Byron Becker ID #104459657

COEN 1300 Section 408

**Particle Tracking**

In this project, we are asked to model the movement of one thousand particles as they move through an aquifer towards a well. The reason that the position of these particles over time is important, is that in the case of a leak of trichloroethylene, a contaminant, the position of these particle contaminants must be known in order to know when to shut off the flow of the aquifer to the well, keeping the contaminants out of the drinking water. The movement of these particles varies based on their diffusivity constants(D), and velocities(v), which can be tracked at variable time increments(dt).

For this project, I created two functions and one script, each of whose full descriptions is commented into their code. The first, called Particle\_tracking\_function, takes input diffusivity constants, velocities and time increments between t = 0 and 0.5 seconds and tracks the x and y positions of 1000 particles throughout this time. For example, Particle\_tracking\_function(1,20,0.001) would correspond to a D =1, v=20, and dt =0.001 seconds. I additionally created a script of this function called Particle\_tracking\_script that requires no inputs, as D, v, and dt are defined in the script for D = 1, v = 20, and dt = 0.001 seconds. After 20 to 30 seconds of calculation, the function displays an animation of the x and y positions of these particles over time, as well as the location or region of the well. From this, one can interpret visually when the particles will enter the well, and how many will be there. After the animation, it returns a matrix with the x and y positions for each particle at each time increment. The first 1000 rows pertain to the x values of particles numbered 1-1000, and the next 1000 rows(row 1001-2000) pertain to the y values of these same particles. The columns are organized so that each new column represents an incremental increase in time, in this case, every 0.001 seconds.

For reference, I have included the equations of position of the particles below:

xi+1 = xi + (2DΔt)^1/2 ∗rand +υΔt

yi+1 = yi + (2DΔt)^1/2 ∗rand

From testing this function at different diffusivity constants and velocities, I found that the D value corresponds to the variability of the particle in both the x and y directions, which makes sense as the equations for the x and y position values are proportional to the square root of D multiplied by a random number from a normal distribution (mean = 0, sigma = 1). Therefore, I found that *as D increased, the variability of particle position increased over time, and vice versa, to the point that if D = 0, all the particles take the same path*. I also found that *as v increased, the x position of the particle increased at a faster rate, corresponding to faster movement or velocity* (as the variable is titled) through the aquifer.

The second function I created, called Calc\_conc, is also a function of D, v, and dt, and should be run as Calc\_conc(D,v,dt). This function corresponds directly to the problem statement of knowing when to shut off the well. It returns a vector called number\_in\_region, which contains the number of contaminants in the well region over a period of time, as well as graphing the number of contaminants in the well region as a function of time. Based on this information, one can determine when they want the aquifer shut off based on the number of particles in the region at a certain time. According to the problem statement, the well should be turned off when the number of contaminants in the region = 10. From this function, it was determined graphically that for D = 1, v = 20, and dt = 0.001, *the well should be shut of at approximately t = 0.15, and turned back on near t = 0.39*. While the number\_in\_region vector first exceeds 10 at t = 1.166 and last exceeds it at t = 1.368, I have added a slight amount of buffer room in case of extremes in the random component of the equation.

In this project, I learned how to efficiently catalog and analyze large sets of data. By making nested large for loops for this data, I learned that MATLAB works much more efficiently if as many operations as possible are done first are done outside of the loop, and as many matrix operations are utilized as possible, instead of nesting one for loop inside of another. Additionally, I learned how to use the pause feature with a loop to animate a plot smoothly over a loop.