Numerical methods in phycics - week 2 assignment

By Oliver Sørensen (qzk375)

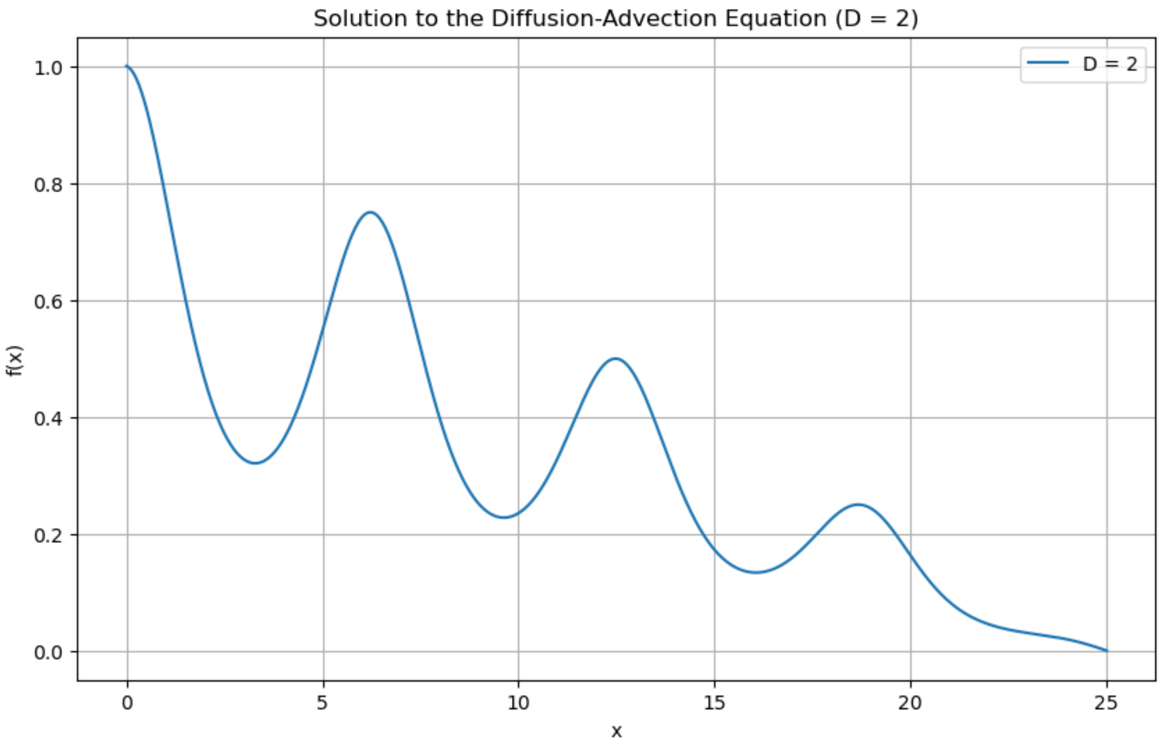
3.6 Diffusion–Advection

a

Use a second order finite difference scheme to solve the diffusion- advection with $D = 2$ and $v(x) = -sinx$ on $[0, 25]$ with boundary conditions $f(0) = 1$ and $f(25) = 0$. Use $N = 1000$ grid points.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import solve

def diffusion_advection(D):
    N = 1000
    L = 25
    dx = L / N
    x = np.linspace(0, L, N)

    # Velocity field
    v = -np.sin(x)


    # diag(v(x))
    V = np.diag(v)
    A1 = np.zeros((N, N))

    # used: https://en.wikipedia.org/wiki/Finite_difference_coefficient
    # foward difference
    A1[0,0], A1[0,1], A1[0,2] = -3/2, 2, -1/2
    for i in range(1,N-1):
        # central difference
        A1[i,i-1], A1[i,i], A1[i,i+1] = -1/2, 0, 1/2
    # backward difference
    A1[N-1,N-3], A1[N-1,N-2], A1[N-1,N-1] = 1/2, -2, 3/2

    A1 = (1 / dx) * A1

    A2 = np.zeros((N, N))
    A2[0,0], A2[0,1], A2[0,2] = 1, -2, 1 # foward difference
    for i in range(1,N-1):
        # central difference
        A2[i,i-1], A2[i,i], A2[i,i+1] = 1, -2, 1
    # backward difference
    A2[N-1,N-3], A2[N-1,N-2], A2[N-1,N-1] = 1, -2, 1

    A2 = (D / dx**2) * A2
```

```python
    # T1 = A1 * V
    T1 = A1 @ V

    # second boundary condition (f(25) = 0)
    b = np.zeros(N)

    # first boundary condition (f(0) = 1),
    b[0] = 1

    A = A2 - T1
    A[0,:] = 0
    A[0,0] = 1
    A[-1,:] = 0
    A[-1,-1] = 1

    f = np.linalg.solve(A,b)
    return x, f

D = 2
x, f = diffusion_advection(D)
# Plotting the result
plt.figure(figsize=(10, 6))
plt.plot(x, f, label=f'D = {D}')

plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('Solution to the Diffusion-Advection Equation (D = 2)')
plt.legend()
plt.grid(True)
plt.show()
```



b

Explain the shape of $f(x)$. Does it make sense compared to the physical

interpretation of the diffusion-advection equation?

Diffusion Term that spreads particles: $D\frac{d^2f(x)}{dx^2}$

Advection Term, particles are transported by the velocity field: $\frac{d}{dx}(v(x)f(x))$

The shape of f(x) is determined by the balance between diffusion and advection. The strength of the diffusion is determined by D, therefore a high D would mean that diffusion will dominate. Whereas a lower D would mean that advection dominates. Since advection term transports particles by the velocity field it makes sense that we would see some sort of sinusoidal motion since $v(x) = -sin(x). For$D = 2$ we will stil see the effect of advection and therefore we would also expect some sinusoidal motion.

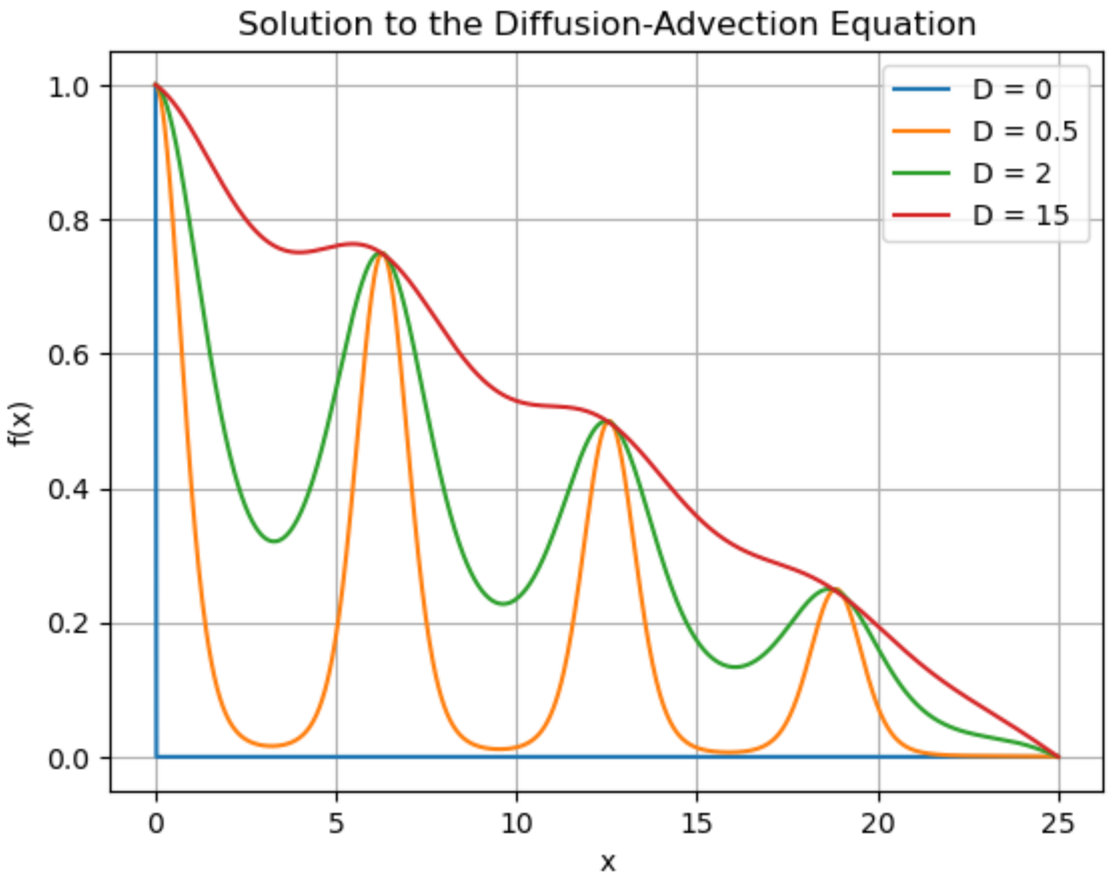
Because of the boundary case we would expect $f(0) = 1$ and $f(25) = 0$

C

Explain what happens for $D \to 0$ and $D \to \infty$. Plot for instance $D = 0.5$ and $D = 15$ and compare to $D = 2$. Would your code work for D = 0?

In [ ]:

```
D_values = [0, 0.5, 2, 15]

for D in D_values:
    x, f = diffusion_advection(D)
    plt.plot(x, f, label=f'D = {D}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title('Solution to the Diffusion-Advection Equation')
    plt.legend()
    plt.grid(True)
plt.show()
```



For $D \to 0$ Diffusion term disappears, the equation becomes dominated by advection, leading to a sharp and discontinuous solution. However the code still works for $D = 0$, we see $f(x) = 0$ across the entire domain. This is because the equation becomes

$$-\frac{d}{dx}(v(x)f(x)) = 0$$

This means that

$$v(x)f(x) = C$$

Where C is a constant.

For $v(x)f(x) = C$, $f(x)$ must be equal to 0 across the entire domain.

For $D \to \infty$ Diffusion dominates, leading to a much smoother solution and less sinusodial motion.

4.1 1D Heat Equation

a

Use a second order finite difference scheme to turn the above into a linear algebra problem using N = 1000 grid points.

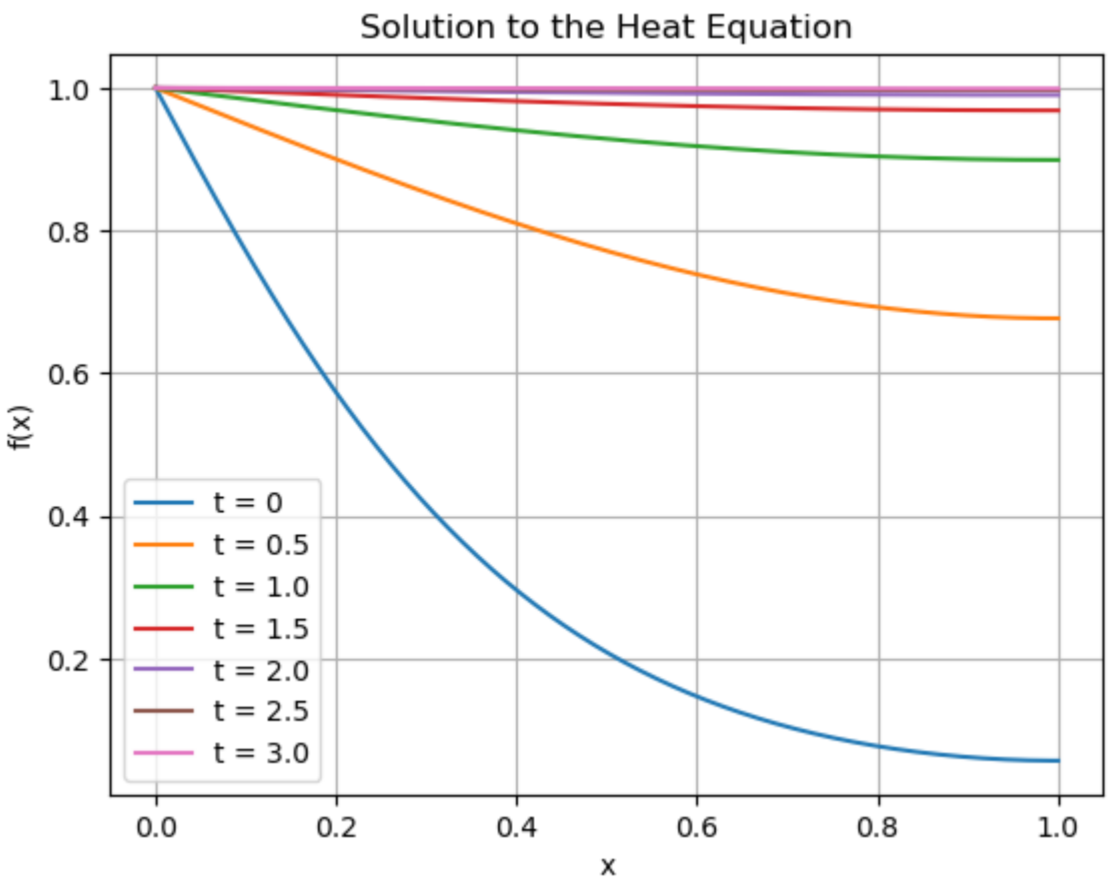
In [ ]:
```
def heat_equation(dt = 0.05, t_values = [0, 0.5, 1, 1.5, 2, 2.5, 3]):
```

```
    N = 1000
    x = np.linspace(0, 1, N)
    dx = x[1] - x[0]
    t = 0

    A1 = np.eye(N, N) # identity matrix
    A2 = np.zeros((N, N))

    # foward difference
    A2[0,0], A2[0,1], A2[0,2] = 1, -2, 1
    for i in range(1,N-1):
        # central difference
        A2[i,i-1], A2[i,i], A2[i,i+1] = 1, -2, 1
    # backward difference
    A2[N-1,N-3], A2[N-1,N-2], A2[N-1,N-1] = 1, -2, 1

    A2 = (dt / dx**2) * A2

    A = A1 - A2

    A[0,:] = 0
    A[0,0] = 1
    A[-1,:] = 0
    A[-1,-1] = 3/2
    A[-1,-2] = -2
    A[-1,-3] = 1/2


    f = np.exp(-5*x)
    b = f.copy()
    # boundary conditions
    b[0] = 1
    b[-1] = 0
    while t <= 3:
        t = round(t, 2)
        f = np.linalg.solve(A,b)
        b = f.copy() # update each time step
        b[0] = 1 # first boundary condition
        b[-1] = 0 # second boundary condition
        if t in t_values:
            plt.plot(x, f, label=f't = {t}')
        t += dt # update time step
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title('Solution to the Heat Equation')
    plt.legend()
    plt.grid(True)
    plt.show()
```

b

Solve the system using Δt = 0.05 for $t \in [0, 3]$ and plot curves for $t \in \{0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0\}$.

In [ ]:
```
heat_equation(dt = 0.05, t_values = [0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0])
```



4.3 Shade sail

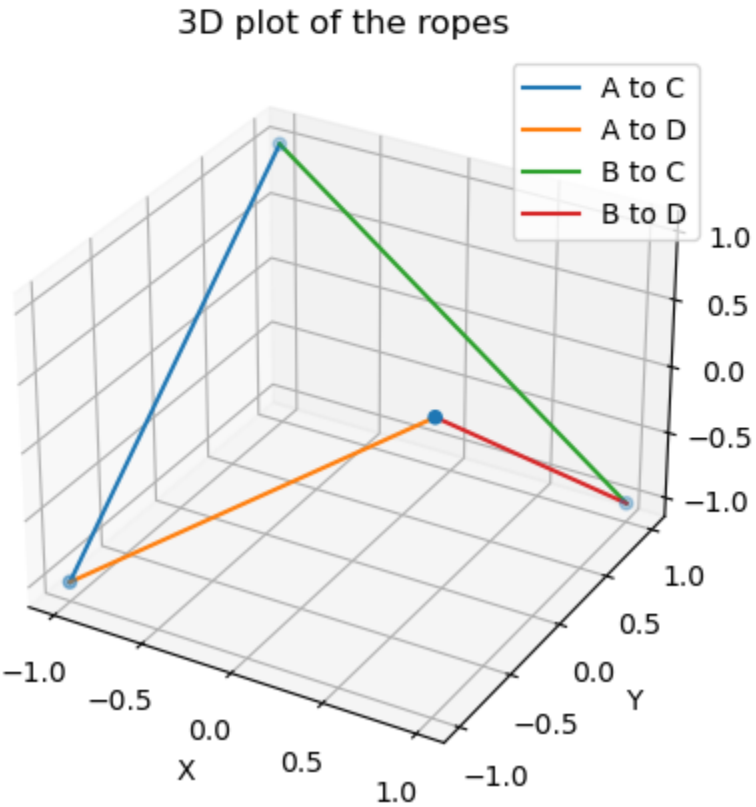
a

Plot the four ropes in a 3D plot.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

def plot_ropes(ax):
    A = np.array([-1, -1, -1])
    B = np.array([1, 1, -1])
    C = np.array([-1, 1, 1])
    D = np.array([1, -1, 1])

    # Plotting the ropes
    ax.plot([A[0], C[0]], [A[1], C[1]], [A[2], C[2]], label='A to C')
    ax.plot([A[0], D[0]], [A[1], D[1]], [A[2], D[2]], label='A to D')
    ax.plot([B[0], C[0]], [B[1], C[1]], [B[2], C[2]], label='B to C')
    ax.plot([B[0], D[0]], [B[1], D[1]], [B[2], D[2]], label='B to D')
    ax.legend()

    ax.scatter([A[0], B[0], C[0], D[0]], [A[1],
    B[1], C[1], D[1]], [A[2], B[2], C[2], D[2]])

    ax.set_xlabel('X')
    ax.set_ylabel('Y')
    ax.set_zlabel('Z')
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
plot_ropes(ax)
ax.set_title('3D plot of the ropes')

plt.show()
```



b

Write down the boundary conditions.

all values at the edge of the domain are known function. looking at each rope, we see the following:

$f_1(x=-1,y)=x+y$ (A to C)

$f_2(x,y=-1)=x+y$ (A to D)

$f_3(x,y=1)=-x+y$ (B to C)

$f_4(x=1,y)=x-y$ (B to D)

The domain of x,y is: $x,y \in [(-1,-1);(1,1)]$ Therefore we get the following boundary conditions:

$$\begin{aligned} f(-1,y) &= y, \qquad && \text{for } y \in [-1,1], \\ f(x,-1) &= x, && \text{for } y \in [-1,1], \\ f(x,1) &= -x, && \text{for } x \in [-1,1], \\ f(1,y) &= -y, && \text{for } x \in [-1,1]. \end{aligned}$$

C

Solve Eq. (4.8) with the boundary conditions.

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
# Finite difference method, 4 points.
def laplace(N = 4):
    x = np.linspace(-1, 1, N)
    y = np.linspace(-1, 1, N)
    dx = x[1] - x[0]
    X, Y = np.meshgrid(x, y)

    A = np.zeros((N**2, N**2))
    b = np.zeros(N**2)

    for i in range(N):
        for j in range(N):
            n = i * N + j
            # Boundary conditions:
            if i == 0: # x = -1
                A[n,n] = 1
                b[n] = y[j]
            elif j == 0: # y = -1
                A[n,n] = 1
                b[n] =  x[i]
            elif j == N-1: # y = 1
                A[n,n] = 1
                b[n] = - x[i]
            elif i == N-1: # x = 1
                A[n,n] = 1
                b[n] = - y[j]
            else:
                # Interior points
                # Using equation 4.27
```

```
                A[n,n] = - 4/dx**2 # center -4f(xi, yj)
                A[n, n + N] = 1/dx**2 # up  f(xi, yj+1)
                A[n, n - N] = 1/dx**2 # down f(xi, yj-1)
                A[n, n+1] = 1/dx**2 # right f(xi+1, yj)
                A[n, n-1] = 1/dx**2 # left f(xi-1, yj)
                b[n] = 0
    f = np.linalg.solve(A, b)
    return X, Y, f

X, Y, f = laplace(4)
Z = f.reshape(X.shape)
```

d

Plot your solution together with the ropes.

In [ ]:
```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z)
plot_ropes(ax)
plt.show()
```