MODELING ANIMAL MOVEMENT IN ONE DIMENSION

by Rhemi Toth

MATH 19A Modeling and Differential Equations for the Life Sciences

Dr. John Wesley Cain

Harvard University

Cambridge, Massachusetts

December 7, 2022

Modeling Animal Movement in One Dimension

December 7, 2022

1 Introduction

Mechanistic home range models (MHRMs) are a useful tool for analyzing patterns of animal space use. Due to their spatially explicit design, MHRMs are often used to generate predictions of animal movement (Moorcroft & Barnett 2008). In Moorcroft & Barnett (2008), the authors illustrate how MHRMs can be used to generate predictions for a simplified case of animal movement — an individual moving on a one-dimensional landscape. Specifically, Moorcroft & Barnett utilize the advection-diffusion equation to model expected patterns of animal space use. Furthermore, Moorcroft & Barnett reveal that the steady-state pattern of space use is approximately proportional to the square of the animal's resource preference function; thus, providing an analytical approximation of an equilibrium solution to the advection-diffusion equation. In this paper, I will summarize the analysis in Moorcroft & Barnett (2008) and attempt to replicate their analytical solution for steady state space use by numerically solving the advection-diffusion equation.

According to Moorcroft & Barnett (2008), an animal moving in a one-dimensional landscape has a preference for various habitat than can be expressed by a resource selection function $w(x)$. Moreover, Moorcroft & Barnett (2008) dictate that if $w(x)$ is constant (i.e. the animal has no habitat preference), then an animal's movement is decided exclusively by $\phi(x-x')$. Here, $\phi(x-x')$ refers to the probability of an animal moving right or left of its current position x' during time interval τ (Moorcroft & Barnett, 2008). Together, $w(x)$ and $\phi(x-x')$ form a redistribution kernel that describes the probability of an animal moving from its current position x' to a new location x over a period τ in a landscape with varying preference (Moorcroft & Barnett, 2008). In Moorcroft & Barnett (2008), the redistribution kernel is written as follows:

$$P(x' \longrightarrow x) = k_\tau(x, x') = \frac{\phi(x-x')w(x)}{\int_{-\infty}^{\infty} \phi(x''-x')w(x)dx''} \tag{1}$$

After defining the redistribution kernel, Moorcroft & Barnett (2008) introduce $u(x,t)dx$ as the "probability that an individual is located between x and $x+dx$ at time t." The integral of the product of the redistribution kernel and $u(x,t)dx$ is equal to $u(x,t+\tau)$ — an expression which describes all of the possible ways that an "individual located at x' can arrive within the interval $(x, x+dx)$ at time $t+\tau$" (Moorcroft & Barnett, 2008). In Moorcroft & Barnett (2008), $u(x,t+\tau)$ is written accordingly:

$$u(x,t+\tau) = \int_{-\infty}^{\infty} k_\tau(x,x')u(x',t)dx' \tag{2}$$

After defining $u(x, t+\tau)$, Moorcroft & Barnett (2008) re-write Equation 2 in the form of an advection-diffusion equation by employing a Taylor series expansion and taking the limit as $\tau \longrightarrow 0$:

$$\frac{\partial u(x,t)}{\partial t} = -\frac{\partial}{\partial x}[c(x)u(x,t)] + \frac{\partial^2}{\partial x^2}[d(x)u(x,t)] \tag{3}$$

Moorcroft & Barnett further define the advection coefficient $c(x)$ and the diffusion coefficient $d(x)$ using:

$$c(x) = \lim_{\tau \to 0} \frac{M_2(\tau)}{\tau} \frac{w_x(x)}{w(x)} \tag{4}$$

$$d(x) = \lim_{\tau \to 0} \frac{M_2(\tau)}{2\tau} \tag{5}$$

M_2 in Equations 3 and 4 refers to the "second moment" and is characterized by Moorcroft & Barnett (2008) as:

$$M_2(\tau) = \int \rho^2 \phi_\tau(x) dx \tag{6}$$

Here ρ refers to the mean step length of the animal and $\phi(x)$ is the probability distribution of an animal's movement distances. Because ρ is a constant, it can be factored out of the integral in Equation 6. Furthermore, since $\phi(x)$ is a probability distribution function, its integral is equal to 1. Therefore, Equation 6 may be re-written as (P. Moorcroft, personal communication, November 18, 2022):

$$M_2(\tau) = \rho^2 \tag{7}$$

It is important to note that based on the definitions of $c(x)$ and $d(x)$ given by Equations 4, 5, and 7, the advection coefficient $c(x)$ varies in space, while the diffusion coefficient $d(x)$ is constant. After defining $c(x)$ and $d(x)$, Moorcroft & Barnett substitute Equations 4 and 5 into Equation 3. They then set $\frac{\partial u}{\partial t}$ in Equation 3 equal to 0 and take the limit of $u(x,t)$ as $\tau \longrightarrow \infty$. In doing so, the authors derive an analytical approximation for the expected steady-state pattern of space use:

$$u^*(x) = \frac{1}{W_0} w(x)^2 \tag{8}$$

Here $W_0 = \int_\Omega w(x)^2$ which is the integral of the square of the resource selection function taken over the entire spatial domain. Evidently, Equation 8 suggests that the expected steady-state pattern of space use by an individual is proportional to the square of the individual's resource selection function. Now that I have summarized the analysis in Moorcroft & Barnett (2008), I will attempt to replicate the authors' analytical solution for the expected pattern of space use by numerically solving Equation 3.

2 Numerical Solution for the Diffusion Equation in One Dimension

In order to develop a numerical solution for Equation 3, I will begin by seeking a numerical solution for the diffusion equation in one dimension. Specifically, I will employ the finite difference method as presented in *Computational Aerodynamics and Fluid Dynamics: an Introduction* by Jean-Jacques Chattot (2002). In accordance with Chattot (2002), I will start by discretizing the spatial and temporal domains using a set of equally spaced mesh points given by:

$$x_i = i\Delta x, i = 0, ..., N_x,$$
$$t_j = j\Delta t, j = 0, ..., N_t$$

Where x_i is the total number of points of the spatial discretization, Δx is the spatial discretization step, t_j is the total number of points of the temporal discretization, and \$ t\$ is the temporal discretization step. After discretizing the domains, the derivatives in the diffusion equation can be replaced by finite difference approximations. For the time derivative, Chattot (2002) utilizes a forward difference approximation given by:

$$\frac{\partial u}{\partial t} = \frac{u_{i,j+1} - u_{i,j}}{\Delta t} \tag{9}$$

For the space derivative, Chattot (2002) uses central difference approximation given by:

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2} \tag{10}$$

Substituting these approximations into the the diffusion equation yields the following:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = k\left(\frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}\right) \tag{11}$$

Here, k refers to the diffusion coefficient. Finally, rearranging the terms of equation 3 produces the following finite difference scheme for a numerical solution of the diffusion equation in one dimension:

$$u_{i,j+1} = u_{i,j} + r(u_{i+1,j} - 2u_{i,j} + u_{i-1,j}) \tag{12}$$

Where r, the Fourier number, is given by:

$$r = k\left(\frac{\Delta t}{\Delta x^2}\right)$$

Using Python version 3.9.13, I will implement the finite difference scheme in Equation 12 to numerically solve the diffusion equation in one dimension. The first step in my analysis is to specify the bounds of the spatial domain, the bounds of the time domain, Δt, Δx, and the diffusion coefficient. Here, I set the bounds of the spatial domain to be $[0, 20]$, the bounds of the time domain to be $[0, 500]$, Δt to be 0.01, and Δx to be 0.05.

```
# Domain Bounds
start = 0 # start bound
stop = 20 # stop bound

# Model Parameters
dt = 0.01 # delta t
T = 500 # total time
Nt = int(T / dt)  # number of time-steps
dx = 0.05  # delta x
Nx = int((abs(stop-start))/dx)  # number of x-steps
```

According to Barnett & Moorcroft (2008), the diffusion coefficient should be defined using Equation 5 above. Assuming Δt is sufficiently small, we may drop $\lim_{\tau \to 0}$ in Equation 5 and express the diffusion coefficient simply as:

$$d(x) = k = \rho^2 \tag{13}$$

Where ρ is the mean step length of the animal. Here, the mean step length of the animal is assumed to be 0.04. Ideally, the mean step length of the animal should be obtained from observations of the animal's movement in the wild (P. Moorcroft, personal communication, November 18, 2022). However in the absence of movement data, a "realistic" choice for the mean step length can be used (P. Moorcroft, personal communication, November 18, 2022). For example, if we assume here that Δt has units of hours and Δx has units of kilometers, then a mean step length of 0.04 would correspond to a movement speed of $4\frac{km}{hour}$. Depending on the animal, a movement speed of $4\frac{km}{hour}$ may or may not be realistic.

```
mean_sl = 0.04 # mean step length of the animal
k = (mean_sl**2)/2/dt # diffusion coefficient equal to mean step length squared
r = k * dt / dx / dx  # Fourier number
```

The stability of the finite difference scheme in Equation 12 is sensitive to certian choices of Δt, Δx, and mean step length. In order to avoid numerical instability, the fourier number, r, must obey the following inequality (J. Cain, personal communication, November 18, 2022):

$$r = k\left(\frac{\Delta t}{\Delta x^2}\right) < \frac{1}{2}$$

As such, I include a flag that generates a warning if current choices of Δt, Δx, and mean step length will yield numerical instability.

```
# Fourier number flag
if r > 0.5:
    print("WARNING: Fourier number > 0.5")
```

After specifying the model parameters and checking that the stability conditions are satisfied, the spatial and temporal domains can be discretized using a numpy array. The number of rows in the array are given by the number of time-steps, N_t, as specified above. The number of columns in the array are given by the number of x-steps, N_x, also specified above. In each cell of the array, the spatial position, x_i, and the probability density at position x_i and time t_j, given by $u(x_i, t_j)$, are stored.

```
# Initializing x values where u(x,t) will be calculated
Xs = np.arange(start, stop + dx, dx)

# Initializing array with Nt rows, Nx columns, and [x,u(x,t)] per cell
u = np.zeros((Nt, 2, len(Xs)))

# Populating x values in array u
for j in range(0, Nt):
    u[j][0]=Xs
```

For the inital condition, I am using a Gaussian function with parameters $\mu = 10$ and $\sigma = 0.5$. Because this script models the evolution of a probability density function (PDF), this initial condition is appropriate because it integrates to 1.

```
# Gaussian function used to specify initial conditions
def gaussian(x, mu, sigma):
    denom = sigma * ((2 * math.pi)**0.5)
    numerator = math.exp(((x - mu)**2) / ((sigma) **2) / -2)
    res = numerator / denom
    return res

# Specifying inital conditions
IC=[]
for i in range(0, len(Xs)):
    x = Xs[i]
    res = gaussian(x = x,
                   mu = 10,
                  sigma = 0.5)
    res = np.float64(res)
    IC.append(res)
u[0][1] = IC
```

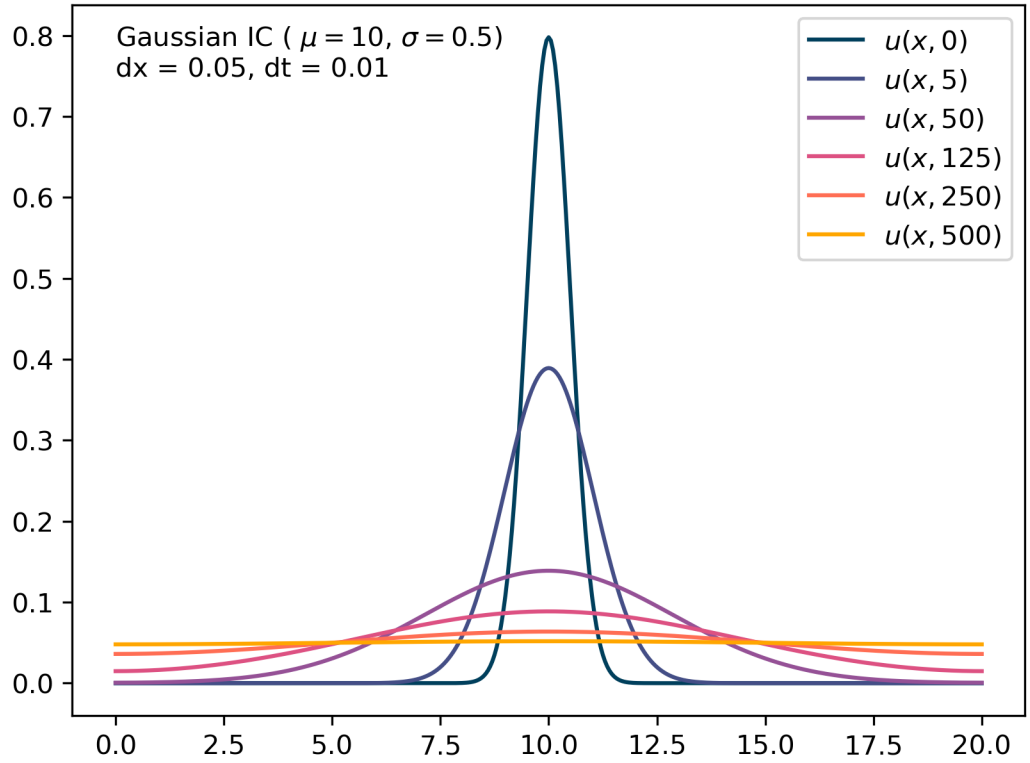
After discretizng the temporal and spatial domains as well as specifying the initial conditions, I then numerically solve for $u(x,t)$ at future time-steps using the finite difference scheme described by Equation 12. When executing the finite difference scheme in Equation 12, it is imperative that one chooses appropriate boundary conditions. Dirichlet boundary conditions should be avoided since such boundary conditions do not allow for conservation of area under the curve. Because I am modeling the evolution of a PDF, the area under the curve should remain constant as $t \longrightarrow \infty$. Zero-flux Neumann boundary conditions are a more appropriate choice, as they allow for conservation of area under the curve, but such boundary conditions are difficult to execute in code.

Here, I opt to use periodic boundary conditions as an alternative to zero-flux Neumann conditions (P. Moorcroft, personal communication, November 18, 2022). Like the Neumann boundary conditions, periodic boundary conditions allow for the area under the curve to be conserved which is necessary when modeling the evolution of a PDF. However, it should be noted that periodic boundary conditions may be unlikely in a biological context due to the fact that the bounds of the domain are treated as though they are physically connected.

```
# Explicit finite difference scheme for diffusion
for j in range(0,Nt-1):
```

```
for i in range(0,Nx+1):
    if i == 0:
        u[j + 1][1][i] = u[j][1][i] + r * (u[j][1][i + 1]
- 2 * u[j][1][i] + u[j][1][Nx])
    elif i == Nx:
        u[j + 1][1][i] = u[j][1][i] + r * (u[j][1][0]
- 2 * u[j][1][i] + u[j][1][i - 1])
    else:
        u[j + 1][1][i] = u[j][1][i] + r * (u[j][1][i + 1]
- 2 * u[j][1][i] + u[j][1][i - 1])
```

Figure 1. Central Difference Approximation for Diffusion




```
Area under u(x,0) = 1.0000000000000004
Area under u(x,100) = 0.9995508154016008
Area under u(x,200) = 0.9985210787796398
Area under u(x,300) = 0.9979724740159428
Area under u(x,400) = 0.9977189148875474
Area under u(x,500) = 0.9976031859309636
```

Figure 1 shows numerical solutions of $u(x,t)$ generated by Equation 12 at $t = 5, t = 50, t = 125, t = 250$, and $t = 500$ as well as the initial condition at $t = 0$. By $t = 300$, $u(x,t)$ appears to have flattened out which is expected when movement is due to diffusive flux. However, calculations of the integral of $u(x,t)$ at $t = 5, t = 50, t = 125, t = 250$, and $t = 500$ reveal a slight loss of

area under the curve with each time step. As such, future work on this model should involve the implementation of numerical solutions of the diffusion equation that better conserve the area under the curve.

After developing a numerical solver for the diffusion equation in one dimension, my next step in replicating the findings of Moorcroft & Barnett 2008 is to develop a process for numerically solving the advection equation in one dimension. Details on the development of a numerical solution for the advection equation in one dimension are described in the following section.

3 Numerical Solution for the Advection Equation in One Dimension

While Moorcroft & Barnett (2008) define advection speed based on the spatial distribution of resources, I will start with the simpler case of modeling advection with constant speed.

3.1 Advection with Constant Speed: First-Order Upwinding Approximation

The process used to construct a numerical solution for the advection equation with constant speed is highly similar to the process used to construct a numerical solution for the diffusion equation in the previous section. Following Chattot (2002), I start by discretizing my domain using a set of equally spaced mesh points in the same way that I did for the diffusion equation. Similarly, I then replace the derivatives in the advection equation using finite difference approximations. I will continue to use a forward difference approximation for the time derivate (Equation 9) and instead of using a central difference approximation like I did for the spatial derivative in the diffusion equation, I will approximate the spatial derivative in the advection equation using an upwind approach (Chattot, 2002). Here, I am using a first-order upwind approximation for the spatial derivative given by:

$$\frac{\partial u}{\partial x} = \frac{u_{i,j} - u_{i-1,j}}{\Delta x} \text{ if c} > 0 \tag{14}$$

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i,j}}{\Delta x} \text{ if c} < 0 \tag{15}$$

Which substituting into the PDE along with the forward difference approximation of the time derivative yieds:

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = -c\left(\frac{u_{i,j} - u_{i-1,j}}{\Delta x}\right) \text{ if c} > 0 \tag{16}$$

$$\frac{u_{i,j+1} - u_{i,j}}{\Delta t} = -c\left(\frac{u_{i+1,j} - u_{i,j}}{\Delta x}\right) \text{ if c} < 0 \tag{17}$$

Finally, re-arranging terms produces the following finite difference scheme for the numerical solution of the advection equation in one dimension:

$$u_{i,j+1} = u_{i,j} - \rho(u_{i,j} - u_{i-1,j}) \text{ if c} > 0 \tag{18}$$

$$u_{i,j+1} = u_{i,j} - \rho(u_{i+1,j} - u_{i,j}) \text{ if c} < 0 \tag{19}$$

Where ρ, the Courant number, is given by:

$$\rho = c\left(\frac{\Delta t}{\Delta x}\right)$$

After defining a finite difference scheme for the numerical solution of the advection equation in one dimension, I will now specify the bounds the spatial domain, the bounds of the time domain, Δt, Δx, and the velocity of the animal. Here, I set the bounds of the spatial domain to be $[0, 50]$, the bounds of the time domain to be $[0, 300]$, Δt to be 0.01, Δx to be 0.05, and velocity, c, to be 0.1.

```
# Bounds
start = 0 # start bound
stop = 50 # stop bound

# Model parameters
dt = 0.01 # delta t
dx = 0.05  # delta x
T = 300 # Total time
Nt = int(T / dt)  # Number of time steps
Nx = int((abs(stop-start))/dx)  # Number of x steps
c = 0.1 # advection speed
p = c * dt / dx /2 # courant number
```

Like the finite difference scheme for the diffusion equation described above, the stability of the finite difference scheme for the advection equation is sensitive to choices of various inputs. Specifically, the finite difference scheme for the advection equation is sensitive to certain choices for Δt, Δx, and c. In order to avoid numerical instability, the Courant number, ρ, must obey the Courant–Friedrichs–Lewy (CFL) condition (Chattot, 2002):

$$\rho = c\left(\frac{\Delta t}{\Delta x}\right) < 1$$

As such, I include a flag that generates a warning if the current values of Δt, Δx, and c will cause numerical instability.

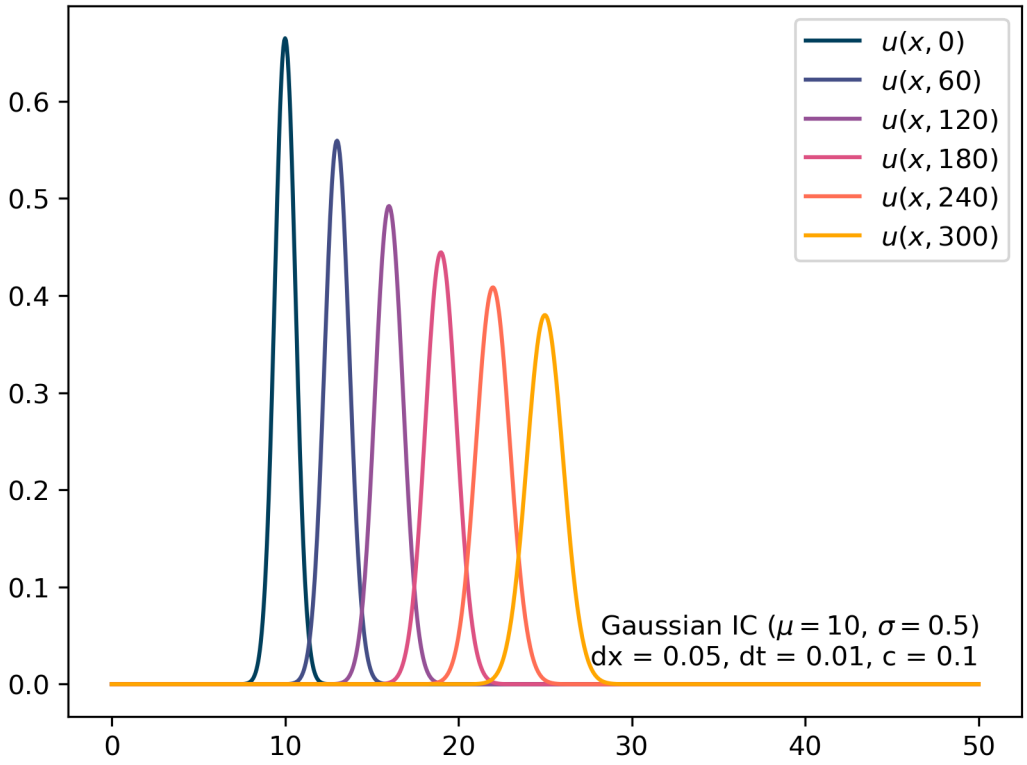
```
# CFL flag
if p > 1:
    print("WARNING: Courant number > 1")
```

In an identical fashion to the diffusion example described in the previous section, the spatial and temporal domains are discretized using a numpy array with N_t rows and N_x columns. In each cell of the array, the spatial position, x_i, and the probability density at position x_i and time t_j, $u(x_i, t_j)$, are stored. Additionally, I am continuing to use a Gaussian initial condition, this time with $\mu = 10$ and $\sigma = 0.6$

After discretizing the domains and specifying the initial conditions, I then numerically solve for $u(x,t)$ at future time-setps using the finite difference scheme described by Equations 18 and 19. Again, I am using periodic boundary conditions.

```
# First-order upwind scheme
for j in range(0,Nt-1):
    for i in range(0,Nx+1):
        if i == 0:
            u[j+1][1][i] = u[j][1][i] - p * (u[j][1][i] - u[j][1][-1])
        else:
            u[j + 1][1][i] = u[j][1][i] - p * (u[j][1][i] - u[j][1][i - 1])
```

Figure 2. First-Order Upwinding Approximation for Advection




```
Area under u(x,0) = 0.9999999999999998
Area under u(x,60) = 1.0
Area under u(x,120) = 1.0
Area under u(x,180) = 1.0
Area under u(x,240) = 1.0000000000000002
Area under u(x,300) = 1.0000000000000002
```

Figure 2 illustrates numerical solutions of $u(x,t)$ generated by Equations 18 and 19 at $t = 60, t = 120, t = 180, t = 240$, and $t = 300$ as well as the initial condition at $t = 0$. Although the area under the curve is conserved as the PDF evolves over time, the PDF exhibits some concerning behaviors that do not follow what is predicted by the analytical solution of the simple advection

equation. The simple advection equation is given by:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0$$

And its solution is:

$$\phi(x - ct)$$

According to the simple advection equation, if c is constant, the solution at future time steps is the profile of the initial condition shifted by ct units in space. However, as can be seen in Figure 2, the numerical solution presented in Equations 18 and 19 does not exhibit the same behavior as the analytical solution of the simple advection equation. Specifically, the initial profile does not maintain the same shape as t increases nor does the profile at future time-steps move left to right by ct units. In an attempt to achieve greater compliance with the analytical solution of the simple advection equation, I will now try replacing the first-order upwind approximation for the spatial derivative with a second-order upwind approximation.

3.2 Advection with Constant Speed: Second-Order Upwind Approximation

According to Shyy (1985), finite difference schemes employing a second-order upwind approximation preform better than those relying on a first-order upwind approximation. Keeping with Shyy (1985), a second-order upwind approximation of the spatial derivative may be defined by:

$$\frac{\partial u}{\partial x} = \frac{3u_{i,j} - 4u_{i-1,j} + u_{i-2,j}}{2\Delta x} \text{ if c} > 0 \tag{20}$$

$$\frac{\partial u}{\partial x} = \frac{-u_{i+2,j} + 4u_{i+1,j} - 3u_{i,j}}{2\Delta x} \text{ if c} < 0 \tag{21}$$

Substituting Equations 20 and 21 along with Equation 9 into the simple advection equation yields:

$$u_{i,j+1} = u_{i,j} - \rho(3u_{i,j} - 4u_{i-1,j} + u_{i-2,j}) \text{ if c} > 0 \tag{22}$$

$$u_{i,j+1} = u_{i,j} - \rho(-u_{i+2,j} + 4u_{i+1,j} - 3u_{i,j}) \text{ if c} < 0 \tag{23}$$

Where the Courant number ρ is now given by:

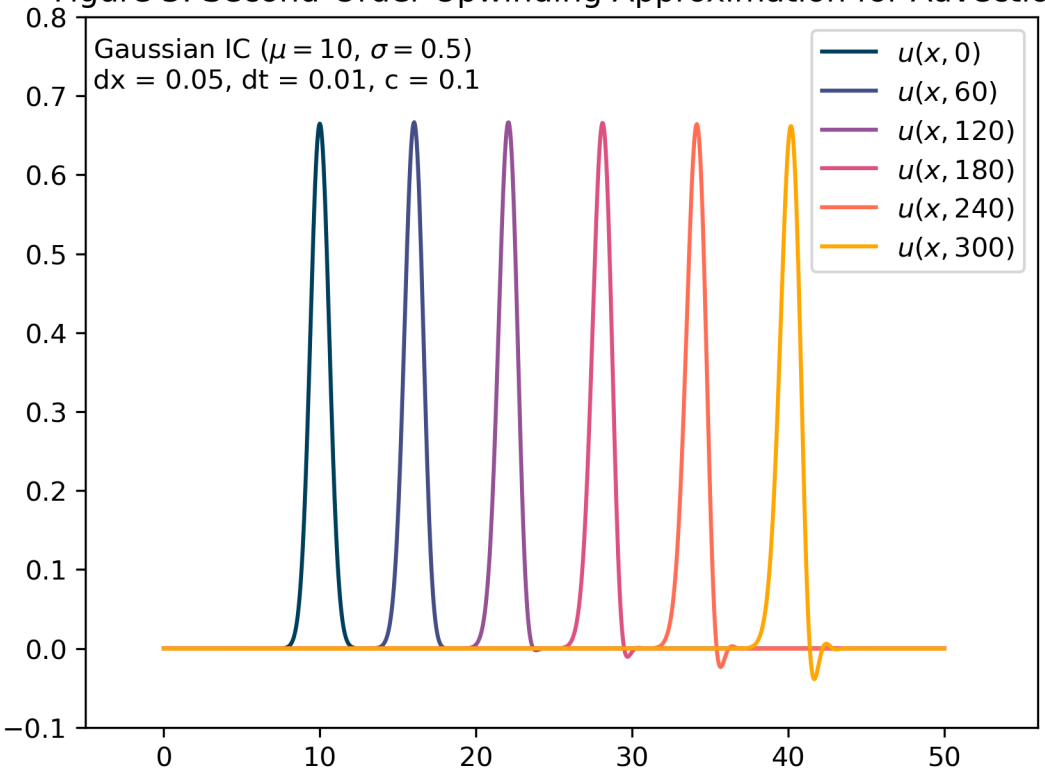
$$\rho = c\left(\frac{\Delta t}{2\Delta x}\right)$$

Equations 22 and 23 can be implemented in Python with period boundary conditions via the following code:

```
# Second-order upwinding approach
for j in range(0,Nt-1):
    for i in range(0,Nx+1):
        if c > 0:
            if i == 0:
```

```
            u[j + 1][1][i] = -p * (3*u[j][1][i] - 4 * u[j][1][Nx]
                                  + u[j][1][Nx-1]) + u[j][1][i]
        elif i == 1:
            u[j + 1][1][i] = -p * (3 * u[j][1][i] - 4 * u[j][1][i - 1]
                                  + u[j][1][Nx]) + u[j][1][i]
        else:
            u[j + 1][1][i] = -p * (3*u[j][1][i] - 4 * u[j][1][i-1]
                                  + u[j][1][i-2]) + u[j][1][i]
    else:
        if i == Nx:
            u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][1] + 4 * u[j][1][0]
                                               - 3 * u[j][1][Nx])
        elif i == (Nx-1):
            u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][0] + 4 * u[j][1][Nx]
                                               - 3 * u[j][1][Nx-1])
        else:
            u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][i+2] + 4 * u[j][1][i + 1]
                                               - 3 * u[j][1][i])
```

Figure 3. Second-Order Upwinding Approximation for Advection




```
Area under u(x,0) = 0.9999999999999998
Area under u(x,60) = 0.9999999999999993
Area under u(x,120) = 1.0000000000000002
```

```
Area under u(x,180) = 1.0000000000000013
Area under u(x,240) = 1.0000000000000002
Area under u(x,300) = 0.9999999999999998
```

A comparison of Figure 2 with Figure 3 supports the finding in Shyy (1985) that second-order upwinding more accurately approximates the spatial derivative than first order upwinding. In Figure 2, a first-order upwind approximation of the spatial derivaitve did not result in the initial density profile shifting by ct units during each progressive time-step. However, the use of a second-order upwind approximation in Figure 3 caused the initial density profile to shift by ct units as t increased. Furthermore, the shape of the initial density profile is conserved at future time-steps in Figure 3, while the initial density profile changed shape in Figure 2. Be that as it may, the second-order upwind approximation generated some numerical instability which is visible in Figure 3 starting at $t = 120$. Ideally, a more numerically stable method of solving the simple advection equation should be used. However given the time constraints of this project, a second-order upwind approximation was the best scheme that I was able to implement for numerically solving the advection equation.

3.3 Advection with Variable Speed

Now that I have identified a finite difference scheme for modeling advection with constant speed, I will now move on to tackling the more biologically relevant problem of modeling advection with variable speed. Adhering to Moorcroft & Barnett (2008), the advection coefficient should now take the form of Equation 4 above. Assuming Δt is sufficiently small, we may drop $\lim_{\tau \to 0}$ in Equation 4 and express $c(x)$ simply as:

$$c(x) = \frac{M_2(\tau)}{\tau} \frac{w_x(x)}{w(x)} \tag{24}$$

Because the the value of the advection coefficient varies spatially with the habitat preference function $w(x)$, $c(x)$ must be calculated at every value x_i in the spatial domain. Using a forward difference approximation for the time derivative and a second-order upwind approximation for the spatial derivative, a finite difference scheme for advection with variable speed may be written as:

$$u_{i,j+1} = u_{i,j} - \frac{M_2(\tau)}{\tau} \frac{w_x(x_i)}{w(x_i)} \left(\frac{\Delta t}{2\Delta x} \right) (3u_{i,j} - 4u_{i-1,j} + u_{i-2,j}) \text{ if c} > 0 \tag{25}$$

$$u_{i,j+1} = u_{i,j} - \frac{M_2(\tau)}{\tau} \frac{w_x(x_i)}{w(x_i)} \left(\frac{\Delta t}{2\Delta x} \right) (-u_{i+2,j} + 4u_{i+1,j} - 3u_{i,j}) \text{ if c} < 0 \tag{26}$$

Where $-\frac{M_2(\tau)}{\tau} \frac{w_x(x_i)}{w(x_i)} \left(\frac{\Delta t}{2\Delta x} \right)$ is equal to the Courant number ρ at x_i.

As a first example, I will consider a habitat preference function of the form:

$$w(x) = 0.1 + \sin^2 \left(\frac{2\pi x}{50} \right) \tag{27}$$

With spatial derivative:

$$w_x(x) = \frac{2\pi}{25} \cos\left(\frac{\pi x}{25}\right) \sin\left(\frac{\pi x}{25}\right) \tag{28}$$

In order to incorporate Equations 27 and 28 into the finite difference scheme described by Equations 25 and 26, I must first calculate the values of $w(x)$ and $w_x(x)$ at every value x_i in my domain. This may be accomplished using the following Python code:

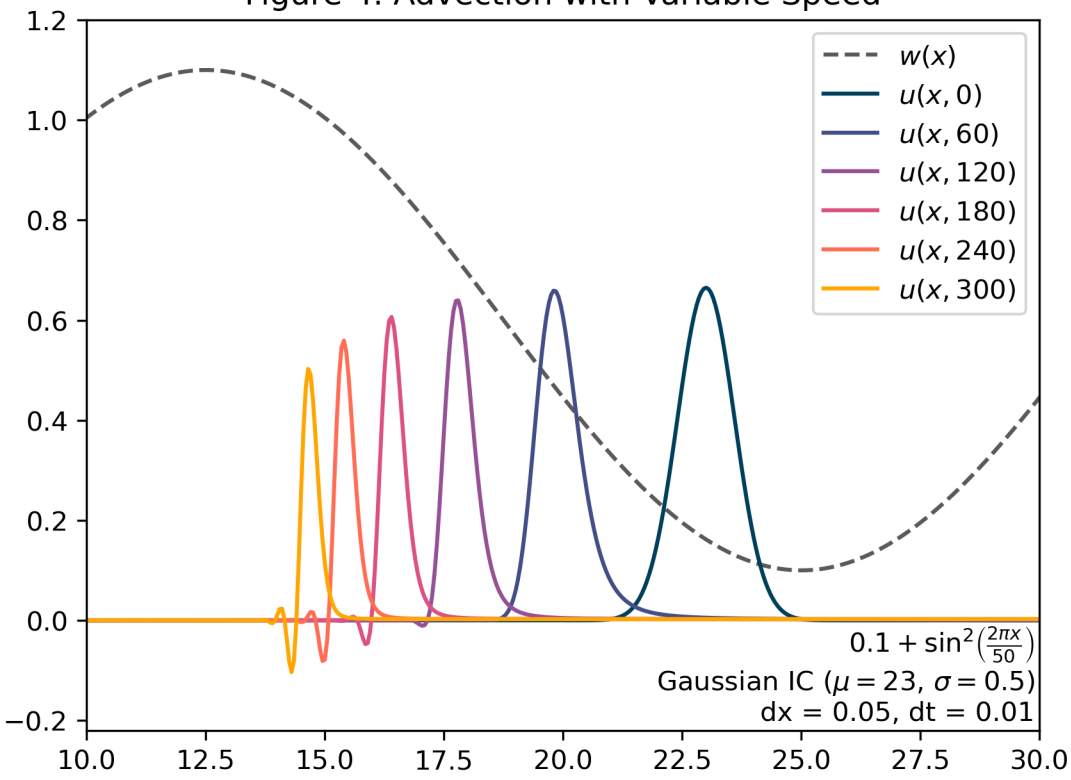
```
# Habitat Preference Function
def preference(x):
    pi = math.pi
    res = 0.1 + math.sin(2 * pi * x / 50) ** 2
    return res

# Derivative of Habitat Preference Function
def preference_slope(x):
    pi = math.pi
    res = (2 * pi / 25) * math.cos(pi * x / 25) * math.sin(pi * x / 25)
    return res

# Calculating habitat preference at x_i
w = np.zeros((1,2,len(Xs)))
w[0][0] = Xs
for i in range(0,len(Xs)):
    w[0][1][i] = preference(w[0][0][i])

# Calculating derivative of habitat preference at x_i
wx = np.zeros((1,2,len(Xs)))
wx[0][0] = Xs
for i in range(0,len(Xs)):
    wx[0][1][i] = preference_slope(w[0][0][i])
```

After calculating $w(x)$ and $w_x(x)$ for every value x_i in my domain, I can now model advection with variable speed using the finite difference scheme described in Equations 25 and 26:

```
for j in range(0,Nt-1):
    for i in range(0,Nx+1):
        c = (mean_sl**2) * wx[0][1][i] / w[0][1][i] / dt  # Advection coefficient
        p = c * dt / dx /2 # Courant number
        if c > 0:
            if i == 0:
                u[j + 1][1][i] = u[j][1][i] -p * (3*u[j][1][i] - 4 * u[j][1][Nx]
                                                  + u[j][1][Nx-1])
            elif i == 1:
                u[j + 1][1][i] = u[j][1][i] -p * (3 * u[j][1][i] - 4 * u[j][1][i - 1]
                                                  + u[j][1][Nx])
            else:
                u[j + 1][1][i] = u[j][1][i] -p * (3*u[j][1][i] - 4 * u[j][1][i-1]
                                                  + u[j][1][i-2])
        else:
```

```
if i == Nx:
    u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][1] + 4 * u[j][1][0]
                                       - 3 * u[j][1][Nx])
elif i == (Nx-1):
    u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][0] + 4 * u[j][1][Nx]
                                       - 3 * u[j][1][Nx-1])
else:
    u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][i+2] + 4 * u[j][1][i + 1]
                                       - 3 * u[j][1][i])
```

Figure 4. Advection with Variable Speed




```
Area under u(x,0) = 1.0000000000000002
Area under u(x,60) = 0.7792795317052899
Area under u(x,120) = 0.5287335829487156
Area under u(x,180) = 0.38943306821815143
Area under u(x,240) = 0.3007411616552028
Area under u(x,300) = 0.23959897942307748
```

Figure 4 depicts the expected space use of an animal moving with variable speed according to the resource preference function given by Equation 27. As was foreseeable, the finite difference scheme defined in Equations 25 and 26 predicted that the animal would move away from its initial condition in an area of low resource preference to an area of high resource preference. However, the finite difference scheme in Equations 25 and 26 resulted in a loss of area under $u(x,t)$ as $t \longrightarrow \infty$.

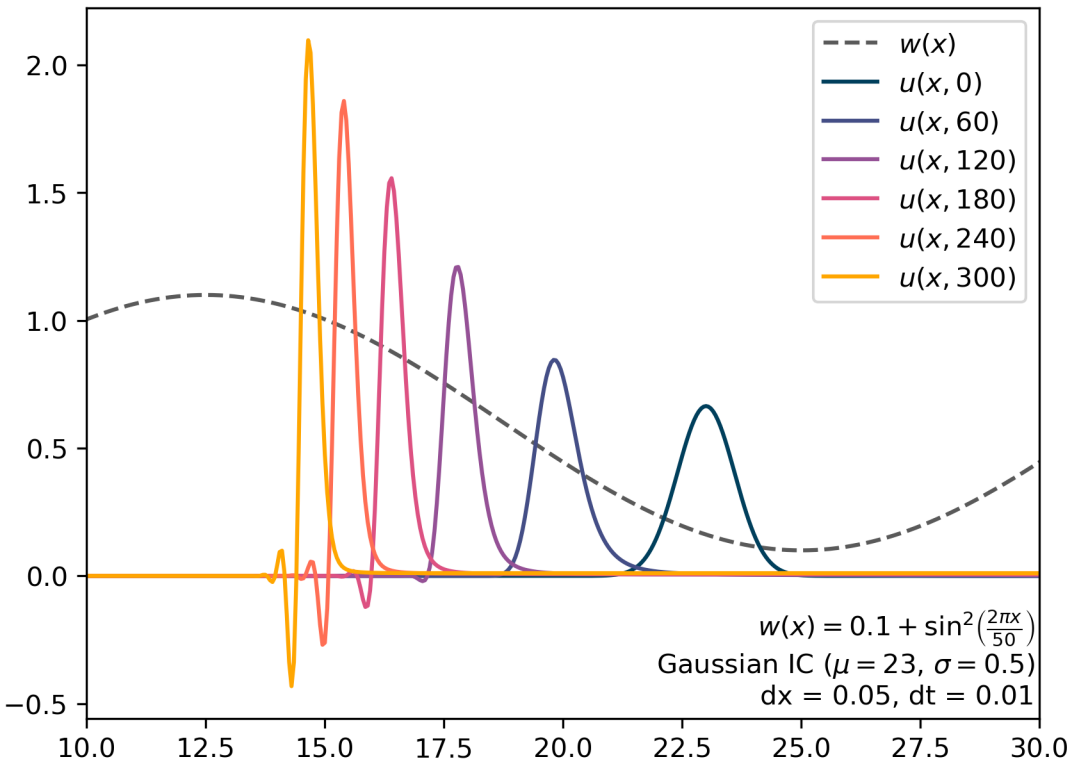
Additionally, this scheme generated numerical instability as $u(x,t)$ was calculated at increasing values of t. In the next section, I will attempt to modify the finite difference scheme in Equations 25 and 26 to better conserve the area under $u(x,t)$ and reduce numerical instability.

3.4 Advection with Variable Speed (Modified Scheme)

After calculating $u(x,t)$ at each time-step using Equations 25 and 26, conservation of the area under the curve can be introduced by normalizing each value $u(x_i,t_j)$ by the area under $u(x,t_j)$ (P. Moorcroft, personal communication, November 18, 2022). I will refer to this procedure as a Modified Second-Order Upwind Scheme (MSUS). MSUS can be implemented in Python using the following code:

```
# Second-order upwind approximation with normalization by area under the curve
for j in range(0,Nt-1):
    for i in range(0,Nx+1):
        c = (mean_sl**2) * wx[0][1][i] / w[0][1][i] / dt  # Advection coefficient
        p = c * dt / dx /2 # Courant number
        if c > 0:
            if i == 0:
                u[j + 1][1][i] = u[j][1][i] -p * (3*u[j][1][i] - 4 * u[j][1][Nx]
                                                  + u[j][1][Nx-1])
            elif i == 1:
                u[j + 1][1][i] = u[j][1][i] -p * (3 * u[j][1][i] - 4 * u[j][1][i - 1]
                                                  + u[j][1][Nx])
            else:
                u[j + 1][1][i] = u[j][1][i] -p * (3*u[j][1][i] - 4 * u[j][1][i-1]

        else:
            if i == Nx:
                u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][1] + 4 * u[j][1][0]
                                                   - 3 * u[j][1][Nx])
            elif i == (Nx-1):
                u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][0] + 4 * u[j][1][Nx]
                                                   - 3 * u[j][1][Nx-1])
            else:
                u[j + 1][1][i] = u[j][1][i] - p * (-u[j][1][i+2] + 4 * u[j][1][i + 1]
                                                   - 3 * u[j][1][i])
    area = integrate(u = u[j + 1][1],
                     dx = dx,
                     x_vals = Xs)
    u[j+1][1] = u[j+1][1]/area
```

Figure 5. Advection with Variable Speed (Modified Scheme)




```
Area under u(x,0) = 1.0000000000000002
Area under u(x,60) = 0.9999999999999989
Area under u(x,120) = 0.9999999999999992
Area under u(x,180) = 1.0000000000000002
Area under u(x,240) = 1.0000000000000009
Area under u(x,300) = 1.0000000000000002
```

Figure 5 depicts the results of using MSUS to model advection with variable speed. Like Equations 25 and 26, MSUS predicts that the animal will move away from its initial condition in an area of low resource preference to an area of high resource preference. Unlike Equations 25 and 26 however, MSUS conserves the area under the curve as $u(x,t)$ is calculated at future time-steps. Despite conserving the area under the curve, MSUS does not ameliorate the numerical instability present in solutions generated by Equations 25 and 26. In fact, a comparison of Figure 4 and Figure 5 suggests that the numerical instability produced by MSUS is of a greater magnitude than the instability produced by Equations 25 and 26. While numerical solutions produced by MSUS are far from perfect, I will continue to model advection using MSUS in the next section due to the time constraints of this project.

4 Numerical Solution for the Advection-Diffusion Equation in One Dimension

In the previous sections, I generated numerical solutions for the diffusion and advection equations separately. Now, I will attempt to model advection and diffusion simultaneously, by combining the numerical solutions that I have already defined. Keeping with previous sections, I will solve for $u(x,t)$ at future time-steps using a forward difference approximation for the time derivative, a central difference approximation for the spatial derivative of the diffusion term, and a second-order upwind approximation for the spatial derivative of the advection term. Substituting these approximations into the advection-diffusion equation yields the following finite difference scheme:

$$u_{i,j+1} = u_{i,j} + r(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) - \rho(3u_{i,j} - 4u_{i-1,j} + u_{i-2,j}) \text{ if c} > 0 \tag{29}$$

$$u_{i,j+1} = u_{i,j} + r(u_{i-1,j} - 2u_{i,j} + u_{i+1,j}) - \rho(-u_{i+2,j} + 4u_{i+1,j} - 3u_{i,j}) \text{ if c} < 0 \tag{30}$$

After calculating $u(x,t)$ at each time-step, I will normalize each value $u(x_i,t_j)$ by the area under $u(x,t_j)$ like I did for MSUS. I will refer to the process of solving for $u(x,t)$ at future time steps using Equations 29 and 30 and normalizing by the area under the curve as the Finite Difference Scheme for Advection and Diffusion (FDSAD). I will now show how FDSAD can be implemented in Python. As I did in previous sections, I will start by defining the bounds of the spatial and temporal domains, Δt, Δx, and the the animal's mean step length.

```
# Bounds
start = 0 # start bound
stop = 50 # stop bound

# Model parameters
dt = 0.01 # delta t
dx = 0.05  # delta x
T = 1000 # Total time
Nt = int(T / dt)  # Number of time steps
Nx = int((abs(stop-start))/dx)  # Number of x steps
mean_sl = 0.04 #mean step length
k = (mean_sl**2)/2/dt # diffusion coefficient equal to mean step length squared
r = k * dt / dx / dx  # Fourier number
```

Similarly, I will continue to use the habitat preference function introduced in Equation 27. In addition, I will continue using a Gaussian initial condition; this time with $\mu = 28$ and $\sigma = 0.5$ With the model parameters, habitat preference function, and initial conditions now defined, I can solve for $u(x,t)$ at future time-steps using FDSAD:

```
# Finite difference scheme for advection and diffusion
for j in range(0,Nt-1):
    for i in range(0,Nx+1):
        c = (mean_sl**2) * wx[0][1][i] / w[0][1][i] / dt  # Advection coefficient
        p = c * dt / dx /2 # Courant number
        if p > 1: # CFL flag
```

```
            print("WARNING: Courant number > 1")
        if c > 0:
            if i == 0:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][i + 1] - 2 * u[j][1][i] + u[j][1][Nx])
                - p * (3*u[j][1][i] - 4 * u[j][1][Nx] + u[j][1][Nx-1])
            elif i == 1:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][i + 1] - 2 * u[j][1][i] + u[j][1][i - 1])
                - p * (3 * u[j][1][i] - 4 * u[j][1][i - 1] + u[j][1][Nx])
            elif i == Nx:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][0] - 2 * u[j][1][i] + u[j][1][i - 1])
                - p * (3 * u[j][1][i] - 4 * u[j][1][i - 1] + u[j][1][i - 2])
            else:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][i + 1] - 2 * u[j][1][i] + u[j][1][i - 1])
                - p * (3*u[j][1][i] - 4 * u[j][1][i-1] + u[j][1][i-2])
        else:
            if i == Nx:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][0] - 2 * u[j][1][i] + u[j][1][i - 1])
                - p * (-u[j][1][1] + 4 * u[j][1][0] - 3 * u[j][1][Nx])
            elif i == (Nx-1):
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][i + 1] - 2 * u[j][1][i] + u[j][1][i - 1])
                - p * (-u[j][1][0] + 4 * u[j][1][Nx] - 3 * u[j][1][Nx-1])
            elif i == 0:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][i + 1] - 2 * u[j][1][i] + u[j][1][Nx])
                - p * (-u[j][1][i + 2] + 4 * u[j][1][i + 1] - 3 * u[j][1][i])
            else:
                u[j + 1][1][i] = u[j][1][i]
                + r * (u[j][1][i + 1] - 2 * u[j][1][i] + u[j][1][i - 1])
                - p * (-u[j][1][i+2] + 4 * u[j][1][i + 1] - 3 * u[j][1][i])
    area = integrate(u = u[j + 1][1],
                     dx = dx,
                     x_vals = Xs)
    u[j+1][1] = u[j+1][1]/area
```

Figure 6. Advection and Diffusion

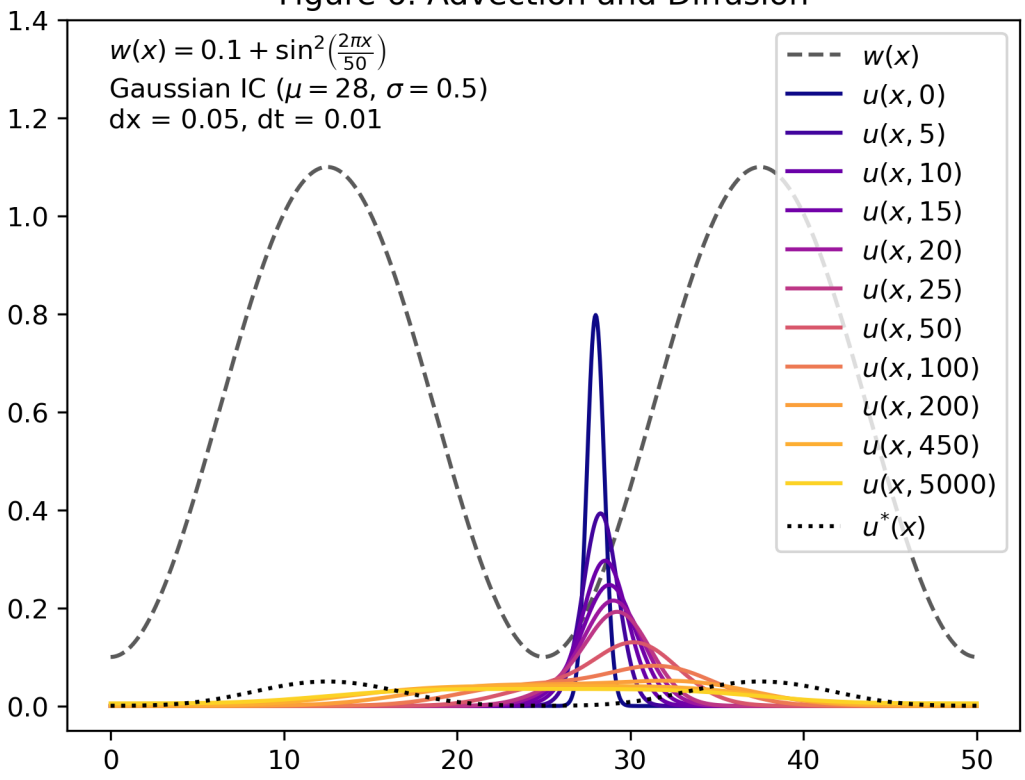


Figure 6 depicts numerical solutions generated by FDSAD. An animation of Figure 6 is also included as an email attachment in the submission of the final report (see Figure6.mp4). Note, that Figure 6 also shows an analytical approximation of the expected steady-state pattern of space use $u^*(x)$ that was generated using the methods in Moorcroft & Barnett (2008) (see Equation 8). Interestingly, the numerical instability produced by the second-order upwind approximation that was visible in Figures 5 and 6 appears to have disappeared when advection and diffusion are modeled together. Furthermore, as expected when movement is due in part to diffusion, the PDF flattens out over time. Additionally, during the earlier time-steps, the initial PDF appears to shift from from an area of low resource preference to an area of high resource preference which is expected when movement is due in part to advection.

During the later time-steps however, the evolution of the PDF that is predicted by my numerical solution deviates from the expected steady-state pattern of space use given by Equation 8. Instead of concentrating in the two areas of high resource preference, as predicted by $u^*(x)$, the PDF appears to settle in an area of low resource preference in the middle of the domain. In Figure 6, I ran the simulation out to $t = 5000$, but to generate Figure6.mp4 (see email attachment) I ran the simulation out to $t = 10000$. Running the simulation out to $t = 10000$ in Figure6.mp4 did not change the final result very much; thus, suggesting that the expected steady state pattern of space use predicted by the numerical solution is approximately represented by $u(x, 5000)$ in Figure 6. Regardless, it is clear that the numerical solution is not behaving as expected by Equation 8.

After modeling advection and diffusion using a resource preference function in the form of

Equation 27, where resources are concentrated in two areas, I will now investigate the expected pattern of animal space use in a landscape where the resources are concentrated in one area in the the center of the domain. For this purpose, I will adopt a new resource preference function:

$$w(x) = e^{-0.01|x-50|^2} \tag{31}$$

With spatial derivative:

$$w_x(x) = -0.02e^{-0.01|x-50|^2}\,(x-50) \tag{32}$$

Figure 7. Advection and Diffusion

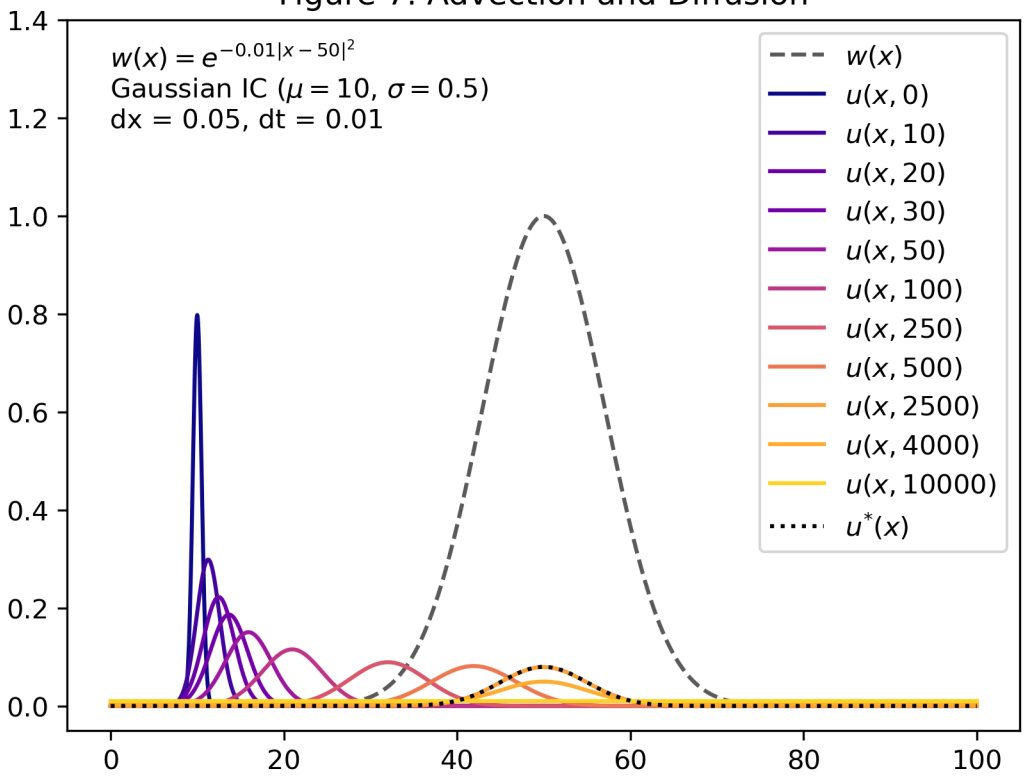


Figure 7 illustrates numerical solutions generated using FDSAD in conjunction with the resource preference function given by Equation 31. An animation of Figure 7 is also included as an email attachment in the submission of the final report (see Figure7.mp4). As expected when movement is due in part to diffusion, the PDF flattens out over time. Additionally, the initial PDF appears to shift from an area of low resource preference to an area of high resource preference which is expected when movement is due in part to advection. According to $u^*(x)$, the animal in question should form a home range centered around the area of highest resource preference at $x = 50$. However, my numerical solution predicts that the expected steady-state pattern of space use by the animal will be uniform across the domain. Evidently, the results depicted in Figures 6 and 7 suggest that FDSAD is unable to accurately predict the steady-state pattern of space use for an animal moving in a one-dimensional landscape with varying resource quality.

5 Discussion

In this paper, I attempted to develop a numerical solution for the advection-diffusion equation in order to replicate the finding in Moorcroft & Barnett (2008) that the expected steady-state pattern of space use by an animal is proportional to the square of its resource preference function (see Equation 8). The numerical solution that I developed, FDSAD, exhibited some, but not all of the behaviors that are expected when movement is due to advection and diffusion. First, FDSAD predicted that as $t \longrightarrow \infty$, the profile of $u(x,t)$ will flatten out due to diffusion. Second, FDSAD predicted some movement of the profile of $u(x,t)$ towards areas of high resource preference due to advection. Despite exhibiting these behaviors, I was unable to reproduce Moorcroft & Barnett's analytical approximation for the expected steady-state pattern of space use utilizing the finite difference scheme in FDSAD.

The disparity between my numerical solution and Moorcroft & Barnett's analytical approximation may be due to several possible factors. To start, my decision to normalize each value $u(x_i, t_j)$ by the area under $u(x, t_j)$ was likely not an appropriate remedy for the undesirable behavior that I witnessed in earlier versions of my finite difference scheme. In reality, normalizing in this way could be viewed as "*gaming the system*" in order to produce a desirable outcome without actually addressing the root cause of the numerical instability and loss of area under the curve. In the future, I will need to develop a numerical solution that: (1) conserves the area under $u(x,t)$ at future time-steps without normalizing each value $u(x_i, t_j)$ by the area under $u(x, t_j)$ and (2) addresses the numerical instability that arises when modeling advection.

Another potential flaw with my numerical solution is that upwind approximations may not be appropriate when modeling advection with variable speed. When approximating using an upwind approach, FDSAD associates the velocity at x_i with positions upwind of x_i (namely x_{i-1} and x_{i-2} if $c > 0$ and x_{i+1} and x_{i+2} if $c < 0$). At first glance, one might think to remedy this by simply calculating the velocity at each position upwind of x_i as opposed to associating the velocity at x_i with positions upwind of x_i. However, complications arise when the sign of the velocity at x_i is different than the sign of the velocity at positions upwind of x_i. By design, the upwind approximation requires the user to define the sign of the velocity before choosing a scheme to approximate with. If the velocity at x_i is different from the velocity at positions upwind of x_i, then the definition the sign of the velocity in an upwind approximation becomes unclear. While it is inappropriate to associate the velocity at x_i with positions upwind of x_i, it is unclear how to incorporate the velocities at positions upwind of x_i in an upwinding scheme. Looking ahead, I will need to model advection using a finite difference method that more accurately characterizes the velocity at a point (i.e. one that does not associate the velocity at x_i with x_{i-1}, x_{i-2}, etc.).

Once I resolve the issues described above, it is imperative that I apply what I have learned about modeling the movement of animals in one dimension to modeling movement in two dimensions. Doing so will allow me to answer more interesting questions regarding animal movement. Namely, how anthropogenic disturbance influences the movement of animals.

6 References

1. Chattot, J.-J. (2002). Computational Aerodynamics and Fluid Dynamics: An Introduction. Springer Berlin Heidelberg.
2. Moorcroft, P., & Lewis, M. (2006). Mechanistic home range analysis. Princeton University Press.

3. Moorcroft, P. R., & Barnett, A. (2008). MECHANISTIC HOME RANGE MODELS AND RESOURCE SELECTION ANALYSIS: A RECONCILIATION AND UNIFICATION. Ecology, 89(4), 1112–1119. https://doi.org/10.1890/06-1985.1
4. Shyy, W. (1985). A study of finite difference approximations to steady-state, convection-dominated flow problems. Journal of Computational Physics, 57(3), 415–438. https://doi.org/10.1016/0021-9991(85)90188-3