Mandatory assignment 2

Håkon Silseth

UiT - Norges Arktiske Universitet

November 9, 2022

Exercise 1: Solving the Poisson equation

a)

To show that the Fourier coefficients are given by:

$$\hat{E}_m = -\mathrm{i}\frac{L}{2\pi\varepsilon_0 m}\left(\hat{\rho}_{i,m} - \hat{\rho}_{e,m}\right)$$

We can insert the expressions for E(x) and $\rho_s(x)$ into the 1D Poisson equation and solve for $\hat{E}_m$:

$$\begin{aligned}
\epsilon_0 \frac{\partial}{\partial x} E(x) &= \rho_{i,m} - \rho_{e,m} \\
\frac{\partial}{\partial x}\frac{1}{J}\sum_{m=0}^{J-1}\hat{E}_m \exp(\mathrm{i}2\pi m x/L) &= \frac{1}{J}\sum_{m=0}^{J-1}\hat{\rho}_{i,m}\exp(\mathrm{i}2\pi m x/L) - \frac{1}{J}\sum_{m=0}^{J-1}\hat{\rho}_{e,m}\exp(\mathrm{i}2\pi m x/L) \\
\frac{1}{J}\sum_{m=0}^{J-1}\hat{E}_m \frac{i2\pi m}{L}\exp(\mathrm{i}2\pi m x/L) &= \frac{1}{J}\sum_{m=0}^{J-1}\exp(\mathrm{i}2\pi m x/L)(\hat{\rho}_{i,m} - \hat{\rho}_{e,m})
\end{aligned}$$

Then for any arbitrary m we have that:

$$\begin{aligned}
\frac{1}{J}\hat{E}_m \frac{i2\pi m}{L}\exp(\mathrm{i}2\pi m x/L) &= \frac{1}{J}\exp(\mathrm{i}2\pi m x/L)(\hat{\rho}_{i,m} - \hat{\rho}_{e,m}) \\
\hat{E}_m \frac{i2\pi m}{L} &= (\hat{\rho}_{i,m} - \hat{\rho}_{e,m}) \\
\hat{E}_m &= \frac{L}{\mathrm{i}2\pi\varepsilon_0 m}\left(\hat{\rho}_{i,m} - \hat{\rho}_{e,m}\right) \\
\hat{E}_m &= -\mathrm{i}\frac{L}{2\pi\varepsilon_0 m}\left(\hat{\rho}_{i,m} - \hat{\rho}_{e,m}\right)
\end{aligned}$$

b)

(The code for this task, as well as task 1 c & d can be found in the file Task 1.py as well as in the appendix)
Since this task is just writing the code I won't write anything here, but rather have any necessary comments to the code in the next tasks or in the code itself.

c)

With $E(x) = sin(2\pi x/L) + sin(6\pi x/L)$ and $J = 2^8$ I get the following result.

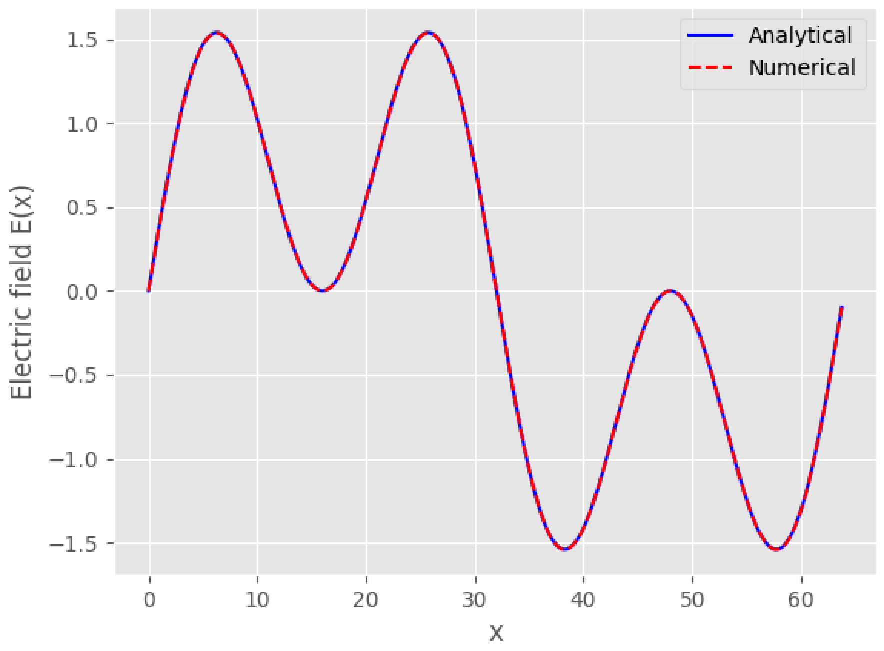


Figure 1: E(x) with $J = 2^8$

This is exactly what we expect (and hope) to get. We have good agreement between the analytical and numerical solutions. Also we don't have any distortion, which is what we expect since we have $J = 2^8 = 256$ grid points, any visible distortion shouldn't happen until we decrease the number of grid points. If we decrease n to n=6 we get:

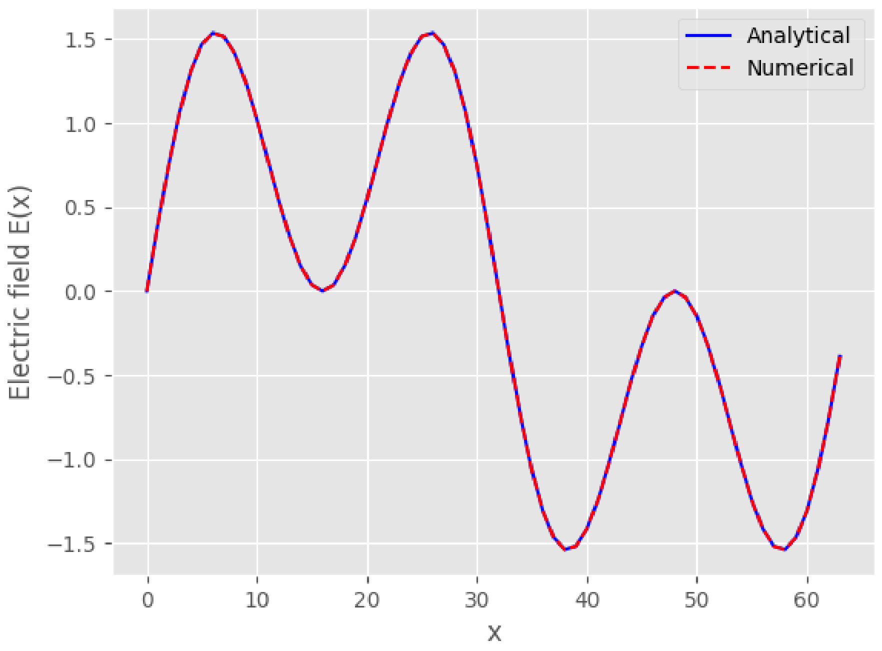
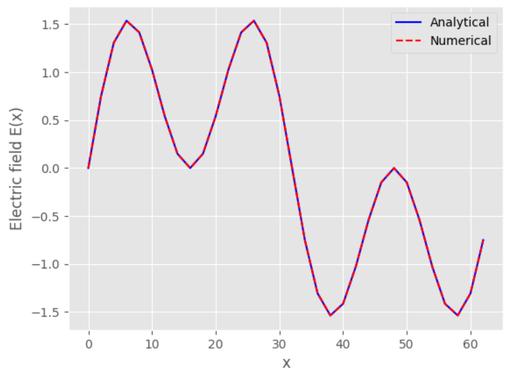
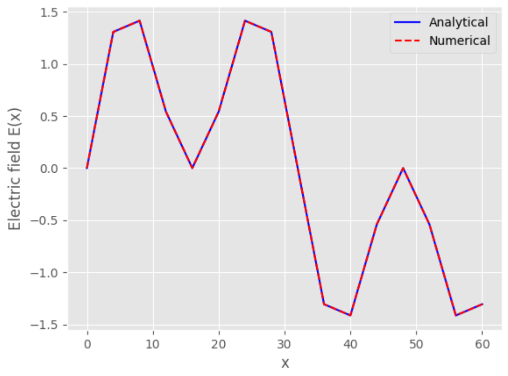


Figure 2: E(x) with $J = 2^6$

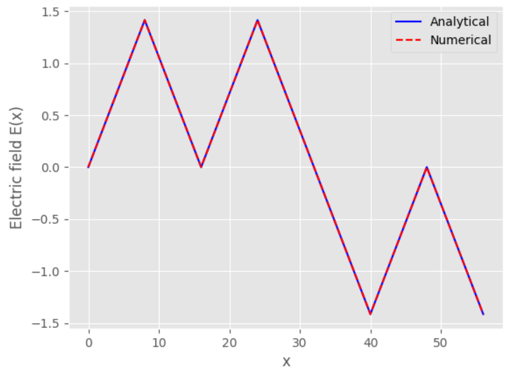
In figure 2 we can barely see some distortion in the plots for the electric field. There is still good agreement between the two solutions, but the graphs are a bit 'choppy', they are not as smooth as for $J = 2^8$. (I skipped showing $J = 2^7$ because there was no visible distortion, so it seemed unnecessary to show.)
If we continue to decrease n we get:



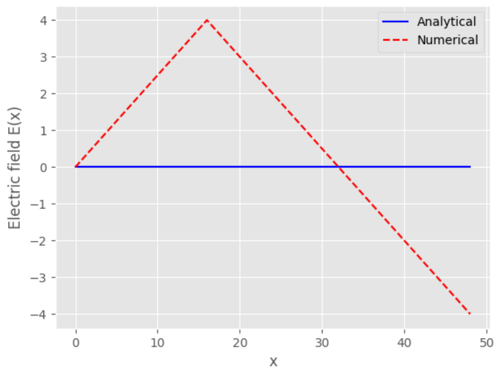
(a) $J = 2^5$



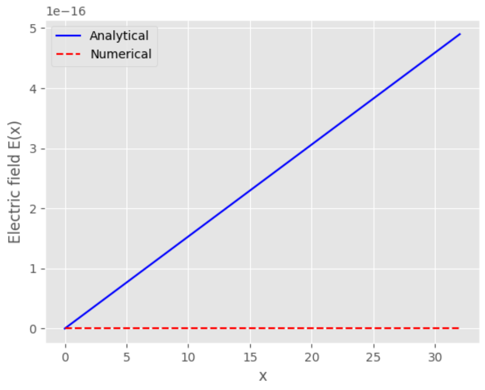
(b) $J = 2^4$



(c) $J = 2^3$



(d) $J = 2^2$



(e) $J = 2^1$

Figure 3: E(x) for decreasing number of grid points

Here we can see that it goes downhill quickly, for n=5 we see that there is more distortion, the graphs are less smooth, but we still see clearly that it is a sine function (or a superposition of sine functions), but the 'resolution' is low. The same holds true for n=4, just that it is worse again, the 'resolution' is lower. For n=3 and lower the graphs don't even resemble a superposition on sine functions, the number of grid points is just too low to give good enough resolution for them to resemble sine functions. And especially for n=2 and n=1 we can see this. Here they don't even resemble anything close to a sine function, they also don't really have any agreement, the analytical and numerical solutions look completely different. I assume this comes from the fact that when we have a low number of grid points the graphs become distorted, but the two solutions will become distorted differently, the distortion will affect the two solutions in slightly different ways, and therefore we get two completely different graphs. Just as a quick summary we can conclude here that firstly the code works, when we have a sufficient number of grid points we get a really good agreement between the numerical and analytical solution. We also see that the graphs get more and more distorted when we decrease the number of grid points, and when we decrease J to a low enough value we get so much distortion that the graphs become completely unrecognisable and they eventually don't agree because of the distortion.

d)

For this task we have $E(x) = (x - L/2)exp(-\sigma(x - L/2)^2)$ where σ has to be valued such that E(0)=E(L) is very close to zero. The plot for this task then looks like this:

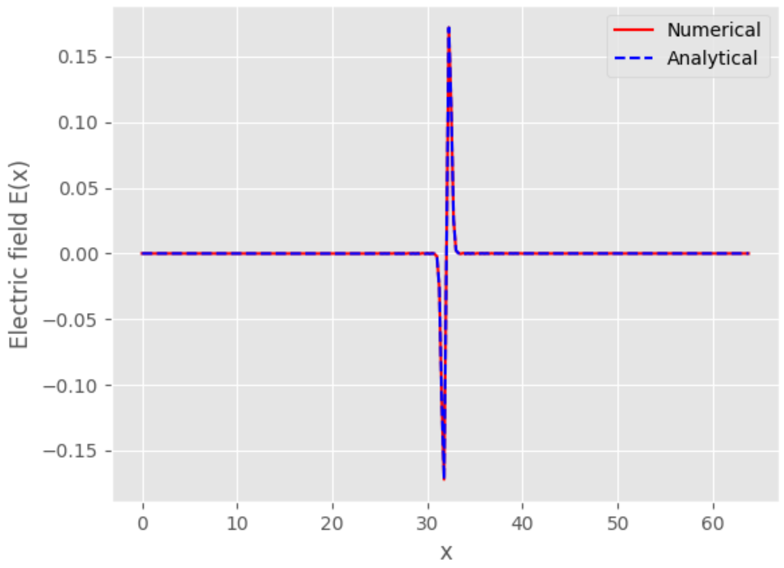
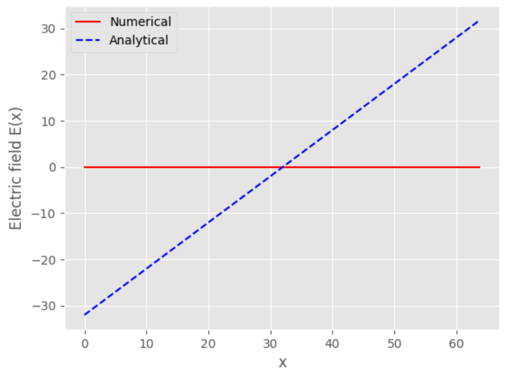
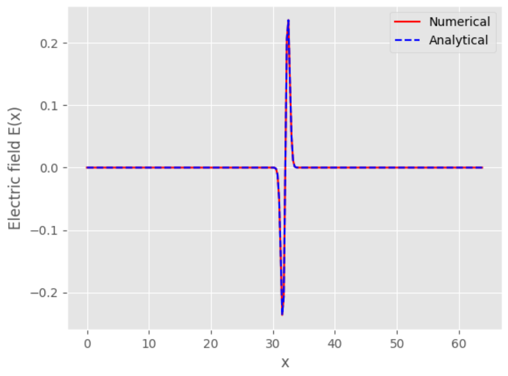


Figure 4: Numerical and analytical solution of $E(x) = (x - L/2)exp(-\sigma(x - L/2)^2)$, with $\sigma = 6$

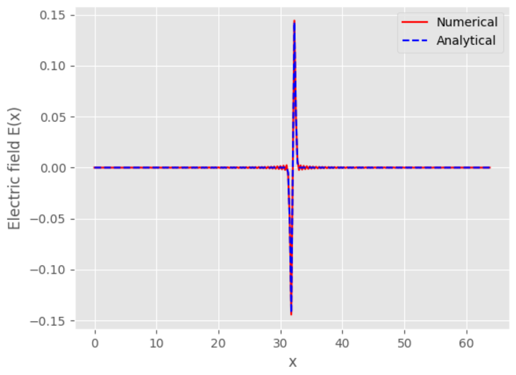
Here in figure 4 we can see that there is a very close agreement between the numerical and analytical solution. Which is what we expect, here I have chosen a value for σ such that E(0)=E(L) is very close to zero, so this is the expected result. There is not really that much to discuss here, there is a very close agreement between the two solutions, so this is just what we expect, what is more interesting is checking what happens if we choose more 'extreme' values for σ.



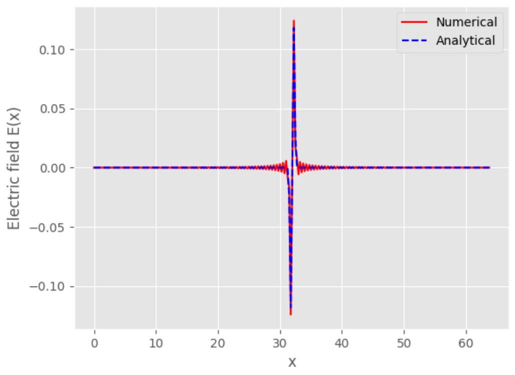
(a) $\sigma = 0$



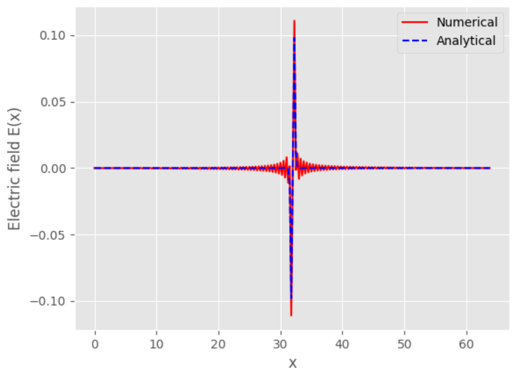
(b) $\sigma = 3$



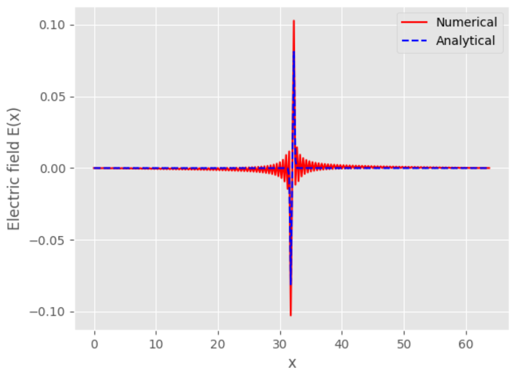
(c) $\sigma = 9$



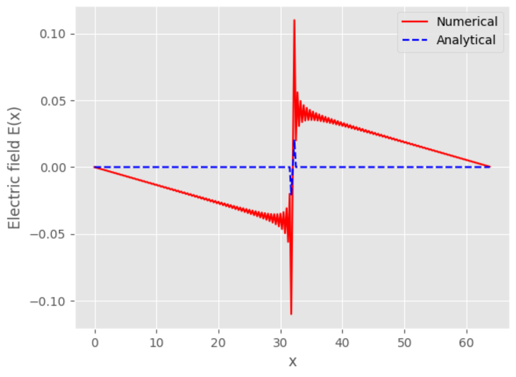
(d) $\sigma = 12$



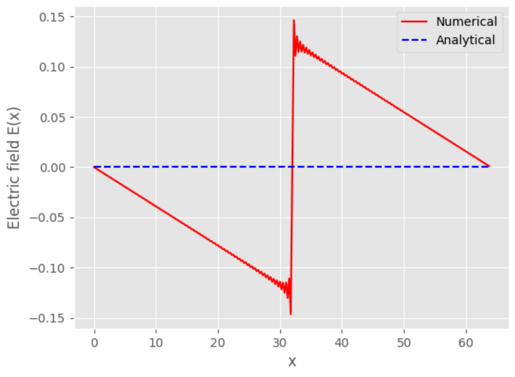
(e) $\sigma = 15$



(f) $\sigma = 18$



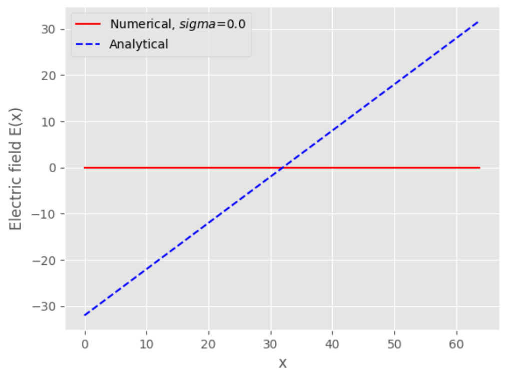
(g) $\sigma = 40$



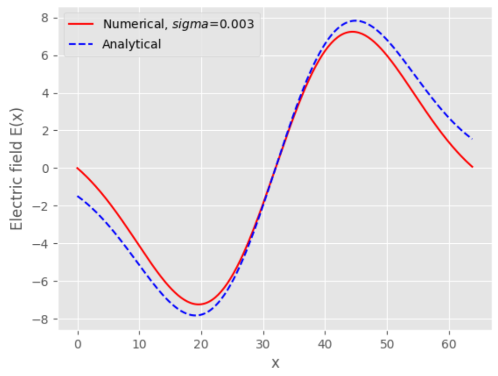
(h) $\sigma = 1000$

Figure 5: E(x) for increasing values of σ

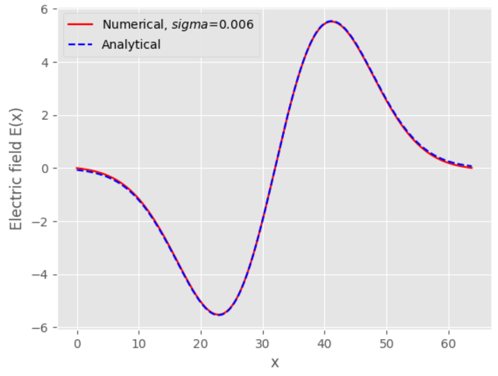
Here something interesting happens when we choose a large σ, as well as when we set it to zero. We see that for values of σ in the range 3-15 approximately we get a decent agreement between the numerical and analytical solutions. In figures 5g and 5h we can see clearly that the numerical solution and the analytical solution don't agree at all. What this means is that there must be a range for σ that gives the best agreement between the two solutions. In fact there are different reasons why we get disagreement for low values and for high values. For low values the disagreement comes from the fact that our condition $E(0) = E(L) \approx 0$ is not met, we can easily see this in figure 5a, where they are not even close to agreeing at x=0 and x=L. I wont calculate the lower range as its a bit beyond the scope of this exercise (and I've been told not to) but we can test for values of σ close to 0 and see when we get agreement between the numerical and analytical solutions.



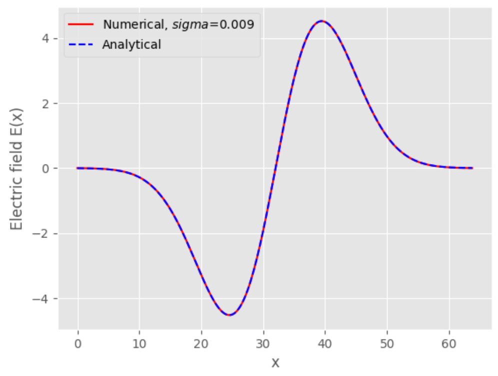
(a) $\sigma = 0$



(b) $\sigma = 0.003$



(c) $\sigma = 0.006$



(d) $\sigma = 0.009$

As we can see here we start to get really good agreement around $\sigma = 0.009$, at this point we can't see any disagreement unless we zoom in. So we can set the lower range for σ to be 0.01.
The upper limit however does not come from the fact during the calculation for the numerical solution we calculate the frequency for the Fourier coefficients, with a higher σ we will get higher frequency components, and in the code we are in a way sampling the numerical solution at a fixed rate. Meaning that for high values of σ we get aliasing. Aliasing is a concept we see in signal processing where the sample rate is not high enough compared to the signal we measure so the measurement does not match with the actual signal and we get an inaccurate measurement. In figure 5 we can see that this effect is starting to show at around $\sigma = 15$, it is a bit harder to determine a value where they begin to disagree for the upper limit as it's a bit more based on judgement, but at around 15 is where the disagreement really becomes visible. So for this to work we need the requirement that $0.01 < \sigma < 15$

Exercise 2: Particle weighting to the grid

a&b)

(The code for this task can be found in the file named Task 2.py or in the appendix.) The plot we get for this task looks like this:

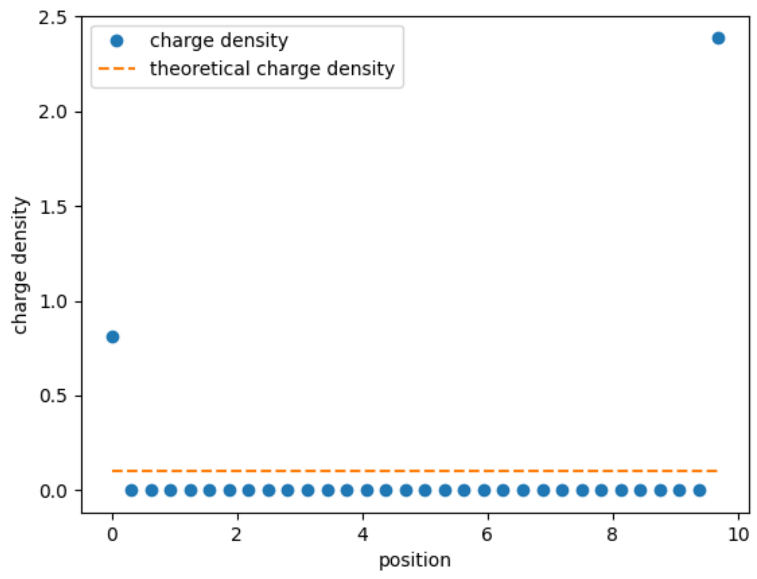
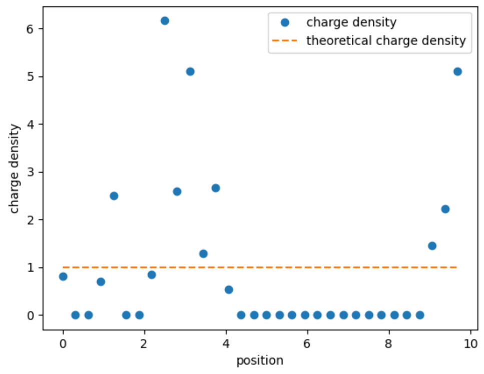
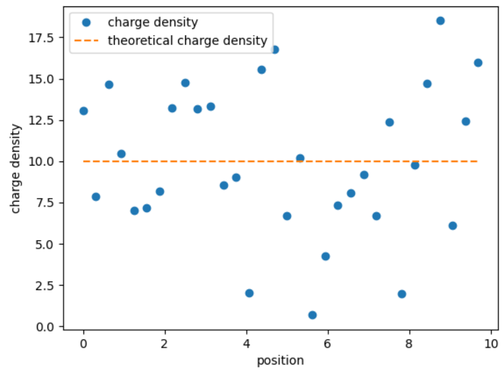


Figure 7: Theoretical and numerical charge density for N=1 particle

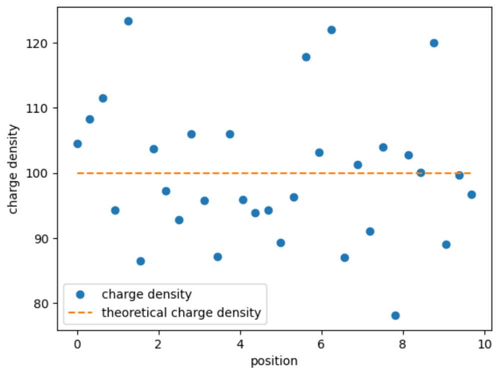
Figure 7 looks just like we expect, here we only have one particle so there is not much to go by, but we can see that the numerical charge density on the grid is only updated on two of the grid points. This is exactly what we want as we only update the charge density on the two closest grid points. We can also check to see how close of an agreement we have between the two different charge densities. We know that the agreement will be better for larger values of N, so we expect a rather large number here (larger number as larger means more disagreement). Calculating this in python we get approximately 2300%. If we increase N we get the following plots:



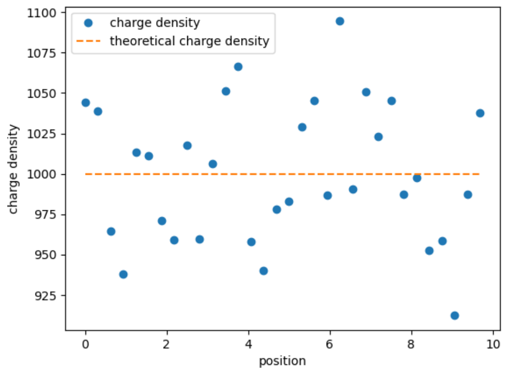
(a) N = 10



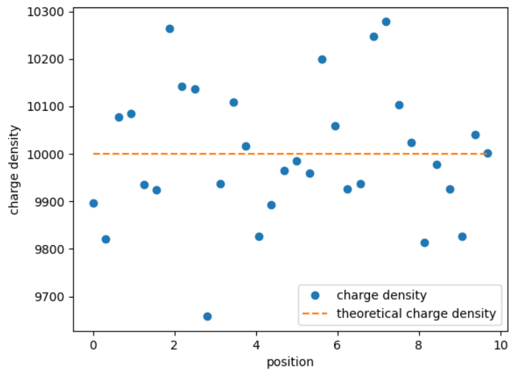
(b) N = 100



(c) N = 1000



(d) N = 10 000



(e) N = 100 000

Figure 8: Plots of theoretical and numerical charge density for increasing values of N

Here we can see the plots we get for different values of N. The exact details of the plots are not really that interesting. We can see that we get a more or less even distribution around the theoretical charge density, but the specifics are not too interesting here. What we want to know is the agreement between the numerical and theoretical charge densities. Using the little line of code given in the task I av calculated the agreement for each value of N and plotted this against N:

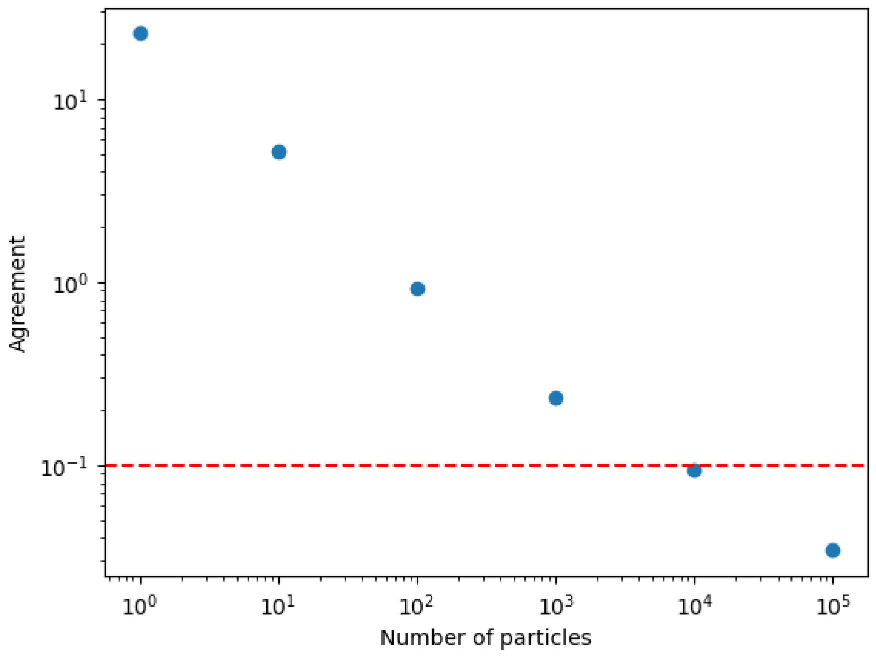


Figure 9: Agreement between the numerical and theoretical charge densities plotted against N

Here we can see that they agree more and more for increasing values of N. The red line in the plot is indicating 10% agreement. So from this plot we can conclude that we need just under 10 000 particles to get 10% agreement between the theoretical and numerical charge densities. The fact that we got the expected result for N=1 and the other values of N and the fact that the agreement-plot decreases for increasing N means that we can be confident that the code for the numerical charge density on the grid works correctly.

Appendix

Code for task 1

```python
import numpy as n
import matplotlib.pyplot as plt
import math as m
plt.style.use('ggplot')

'''defining constants'''
e0 = 8.854e-12 #Vacuum permittivity
L = 64 #Size of the grid
J = 2**8 #Points on the grid
dx = L/J #Spacing on the grid
grid_x = dx*n.arange(J) #Grid points

def task_c():
    grid_E = n.sin(2*n.pi*grid_x/L) + n.sin(6*n.pi*grid_x/L) #Electric field on the grid
    rho_i_e = e0*(2*n.pi*(3*n.cos((6*n.pi*grid_x)/L)+n.cos((2*n.pi*grid_x)/L)))/L # p_i - p_e (from Poisson's equation)

    rho_hat_i_e = n.fft.rfft(rho_i_e) #calculates fourier coefficients for p_i - p_e
    rho_hat_freq = n.fft.rfftfreq(J, dx) #calculates the frequency of the fourier coefficients

    Em_hat = n.zeros(len(rho_hat_freq), dtype=complex) #creating an array to fill with Em_hat
    Em_hat[1:] = -1.j*(1/(2*n.pi*e0*rho_hat_freq[1:]))*rho_hat_i_e[1:] #calculating the fourier coefficients for Em
        #the slicing is to ensure E_0 = 0

    Ex = n.fft.irfft(Em_hat) #calculating E(x) using inverse fourier transform on the fourier coefficients

    plt.plot(grid_x, grid_E, color = 'b', label = 'Analytical')
    plt.plot(grid_x, Ex, '--', color = 'r', label = 'Numerical')
    plt.ylabel('Electric field E(x)')
    plt.xlabel('x')
    plt.legend()
    plt.show()

def task_d(sigma):
    grid_E = (grid_x - L/2) * n.exp(-sigma*(grid_x - L/2)**2) #Electric field on the grid
    rho_i_e = e0* (n.exp(-sigma*(grid_x-L/2)**2)*(1 - 2*sigma*(grid_x-L/2)**2))#p_i - p_e (from Poisson's equation)

    rho_hat_i_e = n.fft.rfft(rho_i_e) #calculates fourier coefficients for p_i - p_e
    rho_hat_freq = n.fft.rfftfreq(J, dx) #calculates the frequency of the fourier coefficients

    Em_hat = n.zeros(len(rho_hat_freq), dtype=complex) #creating an array to fill with Em_hat
    Em_hat[1:] = -1.j*(1/(2*n.pi*e0*rho_hat_freq[1:]))*rho_hat_i_e[1:] #calculating the fourier coefficients for Em
        #the slicing is to ensure E_0 = 0

    Ex = n.fft.irfft(Em_hat) #calculating E(x) using inverse fourier transform on the fourier coefficients

    plt.plot(grid_x, Ex,color = 'r', label = 'Numerical, $sigma$={x}'.format(x=round(sigma, 4)))
    plt.plot(grid_x, grid_E, '--', color = 'b', label = 'Analytical')
    plt.ylabel('Electric field E(x)')
    plt.xlabel('x')
    plt.legend()
    plt.show()

if __name__ == '__main__':
    task_c()
    task_d(5)
```

Code for task 2

```python
import numpy as n
import math as m
import matplotlib.pyplot as plt

'''defining constant values'''
q=1
J=32
rho = n.zeros(J)

def update_field(grid, N):
    L = 10 # length of domain in units of Debye length
    seed = 1234 # seed for the random number generator
    rng = n.random.default_rng(seed=seed)
    dx=L/J
    particle_positions = rng.uniform(0, L, size=N) # creating N particles uniformly distributed

    for pos in particle_positions:
        # finding the closest grid points
        j_0 = m.floor(pos/dx)
        j_1 = j_0+1

        #Updating the charge density at the two closest grid points
        grid[j_0] += (q/dx)*((j_1*dx-pos)/dx)
        grid[j_1 % J] += (q/dx)*((pos-j_0*dx)/dx)

    # Defining the theoretical charge density
    rho_theo = n.zeros(J)
    for i in range(J):
        rho_theo[i] = q*N/L

    plt.plot(n.arange(0, L, dx), grid,'o', label = 'charge density')
    plt.plot(n.arange(0,L, dx), rho_theo, '--', label = 'theoretical charge density')
    plt.xlabel('position')
    plt.ylabel('charge density')
    plt.legend()
    plt.show()
    print(max(n.abs(grid-rho_theo)/rho_theo)) #printing the agreement between the two solutions
    return (n.abs(grid-rho_theo)/rho_theo)


if __name__ == "__main__":
    update_field(rho, 10000)
```