Design Document ADE_VelocityField.py

Section 1: setting the parameters

- Here we just assign values to the parameters for our solution. Note how the time step is calculated from the delta x and y.

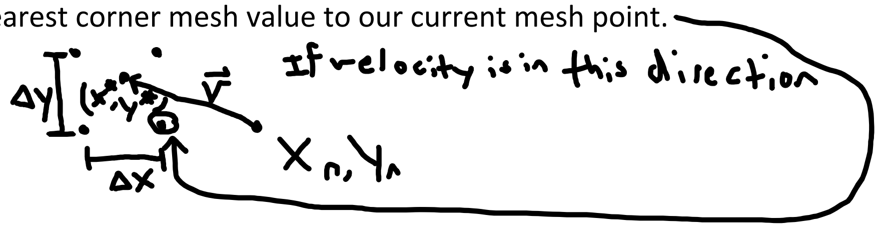
Section 2: defining the mesh class

- Here we are just creating a class for our meshes to inherit.

Section 3: math functions

- Subsection 3.1: the diffusion function “def Diffusion”
    - This begins with creating a 2D array to store our diffused values. Next in a nested for loop we calculate diffusion and store it in our 2D array “diffusion” which we subsequently return.

- Subsection 3.2: Method of Characteristics Function “def MOC”
    - We first define an x and y stepper which will be used later. Next we generate a 2D array to store out advection values from our method of characteristics function, which we will later return. Then in a nested for loop, we perform calculations of our x*. This is done by determining the direction, magnitude, and angle of the velocity vector at a given point. This magnitude is used to determine how far back the function reaches when performing the MOC by multiplying by the time step. Then, the x and y coordinates of this back step are determined by the “d_xstar and d_ystar” by using the magnitude and angle of the velocity vector. Finally the number of x and y steps are determined by rounding the d_xstar and d_ystar by the x and y steps and rounding down. By doing this, we always get the nearest corner mesh value to our current mesh point.



Now using the conditional statements that follow, we can determine which way to step from this nearest corner point to get all of the surrounding mesh point values. Then, except for at the edges, the function “characteristic_info” is called to calculate the advected value which is stored in our “u_pstar” 2D array.

- Subsection 3.3 Interpolating the Advected value based on surrounding points "characteristic_info"
    - To do this interpolation, we start by initializing a 2D array, this will store the a. concentration values, b. the distance between the mesh point and the x*, and c. Four weight values for interpolation based on these distances. The function stores values for concentration at the surrounding points based on the previous time step. Then is stores values for distances between x* and each of the respective mesh points. Each of these distances is then divided by the sum of all of the distances to get a weight, $0<w<1$, that will be used in interpolation. These weights are then multiplied to their respective mesh point's concentration value and all of these products are summed to get an advected concentration, u_pstar, which the function returns.

Section 4: directional, magnitude, angle check for MOC and defining the Velocity Field

- Subsection 4.1: direction, magnitude and angle calculation "def check_direction_and_magnitude"
    - This function uses the sign of the x and y components of the velocity vector to determine direction, then calculates the magnitude of the vector and the angle between it and the horizontal.
- Subsection 4.2: generation of the velocity field "def Velocity_Field"
    - Here we initialize a velocity field in a 2D array, with each array value being an array of 2 numbers, the x and y components of velocity. This function returns this field.

Section 5: initializing everything

- First we create a mesh of 0 concentration.
- Subsection 5.1: setting the initial condition " def initial_condition"
    - Here we set the initial condition in each point of the 2D mesh
- Subsection 5.2: creating the grid "def create_x_and_y_points"
    - This creates store for grid numbers, not sure how useful.
    - Then the initial condition field is created.

Section 6: generating the mesh

- The mesh object is created that inherits from the Mesh_Object class

Section 7: setting boundary conditions

- These for loops set the boundary conditions along the edge.

Section 8: Time-stepping

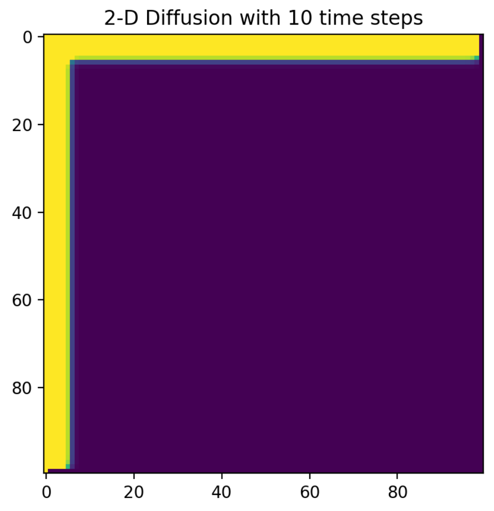
- First, the diffusion profile is calculated via the Diffusion function. Next the velocity field is created. Then the advected profile is calculated via the MOC function. Finally the values of each are added to the original mesh and the new mesh is created.

Section 9: plotting

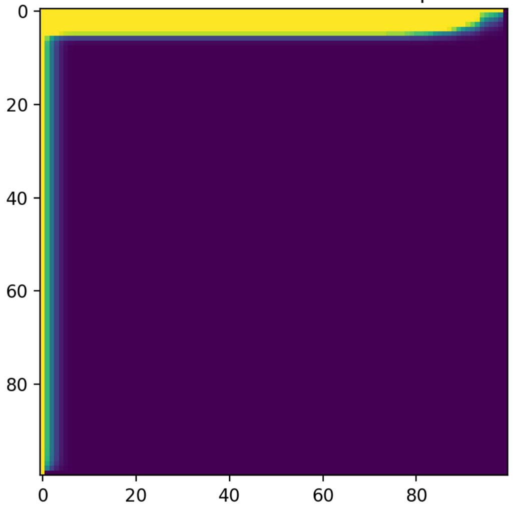
- Here the resulting field is plotted.

This code currently needs some ironing out, especially on edge cases where some additional conditional statements may be required. Below, there are some of the graphs created by the code with different velocity fields.

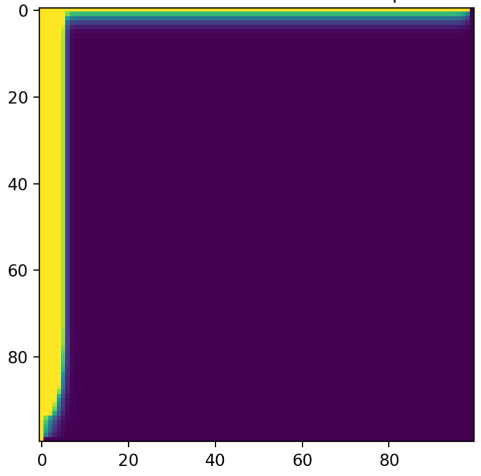
GRAPHS



2-D Diffusion with 10 time steps



2-D Diffusion with 10 time steps


0
20
40
60
80
0
20
40
60
80