Sheet 4

Ex 1

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

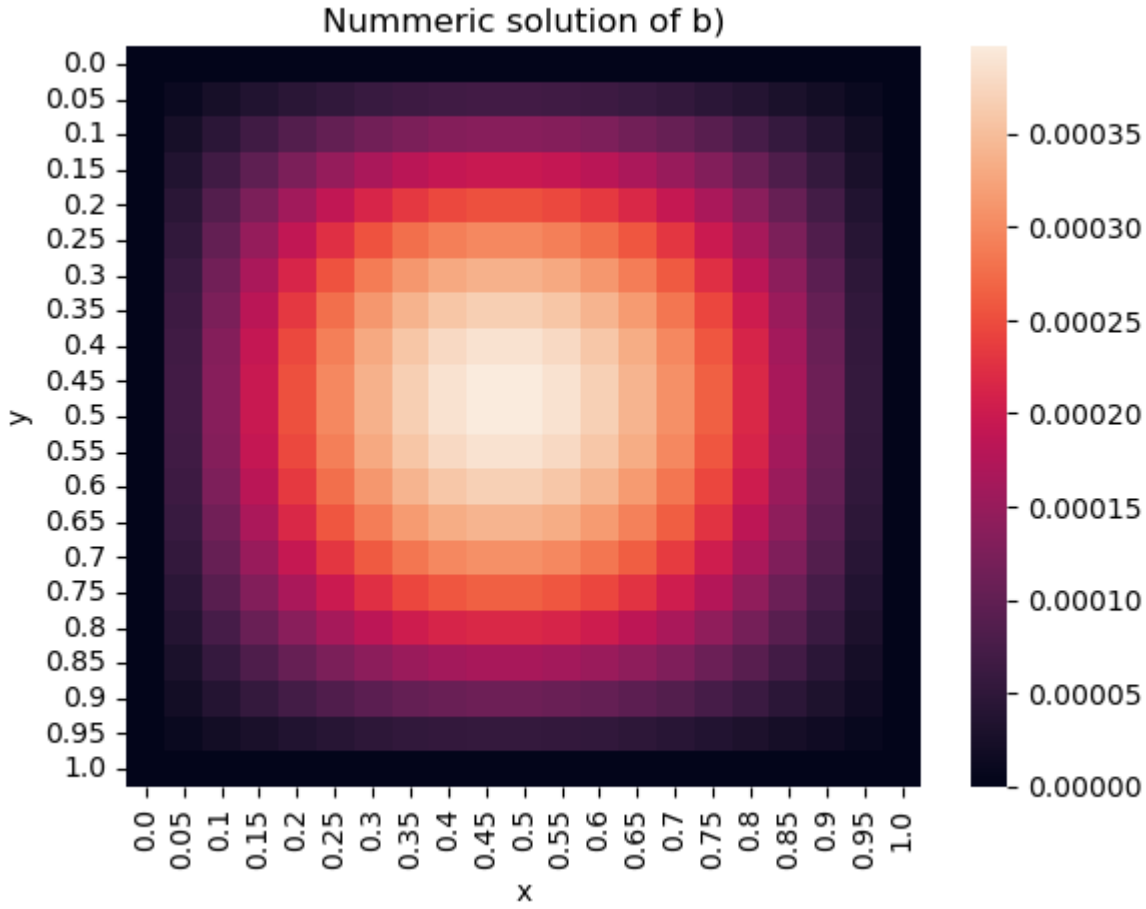
a)

In [ ]:
```
def n_grad(potential):
    # Calculate the gradient using central differences
    gradient_x = np.gradient(potential, axis=1)
    gradient_y = np.gradient(potential, axis=0)

    # Return the negative gradient components
    return -gradient_x, -gradient_y


def Gauss_Seidel(Phi, Delta, epsilon, rho, start_condions):
    J = int(1/Delta)
    L = int(1/Delta)
    #boundry conditions:
    Phi[0,:] = start_condions[0]
    Phi[J,:] = start_condions[1]
    Phi[:,0] = start_condions[2]
    Phi[:,L] = start_condions[3]

    index = np.arange(1,J) # index to go through the grid
    count = 0 # counter for number of iterations
    while True: # do while
        Phi_pre = Phi.copy()
        count += 1
        for j in index:
            for l in index:
                Phi[j,l] = 0.25 * (Phi[j+1,l] + Phi[j-1,l] + Phi[j,l+1] + Phi[j,l-1
        if ((np.abs(Phi-Phi_pre) < epsilon).all()):
            break
    print("Iterations:", count)
    E_x , E_y = n_grad(Phi) # calc E
    return Phi , E_x , E_y
```

b)

In [ ]:
```
Delta = 0.05
epsilon = 1e-5

J = int(1/Delta)
L = int(1/Delta)
x_array = np.arange(0,1+Delta,Delta)
y_array = np.arange(0,1+Delta,Delta)

Phi_b = np.ones((J+1,J+1))
rho_b = np.zeros((J+1,J+1))
start_conditions_b = np.zeros(4)

Phi_solution_b = Gauss_Seidel(Phi_b, Delta, epsilon, rho_b, start_conditions_b)
ax = sns.heatmap(Phi_solution_b[0],xticklabels= x_array.round(2), yticklabels=y_arr
ax.set_title('Nummeric solution of b)')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```

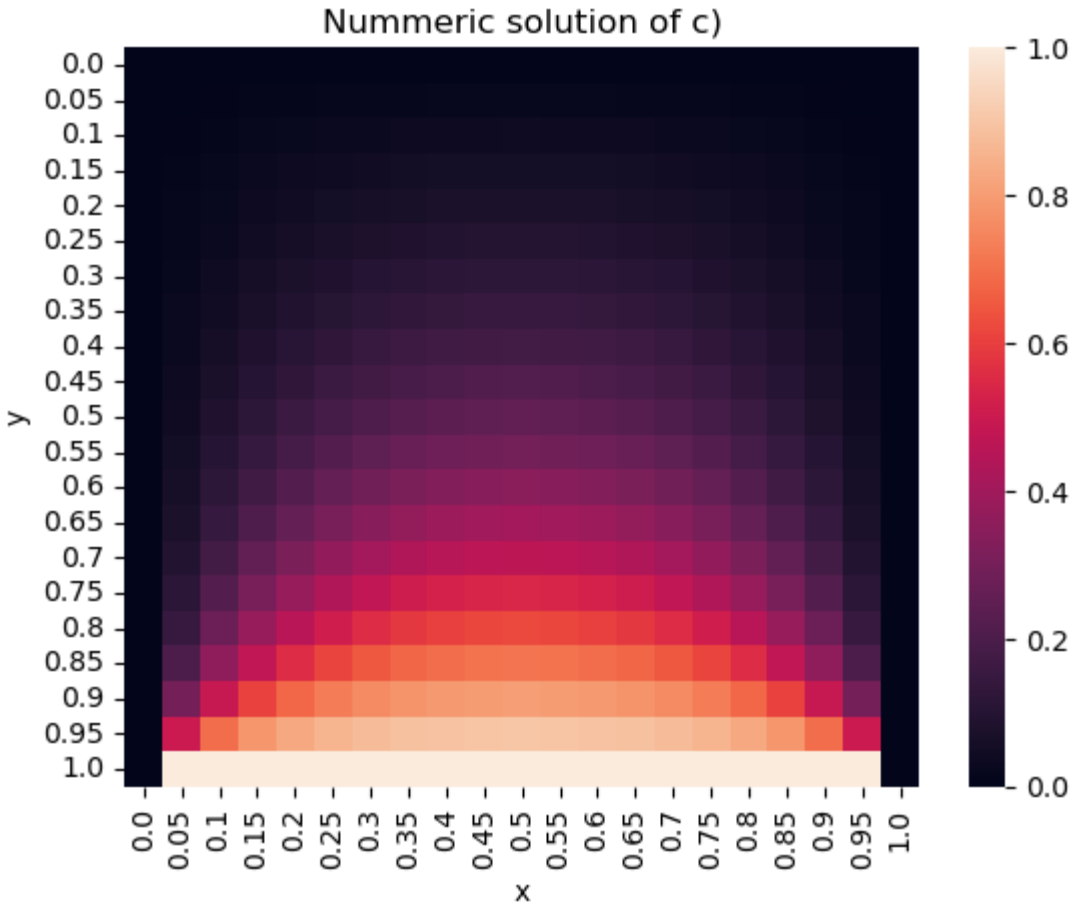
```
Iterations: 336
```

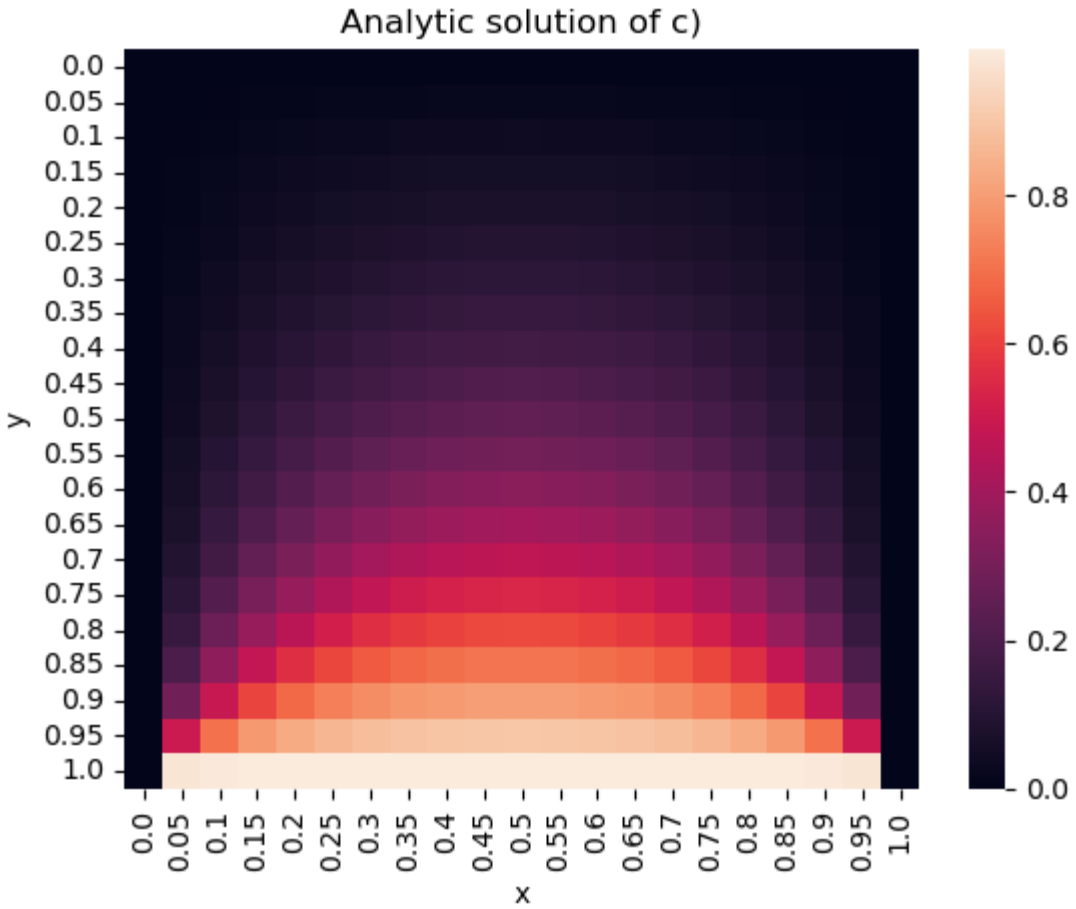


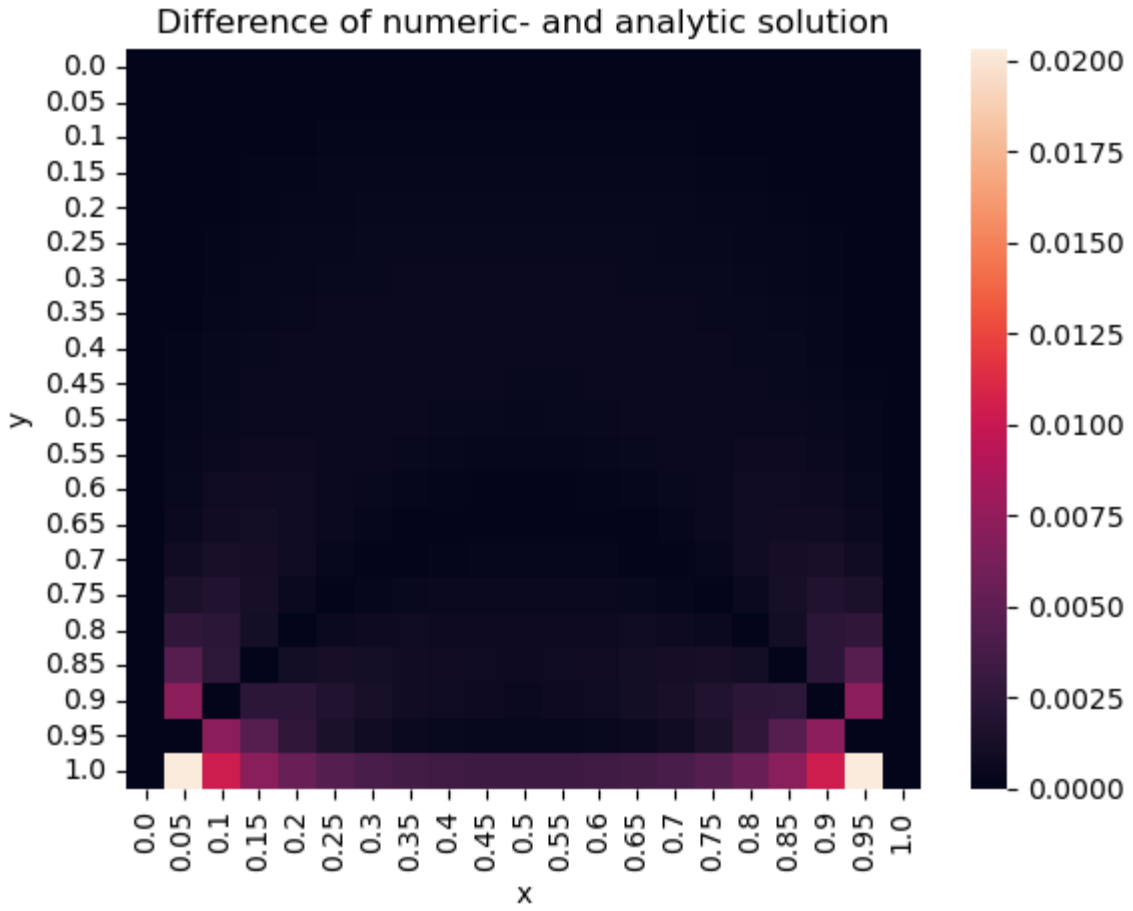
c)

In [ ]:
```
Phi_c = np.ones((J+1,J+1))
rho_c = np.zeros((J+1,J+1))
start_conditions_c = np.array([0,1,0,0])
Phi_solution_c = Gauss_Seidel(Phi_c, Delta, epsilon, rho_c, start_conditions_c)



### analytic solution:
def analytic_solution(x_values, y_values, n_iterations):
    x, y = np.meshgrid(x_values, y_values)
    n_index = np.arange(1, n_iterations)
    result = np.zeros_like(x)
    for n in n_index:
        result += 2 * (1-np.cos(n*np.pi)) / (n*np.pi*np.sinh(n*np.pi)) * np.sin(n*n
    return result

Phi_analytic = analytic_solution(x_array,y_array,200)


ax = sns.heatmap(Phi_solution_c[0],xticklabels= x_array.round(2), yticklabels=y_arr
ax.set_title('Nummeric solution of c)')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()

ax = sns.heatmap(Phi_analytic,xticklabels= x_array.round(2), yticklabels=y_array.ro
ax.set_title('Analytic solution of c)')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()

ax = sns.heatmap(np.abs(Phi_analytic-Phi_solution_c[0]),xticklabels= x_array.round(
ax.set_title('Difference of numeric- and analytic solution')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()
```

```
Iterations: 323
```


Nummeric solution of c)
y
x
0.0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1.0
0.0
0.2
0.4
0.6
0.8
1.0

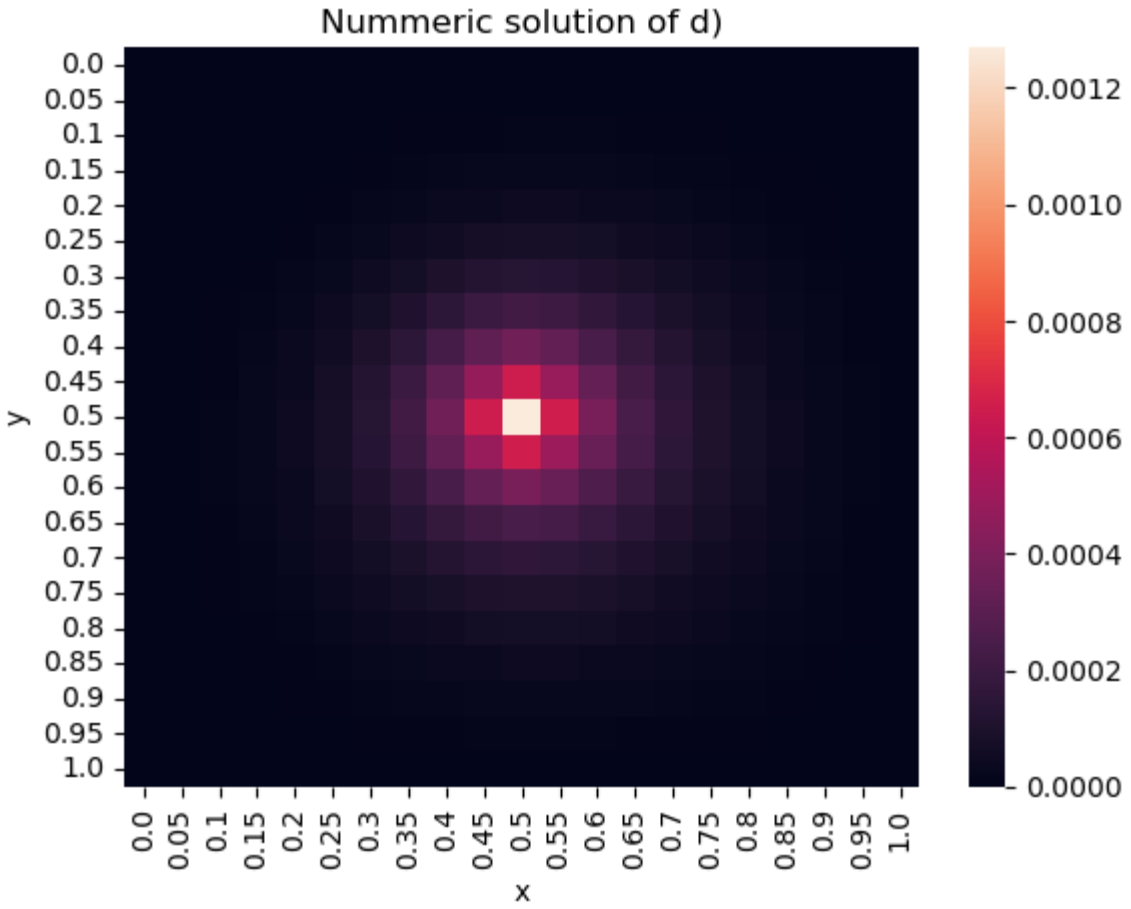

Analytic solution of c)
y
x
0.0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1.0
0.0
0.2
0.4
0.6
0.8

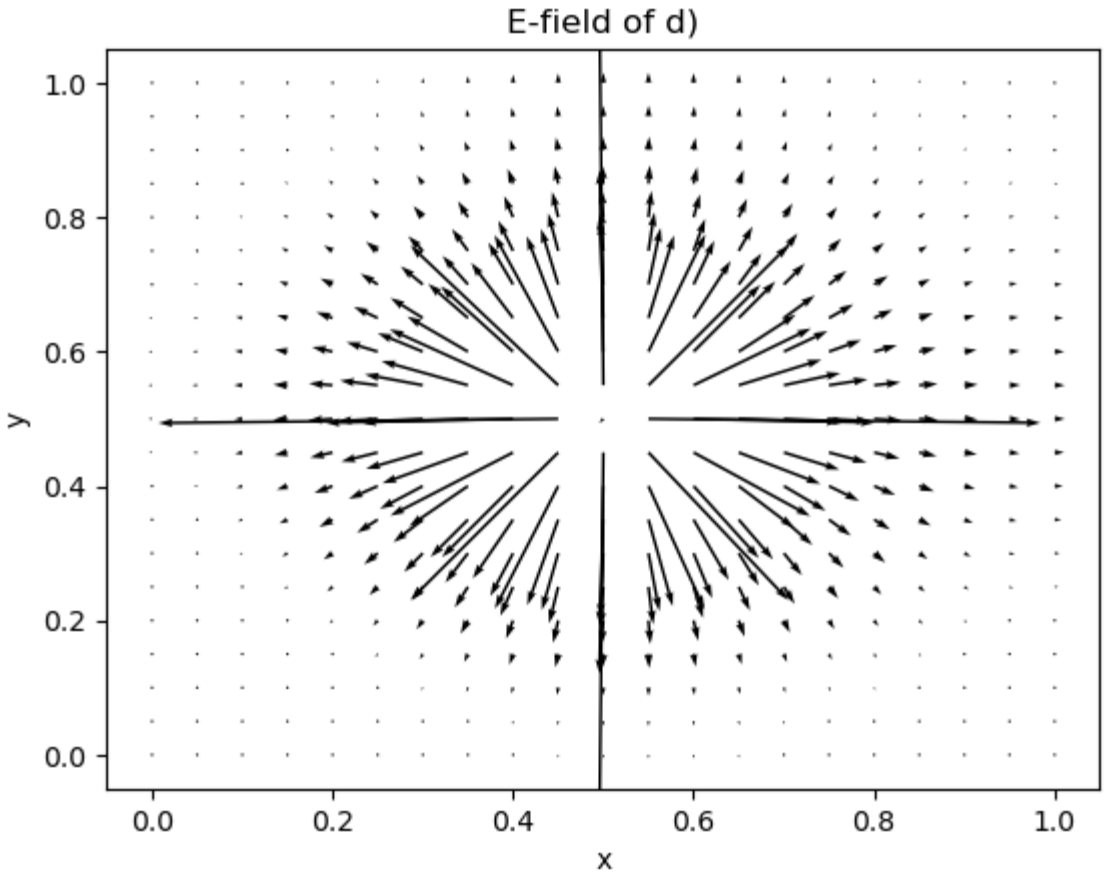


d)

In [ ]:
```
Phi_d = np.zeros((J+1,J+1))
rho_d = np.zeros((J+1,J+1))
rho_d[int(J/2),int(L/2)] = 1
start_conditions_d = np.zeros(4)
Phi_solution_d = Gauss_Seidel(Phi_d, Delta, epsilon, rho_d, start_conditions_d)


ax = sns.heatmap(Phi_solution_d[0],xticklabels= x_array.round(2), yticklabels=y_arr
ax.set_title('Nummeric solution of d)')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()


plt.quiver(x_array, y_array, Phi_solution_d[1], Phi_solution_d[2] )
plt.title('E-field of d)')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
Iterations: 21
```

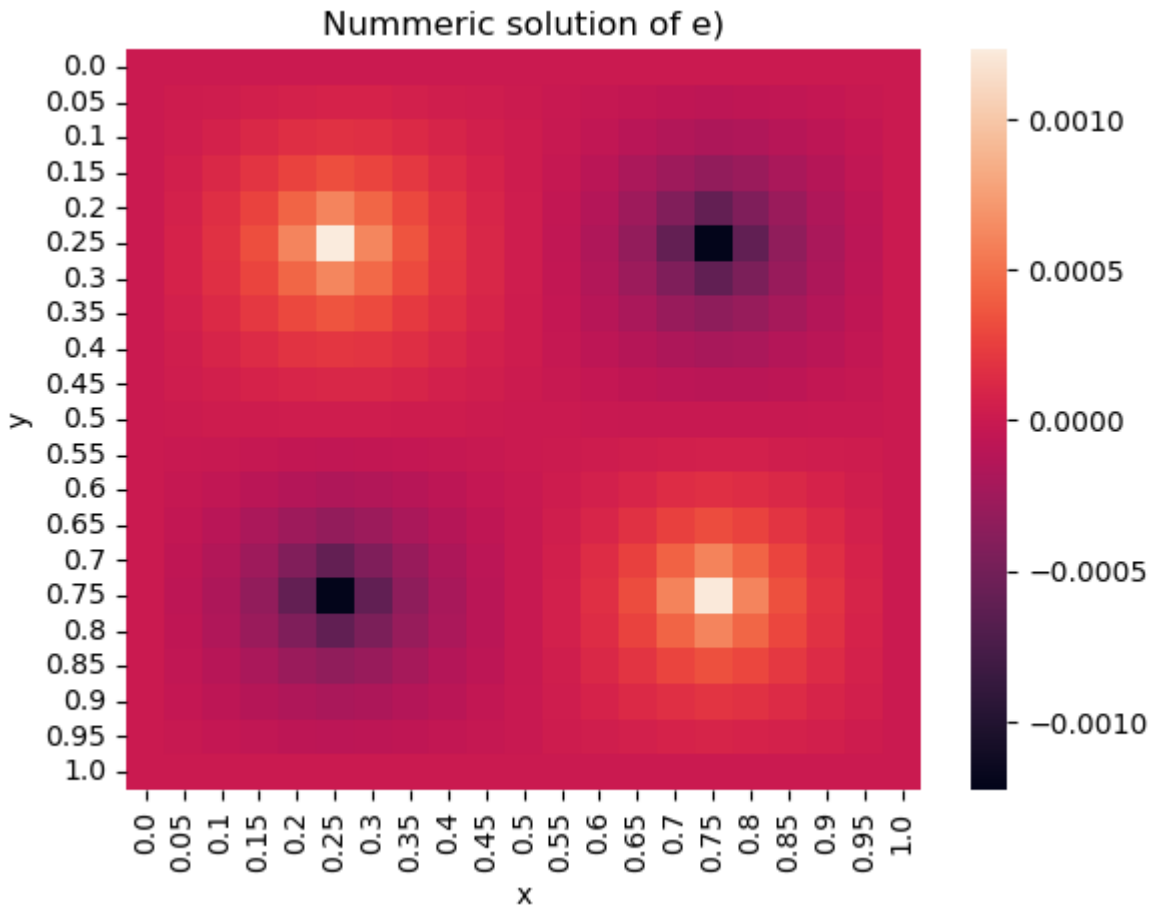

Nummeric solution of d)
y
x
0.0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5
0.55
0.6
0.65
0.7
0.75
0.8
0.85
0.9
0.95
1.0
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000

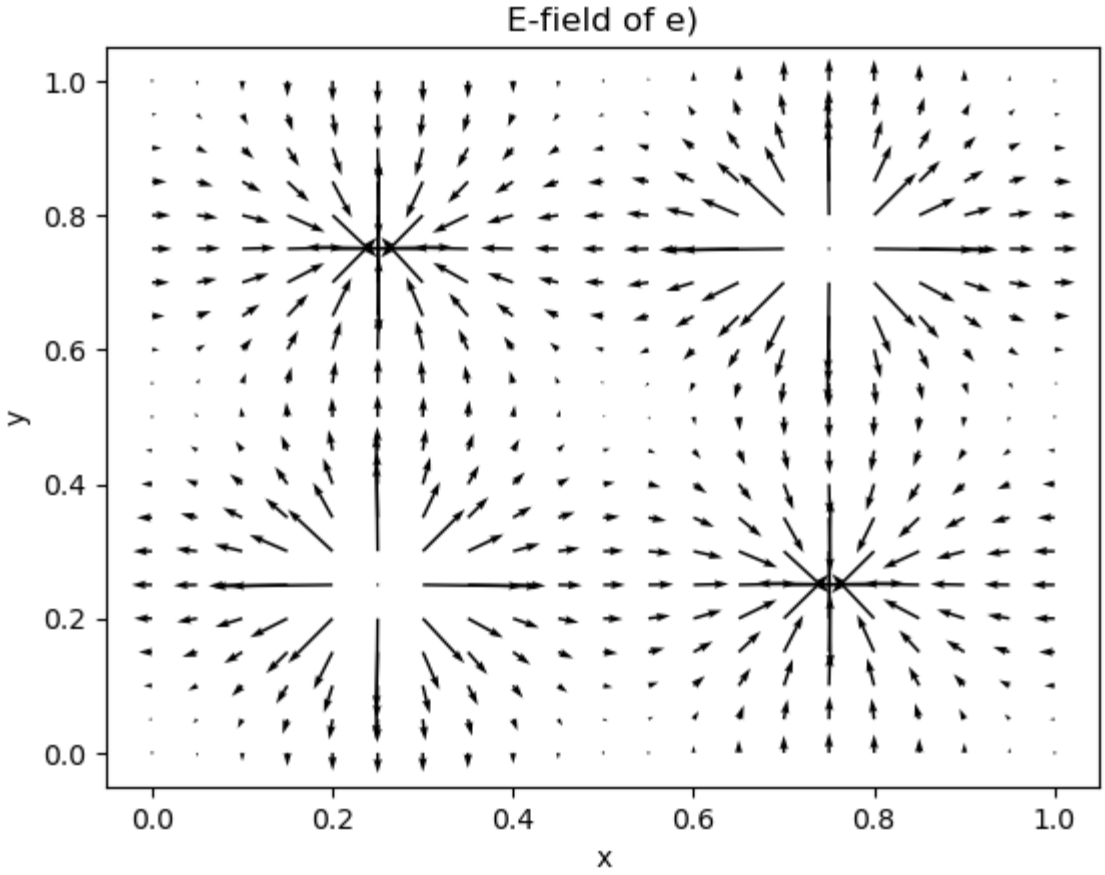

E-field of d)
y
x
0.0
0.2
0.4
0.6
0.8
1.0

e)

In [ ]:
```
Phi_e = np.zeros((J+1,J+1))
rho_e = np.zeros((J+1,J+1))
rho_e[int(J/4),int(L/4)] = 1
rho_e[int(3*J/4),int(3*L/4)] = 1
rho_e[int(J/4),int(3*L/4)] = -1
rho_e[int(3*J/4),int(L/4)] = -1
start_conditions_e = np.zeros(4)
Phi_solution_e = Gauss_Seidel(Phi_e, Delta, epsilon, rho_e, start_conditions_e)


ax = sns.heatmap(Phi_solution_e[0],xticklabels= x_array.round(2), yticklabels=y_arr
ax.set_title('Nummeric solution of e)')
ax.set_xlabel('x')
ax.set_ylabel('y')
plt.show()

plt.quiver(x_array, y_array, Phi_solution_e[1], Phi_solution_e[2] )
plt.title('E-field of e)')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

```
Iterations: 18
```




E-field of e)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x

Sheet04

Computational Physics

19. Mai 2023

1 Solution

1.1 Exercise 2

We chose to make the animations as a 1D-Colormap (matplotlib magma) that is streched to fit a 2D screen. So a bright yellow color means high value of $u(x,t)$ and a dark violet color means low value. So basically we plot the 1D Data onto a 1D strip that we extend upwards so we get a nice colormap. We chose this implementation as we thought a colormap would fit better to Diffusion than a simple graph. So even if the .mp4 files look like they show a 2D area they actualy just show a 1D strip.

a) In the first Part we should test the implementation by setting the intial condition to

$$u(x,0) = 1 = const \tag{1}$$

The result can be seen in "2a.mp4". As the boundary condition are set to be isolating no heat escapes to the sides. The whole strip stays at $u(x,t) = 1$, as expected.

b) Next we were supposed to test the stability criterion of the FTCS scheme

$$\frac{2D\Delta t}{\Delta x^2} < 1 \tag{2}$$

for the intial condition

$$u(x,0) = \delta(x - 0.5), \tag{3}$$

which is a delta peak at the middle of the strip. As the value for $\Delta x = 0.01$ and $D = 1$ was already given we just changed Δt. The value of Δt we chose as the one just above the criterion was

$$\Delta t = 6 \cdot 10^{-05},$$

which resulted in a non stable schmeme. The generated video can be seen in "2b_bad.mp4". The stable value of Δt we chose is

$$\Delta t = 4 \cdot 10^{-05},$$

which gives a stable scheme, even though it is a bit choppy. The generated video can be seen in file "2b_good.mp4".

c) In the last part we should test out different starting condition

$$u_1(x,0) = \delta(x-0.5) \tag{4}$$
$$u_2(x,0) = \theta(x-0.5) \tag{5}$$
$$u_3(x,0) = \frac{1}{9}\sum_{n=1}^{9}\delta(x-0.1n) \tag{6}$$

for which we should test out which system approaches the equilibrium state the fastest. As the timescales were rather different we made the choice to use different time discretization for each condition, even though this makes the comparision harder. The different time discretization are

$$u_1(x,0) : \Delta t = 4 \cdot 10^{-05} \tag{7}$$
$$u_2(x,0) : \Delta t = 1 \cdot 10^{-05} \tag{8}$$
$$u_3(x,0) : \Delta t = 1 \cdot 10^{-06}\,. \tag{9}$$

Another thing that makes comparing the differnt conditions difficult is that the integral over x is not 1 for all of the function. As the Heavyside function has a integral value of 50. For this reason we also normalized the Heavyside function. With all of this in mind we can compare the three conditions. The Delta ridge approached the equilibrium the fastest, which is why we chose the smalltest time discretization for this condition. The Heavyside function was the slowest of the three. The animations to every part can be seen in "2c_delta.mp4"for the $u_1(x,0)$, "2c_haevy.mp4"for $u_2(x,0)$ and "2c_weird.mp4"for $u_3(x,0)$. If we consider just the normalized condition the space integral of $u(x,t)$ stays the same at a value of 1 for all times.