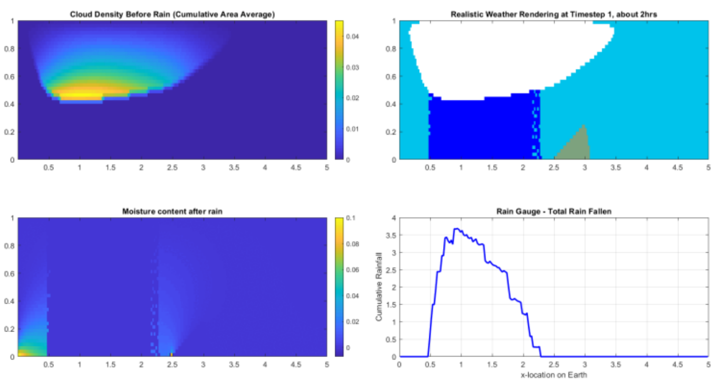


Rainshadow Simulation:

Using the SIMPLE Algorithm to Model Orographic Precipitation with CFD

Ted Yee

ME7310: Graduate Computational Fluid Dynamics with Heat Transfer

Term Project Phase 3 Report

Wednesday, April 23, 2025

Overview

Code modification or studies: For this portion of your exam, you need to make **two types** of modifications to your code. You have six options on how you can change your code. For each option, you need to submit the lines of code you need to change before and after modification and an explanation for this change (Use the Compare feature). Then you will need to demonstrate that the change was effective, and what the impact was on the rest of the domain. The necessary demonstration will be detailed in each of the options below.

When you modify your codes, copy all of your original codes into a new folder named `Option_N` where N is the option number you work on. Submit to blackboard your code modification description (pdf), figures, and each of the folders with the entire code for each modification.

In the event that you were not able to get a functioning code, you can make modifications to the sample code contained in `step_down.zip`.

This is the report of the 2 modification and some closing remarks about the model's validity and a bug fix since Phase 2. Options 1 and 2 were chosen for the modifications because the others would take too long to run all parameter variations. Changing the inlet velocity profile was simply a weather pattern not capable of producing a rainshadow, and changing surfaces to no-slip had negligible effect.

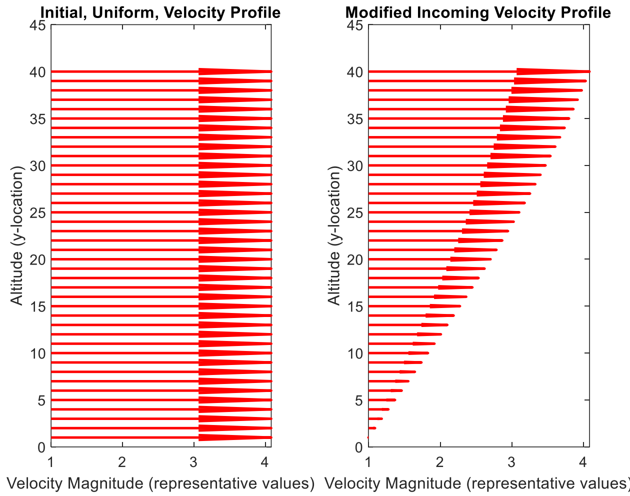
Option 1: Inlet BC

Option 1: Change the inlet boundary condition. Change one of your inlet boundary condition from a constant velocity inlet to the spatially varying inlet

$$u_{inlet}(y) = \sin\left(\frac{y - y_{bottom}}{y_{top} - y_{bottom}}\pi\right)$$

where y_{bottom} is the position of the bottom of the inlet and y_{top} is the position of the top of the inlet (or equivalent in case you have a horizontal inlet, with $u_{inlet}(x)$). To demonstrate the change, in one figure, plot the horizontal velocity of the u-cell centers one column in from the inlet before and after the change (see red dashed line/arrows for example placement and velocity component in figure). This will show the contrast between the constant and variable inlet condition. Also plot the full domain result before and after the change and comment on what you find.

The most direct result of changing the velocity profile is that the conditions to create a rainshadow are not met. In the base simulation, the inlet is a uniform velocity value of 10 at every altitude. Below is a graphical representation (not that the arrow lengths are representative and not the actual velocity values) of the new input velocity profile.



Changing the code to implement this was really as simple as changing one line, but I included the plotting functions as well. In order to display both the uniform and varying inlet profile, I plotted and *then* updated within the main script after the values are set by solver_gui.m. The comparison was done in VSCode because the MATLAB comparison was hard to read.

```
clear; close all; clc;
tic

%%
cell_visualizations_on_off =1;
blocked = 1;
rainshadow = 0;
%%%%ideally a geometry gui would set the values currently in
%%%%system_parameters, then once submitted would generate, then gui wo
%%%%setup methods then begin once the initialize and run button is cl
system_parameters                                   %% Set system, g
solver_gui
if blocked ==1
    set_cell_type_blocked
else
    set_cell_type
end
if rainshadow ==1
    %% Mountain Parameters
    rise_slope = .5;
    fall_slope = -1;
    x_mountain_start = floor(N.x_p*50/100);
    x_mountain_peak = floor(N.x_p*60/100);
    x_mountain_end = floor(N.x_p*120/100);
    ambienthumidity = 0;
    scalefactor     = 10000;                %Default = 10000, Vert
```

```
clear; close all; clc;
tic

%%
cell_visualizations_on_off =1;
blocked = 1;
rainshadow = 0;
%%%%ideally a geometry gui would set the values currently in
%%%%system_parameters, then once submitted would generate, then gui would
%%%%setup methods then begin once the initialize and run button is clicked
system_parameters                                   %% Set system, geometry, and solver parameters
solver_gui

[X,Y] = meshgrid(1:N.y_u,1:N.x_u);
left_boundary_x = ones(N.y_u, 1);
left_boundary_y = (1:N.y_u)';
figure(19)
tiledlayout(1,2)
nexttile
quiver(left_boundary_x, left_boundary_y, BC.u_lef_value, zeros(N.y_u,1), 0.5, 'r', 'LineWidth', 1.5);
title('Initial, Uniform, Velocity Profile')
xlabel('Velocity Magnitude (representative values)')
ylabel('Altitude (y-location)')
nexttile
BC.u_lef_value = sin(grids.y_u'-grids.y_u(1)/(grids.y_u(end)-grids.y_u(1)*pi));
quiver(left_boundary_x, left_boundary_y, BC.u_lef_value, zeros(N.y_u,1), 0.5, 'r', 'LineWidth', 1.5);
title('Modified Incoming Velocity Profile')
xlabel('Velocity Magnitude (representative values)')
ylabel('Altitude (y-location)')


if blocked ==1
    set_cell_type_blocked
else
    set_cell_type
end
if rainshadow ==1
    %% Mountain Parameters
    rise_slope = .5;
    fall_slope = -1;
    x_mountain_start = floor(N.x_p*50/100);
    x_mountain_peak = floor(N.x_p*60/100);
    x_mountain_end = floor(N.x_p*120/100);
    ambienthumidity = 0;
    scalefactor     = 10000;                %Default = 10000, Vertical scale factor in simulation lengt
```

Since ANTSSSS is already set up to handle varying (text string, function, scalar, and vector) inputs, there were no further changes needed. The input data is read from the GUI textboxes and passed through a local function (inside solver_gui.m) that converts all boundary conditions into 1D vectors of the length of the edge they are applied to.

The boundary condition vectors are then stored in a structure called BC and when they are applied in the simple algorithm, the nonuniform inlet profile is (and was in the base code) handled by a set of indexes.

```matlab
function BC = update_plot(fig, grids, N,base)
    try
        % Retrieve the stored value from guidata
        data = guidata(fig);
        disp(fieldnames(data));  % This will help in debugging if the fields ex

        % Retrieve and evaluate inputs for boundary conditions
        BC.u_lef_value = evaluate_input(data.u_lef_value_box.Value, grids, N);
        BC.u_rig_value = evaluate_input(data.u_rig_value_box.Value, grids, N);
        BC.u_bot_value = evaluate_input(data.u_bot_value_box.Value, grids, N);
        BC.u_top_value = evaluate_input(data.u_top_value_box.Value, grids, N);
        BC.u_lef_type  = data.u_lef_type_dropdown.Value;
        BC.u_rig_type  = data.u_rig_type_dropdown.Value;
        BC.u_bot_type  = data.u_bot_type_dropdown.Value;
        BC.u_top_type  = data.u_top_type_dropdown.Value;


        BC.v_lef_value = evaluate_input(data.v_lef_value_box.Value, grids, N);
        BC.v_rig_value = evaluate_input(data.v_rig_value_box.Value, grids, N);
        BC.v_bot_value = evaluate_input(data.v_bot_value_box.Value, grids, N);
        BC.v_top_value = evaluate_input(data.v_top_value_box.Value, grids, N);
        BC.v_lef_type  = data.v_lef_type_dropdown.Value;
        BC.v_rig_type  = data.v_rig_type_dropdown.Value;
        BC.v_bot_type  = data.v_bot_type_dropdown.Value;
        BC.v_top_type  = data.v_top_type_dropdown.Value;

        assignin('base', 'BC', BC);
        disp('✅ BC stored in base workspace');
        uiresume(base);
```

```matlab
function value = evaluate_input(input_str, grids, N)
    try
        % Replace 'x' and 'y' with grid variables
        input_str = strrep(input_str, 'x', 'grids.x_p');
        input_str = strrep(input_str, 'y', 'grids.y_p');

        func = str2func(['@(grids) ' input_str]);
        value = func(grids);

        if isscalar(value)
            if isvector(value)
                value = value * ones(N.y_p, 1);
            else
                value = value * ones(N.x_p, 1);
            end
        end
    catch ME
        error('Error evaluating input "%s": %s', input_str, ME.message);
    end
end
```

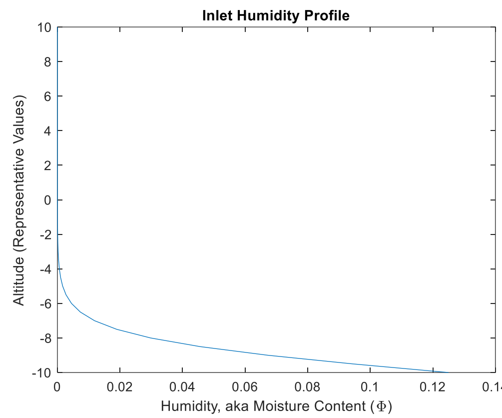
A vector of 4 indexes keeps track of which point in the varying boundary condition is being applied, and since MATLAB reads the cell types in order, the indexes can be incremented and applied one data point at a time

```matlab
index=ones(1,4);    %%% indices for scrolling through nonuniform BC

%% Update Matrices by cell type
for i = 1:length(u_type(:))
    %%Right/East Edge
    if u_type(i) == 2
            A_u(i,i) = 1;
            A_u(i,i-1) = -1;
            b_u(i) =  BC.u_rig_value(index(1));


    %%Bottom/South Edge
    elseif u_type(i) == 4
        if BC.u_bot_type == "Dirichlet"
            A_u(i,i)       =  1;
            b_u(i) = BC.u_bot_value(index(2));
            index(2)=index(2)+1;
        elseif BC.u_bot_type == "Neumann"
            A_u(i,i) = 1;
            A_u(i,i-1) = -1;
            b_u(i) =  BC.u_bot_value(index(2));
            index(2)=index(2)+1;
        else                    %%% for free-slip
            A_u(i,i)       =  Fe_u(i)-Fw_u(i)+Fn_u(i)-Fs_u(i)+De+Dw+Dn+2*Ds + 1/dt;
            A_u(i,i-Ny_u) = -Fw_u(i)-Dw;
            A_u(i,i+Ny_u) =  Fe_u(i)-De;
```

Results

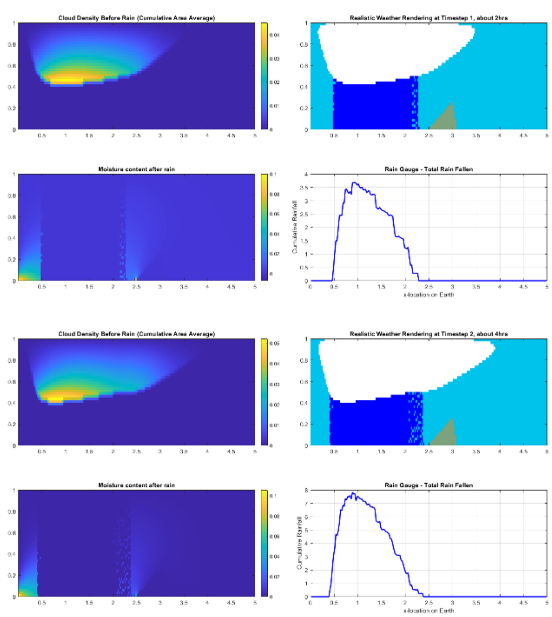
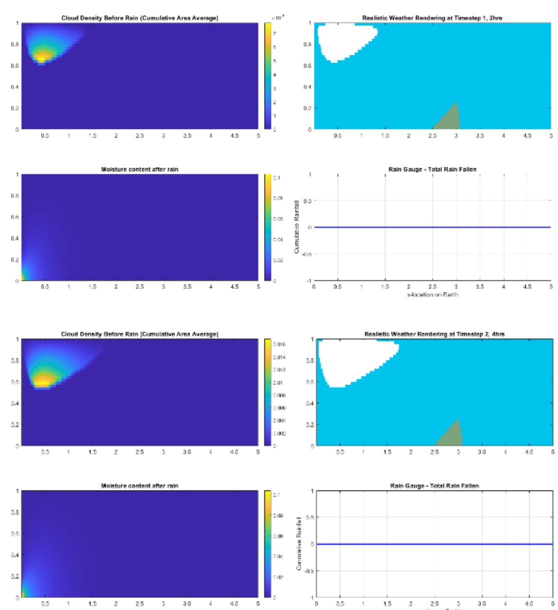
It is important to note that the inlet humidity profile (which is handled in the same way as the velocity inlet conditions) is by default set to a sigmoid function. In the solver_gui.m, where the rainshadow preset values are applied, line 211 (which existed in the base simulation, though labels were added here) can be uncommented to show the humidity profile. The default setting is to have high humidity at low altitudes (modeling evaporation low over a body of water), which drops off as altitude increases:

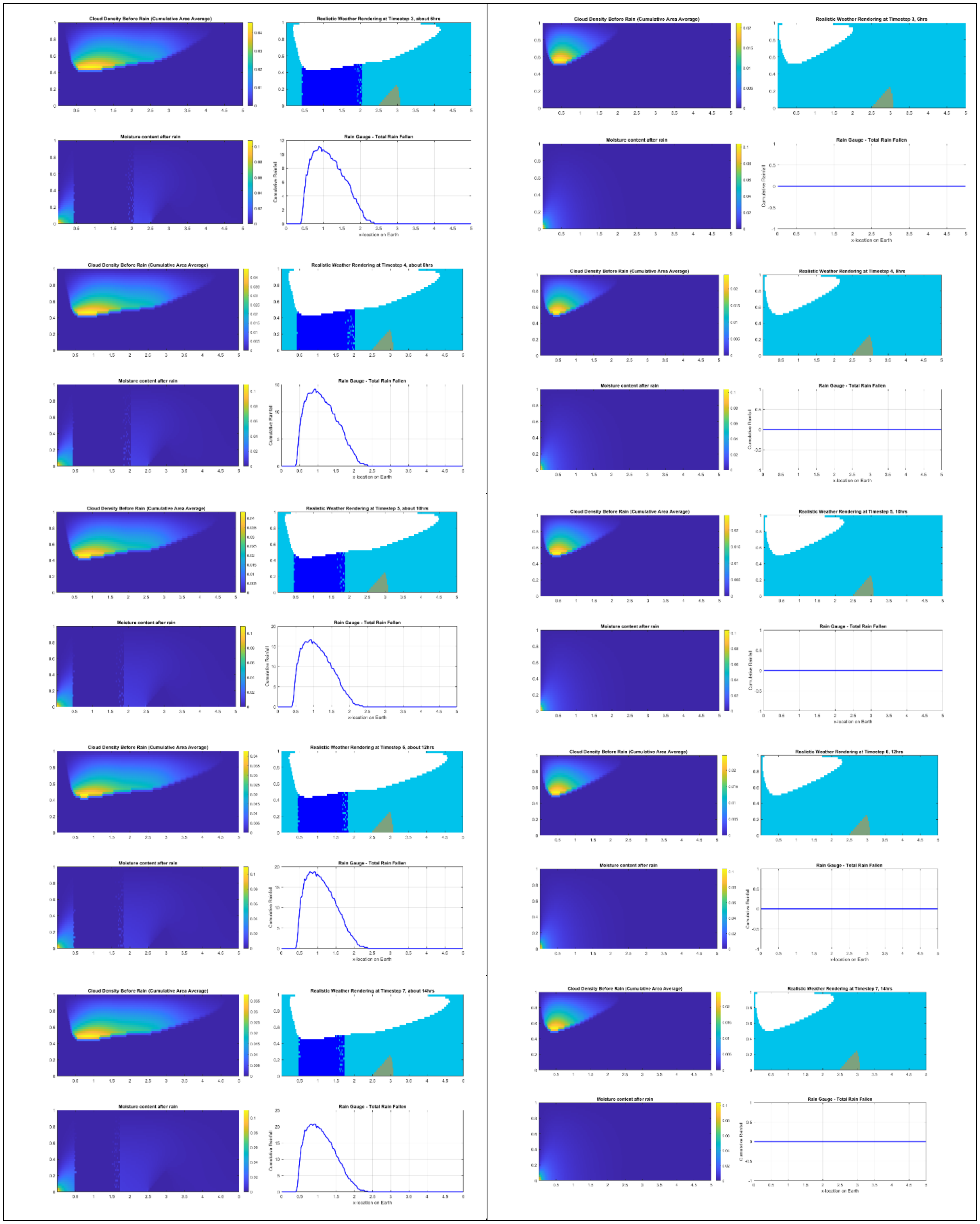
```
            BC.v_bot_value = zeros(N.x_v,1);
            BC.v_top_type = "Dirichlet";    % edge of atmosphere
            BC.v_top_value = zeros(N.x_v,1);
        x = linspace(-10, 10, N.y_p+1);       % Adjust range for shar
        phi_in = .25*(1- (1 ./ (1 + exp(-10-x))));      % Sigmoid fu
        % plot(phi_in,x)
        % title('Inlet Humidity Profile')
        % ylabel('Altitude (Representative Values)')
        % xlabel('Humidity, aka Moisture Content (\Phi)')

        uiresume(base);
    end
```

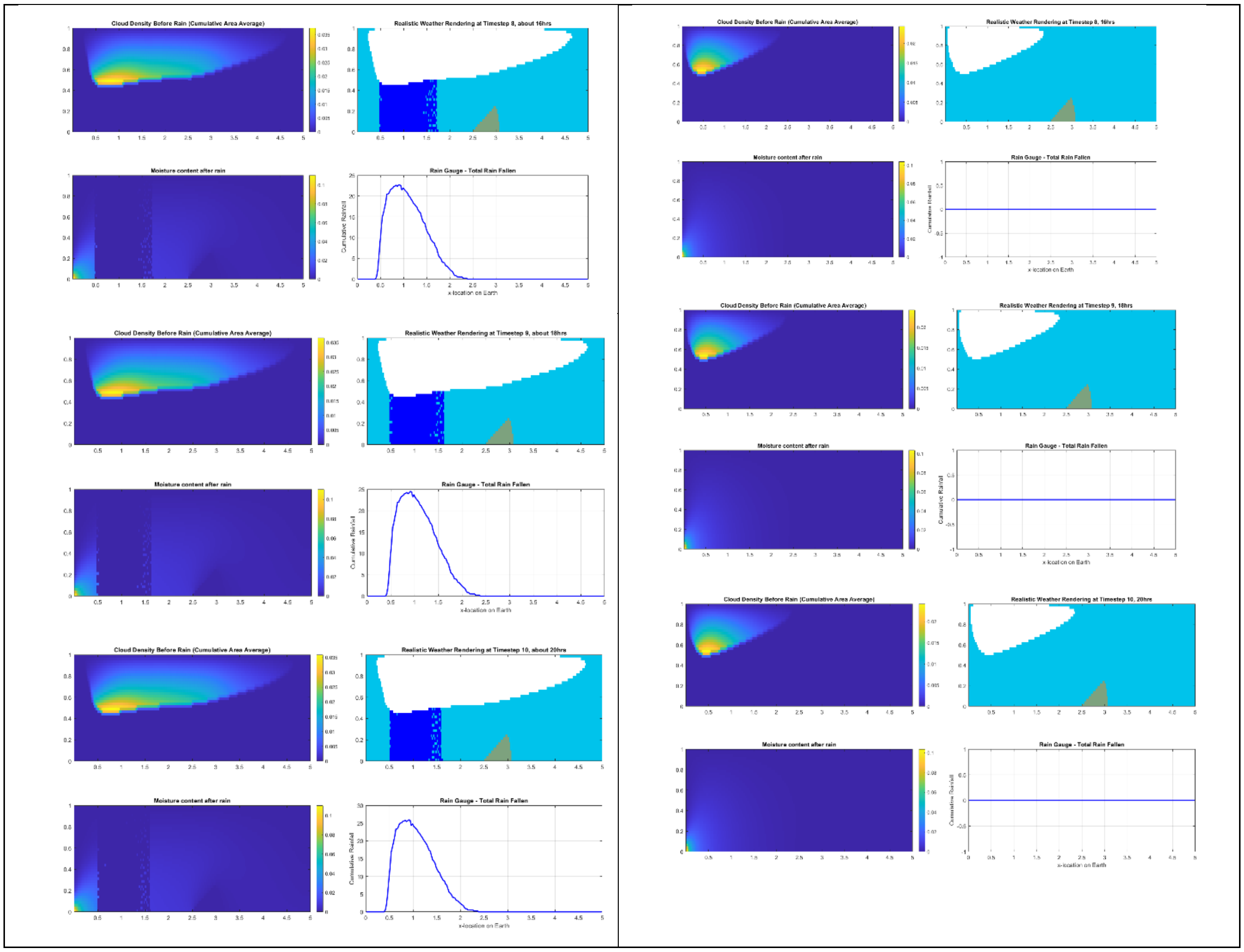


By changing the inlet velocity to be very low at the bottom (where most of the moisture is) and high at the top (where there is little moisture), there is not enough movement to model a rainshadow within the first 10 timesteps. In the below table, each timestep is compared to the default preset values where velocity is uniform.

The simple conclusions are that the nonuniform boundary condition-handling works as expected and that this wind profile does not create a rainshadow. As the vector-handling was already tested and the rainshadow conditions were derived through testing, neither were surprising.

Uniform Inlet Velocity Profile	Varying Inlet Veloctiy Profile
	

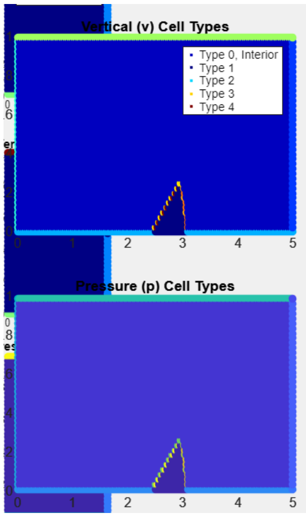

Cloud Density Before Rain (Cumulative Area Average)
Realistic Weather Rendering at Timestep 3, about 6hrs
Moisture content after rain
Rain Gauge - Total Rain Fallen
Cumulative Rainfall
x-location on Earth
Realistic Weather Rendering at Timestep 4, about 8hrs
Realistic Weather Rendering at Timestep 5, about 10hrs
Realistic Weather Rendering at Timestep 6, about 12hrs
Realistic Weather Rendering at Timestep 7, about 14hrs
Realistic Weather Rendering at Timestep 3, 6hrs
Realistic Weather Rendering at Timestep 4, 8hrs
Realistic Weather Rendering at Timestep 5, 10hrs
Realistic Weather Rendering at Timestep 6, 12hrs
Realistic Weather Rendering at Timestep 7, 14hrs

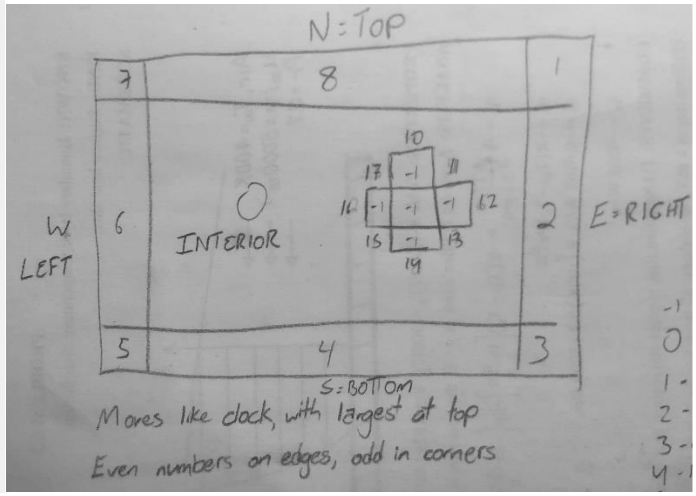


Option 2: No-Slip/Free-Slip

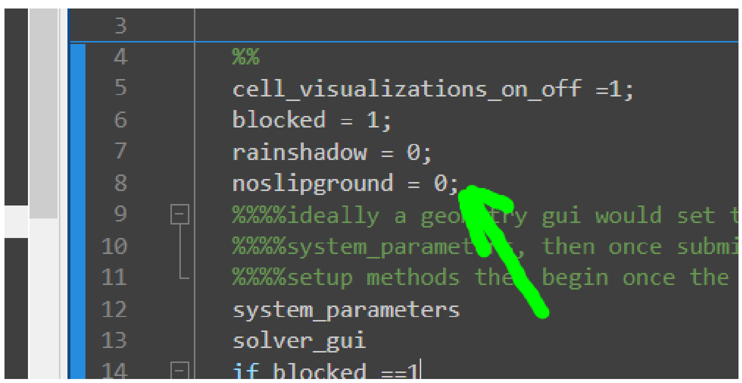
Option 2: Change a no-slip or a free-slip conditions. For this option, pick a no-slip or free-slip boundary condition in your domain and change it to the opposite (no-slip becomes free-slip or free-slip becomes no-slip). State clearly which boundary you are going to change and what it is changed to. Plot a before and after of a vertical profile of the horizontal velocity (see green in figure). Also plot the full domain result before and after the change and comment on what you find.

This change of free-slip to no-slip on the mountain surface actually had negligible effect. The boundary condition that I changed was actually one that should've been set in the base simulation to better simulate real-life. In the base simulation, the surfaces of the mountain (which contain 7 different cell types, per the cell-type setting and key below) were all free-slip. To model the wind-resistance caused by surface roughness, trees, etc., both sides of the mountain should be no-slip.



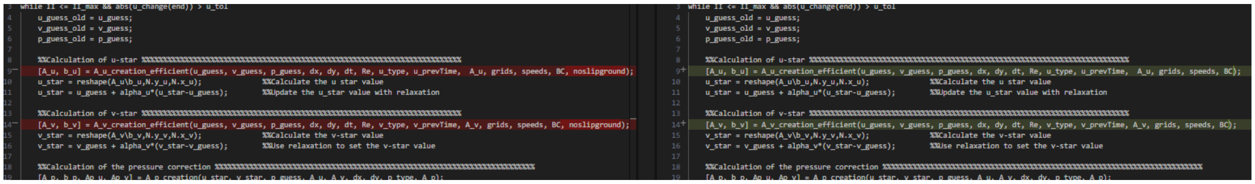


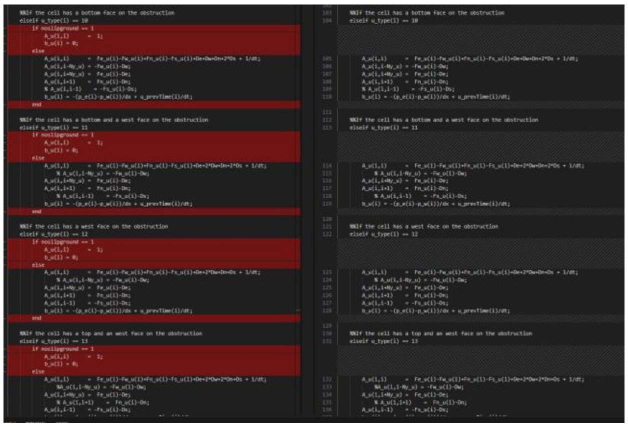
Once again, the base model already had a lot of the structure to handle cases like these. First, by initializing a value called noslipground in the main.m script, ANTSSSSS can handle cases both with and without noslip mountains.

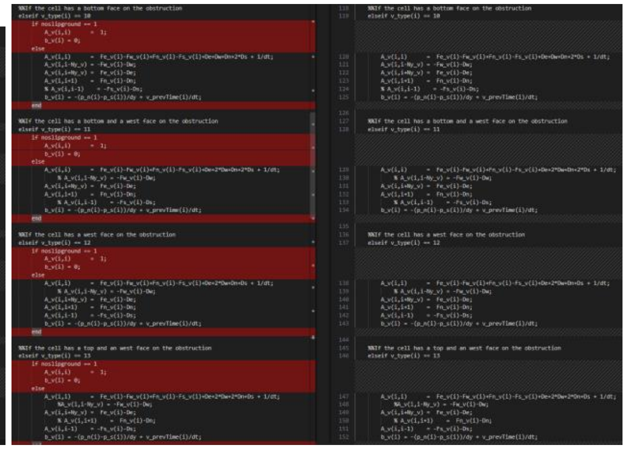


To ensure that this parameter is 1 when the ‘Run Preset Rainshadow Simulation Instead’ button is pressed, the value just needs to be set in solver_gui, and passed out of the GUI.

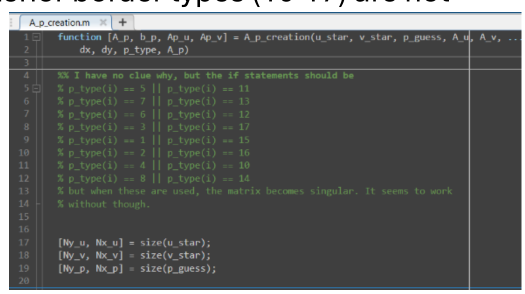
By passing the variable into A_u and A_v creation scripts, the variable can be used in an if statement (so it can be changed easily later) and the A_matrices are set in the exact same way as Dirichlet conditions in the other cell types.







The value is not passed to A_p_creation, because the interior border types (10-17) are not even used there. During testing of the base simulation, it was found that the simulation ran exactly as expected without any conditions applied and that applying conditions to them actually resulted in singular matrices, so they were left out per the note at the top.



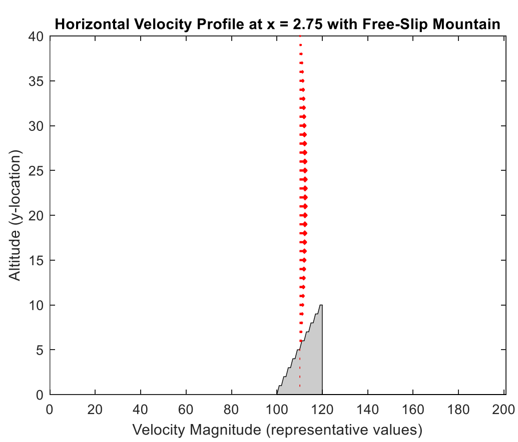
Results

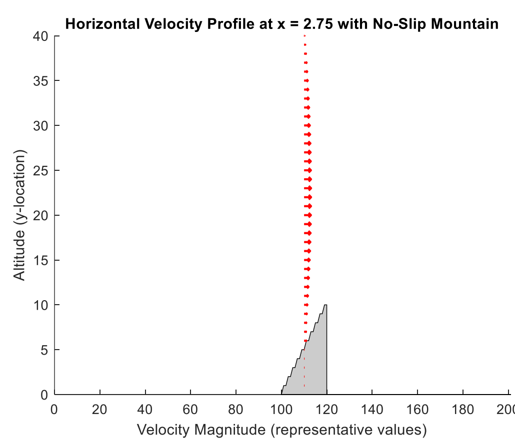
To better visualize the changes, the following code can be added to the end of the main.m script.

```
end
plot_cell_type
guess_initialization                                  %% Generate the initial guess for


for n = 1:Nt-1                                       %%%%%%%%%%%%%% TIME MARCHING
    disp(['TIME STEP NUMBER: ', num2str(n)])
    disp('time_step_initialization')
    time_step_initialization                          %% Set the variables to begin a ne
    disp('simple_algorithm')
    simple_algorithm                                  %% Begin recursive calculation of
    disp('heat_transfer_step')
    heat_transfer_step                                %% Perform a heat transfer step
    disp('plot_store_results')
    plot_store_results                                %% Plot results and store the u^n+
    disp('clouds')
    clouds                                            %% Clouds and Rain
end

figure(19)
hold on
x_mountain = [x_mountain_start, x_mountain_peak, x_mountain_end];
% Number of points on each side of mountain
n_left = x_mountain_peak - x_mountain_start + 1;
n_right = x_mountain_end - x_mountain_peak;

left_x = linspace(x_mountain_start, x_mountain_peak, n_left);
left_y = round(rise_slope * (left_x - x_mountain_start));
right_x = linspace(x_mountain_peak, x_mountain_end, n_right);
right_y = round(fall_slope * (right_x - x_mountain_peak));
x_values = [left_x, right_x];
y_mountain = [left_y, right_y];

x_values = [x_values, x_mountain_end];  % Add the end point for x
y_mountain = [y_mountain, 0];  % Add the bottom point (y = 0) for closing the shape
fill(x_values, y_mountain, 'k', 'FaceAlpha', 0.2);
%Quiver
xlocation = floor(2.75/dx);
x_quiver = ones(N.y_u, 1) * xlocation;
left_boundary_y = (1:N.y_u)';
quiver(x_quiver, left_boundary_y, u_guess(1:end, xlocation), v_guess(1:end-1, xlocation)
title(['Horizontal Velocity Profile at x = ', num2str(xlocation*dx), ' with No-Slip Mour
xlabel('Velocity Magnitude (representative values)');
ylabel('Altitude (y-location)');
axis([0 N.x_u 0 N.y_u]);
hold off;

%%
toc
```

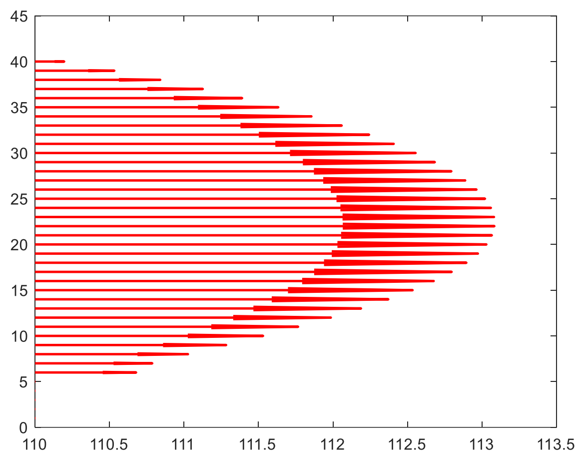
```
end
plot_cell_type
guess_initialization                                  %% Generate the initial gu


for n = 1:Nt-1                                       %%%%%%%%%%%%%% TIME MARCHIN
    disp(['TIME STEP NUMBER: ', num2str(n)])
    disp('time_step_initialization')
    time_step_initialization                          %% Set the variables to be
    disp('simple_algorithm')
    simple_algorithm                                  %% Begin recursive calcula
    disp('heat_transfer_step')
    heat_transfer_step                                %% Perform a heat transfer
    disp('plot_store_results')
    plot_store_results                                %% Plot results and store
    disp('clouds')
    clouds                                            %% Clouds and Rain
end

%%
toc
```

It turns out that changing the free-slip/no-slip conditions of the mountain surface has negligible (but measurable) effect. This is probably due to the fact that ample space is given both upwind and downwind of the mountain and that the atmospheric ceiling (10,000m, where the temperature lapse rate deviates from linear) is well above the 3000m mountain. Plotting the velocity profiles in each appear almost identical:

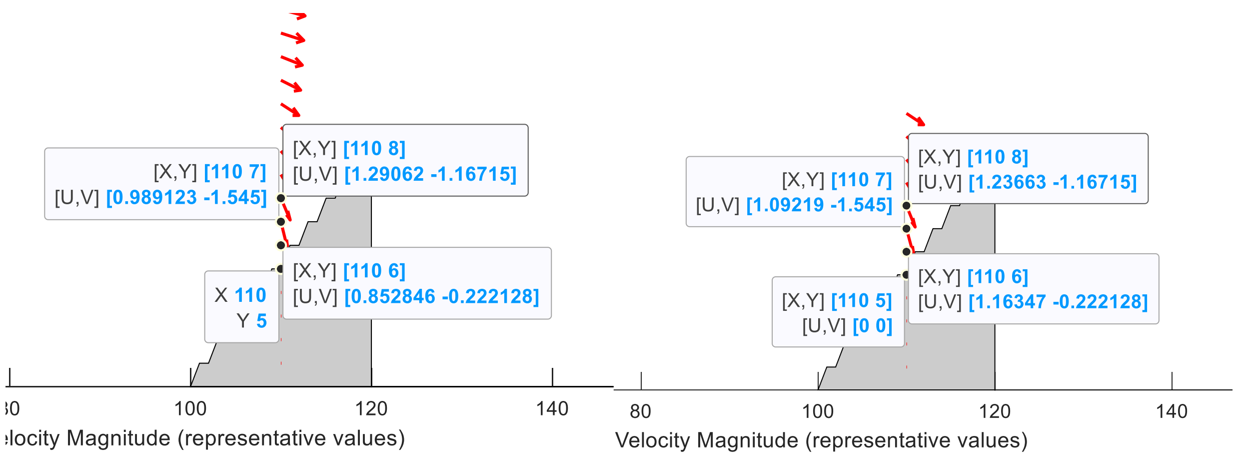




By rescaling the profile, confidence is added that the velocity profile is the typical fluid parabola:



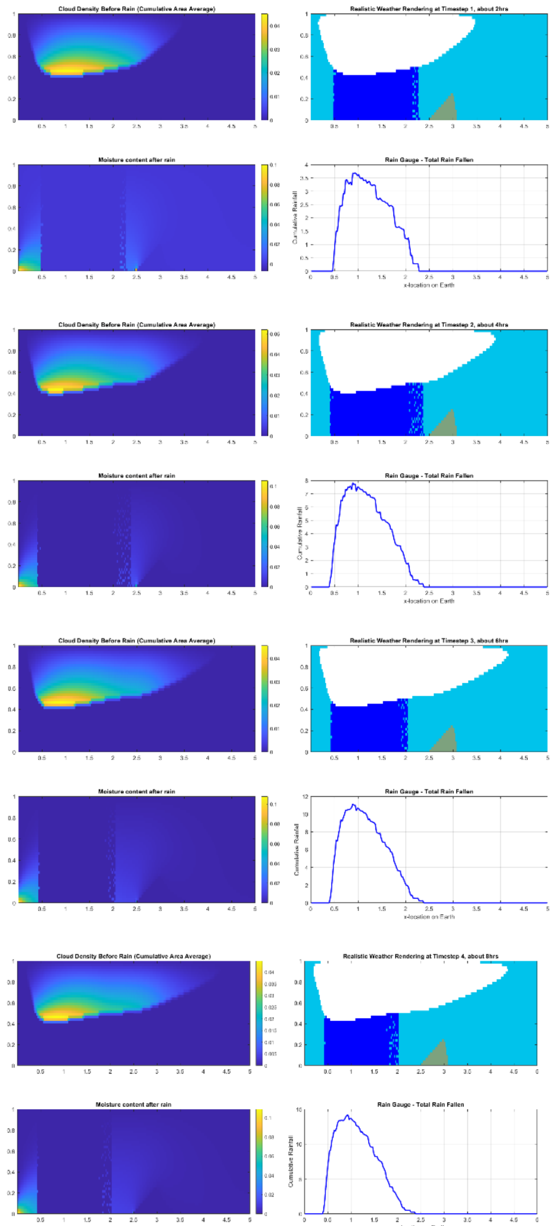
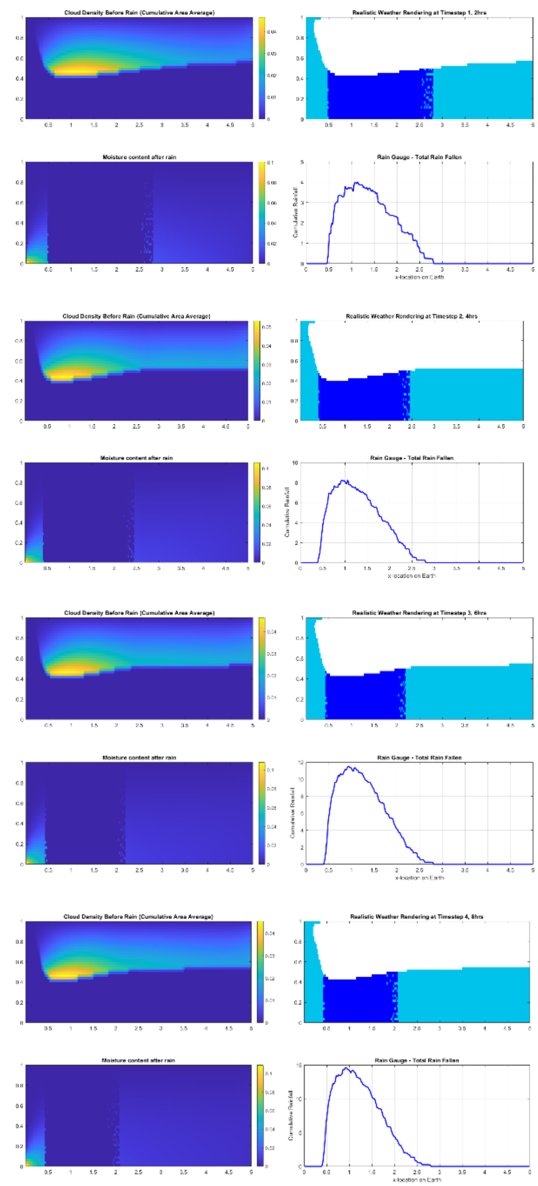
And by adding data labels, the values are definitely different, with a value able to be set on the boundary in the right (no-slip) setup.

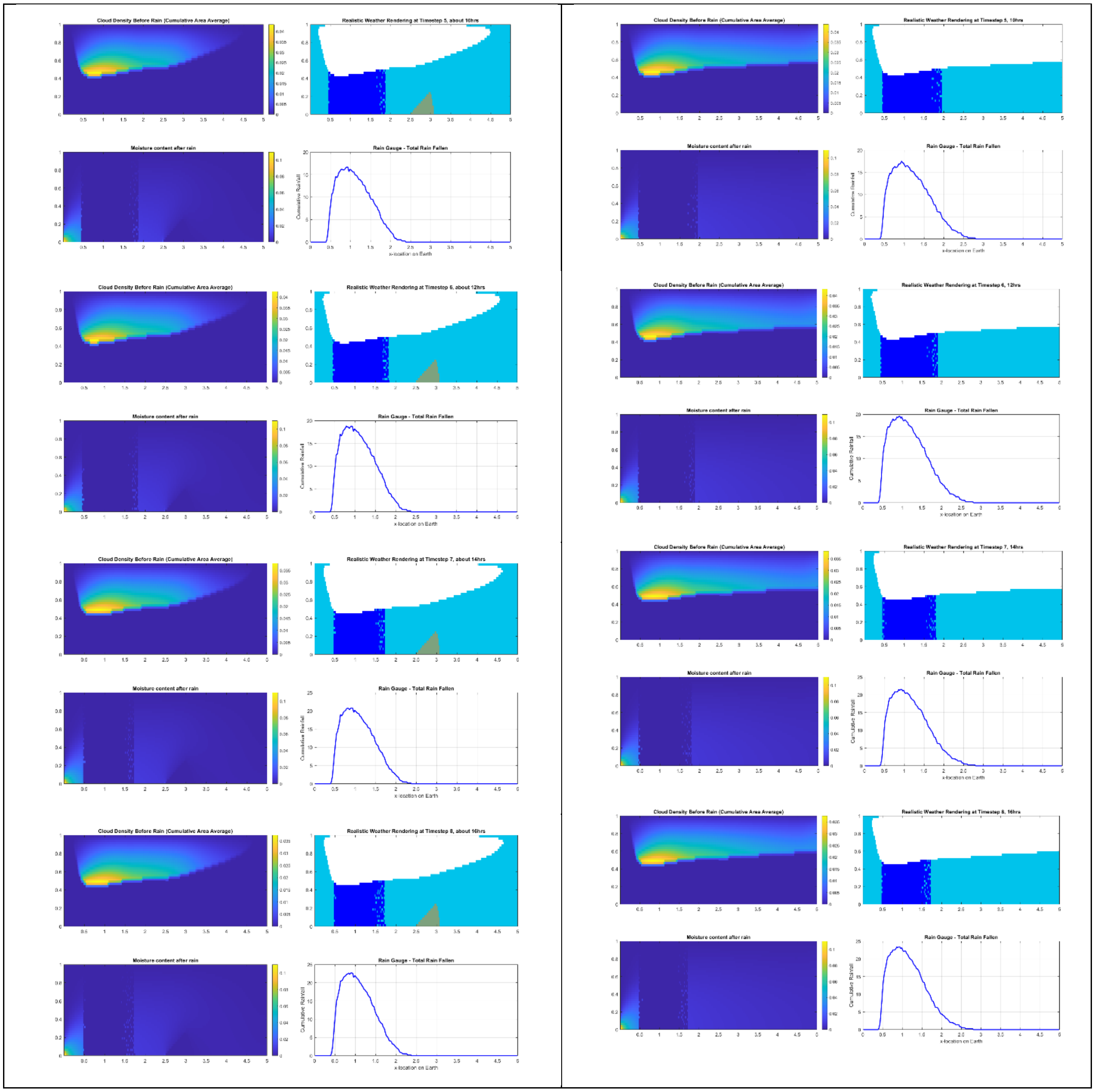


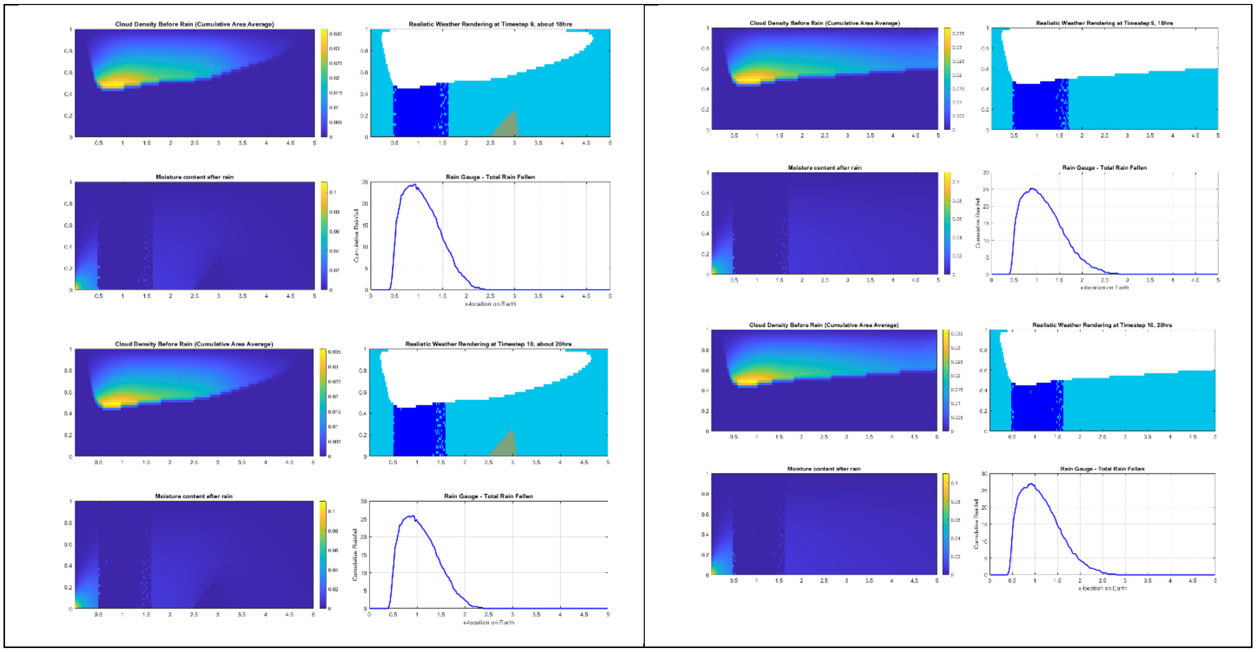
Closing Remarks

While it would have been more interesting to vary parameters of the rainshadow, resolution, time, geometry etc., the inordinate amount of time to complete even 10 timesteps is not worth the cost. These changes were chosen because they required only one different run. It is good practice in ensuring that code is complete before submitting a job, as is paramount in using high-powered computers.

To test the validity of the model as a rainshadow, one more simulation was run with no mountain. The upwind rain accumulation is not the most reassuring, but the mountain does prove to condense the clouds and moisture together. Otherwise, the clouds do not reach the critical mass required to rain and just pass onto land uniformly. This would be the equivalent of a rainy shore, and then downwind, other patterns would disperse the moisture.

Mountain – Shrinking Cloud	No Mountain – Results in Uniform Cloud
	

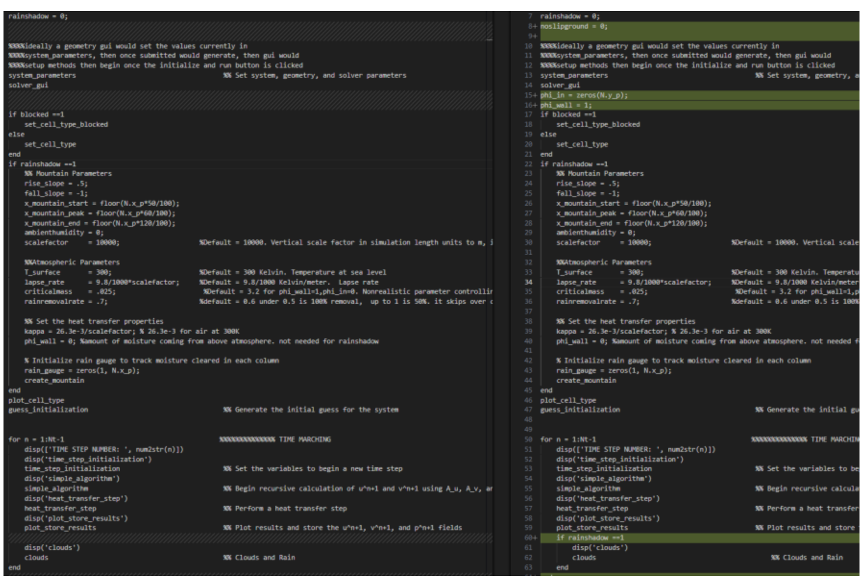

Cloud Density Before Rain (Cumulative Area Average)
Realistic Weather Rendering at Timestep 5, about 10hrs
Moisture content after rain
Rain Gauge - Total Rain Fallen
Cumulative Rainfall
x-location on Earth
Realistic Weather Rendering at Timestep 6, about 12hrs
Realistic Weather Rendering at Timestep 7, about 14hrs
Realistic Weather Rendering at Timestep 8, about 16hrs
Realistic Weather Rendering at Timestep 5, 10hrs
Realistic Weather Rendering at Timestep 6, 12hrs
Realistic Weather Rendering at Timestep 7, 14hrs
Realistic Weather Rendering at Timestep 8, 16hrs



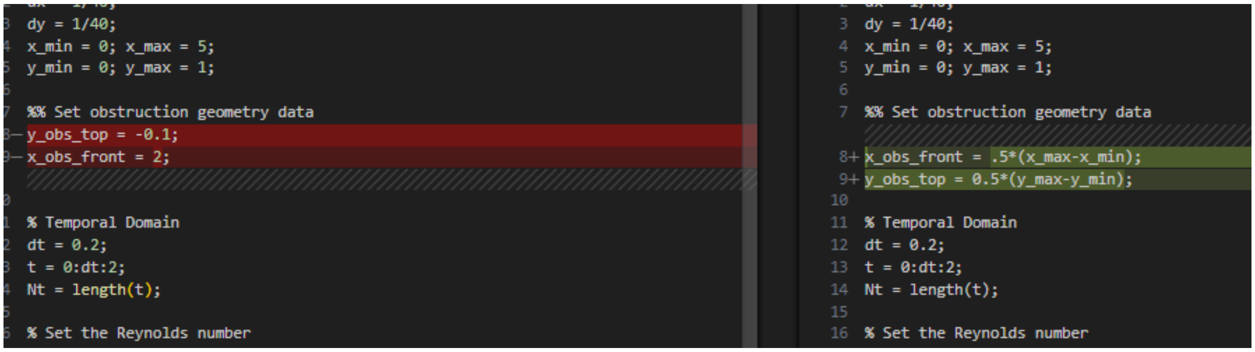
Bug Fix For Non-preset Option

In the base simulation, there were some bugs that prevented the user from successfully running a non-rainshadow simulation. Fixes were made as below and incorporated into the option_2 codebase.

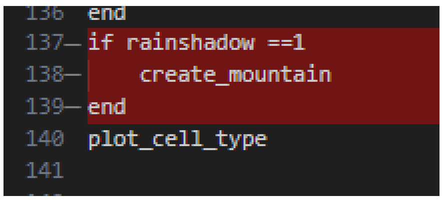
First, the main.m code needed to initialize the variables that were only meant for the clouds because they are referenced in shared functions:



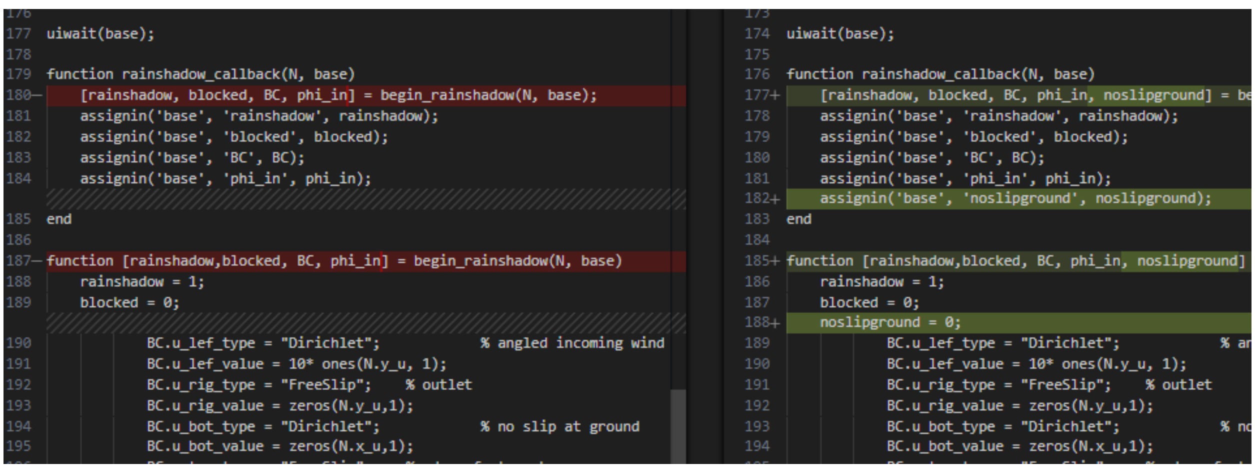
Next, the obstruction edges of the block were hard coded from the sample code and didn't even fit within the domain, so they were made dynamic:



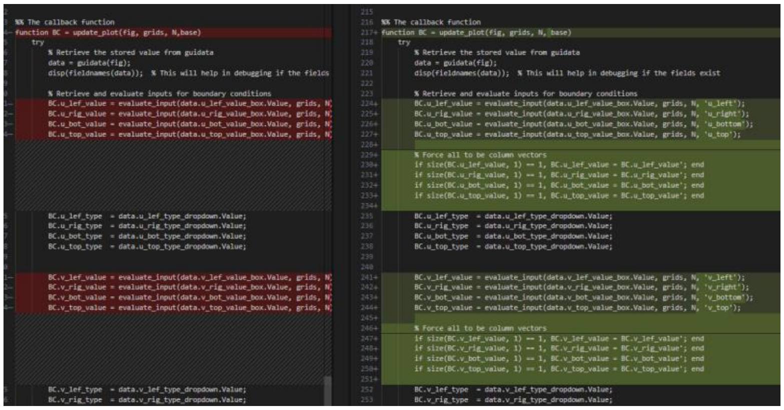
In solver_gui.m, this section was removed because rainshadow will never be set before the button is set.

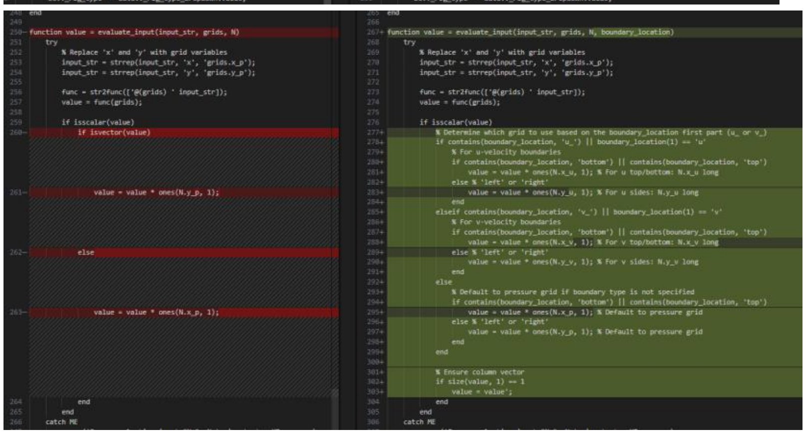


As noted above, the no slip ground variable needed to be included:

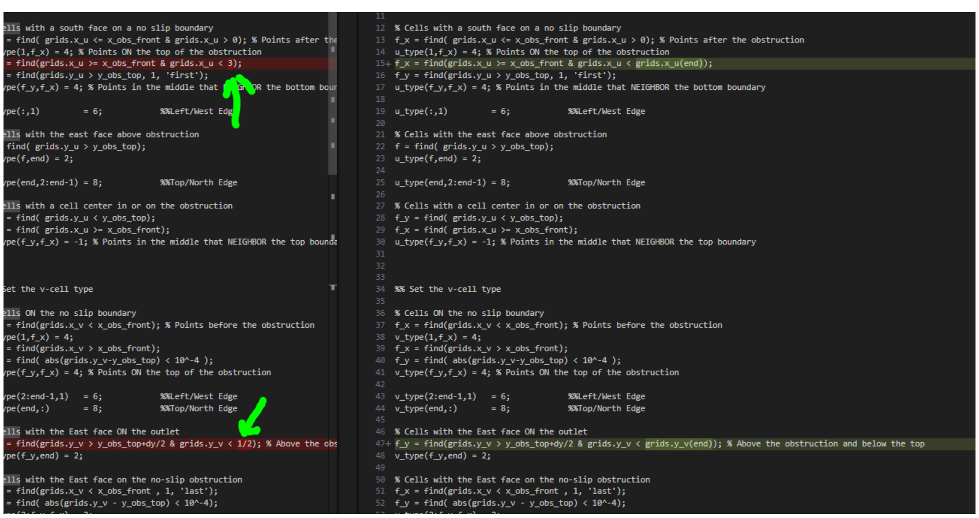


The final fix was that each input BC needed to adjust its length to whatever side it was applied to, so logic determines what length applied to which condition:

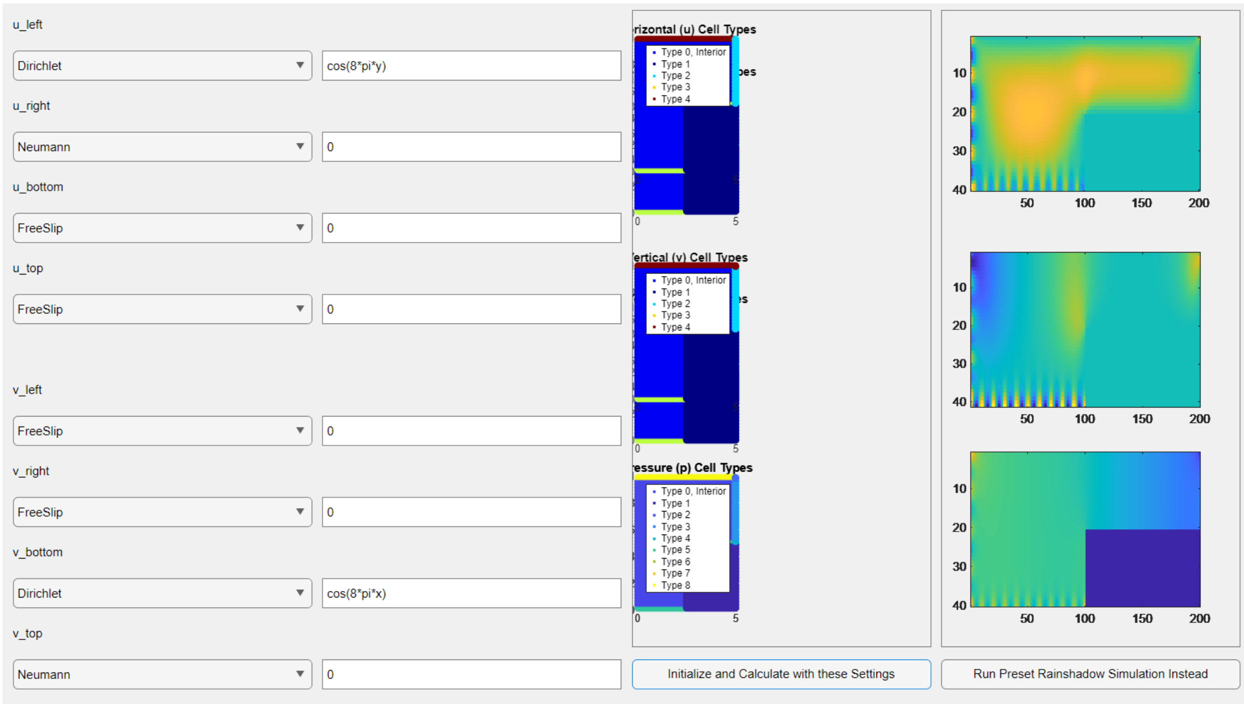




In the set_cell_type_blocked, there were also leftover hard-coded boundary conditions that were limiting the setting of most of the cells. This one was particularly annoying to find, but picked up by comparing a lot of different files:



The result is that the code can successfully simulate GUI input conditions:



Congratulations to Professor Michael Allshouse for his new position in Ohio! It was a pleasure taking classes with you and seeing you on the marathon course.