CS279 Assignment 3

Code Tutorial

Section 3: Diffusion Simulations

Question 10

- Starting from the origin: At each time step, the particle can move either left (-1) or right (+1) with equal probability — 50% chance for each
- Probability distribution: list that shows all possible outcomes and how likely each one is
  - Example: t = 1
- Expected position: where the particle is likely to be at a certain point in time
- Expected squared displacement: how far the particle is from the start, on average, in terms of the square distance

3 Diffusion Simulations

Once we have a model for the structure of a given type of cell, we frequently want to model the dynamics within the cell. Next, we will look at how biological molecules diffuse throughout cells.

Diffusion is the process by which particles spread out and mix together with other particles. Fundamentally, diffusion is governed by the random motion of individual particles. However, by combining many particles' random motions, we can derive deterministic laws that govern the aggregate movement of a substance through solution.

Let's start by thinking about motion in one dimension.

Question 10: *Assume there is a single particle on a number line, starting at position 0, and at each time step it moves left or right by 1 unit with equal probability.*

(a) Give the probability distribution over positions at $t = 4$ and $t = 5$. In other words, list the probability of the particle being at each possible position, at $t = 4$ and $t = 5$ respectively. A two-column table (one column of position, one column of probability) has historically worked well.
Extra Credit: Give a general expression for the probability of the particle being at position $x = x'$ at time $t = t'$.

(b) What is the expected position x at time $t = t'$? What is the expected squared displacement after t' time steps?

***Hint**: The **Diffusion and Cellular-Level Simulation** slides could be helpful here.*

Now, let's consider what happens when we add more particles.

Probability distribution: t = 1

position	probability
-1	50%
0	0%
1	50%

Question 11

- n(x, t): try to move from the initial probability table (where the starting point was defined) to a more general expression of one time step relative to where you're currently positioned
- Flux (J) = amount of substance passing through a unit area per unit time
- A concentration gradient can lead to flux, which then results in diffusion

Question 11: *Under the same model as the previous question, assume we have 1000 particles, all starting at position 0.*

(a) At the first time step, what is the expected number of particles that pass from $x = 0$ to $x = 1$?

(b) Let $n(x, t)$ be the number of particles at position x at time t. After 1 time step, what is the expected number of particles that pass from $x = x_0$ to $x = x_0 + 1$? What is the expected number of particles that pass from $x = x_0 + 1$ to $x = x_0$? And combining these two numbers, what is the net movement of particles from $x = x_0$ to $x = x_0 + 1$?
Your answers should be expressed in terms of $n(x, t)$.

This result underpins "Fick's Law," which states that the flux (the net movement of particles across a unit area per unit time) is proportional to the concentration gradient.

$$J = -D\frac{\partial C}{\partial x}$$

where J is the flux, C is the concentration, and D is a diffusion coefficient and is given in units of $(\text{distance})^2$ per time.

While it is important to understand that particles move from high concentrations to low concentrations in space — and that this fact can be derived from a simple random motion model — what we'd really like is a way to express the change in concentrations over time. If we assume that particles are neither lost nor created, then the change in concentration over time must equal the change in flux over distance.

$$\frac{\partial C}{\partial t} = -\frac{\partial J}{\partial x}$$

Using this fact and substituting our first expression for J, we can derive the diffusion equation, which states that the change in concentration over time is proportional to the change in concentration *gradient* over space.

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2}$$

Thus, at a given point in space, the change in local concentration over time is proportional to the degree to which the concentration gradient changes spatially.

Question 12

- Constant concentration gradient: the concentration increases or decreases by the same amount for every unit of distance you move in the system
- **Hint**: consider all three concentrations (i.e., what does a constant gradient mean for the value, J, of flux?)
-

This result underpins "Fick's Law," which states that the flux (the net movement of particles across a unit area per unit time) is proportional to the concentration gradient.

$$J = -D\frac{\partial C}{\partial x}$$

where J is the flux, C is the concentration, and D is a diffusion coefficient and is given in units of $(\text{distance})^2$ per time.

While it is important to understand that particles move from high concentrations to low concentrations in space — and that this fact can be derived from a simple random motion model — what we'd really like is a way to express the change in concentrations over time. If we assume that particles are neither lost nor created, then the change in concentration over time must equal the change in flux over distance.

$$\frac{\partial C}{\partial t} = -\frac{\partial J}{\partial x}$$

Using this fact and substituting our first expression for J, we can derive the diffusion equation, which states that the change in concentration over time is proportional to the change in concentration *gradient* over space.

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2}$$

Thus, at a given point in space, the change in local concentration over time is proportional to the degree to which the concentration gradient changes spatially.

Question 12: *Assume that concentration changes linearly across a system, such that the concentration gradient is constant: $\frac{\partial C}{\partial x} = k$ for some constant k. How does the concentration throughout the system change over time? Explain this macroscopic phenomenon in terms of the motions of individual particles of the system.*

Exercise 9

- random() generates a random float between 0 and 1
- To run: python diffuser.py stoc
- Ctrl+C in terminal to stop
- Step = essentially. How far the particle moves with each iteration (determined by diffusion constant)

```
# Returns the correct x index after moving step blocks
# to the right (to the left for negative step). Accounts for
# the "wrap-around" at the border.
# Note: When calling nextX() in your update() function,
#   you should use self.nextX(x, step)
def nextX(self,x,step):
  return int((x+step)%self.nX)
# Returns the correct y index after moving step blocks
# up (down for negative step). Accounts for
# the "wrap-around" at the border.
# Note: When calling nextY() in your update() function,
#   you should use self.nextY(y, step)
def nextY(self,y,step):
  return int((y+step)%self.nY)
```

In the `StochasticDiffuser`, you will fill in the number of particles at each block (cell) for the next state (`newBlocks`) by iterating through the current cell state and randomly assigning a new direction for each particle to move.

In the `LaplacianDiffuser`, you will update the "concentration" of particles deterministically based on the discrete second derivative of the concentrations (also called the Laplacian).

Before you get started, carefully read over the code in `diffuser.py`. You should understand what `self.blocks` represents and where the initial simulation parameters are set, as well as how you would use the functions `self.nextX()` and `self.nextY()`.

Exercise 9: *Implement* `update()` *in* `StochasticDiffuser` *in* `diffuser.py`.

```
class StochasticDiffuser(Diffuser):

  def update(self):
    # First, we will initialize a new blocks array with the same dimensions.
    # All values in new blocks are set to 0, which you will update in your code below.
    newBlocks = np.zeros((self.nY, self.nX))

    # Next, we will iterate through the coordinates of our cell,
    # so that we can then iterate through each particle at that coordinate.

    for row in range(self.nY):
      for col in range(self.nX):
        numberOfPart = self.blocks[row, col]
        # The delta/step size depend on the diffusion constant which is set in
        # the main method
        delta = ceil(self.diffusion)

        # Now we will iterate through every particle in each block
        for part in range(int(numberOfPart)):
          # Introduce some randomness in step size
          stepSize = delta - (random() > 0.7)

          # TODO:  YOUR CODE HERE
          # Fill in this section.
          # The particle should choose left, up, right, or down with equal probability.
          # Hint: use the random() function, which will return a random float number between 0 and 1.
          #
          # After you've chosen a random direction,
          # "add" a particle to the newBlock coordinate that is "stepSize" away in the chosen direction.
          #
          # Note: To achieve the "wrap-around" effect it will be helpful to use
          # the self.nextX() and self.nextY() methods.

          pass # this is a placeholder; remove this line once you start implementing
```

Exercise 10

- h = 1 simplifies the ddx, ddy implementations
- ddx[row, col]: difference in concentration between the current block and the block immediately to its right
- ddy[row, col]: difference in concentration between the current block and the block immediately above it
- Same principles apply for lap_x and lap_y

Exercise 10: *Implement* `update()` *in* `LaplacianDiffuser` *in* `diffuser.py`.

Hint: *To derive the Laplacian, we will approximate the appropriate derivatives using finite differences (i.e.* $\frac{dF}{dx} \approx \frac{f(x+h)-f(x)}{h}$*). Specifically, you will approximate the second derivative of concentration with respect to* x $(\frac{d^2C}{dx^2})$ *or* y $(\frac{d^2C}{dy^2})$*. This means that you will also approximate the first derivatives,* $\frac{dC}{dx}$ *and* $\frac{dC}{dy}$*. Assume that the values stored in self.blocks represent concentrations.*

These approximations can be numerically unstable if both the first and second derivatives are calculated using the same interval, i.e. $\frac{df}{dx} = \frac{f(x+h)-f(x)}{h}$ *and* $\frac{d^2f}{dx^2} = \frac{df_x(x+h)-df_x(x)}{h}$ *are using the same interval between* x *and* $x+h$*. To avoid instability, you will need to balance the derivatives so that they are calculated using different intervals, i.e.* $\frac{df}{dx} = \frac{f(x+h)-f(x)}{h}$ *and* $\frac{d^2f}{dx^2} = \frac{df_x(x)-df_x(x-h)}{h}$*.*

```
class LaplacianDiffuser(Diffuser):

  def update(self):
    """
    Simulates one step of Laplacian Diffusion.
    """
    # First, we will initialize the derivative matrices, with all values set to 0
    ddx = np.zeros((self.nY, self.nX))
    ddy = np.zeros((self.nY, self.nX))
    ddt = np.zeros((self.nY, self.nX))

    # Next, we will iterate through the coordinates of self.blocks
    # to compute the first discrete derivatives (change in particles with respect to x or y).
    for row in range(self.nY):
      for col in range(self.nX):

        # TODO: YOUR CODE HERE
        # Compute a discrete (first) derivative in x and y.
        # i.e., fill in ddx, ddy
        # Again, self.nextX() and self.nextY() will be helpful.
        # Hint: The formula for approximating the first derivative is df/dx = (f(x+h)+f(x))/h. Note
        # that since we're simulating one time step, the interval h = 1.
        pass


        # END YOUR CODE HERE

    # Finally, we will once again iterate through the coordinates of self.blocks to compute the second discrete derivatives (ie the change in the first derivative with respect to x or y)
    for row in range(self.nY):
      for col in range(self.nX):
        # TODO:  YOUR CODE HERE
        # Compute the discrete second derivative in x and y,
        # i.e. the derivative of the derivatives you computed above, and
        # store the result in the lap_x and lap_y variables, respectively.
        # Make sure to balance your discrete derivatives so that the are not
        # using the same interval over x or y for both derivatives. (see assignment sheet for details)
        #
        # Hint: The formula for approximating the second derivative is df/dx = (f(x)-f(x-h))/h, accounting for balancing.
        # Note that since we're simulating one time step, the interval h = 1.
        lap_x = None # fill in your own code
        lap_y = None # fill in your own code

        # END YOUR CODE HERE


        ddt[row, col] = self.diffusion*(lap_x + lap_y)

    # Update the changes in concentration
    for row in range(self.nY):
      for col in range(self.nX):
        self.blocks[row, col] += ddt[row, col]
    return self.blocks
```

Question 13

- To run initial conditions:
  - Uncomment desired condition/s
  - Python diffuser.py stoc
  - cmd+/ works to comment/un-comment large blocks of code
- To edit the plotting script as in 13c:

```
def main():
  usage = "Usage: python diffuser.py [stoc | lap]"
  cm = "cool"

  iters = 1000 # number of iterations for diffusion
  length = 25 # size (x = y) of the plane

  if len(sys.argv) < 2:
    print(usage)
    exit(1)

  # EDIT DIFFUSION PARAMETERS below for Stochastic and Laplacian Diffusers
  if sys.argv[1] == "stoc":
    # play around with different values for diffusion
    # (it only makes sense to use integers here)
    diffusion = 1
    c = StochasticDiffuser(nX = length,
                           nY = length,
                           diffusion = diffusion,
                           )
  elif sys.argv[1] == "lap":
    # play around with different values for diffusion
    # (note what happens when diffusion > 0.25)
    diffusion = 0.24
    c = LaplacianDiffuser(nX = length,
                          nY = length,
                          diffusion = diffusion,
                          )
  else:
    print(usage)
    exit(1)
```

Question 13: *Run the* `StochasticDiffuser` *(*`python diffuser.py stoc`*) under a variety of initial conditions. You can locate pre-populated conditions at the end of* `diffuser.py`*, and un-comment the condition to run.*

(a) Include your code for `update` *in* `StochasticDiffuser`*.*
You may remove the comments inside the function if it takes up too much space.

(b) Start with the "Single Particle" simulation. How would you characterize the motion of the particle? ***Hint:*** *The scale bar is indicative of how many particles are in a block.*

(c) Next, run the "Point Mass" simulation. What does the state of the diffuser look like after 100 iterations? After 500?
Hint: *You can edit the plotting script to make it only plot once after N iterations. You may include screenshots if it helps your explanation, but this is not necessary.*

(d) Can you predict where a specific particle will be after many iterations? Can you predict the distribution of particles after many iterations?

(e) Convergence occurs when the diffuser reaches a state of equilibrium. Play around with the diffusion coefficient and starting number of particles. Do either of these parameters affect the rate of convergence? If yes, identify which parameter(s) affect the rate of convergence and describe how the rate of convergence changes.

```
""" Here are a variety of settings for the initial conditions of the
    diffusion simulations.  Feel free to try some of your own.
    Note: The units in StochasticDiffuser are number of particles.
          The units in LaplacianDiffuser are an aribitrary measure of concentration.
"""

############################################################
# EDIT INITIAL SIMULATION PARAMETERS BELOW.

"""SINGLE PARTICLE"""
# c.setBlock((length//2,length//2),1)

"""Point Mass"""
c.setBlock((length//2,length//2),1250)

"""1D Diffusion"""
#for i in range(length):
  #c.setBlock((i,length//2),125)

"""1D Gradient Diffusion"""
#for i in range(length):
#  for j in range(length):
#    c.setBlock((i,j), 2*abs(int(length/2-i)))

# END PARAMETERS
############################################################
```

Question 14

- Starting concentration: third argument in c.setBlock

Question 14: *Run the* `LaplacianDiffuser` *(*`python diffuser.py lap`*) with some of the diffusion conditions in* `diffuser.py`.

(a) Include your code for `update` *in* `LaplacianDiffuser`.
You may remove the comments inside the function if it takes up too much space.

(b) Play around with the diffusion coefficient and the starting concentration. Do either of these parameters affect the rate of convergence? If yes, identify which parameter(s) affect the rate of convergence and describe how the rate of convergence changes.

(c) Using the "Point Mass" simulation and starting with an initial concentration of $1,250$ *and a diffusion coefficient of* 0.24*, how many iterations does it take for every block (voxel) to have roughly an equal concentration of particles (i.e. to converge)? Note that our criterion for convergence is* $\Delta concentration < (\epsilon * initial\ concentration)$ *for all voxels, where* $\epsilon = 0.001$ *(about* $1/1,250$*). Provide a brief description of how you determined when the simulation converged.*
Hint: *You can do an approximate calculation using the scale bar to estimate the number of iterations needed for convergence.*
Note*: We will accept a wide-range of answers, given a sufficient justification.*

```
""" Here are a variety of settings for the initial conditions of the
    diffusion simulations.  Feel free to try some of your own.
    Note: The units in StochasticDiffuser are number of particles.
          The units in LaplacianDiffuser are an aribitrary measure of concentration.
"""

############################################################
# EDIT INITIAL SIMULATION PARAMETERS BELOW.

"""SINGLE PARTICLE"""
# c.setBlock((length//2,length//2),1)

"""Point Mass"""
# c.setBlock((length//2,length//2),1250)

"""1D Diffusion"""
for i in range(length):
  c.setBlock((i,length//2),125)

"""1D Gradient Diffusion"""
#for i in range(length):
#  for j in range(length):
#    c.setBlock((i,j), 2*abs(int(length/2-i)))

# END PARAMETERS
############################################################
```

Question 15 + 16

- Laplacian model of diffusion is deterministic and based on the diffusion equation
- Particle movement is random in the stochastic model

Question 15: *As time goes to $+\infty$, what is the probability that the Laplacian model has a block of concentration 0? What about in the stochastic model — is the probability positive or equal to 0? (Hint: Do NOT try to explicitly compute this probability. We are asking you to use your intuition based on how these two models work.)*

Question 16: *Based on the answer to the previous question, and any other observations you've made, when do you think it would be appropriate to use the Laplacian Model versus the stochastic model? What are the benefits and drawbacks to each?*