3_diffusion_equation

May 20, 2024

1 Diffusion equation

So far, we have always looked at ordinary differential equations, i.e. differential equations where the physical quantity we considered was depending only on one variable. In a lot of physical problems, the observable quantities depend on multiple variables like time and space. The differential equations, which govern those problems are partial differential equations. The diffusion equation is one of them. It pops up in various forms in physics, describing also heat conduction and in a slighly modified way this is corresponding to the time dependent Schrödinger equation.

[1]:
```
import numpy as np
import scipy.sparse as sparse
import scipy.sparse.linalg
import matplotlib.pyplot as plt

%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# default values for plotting
plt.rcParams.update({'font.size': 12,
                     'axes.titlesize': 18,
                     'axes.labelsize': 16,
                     'axes.labelpad': 14,
                     'lines.linewidth': 1,
                     'lines.markersize': 10,
                     'xtick.labelsize' : 16,
                     'ytick.labelsize' : 16,
                     'xtick.top' : True,
                     'xtick.direction' : 'in',
                     'ytick.right' : True,
                     'ytick.direction' : 'in',})
```

1.1 Physical Model

The diffusion equation if given by

$$\frac{\partial c(\mathbf{r},t)}{\partial t} = D\Delta c(\mathbf{r},t) \tag{1}$$

If we give the quantity c the meaning of a concentration, then the concentration will depend on space $\mathbf{r}$ and time t. D is then the mass diffusion coefficient for example.

We want to tackle the solution of partial differential equations first in one dimension in space. Thus

$$\frac{\partial c(x,t)}{\partial t} = D\frac{\partial^2 c(x,t)}{\partial x^2} \tag{2}$$

which looks a bit friendlier that before. In this equation, we have now two derivatives, one with respect to time and the other one with respect to position. According to ourt previous considerations in lecture 5, we need to discretize the concentration in space and time. We will therefore write c_i^n, where the index **n denotes the index in time** and **i** is the **index in space**.

1.1.1 Spatial derivative

So, lets have a look at the positional derivative first. As we did previously already, we can approximate our second derivative by

$$\frac{\partial^2 c(x,t)}{\partial x^2} \approx \frac{c_{i+1}^n - 2c_i^n + c_{i-1}^n}{\Delta x^2} \tag{3}$$

If we have now given the concentration in space at a time index n by $\mathbf{C} = \{c_0^n, c_1^n, c_2^n, \dots, c_5^n\}$, then the second derivative in the finite difference scheme can be written in the implicit form by

$$M = \frac{\partial^2 c}{\partial x^2} = \frac{1}{\delta x^2}\begin{bmatrix} -2 & 1 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{bmatrix}$$

where we, of course, did not include any boundary conditions so far.

In this case the diffusion equation can be written as

$$\frac{\partial c(x,t)}{\partial t} \approx DM\mathbf{C}^n \tag{4}$$

where M is the above matrix and $\mathbf{C}$ the concentration "vector". Note that the vector $\mathbf{C}$ still contains the index n, so the equation should be valid at each time index n.

1.1.2 Temporal derivative

As a second step in our solution of the diffusion equation we now also discretize the temporal derivative. We do that by a simple forward derivative here.

$$\frac{\partial c(x,t)}{\partial t} = \frac{c_i^{n+1} - c_i^n}{\delta t} \tag{5}$$

Remember that we have to write that in terms of the index n now, which is characterizing the discrete positions in time. Here, the solution is valid on each spatial position as characterized by the index i.

According to the Crank Nicolson scheme the time derivative

$$\frac{\partial c}{\partial t} = f(x) \tag{6}$$

which corresponds to a function $f(x)$, which is

$$f(x) = D\frac{\partial^2 c(x,t)}{\partial x^2} \tag{7}$$

can be appxocimated by

$$\frac{\partial c}{\partial t} \approx \frac{1}{2}\left(f^{n+1}(x) + f^{n}(x)\right) \tag{8}$$

where n is the index for the time.

1.1.3 Bringing all together

We can now bring all sides together to develop our implicit scheme.

$$\frac{\mathbf{C^{n+1}} - \mathbf{C}^n}{\delta t} = \frac{1}{2}\left(DM\mathbf{C}^{n+1} + DM\mathbf{C}^{n}\right) \tag{9}$$

We can transform the last equation to yield the value of of the concentration at the time index $n+1$, i.e.

$$\left(\mathbf{I} - \frac{\delta t}{2}DM\right)\mathbf{C}^{n+1} = \left(\mathbf{I} + \frac{\delta t}{2}DM\right)\mathbf{C}^{n} \tag{10}$$

where $\mathbf{I}$ is the identity matrix. This will correspond in our code to

$$\mathbf{A}\mathbf{C}^{n+1} = \mathbf{B}\mathbf{C}^{n} \tag{11}$$

where $\mathbf{A} = \left(\mathbf{I} - \frac{\delta t}{2}DM\right)$ and $\mathbf{B} = \left(\mathbf{I} + \frac{\delta t}{2}DM\right)$.

1.2 Numerical Solution

We are now ready to write some code. The only thing we have to define are the boundary and initial condition.

We would like to simulate the diffusion in a domain $[0, L]$. The concentration at the boundaries shall be zero at all times,i.e.

$$c(0,t) = c(L,t) = 0 \tag{12}$$

The initial condition for the concentration shall be a Gaussian centered at $x = L/2$.

$$c(x,0) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-L/2)^2}{2\sigma^2}} \tag{13}$$

with $\sigma = 0.05$ and $L = 1$.

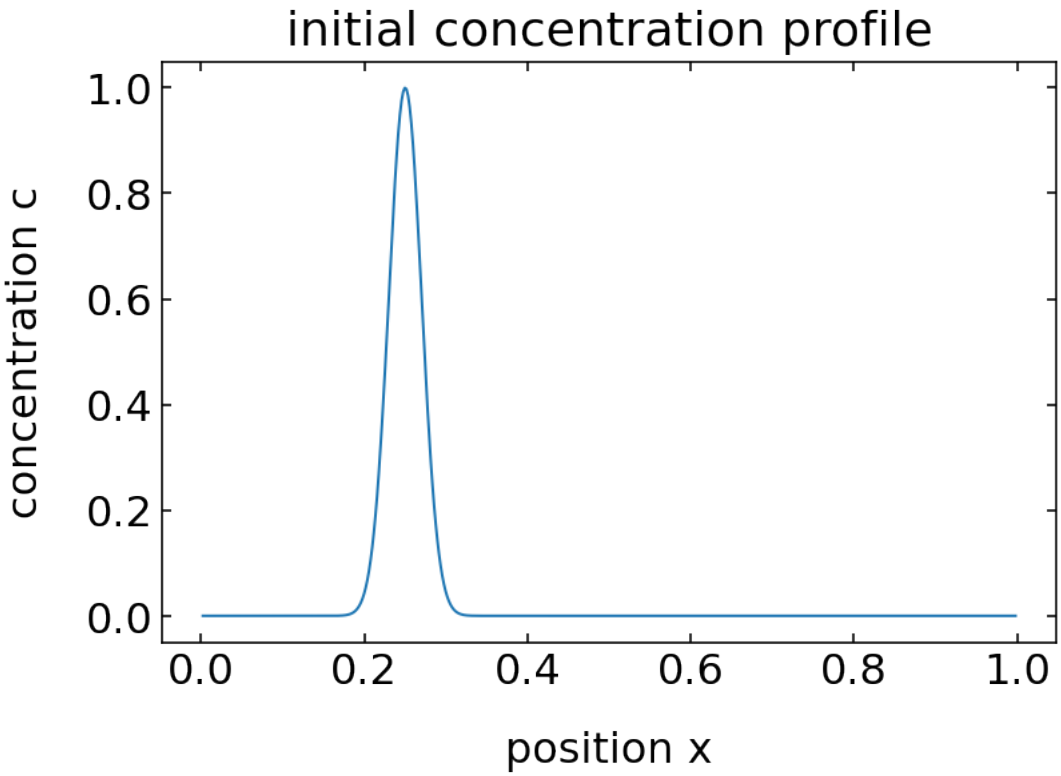
1.2.1 Setup Domain

[19]:
```
L=1.0 ## domain size

NX = 500 ## data points along position direction

dx = 1/(NX+1.0) ## position intervall
x = np.linspace(0,L,NX+2) ## position vector
x = x[1:-1] ## just skip the position at the beginning and end of the vector␣
 ↪due to the boundary conditions

T = 0.5 ## time intervall
dt = dx/4 ## time step
NT = int(T/dt) ## number of time steps

D=1 ## diffusion coefficient
```

1.2.2 Initial Conditions

[47]:
```
sigma=0.02 ## initial distribution width
c = np.transpose(np.mat(np.exp(-(x-L/4)**2/(2*sigma**2)))) ## Gaussian␣
 ↪distribution
```

[48]:
```
plt.xlabel('position x')
plt.ylabel('concentration c')
plt.title('initial concentration profile')
plt.plot(x,c)
plt.show()
```



1.2.3 Matrix Setup

[49]:
```
## Second derivative Matrix, this time created by spdiags of the scipy.sparse␣
 ↪module
## the setup is a bit different from the scipy.diags, so we learn something new␣
 ↪;-)
data = np.ones((3, NX))
data[1] = -2*data[1]
diags = [-1,0,1]
M = sparse.spdiags(data,diags,NX,NX)/(dx**2)

# Identity Matrix
I = sparse.identity(NX)
```

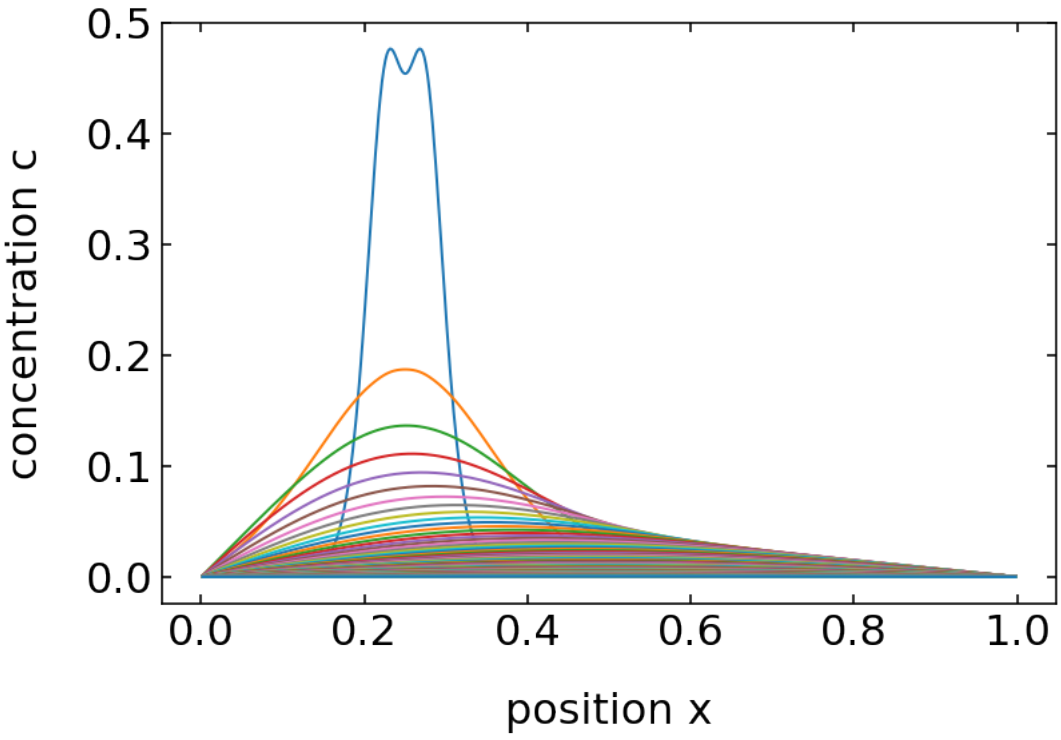
[50]:
```
data
```

[50]:
```
array([[ 1.,  1.,  1., …,  1.,  1.,  1.],
       [-2., -2., -2., …, -2., -2., -2.],
       [ 1.,  1.,  1., …,  1.,  1.,  1.]])
```

1.2.4 Solution

[51]:
```
data = [] ## store the solutions for the individual timesteps

for i in range(NT): ## loop over all timesteps

    A = (I -dt/2*D*M) ## matrix multiplied to the next timestep solution
    B = ( I + dt/2*D*M )*c ## matrix multiplied to the current timestep␣
 ↪solution, which is c
    c = np.transpose(np.mat( sparse.linalg.spsolve( A,  B ) )) ## solve the␣
 ↪system of linear equations

    data.append(c) ## store the solution
```

[52]:
```
for i in range(0,NT,10):
    plt.plot(x,data[i])

plt.xlabel('position x')
plt.ylabel('concentration c')
plt.show()
```



[46]:
```
plt.imshow(np.array(data).reshape(NT,NX),vmin=0,vmax=0.
 ↪1,cmap='gray_r',extent=(0,L,T,0))
```

```
plt.xlabel('position x')
plt.ylabel('time t')
plt.show()
```