Clinic 1.2 - ANUGA

2017 CSDMS Annual Meeting

May 23, 2017, SEEC N128, 1:30 PM

Mariela Perignon, CSDMS Integration Facility, CU Boulder
perignon@colorado.edu

Requirements: Basic programming experience in Python to experiment with the model. However, participants without prior Python skills should still be able to follow the material. Please discuss and collaborate with your neighbors during the hands-on portions of the clinic.

Offline instructions

The setup section of this document describes the use of ANUGA within a pre-built Cloud9 workspace. To use the model outside Cloud9, you need to install Python 2.7, Jupyter Notebooks, ANUGA, and several other necessary packages. This is most easily done through the Anaconda distribution:

- Install Anaconda Python 2.7.
- In a bash terminal window, use `pip` to install ANUGA:

```
pip install anuga
```

  You can also install the development version of ANUGA from GitHub: GitHub - GeoscienceAustralia/anuga_core: AnuGA for the simulation of the shallow water equation.
- To install the beta versions of the sediment transport and vegetation drag operators, install AnugaSed from the CSDMS model repository.
- To run the Jupyter notebooks we used in the clinic, use a bash terminal window to start the Jupyter Notebook server from within the directory where the `.ipynb` files are located. Use the command `jupyter notebook` (without the additional flags we used in Cloud9). A browser window should open automatically.

Setup on Cloud9

To avoid the hassles of installing software in everyone's computers, we are going to use the

Cloud9 IDE explore and run the model ANUGA.

Cloud9 IDE is an online integrated development environment that supports a wide variety of programming languages. It provides developers with pre-configured workspaces in Ubuntu containers via Docker. Cloud9 is owned by Amazon and hosted on Google Compute Engine.

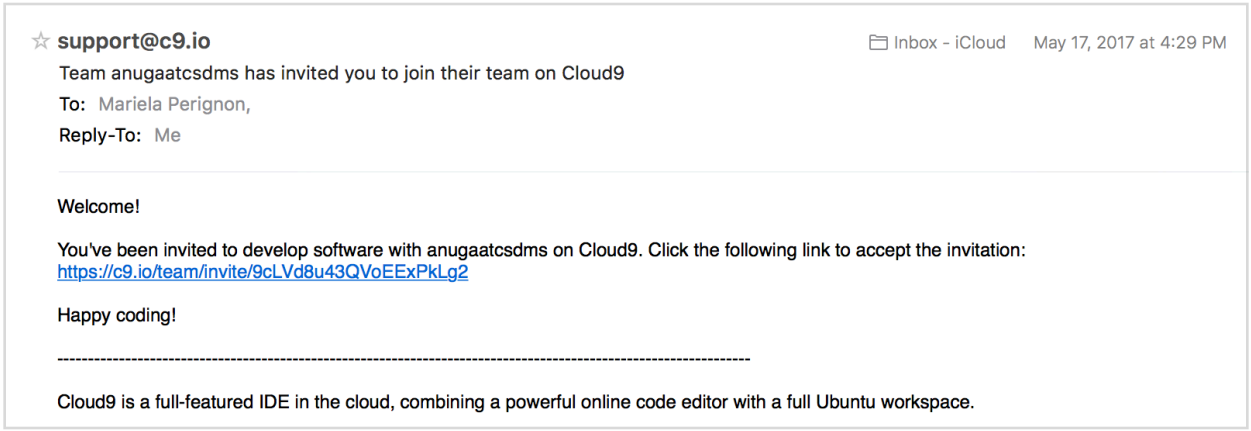
Joining our Cloud9 team:

If you signed up for this clinic in advance, you should have received an email this morning inviting you to join our Cloud9 team. Ask for help now if you did not receive it.

Already have a Cloud9 account and didn't get the email?
If the email address you used to register for the meeting is also the email for your Cloud9 account, you *might* have been automatically added to the team. Log into Cloud9 at https://c9.io. The team "Anuga @ 2017 CSDMS meeting" should be listed in your Dashboard under Your Team Subscriptions. If it's not there, ask for a new invitation.

The email should look like this:

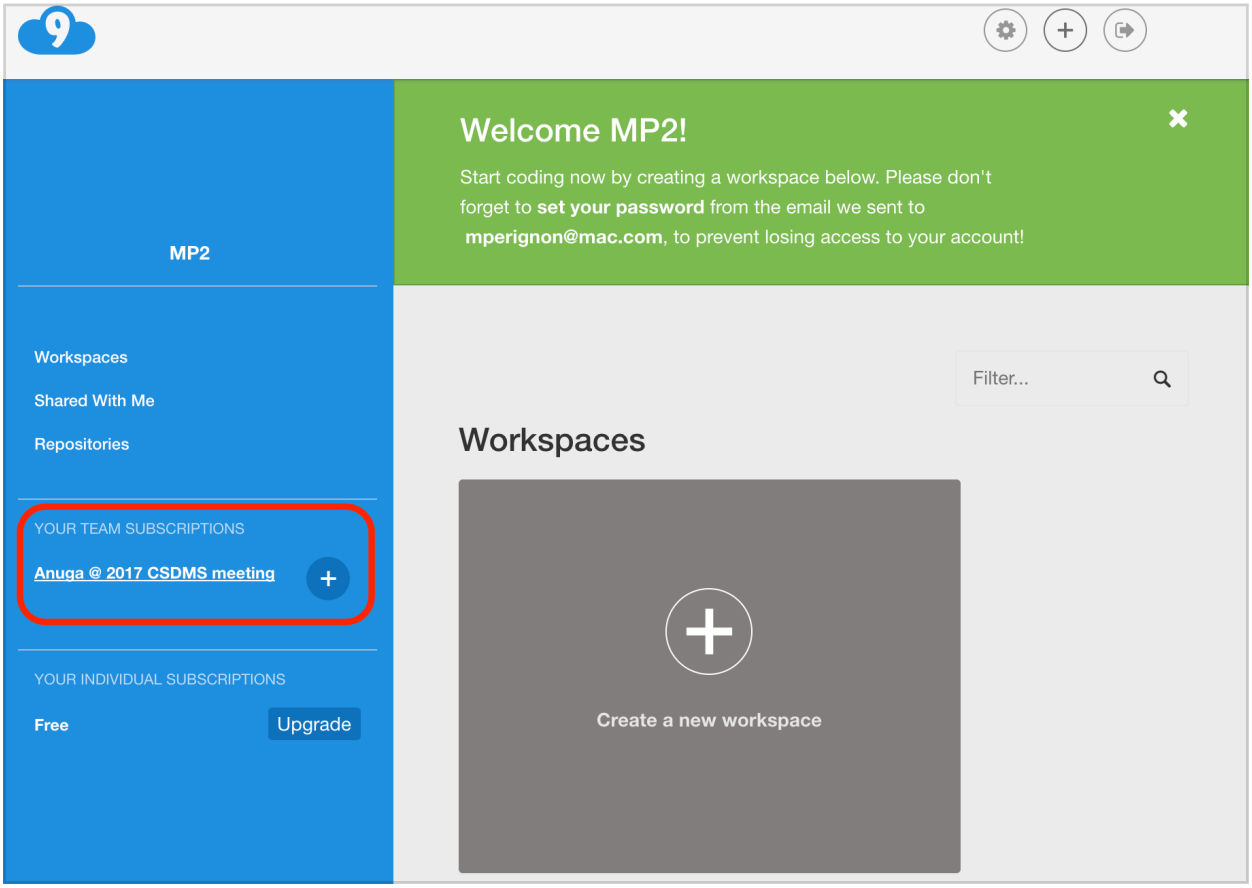
- Email sender: support@c9.io
- Subject: Team anugaatcsdms has invited you to join their team on Cloud9



Follow the link in the email to log into Cloud9 and join our team:

- If you do not have an account on Cloud9, click on "Create new account" and follow the instructions.
- If you already have an account, click on "Sign in".

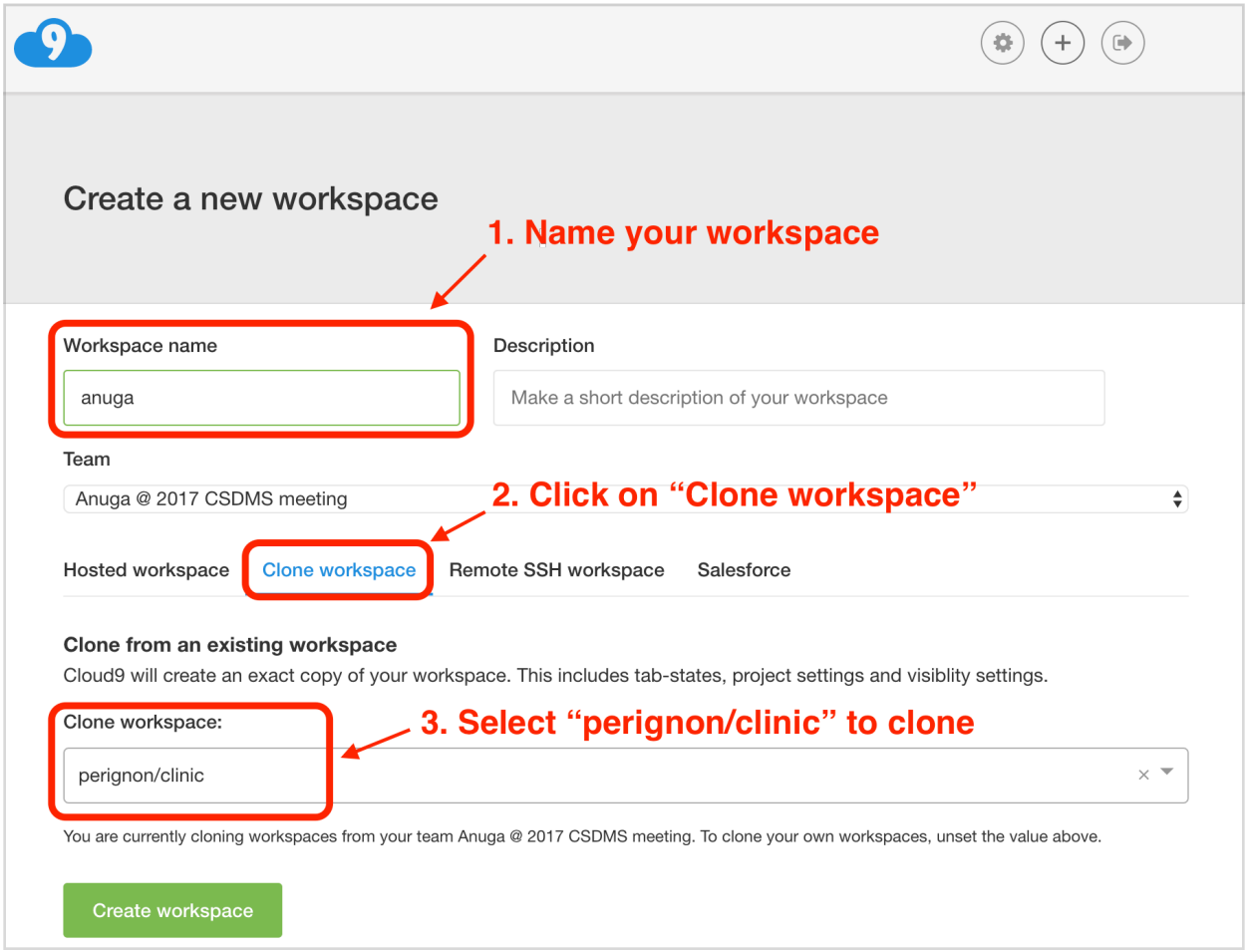
This is your Cloud9 Dashboard. The team "Anuga @ 2017 CSDMS meeting" is listed under Your Team Subscriptions on the left side:



- Go back to your email client and look for a welcome email from support@c9.io. Follow the link to set a password for your account. This will let you jump back in if you accidentally close the browser during the clinic.

Creating a workspace:

- In your Dashboard, click on “Create a new workspace” and follow the instructions below:



This process creates a new Ubuntu workspace for your user that is a copy of an existing workspace where all the necessary software and files for this clinic are pre-installed.

- **Dismiss the “Disk is full” warnings.** We added a script to the login process that cleans up files left over from software installation. The 2GB of disk space allocated to your new workspace should be only half full.

Starting a Jupyter Notebook:

- In the Cloud9 menu, go to Window ➔ New Terminal to open a new Bash terminal window.

- In the terminal window, type:

```
jupyter notebook --ip=0.0.0.0 --port=8080 --no-browser
```

- In a new browser tab, go to:

```
https://<name-of-workspace>-<your-username>.c9users.io
```

- For added security, the Jupyter notebook server requires an authentication token to run for the first time. Go back to your workspace and copy the string of characters after (and including) `:8080`. Each Jupyter notebook server has its own individual authentication token, so you must **copy the token from your own terminal window**:

```
Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://0.0.0.0:8080/?
token=772d717017233ec7b50a38a31a03f55f77b1e5eafd7dc2cb
```

Add it to the URL so it looks like this:

`https://<name-of-workspace>-<your-username>.c9users.io:8080/?token=<your-token>`

The browser will log into the notebook server and open the Jupyter dashboard.

How to use the Jupyter notebook:

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.

You can run the notebook document step-by-step (one cell a time) by pressing Shift + Enter.

Python will interpret the code in the order the cells are run, not their order in the notebook.

To learn the basics of Jupyter notebooks, see this tutorial.

Exercise 1 - Runup:

This example solves the shallow-water wave equation for a flat bed that is sloping at a fixed angle in one direction. It demonstrates the basic ideas involved in setting up a modeling scenario. In general, the user specifies the geometry (bathymetry and topography), the initial water level, boundary conditions such as tide, and any forcing terms that may drive the system such as rainfall. This simple scenario does not include any forcing terms nor is the elevation data taken from a file, as it would typically be.

- In the Jupyter dashboard, click on the notebook *runup.ipynb*. A new browser tab will open.
- Read the explanatory comments in the code and run each cell in order.

Outline of the Program:

To run the model scenario, the *runup* notebook performs the following steps:

1. Sets up a triangular mesh.
2. Sets certain parameters governing the mode of operation of the model, specifying, for instance, where to store the model output.
3. Inputs various quantities describing physical measurements, such as the elevation, to be specified at each mesh point (vertex).
4. Sets up the boundary conditions.

5. Carries out the evolution of the model through a series of time steps and outputs the results into a file.

Establishing the domain:

ANUGA is installed as a Python package. Import ANUGA into the Python script as with any other library:

```
import anuga
```

Initial conditions:

We can specify a number of quantities at each mesh point. The conserved quantities (that evolve over time) are *stage* (absolute height of water surface), *x-momentum* and *y-momentum*.

The method `set_quantity` is used to specify quantities. It allows the user to set quantities in a variety of ways – using constants, functions, numeric arrays, expressions involving other quantities, or arbitrary data points with associated values, all of which can be passed as arguments. All quantities (conserved, non-conserved, and user-defined) can be initialized using `set_quantity`.

We set *elevation* using a function to be a ramp with a constant slope. The quantity *stage* is set to constant lower than the minimum elevation to guarantee no flow depth throughout the domain.

Other domain options:

- The method `set_name` specifies the name of the output file.
- The method `set_quantities_to_be_stored` defines which (and how frequently) quantities are saved to the output file. Quantities given a value of "1" will be saved once at the beginning of the model run. Quantities given a value of "2" are stored at every timestep.

Boundary conditions:

ANUGA can use different boundary conditions at the edges of the model grid. Each boundary condition specifies the behavior at a boundary in terms of the behavior in neighboring elements. There are four basic boundary types:

- **Reflective boundary:** A solid wall. Returns same stage as in its neighbor volume but momentum vector reversed 180 degrees (reflected).
- **Transmissive boundary:** Returns same conserved quantities as those present in its neighbor volume. This is one way of modeling outflow from a domain but may cause numerical instabilities if flow is not at steady state.

- **Dirichlet boundary:** Specifies constant values for stage, x-momentum and y-momentum at the boundary. Use for outflow by setting stage to be lower than the elevation.
- **Time boundary:** Like a Dirichlet boundary but with behavior varying with time.

Each edge of a domain has an associated *tag*. By default, the rectangular domain uses *left*, *right*, *top*, *bottom*. We'll set custom tags to boundaries in a later example. The method `set_boundary` is used to associate different types of boundary conditions to each boundary tag.

Evolve:

Calling the method `evolve` in a loop causes the domain to evolve over a series of steps indicated by the values of *yieldstep* and *duration*. The value of yieldstep controls the time interval between successive model outputs. Behind the scenes, the model calculates a smaller, internal timestep.

Model output file:

The model output is saved to a NetCDF file with the extension `.sww`. The contents of the ANUGA output file can be accessed just like any other NetCDF file.

Other scripts in this notebook:

- **Load data from SWW output file:** This script loads data from an ANUGA output file into Numpy arrays.
- **Plot the triangulated output data:** This script triangulates the loaded output data and creates a figure of one time-slice of a quantity.
- **Animate:** This script creates and displays an animation of the bed and evolving water surface. SLOW!

Experiment with Example 1:

1. Change the boundary conditions:

Change the boundary conditions for the `right` boundary tag to a *Time Boundary* by changing the boundary type in the method `set_boundary`:

```
domain.set_boundary({'left': Br,    # Reflective
                     'right': Bw,   # Time Boundary
                     'top': Br,     # Reflective
                     'bottom': Br}) # Reflective
```

2. Alter the domain size and grid density:

Modify the arguments of the function `rectangular_cross_domain` to create a different sized rectangular domain or a grid with different node spacing.

3. Change the bed topography:

In the function that creates the topography (`topography(x,y)`), change the equation for bed elevation to a curved slope:

```
def topography(x, y):
    return x*(-(2.0-x)*.5) + 5
```

4. Add a forcing term (friction):

The quantity *friction* is used to specify the Manning friction coefficient at every mesh point. Use the method `set_quantity` to assign a constant numerical value to the quantity *friction*: `domain.set_quantity('friction', 0.03)`

5. Plot flow velocity:

Both x-momentum and y-momentum are saved to the output file at every timestep. Add them to the quantities imported from the NetCDF file and convert them to flow velocities:

```
xmom = fid.variables['xmomentum'][:]
xvel = xmom / (depth + 1e6) # to avoid division by 0

ymom = fid.variables['ymomentum'][:]
yvel = ymom / (depth + 1e6) # to avoid division by 0
```

Change the plotting commands to plot velocity:

```
plt.figure()
plt.tripcolor(triang, xvel[-1], edgecolors='None', cmap='YlGnBu')
plt.colorbar()
plt.axis('equal');
```

Exercise 2 - Topography:

This example introduces methods for importing and using a DEM to create the model domain. We are using the DEM for a short reach of the lower Rio Puerco, NM: https://goo.gl/maps/7UHQXU9kPKB2

Instead of imposing a mesh structure on a rectangular grid, this example uses a mesh generator to build mesh structures inside a polygon set by the user. The *mesh resolution* (the maximum area of a triangle used for triangulation) and the boundary tags are also set during the creation of the domain.

It is also possible to use this technique to specify areas of increased resolution defined by multiple interior polygons, each with a specified resolution. The user can also specify one or more 'holes' – that is, areas bounded by polygons in which no triangulation is required.

The function used to implement this process is `create_domain_from_regions`. This creates a Domain and a mesh file from the bounding polygon and its resolution, a list of boundary tags, and (optionally) a list of interior polygons and their resolution. Note that every point on each polygon defining the mesh will be used as vertices in triangles. Consequently, *polygons with points very close together will cause triangles with very small areas to be generated irrespective of the requested resolution*. Make sure points on polygons are spaced to be no closer than the smallest resolution requested.

Experiment with Example 2:

1. Change the downstream boundary condition:

Replace the boundary condition at the outlet with a wall (`Reflective_boundary`) and observe the flow depth.

2. Increase resolution for an interior region:

Add to the domain an interior region of higher resolution defined by the polygon in the file `'data/inner_polygon.csv'`. Import the interior polygon into the script using the function `read_polygon` and add the argument `interior_regions` to the function `create_domain_from_regions`. `interior_regions` requires a list of pairs of interior polygons and the resolutions within them:

```
inner_polygon = anuga.read_polygon('data/inner_polygon.csv')
domain = anuga.create_domain_from_regions(
                bounding_polygon = bounding_polygon,
```

```
                boundary_tags = boundary_tags,
                maximum_triangle_area = 200,
                mesh_filename = filename_root + '.msh',
                interior_regions = [[inner_polygon, 100]])
```

To visualize the grid density, create a scatter plot of the x and y coordinates of the grid. Set the marker size to 2:

`plt.scatter(x, y, s = 2)`

2. Construct a hydrograph:

Use the script provided in the last cell of the Jupyter notebook to extract, at every timestep, the maximum value of depth across the domain at a given value of `y`.

3. Measure flow velocity:

Copy the hydrograph script into a new cell and modify it to show flow velocity instead of depth. Be aware that negative y velocities are directed downstream.

2. Add a forcing term (friction):

Use the method `set_quantity` to assign a constant numerical value to the quantity *friction*. Compare the maximum flow velocity and depth at the location of the "stream gage".

`domain.set_quantity('friction', 0.03)`

Exercise 3 - Operators:

Fractional step operators can change the values of quantities at each inner timestep. ANUGA includes several operators including ones for infiltration, rainfall, kinematic viscosity, and flow through culverts. Users can also create their own operators.

In this exercise, we will use the *beta versions* of two custom operators:

- A **vegetation operator** that calculates the drag that vegetation imparts on the flow following the formulation of *Kean and Smith, 2004*. Vegetation is simulated fields of cylinders of a given diameter set a given distance apart. In this exercise, we will use a uniform field of vegetation over the entire domain.
- A **sediment transport operator** that calculates entrainment and deposition rates, sediment fluxes in and out of cells, and the resulting changes of bed elevation.

The suspended sediment operator requires that the new quantity *concentration* be treated by

the model as an *evolved quantity*. In this example, we have to create the mesh and the domain with separate commands in order to override the default evolved quantities. We first call the function `anuga.pmesh.mesh_interface.create_mesh_from_regions` to generate the mesh from the DEM and boundary polygon. We then initialize the domain with a direct call to the class `Domain`.

The sediment transport operator modifies the elevation of the bed through erosion and deposition. To record these changes, we changed the output frequency of `elevation` to 2 (at every timestep). We also included the new quantity `concentration` in `set_quantities_to_be_stored`.

Experiment with Example 3:

1. Run the two operators separately:

Run each of the operators separately by commenting them out of the script (Command-/ or Control-/ will turn the any selected lines in the code to a comment).

2. Change the sediment concentration:

Increase and decrease the value of `concentration` and observe the patterns of elevation change. Keep this quantity between 0 and 0.05!

3. Change the vegetation characteristics:

Change the vegetation spacing and diameter and observe the effects on flow velocity and sediment transport.