# HW8

May 12, 2023

# 1 Exercise Set 8

## 1.1 Mohaddeseh Mozaffari

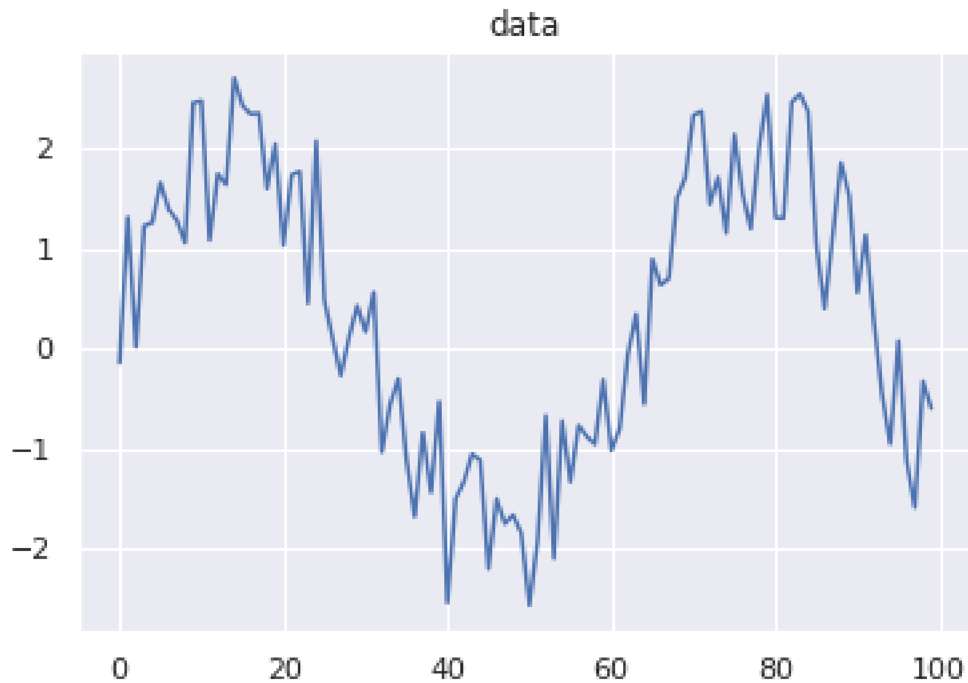
[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

# 2 Load data

[ ]:
```
data = np.loadtxt("/home/mohaddeseh/Documents/Programing/Computational/HW8/
    ↪dataprofile.txt")
```

[ ]:
```
data = data[:,1]
```

[ ]:
```
plt.plot(data)
plt.title("data")
plt.show()
```




## 3 Q1:

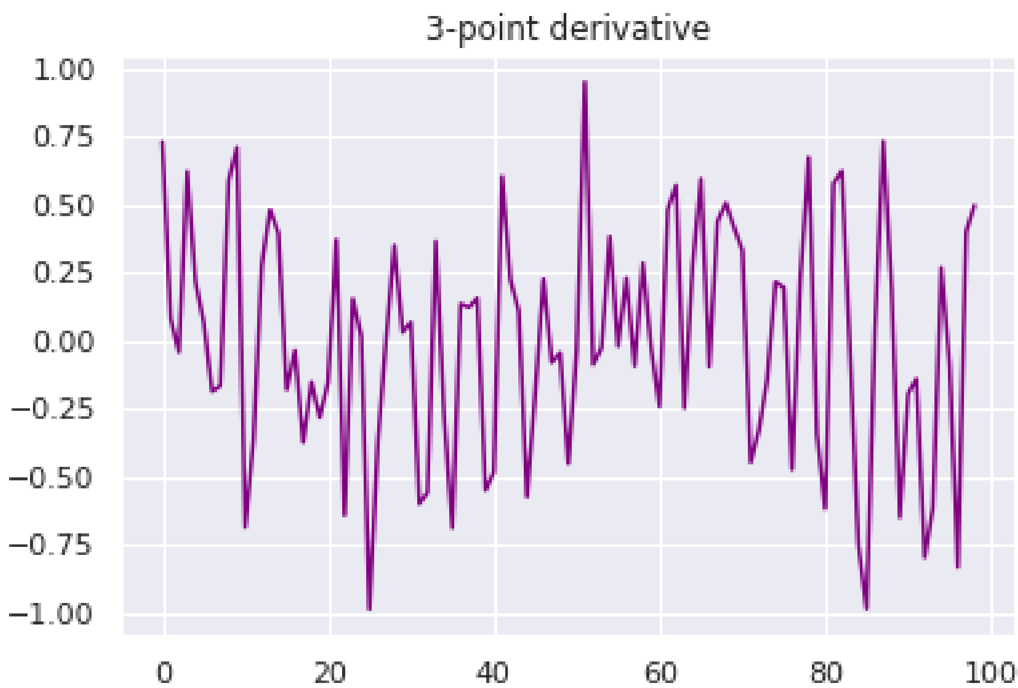
| $N$ | $N$-point stencil Central Differences |
|---|---|
| 3 | $\dfrac{f_1 - f_{-1}}{2h}$ |
| 5 | $\dfrac{f_{-2} - 8f_{-1} + 8f_1 - f_2}{12h}$ |
| 7 | $\dfrac{-f_{-3} + 9f_{-2} - 45f_{-1} + 45f_1 - 9f_2 + f_3}{60h}$ |
| 9 | $\dfrac{3f_{-4} - 32f_{-3} + 168f_{-2} - 672f_{-1} + 672f_1 - 168f_2 + 32f_3 - 3f_4}{840h}$ |

### 3.1 3-point:

[ ]:
```
def derivative_3p(f):
    """ompute the derivative of signal with 3-point neighbors in central␣
 ↪difference formula

    Args:
        f (list or array): data

    Returns:
        1d_array: derivative of data
    """
    h = 1
    N = len(f)
    dr = []
    dr.append((f[1]-f[0])/(2*h))
    for i in range(1,N-1):
        dr.append((f[i+1]-f[i-1])/(2*h))
    return np.array(dr)
```

[ ]:
```
d3 = derivative_3p(data)
```

[ ]:
```
plt.plot(d3, color="purple")
plt.title("3-point derivative")
plt.show()
```

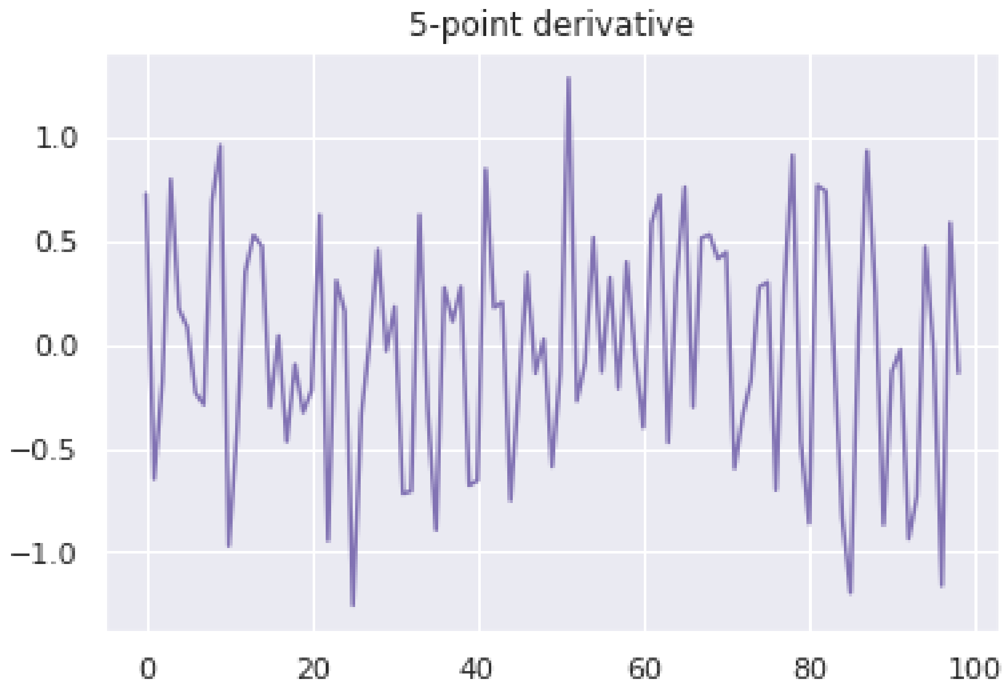


### 3.2 5-point:

[ ]:
```
def derivative_5p(f):
    """ompute the derivative of signal with 5-point neighbors in central␣
 ↪difference formula

    Args:
        f (list or array): data

    Returns:
        1d_array: derivative of data
    """
    h = 1
    N = len(f)
    dr = []
    dr.append((f[1]-f[0])/(2*h))
    dr.append((f[2]-f[1])/(2*h))
    for i in range(2,N-2):
        dr.append((f[i-2]-8*f[i-1]+8*f[i+1]-f[i+2])/(12*h))
    dr.append((f[N-1]-f[N-2])/(2*h))
    return np.array(dr)
```

[ ]:
```
d5 = derivative_5p(data)
```

[ ]:
```
plt.plot(d5, color="m")
plt.title("5-point derivative")
plt.show()
```

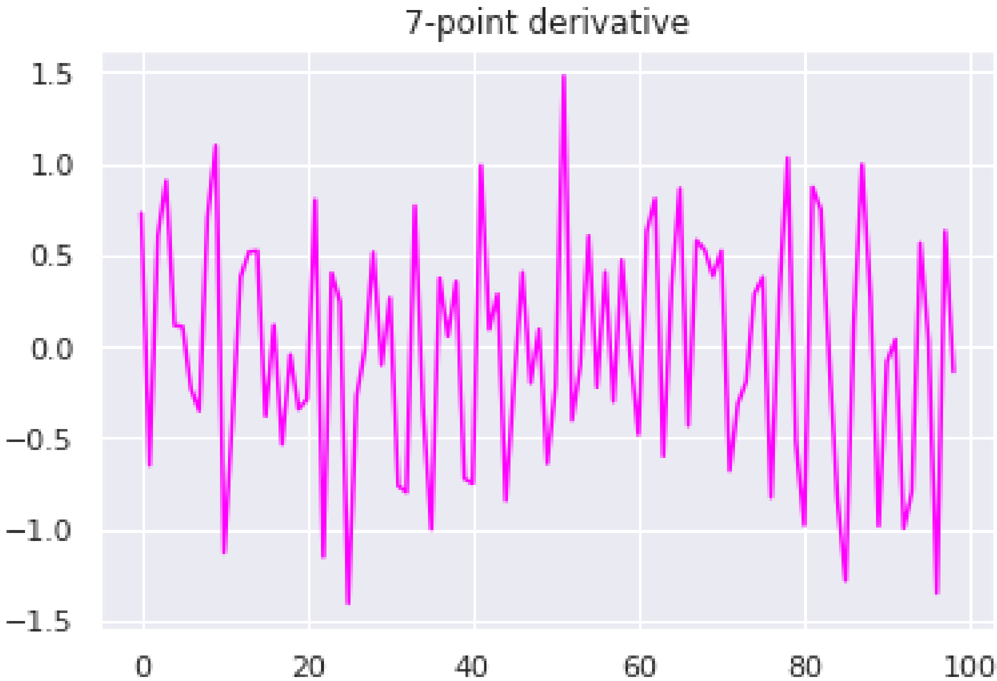



### 3.3 7-point:

```
[ ]: def derivative_7p(f):
         """ompute the derivative of signal with 7-point neighbors in central␣
      ↪difference formula

         Args:
             f (list or array): data

         Returns:
             1d_array: derivative of data
         """
         h = 1
         N = len(f)
         dr = []
         dr.append((f[1]-f[0])/(2*h))
         dr.append((f[2]-f[1])/(2*h))
         dr.append((f[3]-f[2])/(2*h))
         for i in range(3,N-3):
             dr.append((-f[i-3]+9*f[i-2]-45*f[i-1]+45*f[i+1]-9*f[i+2]+f[i+3])/(60*h))
         dr.append((f[N-2]-f[N-3])/(2*h))
         dr.append((f[N-1]-f[N-2])/(2*h))
         return np.array(dr)
```

[ ]:
```python
d7 = derivative_7p(data)
```

[ ]:
```python
plt.plot(d7, color="magenta")
plt.title("7-point derivative")
plt.show()
```



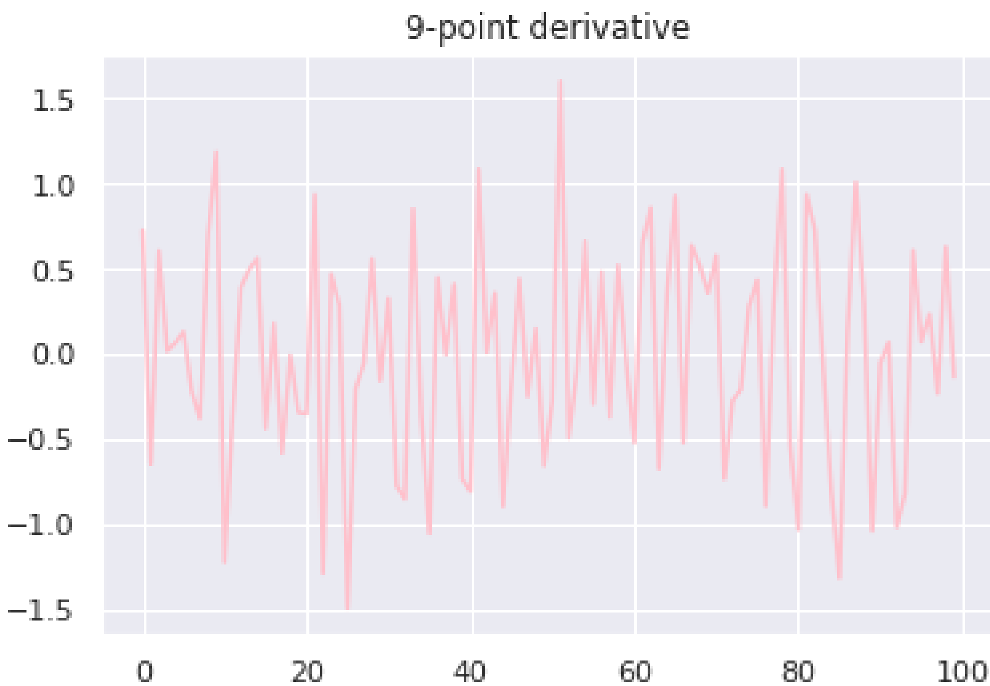

### 3.4 9-point:

[ ]:
```python
def derivative_9p(f):
    """ompute the derivative of signal with 9-point neighbors in central␣
 ↪difference formula

    Args:
        f (list or array): data

    Returns:
        1d_array: derivative of data
    """
    h = 1
    N = len(f)
    dr = []
    dr.append((f[1]-f[0])/(2*h))
    dr.append((f[2]-f[1])/(2*h))
    dr.append((f[3]-f[2])/(2*h))
```

```
    dr.append((f[4]-f[3])/(2*h))
    for i in range(4,N-4):
        dr.
↪append((3*f[i-4]-32*f[i-3]+168*f[i-2]-672*f[i-1]+672*f[i+1]-168*f[i+2]+32*f[i+3]-3*f[i+4])/
↪(840*h))
    dr.append((f[N-4]-f[N-3])/(2*h))
    dr.append((f[N-3]-f[N-4])/(2*h))
    dr.append((f[N-2]-f[N-3])/(2*h))
    dr.append((f[N-1]-f[N-2])/(2*h))
    return np.array(dr)
```

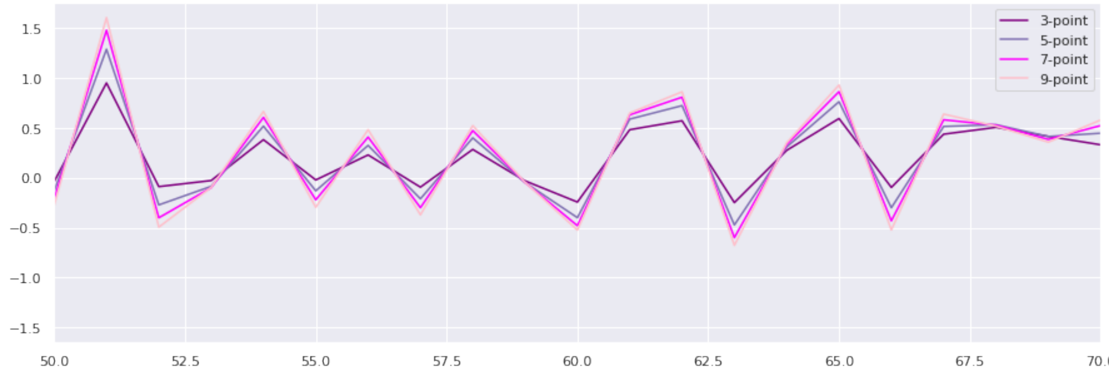
[ ]:
```
d9 = derivative_9p(data)
```

[ ]:
```
plt.plot(d9, color="pink")
plt.title("9-point derivative")
plt.show()
```




[ ]:
```
plt.figure(figsize=(15,5))
plt.plot(d3, color="purple", label="3-point")
plt.plot(d5, color="m", label="5-point")
plt.plot(d7, color="magenta", label="7-point")
plt.plot(d9, color="pink", label="9-point")
plt.legend()
plt.xlim(50,70)
```

```
plt.show()
```




# 4 Q2:

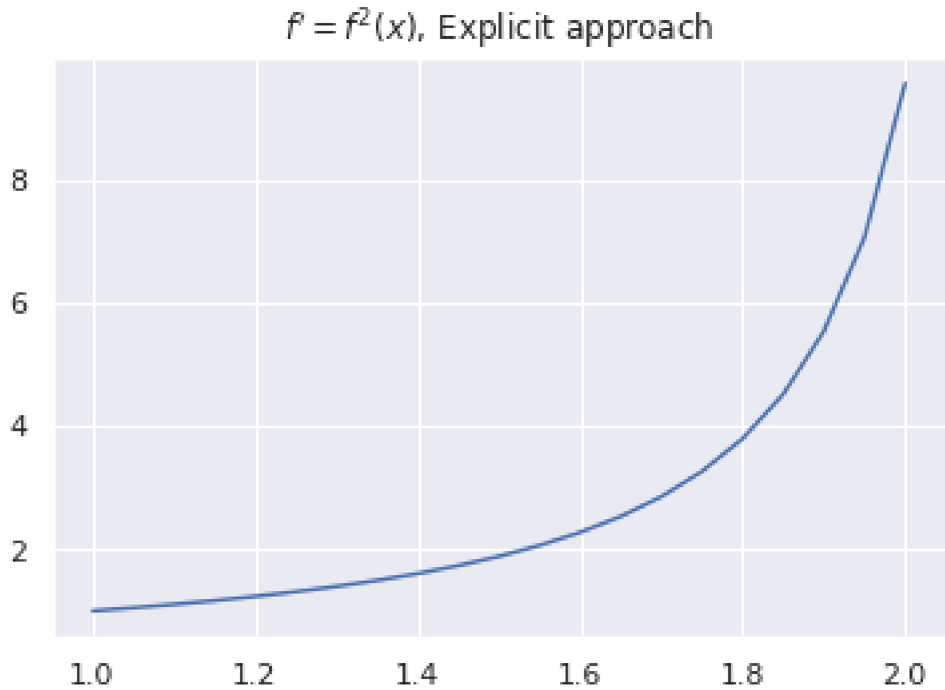
## 4.1 A)

[ ]:
```
#initial condition
dx = 0.05
f1 = 1
X = np.arange(1, 2.05, dx)
N = len(X)
```

### 4.1.1 Explicit approach

[ ]:
```
fe = np.zeros(N)
fe[0] = f1
for i in range(N-1):
    fe[i+1] = fe[i]+ dx*(fe[i]**2)
```

[ ]:
```
plt.plot(X, fe)
plt.title(r"$f'=f^2(x)$, Explicit approach")
plt.show()
```

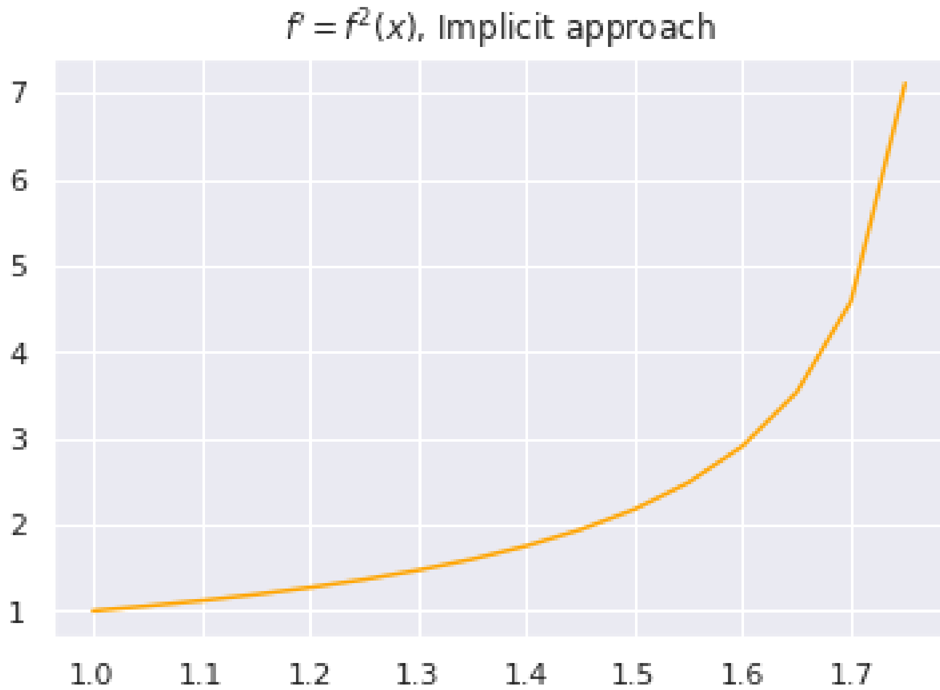



#### 4.1.2 Implicit approach

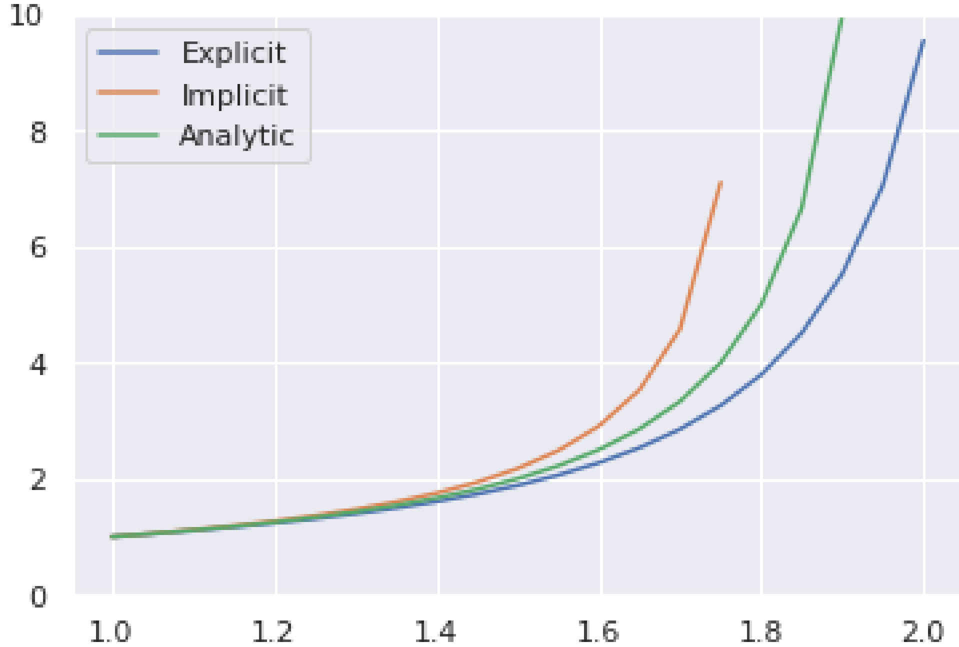
[ ]:
```
fi = np.zeros(N)
fi[0] = f1
for i in range(N-1):
    fi[i+1] = (1 - ((1 - (4*fi[i]*dx))**0.5))/(2*dx)
```

```
/tmp/ipykernel_27044/208837291.py:4: RuntimeWarning: invalid value encountered
in double_scalars
  fi[i+1] = (1 - ((1 - (4*fi[i]*dx))**0.5))/(2*dx)
```

[ ]:
```
plt.plot(X, fi, color="orange")
plt.title(r"$f'=f^2(x)$, Implicit approach")
plt.show()
```




[ ]:
```
plt.plot(X,fe, label="Explicit")
plt.plot(X,fi, label="Implicit")
plt.plot(X[:-1], (1/(2-X))[:-1], label="Analytic")
plt.ylim(0,10)
plt.legend()
plt.show()
```




As we can see, the explicit approach is closest to analytic approach.

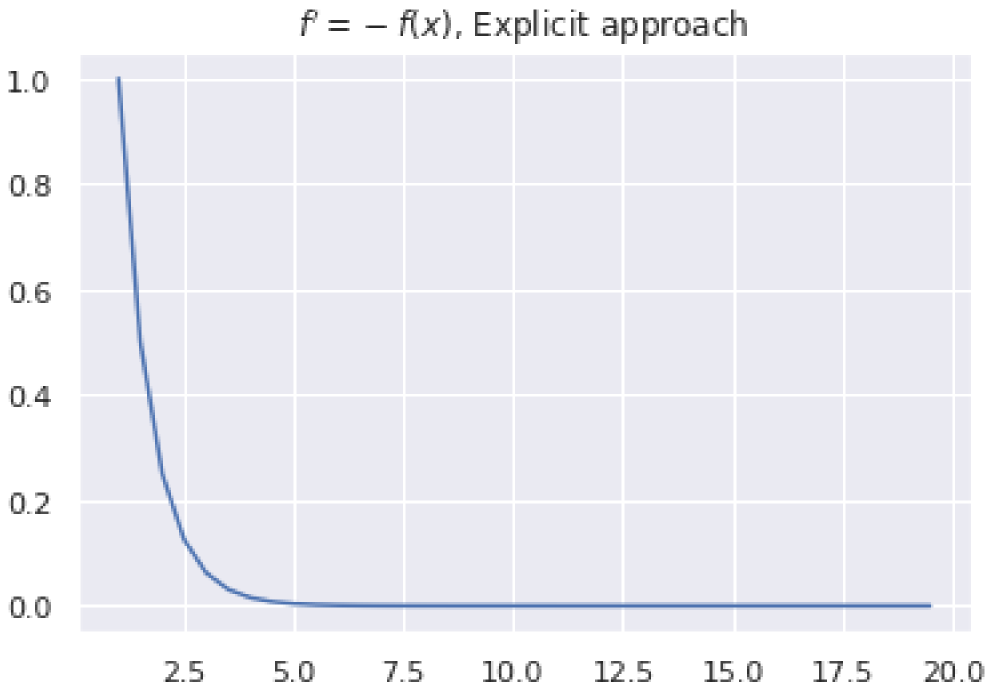
## 4.2 B)

[ ]:
```
#initial condition
dx = 0.5
f1 = 1
X = np.arange(1,20,dx)
N = len(X)
```

### 4.2.1 Explicit approach

[ ]:
```
fe = np.zeros(N)
fe[0] = f1
for i in range(N-1):
    fe[i+1] = (1-dx)*fe[i]
```

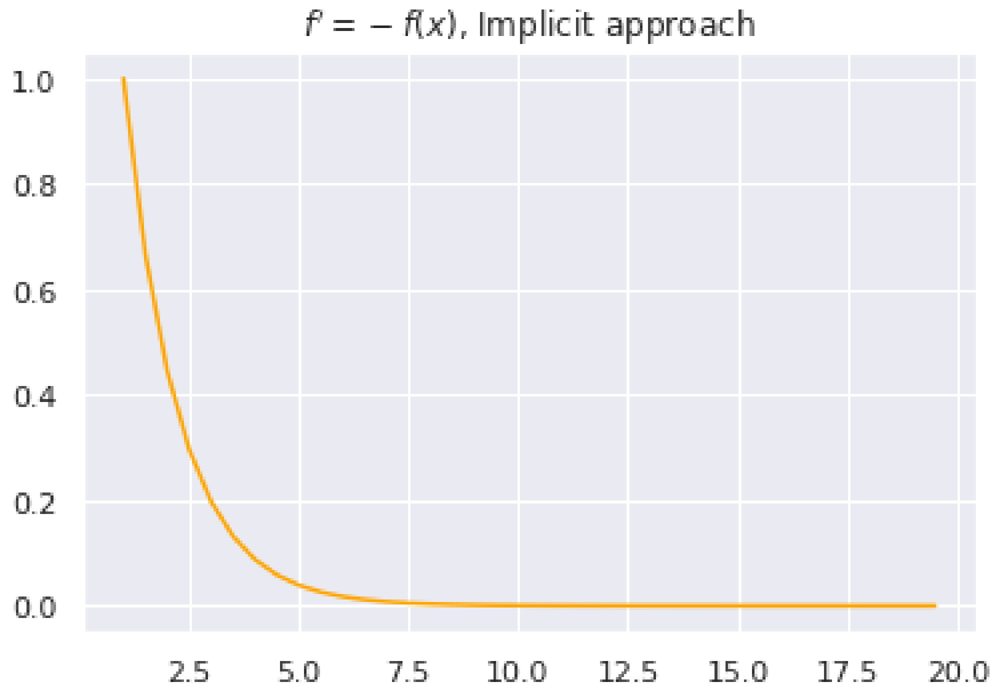
[ ]:
```
plt.plot(X,fe)
plt.title(r"$f'=-f(x)$, Explicit approach")
plt.show()
```



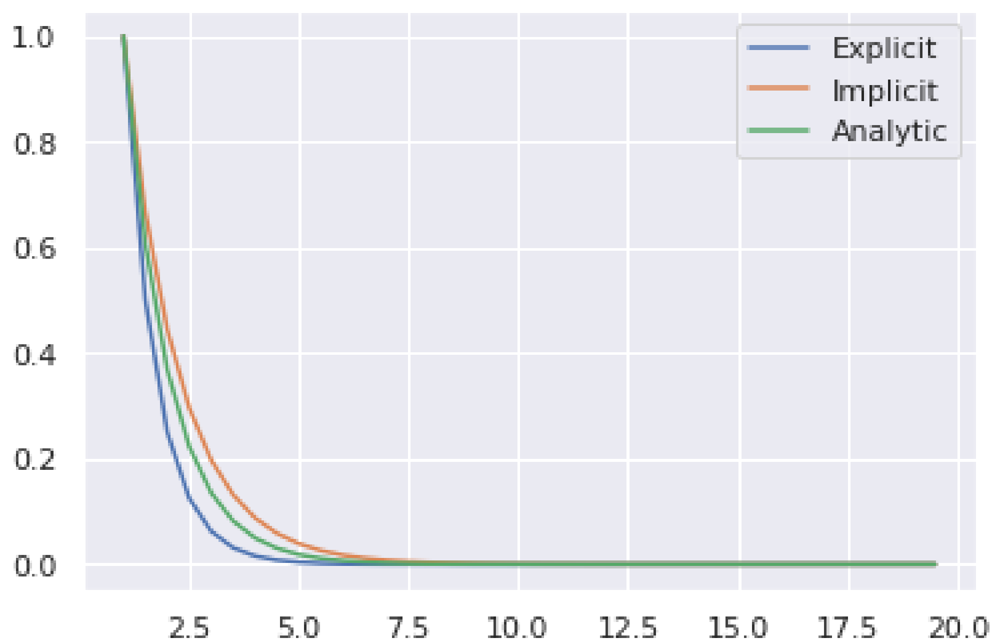

#### 4.2.2 Implicit approach

[ ]:
```
fi = np.zeros(N)
fi[0] = f1
for i in range(N-1):
    fi[i+1] = (fi[i])/(1+dx)
```

[ ]:
```
plt.plot(X,fi, color="orange")
plt.title(r"$f'=-f(x)$, Implicit approach")
plt.show()
```




[ ]:
```
plt.plot(X,fe, label="Explicit")
plt.plot(X,fi, label="Implicit")
plt.plot(X,np.exp(-X+1), label="Analytic")
plt.legend()
plt.show()
```



As we can see, the implicit approach is closest to analytic approach.

# 5 Q3:

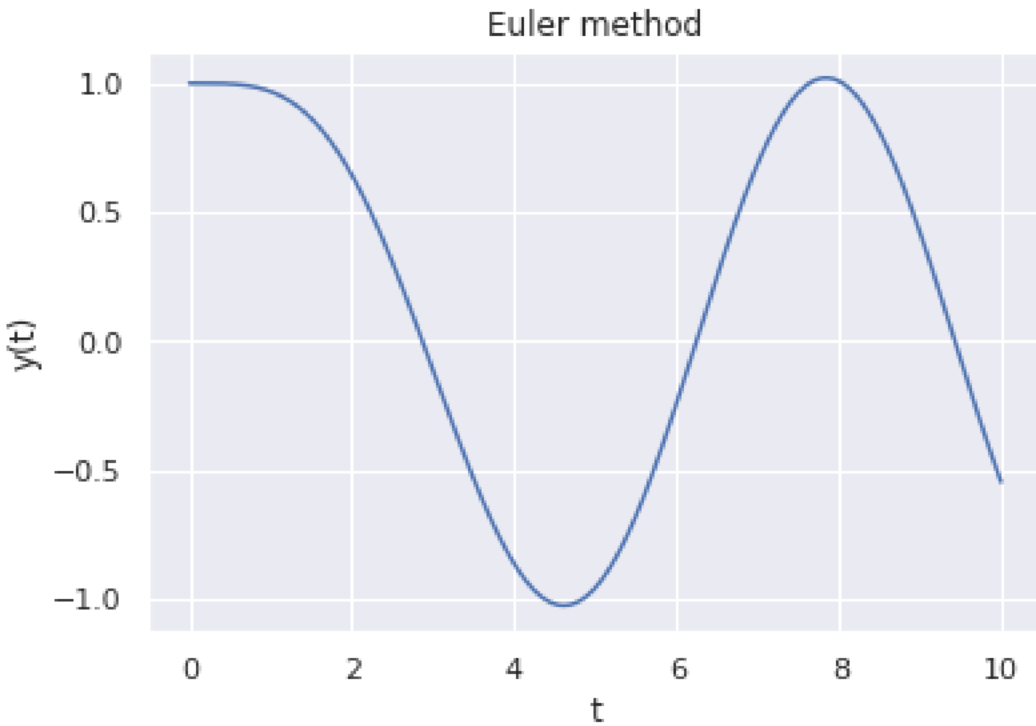
[ ]:
```
a = 1
w2 = 1
w1 = 1
dt = 0.01
T = np.arange(0, 10, dt)
N = len(T)
```

## 5.1 Euler method

[ ]:
```
y = np.zeros(N)
y[0] = 1
v = np.zeros(N)
```

[ ]:
```
for t in range(N-1):
    v[t+1] = v[t] + dt*(np.cos(w1*T[t]) - a*v[t] - w2 * y[t])
    y[t+1] = y[t] + dt*(v[t])
```

[ ]:
```
plt.plot(T,y)
plt.title("Euler method")
plt.xlabel("t")
plt.ylabel("y(t)")
plt.show()
```

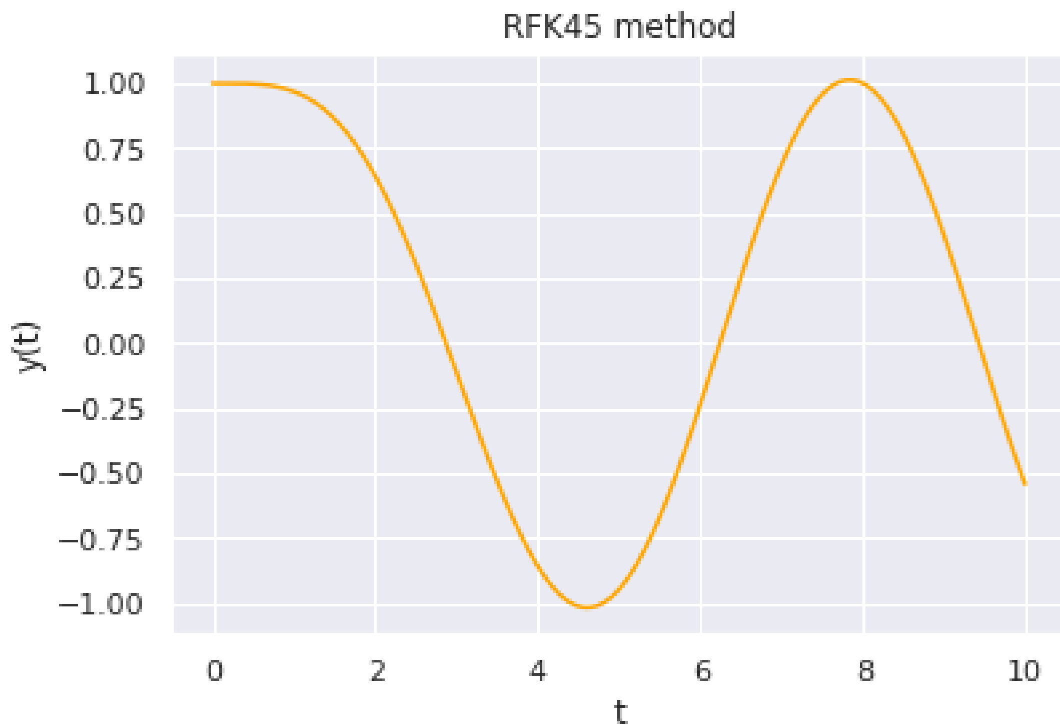



### 5.2 RKF45:

[ ]:
```
Y = np.zeros(N)
Y[0] = 1
V = np.zeros(N)
```

[ ]:
```
for t in range(N-1):
    f1 = V[t]
    k1 = np.cos(w1*T[t]) - w2*Y[t] - (a * f1)

    f2 = V[t] + (dt/2)*k1
    k2 = np.cos(w1*(T[t]+(dt/2))) - w2*(Y[t]+(dt/2)*f1) - f2

    f3 = V[t] + (dt/2)*k2
    k3 = np.cos(w1*(T[t]+(dt/2))) - w2*(Y[t]+(dt/2)*f2) - f3

    f4 = V[t] + (dt/2)*k3
    k4 = np.cos(w1*(T[t]+(dt/2))) - w2*(Y[t]+(dt/2)*f3) - f4

    V[t+1] = V[t] + (dt/6) * (k1+ 2*k2 + 2*k3 + k4)
    Y[t+1] = Y[t] + (dt/6) * (f1+ 2*f2 + 2*f3 + f4)
```

```
[ ]: plt.plot(T,Y, color="orange")
     plt.title("RFK45 method")
     plt.xlabel("t")
     plt.ylabel("y(t)")
     plt.show()
```




## 6 Q4:

$$y''(t) + a y'(t) + \omega^2 y(t) = \cos(\omega, t)$$

$$y''(t) = \frac{y(t+\Delta t) + y(t-\Delta t) - 2y(t)}{(\Delta t)^2}, \quad y'(t) = \frac{y(t+\Delta t) - y(t-\Delta t)}{2\Delta t}$$

$$\Rightarrow \frac{y(t+\Delta t) + y(t-\Delta t) - 2y(t)}{(\Delta t)^2} + \frac{a y(t+\Delta t) - a y(t-\Delta t)}{2\Delta t} + \omega^2 y(t) = \cos(\omega, t)$$

$$\Rightarrow 2y(t+\Delta t) + 2y(t-\Delta t) - 4y(t) + a\Delta t\, y(t+\Delta t) - a\Delta t\, y(t-\Delta t) + 2a\omega^2(\Delta t)^2 y(t) = 2(\Delta t)^2\cos(\omega, t)$$

$$\Rightarrow y(t)\left[-4 - 2a\omega^2(\Delta t)^2\right] = 2(\Delta t)^2\cos(\omega, t) - y(t+\Delta t)\left[2 + a\Delta t\right] - y(t-\Delta t)\left[2 - a\Delta t\right]$$

$$\Rightarrow y(t) = \frac{(2 + a\Delta t)\, y(t+\Delta t) + (2 - a\Delta t)\, y(t-\Delta t) - 2(\Delta t)^2\cos(\omega, t)}{4 + 2a\omega^2(\Delta t)^2}$$

[ ]:
```
a = 1
w2 = 1
w1 = 1
dt = 0.01
T = np.arange(0, 10, dt)
N = len(T)
f = np.random.uniform(-1,1,N)
f[0] = 1
f[-1] = (y[-1] + Y[-1]) /2
```

[ ]:
```
for _ in range(N**2):
    x = np.random.randint(1,N-1)
    f[x] = ((2+a*dt) * y[x+1] + (2-a*dt) * y[x-1] - (2*(dt**2)*np.
 ↪cos(w1*T[x])))/(4+(2*a*w2*dt**2))
```

[ ]:
```
plt.plot(T,f, color="green")
plt.title("Relaxation method")
plt.xlabel("t")
plt.ylabel("y(t)")
plt.show()
```


As we can see, all four method are same result.

# 7 Q5:

[ ]:
```
a = 1
w2 = 1
w1 = 1
N = 1000
Tt = np.linspace(0, 10, N)
h = (Tt[-1] - Tt[0])/N
alpha = 1
beta = (y[-1] + Y[-1])/2
```

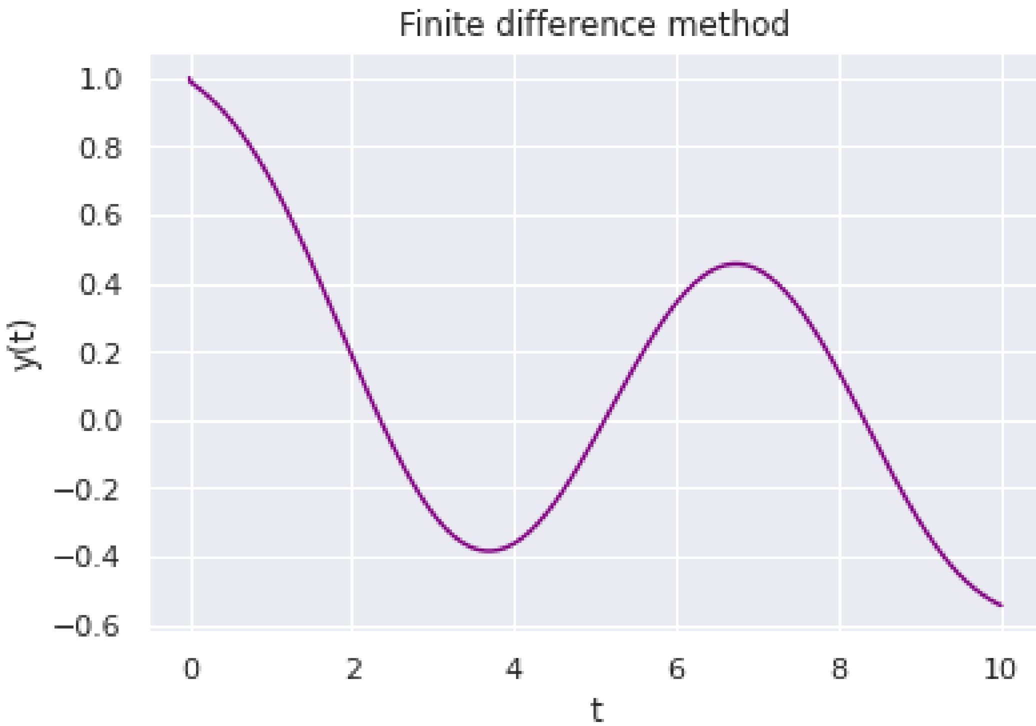
[ ]:
```
L = np.zeros((N-2, N-2))
L[0,0] = 2 + w2 * h**2
L[0,1] = (h/2)*a - 1
for i in range(1, N-3):
    L[i,i] = 2 + w2 * h**2
    L[i,i+1] = (h/2)*a - 1
    L[i,i-1] = (-h/2)*a - 1
L[N-3,N-3] = 2 + w2 * h**2
L[N-3,N-4] = (-h/2)*a - 1
```

[ ]:
```
A = (h**2)* np.cos(w1* Tt)
A = np.delete(A,0)
A = np.delete(A,-1)
A[0] += ((h/2)*(-a) + 1)* alpha
A[-1] += ((h/2)*(-a) + 1)* beta
```

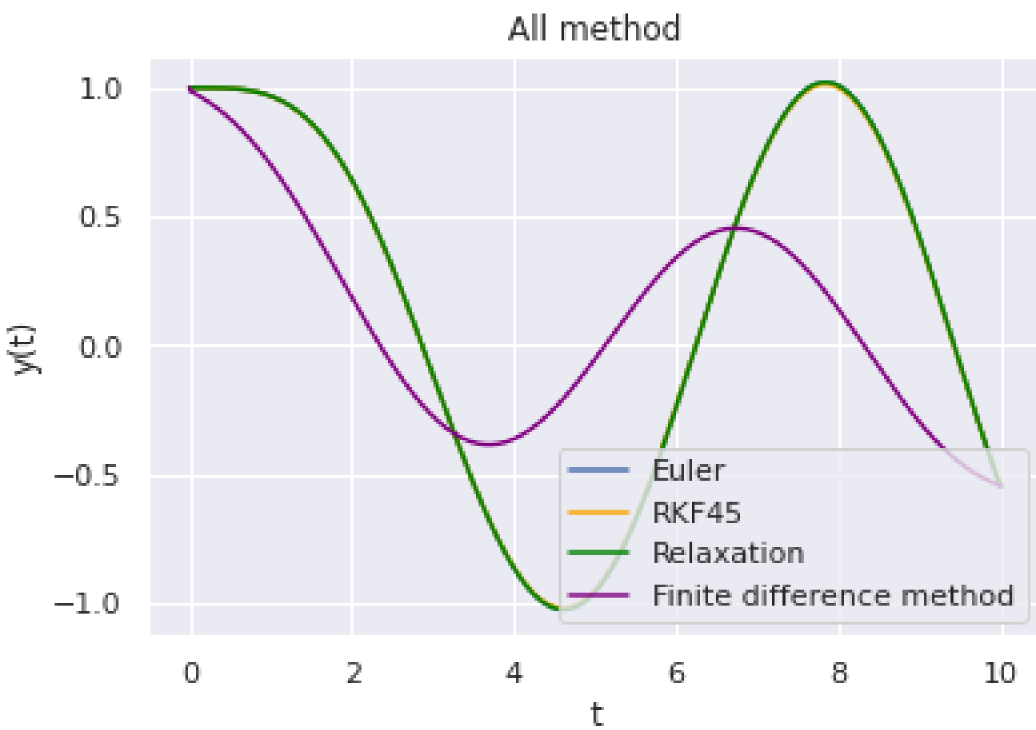
[ ]:
```
y3 = np.linalg.solve(L, A)
```

[ ]:
```
yy = np.zeros(N)
yy[0] = alpha
yy[1:N-1] = y3
yy[N-1] = beta
```

[ ]:
```
plt.plot(Tt, yy, color="purple")
plt.xlabel("t")
plt.ylabel("y(t)")
plt.title("Finite difference method")
plt.show()
```




[ ]:
```
plt.plot(T,y, label="Euler")
plt.plot(T,Y, color="orange", label="RKF45")
plt.plot(T,f, color="green", label="Relaxation")
plt.plot(Tt, yy, color="purple", label="Finite difference method")
plt.title("All method")
plt.legend()
plt.xlabel("t")
plt.ylabel("y(t)")
plt.show()
```



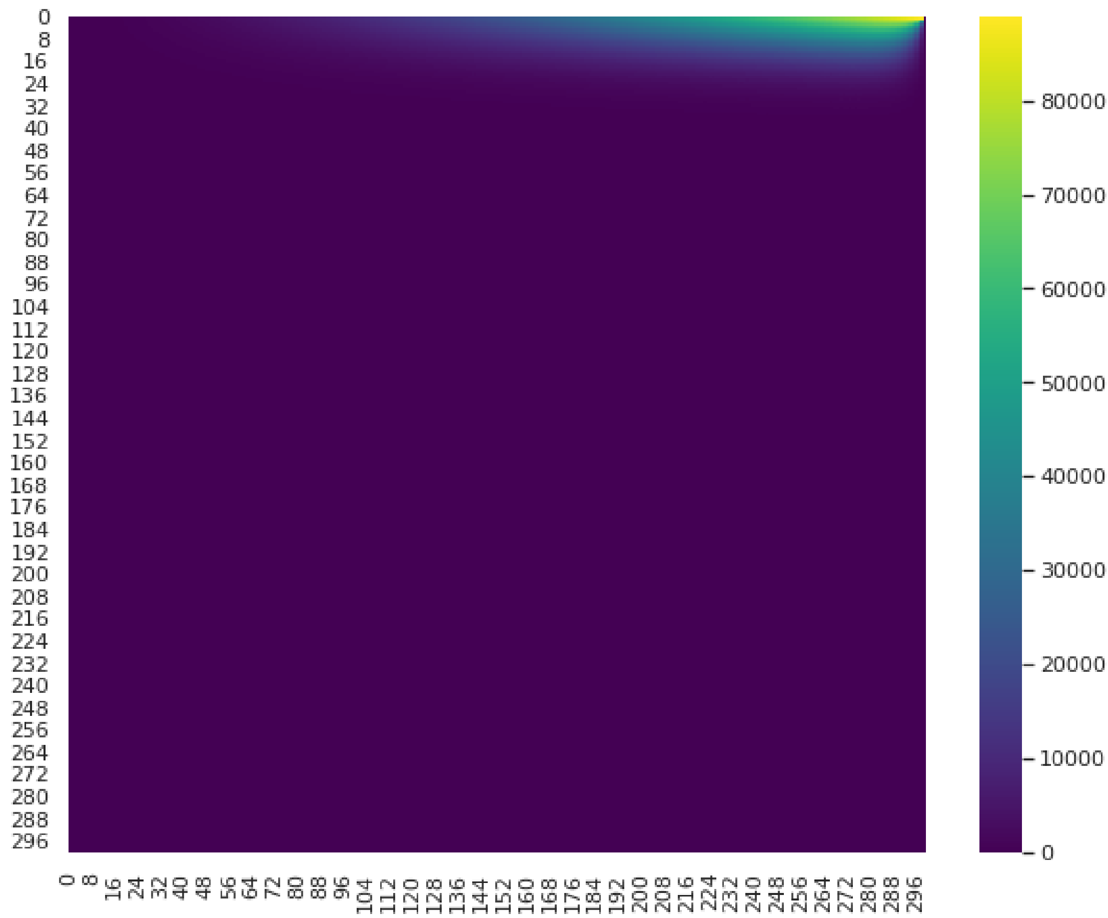

## 8 Q6:

```
[ ]: #initial random lattice
lattice = np.random.random((300,300))
```

```
[ ]: #set boundry value
lattice[:,0] = [x for x in range(300)]
lattice[0,:] = [y**2 for y in range(300)]
lattice[:,299] = 1
lattice[299,:] = 0
```

```
[ ]: #use relaxation method for solving laplace eq.
for _ in range(300**3):
    x,y= np.random.randint(1,299,2)
    lattice[x,y] = (lattice[x+1, y] + lattice[x-1, y] + lattice[x, y+1] +␣
 ↪lattice[x, y-1]) / 4
```

```
[ ]: plt.figure(figsize=(10,8))
sns.heatmap(lattice, cmap="viridis")
plt.show()
```



0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
160
168
176
184
192
200
208
216
224
232
240
248
256
264
272
280
288
296
80000
70000
60000
50000
40000
30000
20000
10000
0