AMATH 581 Homework #3

Tyler Shakibai

1. We examine the advection equation and how its solutions are affected by a velocity field. In our problem, advection is defined as the transfer of a specific quantity, $u(x,t)$, by the flow in some velocity field $c(x,t)$. We want to analyze how two different velocity fields $c_1(x,t) = -0.5$ and $c_2(x,t) = -(1 + 2\sin(5t) - H(x-4))$ advect our solution u differently on the boundaries $-10 \leq x \leq 10$ and $0 \leq t \leq 10$ and understand them in a physical context. A physical interpretation of the advection equation could be the horizontal flow of water in a sea or ocean. The solution u represents the height of a wave that is being advected, or that is flowing horizontally, at a speed defined by the field c. First, we examine the simpler case with $c_1(x,t) = -0.5$, where the velocity field is constant.

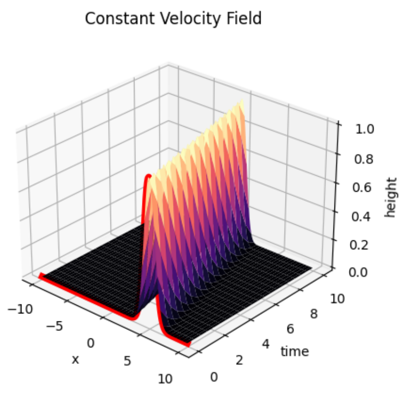

Figure 1: 3D plot of the solution $u(x,t)$ to our advection equation in velocity field $c_1(x,t) = -0.5$.

This matches our expectation as this constant velocity field itself lacks both a spatial or temporal component, meaning the effect of the field remains constant regardless of its location in space or place in time. To retain our physical intuition, our solution can be represented as a uniform wave advecting at the same rate throughout. We can now examine a more interesting case.

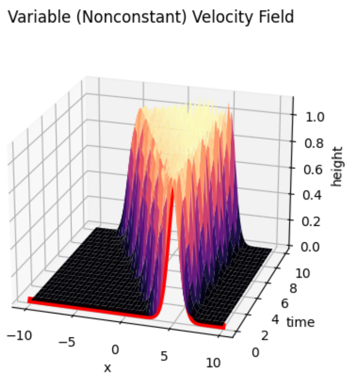

Figure 2: 3D plot of $u(x,t)$ in velocity field $c_2(x,t) = -(1 + 2\sin(5t) - H(x-4))$.

The principal difference between this velocity field and the previous is that c_2 does have spatial and temporal components. In other words, the motion of the wave varies with respect to not just time, but also the spacial component which in our case is x. In a physical context, the velocity field c_2 may come from a periodic source such as waves reaching the beach, which would be an example of a nonconstant, or variable, velocity field.

This interpretation agrees with our intuition as our 3D plot visibly resembles how waves crash in a staggered way depending on their location parallel to the shoreline. The periodic nature of wave height is also represented as the quantity of water $u(x,t)$ bounces back to the ocean or sea after it crashes with the shore, causing it to be amplified by other incoming waves in the velocity field. We can conclude that in our interpretation, $x = 10$ represents the shoreline in space that the wave is crashing into. An interesting direction to expand this problem in the future would be to examine the behavior of more dynamic velocity fields, for example, summing the sinusoid with a nonlinear term instead of the Heaviside function which is a unit step function. Different velocity fields could be used to model other physical phenomena and study their behavior in space and time.

2. We are interested in comparing the differences in computational time and complexity between the Gaussian elimination and LU decomposition for solving partial differential equations. In this instance, we will be solving the following PDE:

$$\omega_t + [\phi, \omega] = \nu \nabla^2$$

To solve this differential equation using the two aforementioned numerical methods, we will be discretizing our problem using a meshgrid with 128 points in the x and y directions. Using Python's time() module, we are able to record the time required to perform our computations. It is important to note that the actual values for these computation times are dependent on both hardware and the internal state of the computer, but qualitative behavior should be the same across computing environments.

The time required to solve the PDE using Gaussian elimination was 2.22165 seconds. The time required to solve the PDE using LU decomposition was 0.09318 seconds. The ratio between these computation times claims that Gaussian elimination was approximately 23.8 times slower. Additional executions of the code show that the ratio of computation time between Gaussian elimination and LU decomposition can vary from as low as 22 times slower to as high as 33 times slower.

It is crucial to note that while the time complexities of these numerical methods are of the same order for finding an initial solution of some linear system $Ax = b$, their complexities diverge when repeated solutions must be found. Gaussian elimination has a time complexity of $O(n^3)$, or cubic complexity. While LU decomposition also has a time complexity of $O(n^3)$ for the initial solve, it boasts an entire factor of n more efficient, $O(n^2)$, for repeated solves of the system by splitting the matrix A into a lower triangular matrix L, and an upper triangular matrix U. Thus our result matches our expectations as the solve time for LU decomposition was significantly more efficient than that of Gaussian elimination. An interesting next step for future work on this problem could be to examine the computation times for additional methods in numerical linear algebra such as the Gauss-Seidel iteration and the Jacobi iteration. Additionally, the scope of this problem could be further widened to include entirely different partial differential equations whose time to solve could be studied.

1. Code:

```
x_evals = np.arange(-10, 10, 0.1)
n = 200
b = np.ones((n))
Bin = np.array([-b, b, -b, b])
d = np.array([-1,1, n-1, 1-n])
matrix1 = scipy.sparse.spdiags(Bin, d, n, n, format='csc')/(2*0.1)
A1 = matrix1.todense()

## b

u0 = np.exp(-(x_evals-5)**2)
def advection1(t, u, A, c):
    return -c*(A@u)

sol1 = scipy.integrate.solve_ivp(advection1, [0, 10], u0, t_eval = \
np.arange(0, 10 + 0.5, 0.5), args = (matrix1, -0.5))
A2 = sol1.y

## c

def advection2(t, u, A, cFunc, x):
    c = cFunc(t, x)
    return c*(A@u)

def cFunc(t, x):
    return (1 + 2*np.sin(5*t) - np.heaviside(x - 4, 0))

sol2 = scipy.integrate.solve_ivp(advection2, [0, 10], u0, t_eval = \
np.arange(0, 10 + 0.5, 0.5), args = (matrix1, cFunc, x_evals))
A3 = sol2.y
```

```
## 3D Plot

X, T = np.meshgrid(x_evals, sol1.t)
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(X, T, sol1.y.T, cmap='magma')
ax.plot3D(x_evals, 0*x_evals, u0,'-r',linewidth=5)
ax.set_xlabel('x')
ax.set_ylabel('time')
ax.set_zlabel('height')
ax.set_title('Constant Velocity Field')

X, T = np.meshgrid(x_evals, sol2.t)
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(X, T, sol2.y.T, cmap='magma')
ax.plot3D(x_evals, 0*x_evals, u0,'-r',linewidth=5)
ax.set_xlabel('x')
ax.set_ylabel('time')
ax.set_zlabel('height')
ax.set_title('Variable (Nonconstant) Velocity Field')
```

2. Code:

```
m = 64
n = m**2
e1 = np.ones(n)
e0 = e1.copy()
e0[0] = -0.5
Low1 = np.tile(np.concatenate((np.ones(m-1), [0])), (m,))
Low2 = np.tile(np.concatenate(([1], np.zeros(m-1))), (m,))
Up1 = np.roll(Low1, 1)
Up2 = np.roll(Low2, m-1)
matrix2 = scipy.sparse.spdiags([e1, e1, Low2, Low1, -4*e0, Up1, Up2, \
e1, e1], [-(n-m), -m, -m+1, -1, 0, 1, m-1, m, (n-m)], n, n,
format='csc')/((20/64)**2)
```

```
A4 = matrix2.todense()

m = 64
n = m**2
e1 = np.ones(n)
matrix3 = scipy.sparse.spdiags([e1, -e1, e1, -e1], [-(n-m), -m, m, \
(n-m)], n, n, format='csc')/(2*(20/64))
A5 = matrix3.todense()

m = 64
n = m**2
e1 = np.ones(n)
Lower1 = np.tile(np.concatenate((np.ones(m-1), [0])), (m,))
Lower2 = np.tile(np.concatenate(([1], np.zeros(m-1))), (m,))
Upper1 = np.roll(Lower1, 1)
Upper2 = np.roll(Lower2, m-1)
matrix4 = scipy.sparse.spdiags([Lower2, -Lower1, Upper1, -Upper2], \
[1-m, -1, 1, m-1], n, n, format='csc')/(2*(20/64))
A6 = matrix4.todense()

## b

x_span = np.linspace(-10, 10, num=64, endpoint=False)
y_span = np.linspace(-10, 10, num=64, endpoint=False)
X, Y = np.meshgrid(x_span, y_span)
w0 = np.exp(-2*X**2 - ((Y**2)/20))
w0_temp = (w0.T).flatten()
t_span = np.arange(0, 4 + 0.5, 0.5)

def omegaFunc1(t, omega):
    psi = scipy.sparse.linalg.spsolve(matrix2, omega)
    w_t = ((0.001*matrix2@omega) - (matrix3@psi)*(matrix4@omega) + \
    (matrix4@psi)*(matrix3@omega))
    return w_t

start1 = time.time()
sol3 = scipy.integrate.solve_ivp(omegaFunc1, [0, 4], w0_temp, \
t_eval = t_span)
end1 = time.time()
print("Gaussian elimination computation time:", end1 - start1)
A7 = sol3.y.T

LU = scipy.sparse.linalg.splu(matrix2)
def omegaFunc2(t, omega):
    psi = LU.solve(omega)
    w_t = ((0.001*matrix2@omega) - (matrix3@psi)*(matrix4@omega) + \
    (matrix4@psi)*(matrix3@omega))
    return w_t

start2 = time.time()
sol4 = scipy.integrate.solve_ivp(omegaFunc2, [0, 4], w0_temp, \
t_eval = t_span)
end2 = time.time()

A8 = sol4.y.T
A9 = np.zeros((9, 64, 64))
for i in range(9):
    A9[i] = A8[i].reshape(64, 64)
```