(year)+1, sep=""), "", m);
    m = as.integer(m);
  }


  # Step 3: Open the file
  con = file(filename, open="r");

  #Step 4: read off the grid data.
  seek(con, offset ); # Move the file pointer to the start of the year data.
  line = readLines(con, n=1); # read off the year marker.
  grid = read.asciigrid(fcon = con); # Read off the grid data.

  # Step 5: read off the table data.
  n = 6 + nrow(grid) + s + 1;
  nlines = m - n - 1;
  table = read.csv(con,nrows=nlines); # read off the csv data.

  # Step 6: close the connection.
  close(con);

  # Step 7: Return the asciigridplus object.
  return( list(grid=grid, table=table));
}

read.asciigridplus = function(filename)
{
   grid = read.asciigrid(filename);

   table = read.csv(filename, skip=6+attr(grid,"header")$nrows);
```

```
    return(list(grid=grid, table=table));
}

write.asciigridplus = function(filename, grid, table)
{
    write.asciigrid(filename, grid);
    write.csv(filename, table, append=TRUE);
}

read.landscape = function(filename, rescale = 1, maskfile.asciigrid=NA)
{
    filename_meta = paste(filename,".meta",sep="");
    filename_bin  = paste(filename,".bin",sep="");

    re_int      = "(-?[0-9]+)";
    re_float    = "(-?[0-9]+\\.[0-9]+)";
    re_ncols    = "^NUM_COLS *: *: *";
    re_nrows    = "^NUM_ROWS *: *: *";
    re_xllcorner = "^WESTING *: *: *";
    re_yllcorner = "^SOUTHING *: *: *";
    re_cellsize  = "^CELLSIZE_WIDTH *: *: *";
    re_nlayers   = "^REPEAT_CHILD *:[[:alnum:][:blank:]]*: *";
    re_type      = "^FUND_TYPE *:[[:alnum:][:blank:]]*: *";
    re_nbytes    = "^NUM_BYTES *:[[:alnum:][:blank:]]*: *";

    ncols       = NA;
    nrows       = NA;
    xllcorner   = NA;
    yllcorner   = NA;
    cellsize    = NA;
    NODATA_value = NA;
    nlayers     = NA;
    type        = NA;
    nbytes      = NA;
```

```
byteorder    = NA;

fcon = file(filename_meta, "r");
line = readLines(fcon,n=1);
while ( length(line) != 0 ) {

    if ( length(grep(re_ncols,line))  != 0 ) {
      start = regexpr(re_int, line);
      stop  = start + attr(start,"match.length");
      ncols = as.integer(substr(line,start,stop));
    }

    if ( length(grep(re_nrows,line))  != 0 ) {
      start = regexpr(re_int, line);
      stop  = start + attr(start,"match.length");
      nrows = as.integer(substr(line,start,stop));
    }

    if ( length(grep(re_xllcorner,line))  != 0 ) {
      start = regexpr(re_float, line);
      stop  = start + attr(start,"match.length");
      xllcorner = as.numeric(substr(line,start,stop));
    }

    if ( length(grep(re_yllcorner,line))  != 0 ) {
      start = regexpr(re_float, line);
      stop  = start + attr(start,"match.length");
      yllcorner = as.numeric(substr(line,start,stop));
    }

    if ( length(grep(re_cellsize,line))  != 0 ) {
      start = regexpr(re_float, line);
      stop  = start + attr(start,"match.length");
      cellsize = as.numeric(substr(line,start,stop));
```

```
    }

    if ( length(grep(re_nlayers,line)) != 0 && is.na(nlayers) ) {
      start = regexpr(re_int, line);
      stop  = start + attr(start,"match.length");
      nlayers = as.integer(substr(line,start,stop));
    }

    if ( length(grep(re_type,line)) != 0 ) {
      start = regexpr(re_type, line);
      stop  = start + attr(start,"match.length");
      type  = substr(line,stop,nchar(line));
    }

    if ( length(grep(re_nbytes,line)) != 0 ) {
      start = regexpr(re_nbytes, line);
      stop  = start + attr(start,"match.length");
      nbytes  = as.integer(substr(line,stop,nchar(line)));
    }

    line = readLines(fcon,n=1);
}
close(fcon);


fcon = file(filename_bin, "rb");
if ( type == "int" ) {
    values = readBin(fcon, "integer", nrows*ncols*nlayers, size=nbytes, endian="big");
} else if ( type == "float" ) {
    values = readBin(fcon, "numeric", nrows*ncols*nlayers, size=nbytes, endian="big");
} else {
    cat("myGISlib.R : read.landscape(...) : Unknown fundemental type in file ",filename_meta,"\n");
    close(fcon);
    return(NA);
```

```
}
close(fcon);

data = array(values, dim=c(ncols,nrows,nlayers), dimnames=c("cols","rows","layers"));

for ( l in 1:nlayers ) { data[,,l] = data[1:ncols,nrows:1,l] }

if ( !is.na(maskfile.asciigrid) ) {
    mask = read.asciigrid(maskfile.asciigrid);
    header = attr(mask,"header");
    mask = t(mask);
    attr(mask,"header") = header;
    index = which( is.na(mask));
    for ( l in 1:nlayers ) {
      temp = data[,,l];
      temp[index] = NA;
      data[,,l] = temp;
      # data[index[,2],index[,1],l] = NA;
    }
}

data_t = array(0, dim=c(nrows,ncols,nlayers), dimnames=c("cols","rows","layers"));
for ( l in 1:nlayers ) { data_t[,,l] = t(data[,,l]); };
data = data_t;

header =
      list(nrows=nrows,ncols=ncols,xllcorner=xllcorner,yllcorner=yllcorner,cellsize=cellsize,NODATA_value=NODATA_value,byteorder=byteorder,nlaye
      rs=nlayers);
attr(data,"header") = header;
remove(values);
gc();


return(data);
```

```
}

read.erdas = function(filename,band='')
{
   data = readGDAL(filename);

   header = list(nrows = data@grid@cells.dim[2],
            ncols = data@grid@cells.dim[1],
            xllcorner= data@bbox['x','min'],
            yllcorner= data@bbox['y','min'],
            cellsize= data@grid@cellsize[1],
            NODATA_value=-9999);

   if ( band == '' ) {
      band = names(data@data)[1];
   }

   ret = matrix( data@data[[band]], nrow=header$nrows, ncol=header$ncols, byrow=TRUE);
   ret = ret[nrow(ret):1,1:ncol(ret)];
   attr(ret, 'header') = header;
   return(ret);

}
#######################################################
# Define functions to manipulate raster data
#
#######################################################
summary.gridfloat = function(data,show_header=TRUE,show_only_header=FALSE)
{
    if ( show_header == TRUE ) {
        cat("( dim )  nrows  ncols  xllcorner  yllcorner  xurcorner  yurcorner  cellsize  NODATA_value  LSBFIRST\n");
    }

    if ( show_only_header == FALSE ) {
```

```
        if ( class(data) == "list" ) {
          bbox = bbox.gridfloat(data);
          for ( elt in data ) { summary.gridfloat(elt$data, show_header=FALSE); }

        } else if ( class(data) == "matrix" ) {
          a    = attributes(data)$header;
          bbox = bbox.gridfloat(data);

          cat("(",dim(data),")", a$nrows, a$ncols, a$xllcorner, a$yllcorner, bbox$xurcorner, bbox$yurcorner, a$cellsize, a$NODATA_value,
          a$LSBFIRST,"\n",sep="  ");

        }
    }
}

bbox.gridfloat = function(data)
{
    if ( class(data) == "list" ) {
        bbox = bbox.gridfloat(data[[1]]$data);
        xll = bbox$xllcorner;
        yll = bbox$yllcorner;
        xur = bbox$xurcorner;
        yur = bbox$yurcorner;
        for ( elt in data ) {
          bbox = bbox.gridfloat(elt$data);
          xll  = min(c(xll,bbox$xllcorner));
          yll  = min(c(yll,bbox$yllcorner));
          xur  = max(c(xur,bbox$xurcorner));
          yur  = max(c(yur,bbox$yurcorner));
        }
        return( list(xllcorner=xll,yllcorner=yll,xurcorner=xur,yurcorner=yur) );
    } else if ( class(data) == "matrix" ) {
        ncols     = attr(data,"header")$ncols;
```

```
        nrows     = attr(data,"header")$nrows;
        xllcorner = attr(data,"header")$xllcorner;
        yllcorner = attr(data,"header")$yllcorner;
        cellsize  = attr(data,"header")$cellsize;

        xurcorner = xllcorner + cellsize*ncols;
        yurcorner = yllcorner + cellsize*nrows;

        return( list(xllcorner=xllcorner,yllcorner=yllcorner,xurcorner=xurcorner,yurcorner=yurcorner, xmin=xllcorner, xmax=xurcorner, ymin=yllcorner,
          ymax=yurcorner));
    }
}

bbox.grid = function(grid)
{
    return(bbox.gridfloat(grid));
}

bbox.asciigrid = function(grid)
{
   return(bbox.gridfloat(grid));
}

rescale.gridfloat = function(data, ncols_new=10, nrows_new=10)
{
    ncols_sample = unique(as.integer(seq(  1,ncol(data),length.out=ncols_new  )));
    nrows_sample = unique(as.integer(seq(  1,nrow(data),length.out=nrows_new  )));

    data_new = data[nrows_sample,ncols_sample];

    header = attr(data,"header");
    header$cellsize = header$cellsize * header$nrows/nrows_new;
    header$nrows = nrows_new;
    header$ncols = ncols_new;
```

```
    attr(data_new,"header") = header;
    return(data_new);
}

allign.gridfloat = function(data)
{
    # Step 1: We need to establish the common registration
    # Step 2: We need to find a common cellsize.

    # Step 3: Now adjust the xllcorner and yllcorner and cellsize of all the grids.

    bbox          = bbox.gridfloat(data);
    base_xllcorner = bbox$xllcorner;
    base_yllcorner = bbox$yllcorner;
        cs = 1:length(data) * 0; for ( i in 1:length(data) ) { cs[i] = attr( data[[i]]$data, "header")$cellsize; }
    base_cellsize  = mean(cs);

    for ( i in 1:length(data) ) {
        attr(data[[i]]$data, "header")$cellsize = base_cellsize;

        xllcorner = attr(data[[i]]$data, "header")$xllcorner;
        delta = base_cellsize * ((xllcorner - base_xllcorner)/base_cellsize - floor((xllcorner - base_xllcorner)/base_cellsize));
        attr(data[[i]]$data, "header")$xllcorner = xllcorner - delta;
        #cat("delta_x, delta_y = ",delta,sep="");

        yllcorner = attr(data[[i]]$data, "header")$yllcorner;
        delta = base_cellsize * ((yllcorner - base_yllcorner)/base_cellsize - floor((yllcorner - base_yllcorner)/base_cellsize));
        attr(data[[i]]$data, "header")$yllcorner = yllcorner  - delta;
        #cat(", ",delta,"\n",sep="");

    }

    ncol = (bbox$xurcorner - bbox$xllcorner)/base_cellsize;
    celledge_x = bbox$xllcorner + (0:ncol)*base_cellsize;
```

```
    nrow = (bbox$yurcorner - bbox$yllcorner)/base_cellsize;
    celledge_y = bbox$yllcorner + (0:nrow)*base_cellsize;

    range_x = range(celledge_x);
    range_y = range(celledge_y);
    plot(range_x,range_y,type="n");

    for ( x in celledge_x ) { lines(c(x,x),range_y);}
    for ( y in celledge_y ) { lines(range_x,c(y,y));}

    color_lut = rainbow(length(data));
    for ( i in 1:length(data) ){
        bbox = bbox.gridfloat(data[[i]]$data);
        rect(bbox$xllcorner, bbox$yllcorner, bbox$xurcorner, bbox$yurcorner, border=color_lut[i]);
    }

    invisible(data);

}

merge.gridfloat = function(data_list)
{
    # This function merges a number of individual gridfloat grids into
    # a single gridfloat grid. The function assumes that the cell sizes
    # for all the input grids are similar.
    # This function works best if the grids have first been aligned using
    # allign.gridfloat.


    # Step 1 : Get the bounding box that encloses all the  individual grids.
    bbox_tgt = bbox.gridfloat(data_list);

    # Step 2 : Make the header information for the new merged grid.
```

```
    header = attr(data_list[[1]]$data,"header");
    header$xllcorner    = bbox_tgt$xllcorner;
    header$yllcorner    = bbox_tgt$yllcorner;
    header$ncols        = round( (bbox_tgt$xurcorner - bbox_tgt$xllcorner)/header$cellsize );
    header$nrows        = round( (bbox_tgt$yurcorner - bbox_tgt$yllcorner)/header$cellsize );
    #header$cellsize     =
    #header$NODATA_value =
    #header$byteorder    =

    # Step 3: Allocation memory for the new grid.
    ret = matrix(header$NODATA_value, nrow = header$nrows, ncol = header$ncols );

    # attached the header information to the new grid.
    attr(ret,"header") = header;

    bbox_tgt     = bbox.gridfloat(ret);
    cellsize_tgt = header$cellsize;

#cat("Merged grid summary\n");
#summary.gridfloat(ret);
#cat("\n\n");

    # Step 4: Cycle over the individual grids and place the data from each into the
    # new grid.
    for ( src in data_list ) {
        # Step 4a: Find the bounding box for the current grid.
        bbox_src = bbox.gridfloat(src$data);
        cellsize_src = attr(src$data,"header")$cellsize;

        # Step 4b:
        rowrange_src = 1:nrow(src$data); nrow_src = length(rowrange_src);
        colrange_src = 1:ncol(src$data); ncol_src = length(colrange_src);

        rowstart_tgt = max(c(1, floor( (bbox_src$yllcorner - bbox_tgt$yllcorner)/cellsize_tgt ) + 1) );
```

```
colstart_tgt = max(c(1, floor( (bbox_src$xllcorner - bbox_tgt$xllcorner)/cellsize_tgt ) + 1) );

rowstop_tgt  = min(c(nrow(ret), rowstart_tgt + nrow_src - 1) );
colstop_tgt  = min(c(ncol(ret), colstart_tgt + ncol_src - 1) );

rowrange_tgt = rowstart_tgt:rowstop_tgt; nrow_tgt = length(rowrange_tgt);
colrange_tgt = colstart_tgt:colstop_tgt; ncol_tgt = length(colrange_tgt);

# Step 4c: Make sure everything is sane, and take corrective action if not.
# The number of rows and columns in the source should match the numbers
# in the target.
# In each direction, there are three things that can go wrong.
# 1) rowstart_tgt is less than 1.
# 2) rowstop_tgt is greater than nrow(ret)
# 3) the number of rows don't match.
# The first problem should not occur at this point in the algorithm


if ( (length(rowrange_src) != length(rowrange_tgt)) || (length(colrange_src) != length(colrange_tgt)) ) {
  cat("Error : matrix size miss match\n");
  cat("\t");

  cat("target[ ",sep="");
  cat(rowstart_tgt,":",rowstop_tgt,"(",length(rowrange_tgt),")","{",nrow(ret),"}",sep="");
  cat(", ",sep="");
  cat(colstart_tgt,":",colstop_tgt,"(",length(colrange_tgt),")","{",ncol(ret),"}",sep="");
  cat("] <-",sep="");

  cat("source[ ",sep="");
  cat(min(rowrange_src),":",max(rowrange_src),"(",length(rowrange_src),")","{",nrow(src$data),"}",sep="");
  cat(", ",sep="");
  cat(min(colrange_src),":",max(colrange_src),"(",length(colrange_src),")","{",ncol(src$data),"}",sep="");
  cat("] \n",sep="");
```

```
        cat("tgt :                    ",bbox_tgt$yurcorner,"\n");
        cat("tgt : ",bbox_tgt$xllcorner,"                              ",bbox_tgt$xurcorner,"\n");
        cat("tgt:                     ",bbox_tgt$yllcorner,"\n");
        cat("tgt: cellsize = ",cellsize_tgt,"\n");
        cat("\n\n");

        cat("src :                    ",bbox_src$yurcorner,"\n");
        cat("src : ",bbox_src$xllcorner,"                              ",bbox_src$xurcorner,"\n");
        cat("src :                    ",bbox_src$yllcorner,"\n");
        cat("src: cellsize = ",cellsize_src,"\n");

      } else {
        ret[rowrange_tgt,colrange_tgt] = src$data;
      }
    }

    #Step 5: Return the data.
    invisible(ret);
}

geo2rc.gridfloat = function(data, x, y)
{
    bbox = bbox.gridfloat(data);

    xll = bbox[[1]];
    yll = bbox[[2]];
    xur = bbox[[3]];
    yur = bbox[[4]];

    cellsize = attr(data, "header")$cellsize;

    # This function determines the row and column
    # within the grid in data associated with the
    # point x,y. The x and y diminsions of each
```

```
    # cell are intervals which are considered
    # to be open at the lower end of the range
    # and closed at the upper end. That is,
    # if in the x directions a grid cell starts
    # at pos x1 and ends at x2, the all
    # x values such that x1 < x <= x2 is true
    # are considered in the cell. If x = x1,
    # the x is in the next cell to the left.
    # The exception to this rule are cells
    # at the southern most or western most edge
    # of a grid. These grid cells are closed
    # on all sizes. That is for a grid cell on
    # the edge all points within the cell
    # satisfy x1 <= x <= x2.

    row = as.integer((y-yll)/cellsize);
    col = as.integer((x-xll)/cellsize);

    row = max(c(  1,  min(c(row, attr(data, "header")$nrows)) ));
    col = max(c(  1,  min(c(col, attr(data, "header")$ncols)) ));
    return(list(row=row,col=col));
}

value.gridfloat = function(data, x, y)
{
    rc = geo2rc.gridfloat(data,x,y);
    nrows = attr(data,"header")$nrow;
    ncols = attr(data,"header")$ncol;

    if ( 1 <= rc$row && rc$row <= nrows & 1 <= rc$col & rc$col <= ncols ) {
        return(data[rc$row,rc$col]);
    } else {
        return(attr(data,"header")$NODATA_value);
    }
```

```
}

value.gridfloat = function(data, xll, yll, xur, yur)
{
    rc = geo2rc.gridfloat(data,xll,yll);
    row_start = rc$row;
    col_start = rc$col;
    rc = geo2rc.gridfloat(data,xur,yur);
    row_end   = rc$row;
    col_end   = rc$col;

    return(data[row_start:row_end,col_start:col_end]);
}

pointingrid.gridfloat = function(data, x, y)
{
    bbox = bbox.gridfloat(data);

    xll = bbox[1];
    yll = bbox[2];
    xur = bbox[3];
    yur = bbox[4];

    if ( xll <= x && x <= xur && yll <= y && y <= yur ) { return(TRUE)} else {return(FALSE)}
}

polyingrid.gridfloat = function(data, x, y)
{
    bbox = bbox.gridfloat(data);
    xll = bbox[1];
    yll = bbox[2];
    xur = bbox[3];
    yur = bbox[4];
    index = which ( xll <= x & x < xur & yll <= y & y <= yur, arr.ind=TRUE);
```

```
    if ( length(index) == length(x) ) { return(TRUE); } else {return(FALSE)};
}

value.gridfloatlist = function(data, x, y)
{
    for ( ned in data ) {
        value = value.gridfloat(ned, x, y);
        if ( value != attr(ned,"header")$NODATA_value ) { return(value);}
    }
    return(value);
}

sum.landscape = function(data)
{
    header = attr(data,"header");
    nlayers = header$nlayers;

    total = data[,,1]*0;
    for ( l in 1:nlayers ) {
        total = total + data[,,l];
    }

    header =
          list(nrows=header$nrows,ncols=header$ncols,xllcorner=header$xllcorner,yllcorner=header$yllcorner,cellsize=header$cellsize,NODATA_value=h
          eader$NODATA_value,byteorder=header$byteorder);
    attr(total,"header") = header;
    return(total);
}

clip.grid2shapelist = function(ag, shapelist)
{
    # ag is the ascii grid
    # shapelist is a R list of shapes
```

```
header_ag  = attr(ag,"header");
bbox_ag    = bbox.grid(ag);

bbox_shape = bbox.shapefile(shapelist[[1]]);
for ( shape in shapelist ) {
    bbox = bbox.shapefile(shape);
    bbox_shape$xmin = min( c(bbox_shape$xmin, bbox$xmin) );
    bbox_shape$ymin = min( c(bbox_shape$ymin, bbox$ymin) );
    bbox_shape$xmax = max( c(bbox_shape$xmax, bbox$xmax) );
    bbox_shape$ymax = max( c(bbox_shape$ymax, bbox$ymax) );
}


cellsize  = header_ag$cellsize;

x_ag = header_ag$xllcorner + cellsize*(1:header_ag$ncols - 1);
y_ag = header_ag$yllcorner + cellsize*(1:header_ag$nrows - 1);


xllcorner = max(   c( x_ag[ x_ag <= bbox_shape$xmin ], header_ag$xllcorner )   );
yllcorner = max(   c( y_ag[ y_ag <= bbox_shape$ymin ], header_ag$yllcorner )   );
xurcorner = min(   c( x_ag[ x_ag >= bbox_shape$xmax ], bbox_ag$xurcorner   )   );
yurcorner = min(   c( y_ag[ y_ag >= bbox_shape$ymax ], bbox_ag$yurcorner   )   );


nrows = ceiling(   (yurcorner - yllcorner)/cellsize   );
ncols = ceiling(   (xurcorner - xllcorner)/cellsize   );

cat("clip.grid2shapelist: creating grid\n");
ret = create.asciigrid(ncols, nrows, xllcorner, yllcorner, cellsize, -9999, -9999);

ret_header = attr(ret,"header");

ret_index = 1:(ret_header$nrows * ret_header$ncols) - 1;
```

```
    ret_allRows  = floor( ret_index/ret_header$ncols ) + 1;
    ret_allCols  = ret_index%%ret_header$ncols + 1;

    x_cent    = ret_header$xllcorner + ret_header$cellsize*(ret_allCols-0.5);
    y_cent    = ret_header$yllcorner + ret_header$cellsize*(ret_allRows-0.5);


    for ( shape in shapelist ) {
        cat("clip.grid2shape: calling point.in.shape\n");
        ret_index = point.in.shapefile(shape, coords = data.frame(x=x_cent, y = y_cent));

        ret_rows = ret_allRows[ret_index];
        ret_cols = ret_allCols[ret_index];

        cat("clip.grid2shape: packing data into final grid\n");
        for ( i in 1:length(ret_rows) ) {
          ret_r = ret_rows[i];
          ret_c = ret_cols[i];

          ret_y = ret_header$yllcorner + ret_header$cellsize*(ret_r - 0.5);
          ret_x = ret_header$xllcorner + ret_header$cellsize*(ret_c - 0.5);

          ag_r = ceiling(   (ret_y - header_ag$yllcorner)/header_ag$cellsize   );
          ag_c = ceiling(   (ret_x - header_ag$xllcorner)/header_ag$cellsize   );
          if ( 1 <= ag_r && ag_r <= header_ag$nrows && 1 <= ag_c && ag_c <= header_ag$ncols ) {
            ret[ret_r, ret_c] = ag[ag_r, ag_c];
          }
        }
    }
    return(ret);
}

clip.grid2shape = function(ag,shape)
{
```

```
    # ag is the ascii grid
    # shape is the shapefile.

    header_ag  = attr(ag,"header");
    bbox_ag    = bbox.grid(ag);
    bbox_shape = bbox.shapefile(shape);

    cellsize  = header_ag$cellsize;

    x_ag = header_ag$xllcorner + cellsize*(1:header_ag$ncols - 1);
    y_ag = header_ag$yllcorner + cellsize*(1:header_ag$nrows - 1);


    xllcorner = max(   c( x_ag[ x_ag <= bbox_shape$xmin ], header_ag$xllcorner )   );
    yllcorner = max(   c( y_ag[ y_ag <= bbox_shape$ymin ], header_ag$yllcorner )   );
    xurcorner = min(   c( x_ag[ x_ag >= bbox_shape$xmax ], bbox_ag$xurcorner   )   );
    yurcorner = min(   c( y_ag[ y_ag >= bbox_shape$ymax ], bbox_ag$yurcorner   )   );


    nrows = ceiling(   (yurcorner - yllcorner)/cellsize   );
    ncols = ceiling(   (xurcorner - xllcorner)/cellsize   );

cat("clip.grid2shape: creating grid\n");
    ret = create.asciigrid(ncols, nrows, xllcorner, yllcorner, cellsize, -9999, -9999);

    ret_header = attr(ret,"header");


    ret_index = 1:(ret_header$nrows * ret_header$ncols) - 1;
    ret_rows  = floor( ret_index/ret_header$ncols ) + 1;
    ret_cols  = ret_index%%ret_header$ncols + 1;

    x_cent    = ret_header$xllcorner + ret_header$cellsize*(ret_cols-0.5);
    y_cent    = ret_header$yllcorner + ret_header$cellsize*(ret_rows-0.5);
```

```
cat("clip.grid2shape: calling point.in.shape\n");
    ret_index = point.in.shapefile(shape, coords = data.frame(x=x_cent, y = y_cent));

    ret_rows = ret_rows[ret_index];
    ret_cols = ret_cols[ret_index];

cat("clip.grid2shape: packing data into final grid\n");
    for ( i in 1:length(ret_rows) ) {
        ret_r = ret_rows[i];
        ret_c = ret_cols[i];

        y_ret = ret_header$yllcorner + ret_header$cellsize*(ret_r - 0.5);
        x_ret = ret_header$xllcorner + ret_header$cellsize*(ret_c - 0.5);

        r_ag = ceiling(   (y_ret - header_ag$yllcorner)/header_ag$cellsize   );
        c_ag = ceiling(   (x_ret - header_ag$xllcorner)/header_ag$cellsize   );
        if ( 1 <= r_ag && r_ag <= header_ag$nrows && 1 <= c_ag && c_ag <= header_ag$ncols ) {
          ret[ret_r, ret_c] = ag[r_ag, c_ag];
        } else {
          ret[ret_r, ret_c] = -9999;
        }
    }
    return(ret);
}

asciigridplus2asciigrid = function(data, layer="")
{
    if ( layer == "" ) {
        layer = names(data$table)[2];
    }

    grid       = data$grid;
    N = length(grid) - sum(is.na(grid));
    grid[order(grid)[1:N]] = data$table[[layer]][order(data$table[[1]])]
```

```
    return(grid);

}

#######################################################
# Define functions to visualize raster data.
#
#######################################################
colors.generic = function(data, color_hints = list() )
{
    #######################################################
    # Default parameter settings.
    #
    #######################################################
    NA_color     = c(128, 128, 128);
    minneg_color = c( 64,   0,   0);
    maxneg_color = c(255,   0,   0);
    zero_color   = c(255, 255, 255);
    minpos_color = c(  0, 255, 255);
    maxpos_color = c(  0,   0, 128);

    Nnegcolors   = 10;
    Nposcolors   = 10;

    #######################################################
    # User override of default parameter values.
    #
    #######################################################
    if ( length(names(color_hints)[names(color_hints)=="NA_color"    ]) == 1 ) { NA_color     = color_hints$NA_color;     }
    if ( length(names(color_hints)[names(color_hints)=="minneg_color"]) == 1 ) { minneg_color = color_hints$minneg_color; }
    if ( length(names(color_hints)[names(color_hints)=="maxneg_color"]) == 1 ) { maxneg_color = color_hints$maxneg_color; }
    if ( length(names(color_hints)[names(color_hints)=="zero_color"  ]) == 1 ) { zero_color   = color_hints$zero_color;   }
    if ( length(names(color_hints)[names(color_hints)=="minpos_color"]) == 1 ) { minpos_color = color_hints$minpos_color; }
```

```
if ( length(names(color_hints)[names(color_hints)=="maxpos_color"]) == 1 ) { maxpos_color = color_hints$maxpos_color; }

#####################################################
# Building color table.
#
#####################################################

breaks     = sort(    unique( c(quantile(c(data[data<0],data[data>0]),na.rm=TRUE),-.Machine$double.xmin, 0) )    );
ncolors    = length(breaks)-1;
zero_index = (1:length(breaks))[breaks==0]-1; # This is the position of the zero color in the colors array.

colors = rep("#FFFFFF",ncolors);

colors[zero_index] = rgb(zero_color[1]/255,zero_color[2]/255,zero_color[3]/255);

if ( zero_index != 1 ) {
    # Then there are negative values.
    red       = seq(minneg_color[1],maxneg_color[1],length.out=zero_index-1);
    green      = seq(minneg_color[2],maxneg_color[2],length.out=zero_index-1);
    blue      = seq(minneg_color[3],maxneg_color[3],length.out=zero_index-1);
    alpha      = rep(1,zero_index-1);
    colors_neg = rgb(red/255,green/255,blue/255,alpha);
    colors[1:(zero_index-1)] = colors_neg;
}

if ( zero_index != ncolors ) {
    # Then there are positive values.
    Nposcolors = ncolors - zero_index;
    red       = seq(minpos_color[1],maxpos_color[1],length.out=Nposcolors);
    green      = seq(minpos_color[2],maxpos_color[2],length.out=Nposcolors);
    blue      = seq(minpos_color[3],maxpos_color[3],length.out=Nposcolors);
    alpha      = rep(1,Nposcolors);
    colors_pos = rgb(red/255,green/255,blue/255,alpha);
    colors[(zero_index+1):ncolors] = colors_pos;
```

```
    }

    n_breaks = length(breaks);
    return( list(colors_lut = colors, breaks = breaks, labels=paste(breaks[1:(n_breaks-1)]," - ",breaks[2:n_breaks],sep="")) );
}

colors.gridfloat = function(data, color_hints = list())
{
    return(colors.generic(data, color_hints=color_hints));
}

colors.asciigrid = function(data, color_hints = list())
{
    return(colors.generic(data, color_hints=color_hints));
}

colors.ATLSS_rainbow = function()
{
    breaks = c(-100,seq(0,1,by=0.05),100);

   colors_lut = c("#FFFFFF",
            "#F7F7F7",
            "#0000FF",
            "#0040FF",
            "#0080FF",
            "#00C0FF",
            "#00FFFF",
            "#33FFC6",
            "#66FF99",
            "#99FF66",
            "#C6FF33",
            "#FFFF00",
            "#FFD500",
            "#FFAA00",
```

```
            "#FF8000",
            "#FF5500",
            "#FF2A00",
            "#FF0000",
            "#FF0055",
            "#FF00AA",
            "#FF00FF",
            "#000000" );

    return(list(colors_lut= colors_lut, breaks=breaks));
}

colors.ATLSS_diff = function()
{
    breaks = c(-100,seq(-1,1,by=0.1),100);

    colors_lut = c("#FFFFFF", # 1    00
                   "#4F9AAD", # 2   01
                   "#60A5B6", # 3   02
                   "#71B0BF", # 4   03
                   "#83BBC8", # 5   04
                   "#94C6D1", #     05
                   "#A5D2DA", #     06
                   "#B7DDE3", #     07
                   "#C8E8EC", #     08
                   "#D9F3F5", # 10  09
                   "#EBFFFF", #     0A
                   "#F2F2F2", #     0B
                   "#FFFF9E", #     0C
                   "#F7EF90", #     0D
                   "#EFE082", # 15  0E
                   "#E7D175", #     0F
                   "#DFC567", #     10
                   "#D7B55A", #     11
```

```
                "#CFA44C", #     12
                "#C7953F", # 20  13
                "#BF8631", #     14
                "#B77724"); #     15
                # "#AF6817"); # 23 16

    return(list(colors_lut= colors_lut, breaks=breaks));
}

colors.ATLSS_hydrodiff = function(data)
{
    largest_neg = max(data[data<0]);

    breaks = c(-100,seq(-1,0.1,by=0.1),largest_neg,seq(0,1,by=0.1),100);
    colors_lut = c("#000000",   #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "#F2F2F2",      # 12, 0D
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "# ",     #
                "#FF0000");      #
}

colors.ATLSS_fire = function()
```

```
{
    breaks = c(-100,0,1,2);
    colors_lut = c("#00F000","#FFFF00","#FF0000");
    return(list(colors_lut=colors_lut, breaks=breaks));
}

colors.shape = function(shape, by="", color_hints=list(), colors_lut=list() )
{

    values = shape$dbf$dbf[[by]];
    l_colors_lut = colors.generic(values, color_hints=color_hints);

    if ( sum( names(l_colors_lut) == names(colors_lut )) != 3 ){
        l_colors_lut = colors_lut;
    }

    colors = rep("",length(values));

    for ( v in unique(values) ) {
        index = which(v <= l_colors_lut$breaks, arr.ind=TRUE)[1]-1;
cat("v    = ",v   ,"\n");
cat("index = ",index,"\n");
        colors[ values == v ] = l_colors_lut$colors_lut[index];
    }

    ret = list(colors_lut=colors, breaks = l_colors_lut$breaks, labels = l_colors_lut$labels);
    return(ret);
}


plot.gridfloat = function(data, colors_lut=list(),  ...)
{
    ncols    = attr(data,"header")$ncols;
    nrows    = attr(data,"header")$nrows;
```

```
    xllcorner = attr(data,"header")$xllcorner;
    yllcorner = attr(data,"header")$yllcorner;
    cellsize  = attr(data,"header")$cellsize;

    x = seq(xllcorner,xllcorner+cellsize*ncols,length.out=ncols+1);
    y = seq(yllcorner,yllcorner+cellsize*nrows,length.out=nrows+1);

    if ( sum(names(colors_lut)%in%c("colors_lut", "breaks") ) == 2 ) {
        image(x, y, t(data), col=colors_lut$colors_lut, breaks=colors_lut$breaks, ...);
    } else {
        image(x, y, t(data), ...);
    }
}


plot.asciigrid = function(data, ...)
{
    return(plot.gridfloat(data,...));
}

plot.asciigridplus = function(data, layer="", ...)
{
    if ( layer == "" ) {
        layer = names(data$table)[2];
    }

    grid = asciigridplus2asciigrid(data, layer=layer)

    plot.gridfloat(grid,...);

    invisible(grid);
}

plot.gridfloat_bbox = function(data, add=TRUE,...)
```

```
{
    ncols     = attr(data,"header")$ncols;
    nrows     = attr(data,"header")$nrows;
    xllcorner = attr(data,"header")$xllcorner;
    yllcorner = attr(data,"header")$yllcorner;
    cellsize  = attr(data,"header")$cellsize;

    xurcorner = xllcorner + cellsize*ncols;
    yurcorner = yllcorner + cellsize*nrows;

    usr = par()$usr;
    usr_y = usr[3];
    usr_x = usr[1];

    xpd = par()$xpd;
    par(xpd=NA);
    lines(c(xllcorner,xllcorner),c(yurcorner,usr_y));
        text( c(xllcorner), c(usr_y), labels=c(format(xllcorner,digits=5)) );
    lines(c(xurcorner,xurcorner),c(yurcorner,usr_y));
        text( c(xurcorner), c(usr_y), labels=c(format(xurcorner,digits=5)) );
    lines(c(xurcorner,usr_x),c(yurcorner,yurcorner));
        text( c(usr_x), c(yurcorner), labels=c(format(yurcorner,digits=5)) );
    lines(c(xurcorner,usr_x),c(yllcorner,yllcorner));
        text( c(usr_x), c(yllcorner), labels=c(format(yllcorner,digits=5)) );
    par(xpd=xpd);
}

plot.landscape = function(data, layer = 1, col=NA, ...)
{
    header = attr(data,"header");
    d = data[,,layer];
    attr(d,"header") = header;
    if ( is.list(col) ) {
        colors_lut = col$colors_lut;
```

```
        breaks     = col$breaks;
        plot.gridfloat((d),col=colors_lut, breaks=breaks);
    } else {
        plot.gridfloat((d), ... );
    }
}

#######################################################
# Define functions to manipulate shapefile data.
#
#######################################################
subset.shapefile = function(shape, attr, attr_list)
{
    index = shape$dbf$dbf[[attr]] %in% attr_list;

    # Build the shp list
    shp =  shape$shp$shp[index];
    header = shape$shp$header;

    header$xmin = shp[[1]]$box["xmin"];
    header$ymin = shp[[1]]$box["ymin"];
    header$xmax = shp[[1]]$box["xmax"];
    header$ymax = shp[[1]]$box["ymax"];
    for ( i in 1:length(shp) ) {
        header$xmin = min(c(header$xmin, shp[[i]]$box["xmin"] ));
        header$ymin = min(c(header$ymin, shp[[i]]$box["ymin"] ));
        header$xmax = max(c(header$xmax, shp[[i]]$box["xmax"] ));
        header$ymax = max(c(header$ymax, shp[[i]]$box["ymax"] ));
    }
    shp = list(shp=shp, header=header);

    #Build the dbf list
    dbf_list = list();
    for ( n in names(shape$dbf$dbf) ) {
```

```
        dbf_list[[n]] = shape$dbf$dbf[[n]][index];
    }
    dbf = list( dbf=as.data.frame(dbf_list), header=shape$dbf$header);

    #Build the shx list
    shx_index  = shape$shx$index[index,];
    header     = shape$shx$header;
    header     = shape$shx$header$xmin = shp$header$xmin;
    header     = shape$shx$header$ymin = shp$header$ymin;
    header     = shape$shx$header$xmax = shp$header$xmax;
    header     = shape$shx$header$ymax = shp$header$ymax;

    shx = list(shx=shx_index, header=header);

    newshape = list(shp=shp, shx=shx, dbf=dbf);

    return(newshape);
}

centroids.shapefile = function(shape)
{
}

bbox.shapefile = function(shape)
{
    ret = list(xmin = shape$shp$header$xmin,
               xmax = shape$shp$header$xmax,
               ymin = shape$shp$header$ymin,
               ymax = shape$shp$header$ymax );
    return(ret);
}

point.in.shapefile = function(shape, coords = data.frame(x=c(),y=c()), shapeIDname="" )
{
```

```
    num_shapes = length(shape$shp$shp);
    ret.index  = rep(FALSE,length(coords$x));
    ret.shape  = rep("",length(coords$x));
    for ( i in 1:num_shapes ) {

        shape_ID = 0;
        if ( nchar(shapeIDname) != 0 ) {
          shape_ID  = shape$dbf$dbf[[shapeIDname]][i];
        }
        parts     = c(shape$shp$shp[[i]]$parts, shape$shp$shp[[i]]$num.points) + 1;
        num.parts = shape$shp$shp[[i]]$num.parts;
        for ( p in 1:num.parts ) {
          start          = parts[p]
          end            = parts[p+1] - 1
          x              = shape$shp$shp[[i]]$points$X[start:end];
          y              = shape$shp$shp[[i]]$points$Y[start:end];
          index          = which(point.in.polygon(coords$x, coords$y, x, y) != 0, arr.ind=TRUE);
          ret.index[index] = TRUE;
#cat("length(index) = ",length(index),"\n");
#cat("shape_ID = ",shape_ID,"\n");
          ret.shape[index] = shape_ID;
        }
    }
    attr(ret.index, "shape_ID") = ret.shape;
    return(ret.index);
}

grid.to.SpatialPolygons = function(grid, projection)
{
   header = attr(grid,'header');

   nrows     = header$nrows;
   ncols     = header$ncols;
   xllcorner = header$xllcorner;
```

```
  yllcorner = header$yllcorner;
  cellsize  = header$cellsize;

  m          = matrix(0, nrow=5, ncol=2);
  polygonList = list();
  count       = 1;
  for ( r in 1:nrows ) {
    for ( c in 1:ncols ) {

      if ( !is.na(grid[r,c]) ) {
        x0 = xllcorner + cellsize*(c-1);
        x1 = x0 + cellsize;
        y0 = yllcorner + cellsize*(r-1);
        y1 = y0 + cellsize;

        m[,1] = c(x0, x1, x1, x0, x0);
        m[,2] = c(y0, y0, y1, y1, y0);

        polygonList[count] = Polygon(m, hole=FALSE);
        count = count + 1;
      }
    }
  }

  polygonsObject = Polygons(polygonList, ID='Polygons');
  ret            = SpatialPolygons( list(polygonsObject), proj4string=CRS(projection));
  invisible(ret);

}

intersects.grid.and.SpatialPolygons = function(grid, spObject)
{
  ret    = grid*0;
  header = attr(grid,'header');
```

```
nrows     = header$nrows;
ncols     = header$ncols;
xllcorner = header$xllcorner;
yllcorner = header$yllcorner;
cellsize  = header$cellsize;

m          = matrix(0, nrow=5, ncol=2);
count      = 1;
for ( r in 1:nrows ) {
   for ( c in 1:ncols ) {

      if ( !is.na(grid[r,c]) ) {
         x0 = xllcorner + cellsize*(c-1);
         x1 = x0 + cellsize;
         y0 = yllcorner + cellsize*(r-1);
         y1 = y0 + cellsize;

         m[,1] = c(x0, x1, x1, x0, x0);
         m[,2] = c(y0, y0, y1, y1, y0);

         polygonList          = list( Polygon(m, hole=FALSE) );
         polygonsObject       = Polygons(polygonList, ID='gridCells');
         spatialPolygonsObject = SpatialPolygons(list(polygonsObject), proj4string=spObject@proj4string );
         if ( gIntersects(spatialPolygonsObject, spObject) ) {
            ret[r,c] = 1;
         }
      }
   }
}

attr(ret, 'header') = header;
invisible(ret);
```

```
}

shapefile.to.SpatialPolygons = function(shape, projection)
{
   polygonList = list();

   num_shapes = length(shape$shp$shp)

   count = 1;
   for ( i in 1:num_shapes ) {
      parts     = c(shape$shp$shp[[i]]$parts, shape$shp$shp[[i]]$num.points) + 1;
      num_parts = shape$shp$shp[[i]]$num.parts;
      for ( p in 1:num_parts ) {
         start            = parts[p];
         end              = parts[p+1] - 1;
         m                = matrix(0, nrow=(end - start + 1), ncol=2)
         m[,1]            = shape$shp$shp[[i]]$points$X[start:end];
         m[,2]            = shape$shp$shp[[i]]$points$Y[start:end];
         poly             = Polygon(m, hole=FALSE);
         polygonList[count] = poly
         count            = count + 1;
      }
   }

   #ret = SpatialPolygons(polygonList, proj4string=CRS('+proj=utm +zone=15 +north') );
   polygonsObject = Polygons(polygonList, ID='Polygons');
   ret = SpatialPolygons(list(polygonsObject), proj4string=CRS(projection) );

   invisible(ret );
}

transform.shapefile = function(shape, proj_orig, proj_new, offset_new=list(x=0,y=0) )
{
    ret       = shape;
```

```
  num_shapes = length(shape$shp$shp);

# Dec 5, 2011. SMS. This would seem like the slick way to do this,
# but it doesn't work correctly. That is, it doesn't give you
# an accurate assesment of the new bounding box diminsions.
# So, I use this approach to get a starting point for each
# box diminsion, but I update the bounding box diminsions
# after converting each shape.
  d = data.frame( x = c(shape$shp$header$xmin,shape$shp$header$xmax),
                  y = c(shape$shp$header$ymin,shape$shp$header$ymax) );
  SP_orig  = SpatialPoints( d, proj4string=CRS(proj_orig) );
  SP_trans = spTransform(SP_orig,                CRS(proj_new)  );
  ret$shp$header$xmin = SP_trans@coords[1,1] + offset_new$x;
  ret$shp$header$ymin = SP_trans@coords[1,2] + offset_new$y;
  ret$shp$header$xmax = SP_trans@coords[2,1] + offset_new$x;
  ret$shp$header$ymax = SP_trans@coords[2,2] + offset_new$y;


  for ( i in 1:num_shapes ) {

      d = data.frame(x = shape$shp$shp[[i]]$points$X, y = shape$shp$shp[[i]]$points$Y);
      SP_orig  = SpatialPoints(d, proj4string=CRS(proj_orig) );
      SP_trans = spTransform(SP_orig, CRS(proj_new));
      ret$shp$shp[[i]]$points$X = SP_trans@coords[,1] + offset_new$x;
      ret$shp$shp[[i]]$points$Y = SP_trans@coords[,2] + offset_new$y;

      ret$shp$shp[[i]]$box["xmin"] = min(ret$shp$shp[[i]]$points$X);
      ret$shp$shp[[i]]$box["ymin"] = min(ret$shp$shp[[i]]$points$Y);
      ret$shp$shp[[i]]$box["xmax"] = max(ret$shp$shp[[i]]$points$X);
      ret$shp$shp[[i]]$box["ymax"] = max(ret$shp$shp[[i]]$points$Y);

    ret$shp$header$xmin = min(c(ret$shp$header$xmin, ret$shp$shp[[i]]$box["xmin"]));
    ret$shp$header$ymin = min(c(ret$shp$header$ymin, ret$shp$shp[[i]]$box["ymin"]));
    ret$shp$header$xmax = max(c(ret$shp$header$xmax, ret$shp$shp[[i]]$box["xmax"]));
```

```
        ret$shp$header$ymax = max(c(ret$shp$header$ymax, ret$shp$shp[[i]]$box["ymax"]));
    }


    return(ret);
}

#######################################################
# Define functions to visualize shapefile data.
#
#######################################################
plot.shapefile = function(shape,add=FALSE,col=list(colors_lut=c("#FFFFFF00")),...)
{
    if ( !add ) {
        plot( c(shape$shp$header$xmin,shape$shp$header$xmax),
            c(shape$shp$header$ymin,shape$shp$header$ymax),
            xlab="",ylab="",type="n");
    }


    shape$num_shapes = length(shape$shp$shp);
    for ( i in 1:shape$num_shapes ) {
        color = col$colors_lut[(i-1)%%length(col$colors_lut)+1];

        parts = c(shape$shp$shp[[i]]$parts, shape$shp$shp[[i]]$num.points) + 1;
        num.parts = shape$shp$shp[[i]]$num.parts;
        for ( p in 1:num.parts ) {
          start = parts[p]
          end   = parts[p+1] - 1
          x = shape$shp$shp[[i]]$points$X[start:end];
          y = shape$shp$shp[[i]]$points$Y[start:end];
          polygon(x,y,col=color,...)
        }
    }
```

```
}

plot.shape = function(shape,...)
{
    plot.shapefile(shape,...);
}

PDF.asciigrid = function(grid, file)
{
   colors = list( colors_lut = c("#878787",   "#C6C73CFF", "#A0CA3AFF", "#7ACD37FF", "#53CF35FF", "#009900FF"),
                  breaks = c( -2.225074e-308,   0.000000e+00,   0.2, 0.4, 0.6, 0.8, 1.0),
                  labels = c( "0", "up to 0.2", "0.2 to 0.4", "0.4 to 0.6", "0.6 to 0.8", "0.8 - 1") );

   pdf(file=file, height=7.5, width=11, paper="USr", pagecentre=TRUE, onefile=TRUE);

   plot.asciigrid(grid, colors_lut = colors);

   dev.off();

}

PDF.asciigridplus = function(grid, file, layer)
{
   colors = list( colors_lut = c("#878787",   "#C6C73CFF", "#A0CA3AFF", "#7ACD37FF", "#53CF35FF", "#009900FF"),
                  breaks = c( -2.225074e-308,   0.000000e+00,   0.2, 0.4, 0.6, 0.8, 1.0),
                  labels = c( "0", "up to 0.2", "0.2 to 0.4", "0.4 to 0.6", "0.6 to 0.8", "0.8 - 1") );

   pdf(file=file, height=7.5, width=11, paper="USr", pagecentre=TRUE, onefile=TRUE);

   plot.asciigridplus(grid, layer=layer, colors_lut = colors);

   dev.off();
}
```

```
#####################################################
# Define some general utility functions
#
#####################################################
km2deg = function(distance,phi)
{
    # phi is the the latitude where the distance is begin computed.
    M          = 6367.854; #km # mean meridianal radius of curvature
    N          = 6388.854; #km # mean normal radius of curvature
    kmPerDeg_lat = pi/180 * M; # ~111 km per degree of arc lat.
    kmPerDeg_lng = pi/180 * N * cos(phi*pi/180);
    dist_lat  = distance/kmPerDeg_lat;
    dist_lng  = distance/kmPerDeg_lat;

    return(list(dist_lat = dist_lat, dist_lng = dist_lng));
}

deg2km = function(distance,phi)
{
    # phi is the the latitude where the distance is begin computed.
    M          = 6367.854; #km # mean meridianal radius of curvature
    N          = 6388.854; #km # mean normal radius of curvature
    kmPerDeg_lat = pi/180 * M; # ~111 km per degree of arc lat.
    kmPerDeg_lng = pi/180 * N * cos(phi*pi/180);

    dist_lat  = distance*kmPerDeg_lat;
    dist_lng  = distance*kmPerDeg_lat;

    return(list(dist_lat = dist_lat, dist_lng = dist_lng));

}
```

VIV. Source code for lavegmod_template.config

```
// RunType = Run // Default
// RunType = ConfigOnly
// RunType = ConfigAndReport
// RunType = Test
// RunType = ReadConfigOnly

StartYear   = __STARTYEAR__
JumpToYear  = __STARTYEAR__
NumYears    = 25

Scenario    = __SCENARIO__
Group       = __GROUP__
Variation   = __VARIATION__
Region      = __REGION__
Realization = __REALIZATION__
Year        = __YEAR__
TablePath   = __TABLEPATH__

PreProc.EcoHydroBoxIDFile          = __EcoHydroBoxIDFile__
PreProc.EcoHydroStgHeightFile      = __EcoHydroStgHeightFile__
PreProc.EcoHydroSalinityFile       = __EcoHydroSalinityFile__
PreProc.EcoHydroTempFile           = __EcoHydroTempFile__

PreProc.WetlandMorphCellIDFile     = __WetlandMorphCellIDFile__

PreProc.WetlandMorphLandWaterFile  = __WetlandMorphLandWaterFile__
PreProc.WetlandMorphElevationFile  = __WetlandMorphElevationFile__

InitialConditionFile               = __InitialConditionFile__

WaveAmplitudeFile                  = __WaveAmplitudeFile__
MeanSalinityFile                   = __MeanSalinityFile__
ElevationFile                      = __ElevationFile__
```

```
SummerMeanWaterDepthFile          = __SummerMeanWaterDepthFile__
SummerMeanSalinityFile            = __SummerMeanSalinityFile__
SummerMeanTempFile                = __SummerMeanTempFile__

// Official output file name format
// A##_<projectid/groupid>_A##_<PMtext> (A=letter, #=number)
// For the veg model output:
// A##_<projectid/groupid>_A##_VHRABRYR
//                      ^^ ^  ^
//                      || |  |- two digit year code with the first year of data in the file.
//                      || |---- three digit abbreviation, model dependent
//                      ||------ two letter region code, HR = one of PB, AA, CP, LA
//                      |------- Initicates that this is veg model output, always = V for this model.
// OutputTemplate     = ./A##_<projectid/groupid>_A##_V__CVR<Year>.asc+
//
// If you omit the OutputYears parameter, the model will produce output for every year.

// VEG file
OutputFile          = __OutputFile__
// VG file
OutputTemplate       = __OutputTemplate__
OutputYears          = 5,10,15,20,25


// ##########################
// # Species Parameters
// #
// ##########################
EstablishmentFile = __TABLEPATH__/Establishment.csv

Species0.Name    = BareGround
Species0.Abr     = BAREGRND
Species0.Index   = 0
Species0.D       = __TABLEPATH__/DOMDeath_BAREGRND.csv
```

```
Species1.Name    = DeltaSplay
Species1.Abr     = SPLAY
Species1.Index   = 1
Species1.D       = __TABLEPATH__/DOMDeath_SPLAY.csv

Species2.Name  = WaxMyrtle
Species2.Abr   = WAXM
Species2.Index = 2
Species2.D       = __TABLEPATH__/DOMDeath_WAXM.csv

Species3.Name   = Cut-grass
Species3.Abr    = CUTGR
Species3.Index  = 3
Species3.D       = __TABLEPATH__/DOMDeath_CUTGR.csv

Species4.Name   = Maidencane
Species4.Abr    = MAID
Species4.Index  = 4
Species4.D       = __TABLEPATH__/DOMDeath_MAID.csv

Species5.Name   = Thin-mat
Species5.Abr    = THIN
Species5.Index  = 5
Species5.D       = __TABLEPATH__/DOMDeath_THIN.csv

Species6.Name   = Swamp_Forest
Species6.Abr    = SWAMP
Species6.Index  = 6
Species6.D       = __TABLEPATH__/DOMDeath_SWAMP.csv

Species7.Name   = Cattail
Species7.Abr    = CAT
Species7.Index  = 7
```

```
Species7.D       = __TABLEPATH__/DOMDeath_CAT.csv

Species8.Name   = Sawgrass
Species8.Abr    = SAWG
Species8.Index  = 8
Species8.D      = __TABLEPATH__/DOMDeath_SAWG.csv

Species9.Name   = Bulltongue
Species9.Abr    = BULL
Species9.Index  = 9
Species9.D      = __TABLEPATH__/DOMDeath_BULL.csv

Species10.Name   = RoseauCane
Species10.Abr    = ROSEAU
Species10.Index  = 10
Species10.D      = __TABLEPATH__/DOMDeath_ROSEAU.csv

Species11.Name   = Bullwhip
Species11.Abr    = WHIP
Species11.Index  = 11
Species11.D      = __TABLEPATH__/DOMDeath_WHIP.csv

Species12.Name   = Shrub-scrub
Species12.Abr    = SCRUB
Species12.Index  = 12
Species12.D      = __TABLEPATH__/DOMDeath_SCRUB.csv

Species13.Name   = Wiregrass
Species13.Abr    = WIRE
Species13.Index  = 13
Species13.D      = __TABLEPATH__/DOMDeath_WIRE.csv

Species14.Name   = Paspalum
Species14.Abr    = PASP
```

```
Species14.Index  = 14
Species14.D      = __TABLEPATH__/DOMDeath_PASP.csv

Species15.Name   = BrackishMarsh
Species15.Abr    = BRACK
Species15.Index  = 15
Species15.D      = __TABLEPATH__/DOMDeath_BRACK.csv

Species16.Name   = Needlerush
Species16.Abr    = NEEDL
Species16.Index  = 16
Species16.D      = __TABLEPATH__/DOMDeath_NEEDL.csv

Species17.Name   = Saltgrass
Species17.Abr    = SALT
Species17.Index  = 17
Species17.D      = __TABLEPATH__/DOMDeath_SALT.csv

Species18.Name   = Oystergrass
Species18.Abr    = OYST
Species18.Index  = 18
Species18.D      = __TABLEPATH__/DOMDeath_OYST.csv

Species19.Name   = BlackMangrove
Species19.Abr    = MANGR
Species19.Index  = 19
Species19.D      = __TABLEPATH__/DOMDeath_MANGR.csv

Species20.Name  = Water
Species20.Abr   = WATER
Species20.Index = 20
Species20.D     = __TABLEPATH__/DOMDeath_WATER.csv

SpeciesSAV.Name  = SAV
```

```
SpeciesSAV.Abr   = SAV
SpeciesSAV.Index = 21
SpeciesSAV.Intercept = 1.83
SpeciesSAV.Temp  = -3.731E-2
SpeciesSAV.Sal   = -7.766E-2
SpeciesSAV.Depth = -2.588E-4
SpeciesSAV.D     = __TABLEPATH__/DOMDeath_SAV.csv

Species21.Name  = Not modeled
Species21.Abr   = NOTMOD
Species21.Index = 22
Species21.D     = __TABLEPATH__/DOMDeath_NOTMOD.csv
```