Session 1: Introduction

1. Download the tutorial.tar.gz file from the dhsvm ftp site

https://dhsvm.pnnl.gov/code.stm

Select Source code & Test Site Data for DHSVM 3.1 and save it to your working directory.

2. Unzip and untar the file

>>> tar -xzvf tutorial.3.1.1.tar.gz

Note: Before you start, note that you can choose between two types of input/output files: binary or NetCDF. The model is programmed so that binary input files generate binary output files, and NetCDF input files generate NetCDF output files. ***To prepare binary input files, read the follow Session 2, 3 and 5, but skip Session 4. To prepare NetCDF input files, skip Session 3, and follow through Session 2, 4 and 5.***

Session 2: Generate Inputs

1. Run ArcMap/ArcInfo and view raster/grid files

ArcInfo only runs on the PC computer (let's say the machine name is 'plane'), so here are the system-specific commands for running arc/info.

>>> cd tutorial3.1/input
>>> arc

2. Export SOIL, SOIL DEPTH, VEGETATION, MASK and DEM files as ascii grids

>>> Arc: asciigrid soil.asc soil

Take the same steps to export the SOIL DEPTH, VEG, MASK and DEM files

View raster/grid files: soil (soil class), veg (vegetation class/land cover type), dem & mask.

3. Create the stream, soil, and road inputs

Compile required programs (fixroads.c and AddAat2.java). The JAVA program (*jdk1.7.0_05)* is required to run the following commands.

>>> cd /tutorial3.1/programs/
>>> gcc fixroads.c -lm -o fixroads
>>> javac AddAat2.java Aat.java AatError.java

Upon successful run of this script, 3 class files, including **AddAat2.class**, **Aat.class** and **AatError.class**, will appear in the "programs" directory.

The next steps are performed using a set of arc commands and aml scripts via Arc/Info on a PC system.

Create a folder under tutorial3.1\ called arcinfo (eg. \tutorial3.1\arcinfo\). Copy all the necessary files for creating the stream, soil, and road inputs in Arc/Info. The necessary input files are the DEM and MASK of the site. Create a sub-folder in your workspace called "program", and copied the three java .class files that you just compiled into this sub-directory (eg. C:\workspace\tutorial3.1\arcinfo\program\AddAat2.class).

```
>>> cd ../input/arcinfo
>>> plane% arc
```

Set the workspace to your working directory, which should be the arcinfo folder you created.

```
>>> Arc: &workspace C:\workspace\tutorial3.1\arcinfo\
```

Set the path for aml scripts:

```
>>> Arc: &amlpath C:\workspace\tutorial3.1\amlscripts\
```

Note: Open the **createstreamnetwork.aml** in the amscripts folder (with Notepad or WordPad). Look for the call to the AddAat2 java program. Make sure that the path to the class file is correct. You should have AddAat2.class file in the "program" directory of the workspace, therefore the line specifying the class path in the createstreamnetwork.aml must look like:

```
>>> &sys java -classpath program\ AddAat2 %streamnet%
```

4. Run the aml scripts

Before you start, it is very important to note that, if any error occurs before the aml is done, you **MUST** RE-RUN the aml script with THE ORIGINAL INPUT FILES. Always keep a copy of the original arcinfo directory. Within "arcinfo", you should have original input GRID files and a "program" folder including 3 class files. Once any error occurs, read the error statement to find out the potential cause. Then delete the content of the "arcinfo" folder you worked on, copy the arcinfo-Copy folder into the specified workspace, and rerun the script.

Usage:

STREAMNETWORK <dem> <wshed> <soildepth> <stream network> <MOUTH|MASK> {source area} {min depth} {max depth}

```
>>> Arc: &run createstreamnetwork dem mask soild streams MASK 100000 0.76 1.5
```

Note: all grids should have same domain and resolution.

dem: a grid of basin elevations with all sinks filled in. Make sure that dem is in floating point and the regions outside of the watershed are defied as NoData instead of 0. Otherwise you will get a stream

network for the whole raster area. You can accomplish this by using the 'Set Null' function in raster calculator in ArcMap. The format should look like:

```
>>> SetNull("mask"!=1, "dem")
```

wshed: EITHER a basin mask file, or a grid designating the location of the basin mouth. If it's a mask file, indicate MASK in the 5th input argument; if it's a mouth file, indicate MOUTH in the 5th input argument (as indicated by the keyword MOUTH | MASK). If MOUTH is specified, the mask file will be created. Likewise, the mask must be defined as inbasin=1, outside basin=NoData so the outputs share the same boundary as the basin or watershed.

soild: grid of soil depth. This is an output of the createstreamnetwork.aml script, so specify the file name you'd like the output to have in this argument. Make sure there isn't a file with the same name in the working directory already because Arc might not be able to overwrite the existing file, which generates errors in the script. It will be created as a function of cumulative drainage area and slope.

streams: an arc coverage of stream locations. is an output of the createstreamnetwork.aml script, so specify the file name you'd like the output to have in this argument. Make sure there isn't a file with the same name in the working directory already because Arc might not be able to overwrite the existing file, which generates errors in the script. It will be created using the specified contributing threshold area (min source area) and the DEM file provided.

The last three values represent the minimum contributing area before a channel begins, the minimum soil depth, and the maximum soil depth. Input these values based on your knowledge of the watershed. The smaller the minimum contributing area is, the more stream segments will be created by the program (ie. higher stream density). Check the river coverage file output in ArcMap to see if the stream density is desirable. The thicker the soil depth is the slower the peak response to rain events tends to be. All three values must be provided.

Run the script. You will be prompted to see if you want to continue when the script is nearly completed. Type **`y`** and **`enter`**.

This step creates the raster file soild, text files stream.network.dat & stream.map.dat that are needed by DHSVM, and the stream coverage file.

5. Create road inputs

If there are road layers in the study site, take this step to create the files road.network.dat and road.map.dat that are needed by DHSVM.

Usage:
ROADNETWORK <dem> <soildepth> <road network>

>>> Arc: &run createroadnetwork dem soild roads

You will be prompted to see if you want to continue. Type `y` and `enter`.

6. Designate stream save indicator

We want to save the output (flow, etc) at the final stream segment (the outlet/mouth) within the stream network. This is designated within the file /arcinfo/stream.network.dat with a value of -1 in the sixth column (ie. the stream is not connected to any downstream segment). Routing results for this stream segment will be placed in the stream output file if the keyword SAVE appears in the last (seventh) column. If there are more than one segments with a value of -1 in the sixth column, check in ArcMap to see where these segments are and save the most relevant segment. Reducing the minimum contributing area and rerun the "createstreamnetwork" script can also help to reducing the number of outlets in the stream network created.

Within xemacs, open stream.network.dat and make the required changes.

>>> xemacs stream.network.dat

Add **SAVE "SPRINGBROOKCREEK"** following the last column of stream segment that you want to save(in the seventh column). Enter the desired name for this outlet in the quotation marks. To properly run the model, replace the value of -1 in the sixth column in the stream.network.dat file with 0 (change this for the outlet only). Save the file and exit xemacs.

7. Create road and stream class files

These are ascii files that contain a look-up table of channel hydraulic properties for each road or stream class. For an example of a stream class file, look into adjust.classfile in the input folder. The following columns are necessary for each channel class:
<Channel Class> <Hydraulic Width (m)> <Hydraulic Depth or Bank Height (m)> <Manning's friction coefficient> <Max Infiltration (m/s)>

The channel hydraulic width and depth are currently hard-wired in the createroadnetwork and createstreamnetwork scripts, as follows:

Roads:

#Class	Width	Depth
1	0.5	0.5
2	0.5	0.5
3	0.25	0.25
4	0.25	0.25
5	0.25	0.25

Streams:

#Class	Width	Depth	Manning's n	Max Infiltration
1	0.03	0.5	0.0875	0.0001
2	0.03	1.0	0.0875	0.0001
3	0.03	2.0	0.0875	0.0001
4	0.03	3.0	0.0875	0.0001
5	0.03	4.0	0.0875	0.0001
6	0.03	4.5	0.0875	0.0001
7	0.05	0.5	0.0875	0.0001
8	0.05	1.0	0.0875	0.0001
9	0.05	2.0	0.0875	0.0001

Edit these tables (and save as road.class.dat and stream.class.dat) after adding the friction and infiltration columns. Fill in the path to the class file under the section of STREAM NETWORK and ROAD NETWORK in the configuration input file.

The maximum infiltration rate is **NOT** used for the stream class file, but it MUST be specified as a place holder. Manning's roughness can vary from 0.025 to 0.15 for natural channels. Infiltration into the roads may be assumed to be 0 m/s for a starting point (i.e. 100% impervious).

Session 3: Generate and convert input files to BINARY

1. Convert NODATA from -9999 to 0 in the mask file.

```
>>> Arc: grid
>>> Grid: mask_zero = con(isnull(mask), 0, mask)
>>> Grid: q
```

2. Convert grids to binary

1) Remove the headers from the ascii files:

Use the **tail** command to output all lines from the 7th line in the input file. This command generates a new output file.

```
>>> cd tutorial3.1/input
>>> tail –n +7 dem.txt > newdem.txt
```

Generate text files with no header for the DEM, VEG, MASK, SOIL and SOIL DEPTH files.

Use the **head** command to keep a copy of the header file.

```
>>> head –6 mask.txt > headerfile.txt
>>> head –6 mask.txt
```

2) Ascii → Binary conversion: compile the program myconvert.c :

Usage:

*./**myconvert** source_format target_format source_file target_file number_of_rows number_of_column*)

```
>>> programs/myconvert asc float input/newdem.txt input/ dem.bin 213 164
```

Repeat for each of the map files, changing the target format as follows:

Mask, soil, veg = char dem, soil depth = float

3. Create surface routing file for the urban module

1) Run the program **'find_nearest_channel_bin.c'**

Usage:

>>> ./find_nearest_channel_bin nrows ncols binary_flowd_file binary_mask_file stream_map_file n_header_map_file

Example:

>>> programs/find_nearest_channel_bin 213 164 input/flowdir.bin input/mask.bin springbrook.stream.map output/surface.routing.txt 9

where:

flowd_file is a char binary flow direction file in the same format as the DHSVM mask file. Make sure that the flowd_file is free of sinks, etc, and flowdirection is assumed to be from ARC-INFO, i.e. 1 to 128. If needed, use ArcMap to generate the flow direction file, and then use Arc/Info gridascii command to convert raster flow direction file to ascii, remove header, and use myconvert to change it into a char binary format.

binary mask_file and ***stream_map file*** are the DHSVM specific input files for mask and stream_map file, respectively.

output_file are the output surface routing file.

n_header_map_file are the number of header lines in the stream map file, i.e. lines starting with # . Enter 0 if there are no header lines. Caution: make sure you are referring to the map file not the network file.

2) Fill in the path to the surface.routing.txt file under the section of VEGETATION/ IMPERVIOUS SURFACE ROUTING FILE in the configuration input file.

4. Create model states

1) Compile the MakeModelStateBin.c:

>>> cd programs

>>> make -f Makefile.ModelState.Bin

2) Run the shell script

>>> ./MakeModelStateBin ../modelstate/InitialState.txt

This step creates the Interception, Snow and Soil state files for the date specified in InitialState.txt.

Note that input InitialState.txt includes the initial Interception, Snow and Soil state for each grid cell. In the sample input file set up for Springbrook Creek, it is assumed that no interception storage, no snow for 210 days and 35% volumetric soil moisture in all layers.

The date for the model state should be the same as the start date of the model. In this example, it is 10/01/1995 – 00.

The 'initialstate' file MUST contain the following information:

1. path for output file (change it if not ../modelstate)
2. date for the model state, in mm/dd/yyyy-hh
3. number of rows (ny) and number of columns (nx)
4. maximum number of vegetation layers
5. rain interception in m for each vegetation layer
6. snow interception in m for top vegetation layer
7. snow cover mask
8. number of days since last snow fall
9. snow water equivalent in m
10. liquid water content in m of bottom layer of snowpack
11. temperature in C of bottom layer of snow pack
12. liquid water content in m of top layer of snowpack
13. temperature in C of top layer of snow pack
14. cold content of snow pack
15. maximum number of root zone layers
16. volumetric soil moisture content for each layer (including the layer below the lowest root zone layer)
17. temperature in C at soil surface
18. soil temperature in C for each root zone layer
19. ground heat storage
20. runoff

5. Create initial channel state files

If necessary, make the script MakeChannelState.scr executable:

```
>>> chmod 755 MakeChannelState.scr
```

Usage:

```
MakeChannelState.scr  <StreamNetworkFile> <InitialDepth> <Output Date String: MM.DD.YYYY.hh.mm.ss>
```

To run the script (calculates volume of water in each channel segment, given an assumed uniform initial depth in meters):

```
>>> ./MakeChannelState.scr ../input/springbrook.stream.network 0.25 10.01.1995.00.00.00
```

6. Create shadow and skyview files

1) If necessary, make the script '**monthly_shadow_bin.scr'** executable:

```
>>>  chmod 755 monthly_shadow_bin.scr
```

2) Edit '**monthly_shadow_bin.scr',** change the following parameters:

- cell size (**cell**)
- Center latitude (**lat**) and longitude (**lon**)
- number of rows (**rows**) and number of columns (**cols**)
- binary DEM file
- output path #no trailing slash

7. Run the script

>>> ./monthly_shadow_bin.scr

The output files include monthly shadow file (e.g. Shadow.??.bin) and skyview file (?? refers to any month of the year).

Session 4: Generate and convert input files to NetCDF

1. Convert NODATA from -9999 to 0 in the mask file.

>>> Arc: grid
>>> Grid: mask_zero = con(isnull(mask), 0, mask)
>>> Grid: q

2. Convert ascii files to 3D NetCDF files

There are several ways to accomplish this task using various tools, including but not limited to ArcMap and GDAL. These two methods are validated with DHSVM and below are the instructions. Follow the instructions to export 5 input files in .nc format: dem, mask, veg (land cover), soil depth, and soil class. The flow direction file in .nc format will be required to compute the nearest_channel for the urban module.

- Using ArcMap

The link below will direct you to the Esri short tutorial about how to convert raster files to netcdf files in ArcMap.
http://webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=Exporting_a_raster_to_netCDF

- Using GDAL

Only one command line is needed for conversion:
>>> gdal_translate -of netCDF <ascii input file> <netCDF file>

Note that <ascii input file> should contains the header.

With the .nc files converted from .asc files, there are a few more steps to take in order to generate a 3D netcdf file with the DHSVM required variable type & name.

- **Convert 2-D .nc file to 3-D .nc**

The .nc file converted from .asc is generally 2-dimensional (lat and long, or y and x). DHSVM, however, requires 3-dimensional .nc input (the additional dimension is time) so we can realize time series visualization of output maps. To do this, add the time dimension to .nc file (i.e. convert 2-D .nc file to 3-D .nc) using the following command:

>>> ncecat -O -u time <in.nc> <out.nc>

- **Convert the data type.**

Dem and SoilDepth must be float and the others must be unsigned char (byte).

>>> ncap2 -s 'var_name=byte(var_name)' <in.nc> <out.nc>

>>> ncap2 -s 'var_name=float(var_name)' <dem.nc> <dem_out.nc>

For example, if <in.nc> is originated from the ascii file produced in ArcMap, the typical var_name is Band1.

If you are not sure of the original variable type, use the following command to check the header file of .nc file.

>>> ncdump –h <in.nc>

- **Change the variable names for DHSVM**

Lastly, change the variable name. Names must be changed because of model internal settings. For example, the DEM .nc file must be named "Basin.DEM". And the command to do that is:

>>> ncrename -v old_var_name,Basin.DEM <in.nc>

The required variable type and names are listed as follows:

NetCDF file	Type	Name
dem.nc	Float	Basin.DEM
mask.nc	Byte	Basin.Mask
soil.nc	Byte	Soil.Type
soild.nc	Float	Soil.Depth
veg.nc	Byte	Veg.Type
flowdir.nc	Byte	Flow.Dir

>>> ncrename -v oldvarname,newvarname <in.nc>

3. Create surface routing file for the urban module

1) Compile the program '***find_nearest_channel_netcdf.c***':

>>> cd ../programs

>>> make -f Makefile.NearestChannel.NetCDF

2) If necessary, make the script '***create_nearest_channel_netcdf.scr'*** executable:

>>> chmod 755 create_nearest_channel_netcdf.scr

3) Edit ***create_nearest_channel_netcdf.scr,*** change the following parameters:

- number of rows (**rows**) and number of columns (**cols**)
- cell size (**cell**)
- origin at the upper left corner (**XOrig** & **YOrig**)
- flow direction file in netcdf format
- mask file in netcdf format
- stream map file
- output file
- number of header line in stream map file that will be skipped in file reading

4) Run the script:

```
>>>  ./create_nearest_channel_netcdf.scr
```

5) Fill in the path to the surface.routing.txt file under the section of VEGETATION/ IMPERVIOUS SURFACE ROUTING FILE in the configuration input file.

4. Create model states

1) Compile the MakeModelStateNetCDF.c:

```
>>> make -f Makefile.ModelState.NetCDF
```

2) Edit **create_modelstate_netcdf.scr**, change the following parameters:

- cell size (**cell**)
- origin at the upper left corner (**XOrig** & **YOrig**)
- initial state file (see below for the file description)

```
>>> ./create_modelstate_netcdf.scr
```

This step creates the Interception, Snow and Soil state files for the date specified in InitialState.txt.
Note that input InitialState.txt includes the initial Interception, Snow and Soil state for each grid cell. In the sample input file set up for Springbrook Creek, its assumed that no interception storage, no snow for 210 days and 35% volumetric soil moisture in all layers.

The 'initialstate' file MUST contain the following information:

21. path for output file (change it if not ../modelstate)
22. date for the model state, in mm/dd/yyyy-hh
23. number of rows (ny) and number of columns (nx)
24. maximum number of vegetation layers
25. rain interception in m for each vegetation layer
26. snow interception in m for top vegetation layer
27. snow cover mask
28. number of days since last snow fall
29. snow water equivalent in m
30. liquid water content in m of bottom layer of snowpack
31. temperature in C of bottom layer of snow pack

32. liquid water content in m of top layer of snowpack
33. temperature in C of top layer of snow pack
34. cold content of snow pack
35. maximum number of root zone layers
36. volumetric soil moisture content for each layer (including the layer below the lowest root zone layer)
37. temperature in C at soil surface
38. soil temperature in C for each root zone layer
39. ground heat storage
40. runoff

5. Create initial channel state files

Usage:
MakeChannelState.scr <StreamNetworkFile> <InitialDepth> <Output Date String: MM.DD.YYYY.hh.mm.ss>

To run the script (calculates volume of water in each channel segment, given an assumed uniform initial depth in meters):
>>> ./MakeChannelState.scr ../input/springbrook.stream.network 0.25 10.01.1995.00.00.00

6. Create shadow and skyview files

1) Compile the program
>>> make -f Makefile.Monthly.Shadow

2) If necessary, make the script '**monthly_shadow_netcdf.scr'** executable:
>>> chmod 755 monthly_shadow_netcdf.scr

3) Edit monthly_shadow_netcdf.scr**,** change the following parameters:
- cell size (**cell**)
- Center latitude (**lat**) and longitude (**lon**)
- number of rows (**rows**) and number of columns (**cols**)
- origin at the upper left corner (**XOrig** & **YOrig**)
- netcdf DEM file
- output path #no trailing slash

4) Run the script
>>>./monthly_shadow_netcdf.scr

Session 5: Running the model and output analyses

Depending on which I/O file type you chose, run the DHSVM source code with the corresponding input configuration file.

```
>>> cd ../sourcecode/
>>> make
>>> ./DHSVM3.1.1 ../configfiles/INPUT.Springbrook.[Bin.txt or NETCDF.txt]
```

Note that the sediment option is turned off in this sample configuration file for the Springbrook creek. If the sediment module or mass wasting option is going to be implemented, please follow the *Tutorial for DHSVM 3.0 (*https://dhsvm.pnnl.gov/*)*

1. Binary Outputs Mapping

The script **plot_modelmap.scr** allows you to plot maps from binary input, and **plot_ncmodelmap.scr** allows you to plot maps from NetCDF input (requires GDAL). Depending on which variable you want to plot, the scripts must be changed slightly.

This shell script can plot only 1 layer of the binary output. For each map you need to designate the map name, the name of the output file, the name of the input file and the name of the color plot table (cpt). Create the Soil Moisture Map first.

Edit **plot_modelmap.scr** so that the Soil Moisture map name, output file, input file and cpt file do not have a "#" in front of it. Add a "#" in front of all the other map file names. Save the file, but leave it open. The paths to each file may require modification.

```
>>> ./plot_modelmap.scr
>>> ctrl-c              <to close ghostview window>
```

Repeat these steps for any other maps for which you created files.
Each of these can also be viewed in ghostscript, e.g.:

```
>>> gs Map.1.Soil.Moist.ps
```

2. NetCDF Outputs Mapping

One of the obvious advantages of using NetCDF Input/Output is the one-step visualization of spatial maps. A variety of existing data viewers allow you to plot the maps with one command line. And numerous popular visualization packages, e.g. GMT, Python, MatLab, the netCDF Operators (NCO), can read the NetCDF format.

Here we use **ncview** tool to have a quick check on graphical NetCDF files:

```
>>> ncview <in.nc>
```