Partial Differential Equations I: Introduction & Finite Difference Method

Sam Sinayoko

Numerical Methods 4

Contents

```python
import numpy as np
import matplotlib
from IPython.html.widgets import interact
from IPython.display import Image
try:
    %matplotlib inline
except:
    # not in notebook
    pass

LECTURE = False
if LECTURE:
    size = 20
    matplotlib.rcParams['figure.figsize'] = (10, 6)
    matplotlib.rcParams['axes.labelsize'] = size
    matplotlib.rcParams['axes.titlesize'] = size
    matplotlib.rcParams['xtick.labelsize'] = size * 0.6
    matplotlib.rcParams['ytick.labelsize'] = size * 0.6
import matplotlib.pyplot as plt
```

1 Learning Outcomes

After studying this notebook you should be able to

- Describe the difference between an Ordinary Differential Equation (ODE) and a Partial Differential Equation (PDE);
- Give some examples of PDEs;
- Derive the first central difference approximations for the $f'(x)$ and $f''(x)$;

- Use the finite difference method to solve the Laplace equation in 1D;

2 Introduction

Along with the Ordinary Differential Equations (ODEs) we have considered previously, partial differential equations (PDEs) are another way in which we model problems in science and engineering. Their solution occupies a large amount of the time consumed on large scale high performance computers for tasks such as computational fluid dynamics, computational mechanics and computational electromagnetics which are used for modelling systems as diverse in size scale and complexity as climate and environmental modelling, the Universe, cars, planes, trains, ships and optical devices.

Specialist courses exist which will study an individual application domain and its equations in huge detail and often highly tuned and specialised computational methods (and indeed even whole computer architectures and subsystems) have been developed for a particular system. Our purpose in this section is to introduce some of the key concepts and methods that underpin this universe of modelling possibilities.

2.1 Difference between ODEs and PDEs

- ODEs are differential equations containing **a single independent variable** so all the derivatives are ordinary dervatives. For example, given a scalar function $x \mapsto f(x)$, the first and second derivatives are f' or f''.
- PDEs are differential equations containing **two or more independent variables**, so the derivatives are partial derivatives. For example, given a function of two variables $(x, y) \mapsto f(x, y)$, the partial derivatives with respect to x and y are $\frac{\partial f}{\partial x}$ and $\frac{\partial f}{\partial y}$.

2.2 PDE example I: Laplace equation

The Laplace equation is a second orderd PDE appearing for example in Fluid Mechanics (potential flows) and Electromagnetics (electromagnetic field in a charge free region):

$$\text{General:} \qquad \nabla^2 \phi = 0, \tag{1}$$

$$\text{2D:} \qquad \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = 0, \tag{2}$$

where ϕ represents, for example, the velocity potential.

2.3 PDE example II: Heat equation

The heat equation is a second order PDE describing how temperature T diffuses through a medium of thermal diffusivity α:

$$\text{General:} \qquad \frac{\partial T}{\partial t} = \alpha \nabla^2 T \tag{3}$$

$$\text{2D:} \qquad \frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) \tag{4}$$

2.4 PDE example III: Wave equation

The wave equation describes how the sound pressure p propagates at speed c_0 through a medium (at rest), and plays a role in acoustics, fluid mechanics, but also cosmology and quantum mechanics:

$$\text{General:} \qquad \frac{1}{c_0^2} \frac{\partial^2 p}{\partial t^2} + \nabla^2 p = 0 \tag{5}$$

$$\text{2D:} \qquad \frac{1}{c_0^2} \frac{\partial^2 p}{\partial t^2} + \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2} = 0 \tag{6}$$

2.5 General form of a scalar PDE

In a scalar PDE of order n for a function of two variable $(x, y) \mapsto f(x, y)$ will be of the form

$$F\left(f, \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial^2 f}{\partial x^2}, \frac{\partial^2 f}{\partial y^2}, \frac{\partial^2 f}{\partial x \partial y}, \cdots, \frac{\partial^{i+j} f}{\partial x^i \partial y^j}, \cdots, \frac{\partial^n f}{\partial x^n}, \frac{\partial^n f}{\partial y^n} \right) = 0, \tag{7}$$

where F is a function that defines the PDE and $1 \leq i \leq n$, $1 \leq j \leq n$ and $i + j \leq n$.

2.6 Solution strategy for the Heat equation

Let's consider the heat equation from example II in 1D:

$$\text{1D Heat Equation:} \qquad \frac{\partial f}{\partial t} = \alpha \left(\frac{\partial^2 f}{\partial x^2} \right). \tag{8}$$

Given the solution at a time step t_0 the challenge is to estimate the solution at a later time $t_0 + h$. This is almost the same problem we had for ODEs, where we learned how to solve equations of the form

$$g'(t) = F(t, g(t)), \tag{9}$$

using Euler's method or Runge-Kutta methods. Thus, if we could rewrite the right hand side of the heat equation as a function of time and f (rather than $\frac{\partial f}{\partial t}$) we'd be able to apply all the techniques we have learned in our study of ODEs to advance our solution in time.

We therefore seek to approximate the second order partial derivative as

$$\frac{\partial f}{\partial t} \approx F(f, x, t), \tag{10}$$

so that we can write

$$\frac{\partial f}{\partial t} \approx \alpha F(f, x, t), \tag{11}$$

which is an ODE we know how to solve.

The key challenge is therefore to find good approximations for the spatial derivative with respect to x, in terms of only $f(x,t)$, x and t. The *finite difference method* is a simple but powerful method for doing exactly this. The finite difference method will be introduced in the next section. It will then be used to solve the Laplace equation, which corresponds to the steady state solution of the Heat Equation. Solving the Heat Equation itself will be left as an exercise to the reader.

3 The finite difference method

3.1 First central difference for first $f'(x)$

Our aim with these methods is to replace the differential operator with an approximation which averages over nearby points and by using a mesh of such points we derive a set of simultaneous equations to solve. The trick is to apply Taylor's expansions at points near $f(x)$, such as $f(x-h)$ and $f(x+h)$, and to combine these expansions to obtain an approximation for the desired derivative.

Consider the following two Taylor expansions of a function f(x) around x at a (small) distance, h (see e.g. [1])

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + O(h^5), \tag{12}$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + O(h^5). \tag{13}$$

$$\tag{14}$$

If we substract these two equations we get

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{3}f'''(x) + O(h^5). \tag{15}$$

Rearranging gives

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{3}f'''(x) + O(h^4), \tag{16}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2). \tag{17}$$

This is known as the *first central difference approximation* for the *first* derivative of $x \mapsto f(x)$. It is accurate to order 2. It was derived by using Taylor's expansion for f at two points around x: $x+h$ and $x-h$. These points are symmetric with respect to x. It is also possible to use *staggered* finite difference approximations such as the *first forward difference* for the first derivative, that expresses $f'(x)$ in terms of \$f(x), $f(x+h)$ and $f(x+2h)$.

3.1.1 Self study: Exercise 1

Derive the first forward difference in terms of $f(x)$, $f(x+h)$ and $f(x+2h)$.

3.2 First central difference for $f''(x)$

It is also possible to obtain finite difference approximations for higher order derivatives. For example, if we add the two Taylor expansions for $f(x+h)$ and $f(x-h)$ instead of subtracting them we get

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4). \tag{18}$$

Rearranging yields

$$f''(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} + O(h^2). \tag{19}$$

This is known as the *first central difference approximation* for the *second* derivative of $f(x)$.

3.3 General central finite difference for $f^{(n)}(x)$

It is possible to derive these for higher order derivatives to various order of accuracy. A general central difference approximation for $f^{(n)}(x)$ can be written in the form

$$f^{(n)}(x) = \sum_{i=-m}^{m} c_i f(x+ih), \tag{20}$$

where $(c_i)_{-m\leq i\leq m}$ define the *coefficients* of the finite difference. The points $(x+ih)_{-m\leq i\leq m}$ are called the *nodes* of the finite difference. The *nodes* and *coefficients* define a unique finite difference *stencil* . The length of the stencil is therefore $2m+1$. Typically, the length of the stencil must be longer than n, and higher orders of accuracy can be obtained by using longer stencils.

Given a set of nodes $-m \leq i \leq m$, the stencil coefficients can be derived by applying Taylor's approximation to $f(x+ih)$ to the desired order of accuracy, for each value of i. This yields a system of $2m+1$ equations with $2m+1$ unkowns (the stencil coefficients (c_i)), which can be solved to obtain the set of coefficients that approximates a particular derivative to the desired order of accuracy. See for example [2] for listings of these "stencils", or Abramowitz and Stegun 25.3.21-25.3.31.

4 Example: Laplace equation in 1D

A simple time dependent heat equation reduces to the Laplace equation in steady state (once the system has come to equilibrium). To show how the finite difference method can be used to solve PDEs we first start with a 1D problem, so that our PDE reduces to an ODE:

$$T''(x) = 0, \quad \text{with boundary conditions for} \quad T(x). \tag{21}$$

We will first consider simple fixed boundary conditions for the problem.

Consider the problem which represents an inifinitely thin rod of length $L = 1$ m, with temperature held constant at $0\,\text{deg}$ at one end and $100\,\text{deg}$ at the other:

$$T''(x) = 0, \qquad T(0) = 0\,\text{deg}, \qquad T(L) = 100\,\text{deg}\,. \tag{22}$$

Discretizing gives us:

$$T''(x) = \frac{T(x-h) - 2T(x) + T(x+h)}{h^2}, \quad T(0) = 0\,\text{deg}, \quad T(L) = 100\,\text{deg} \tag{23}$$

at each point on the rod, where the mesh separation is h.

4.1 One unknown mesh point

$$\frac{T_{-1} - 2T_0 + T_1}{h^2} = 0, \tag{24}$$

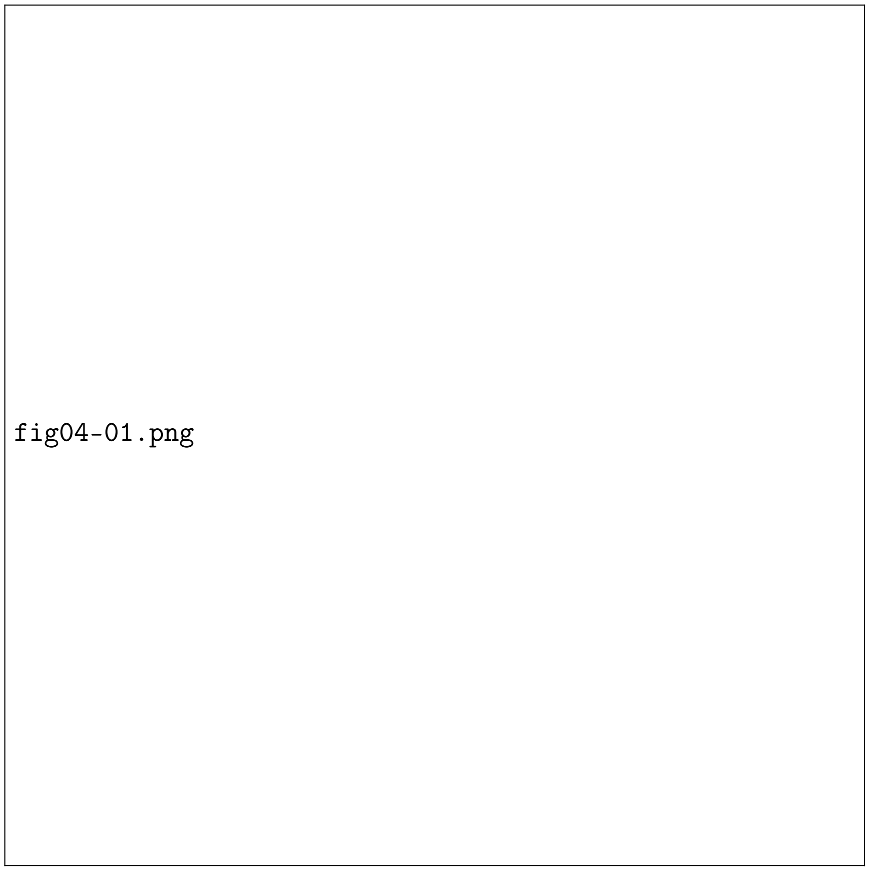

Figure 1: 1D mesh with one unknown mesh point T_0.

where $h = L/2$. Since $T_{-1} = 0\,\mathrm{deg}$ and $T_1 = 100\,\mathrm{deg}$, we have

$$T_0 = \frac{T_{-1} + T_1}{2} = 50\,\mathrm{deg} \tag{25}$$

This is what we expect because when the system is in equilibrium, the temperature should vary linearly along the rod from $0\,\mathrm{deg}$ at $x = 0$ to $100\,\mathrm{deg}$ at $x = L$.

4.2 Two unknown mesh points

If we discretize the Laplace equation at the two unknown mesh points T_0 and T_1 using the first central approximation for the second derivative T'', we

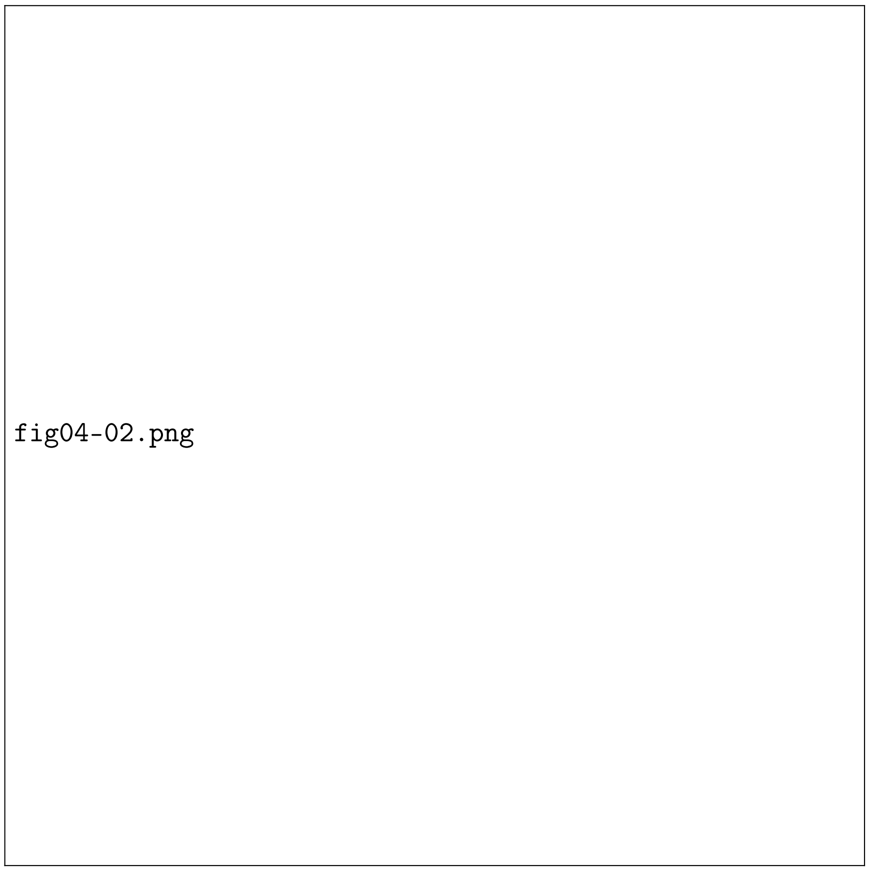

Figure 2: 1D mesh with two unknown mesh point T_0 and T_1.

get

$$\begin{cases} \dfrac{T_{-1} - 2T_0 + T_1}{h^2} = 0 \\ \dfrac{T_0 - 2T_1 + T_2}{h^2} = 0, \end{cases} \tag{26}$$

where $h = L/3$. If we express our unkowns T_0 and T_1 in terms of our boundary conditions $T_{-1} = 0\,\mathrm{deg}$ and $T_2 = 100\,\mathrm{deg}$, we get

$$\begin{cases} -2T_0 + T_1 = -T_{-1} \\ \quad T_0 - 2T_1 = -T_2, \end{cases} \tag{27}$$

or in matrix form

$$\begin{pmatrix} -2 & 1 \\ 1 & -2 \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \end{pmatrix} = \begin{pmatrix} -T_{-1} \\ -T_2 \end{pmatrix}. \tag{28}$$

Inverting the matrix yields

$$\begin{pmatrix} T_0 \\ T_1 \end{pmatrix} = \frac{1}{3}\left(-2 {-1} {-1} {-2}\right) \begin{pmatrix} -T_{-1} \\ -T_2 \end{pmatrix} = \frac{1}{3}\begin{pmatrix} 2T_{-1} + T_2 \\ T_{-1} + 2T_2 \end{pmatrix} = \begin{pmatrix} 33.333333\,\mathrm{deg} \\ 66.666666\,\mathrm{deg} \end{pmatrix} \tag{29}$$

Again, we see that the temperature varies linearly through the rod.

4.3 Three unknown mesh points

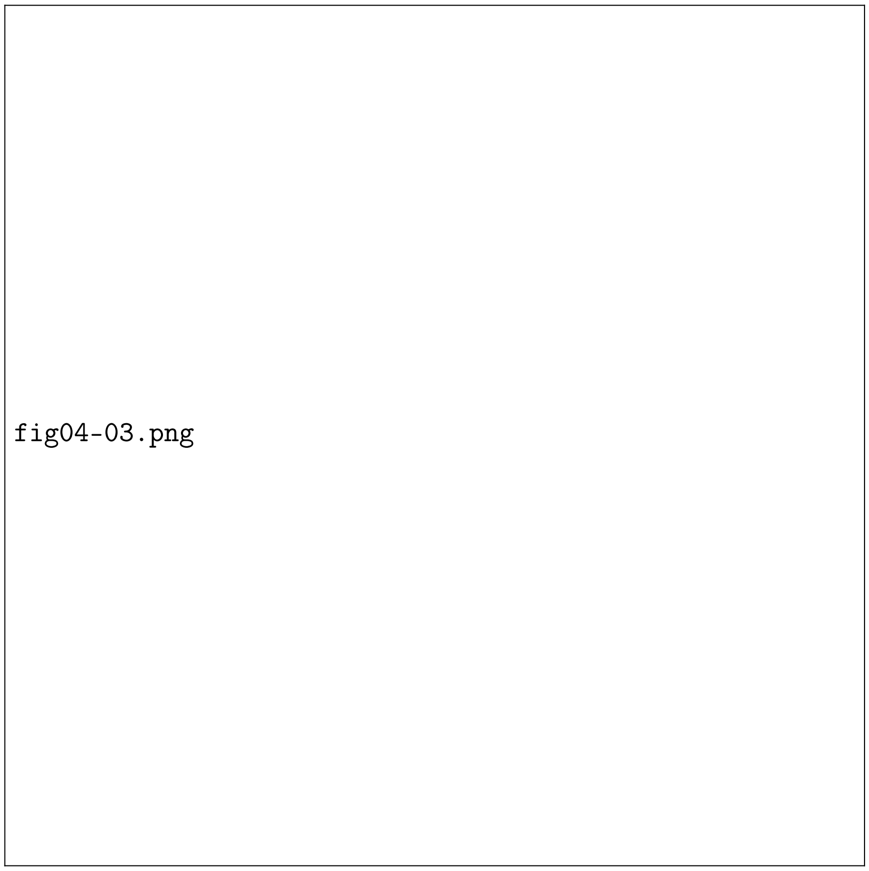

Figure 3: 1D mesh with three unknown mesh point T_0, T_1 and T_2.

For 3 unkown mesh points T_0, T_1, and T_2, we get the system of equations

$$\begin{cases} T_{-1} - 2T_0 + \ \ T_1 \qquad\qquad\quad = 0 \\ \qquad\quad T_0 - 2T_1 + \ \ T_2 \qquad = 0 \\ \qquad\qquad\qquad T_1 - 2T_2 + T_3 = 0, \end{cases} \tag{30}$$

or, in matrix form,

$$\begin{pmatrix} -2 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \\ T_2 \end{pmatrix} = \begin{pmatrix} -T_{-1} \\ 0 \\ -T_3 \end{pmatrix} . \tag{31}$$

Solving this (with SymPy or SciPy, see below) gives

$$\begin{pmatrix} T_0 \\ T_1 \\ T_2 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} -3 & -2 & -1 \\ -2 & -1 & -2 \\ -1 & -2 & -3 \end{pmatrix} \begin{pmatrix} -T_{-1} \\ 0 \\ -T_3 \end{pmatrix} = \frac{1}{4} \begin{pmatrix} 3T_{-1} + T_3 \\ 2T_{-1} + 2T_3 \\ -T_{-1} + 3T_3 \end{pmatrix} = \begin{pmatrix} 25 \\ 50 \\ 75 \end{pmatrix} \text{deg} . \tag{32}$$

4.4 Solving with SymPy, SciPy or Iterative Methods

Note that to solve the above linear system of equations we can use the iterative methods presented in the first section. This is left as an exercise. For such a short system, we can use SymPy to inverse the system symbolically. Another option is to use SciPy's `linalg` module for a numerical solution.

- SymPy solution

```
from sympy import *
init_printing()
# boundary temperatures
Tm1 = 0
T3  = 100
b = Matrix([[-Tm1], [0], [-T3]])
# matrix to be inverted
A = [[-2, 1, 0],
     [1, -2, 1],
     [0, 1, -2]]

Ainv = Matrix(A)**(-1)
Ainv, b
```

```
# Get the temperatures
Ainv * b
```

- SciPy solution

```
import scipy.linalg as linalg
linalg.solve(A, b)
```

4.5 General pattern for $N + 2$ mesh points

For $N + 2$ mesh points, using the first central difference for the second derivative to discretize the 1D Laplace equation, with boundary conditions $T_{-1} = 0\,\text{deg}$ and $T_{N+1} = 100\,\text{deg}$, yields:

$$\begin{pmatrix} -2 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & \cdots & 0 & 1 & -2 \end{pmatrix} \begin{pmatrix} T_0 \\ T_1 \\ \vdots \\ \vdots \\ T_{N-1} \\ T_N \end{pmatrix} = \begin{pmatrix} -T_{-1} \\ 0 \\ \vdots \\ \vdots \\ 0 \\ -T_{N+1} \end{pmatrix}, \tag{33}$$

A similar system of equation (with periodic boundary conditions) was solved using the Gauss-Seidel method in Example 1 of Notebook 1. Whilst these solutions are straightforward our aim was to show the process of going from the differential equation to a set of linear equations that we can solve. This discretization process is the key step to solving an ODE or a PDE with a computer. The finite difference method is a powerful yet simple discretization method that we will focus on in this course.

4.6 Self study: Exercise 2

1. Use the Gauss-Seidel method to solve the above system of equations for $N = 100$.

2. Use a generic solver such as `scipy.linalg.solve` to validate your solution.

3. *Solve the full heat equation in 1D with $T_0 = 0$ and $T_L = 100$, starting from a uniform initial temperature of $T = 0$. Use the Runge-Kutta method (RK4) to solve the 1D heat equation with $\alpha = 1.172 \cdot 10^{-5}\text{m}^2\text{s}^{-1}$ and calculate the temperature distribution for $0 \leq t \leq 25000\text{s}$, using 10000 time steps and $N = 100$ unkown mesh points $0 < x_i < L$ (where $L = 1$ m).

5 Conclusions

- ODEs are differential equations involving 1 independent variable, PDEs involve two or more independent variables;

- Classical examples of PDEs are the Laplace equation, the Heat equation and the Wave equation.
- The first central difference approximations for $f'(x)$ and $f''(x)$ can be derinved by combining the Taylor expansions of $f(x+h)$ and $f(x-h)$ to order 3.
- The finite difference method can be used to discretize the 1D Laplace equation

$$T''(x) = 0, \tag{34}$$

with boundary conditions $T(0) = T_0$ and $T(L) = T_L$, which yields a linear system of equation of the form

$$AX = B, \tag{35}$$

where A is a matrix describing the Laplace operator, X is the vector of unknowns describing the temperature at the inner mesh points, and B is a vector containing the boundary conditions. This system can be solved analytically or symbolically (for small number of unkowns), or using direct or iterative methods for large number of unkowns.

In the next lecture, we will solve our first PDE: the Laplace equation in two dimensions:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = 0. \tag{36}$$

6 References

1. Jaan Kiusalaas, Numerical Methods in Engineering with Python, 2010.
2. Zwillinger, D. (1998). *Handbook of differential equations (Vol. 1)*. Gulf Professional Publishing.{google books}

7 Solution to exercise 2.3

7.1 Input parameters

Define the spatial grid and the number of time steps.

```
#%debug
# Spatial grid
N = 100                             # number of unkowns
X = np.linspace(0, 1, N+2)          # grid array
# Remove the edge points from the grid
X = X[1:-1]
H = X[1] - X[0]                     # grid step

# Time grid
TIME_MIN = 0
TIME_MAX = 25000
NTIME = 10000

# Discretised Laplace operator for inner points [1:-1]
A = (np.diag(-2 * np.ones(N)) +
     np.diag(np.ones(N - 1), 1) +
     np.diag(np.ones(N - 1), -1) )
pprint(A)
```

```
[[-2.  1.  0. ...,  0.  0.  0.]
 [ 1. -2.  1. ...,  0.  0.  0.]
 [ 0.  1. -2. ...,  0.  0.  0.]
              ...,
 [ 0.  0.  0. ..., -2.  1.  0.]
 [ 0.  0.  0. ...,  1. -2.  1.]
 [ 0.  0.  0. ...,  0.  1. -2.]]
```

7.2 Initial and boundary conditions

```
def initial_condition(x=X, init='gaussian'):
    """Return the initial temperature over the inner points of the
    grid (where are unkown mesh points are).
    """
    if init == 'gaussian':
        return 100 * np.exp(-(x - 0.5)**2 / (2 * 0.1**2))
    elif init == 'zero':
        return np.zeros_like(X)

def right_bc(t):
    """Return the right boundary condition and its time derivative"""
    return 100.0

def left_bc(t, amplitude=0.0, omega=2 * np.pi / 1000.0):
    """Return the left boundary condition and its time derivative """
    # set amplitude to, say, 50 deg to see the effect of an
    # oscillating temperature
    return amplitude * np.sin(omega * t)
```

7.3 ODE solver

Solve the ODE

$$\frac{\partial T}{\partial t} = F(t, T(t)) = \alpha \frac{\partial^2 T}{\partial x^2} \qquad \approx \alpha \frac{1}{h^2}(\mathbf{AT} + \mathbf{B}), \tag{37}$$

where **A** is the discretized Laplace operator, **T** is the vector of unkown temperatures, and **B** is an additional vector storing the boundary conditions at the left and right boundary. These can be time dependent.

$$\mathbf{A} = \begin{pmatrix} -2 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & \ddots & \ddots & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & 1 \\ 0 & \cdots & \cdots & 0 & 1 & -2 \end{pmatrix}, \quad \mathbf{T} = \begin{pmatrix} T_0 \\ T_1 \\ \vdots \\ \vdots \\ T_{N-1} \\ T_N \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} -T(x=0,t) \\ 0 \\ \vdots \\ \vdots \\ 0 \\ -T(x=L,t) \end{pmatrix}, \tag{38}$$

7.3.1 Define the function $F(t, T)$ for the Runge-Kutta solver

```
def F(t, T, alpha=1.72e-5):
    """Return alpha d^2 T(x,t) / dx^2 using central finite difference
    approximation.
    """
    # Store boundary conditions as a vector
    b = np.zeros(N)
    b[0] = left_bc(t)
    b[-1] = right_bc(t)
    # Return dT/dt = alpha d^2T/dx^2 = F(t, T)
    return  alpha * (np.dot(A, T.transpose()) + b) / H**2
```

7.3.2 Runge-Kutta method

This is copied from notebook 3 (ODE II).

```python
def rk4(F, a, b, ya, n):
    """Solve the first order initial value problem
         y'(t) = F(t, y(t)),
          y(a) = ya,
    using the Runge-Kutta method and return a tuple made of two arrays
    (tarr, yarr) where 'ya' approximates the solution on a uniformly
    spaced grid 'tarr' over [a, b] with n elements.

    Parameters
    ----------
    F : function
         A function of two variables of the form F(t, y), such that
         y'(t) = F(t, y(t)).
    a : float
         Initial time.
    b : float
         Final time.
    n : integer
         Controls the step size of the time grid, h = (b - a) / (n - 1)
    ya : float
         Initial condition at ya = y(a).
    """
    tarr = np.linspace(a, b, n)
    h = tarr[1] - tarr[0]
    ylst = []
    yi = ya
    #import pdb; pdb.set_trace()
    for t in tarr:
        #print yi[len(yi) // 2]
        ylst.append(yi.copy())
        k1 = F(t, yi)
        k2 = F(t + 0.5 * h, yi + 0.5 * h * k1)
        k3 = F(t + 0.5 * h, yi + 0.5 * h * k2)
        k4 = F(t + h, yi + h * k3)
        yi +=  h / 6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4)
    yarr = np.array(ylst)
    return tarr, yarr
```

7.3.3 Define the solver

```
def solve_heat(init='gaussian'):
    """Solve the Heat equation and return the time and Temperature arrays
    init : string
        Controls the initial boundary condition.
        Current options are 'gaussian' and 'zero'
    """
    Tinit = initial_condition(X, init)  # of size N
    tarr, Tarr = rk4(F, TIME_MIN, TIME_MAX, Tinit, NTIME)
    return tarr, Tarr
```

7.3.4 Solve the heat equation and plot the result

```
INIT = 'zero'
tarr, Tarr = solve_heat(INIT)
plt.figure(1)
def plot_sol(frame=0, init='zero'):
    global tarr, Tarr, INIT
    if init != INIT:
        tarr, Tarr = solve_heat(init)
        INIT = init
    plt.clf()
    plt.plot(X, Tarr[frame], '--r', lw=2, label='transient (heat): t=%d' % frame)
    plt.plot(X, 100*X, 'k', lw=2, label='steady state (laplace)')
    plt.axis([0, 1, -10, 100])
    plt.legend(loc='best')
    plt.xlabel('Position along rod x [m]')
    plt.ylabel('Temperature (deg C)')
if LECTURE:
    i = interact(plot_sol, frame=[0, NTIME - 1], init=('zero', 'gaussian'))
else:
    plot_sol(frame=NTIME-1, init='zero')
plt.savefig('fig04-04.pdf')
```

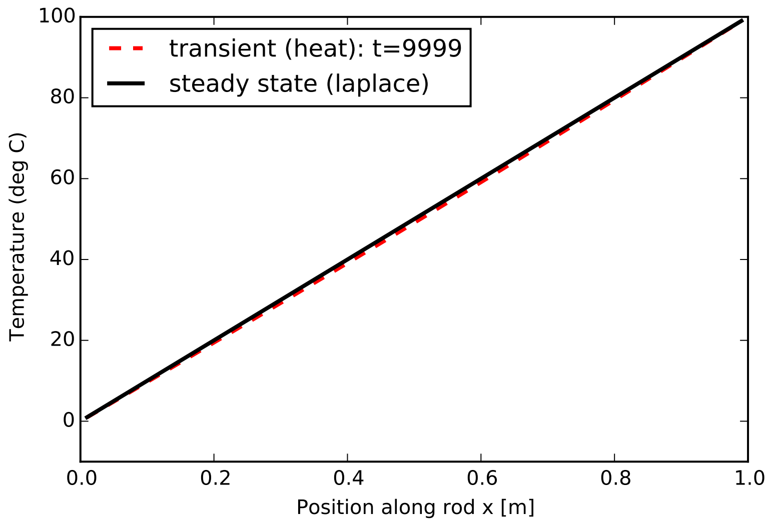

Figure 4: Solution of the 1D Heat Equation using RK4 for time integration, and the 2nd order central approximation for the second derivative for the spatial derivative.