Modeling Animal Skin Pattern Formation: A Tutorial

Group 2.21
Ford Lascari
Mariam Marzouk
Jake McNeil

Link to GitHub repo: https://github.gatech.edu/mmarzouk3/CX4230_P2

Modeling Animal Skin Pattern Formation: A Tutorial

This tutorial illustrates two different ways of modeling pattern formation in biological systems, specifically the formation of animal skin patterns (also referred to as Turing patterns) over time. The two modeling techniques covered are:

- Modeling Technique 1: Cellular Automata
- Modeling Technique 2: Partial Differential Equations

Modeling Technique 1: Cellular Automata

A Definition for the Uninitiated:

Roughly speaking, "automata" (the plural of automaton) refers to the changing state of a system based on some input(s) and the previous state of said system. So, for example, consider a system at time *t*, whose state depends on the system at time *t-1* as well as some input *i*.

Cellular automata (henceforth referred to as CA) are a set of automata arranged along spatial grid (think checkerboard), whose states are simultaneously updated by a uniformly applied state-transition function that refers to the states of their neighbors. This kind of simultaneous updating is also referred to as synchronous updating.
The original idea of CA was invented and developed to aid in modeling complex systems. Specifically, CA can be used to describe self-reproductive and evolvable behavior of living systems. Because CA are very powerful in their ability to concisely describe nonlinear dynamics, they have been extensively utilized to model all kinds of phenomena, such as molecular dynamics, physical properties of materials, reaction-diffusion chemical processes, growth and morphogenesis of organisms, ecological interaction and evolution of populations, propagation of traffic jams, and social dynamics, to name a few.

.....

Here, we're interested in using CA to model how chemical reactions in animal skins lead to the formation of patters over time, such as the evolution of leopard spots.

For our purposes, we will assume a 2D space made of cells where each cell can be in either a passive (0) or active (1) state. A cell becomes "activated" if there are a sufficient number of active cells within its local neighborhood. However, at the same time, other active cells *outside* this neighborhood try to suppress the activation of the focal cell, though with relatively weaker influences than those from the active cells in its close vicinity. These dynamics are called short-range activation (SRA) and long-range inhibition (LRI). [1]

.....

This model can be described mathematically:

$$N_a = \left\{ x' \;\middle|\; |x'| \leq R_a \right\}$$
$$N_i = \left\{ x' \;\middle|\; |x'| \leq R_i \right\}$$
$$a_t(x) = w_a \sum_{x' \in N_a} s_t(x + x') - w_i \sum_{x' \in N_i} s_t(x + x')$$
$$s_{t+1}(x) = \begin{cases} 1 & \text{if } a_t(x) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

Here, R_a and R_i are the radii of neighborhoods for activation (N_a) and inhibition (N_i), respectively. Also, R_a < R_i, and w_a and w_i are the weights that represent their relative strengths. a_t(x) is the result of two neighborhood countings, which tells you whether the SRA dominates (a_t(x) > 0) or if the LRI dominates (a_t(x) ≤ 0) at location x. [1]

.....

To model the formation of animal skin patterns over time, the first step is to decide the size of our 2D CA "checkerboard" space. The board is preset to 50x50 cells.
Feel free to change this by changing the values of `width` and `height`.

```
width = 50
height = 50
```

Next, we need to decide the initial probability that will determine whether a cell starts out as active or passive. `initProb` is preset to 0.5, which means that a cell has a 50% chance of starting out either active (black) or passive (white). Feel free to change this.

```
initProb = 0.5
```

Next, define `Ra`, `Ri`, `Wa`, and `Wi`. These are the radii of activation and inhibition, and the weights for each, as described above. Reasonable values are as follows (but feel free to try your own):
Ra = 1
Ri = 5
Wa = 1
Wi = 0.1

Notice that we import `pylab`, which loads `numpy` and `matplotlib` and allows them to work interactively. This is especially useful for the visualization portion of the model.

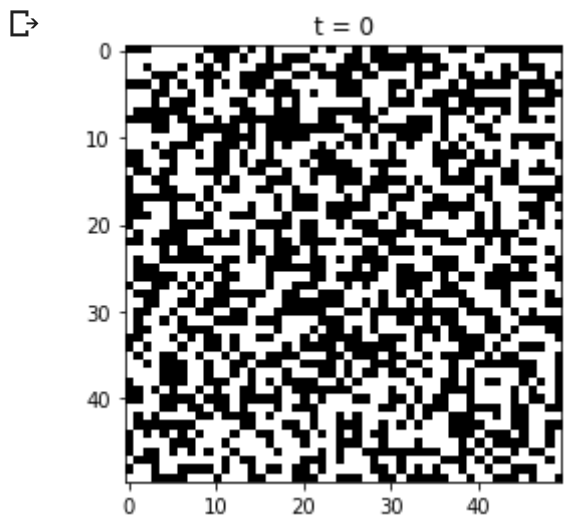
```
from pylab import *

##Edit the values below to observe different behaviors.##

Ra = 1   #radius of activation
Ri = 5   #radius of inhibition
Wa = 1   #weight represeting relative strength of activation
Wi = 0.1 #weight representing relative strength of inhibition
```

Next, let's create an `initalizeSystem()` function to set up the initial state of our system.

```
#Source [2] was referenced.

def initializeSystem():
    global time, currConfig, nextConfig    #the time of our system, the current configuration, and the next configuration

    time = 0

    currConfig = zeros([height, width])    #make all cells passive (white) to start
    for column in range(width):
        for row in range(height):
            if random() < initProb:        #random() returns a probability. This chunk of code runs for each cell
                state = 1                  #in our checkerboard grid and either "activates" it (turns it black,
            else:                          #which corresponds to an animal skin spot / pattern) or leaves it
                state = 0                  #passive.
            currConfig[row, column] = state

    nextConfig = zeros([height, width])
```

Let's see what our system looks like so far. Run the following code:

```
#Displays the current state of our system
def showState():
    cla()
    imshow(currConfig, vmin = 0, vmax = 1, cmap = cm.binary)
    axis('image')
    title('t = ' + str(time))

initializeSystem()
showState()
```



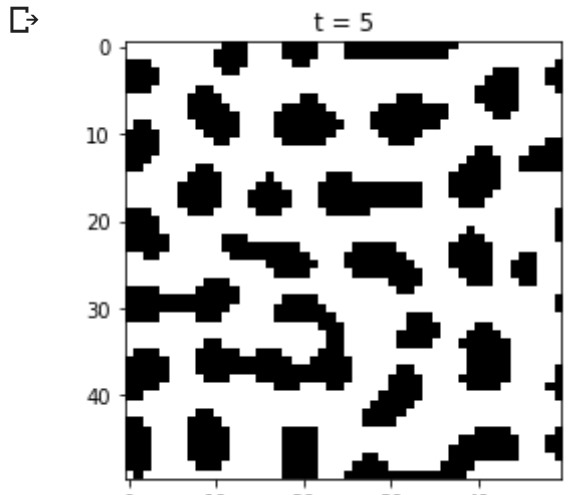
So far so good. Now we need a way to update our system so we can observe how it changes (how the animal skin pattern evolves) over time.

```
def updateSystem():
    global time, currConfig, nextConfig

    time += 1                                #increment time (note the discrete way time is being modeled)

    for x in range(width):                   #iterate over the columns
        for y in range(height):              #iterate over the rows
            state = currConfig[y, x]         #set state of the cell to the current configuration
            na = ni = 0                      #neighborhoods of activation and inhibition are initialized to 0


            for dx in range(- Ra, Ra + 1):
                for dy in range(- Ra, Ra + 1):
                    na += currConfig[(y+dy)%height, (x+dx)%width]      #set the boundaries of the neighborhood of activa
            for dx in range(- Ri, Ri + 1):
                for dy in range(- Ri, Ri + 1):
                    ni += currConfig[(y+dy)%height, (x+dx)%width]      #set the boundaries of the neighborhood of inhibi


            if (na * Wa) - (ni * Wi) > 0:
                state = 1                     #If SRA dominates, activate the cell; otherwise LRI dominates
            else:                             #and the cell is passive.
                state = 0
            nextConfig[y, x] = state          #the next configuration starts where we left off

    currConfig = nextConfig                       #update the system!
    nextConfig = currConfig
```

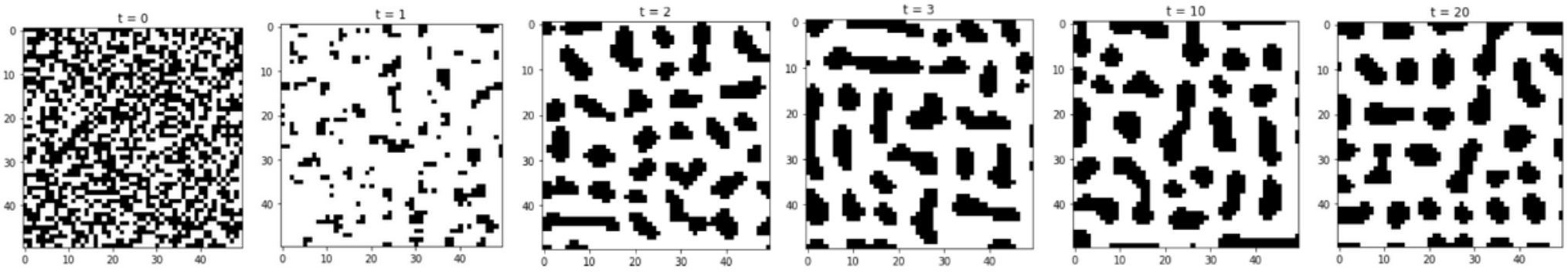
We now have all the pieces we need to model our system!

You can experiment further by going back and changing the values of `Ra`, `Ri`, `Wa`, `Wi`, and `initProb`. Run the below block of code with those different values and see how the behavior of the system changes!

```
initializeSystem()
while time < 5:     #change the time value to observe the system at different times
  updateSystem()
showState()
```



The below images shows the state changes at times 0,1,2,3,10, and 20.



By now you should have a good idea of how cellular automata can be used to model animal skin pattern formation. As mentioned earlier, there are myriad ways to model Turing patterns; the next modeling technique we'll explore is partial differential equations (PDEs).

PDEs can be used to model spatio-temporal dynamical systems, which makes PDEs a particularly good modeling technique for our purposes. PDEs abound in models describing heat transfer, sound, fluid dynamics, and electromagnetism, to name a few. Of particular interest to us is the power of PDEs in modeling reaction-diffusion interactions in biological systems. A reaction-diffusion system describes how different variables

may evolve subject to reaction and diffusion. Here, *reaction* refers to the transformation of the variables into each other, while *diffusion* refers to their spreading across a spatial region.

For our model, we'll use PDEs referred to as the FitzHugh–Nagumo equations to simulate the interaction of two chemical substances infuencing pigmentation (and thus the formation of animal skin patterns) [3] [4] [5].

Modeling Technique 2: Partial Differential Equations

We will be simulating Turing patterns using the finite difference method. This method discretizes time and space and replaces the derivatives with their discrete equivalents.

1) First lets start with imports. Like before, we're using `numpy` and `matplotlib`.

```
import numpy as np
import matplotlib.pyplot as plt
import math
%matplotlib inline
```

2) We will be simulating the following system of differential equations:

$$\frac{\partial u}{\partial t} = a\Delta u + u - u^3 - v + k$$
$$\tau\frac{\partial v}{\partial t} = b\Delta v + u - v$$

These will be simulated over the domain $E = [0, 1]^2$.
The variable u represents the concentration of a substance favoring skin pigmentation, and v represents another substance that reacts with u and impedes pigmentation. The pigmentation and impedence of pigmentation interact to form animal skin patterns!

At initialization time, we assume that u and v contain independent random numbers on every grid point. We also take Neumann boundary conditions: we require the spatial derivatives of the variables with respect to the normal vectors to be null on the domain's boundaries.

3) Now we define the 4 parameters of the model.

```
a = 2.8e-4
b = 5e-3
tau = .1
k = -.005
```

4) Next, we discretize time and space. Note, the time step dt must be small enough to ensure numerical stability of the simulation:

```
size = 100  # size of the 2D grid
dx = 2. / size  # space step
T = 20.0  # runtime of simulation
dt = .001  # time step
n = int(T / dt)  # number of iterations
```

5) We initialize the variables u and v as the matrices U and V where each cell contains the respective concetration values of the vertices in the 2D grid. Note these matrices are initialized with a uniform random noise between 0 and 1:

```
U = np.random.rand(size, size)
V = np.random.rand(size, size)
```

6) We now define the function that computes the discrete Laplace operator of a 2D variable on the grid. We do this by using a five-point stencil finite difference method. This operator is defined by:

$$\Delta u(x, y) \simeq \frac{u(x + h, y) + u(x - h, y) + u(x, y + h) + u(x, y - h) - 4u(x, y)}{dx^2}$$

We can compute the values of this operator on the grid using vectorized matrix operations. Because of side effects on the edges of the matrix, we need to remove the borders of the grid in the computation:

```
def laplacian(Z):
    Ztop = Z[0:-2, 1:-1]
    Zleft = Z[1:-1, 0:-2]
    Zbottom = Z[2:, 1:-1]
    Zright = Z[1:-1, 2:]
```

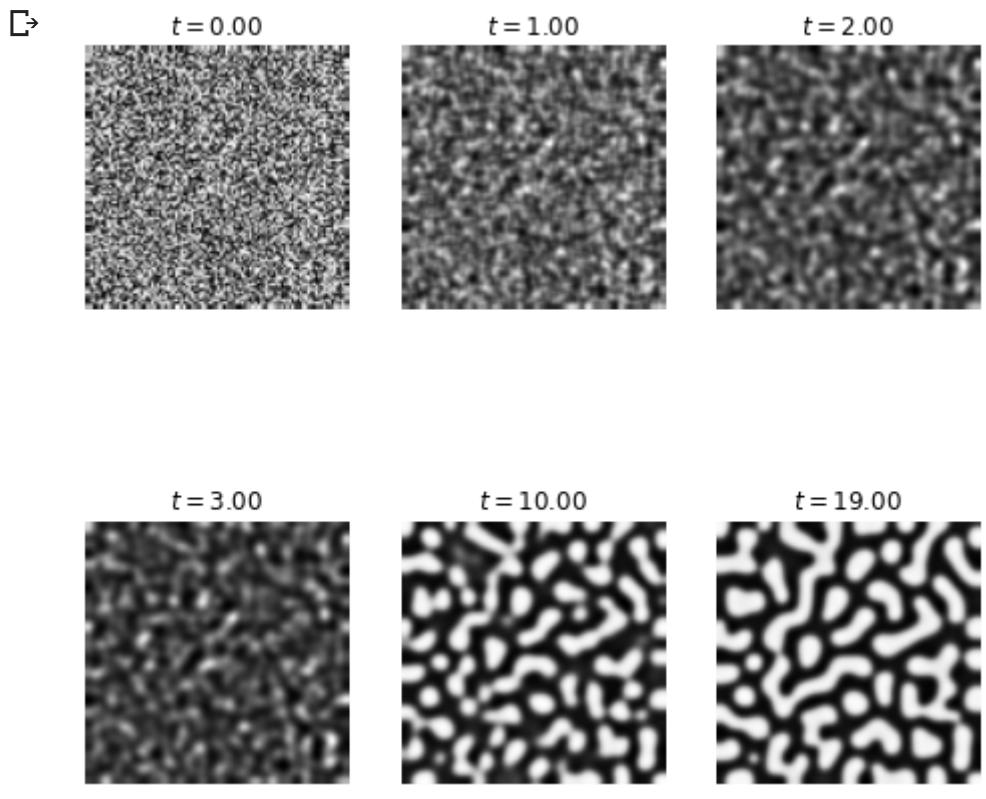
```
    Zcenter = Z[1:-1, 1:-1]
    return (Ztop + Zleft + Zbottom + Zright -
            4 * Zcenter) / dx**2
```

7) Now a function to display the matrix:

```
def show_patterns(U, ax=None):
    ax.imshow(U, cmap=plt.cm.binary,
              interpolation='bilinear',
              extent=[-1, 1, -1, 1])
    ax.set_axis_off()
```

8) Finally, we're ready to simulate the system of equations using the finite difference method. At each time step, we compute the right-hand sides of the two equations on the grid using discrete spatial derivatives (Laplacians). Then, we update the variables using a discrete time derivative. We also show the evolution of the system at 0, 1, 2, 3, 10, and 20:

```
fig, axes = plt.subplots(2, 3, figsize=(8, 8))
step_plot = n // 20
# We simulate the PDE with the finite difference
# method.
count = 0
times = [0,1,2,3,10,20]
for i in range(n):
    # We compute the Laplacian of u and v.
    deltaU = laplacian(U)
    deltaV = laplacian(V)
    # We take the values of u and v inside the grid.
    Uc = U[1:-1, 1:-1]
    Vc = V[1:-1, 1:-1]
    # We update the variables.
    U[1:-1, 1:-1], V[1:-1, 1:-1] = \
        Uc + dt * (a * deltaU + Uc - Uc**3 - Vc + k),\
        Vc + dt * (b * deltaV + Uc - Vc) / tau
    # Neumann conditions: derivatives at the edges
    # are null.
    for Z in (U, V):
        Z[0, :] = Z[1, :]
        Z[-1, :] = Z[-2, :]
        Z[:, 0] = Z[:, 1]
        Z[:, -1] = Z[:, -2]

    # We plot the state of the system at
    # times 0, 1, 2, 3, 10, 19.
    if i % step_plot == 0 and i < 20 * step_plot:
      if ((i * dt == 0.00) or (i * dt == 1.00) or (i * dt == 2.00) or (i * dt == 3.00) or (i * dt == 10.00) or (i * dt ==
        ax = axes.flat[count]
        show_patterns(U, ax=ax)
        ax.set_title(f'$t={i * dt:.2f}$')
        count = count + 1
```



Congratulations! You've made it to the end of this tutorial. Hopefully, you've gained some insight into how to model animal skin pattern formation using the two different techniques of CA and PDEs. If you're interested in learning more, check out the following references.

References

1. Sayama, Hiroki. *Introduction to the Modeling and Analysis of Complex Systems*. Open SUNY Textbooks, Milne Library, 2015.
2. *SourceForge.net*, sourceforge.net/projects/pycx/files/.
3. Takagi, Izumi. *Patterns generated by FitzHugh-Nagumo equations with nondiffusive activator and diffusive inhibitor*. August 2017. https://www.unige.ch/math/pde17/files/1115/0551/2956/Takagi-pde17.pdf
4. Zheng, Qianqian, et al. “Pattern formation in the FitzHugh–Nagumo model.” *Computers & Mathematics with Applications*, vol. 70, 2015, pp. 1082-1097., https://doi.org/10.1016/j.camwa.2015.06.031. https://www.sciencedirect.com/science/article/pii/S089812211500317X
5. “IPython Cookbook, Second Edition (2018).” *IPython Cookbook Full Atom*, ipython-books.github.io/. https://ipython-books.github.io/

Division of Labor:
We split up the work by modeling technique. Mariam covered cellular automata (CA), as the scope of CA was manageable for one person. Ford and Jake were responsible for the technique of PDEs, which was a bit more complex and more befitting to be shared by two people.