RHESSys Bootcamp

SESSION 1. Introduction to RHESSys

1. Introduction

RHESSys (Regional Hydro-Ecological Simulation System) is a GIS-based, terrestrial ecohydrologic modeling framework designed to simulate carbon, water and nutrient fluxes at the watershed scale. RHESSys models the temporal and spatial variability of ecosystem processes and interactions at a daily time step over multiple years by combining a set of physically based process models and a methodology for partitioning and parameterizing the landscape. Detailed model algorithms are available in Tague and Band (2004).

2. Useful external links

RHESSys homepage: http://fiesta.bren.ucsb.edu/~rhessys/ (not updated well)
RHESSys wiki: http://wiki.icess.ucsb.edu/rhessys/RHESSys (updated frequently)
RHESSys sourceforge: http://rhessys.svn.sourceforge.net/ (where get the code)
GRASS full command index: http://grass.fbk.eu/gdp/html_grass64/full_index.html (Check GRASS commands)

3. Required programs

a. *GRASS*: A free UNIX-based GIS software (http://grass.fbk.eu/)
b. *GRASS2WORLD* (*g2w*): A program to generate a worldfile from spatial data layers within GRASS GIS
c. AverageTables (*rat*): Necessary to simulate *g2w*
d. *CREATEFLOWPATH* (*cf*): A program to generate a flowtable which describes connectivity between patches within a hillslope. Only needed in the routing mode.
e. *LAIREAD*: A program to incorporate LAI maps into the existing worldfile. (prescribed LAI mode)
f. *RHESSys*: A main program to simulate the model.
g. *R*: A free software environment for statistical computing and graphics for post-processing. (http://www.R-project.org/)

SESSION 2. Set up programs

SESSION 3. Preparing Spatial Input Datasets (GRASS GIS)

1. GRASS Set-up

Start GRASS
```
Open the terminal
$ grass
Specify the name of location (geographically name; Coweeta)
*Delete the characters with the space key
Specify the name of mapset (usually user name: taehee)
Specify the directory path to database (/home/rhessys/bootcamp/GRASSdata)
ESC + ENTER
*Usually GRASS has directory structure like DATABASE/LOCATION/MAPSET
```

Set up a new location (need to do just once)
```
Create new location? y
Do you have all this information? y
Specify the coordinate system. D (others) & y
One line description. Coweeta & y
```

```
Specify projection name. Type ‘list’ – Hit space key to see next page – Type ‘stp’
   (State Plane)
Datum? y – Type ‘list’ – Type ‘wgs84’
Transformation Parameters? Type ‘list’ – Type ‘1’
FIPS code? Type ‘list’ – Type ‘37’ (NC) – Correct? Y
County FIPS? Type ‘list’ – Type ‘113’ (Macon) Correct? Y
State plane 1927 or 1983? Type ‘83’ – Units? 3 (meter)
```

Define the default region (boundary coordinates from the header of an ascii file)

```
North Edge: 156210 - South Edge: 149500 - West Edge: 200500 - East Edge: 206900
Grid Resolutions: 10 (East-West), 10 (North-South) – Esc+Enter
Check the row and column numbers – Accept this region? Y – Enter – Esc + Enter
```

Create the mapset

```
Create <taehee> as a new mapset? Y – It will start GRASS
```

2. *Touring GRASS*

a. General commands *g.region, g.list, g.rename, g.remove*
b. Display commands: *d.mon, d.rast. d.what.rast, d.vect. d.zoom, d.redraw, d.erase*
c. Raster commands: *r.in.arc, r.out.arc, r.mapcalc, r.mask, r.reclass. r.stats*
d. Vector commands: *v.in.ogr*
   *Tips: ‘--help’ command line option will show all usage and flags for the command

3. *Basic UNIX commands*

```
$ ls : list the name of all files in a directory. It has many command line options.
$ rm : delete a file. It also has many command line options.
$ cd : change directory. For example, you can change directory to our your student
   folder (cd .. : goes to the upper directory, cd / : goes to root directory,
   cd : return to your home directory)
$ pwd: display the current working directory
$ mkdir/rmdir : create/remove directory (rm –f filename: remove the file
   permanently)
$ mv : change file name (mv oldname newname)
$ more : view the content of a text file page by page with the space key, to quit
   type ‘:q’
$ cp : make a copy of a file. (cp filename newfilename)
$ chmod : change file permissions (chmod +x script: make the file executable)
$ ↑ or ↓ (upper/lower cursors): shifts in the command line history
*Use the tab key for auto completion
```

4. *Import Arc Data into GRASS*

Input files (Coweeta Hydrologic Lab, Otto, NC)

```
pwd # where am I?
cd /home/rhessys/bootcamp/ARCdata # change the current directory
ls # show the all files and directories in the current directory
```

dem10_lidar.asc (10-m LiDAR elevation ARC ascii file)
estlai_10m.asc (10-m estimated LAI ARC ascii file)
stream_state.shp (streams ARC shape file)
weir_cwt_state.shp (weirs ARC shape file)
c_basin_state.shp (watershed boundaries ARC shape file)
c_roads_state.shp (roads ARC shape file)

Import the Arc ascii file of DEM

```
ls *.asc # show all files in the working directory which extension is 'asc'
r.in.arc –help # show the usage of r.in.arc command
r.in.arc input=dem10_lidar.asc output=dem10
g.list –help
g.list rast # show the list of raster in the current mapset and type 'q' to exit
d.mon x1 # start the graphics display monitor
d.rast dem10 # display the 'dem10' raster
```

Import the Arc shape files

```
ls *.shp # show all files in the working directory which extension is 'shp'
v.in.ogr dsn=stream_state.shp output=streams -o # override the projection
v.in.ogr dsn=weir_cwt_state.shp output=weirs -o
v.in.ogr dsn=c_basin_state.shp out=watersheds -o
v.in.ogr dsn=c_roads_state.shp out=roads -o
d.vect weirs # display the 'weirs' vector
d.vect streams color=blue # line color is blue
d.vect map=watersheds color=black fcolor=none # no fill color
d.zoom # zoom in the region specified in mouse
d.zoom -r # reverse it
```

5. *Preprocessing DEM data*

Generate a depressionless DEM and a flow direction maps (D-8 direction)

```
r.fill.dir input=dem10 elev=dem10f dir=dir10 # r.terraflow is better for massive
  grid data
```

Generate a flow accumulation and a drainage maps

```
r.watershed elev=dem10f acc=acc10 drain=drain10 # Use the depressionless DEM
d.erase
d.rast acc10
And overlay three vector files (weirs, streams, watersheds)
```

Delineation of a watershed boundary (basin) and a hillslope maps

```
d.zoom # zoom into the target watershed (WS02) and quit
d.what.rast # click and get the coordinate (East, North) for a grid with high
  accumulation values closest to the weir and quit
g.region -d # reverse to the default region set-up
d.redraw # redraw the map with the new region set-up
*REMEMBER that GRASS always simulate the command only in the zoomed region
r.water.outlet drain=drain10 basin=basin_ws02 east=205054.46 north=154747.14 #
  draw the watershed boundary from the specified coordinates
*Copy and paste the coordinates from d.what.rast command
*Basin map will have the value 1 for inside the watershed boundary, 0 for outside
d.erase # erase the screen
d.rast basin_ws02
d.what.rast # check the watershed raster with (1 or 0)
d.vect watersheds fcolor=none # Overlay the watershed boundaries to check the
  result
```

Generate a hillslope and a stream map based on stream lines

```
r.watershed elev=dem10f stream=stream_th1000 th=1000 # use the depressionless DEM
d.rast stream_th1000
d.vect streams
```

```
*Increase or decrease the threshold values to match the generated streams with
  given streams, and generate the final stream raster
*If there are significant differences between generated and given streams, use a
  burning option which artificially decrease the elevation at stream pixels.
r.watershed elev=dem10f stream=streams half=hillslope th=500
d.rast hillslope
d.vect watersheds fcolor=none
d.zoom # zoom into the WS02
d.rast streams -o # -o option for overlay two rasters
*Note that each hillslope is correspondent to each stream reach
```

Setting a mask to ignore outside a watershed boundary

```
r.mask input=basin_ws02 maskcats=1 # mask all regions except for pixels with 1 in
  the ‘basin_ws02’ raster
d.rast dem10 # check the mask was properly set
```

Other landscape representation maps (basin-hillsope-zone-patch-stratum) (All ID-related)

```
r.info basin_ws02 # check the number of columns
r.mapcalc 'patch = row()*<number of columns> + col()'
r.mask -r # remove the mask
d.rast patch # check the patch map
*Pixels with different number (ID) will be simulated independently in the model,
  also associated with different processes
```

Other spatial input data: *slope, aspect, wetness index, horizon, and road* maps

```
r.slope.aspect elev=dem10 slope=slope10 aspect=aspect10 # use the original DEM
r.topidx input=dem10 out=topidx10
r.stats -c topidx10 # check the statics of topidx raster
r.mapcalc ‘topidx10_100 = topidx10 * 100’
r.horizon -d elev=dem10 dir=0 horizon=east # sunrise horizon angle
r.horizon -d elev=dem10 dir=180 horizon=west # sunset horizon angle
g.list rast # check the rasters produced
r.mapcalc ‘e_horizon_1000 = sin(east_0) * 1000’ # convert to degree to sin
r.mapcalc ‘w_horizon_1000 = sin(west_0) * 1000’  # convert to degree to sin
*More details are available RHESSys wiki
  (http://wiki.icess.ucsb.edu/rhessys/Preparing_input_data_sets)
```

Wetness index values are used to simulate the topmodel option in RHESSys. More details on wetness index are available in Beven & Kirkby (1979).

SESSION 4. GRASS2WORLD (*g2w*)

g2w transforms GIS input data into a RHESSys text input (world file) under GRASS environment. The worldfile, is a long text file based on RHESSys's hierarchical structure (ID-related) that fully describes the ecosystem and hydrological state variables for each patch or pixel in the watershed. Grass2world is a program executed from within GRASS to use GIS data and a template file to create the worldfile. (http://wiki.icess.ucsb.edu/rhessys/Worldfiles)

1. *Template File*

A set of instructions that tells g2w how to create the initial values for each of the state variables in a spatial object at each level of the spatial hierarchy.

2. *Structure*

```
more templates/template514 # check the given template file
*Use the space key to see the next page
```

a. Header: specify the directory paths to default files and climate data
b. Definition of hierarchical landscape representation: basin – hillslope – zone – patch – stratum (featured by unique ID)
c. Relate each hierarchical structure to parameters through *default_ID* (same IDs in default files) (basin, hillslope, zone, soil, land use, and vegetation default files)
d. Specify spatially varied topographic values and other paremeters at each hierarchy: elevation, slope, aspect, isohyet, wetness index, rooting depth, impervious percent
e. Initialize state variables: assign a constant value or use a LAI map.

3. *Functions*

a. *daver/aver*: average values of map (integer/float) – Usage: function map
b. *spavg*: spherical average values of map (float) – Usage: function map1 map2 (only used for *aspect*)
c. *mode*: modal values of map (integer) – Every ID-related lines should use the 'mode' function
d. *area*: area – Usage: function
e. *deqn/eqn*: outputs of 'maps * multiply + add' (integer/float) – Usage: function multiply add map
f. *dvalue/value*: specified values (integer/float) – Usage: function value

*Should use the 'mode' or 'dvalue' function for all ID-related maps

4. *Edit the template file*

```
emacs templates/template514&
g.list rast # to see the name of rasters generated
*Template file is version-specific, so use the same version of template file with
  rhessys and cf
```

Change all hierarchical representations (lines starting with _; e.g. '_basin basinmap 1') into proper names 'basin_ws02 (world) - basin_ws02 (basin) – hillslope (hillslope) – patch (patch) – patch (stratum)'

```
*Be careful when you edit a template file. Do not corrupt the lines. Check every
  line.
```

Change every variable as follows (refer to the maps we made)

Variable name	Details	Name	Function
Z	elevation	dem10	aver
Latitude	latitude	35.05	value
Slope	slope	slope10	aver
Aspect	aspect	aspect10 slope10	spavg
Isohyet	long-term isohyet	1.0	value
east_horizon	east horizon angle	0.001 0 e_horizon_1000	eqn
west_horizon	west horizon angle	0.001 0 w_horizon_1000	eqn
soil default	soil default ID	3	dvalue
wetness index	topographic wetness index	0.01 0 topidx10_100	eqn
default ID	vegetation default ID	6	dvalue

5. *Generate the worldfile*

```
g2w –t templates/template514 –w worldfiles/worldfile_ws02_zero
more worldfiles/worldfile_ws02 # check that every variables are properly allocated
```

6. *LAIread*

A program which updates the worldfile carbon and nitrogen state variables based on leaf area index (LAI) maps (only for non-growth mode)

a. Input ascii files: hillslope ID, zone ID, patch ID, vegetation default ID (not stratum ID) and LAI maps with same prefix (e.g. *ws02.hill*, *ws02.patch, ws02.zone, ws02.vegid,* and *ws02.lai*) (GRASS or ARC ascii)
b. Allocation table

Generate input ascii files

```
r.mask –r # remove the current mask
r.in.arc input=../ARCdata/estlai_10m.asc output=lai10
d.rast lai10
d.vect watersheds fcolor=none
d.zoom # zoom into the WS02
d.what.rast # check the LAI value within the WS02
d.zoom –r # zoom out to the previous range
d.legend -ms lai10 # add the legend in the monitor with mouse
r.out.arc lai10 output=templates/ws02.lai
r.mask basin_ws02 maskcats=1 # mask using the basin map for WS02
r.out.arc input=hillslope output= templates/ws02.hill # Repeat this step for zone
  (same with patch), patch, and LAI
r.mapcalc 'vegid = 6' # vegetation ID for deciduous within the WS02 (from
  vegetation default files)
r.out.arc vegid output=templates/ws02.vegid
ls templates/ws02.* # check all input files are properly setup
```

Allocation table: indicate the biomass allocation and CN ratios for each vegetation compartments

```
more templates/allom.txt
*The first row of the text file is only one column – containing the number of
  entries.
*The subsequent rows will consist of ten columns for the following items: 1)
  vegetation id 2) sla 3) leaf:root 4) leaf:stem 5) stem:coarseroot 6)
  livestem:deadstem 7) cn_leaf 8) cn_froot 9) cn_livewood 10) cn_deadwood (You
  can find these information from vegetation default files except for cn_deadwood,
  usualy set as 333.33 for all veg types)
```

Command line options

```
lairead # check command line options
lairead –pre templates/ws02 –a –allom templates/allom.txt –old
  worldfiles/world_ws02_zero –redef worldfiles/world_ws02.Y2003M6D1H1
*Redefined worldfile should have specified extension from the date when satellite
  image was taken with the same prefix.
more worldfiles/world_ws02.Y2003M6D1H1 # type 'q' to quit
*This will give you a new redefine worldfile with the name you specified with the
  -redef option on the lairead command line. This has the value of -9999 for all
  state variables except those under stratum. Under the stratum level should be
  the new values based on the info you put into the allometric table.
```

External link: http://wiki.icess.ucsb.edu/rhessys/Incorporating_LANDSAT_data_into_GRASS

SESSION 5. CREATEFLOWPATH (*cf*)

cf generates a flowtable that describes connectivity between patches within a hillslope. This is only needed when the routing option is used.

1. *Structure of a flowtable*

http://wiki.icess.ucsb.edu/rhessys/Flow_Table
Detailed explanations of gamma parameter are available in Wigmosta et al. (1994)

2. *Additonal Input datasets*

Only additional map needed is a road map which can change flowpaths

Convert the road vector road map into the raster

```
g.list vect # check the list of vector
db.describe –c roads # check the name of columns
v.to.rast input=roads out=roads col=C_ROADS__1 # use the ‘C_ROADS__1’ column
r.stats –c roads # check the rough statistics of ‘roads’ raster
r.mapcalc ‘roads = roads > 0’ # logical true (1) or false (0), none is preserved
r.stats –c roads # recheck the stat
r.null roads null=0 # replace null values into zero – null can be a problem
r.stats –c roads # 1 for road, 0 for non-road
```

If there are no roads within the basin, just produce a raster zero (non-road).

```
r.mapcalc ‘roads = 0’
```

Running *cf*

```
r.mask basin_ws02 maskcats=1 # mask the region with basin map, which will
  significantly reduce the simulation time
cf9.1 output=ws02 template=templates/template514_ws02 stream=streams road=roads
  dem=dem10 slope=slope10 cellsize=10 # cellsize unit is meter; use the original
  elevation map
more flow/ws02.flow # check the flow table
cp ws02.flow ../flow/ # copy the flowtable under the flow directory.
```

External link: http://wiki.icess.ucsb.edu/rhessys/Generating_RHESSys_input_files#Flowtables

SESSION 6. RHESSys Simulations

1. *Directory structure*

calibration
clim
defs
flow
obs
output
src
tecfiles
templates
worldfiles

http://wiki.icess.ucsb.edu/rhessys/Directory_structure

2. *Daily climate data*

a. Base file: description of base station (elevation, screen height)
b. Three required (precipitation, maximum and minimum temperature)
c. Other optional (radiation – direct and diffuse, PAR, VPD or RH, soil temperature, daytime or nighttime average temperature, wind velocity, CO_2 concentration, daytime length, effective LAI etc.)

http://wiki.icess.ucsb.edu/rhessys/Climate_data

3. *Default files*

Default files are used to detail additional model parameters that describe climate, soil, vegetation, and land use factors that RHESSys uses. These will be provided to you. All are ID-related (should be same with what you specified in a worldfile).

a. Zone: climatic parameters
b. Soils: soil parameters (hydraulic, textural, and energy-related)
c. Landuse: human factors (irrigation, fertilizer, and septic)
d. Vegetation: Eco-physiologic parameters (White et al. 2000)

http://wiki.icess.ucsb.edu/rhessys/Default_files.

4. *Command line options*

Now we are finally ready to run RHESSys! This is done by issuing a simple command from the terminal window. Typically this is done by navigating to the model directory that contains the RHESSys executable and necessary sub-directories (clim, defs, flow, tecfiles, worldfiles, out).

a. -st start time in *year month day hour* format
b. -ed end time in *year month day hour* format
c. -b or –p specifies basin or patch scale output.
d. -t is the path to the tecfile
e. -w is the path to the worldfile
f. -r is the path to the flowtable used for explicit routing. If –r is not specified, topmodel is used.
g. -pre is the prefix used for output files
h. -s are sensitivity parameters for hydrology calibration and must have two numerical values after the –s flag. The first value (*sen1*) designates *m*, the decay of hydraulic conductivity with depth, the second required value (*sen2*) designates *Ksat*, hydraulic conductivity at the surface, and the third optional value (*sen3*) designates *soil depth*.
i. -gw specifies two parameter for groundwater calibration. The first value (*gw1*) is the multiplier of the *sat_to_gw_coeff* in the soil default file (representing the amount of water moving from the saturated store to the groundwater store and the reason why the *sat_to_gw_coeff* value in the soil default file should be 1.0). The second value (*gw2*) is the multiplier of the *gw_loss_coeff* in the hillslope default file (representing the amount of water moving from the groundwater store to the stream).

For a complete list of command line options please see:
http://wiki.icess.ucsb.edu/rhessys/Command_Line_Options

5. *TEC (Temporal Event Control) files*

TEC files control what is printed to output files by specifying what output you would like to see along with the start and times to print in Year Month Day Hour format.

TEC Event	Description
print_daily_on	Begin daily printing for regular or custom output
print_daily_off	End daily printing for regular or custom output
print_daily_growth_on	Begin daily growth printing for regular output only
print_daily_growth_off	End daily growth printing for regular output only
output_current_state	Outputs current state into a new worldfile

To create a tec file, you can use a text editor that can accommodate unix files such as TextWrangler or TextEdit or use vi from the terminal. (If using an editor other than vi, be careful not to have erroneous lines or carriage returns and save it in unix format)
Navigate to your rhessys/tec directory.

```
cd /home/rhessys/bootcamp/rhessys/tecfiles
vi
type 'i' for insert
1990 10 1 1 print_daily_on
1993 9 30 1 print_daily_off
Esc # get back to the command line
:w tec_daily # write a new file name 'tec_daily'
```

If you are interested in viewing spatial output, (done by changing command line options discussed below) it is often advisable to print a shorter period of output as all variables will be printed for each patch in the watershed at each specified timestep. You can always create output for longer periods of time and subset it with awk scripts. Let's create monthly patch output for 1993.

```
vi
type 'i'
1993 1 1 1 print_monthly_on
Esc
:w tec_month93
```

http://wiki.icess.ucsb.edu/rhessys/Temporal_Event_Control_files

6. *Using a redefined worldfile*

Use a *redefined worldfile* in order to incorporate the new redefined stratum values (e.g. from *lairead*) into an existing *worldfile* you want to change. Make sure both the *worldfile* you want to change and the *redefined worldfile* have the same name, but the *redefined worldfile* will also include a date extension on it, i.e.: world.example world.example.Y2000M10D2H1 (format for the date extension is Year, Month, Day, Hour)

```
ls worldfiles/
```

The *tecfile* should include a *redefine_world* event that occurs on the date that matches the date extension on the *redefinition worldfile*, as well as an *output_current_state* event that will generate a new *worldfile* that includes the newly incorporated info.

```
more tecfiles/tec_redef
```

Run RHESSys to redefine the 'world' using the *worldfile* you want to change and the *tecfile* with the redefinition command (*tec_redef*) in the RHESSys command line.

```
./rhessys5.14.6 –w worldfiles/world_ws02_zero –r flow/ws02.flow –t
  tecfiles/tec_redef –st 2003 1 1 1 –ed 2006 1 1 1 # no output option is
  specified
*End date should be the very next date in the simulation
```

```
*Just make sure the date extension on your redefine worldfile, the redefine_world
  and output_current_state event dates in the tecfile, and your RHESSys command
  line start and end dates correspond.
ls worldfiles/
```

As a result, you will get a new *worldfile* including a date extension that matches the *output_current_state* event date in the TEC file (the third day of simulation), i.e.: world.example.Y2000M10D3H1 - this is the *worldfile* that you use to start subsequent spin up's or simulations.

Just change the name of this new *worldfile.*

```
mv worldfiles/wodlf_ws02_zero.Y2005M12D31H24.state worldfiles/world_ws02
```

http://wiki.icess.ucsb.edu/rhessys/Performing_a_Redefine_World

7. *Main RHESSys simulations*

Example for basin scale:

```
./rhessys5.14.6 -st 1990 1 1 1 -ed 1993 10 1 1 -b -t tecfiles/tec_daily -w
  worldfiles/world_ws02 -r flow/ws02.flow -pre output/WS02 -s 12.0880 14.2677 -sv
  2.1529 83.7472 -gw 0.4108 0.0823 # Optimal parameter set from calibration
*'./' specifies to use the command in the current directory
```

Example for patch scale:

```
./rhessys5.14.6 -st 1990 1 1 1 -ed 1994 1 1 1 -p -t tecfiles/tec_month93 -w
  worldfiles/world_ws02 -r flow/ws02.flow -pre output/WS02 -s 12.0880 14.2677 -sv
  2.1529 83.7472 -gw 0.4108 0.0823
```

If time allows, you could alter the parameter values and try another simulation:

```
./rhessys5.14.6 -st 1990 1 1 1 -ed 1993 10 1 1 -b -t tecfiles/tec_daily -w
  worldfiles/world_ws02 -r flow/ws02.flow -pre out/WS02 -s 12.0880 14.2677 -sv
  2.1529 83.7472 -gw 0.4108 0.0823
```

8. *RHESSys output files and variables*

http://wiki.icess.ucsb.edu/rhessys/Output_Files

SESSION 7. Temporal Analysis of Model Results

1. *General R Info*

R is a freely available (GNU General Public License) programming language and software environment for statistical computing and graphics. The source and pre-compiled binaries are available at http://cran.r-project.org/. The core R functionality is extended by hundreds of user contributed software packages, which allow R to do everything from map making and spatial analysis to DNA sequencing. Task oriented sketches of R capabilities are available here: http://cran.r-project.org/web/views/. Help for individual functions in R is obtained by typing: ?function at the command line. To search the help files instead, use: ??search_string. A more user-friendly implementation of R search is available on the web: http://www.rseek.org/.

2. *Getting Started with R*

When you first start R (by typing *R* in the command) you will be in your home directory and presented with the R command line interface. Getting the current directory, listing, or switching directories is accomplished with getwd(), dir(), and setwd(), resp.

```
getwd()
setwd("bootcamp/")
dir()
```

R uses <- for assignment. Vector creation can be accomplished in a number of ways:
```
x<-1:6
y<-c(3,1,4,1,5,9)
y[3]
y[4]<-6
y
```

Vectors operations are straightforward and operations apply element-wise by default (in contrast to Matlab, for instance). Vectors are easy to subset (indexing starts at 1instead of 0), and may be the argument to a variety of R functions:
```
a<-seq(0, 1, 0.1)
b<-a^2
mean(a)
b[1:5]
b-a
summary(b)
```

Plotting and fitting linear models is simple in R. First, create some fake data: a sequence of independent variables, and a vector of “observations” which is a linear transform of the independent variable with a normally distributed random error added. Then fit a linear model using lm, plot the data, and add the fitted line:
```
x<-seq(1, 10)
y<-1.3*x+rnorm(10)
lm.1<-lm(y~x)
summary(lm.1)
plot(x, y, col="red")
abline(lm.1, lty=2)
```

The *linear model object* stored as lm.1 has attributes that can be listed using names() and accessed with the $ operator:
```
names(lm.1)
lm.1$coeff
```

Data frames are the fundamental data structure in R and are used to store collections of related variables. *Data frames* may be imported from a variety of data sources such as comma-separated value lists (CSV), or created from vectors. The most flexible function is read.table(), but read.csv() is a quick construct for CSV data. Individual variables in a *data frame* are accessed with the $ operator.
```
my.df<-data.frame(x, y)
my.df$NewVar<-my.df$x^2
summary(my.df)
```

Use read.csv() to import a file of observed streamflow, and convert the included date variable to an R Date object.
```
qobs<-read.csv("obs/ws02Qmm.csv")
names(qobs)<-c("Date","Q")
qobs[1:2,]
qobs$NewDate<-as.Date(qobs$Date, format="%m/%d/%Y")
```

Some of the lines in the observed streamflow have an impossible date value and so as.Date() fails to convert these, leaving a value of NA:
```
qobs[is.na(qobs$NewDate), ]
```

We can eliminate the offending records by subsetting the entire data frame:
```
qobs<-qobs[!is.na(qobs$NewDate), ]
```

3. *Importing RHESSys Output and Plotting*

RHESSys simulation output, such as *ws02_basin.daily*, are stored as lines with individual variables separated by spaces. The first line contains the variable names, but R doesn't like something about the way it is formatted. In order to import these results into R as a data frame, we will use read.table() and skip the first line. Then we'll read the first line as a string, split it up, and then assign it as the names of the data frame:

```
basin.daily<-read.table("output/ws02_basin.daily", sep=" ", header=F, skip=1)
basin.names<-readLines("output/ws02_basin.daily", 1)
names(basin.daily)<-unlist(strsplit(basin.names, split=" "))
```

Since these operations are likely to be used frequently, it is best to create a new function in R for this purpose:

```
importRHESSYS<-function(filename){
tmp<-read.table(filename, sep=" ", header=F, skip=1)
tmp_names<-readLines(filename, 1)
names(tmp)<-unlist(strsplit(tmp_names, split=" "))
return(tmp)
}
```

Using this function, it is easy to import any RHESSys output:

```
basin.daily<-importRHESSYS("output/ws02_basin.daily")
names(basin.daily)
```

Calculate the total simulated streamflow for the year 1992, and the mean simulated daily evaporation for the year 1992 (notice how the data frame variables are subset using a logical operation):

```
sum(basin.daily$streamflow[basin.daily$year==1992])
mean(basin.daily$evap[basin.daily$year==1992])
```

Data frame variables are easily plotted using the plot() function. If no "x" variable is given the variable is plotted as a series (i.e., against its index in the data frame).

```
plot(basin.daily$trans, type="l", ylab="Transpiration (mm)")
```

Nicer plotting can be achieved by creating a variable of type Date, and specifying this as the "x" variable to plot(): First, we create the "Date" variable in the imported data frame using as.Date() which expects a string with the form "YYYY-MM-DD" that we construct out of the "year", "month", and "day" variables using paste(). Next, we plot the simulated daily saturation deficit depth time series.

```
basin.daily$Date<-as.Date(paste(basin.daily$year, basin.daily$month,
  basin.daily$day, sep="-"))
plot(basin.daily$Date, basin.daily$sat_def_z, ylab="sat_def_z", xlab="Date",
  type="l", pch=16)
```

Multiple figures may be plotted in the same graphics device using the mfrow argument to par(). layout() is a better, but more complicated way to accomplish this. Margins are controlled using the mar argument to par():

```
par(mfrow=c(2,1), mar=c(2,4,1,1))
plot(basin.daily$Date, basin.daily$evap,ylab="Evap (mm)", type="l", col=3)
par(mar=c(3,4,0,1))
plot(basin.daily$Date, basin.daily$trans,ylab="Trans (mm)", type="l", col=4)
```

Plots may be combined in the same graphics window, either using a common y-axis, or separate ones. First, plot daily evaporation and transpiration for a single calendar year with the same y-axis, and then add a legend to the plot:

```
plot(basin.daily$Date[basin.daily$year==1992],
  basin.daily$evap[basin.daily$year==1992], type="l", ylab="Vapor Flux (mm)",
  xlab="")
points(basin.daily$Date[basin.daily$year==1992],
  basin.daily$trans[basin.daily$year==1992], type="l", col=2, lty=2)
legend("topleft", legend=c("evap", "trans"), col=c(1, 2), lty=c(1,2))
```

Next, baseflow and saturation deficit for a calendar year will be plotted in the same window, but with unique y-axes.

```
par(mar=c(2,4,1,4)) #get a little extra right margin
plot(basin.daily$Date[basin.daily$year==1992],
  basin.daily$baseflow[basin.daily$year==1992], type="l", col="darkgreen",
  ylab="Baseflow (mm)", xlab="")
par(new=T) #new plot, same window
plot(basin.daily$Date[basin.daily$year==1992],
  basin.daily$sat_def_z[basin.daily$year==1992], type="l", col="firebrick",
  xaxt="n", yaxt="n", xlab="", ylab="", lwd=1.5, lty=2)
axis(4) #add second y-axis
mtext("sat_def_z (mm)", 4, 2) #add margin text
legend("topleft", legend=c("Baseflow", "Sat. Deficit"), col=c("darkgreen",
  "firebrick"), lwd=c(1, 1.5), bty="n", lty=c(1, 2))
```

Plot simulated vs. observed streamflow for the 1991-1993 water years with a logarithmic scale y-axis, and add a legend:

```
start.date<-as.Date("1990-10-01")
end.date<-as.Date("1993-09-30")
plot(seq.Date(start.date, end.date, by=1), qobs$Q[qobs$NewDate>=start.date &
  qobs$NewDate<=end.date], type="l", log="y", ylab="Q (mm)", lwd=3, col="grey75")

# Add the simulated series
points(seq.Date(start.date, end.date, by=1),
  basin.daily$streamflow[basin.daily$Date>=start.date &
  basin.daily$Date<=end.date], type="l", col=2)
legend("topleft", legend=c("Obs", "Sim"), col=c("grey75", 2), lwd=c(3, 1))
# Maximum log NSE = 0.695
```

4. Better time series operations: the zoo package

The zoo package handles time series with arbitrary and possibly irregular intervals. Using zoo objects makes many operations, such as aggregation to coarser temporal intervals (monthly streamflow, for instance), much easier. Additionally, subsetting time series is more straightforward and easier to accomplish with zoo objects than regular R data frames.

First, the zoo package must be installed and loaded:

```
install.packages("zoo")
library(zoo)
```

Next, convert the observed streamflow series to a zoo object using the date variable in that data frame:

```
q.zoo<-zoo(qobs$Q, order.by=qobs$NewDate)
```

Note, zoo can convert entire data frames at once, but relies on as.matrix(), and therefore cannot correctly convert data frames with mixed variable types. If we convert the entire basin.daily data frame, all variables will have the type "character" (strings are character vectors in R) and must be converted using as.numeric() in order to perform mathematical operations or plot efficiently.

```
basin.daily.zoo<-zoo(basin.daily, order.by=basin.daily$Date)
typeof(basin.daily.zoo$psn)
rm(basin.daily.zoo) # remove the object from the workspace
```

Subsetting is accomplished using the zoo function window() with start and end specified as Date objects. For example, to plot the simulated streamflow during the 1991-1993 water year:

```
plot(window(q.zoo, start=as.Date("1990-10-01"), end=as.Date("1993-09-30")),
  ylab="Q (mm)", xlab="")
```

A vector of the time series values can be obtained using coredata(), and the time series indexes can be obtained with index():

```
coredata(q.zoo)[1:10]
index(q.zoo)[1:10]
```

zoo has a number of gap filling algorithms for missing data. For example, if 1000 randomly selected values in the streamflow record are replaced with NA, na.approx may be used to fill them with interpolated values:

```
tmp<-q.zoo
tmp[sample(1:length(tmp), 1000)]<-NA
tmp.filled<-na.approx(tmp)
```

The function aggregate() has a zoo method, which applies a function, such as sum or mean, to a time series based on a vector of shared indices. This can be used to obtain a monthly time series of observed streamflow by summing the daily values in a particular month and year. First, a vector must be created to define the aggregation groups. This vector is easily created using the strtime() function on the Date object, then the monthly aggregated streamflow is plotted as a barplot:

```
#all dates converted to first of month: YYYY-MM-1
q.month<-as.Date(paste(strftime(index(q.zoo), "%Y"), strftime(index(q.zoo), "%m"),
   "1", sep="-"))
q.monthly<-aggregate(q.zoo, by=q.month, FUN=sum)

barplot(window(q.monthly, start=start.date, end=end.date))
box()
```

Finally, use the zoo package and layout() to make a nice hydrograph for the 1991-1993 water year:

```
#convert the precip variable to a zoo object
precip.zoo<-zoo(basin.daily$precip, order.by=basin.daily$Date)
start<-as.Date("1990-10-01"); end.date<-as.Date("1993-09-30")

#find max and min of precip so y-axis can be reversed with ylim
max.p<-max(window(precip.zoo, start=start.date))
min.p<-min(window(precip.zoo, start=start.date))

layout(matrix(1:2, nrow=2, byrow=T), heights=c(0.35, 0.65))
par(mar=c(0,4,1,1))
plot(window(precip.zoo, start=start.date, end=end.date), type="h", ylim=c(max.p,
   min.p), col=4, ylab="P (mm)", xlab="", xaxt="n", lwd=2)

par(mar=c(3,4,0,1))
plot(window(q.zoo, start=start.date, end=end.date), type="l", col="grey55", lwd=2,
   log="y", xlab="Date", ylab="Q (mm)")

#add a fancy annotation to the plot
q_sum<-sum(window(q.zoo, start=start.date, end=end.date))
padding<-c((par()$usr[2]-par()$usr[1])/100, (par()$usr[4]-par()$usr[3])/100)
text(par()$usr[1]+padding[1], 10^(par()$usr[4]-padding[2]),
   labels=substitute(Sigma==subvar, list(subvar=round(q_sum, 2))), cex=1.5,
   adj=c(0,1))
```

5. *Calculating Nash-Sutcliffe Efficiency*

NSE is calculated as one minus the ratio of explained to observed variance (the coefficient of determination). An R function to calculate NSE may be written:

```
NSE<-function(observed, modeled){
mean.obs<-mean(observed)
NSE<-1-sum((observed-modeled)^2)/sum((observed-mean.obs)^2)
return(NSE)
}
```

The function can be used to calculate the NSE for the 1991-1993 water year:

```
mod_q.zoo<-zoo(basin.daily$streamflow, order.by=basin.daily$Date)
obs<- window(q.zoo, start=start.date, end=end.date)
mod<- window(mod_q.zoo, start=start.date, end=end.date)
NSE(obs, mod)
```

NSE may also be calculated using the natural logarithm of the observed and modeled streamflows. The NSE function may be modified to return both NSE and NSE_log:

```
NSE<-function(observed, modeled){
mean.obs<-mean(observed)
NSE<-1-sum((observed-modeled)^2)/sum((observed-mean.obs)^2)
  modeled.log<-log(modeled); observed.log<-log(observed)
mean.obs.log<-mean(observed.log)
NSElog<-1-sum((observed.log-modeled.log)^2)/sum((observed.log-mean.obs.log)^2)
return(c(NSE, NSElog))
}
NSE(obs, mod)
```

Note: zeros in the observed or modeled streamflow will result in values of negative infinity for the logarithm, and above implementations will return a value of NaN for NSE_log. Zero values must either be ignored when calculating NSE_log, or a value must be substituted in their place. This functionality should probably be implemented in the above function, however both cases could be handled without modification:

```
obs.zeros<-obs
obs.zeros[sample(1:365,10)]<-0 #randomly switch 10 values to 0

#calculate NSE_log by ignoring zero values
NSE(obs[!is.infinite(log(obs.zeros))], mod[!is.infinite(log(obs.zeros))])

#substitute a very small value for streamflow instead of 0
obs.tmp<-obs.zeros
obs.tmp[is.infinite(log(obs.tmp))]<-0.00001
NSE(obs, mod)
```

6. *Working with cluster_calibrator* output*

The cluster_calibrator suite automates the Monte Carlo calibration scheme on a cluster computing facility using Load Sharing Facility software, and calculates model efficiencies. SQLite databases are used to keep track of model runs and their associated parameter sets and efficiencies. The R packages DBI and RSQLite may be used to import these types of databases as R data frames.

First, install and load the required packages:

```
install.packages(“DBI”)
install.packages(“RSQLite”)
library(DBI)
library(RSQLite)
```

Next, create an instance of a database driver, a connection to a database file, and then import the “run” table, which contains model parameter and efficiency data, from the db connection into an R data frame:

```
m<-dbDriver(“SQLite”)
con<-dbConnect(m, dbname=”calibration.db”)
calib<-dbReadTable(con, "run")
```

Sort the calibration results by descending Nash-Sutcliffe Efficiency (NSE) and NSElog, and display the model parameters (columns 6:12) for the top 5 entries:

```
calib[order(-calib$nse), ][1:5, 6:12]
calib[order(-calib$nse_log), ][1:5, 6:12]
```

"Dotty plots" are used to visualize the patterns in NSE and the calibration parameters. Typically, 100's to 1000's of model iterations are necessary to sample the parameter space adequately enough to observe patterns in Dotty plots.

```
nf<-layout(matrix(1:6, nrow=2, byrow=T))
par(mar=c(4, 4, 1, 1))
for(i in c(6:7, 9:12))
  plot(calib[, i], calib$nse, xlab=names(calib)[i], ylab="NSE", pch=16, cex=0.5)
```

SESSION 8. Visualization of Model Results

Output variables can be viewed spatially by replacing map ID numbers with values for a particular output variable. This requires using a map that has multiple ID's so that spatial variation exists, such as the hillslope or patch maps. First, you must decide what variable you want to look at spatially, and what the column number of that data is in the output file.

```
head –n 1 output/ws02_patch.monthly # see header of monthly patch output
```

Now you can use the AWK program to extract monthly ET (evapotranspiration) and PSN (net photosynthesis) columns in the format necessary to be read into GRASS. Count the column numbers of these variables (10 and 11). AWK is a pattern matching program, so you must give the AWK program some conditional information so it is able to extract what you want from the patch monthly output file.

```
more templates/extmonthly.awk
```

You need to specify *mth* (month), *year* (year), *col* (column number for the target variable), and *mult* (multiplier) variables to execute AWK program.

```
awk –f templates/extmonthly.awk mth=6 year=1993 col=10 mult=1 <
  output/ws02_patch.monthly > output/ws02.et
awk –f templates/extmonthly.awk mth=6 year=1993 col=11 mult=1 <
  output/ws02_patch.monthly > output/ws02.psn
*'<' indicates the text file you are inputting, while '>' indicates the redirect
  the results into the specified text file.
more output/ws02.et
*type 'q' to exit
```

Make sure that mask for WS02 is on

```
d.erase
r.reclass input=patch output=et_jun93 < output/ws02.et
d.rast et_jun93 # zoom in if necessary using d.zoom
d.legend –ms et_jun93
d.out.file out=et_jun93 format=png # export display monitor into png graphic file
```

Open *Home Folder* in the left section, and go to working directory. Check the figure.
Do the same thing to *psn* variable.

If you make a script to export for each month, we can make some nice animation of monthly ET and PSN from the model.

External link: http://wiki.icess.ucsb.edu/rhessys/Visualizing_Spatial_Output

SESSION 9. Spin-up and Calibration

1. *Spin up process*

a. Growth mode vs. non-growth mode
b. Warm spin-up vs. cold spin-up

2. *Calibration process*

a. Command line options for calibration of key parameters (multiplier options)
‘-s *m Ksat0_lateral* -sv *m_v Ksat0_vertical* -gw *sat_to_gw_coeff gw_loss_coeff*’
b. Nash-Sutcliffe efficiency (normal or log)
c. Parallel programming in Linux Cluster
Cluster_calibrator tools

External Link: http://wiki.icess.ucsb.edu/rhessys/Calibrating_and_running_RHESSys

Other Useful Resources

National Land-Cover Datasets (NLCD): http://www.epa.gov/mrlc/nlcd-2001.html
Soil Survey Geographic (SSURGO) Database: http://soils.usda.gov/survey/geography/ssurgo/
National Elevation Dataset (NED): http://ned.usgs.gov/

References

Band, L. E., P. Patterson, R. Nemani, and S. W. Running (1993), Forest ecosystem processes at the watershed scale: Incorporating hillslope hydrology, Agric. For. Meteorol., 63, 93 – 126.
Beven, K., and M. Kirkby (1979), A physically based variable contributing area model of basin hydrology, Hydrol. Sci. Bull., 24, 43– 69.
Running, S. W., R. R. Nemani, and R. D. Hungerford (1987), Extrapolation of synoptic meteorological data in mountainous terrain and its use for simulating forest evapotranspiration and photosynthesis, Can. J. For. Res., 17, 472– 483.
Running, S. W., and E. R. Hunt (1993), Generalization of a forest ecosystem process model for other biomes, BIOME-BCG, and an application for global-scale models, in Scaling Physiological Processes: Leaf to Globe, edited by J. R. Ehleringer and C. B. Field, pp. 141– 158, Academic, San Diego, Calif.
Tague, C. L., and L. E. Band (2004), RHESSys: Regional Hydro-Ecologic Simulation System—An object-oriented approach to spatially distributed modeling of carbon, water, and nutrient cycling, Earth Interact., 8, 1 –42.
White, M. A., P. E. Thornton, S. W. Running, and R. R. Nemani (2000), Parameterization and sensitivity analysis of the BIOME-BGC terrestrial ecosystem model: Net primary production controls, Earth Interact., 4, 1 – 85.
Wigmosta, M. S., L. W. Vail, and D. P. Lettenmaier (1994), A distributed hydrology-vegetation model for complex terrain, Water Resour. Res., 30 1665-1679.