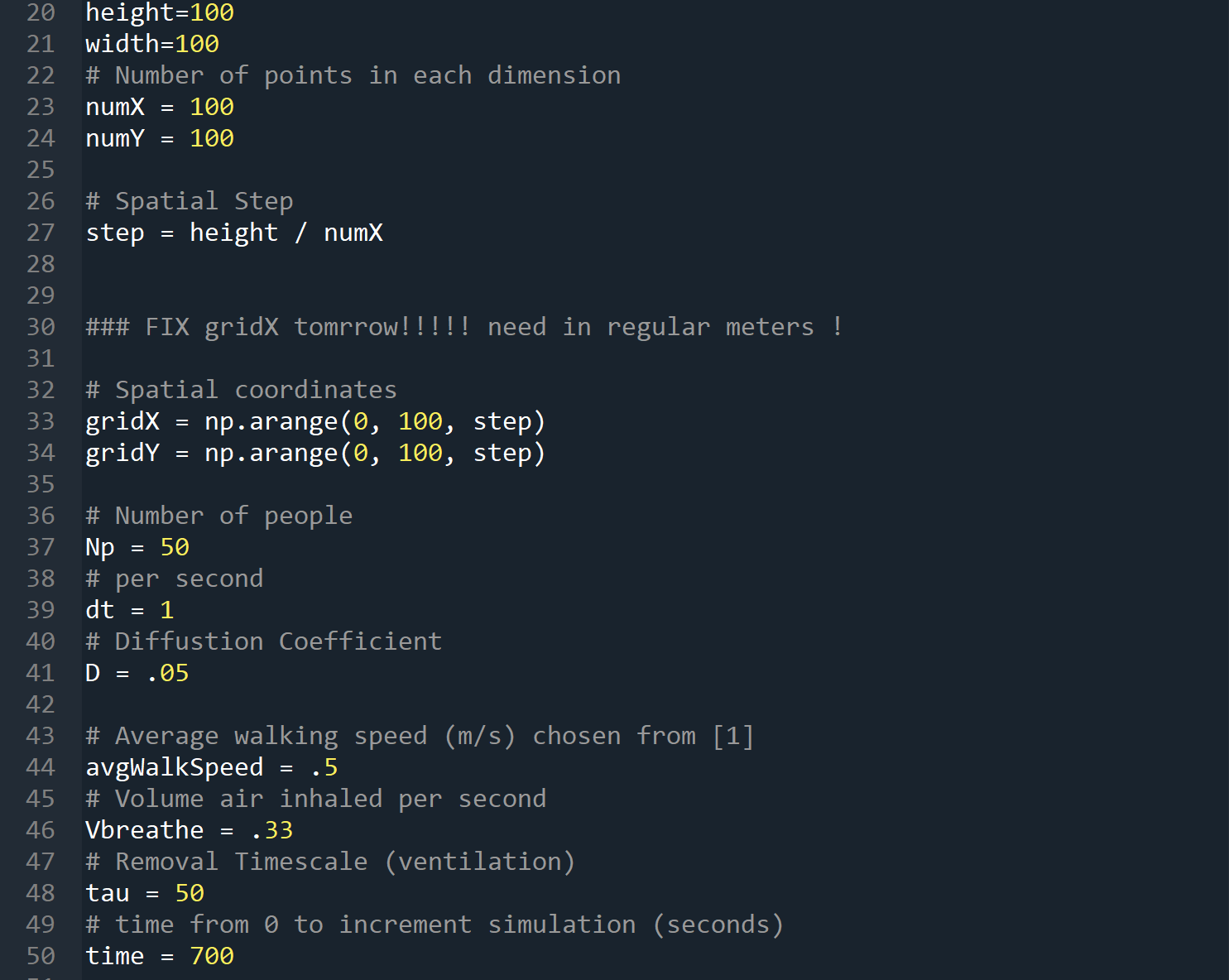
Jack Sullivan

Monte Carlo Simulation of SARS-CoV-2 Aerosol Transport via

Finite Difference Schemes

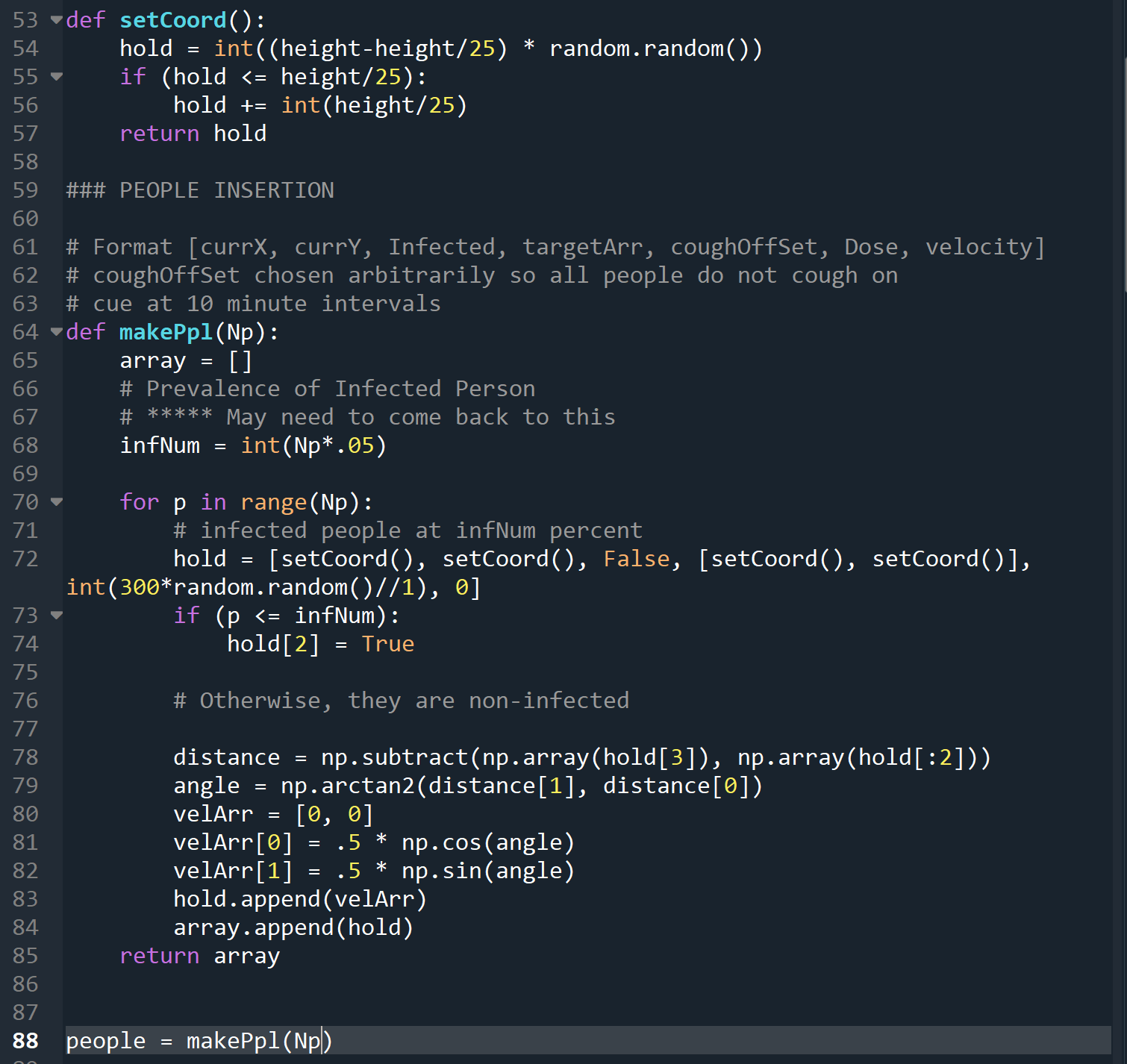
Vuorinen and an extensive team of researchers conducted several different types of simulations of SARS-CoV-2 aersol transport during the current pandemic. Many of these models were based on computational fluid dynamics softwares and too complex to replicate in a short timeframe. Here the Monte-Carlo modeling technique they used to study the spread of the virus based on parameters attained from CFD simulations were replicated in Python.

A series of dots, representing people, were made to move a room of certain dimensions around each second. At each time point, depending on whether or not they were infected, they either coughed or breathed normally (exhaled), or inhaled viral particles to accumulate a dose of the virus and eventually become infected. The infected spread the virus via a diffusion equation finite difference scheme that was central-difference in space and forward-difference in time. This simulation was represented in two dimensions, and

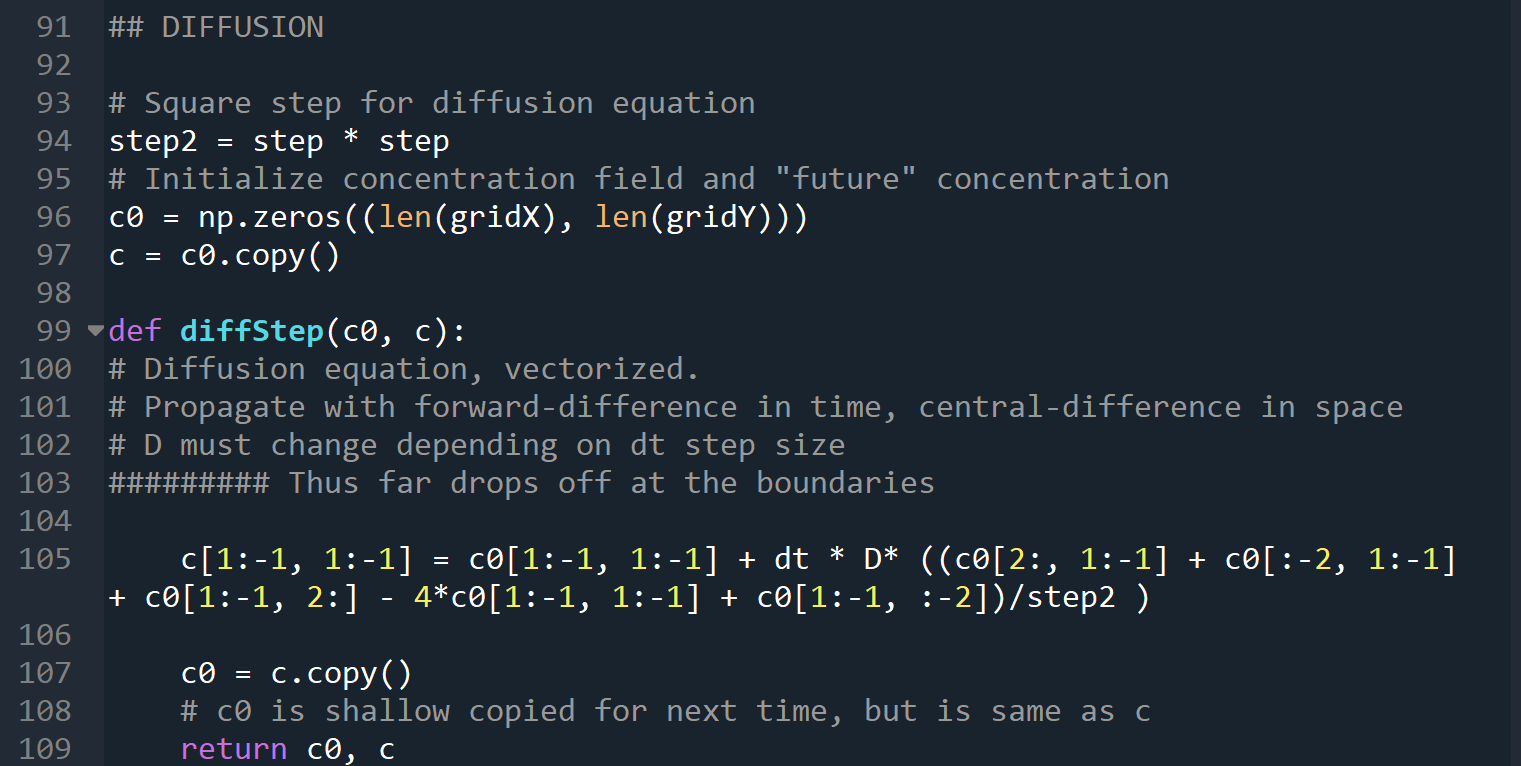


The spatial grid was constructed to be quantized, but that was eventually not used for the spatial coordinates of the “people” themselves, but to instead build the concentration field that the diffusion equation would manipulate.

The time and Np parameters were set arbitrarily, as they effected the outcome of the simulation but not the concentration of the viral particles in the air. Many of the other parameters were known from other simulations carried out by Vuorinen et al. The diffusion coefficient and timestep (dt), for example, had to be consistent with the known values. All values were measured in terms of meters and seconds, unlesss otherwise specified.



The setCoord( ) function created random coordinates inside of the “room” area and is widely used in the rest of the code. The makePpl ( ) function creates data for each person to follow at the various steps of the simulation. It sets initial random coordinates of the person(currX, currY), whether or not they are infected (Infected), where they are going (targetArr), when they should start coughing (coughOffSet), how much of the virus they hae accumulated if healthy (Dose), and their velocity components in the X and Y directions (velocity). It does this for the chosen number of people (Np).

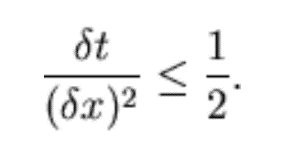


As discussed, the diffusion equation is forward-difference in time and central difference in space. They appropriate time step size (1s) was crucial in working with the diffusion coefficient that was derived from Vuorinen et al.’s earlier simulations. The above code takes advantage of underlying C code for numpy, but accomplished the same as the following central-difference method for solving the diffusion equation [2]:

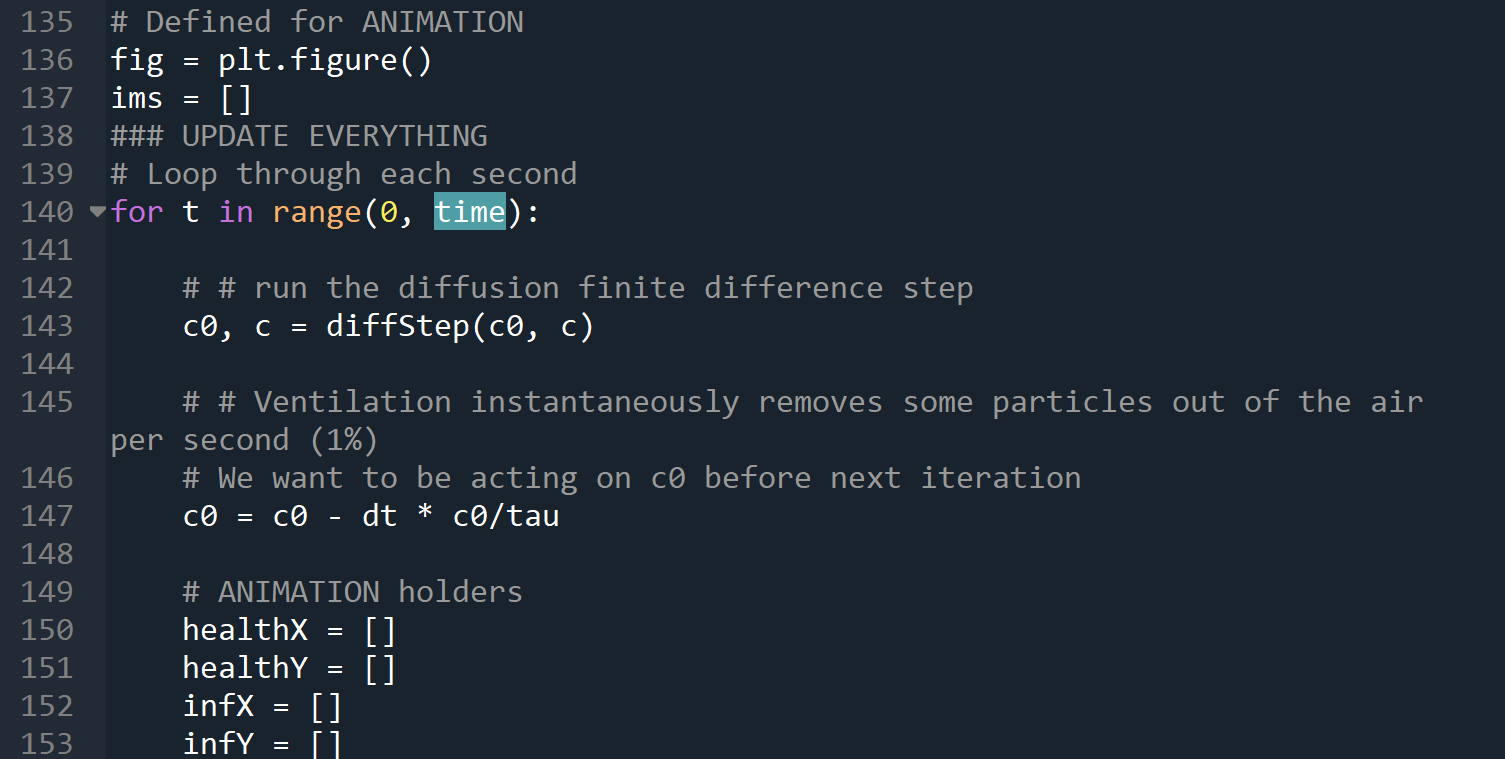
A picture containing diagram

Description automatically generated

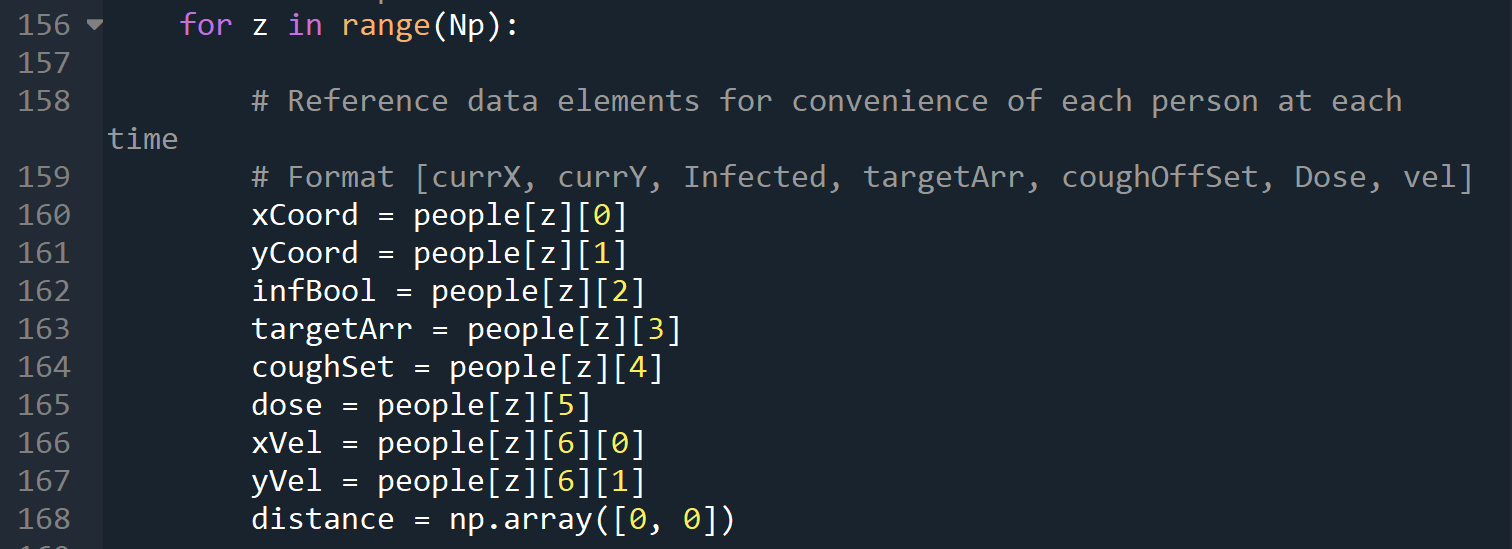
However [3] shows that the convergence of this method would seem to fail given the parameters used herin, where the scheme is only valid for:



And dt = dx^2 = 1. It appears that Vuorinen et al. used the same position and timesteps described herin, and this simulation appeared to still function, but some future investigation is needed to determine what changes a smaller step, in one case, will produce.



The first for loop increments time, and the diffusion equation in time as well as in the particles in the diffusion field in two spatial dimensions. Certain other elements can be observed that store data along the way for an animation of the entire simulation.



The second nested for loop (inside of the time loop) is to loop through each person at each time point, and to have them either add to the concentration of viral particles or inhale them, as well as to move closer to their respective targets before the next timestep. A future modification that needs to be implemented is the subtraction of the viral particles from the concentration field.

The above conditional logic increments the cough or breath if the infected Boolean shows that a person is a virus carrier, and if not, how they accumulate a dose and eventually become infected.

Text

Description automatically generated

The solving method used, from scipy, found the eigenvalues of the system of linear equations that the differential equation was cast into to solve for psi.

Graphical user interface, application

Description automatically generated

After running the post-processing script, the output is displayed, presumably as a probability map for the electron structure of Hydrogen at different energy levels. The solutions are given by the terminal:

Text

Description automatically generated

Where the values in the FEM column are the values of the coefficients in the Ritz-Galerkin approximation.

The solver function was changed to a Newton-Raphson method from a different code, but the result seemed meaningless compared to the earlier one:

Shape

Description automatically generated

It did not produce results for the different Eigenvalues, and it was unclear how to adjust it to the quantum well code or if this was possible.

# Sources

1. Ville Vuorinen, M. A. (2020). Modelling aerosol transport and virus exposure with numerical simulations in relation to SARS-CoV-2 transmission by inhalation indoors. *Safety Science, 130*. Retrieved from http://www.sciencedirect.com/science/article/pii/S0925753520302630

*2. The two-dimensional diffusion equation.* (n.d.). Retrieved from scipython.com: <https://scipython.com/book/chapter-7-matplotlib/examples/the-two-dimensional-diffusion-equation/>

3. Moehlis, J. M. (2001, October 24). *Solution of the Diffusion Equation by Finite Differences.* Retrieved from me.ucsb.edu: https://sites.me.ucsb.edu/~moehlis/APC591/tutorials/tutorial5/node3.html