Intermediate Matlab Course*

Dr Krishnakumar Gopalakrishnan
Computational Methods Hub
Imperial College London

May 13, 2019

*Exercises suitably adapted from the module 'ME2 Computing' taught at Imperial College London with the permission of the lecturer.

Contents

1

Bi-section Scheme for Finding Roots

1.1 Introduction

A numerical solution (or root) of an equation $f(x) = 0$ is a value of x that satisfies the equation approximately, but not necessarily exactly. A simple and robust algorithm (at least for scalar functions) to compute the roots of an equation is the bi-section method.

If, within a given interval $[a, b]$, $f(x)$ is continuous and the equation has a solution, then $f(x)$ will have opposite signs at the end points of $[a, b]$ *i.e.*, $f(a) \cdot f(b) < 0$. To start with, the midpoint between *i.e.*, $x_0 = \frac{1}{2}(a+b)$ is taken as the first estimate of the numerical solution. The true root is bracketed within either one of the shorter intervals $[a, x]$ or $[x, b]$. The process is repeated by considering this shorter interval as the new starting interval. The procedure finishes when the bracketing interval is less than a specified tolerance ε. The process is qualitatively illustrated in fig. 1.1.

1.2 Applying the Basic Bisection Method

Find the roots of the equation $f(x) = x^2 + (x-2)^3 - 5 = 0$.

1. Write a function *myfunc* that receives a value of x and outputs the corresponding value of $f(x)$.

2. Write another function *mybisection* to find the roots of *myfunc* within a given interval $[a, b]$ and a specified termination tolerance ε.

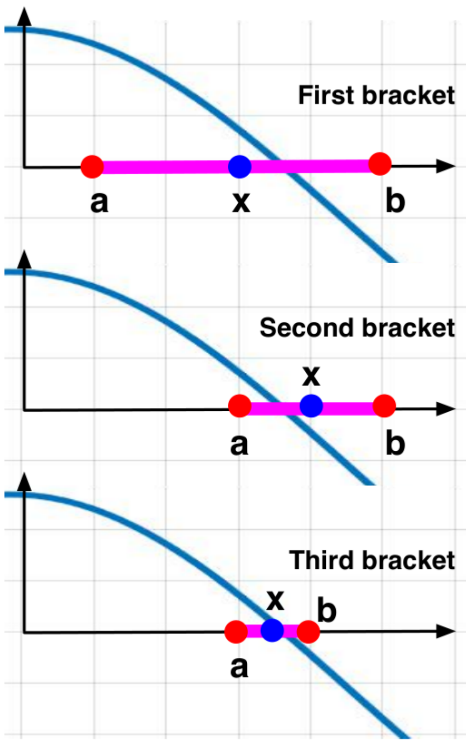

Figure 1.1 Qualitative illustration of the bisection method

3. Find the numerical root of the equation. To decide about the starting interval, it may be helpful to plot $f(x)$ versus x and visually inspect where the solution lies. Find and compare the root with tolerances $\varepsilon = 0.1$, $\varepsilon = 0.01$ and $\varepsilon = 0.001$.

1.3 Exploring the limits of the bi-section method

Find the root of the equation $f(x) = \sin\left(x^2\right) - x + 5$.

1. Alter *myfunc* with this $f(x)$. Invoke *mybisection* with the interval $[3.8, 6]$ and a tolerance of $\varepsilon = 0.001$.

2. Repeat the exercise by slightly changing the initial interval as $[3.8, 5.5]$ whilst keeping the same tolerance of $\varepsilon = 0.001$.

3. Why are the two roots different? Plot the function in a range that comprises both the intervals *e.g.* 0 to 10, to explore what is happening.

4. Find the roots of the equation $f(x) = (x - 2)^2$. What is the problem in finding the roots?

5. Find the roots of the equation $f(x) = \tan(x)$ within the range $[1, 2]$. What is the problem when finding the roots?

Apart from the ad-hoc function *mybisection*, you could also use some of the Scipy built-in functions to find roots. These built-in functions are more robust, offer additional functionality and have a wide-range of exception handling routines. Repeat the above exercises with suitable scipy built-ins.

1.4 Rugby for Dummies

Captain Dylan Hartley, tall $y_{\text{start}} = 1.85\,\text{m}$, throws the ball to his teammate Nathan Hughes, tall $y_{\text{end}} = 1.96\,\text{m}$ who is standing at a distance $x = 20\,\text{m}$ away. The motion of the ball is described by the projectile equation

$$y = x\tan(\theta) - \frac{1}{2}\frac{x^2 g}{v_0^2}\frac{1}{\cos^2(\theta)} + y_{\text{start}} \tag{1.1}$$

where $g = 9.81\,\text{m}\,\text{s}^{-2}$ is the gravitational acceleration, v_0 is the initial velocity of the ball and θ is the angle of the throw.

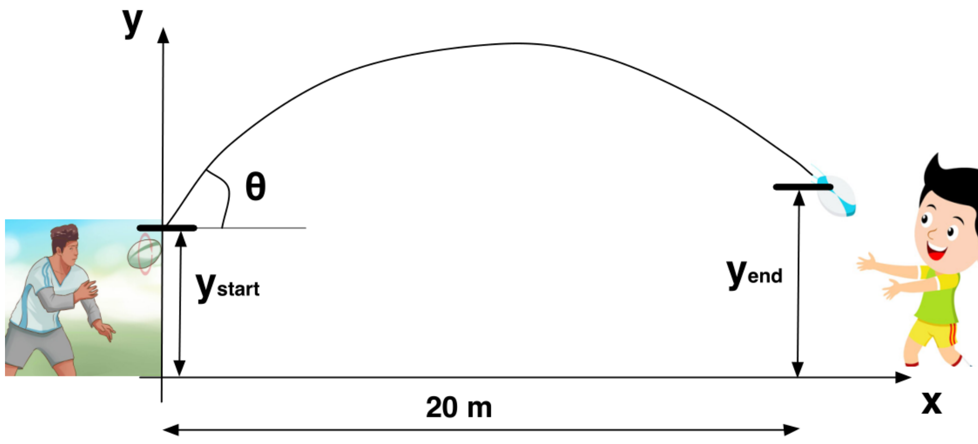

Figure 1.2 Cartoon illustration of the problem

1. Tell Dylan at which angle he needs to throw the ball with $v_0 = 15\,\text{m}\,\text{s}^{-1}$ in order to pass it to Nathan.

2. Plot the animated motion of the ball.

2

Numerical Integration

2.1 Introduction

NUMERICAL integration is useful when the analytical form of an integrand is unknown or when it would be difficult to evaluate the analytical integral.

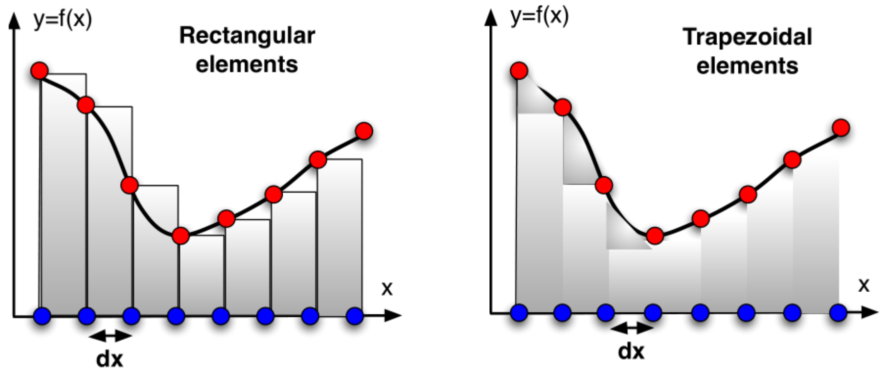

Figure 2.1 Illustration of two basic numerical methods of integration

The most intuitive way of evaluating the integral of a function is by approximating each data point (x, y) with a rectangular bar of height y and width Δx. The overall integral is then computed as the sum of the areas of all the rectangular elements. Though simple, this method is quite inaccurate. A better method is to consider trapezoidal elements.

2.2 Simple One-Dimensional Integration

1. Write a function *mytrapz* receiving a set of points x and y, and outputting the numerical integral of y within the interval specified by x. Test the function by integrating $y(x) = -(x-2)^2 + 4$ in the interval $[0, 4]$. Compare the numerical result against the analytical solution.

2. The accuracy of the numerical result can be improved by reducing the size of the trapezia and augmenting their number within the interval of integration *i.e.*, augmenting the number of data points by reducing their width. What is the maximum step size Δx necessary to contain the numerical error to within 1 % of the exact value?

3. Numpy has its own *trapz* function to numerically integrate a set of data points with the trapezoidal scheme. Apply it to the previous $y(x)$ and compare the result against that from Step 1.

Numerical integration using the trapezoidal method is less accurate in regions where the function changes rapidly. To increase the accuracy, it would be necessary to reduce the step size where the function varies more rapidly *i.e.*, to adapt the width of the step to the derivative of the function.

The Numpy function *trapz* is not adaptive. On the contrary, the Numpy function *quad* is. However, unlike *trapz* it can only be used with closed-form functional expressions and not with a set of data points.

4. Use the function *quad* to integrate $y = \sqrt{x}$ from 0 to 1.

5. Use *quad* to integrate $y(x) = -(x-2)^2 + 4$ in the interval $[0, 4]$. Compare the result against those obtained by the custom function *mytrapz* and the exact solution.

2.3 River Thames Basin in London

The file `Thames.txt` contains the shape of the river Thames basin (units in metres) within the London region.

1. Read in the data from the file and plot the two banks of the river together (with aspect ratio 1:1 for the two axes), to visualise the shape of the basin.
2. Compute the surface occupied by the basin.

2.4 Multiple Integrals: the domes of Samarkand

A two-dimensional integral has the form of:

$$I = \iint_A z(x,y)\,dA = \int_a^b dx \int_{c(x)}^{d(x)} z(x,y)\,dy = \int_a^b G(x)\,dx \tag{2.1}$$

where A is the domain of integration.

The two-dimensional integral can be computed numerically, by applying any of the methods used in section 2.2 twice. Firstly, the integral

$$G(x) = \int_{c(x)}^{d(x)} z(x,y)\,dy \tag{2.2}$$

is computed for any values of x. Then the total integral is obtained as $I = \int_a^b G(x)\,dx$.

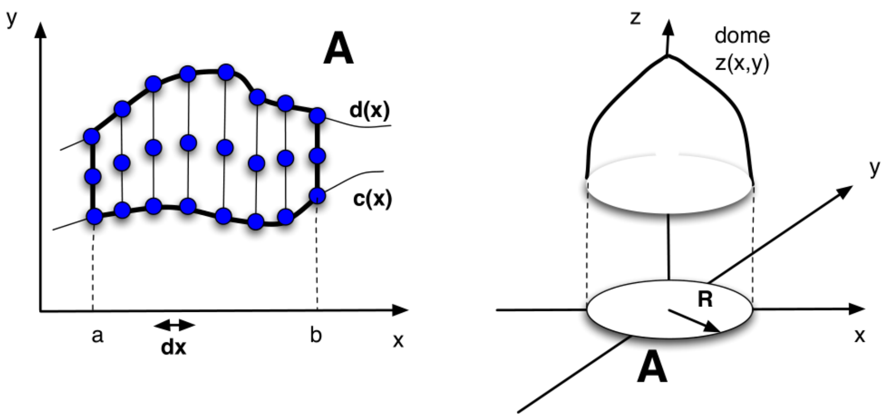

Figure 2.2 Illustration of multiple integrals

1. Plot the function $z(x,y) = \sqrt{R - \sqrt{x^2 + y^2}}$ in the domain $A : x^2 + y^2 < R^2$, with $R = 5$.
2. Compute the volume of the dome described by $z(x,y)$ in the domain A.

3. Show, by differentiating (both analytically and numerically) that the dome has a cusp at $(0, 0)$.

4. Note that the shape of the dome is determined by $z(x, y)$. With $z(x, y) = \sqrt{R^2 - x^2 - y^2}$, the dome is a perfect hemisphere. For this case, you may check your volume computation against the exact value.

3

Heat Conduction in One Dimension

3.1 General Overview & Objective

The goal of this exercise is to numerically solve the one dimensional heat conduction problem *i.e.*, a one-dimensional diffusion equation, with suitable constraints at both boundaries.

The heat condition within an object whose length far exceeds its thickness can be represented by the one-dimensional Partial Differential Equation (PDE) given in eq. (3.1)

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2} \tag{3.1}$$

where $T \equiv T(x, t)$ is the temperature of the object at each spatial location x along the length of the object at time t. α is the thermal diffusivity of the material ($\mathrm{m}^2\,\mathrm{s}^{-1}$). The objective is to evaluate $T(x, t)$ by numerically solving eq. (3.1) with the given boundary condition.

3.2 Finite Difference Discretisation

Let the superscript p refer to time indices and the subscript i refer to spatial nodes. In this workshop, we use only a fixed time step Δt as well as a fixed spatial resolution Δx[*]. Therefore, T_i^p refers to temperature at time p at node-index i of the domain *i.e.*, at $t = p\Delta t$ and $x = i\Delta x$.

[*]In almost all real-life scenarios, adaptive time-stepping can dramatically speed up simulations. Similarly, a non-uniform meshing scheme may be considered depending on the problem's complexity.

A backward difference scheme may be considered for temporal discretisation while a central difference method may be used for approximating the second order derivatives in the spatial co-ordinates.

$$\frac{dT}{dt} \approx \frac{T^{p+1} - T^p}{\Delta t} \tag{3.2}$$

$$\frac{d^2T}{dx^2} \approx \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} \tag{3.3}$$

Therefore, the PDE of eq. (3.1) is discretised as

$$\frac{T^{p+1} - T^p}{\Delta t} = \alpha \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2} \tag{3.4}$$

If the initial temperature is known, then the temperature at all future fixed intervals in the interior of the object can be therefore evaluated as

$$T_i^{p+1} = \frac{\alpha \Delta t}{\Delta x^2}\left(T_{i+1}^p + T_{i-1}^p\right) + \left(1 - \frac{2\alpha \Delta t}{\Delta x^2}\right) T_i^p \qquad i = 2 \dots N-1 \tag{3.5}$$

where N is the total number of spatial nodes.

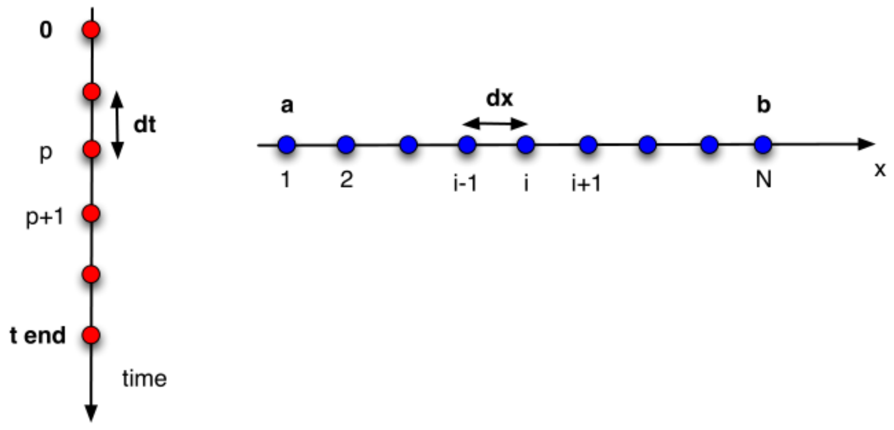

Figure 3.1 Illustration of temporal and spatial discretisation

3.3 Heat conduction in a bar

A steel bar ($\alpha = 1.172 \times 10^3\,\mathrm{m^2\,s^{-1}}$) of length 0.5 m is initially at a uniform temperature of 10 °C. The two extremes are suddenly brought to a temperature of 50 °C and kept at this temperature. Determine the temperature distribution within the bar after one hour. The problem is qualitatively illustrated in fig. 3.2.

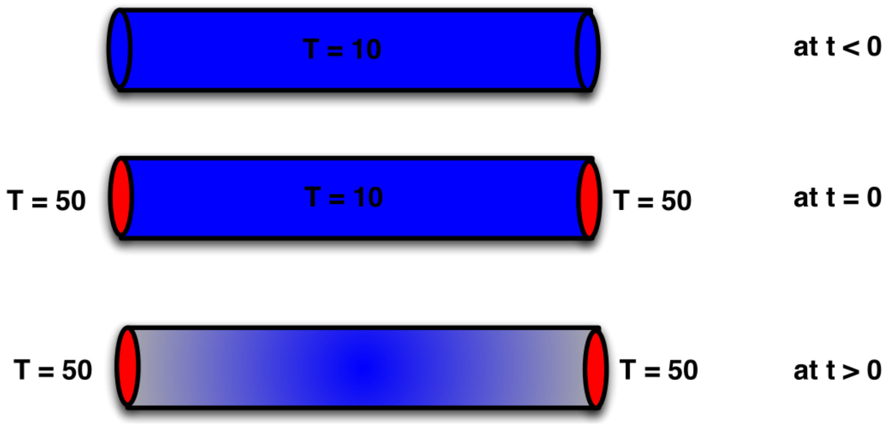

Figure 3.2 Heat conduction in a bar with Dirichlet boundary conditions

1. Write a script to numerically solve the heat conduction equation of eq. (3.1). The input data for the problem are: *a*) the value of the solution at the end-points $T(a,t)$ and $T(b,t)$, *b*) the initial uniform temperature T_0 for all spatial points, *c*) the fixed time-step Δt and the fixed spatial step, Δx, and *d*) the desired computational time span t_{end}.

2. Use $\Delta t = 1\,\text{s}$ and $\Delta x = 0.01\,\text{m}$. Plot $T(x, t_{\text{end}})$.

3. Repeat the above calculation with $T(a) = 50$ and $T(b) = 70$.

3.4 Heat conduction in a bar with external heat source

Consider the same bar as in section 3.3 with an initial uniform temperature of 10 °C. The temperature at the middle of the steel rod is suddenly increased to $T = 100\,^\circ\text{C}$ through a source and kept constant at this value. The bar is immersed in a large pool of water with constant temperature $T_w = 5\,^\circ\text{C}$. Therefore, the bar is subjected to convective heat exchange with the water *i.e.*, $k\left(\frac{\partial T}{\partial x}\right)_a = h\,(T_a - T_w)$ and $k\left(\frac{\partial T}{\partial x}\right)_b = -h\,(T_b - T_w)$. The thermal conductivity for steel is $k = 40\,\text{W}\,\text{m}^{-1}\,\text{K}^{-1}$ and the heat transfer coefficient $h = 500\,\text{W}\,\text{m}^{-2}\,\text{K}^{-1}$.

1. Amend the script of section 3.3 to incorporate the mixed boundary conditions. A few helpful hints:

- Set the temperature in the mid-point of the grid to be 100 °C, irrespective of any other numerical computations in the interior points to simulate the source.
- For the mixed boundary conditions, it may be helpful to write down the form of the numerical solution for the first and last spatial nodes $T_1^{p+1} = \dots,\ T_N^{p+1} = \dots$.

2. Compute the temperature distribution with $\Delta t = 1\,\mathrm{s}$ and $\Delta x = 0.01\,\mathrm{m}$. Plot an animated spatial distribution of temperature within the bar from $t = 0$ to $t = t_{\mathrm{end}} = 1200\,\mathrm{s}$.

3.5 Stability of the finite difference numerical method

Solving a time-marching PDE with boundary conditions can become numerically unstable, particularly when the resolution of the spatial and temporal discretisation values are not chosen carefully.

If the time step Δt becomes too large, the last term in eq. (3.5) can become negative. The same is true when Δx is reduced to a too small value. When the last term in eq. (3.5) becomes negative, the numerical computation becomes unstable *i.e.*, the local truncation error is amplified at each successive time step. Therefore, in order for the computation to remain bounded numerically,

$$\left(1 - \frac{2\alpha\,\Delta t}{\Delta x^2}\right) > 0 \tag{3.6}$$

$$\frac{\alpha\,\Delta t}{\Delta x^2} < \frac{1}{2} \tag{3.7}$$

Equation (3.7) is a stability criterion known as the Courant condition of stability that imposes a constraint relationship in the choices of Δx and Δt. For a chosen spatial resolution, this implies an upper bound on the simulation time step Δt to ensure numerical stability.

Use the script from section 3.4 with $T(a) = 50$ and $T(b) = 50$.

1. Compute $T(x, t_{\mathrm{end}})$ with $\Delta t = 1\,\mathrm{s}$ and $\Delta x = 0.01\,\mathrm{m}, 0.005\,\mathrm{m}$ and $0.001\,\mathrm{m}$. Calculate the Courant condition for all the three cases and plot $T(x, t_{\mathrm{end}})$.
2. Repeat the calculation with $\Delta x = 0.001\,\mathrm{m}$ and a reduced $\Delta t = 0.04\,\mathrm{s}$.