Solving partial differential equations involving multivariable functions

When a function depends on more than one variable, it is called a multivariable function:

https://en.wikipedia.org/wiki/Multivariable_calculus

The derivatives of such functions are called partial derivatives, because we can now define the derivatives with respect to one or more of the variables:

https://en.wikipedia.org/wiki/Partial_derivative

A universal notation that specifies the derivative of a multivariable function with respect to one variable (among many) uses the curly derivative sign "∂". For instance the partial derivative of `f(x,y)` relative to `x` is written:

$$\frac{\partial f(\mathrm{x},\mathrm{y})}{\partial \mathrm{x}} = \frac{\partial f}{\partial \mathrm{x}}$$

In physics, most, if not all, systems, their evolution over time and their spatial variations, is the solution of one or several differential equations involving partial derivatives that usually mixes first and second order partial derivatives.

Two popular examples are the wave equation and the heat equation. Here we will look at the heat equation in details, and look in detail how it can be solved with `python`. More information on the heat equation can be found here:

https://en.wikipedia.org/wiki/Heat_equation

Consider a 1-dimensional rod of length L. The temperature `T(x,t)` is the temperature of the rod at position `x` and time `t`. The heat equation gives the solution `T(x,t):`

$$\frac{\partial \mathrm{T}(\mathrm{x},\mathrm{t})}{\partial \mathrm{t}} = D\frac{\partial^2 \mathrm{T}(\mathrm{x},\mathrm{t})}{\partial \mathrm{x}^2}$$

Where D is the heat diffusion constant. One clearly see that `odeint()` cannot be used "as is" since `odeint()` is designed to solve ordinary differential equations only. There are ways to still use it, but we first need to understand the basic approach on how to solve this kind of problems in general.

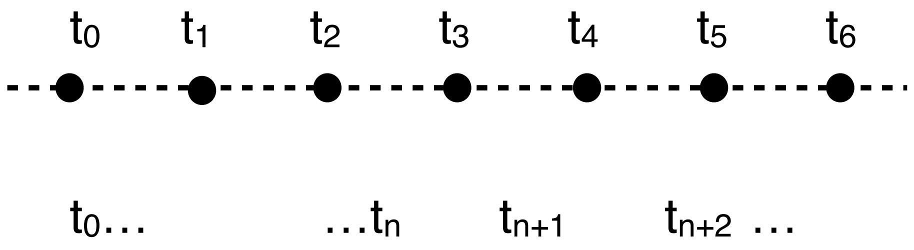
The main idea is to use the **finite difference approximation** to estimate the derivatives numerically. The basic approach is to replace derivatives with algebraic difference quotients, similar to the expressions encountered in the definition of derivative in calculus:

$$\frac{\mathrm{df(x)}}{\mathrm{dx}} = \mathrm{f'(x)} = \lim_{h\to 0} \frac{\mathrm{f(x+h)-f(x)}}{\mathrm{h}}$$

where the quantity:

$$\frac{\mathrm{f(x+h)-f(x)}}{\mathrm{h}}$$

is a finite difference approximation of `f'(x)`. For instance, consider the function position as function of time `x(t)`, this is a one dimensional function. Let's discretize the time axis:



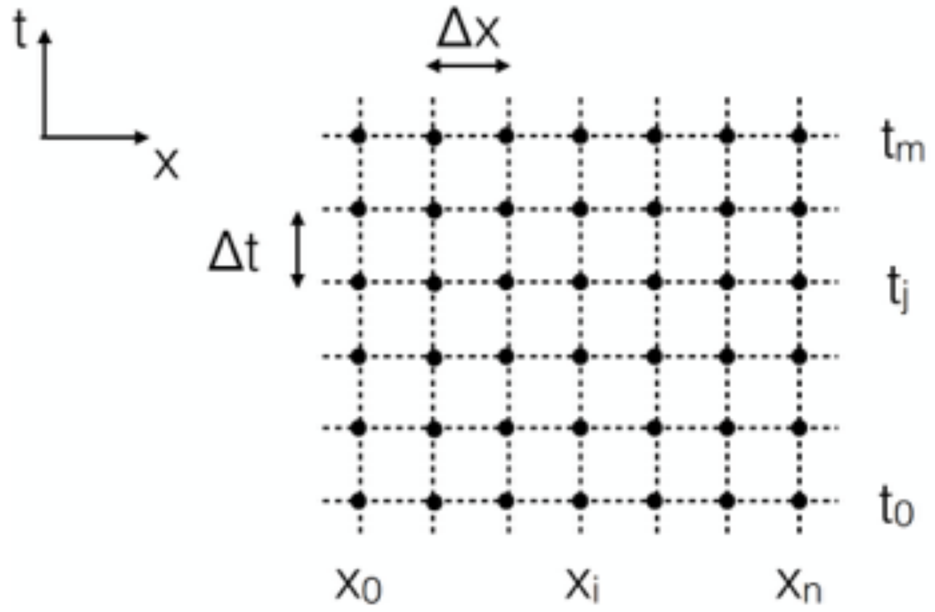
And then we have discrete particle positions $x_n = x(t_n)$. The discretized equation of motion determines $x_n \rightarrow x_{n+1} \rightarrow x_{n+2} \rightarrow$ etc.…
There are many formula that can be used to calculate numerical derivatives, and they depend on the degree of precision we want to achieve. Here we will only consider the simplest formula (therefore the poorest accuracy). For the first order derivative, following the mathematical definition above, we have:

$$\frac{\mathrm{df(x)}}{\mathrm{dx}} \simeq \frac{\mathrm{f(x+h)-f(x-h)}}{\mathrm{2h}}$$

One can derive a similar expression for the second order derivative $d^2f(x)/dx^2$:

$$\mathrm{f''(x)} \simeq \frac{\mathrm{f'(x+\frac{h}{2}) - f'(x-\frac{h}{2})}}{\mathrm{h}} = \frac{\mathrm{f(x+h)+f(x-h)-2f(x)}}{\mathrm{h^2}}$$

These expressions are said to be accurate at the o(h²) order.
When you are working with a multivariable function, the approach is to evaluate the derivatives on a grid and calculate the derivatives using finite differences. In two dimensions, for instance for the function f(x,t), you will have to discretize space ***and*** time:



In order to calculate the derivative of functions on a grid, what grid size do you have to choose? in practice, the grid *cells* should be as small as possible, but it comes with a high computing time cost. There always is a trade-off between accuracy and speed. Instead of accuracy, we would rather talk about *convergence* in this context. The grid cells should be small enough so that the derivatives converge. For this project you will only be concerned with first and second order derivatives.

The heat equation in one dimension:

Consider a rod of length L, with T(x,t) is the temperature at any location x on the rod and time t. The equation that solves T(x,t) is:

$$\frac{\partial \mathrm{T(x,t)}}{\partial \mathrm{t}} = D\frac{\partial^2 \mathrm{T(x,t)}}{\partial \mathrm{x}^2}$$

As mentioned earlier, odeint() cannot solve this multivariate differential equation, we have to use a discretized space-time grid and calculate the derivatives numerically. It is necessary to start from ***initial conditions*** T(x,t=0), and ***boundary conditions*** T(x=0,t) and T(x=L,t) so that the equation above can be solved at later time. Assuming we have calculated the solution `T(x,t)` at time t, how do we calculate the solution at later time `T(x,t+dt)`? The discretized version of the heat equation can be written as:

$$\mathrm{T}(\mathrm{x}, \mathrm{t}+\Delta \mathrm{t}) = \mathrm{T}(\mathrm{x}, \mathrm{t}) + \Delta \mathrm{t} \cdot D \left[\frac{\mathrm{T}(\mathrm{x}+\Delta \mathrm{x}, \mathrm{t}) + \mathrm{T}(\mathrm{x}-\Delta \mathrm{x}, \mathrm{t}) - 2\mathrm{T}(\mathrm{x}, \mathrm{t})}{\Delta \mathrm{x}^2} \right]$$

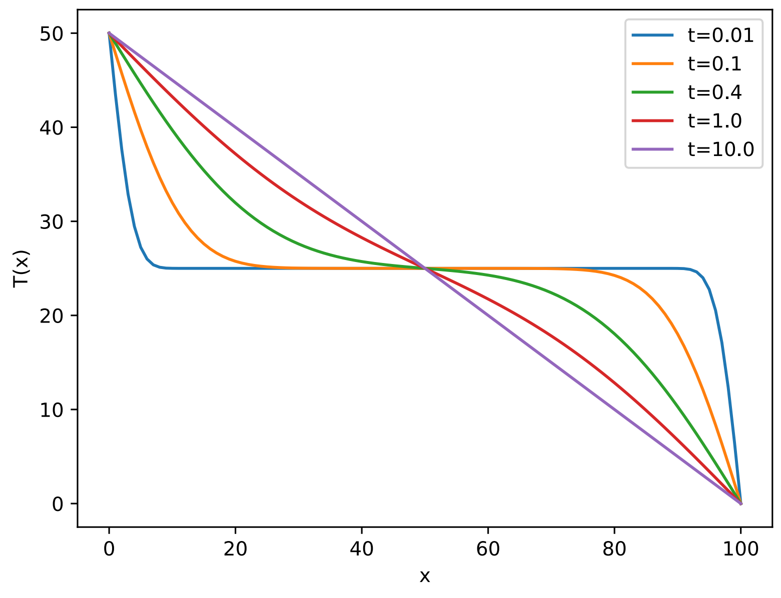
The following code is a `python` implementation of the above equation:

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

#  constants
L = 0.01
D = 4.25e-6 # units: m2 s-1
N = 100
dx = L/N
dt = 1e-3
epsilon = dt/1000

Tlo, Tmid, Thi = 0.0, 25., 50.0  #  initial temperatures in K

#  choosen times to make plots
t1,t2,t3,t4,t5 = 0.01, 0.1, 0.4, 1.0, 10.
tend = t5 + epsilon

#  create arrays
T = np.empty(N+1,float)
T[0] = Thi
T[N] = Tlo
T[1:N] = Tmid
Tp = np.empty(N+1,float)
Tp[0] = Thi
Tp[N] = Tlo

#  main loop
t = 0.0
c = dt*D/(dx**2)
while t<tend:

    #  calculate the new values of T at time t
    for i in range(1,N):
        Tp[i] = T[i] + c*(T[i+1]+T[i-1]-2*T[i])
    T,Tp = Tp,T
    print(t)
    t += dt

    #  make plots at the given times
    if abs(t-t1) < epsilon:
        plt.plot(T)
    if abs(t-t2) < epsilon:
        plt.plot(T)
    if abs(t-t3) < epsilon:
        plt.plot(T)
    if abs(t-t4) < epsilon:
        plt.plot(T)
    if abs(t-t5) < epsilon:
        plt.plot(T)

plt.xlabel("x")
plt.ylabel("T(x)")
plt.show()
```

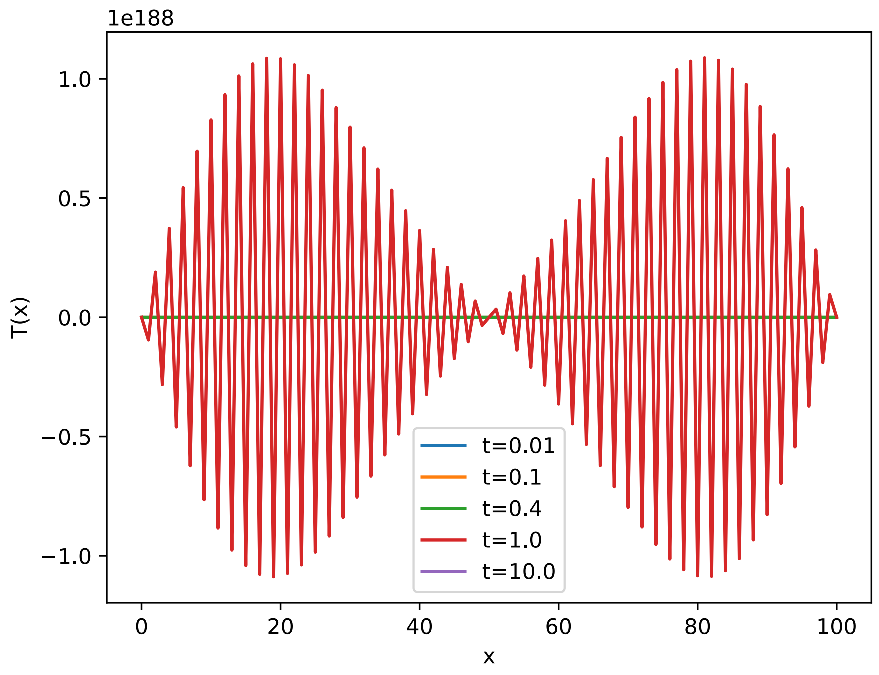
The initial and boundary conditions are defined by the values of `Thi, Tlo` and `Tmid`, and the grid cells are defined by `(dx,dt)`.

Convergence of the solution:

The analysis of the convergence of the solution is beyond the scope of this course, **although it is a very important step**. Once a numerical derivation scheme has been choosen, the granularity of the space-time grid is the main driver of convergence and final accuracy. Running the code above, we get the following temperature profile as function of time:



But changing `dt=1x10`$^{-3}$ to `dt=2x10`$^{-3}$, the solution diverges massively:



A slightly more detailed discussion of what convergence is, how to test it and how to calculate approximation errors is given there:

http://home.uni-leipzig.de/~physik/sites/mona/wp-content/uploads/sites/3/2017/04/differentiation-chap.pdf

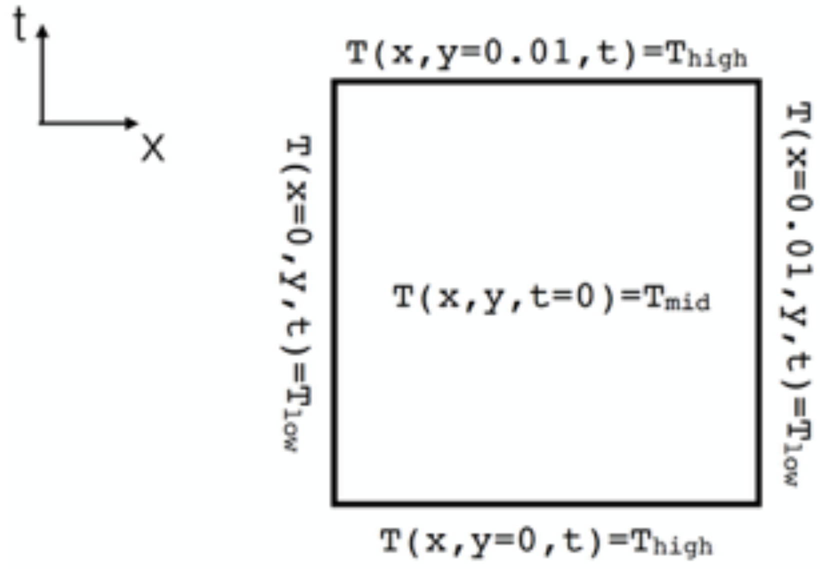
Project 5 (Due date Tuesday November 27th 12p.m.)

For this project you will consider the heat transfer in a 2-dimensional square plate which temperature obeys the usual heat equation:

$$\frac{\partial \mathrm{T}(\mathrm{x},\mathrm{y},\mathrm{t})}{\partial \mathrm{t}} = D\left[\frac{\partial^2 \mathrm{T}(\mathrm{x},\mathrm{y},\mathrm{t})}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{T}(\mathrm{x},\mathrm{y},\mathrm{t})}{\partial \mathrm{y}^2}\right]$$

Now the temperature `T(x,y,t)` is function of three variables (two of space and one of time). The goal of this project is to write a program which solves this equation numerically. This project is an introduction to the non-trivial problem of calculating derivatives with computers, you will only get a thin exposure to what this whole field is about. This area of research is a highly complex one and it is behind many physics situations that we would like to be able to describe with computers: A famous example is the solution to the solution to the Navier-Stokes equation which can describe fluid turbulence and therefore is used to predict weather.

Objectives: For this project you will write a code **`heat2d.py`** which will calculate the solution `T(x,y,t)` for the following system: consider a thermally conducting square plate with sides `L=0.01 m`. The initial and boundary conditions are given by the following:



You will take `T`$_{high}$`=400 K`, `T`$_{mid}$`=250 K` and `T`$_{low}$`=200 K`. Your code should generate automatically two movies: 1- one movie will be called **`heat2d_converged.mp4`** will show the 2-dimensional evolution of the temperature on the plate from `t=0 sec` to `t=10 sec` when it has converged, and 2- the other movie will be called **`heat2d_diverged.mp4`** will show how the solution will badly diverged when the value of `dt` is too big. In order to do the latter, you will have to search, by hand, what value of `dt` will do the job! The values of `dt` will therefore be hardcoded in your code.

Tips: You can start from the 1-dimensional template discussed above. The execution time will greatly benefit from the use of the `np.roll()` method to calculate the derivatives without loops.