ME2 Computing- Session 1: Revision of arrays and introduction to Numpy, graphics

Learning outcomes:

- Regain confidence with concepts of arrays and their implementation
- Be able to implement and manipulate arrays with Numpy formalisms
- Get familiar with advanced graphics tools and techniques
- Be able to choose appropriate array representation for vector field analysis

Before you start

In your H drive create a folder *H:\ME2MCP\Session1* and work within it.

Task A: Generate an array with a non-uniform range

1. Generate an array *x* of numbers in the range *[-5 : 5]* with the following steps:

$$\Delta x = 0.5 \text{ in } -5 \leq x \leq -2$$
$$\Delta x = 0.05 \text{ in } -2 < x < 3$$
$$\Delta x = 0.5 \text{ in } 3 \leq x \leq 5$$

2. Compute the function: $y = \sin(x)$.

3. Plot, with scattered points, *y* vs *x.*

Task B: Multi-dimensional arrays

1. Represent with appropriate variables the two functions:

$$f(x, y) = \sin x \cdot \cos y$$
$$g(x, y) = \cos x \cdot \sin y$$

in the range *x = [-2π : 2π]* and *y = [-π : 2π]* with steps Δx = Δy = *0.1*.

2. Compute the two functions:

$$s(x, y) = f(x, y) + g(x, y)$$
$$p(x, y) = f(x, y) \cdot g(x, y)$$

Task C: Surface plots

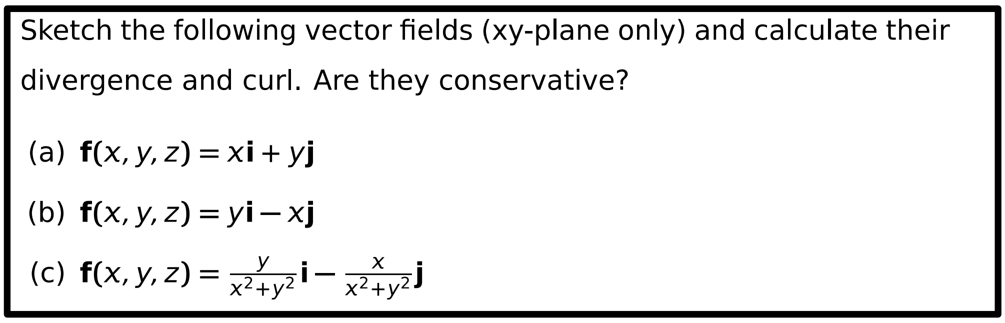
1. Plot, both with a surface plot and a contour plot separately, the functions $s(x, y)$ and $p(x, y)$ of Task B.

2. Consider the function:
$$r(x, y, t) = f(x, y) \cdot e^{-0.5t}$$
in the same range of *x* and *y* as in Task B and *t = [0 : 10]* with Δt = 0.05.

3. Plot, with a surface plot, $r(x, y, t = 0)$ and $r(x, y, t = 5)$.

Task D: Vector plots

From Maths tutorial sheet:



Sketch the following vector fields (xy-plane only) and calculate their divergence and curl. Are they conservative?

(a) $\mathbf{f}(x, y, z) = x\mathbf{i} + y\mathbf{j}$

(b) $\mathbf{f}(x, y, z) = y\mathbf{i} - x\mathbf{j}$

(c) $\mathbf{f}(x, y, z) = \frac{y}{x^2+y^2}\mathbf{i} - \frac{x}{x^2+y^2}\mathbf{j}$

1. Represent with appropriate variables the vector fields $\boldsymbol{f}(x, y, z)$ in the range *x = y = [-5 : 5]* with intervals *dx = dy = 0.1*.

2. Plot as quivers and streamlines the three $\boldsymbol{f}(x, y, z)$, and observe whether the fields are conservatives, irrotational, etc.

Task E**: Bonus (a bit challenging)* ***Slicing with conditions

1. Compute, with vectorised operations, the value:
$$ym = |y| = |\sin(x)|$$
in the range *x = [-5 : 5]* with *dx = 0.1*.

2. Compose the array *ymsat* such that:
$$\begin{cases} ymsat = 0 & \text{for} -5 \leq x \leq 0 \\ ymsat = ym & \text{for } x > 0 \text{ and } ym \leq 0.5 \\ ymsat = 0.5 & \text{for } x > 0 \text{ and } ym > 0.5 \end{cases}$$

ME2 Computing- Session 2: Sets of Linear Equations and Gauss Elimination

Learning outcomes:

- Being able to solve a set of linear equations numerically
- Being able to implements Gauss elimination
- Being able to express engineering problems in matrix-vector terms

Before you start

In your H drive create a folder *H:\ME2MCP\Session2*and work within it.

Introduction

A set of linear equations can be written numerically in the matrix-vector form as:

$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 = b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 = b_4 \end{cases}$	**Eq. 1**	$\begin{bmatrix} \mathbf{a_{11}} & \mathbf{a_{12}} & \mathbf{a_{13}} & \mathbf{a_{14}} \\ \mathbf{a_{21}} & \mathbf{a_{22}} & \mathbf{a_{23}} & \mathbf{a_{24}} \\ \mathbf{a_{31}} & \mathbf{a_{32}} & \mathbf{a_{33}} & \mathbf{a_{34}} \\ \mathbf{a_{41}} & \mathbf{a_{42}} & \mathbf{a_{43}} & \mathbf{a_{44}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$ $A \cdot x = b$

Two types of numerical methods are possible for solving sets of linear equations: **direct** and **iterative**. In direct methods, the solution is computed manipulating the equations. In iterative methods, an initial solution is guessed and then used iteratively to obtain more accurate solutions. In this session we will implement a direct method only, namely the Gauss Elimination.

Task A: Gauss elimination

If the given set of equations can be manipulated into the form:

$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 = b_1 \\ d_{22}x_2 + d_{23}x_3 + d_{24}x_4 = d_2 \\ e_{33}x_3 + e_{34}x_4 = e_3 \\ f_{44}x_4 = f_4 \end{cases}$	$\begin{bmatrix} \mathbf{a_{11}} & \mathbf{a_{12}} & \mathbf{a_{13}} & \mathbf{a_{14}} \\ \mathbf{0} & \mathbf{d_{22}} & \mathbf{d_{23}} & \mathbf{d_{24}} \\ \mathbf{0} & \mathbf{0} & \mathbf{e_{33}} & \mathbf{e_{34}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{f_{44}} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ d_2 \\ e_3 \\ f_4 \end{bmatrix}$

then it is straightforward to find the solutions, starting from x_4 and proceeding backwards by substitution to find x_3, x_2 and x_1.

This manipulation can be done in repeated steps. At each step *i*, the variable x_i is eliminated from the $(i+1)^{th}$ row downwards, (i.e. all the coefficients in column *i* are set to zero, starting from the row below *i*).
For example, at step 1, starting from Eq. 1, the variable x_1 is eliminated from the second row downwards, (i.e. all the coefficients in column 1 are set to zero starting from row 2):

row 1 is unchanged new row 2 = row 2 - $a_{21}/\boldsymbol{a_{11}}$ · row 1 new row 3 = row 3 - $a_{31}/\boldsymbol{a_{11}}$ · row 1 new row 4 = row 4 - $a_{41}/\boldsymbol{a_{11}}$ · row 1	$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & d_{22} & d_{23} & d_{24} \\ 0 & d_{32} & d_{33} & d_{34} \\ 0 & d_{42} & d_{43} & d_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ d_2 \\ d_3 \\ d_4 \end{bmatrix}$	**Eq. 2**

The common coefficient, $\boldsymbol{a_{11}}$, is also knows as the **pivot**.

At step 2, starting from Eq. 2, the variable x_2 is eliminated from the third row downwards, (i.e. all the coefficients in column 2 are set to zero starting from row 3):

row 1 is unchanged new row 2 is unchanged new row 3 = new row 3 - $d_{32}/\boldsymbol{d_{22}}$ · new row 2 new row 4 = new row 4 - $d_{42}/\boldsymbol{d_{22}}$ · new row 2	$\Rightarrow \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ 0 & d_{22} & d_{23} & d_{24} \\ 0 & 0 & e_{33} & e_{34} \\ 0 & 0 & e_{43} & e_{44} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ d_2 \\ e_3 \\ e_4 \end{bmatrix}$	**Eq. 3**

The pivot at this step is the coefficient $\boldsymbol{d_{22}}$.

These steps are repeated, until the matrix A is reduced into an upper triangular form.

If you find difficult to understand this logic, read the supplemental document on Gauss elimination (on BB), by paying particular attention to the colour code adopted in it.

1. Write a function, *MyGauss*, that receives a set of *n* linear equations, in the form of a matrix *A* and a vector *b*, and outputs the vector solution *x*.

2. Test the function to solve the set of equations:

$$\begin{cases} 8x_1 - 2x_2 + x_3 + 3x_4 = 9 \\ x_1 - 5x_2 + 2x_3 + x_4 = -7 \\ -x_1 + 2x_2 + 7x_3 + 2x_4 = -1 \\ 2x_1 - x_2 + 3x_3 + 8x_4 = 5 \end{cases}$$

   Solution: [1.3232   1.5657   -0.6061   0.7172]

Task B: Further skills

Sometimes the set of equations does not have a solution, even though the numerical method may still provide a solution, which is obviously wrong. If we consider the matrix-vector form:

$$A \cdot x = b$$

the solution can be expressed as:

$$x = A^{-1} \cdot b$$

where A^{-1} is the inverse of matrix A. Therefore, there will exist a solution if the matrix *A* is invertible. This is equivalent of saying that the determinant of matrix *A* is non-zero.

1. For this set of linear equations:

$$\begin{cases} 4x + 3y = 2 \\ 8x + 6y = 1 \end{cases}$$

   Plot the two equations, each as a line *y* vs *x*, on the same graph.

Solve them numerically: what does it happen?
Evaluate the determinant of matrix A with the Numpy function numpy.linalg.det().

2. There are cases, when the solution exists, where numerical methods can provide completely inaccurate results. These cases are said to be **ill conditioned**.

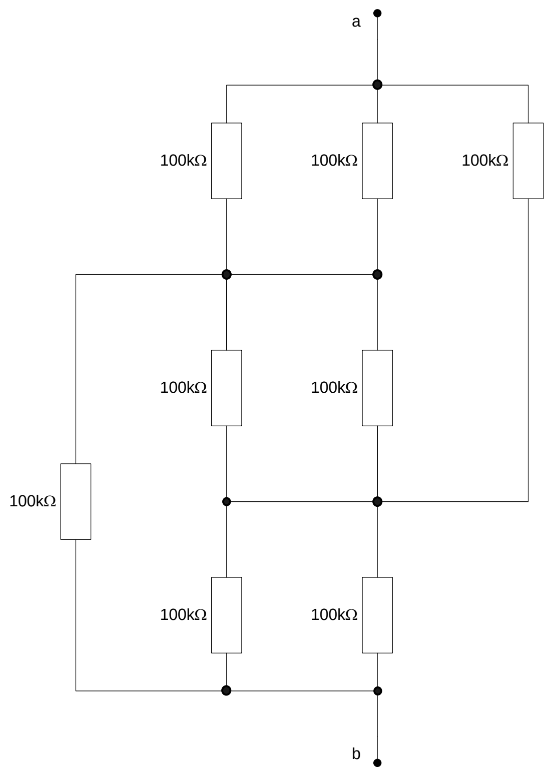
   Consider these two sets of linear equations:

   $$\begin{cases} 400x - 201y = 200 \\ -800x + 401y = -200 \end{cases} \qquad \begin{cases} \mathbf{401x - 201y = 200} \\ \mathbf{-800x + 401y = -200} \end{cases}$$

   Solve the two sets of equations separately. How do the numerical solutions differ from each other?
   This is a case of an ill conditioned problem: with a modest change in one of the coefficients (often induced by a numerical error) one would expect only a small change in the solution. However, in ill conditioned cases the change is quite significant: the solution is very sensitive to the values of the coefficients.

Task C: Electrical linear circuits: ME2-MTX tutorial 1.10.6 (resistor network 5)

1. Consider the ME2 Mechatronics exercise 1.10.6. After writing (in the Mechatronics tutorial) the set of KVL and KCL equations, solve the circuit with the numerical values given for resistors. Make use of the function *MyGauss* to find out the numerical value of the total resistance between terminals **a** and **b**.



ME2 Computing- Session 3: Numerical Integration

Learning outcomes:

- Being able to compute numerically the integration of a proper integral
- Being able to compute numerical integration for a set of points not positioned equidistantly

Before you start

In your H drive create a folder *H:\ME2MCP\Session3*and work within it.

Please provide feedback at: www.menti.com with code 79 79 176

Task A: Trapezium rule for functions with equidistant nodes

1. Write a Python function, *trapzeqd*, receiving a set of points *x* and *y*, and outputting the numerical integral of *y* within the interval specified by *x*. Assume that the nodes *x* are equidistant.
   Test the Python function by integrating:

$$I = \int_0^b \frac{1}{\sqrt{x^{20.10} + 2020}} dx$$

   in the interval *x = [0 : b=1.75]* with 5 nodes.

2. Increase the interval of integration with *b = 10, 100, 1000, 10000*, and recompute the integral with same number of nodes (5).
   Plot the values of *I* vs *b*.

3. Repeat the numerical integration for the intervals in Part 2, but retaining the same interval *h = 0.5*, i.e. by increasing progressively the number of nodes.
   Replot the values of *I* vs *b*.

Task B: Numerical integration of diverging improper integrals

4. Recompute the numerical integrations as in Task A2 and A3, but with the integrand function:

$$I = \int_0^b \frac{1}{\sqrt{x^{1.10} + 2020}}$$

Task C: Trapezium rule for functions with non-equidistant nodes

1. Write another Python function, *trapz*, receiving a set of points *x* and *y*, and outputting the numerical integral of *y* within the interval specified by *x*. The values in x might not be distanced at same intervals.

Task D: The river Thames basin in London

The files *(xn.txt, yn.txt)* and *(xs.txt, ys.txt)* contain the *(x,y)* spatial coordinates of the north and south bank, respectively, of the river Thames (units in meters), within the Central London region (between Chiswick and Woolwich).

you are here doing Computing
Update 2020: in normal conditions!
Marble Arch
SOHO
London
Hyde Park
CITY OF LONDON
Big Ben
Victoria and Albert Museum
Saatchi Gallery
Imperial War Museum
Isle of Dogs
London City Airport
Stamford Bridge
Royal Observatory Greenwich
Clapham
River Thames

1. Read in the data from the files and plot the two banks of the river together, to visualise the shape of the basin. (To plot with aspect ratio 1:1 for the two axes use *pl.axis('equal')*, after plotting.

2. Compute the surface occupied by the basin in Km^2.

Task E:* (Optional) *Multiple integrals*: *the domes of Samarkand in Uzbekistan

A two-dimensional integral has the form of:

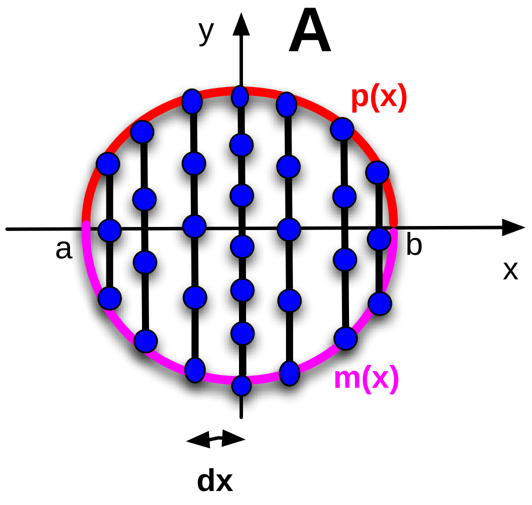
$$I = \iint_A z(x,y)dA = \int_a^b dx \int_{m(x)}^{p(x)} z(x,y)dy = \int_a^b G(x)dx$$

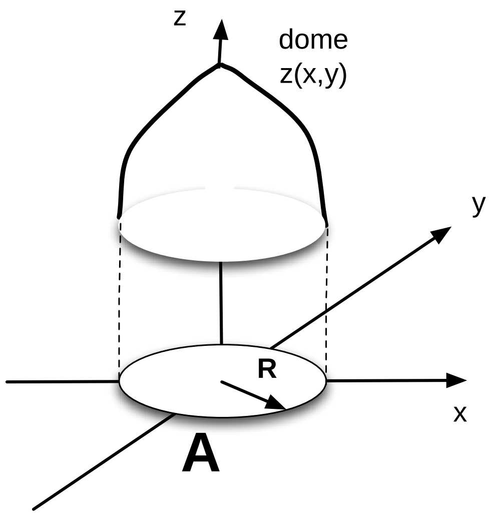
where A is the domain of integration.

The two-dimensional integral can be computed numerically, by applying the trapezium method twice. Firstly, the integral

$$G(x) = \int_{m(x)}^{p(x)} z(x,y)dy$$

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





1. Compute the volume of the dome described by $z(x,y) = \sqrt{R - \sqrt{x^2 + y^2}}$ in the domain A: $x^2 + y^2 < R^2$, with R = 5, using equidistant intervals both along the *x* and *y* axes, i.e. *dx = dy = 0.05*.

2. When $z(x,y) = \sqrt{R^2 - x^2 - y^2}$ the dome is a perfect hemisphere. For this case, you can check your computed volume against the exact value.

3. Plot the function $z(x,y)$.

ME2 Computing- Session 4: Numerical Interpolation

Learning outcomes:

- Being able to compute Lagrangian numerical interpolation
- Being able to compute Newton numerical interpolation
- Being able to interpolate with splines

Before you start:

In your H drive create a folder *H:\ME2MCP\Session4* and work within it.

Please provide feedback at: www.menti.com with code 79 79 176

Task A: Lagrangian polynomials and interpolation

1. Write a function, *Lagrangian*, to compute the Lagrangian polynomial *j* at a point *xp*, with given nodes *xn*.
   The function receives the values *j, xp* and the array of nodes *xn*, and returns the value

$$L_j(x_p) = \prod_{\substack{k=0 \\ k\neq j}}^{n} \frac{(x_p - x_k)}{(x_j - x_k)}$$

2. Write a script to interpolate, with Lagrangian polynomials, the function $f(x) = \sin(x)$ over the range *x = [0:3]* with step *0.05*, given the nodal values at:
   a) *xn = [1:2]* with 2 nodes: linear interpolation $p_1(x)$
   b) *xn = [1:2]* with 3 nodes: quadratic interpolation $p_2(x)$
   c) *xn = [1:2]* with 4 nodes: cubic interpolation $p_3(x)$

   Compare/plot the interpolating polynomials, $p_1(x), p_2(x), p_3(x)$ with/against those calculated manually in slides 46, 47 and 49, respectively. (You should end up with a plot like in slide 50).

3. **Error analysis**: compute the basic error (as defined in slide 52) for $p_1(x), p_2(x), \dots \dots, p_{13}(x), p_{14}(x)$ at x = π/2 (slide 53).

Task B: Newton interpolation

1. Write a function, *NewtDivDiff,* to compute the value of the Newton's Divided Difference $f[x_0, x_1, x_2, \dots, x_N]$. The function receives the two lists of nodal points *xn* and *yn* and returns the corresponding scalar value. (If you write the function in a recursive form it will be much shorter, as defined in slide 63).

2. Write a script to interpolate, with Newton's method, the function $f(x) = \sin(x)$ over the range *x = [0:3]* with step *0.05*, given the nodal values at:
   a) *xn = [1:2]* with 2 nodes: linear interpolation $p_1(x)$
   b) *xn = [1:2]* with 3 nodes: quadratic interpolation $p_2(x)$
   c) *xn = [1:2]* with 4 nodes: cubic interpolation $p_3(x)$

   Compare/plot the interpolating polynomials, $p_1(x), p_2(x), p_3(x)$ with/against those calculated with Lagrangian interpolation.

3. Interpolate the function (slide 76):
$$f(x) = \frac{1}{1 + 25x^2}$$
   in the range $-1 \le x \le 1$, with Newton's interpolation of order n = 1, 2, 3, 4, 5, … 14 and plot the interpolating polynomials (Runge's phenomenon).

Task C: Splines

1. Write a script to interpolate with cubic splines a range of points in the interval $a \le x \le b$ with given nodal information *xn, yn* and given clamped boundary conditions $y'(a)$, $y'(b)$.

   You can test the script with the function
$$f(x) = \frac{1}{1 + 25x^2}$$
   with $a = -1, b = 1, y'(a) = 0.074$, $y'(b) = -0.074$, by using 3, 5 and 11 nodes.

   Note: to invert the matrix you can use the function *MyGauss* you wrote in Session 2.

ME2 Computing- Session 5: Advanced Numerical Integration and Differentiation

Learning outcomes:

- Being able to compute Simpson and Adaptive Simpson Integration
- Being able to compute K-th order derivative
- Being able to combine numerical derivative and interpolation

Before you start:

In your H drive create a folder *H:\ME2MCP\Session5* and work within it.

Please provide feedback at: www.menti.com with code 79 79 176

Task A: Simpson Integration and Adaptive Simpson Integration

1. Write a function, *Simpson*, to integrate over a uniformly distributed set of nodal point.

2. Write an adaptive script to integrate numerically a given function *f(x)*, over a prescribed domain, until a desired tolerance is achieved.
   Test it by integrating $f(x) = \sin(x)$ over the domain [0:2π].

Task B: K-th order derivative

Write a function, *Derivative*, to compute the *k-th* order derivative for a given set of uniformly distributed nodal points (choose yourself to apply either the forward or backward scheme).

Task C: Smoothing derivatives with polynomial interpolation

1. Set the function *f(x) = sin(x)* within the domain [0:π] with 10 nodal points, (x_n, y_n).

2. Compute the 5-th derivative of these nodal points, by using the function *Derivative* from Task B. Plot the scattered nodal points and the derivative.

3. Interpolate, with Lagrangian polynomials, the nodal points (x_n, y_n), for 100 points over the same domain [0:π]. Recompute the 5-th derivative for the interpolated points. Plot the scattered nodal points and the derivative.

ME2 Computing- Session 6: Numerical solution of differential equations: initial value problems

Learning outcomes:

- Being able to solve first order ODEs with explicit methods
- Being able to solve first order ODEs with implicit methods
- Being able to solve a system of first order ODEs and Higher order ODEs

Before you start:

In your H drive create a folder *H:\ME2MCP\Session6* and work within it.

We will be testing Tasks A and B with the ODE: $\frac{dy}{dt} = -2yt - 2t^3$, whose analytical solution is: y$= 1 - t^2 + ce^{-t^2}$

Task A: Explicit methods: Forward Euler and RK4

1. Write a function, *FwEuler*, to solve a general ODE $\frac{dy}{dt} = F(t, y)$ with a forward Euler scheme. The function receives the initial condition, t_0 and y_0, the time step *h* and the desired final computational time t_{end} (all these input arguments are scalars). The function outputs two arrays, *t* and *y*, describing the specific solution *y(t)*. Within *FwEuler*, *F(t,y),* can be evaluated by invoking a separate function *func*.
   Explicit methods are subject to instabilities: consider this when choosing the value of *h*.

2. Write a function RK4 to perform as the function at point 1, but implementing a RK4 method instead.

Task B: Implicit methods: Backward Euler

1. Write a function, *BwEuler*, to solve the specific ODE under study, with a backward Euler scheme. The function receives the initial condition, t_0 and y_0, the time step $\mathrm{h}$and the desired final computational time t_{end}. The function outputs two arrays, *t* and *y*, describing the specific solution *y(t)*.

Task C: System of ODEs, with explicit methods

1. Modify the function *FwEuler*, into a new function *FwEulerTwo*, to solve the set of two ODEs:

$$\begin{cases} \frac{dy_1}{dt} = F_1(t, y_1, y_2) \\ \frac{dy_2}{dt} = F_2(t, y_1, y_2) \end{cases} \quad \text{with initial conditions} \quad \begin{cases} y_1(t = t_0) = y_1^0 \\ y_2(t = t_0) = y_2^0 \end{cases}$$

FwEulerTwo receives the vector Y_0 of initial values, the initial and final time t_0 and t_{end}, and the time step *h*. *FwEulerTwo* should output an array *t* and a two rows array *Y* with the solutions, $Y = \begin{bmatrix} y_1(t) \\ y_2(t) \end{bmatrix}$.
On the side, you also need to define two separate functions, *func1*, *func2*, to evaluate F_1 and F_2, respectively.

2. Test the function to solve the *predator-prey* problem applied to the London property rental market. The cycles of property rental prices vs population are described by the Lokta-Volterra's set of first order ODEs:

$$\begin{cases} \frac{d£}{dt} = 0.3£N - 0.8£ \\ \frac{dN}{dt} = 1.1N - N£ \end{cases} \quad \text{with initial conditions} \quad \begin{cases} £(t=0) = 0.8 \\ N(t=0) = 7 \end{cases}$$

where £ is the average price of house rentals (in thousands pounds) and N is the number of inhabitants (in millions). Compute the cycles in the period [0:40] years, in steps of 0.019 (i.e. weekly).

3. Plot, on the same graph, the results vs time. Plot also, in a different figure, the number of inhabitants vs the rental price.

Task D: Higher order ODEs: damped non-linear motion of a pendulum

The oscillation of a pendulum of mass *m*, attached to a weightless string, is described by the second order ODE:

$$\frac{d^2\theta}{dt^2} + \frac{c}{m}\frac{d\theta}{dt} + \frac{g}{L}sin\theta = 0$$

where *c* is the damping coefficient, *g* the gravitational acceleration and *L* the length of the string.
The pendulum initially is at rest, displaced at an angle θ_0.

The second order ODE can be reduced to a set of two first order ODEs, introducing a new dependent (artificial) variable *w*:

$$\begin{cases} w = \frac{d\theta}{dt} \\ \frac{dw}{dt} = \left(\frac{d^2\theta}{dt^2}\right) = -\frac{c}{m}\frac{d\theta}{dt} - \frac{g}{L}sin\theta = -\frac{c}{m}w - \frac{g}{L}sin\theta \end{cases}$$

1. Determine the motion of the pendulum, $\theta(t)$, for the first initial 15 seconds, with initial condition $\theta(t=0) = \pi/4$. (Use *FwEulertwo* with $\Delta t = 0.005s$). Use a mass of 0.5Kg and a string L = 1m. Observe the difference between the swinging in a dry place (c = 0.05Ns/m) and within a humid viscous environment (c = 0.18Ns/m).

ME2 Computing- Session 7: Numerical solution of differential equations: boundary value problems

Learning outcomes:

- Being familiar with the finite difference scheme
- Being familiar with direct and iterative methods
- Being familiar with the types of boundary conditions

Before you start:

In your H drive create a folder *H:\ME2MCP\Session7* and work within it.

ODE with boundary values

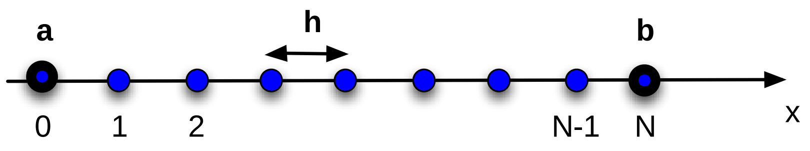
A second order boundary value problem ODE is specified by the ODE itself:

$$\frac{d^2y}{dx^2} + f(x)\frac{dy}{dx} + g(x)y = p(x)$$

by the domain of the solution, i.e. a < x < b, and by the boundary conditions.

The domain of the solution, *[a b]*, is subdivided into N intervals and defined with N+1 points (grid points). The derivatives of the ODEs are approximated at each grid point with the central difference finite difference scheme:

$\frac{dy}{dx} = \frac{y_{i+1} - y_{i-1}}{2h}$	$\frac{d^2y}{dx^2} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$



The ODE is firstly approximated numerically at the *interior points*, i = *1* to *N-1*, i.e.

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + f(x_i)\frac{y_{i+1} - y_{i-1}}{2h} + g(x_i)y_i = p(x_i)$$

Common terms are rearranged together as:

$$\left[\frac{1}{h^2} - \frac{f(x_i)}{2h}\right]y_{i-1} + \left[g(x_i) - \frac{2}{h^2}\right]y_i + \left[\frac{1}{h^2} + \frac{f(x_i)}{2h}\right]y_{i+1} = p(x_i) \quad i = 1 \dots N-1$$

and the equation rewritten in compact form:

$$a_i y_{i-1} + b_i y_i + c_i y_{i+1} = p_i \quad i = 1 \dots N-1$$

This procedure generates a set of *N-1* (linear) algebraic equations: every equation, on the left-hand side, is formed by three terms only.

The boundary conditions at the endpoints are then added to assign the constrains on the solution:

$$y_0 = y_a$$
$$y_N = y_b$$

At the end, a set of *N+1* (linear) algebraic equations is formed altogether, with unknown variables $y_0, y_1, y_1, \dots . y_{N-2}, y_{N-1}, y_N$, i.e.:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & .. & .. & 0 \\ a_1 & b_1 & c_1 & 0 & 0 & .. & .. & 0 \\ 0 & a_2 & b_2 & c_2 & 0 & .. & .. & 0 \\ 0 & 0 & a_3 & b_3 & c_3 & .. & .. & 0 \\ .. & .. & .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. & .. & .. \\ 0 & 0 & 0 & 0 & 0 & a_{N-1} & b_{N-1} & c_{N-1} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \\ \\ y_{N-1} \\ y_N \end{bmatrix} = \begin{bmatrix} y_A \\ p_1 \\ p_2 \\ p_3 \\ \\ \\ p_{N-1} \\ y_b \end{bmatrix}$$

These can then be computed either with a direct or an iterative method.
For a *direct* method, to invert the matrix, you can use your algorithm on Gauss elimination from Session2.
For an *iterative* method you can use either Jacobi or Gauss-Siedel method.

Task 1: Direct methods

1. Write a function, *myodebc*, that receives the boudaries of the domain *a* and *b*, the value of the solutions at these points, $y(a) = y_a$ and $y(b) = y_b$, and the number *N* of desired intervals. *myodebc* returns the grid points *x* and the solution *y(x)* at the grid points. The ODE is defined through an external function, *myfunc*, that receives the value of *x* and returns the values of *f(x)*, *g(x)* and *p(x)*.

2. Solve the ODE $\frac{d^2y}{dx^2} + 2x\frac{dy}{dx} + 2y - \cos(3x) = 0$, in the domain [0 π], with boundary conditions $y(0) = 1.5$ and $y(\pi) = 0$. Discretise the domain with 10 intervals. Plot *y(x)*.

3. Repeat the analysis with 100 intervals, and compare with the previous results.

Task 2: Iterative methods

1. Repeat Task 1, and write an equivalent function, *Jacobi*, by using the Jacobi iterative methods. The function Jacobi receives, in addition, as input argument, also the desired accuracy.
   Note that: since the iterative method is explicit, the algorithm might not converge, depending of the ODE.

2. Run the function Jacobi with various values of accuracy.

Task 3: Types of boundary conditions

The boundary conditions, specified at the two boundaries *a* and *b*, can be of different types:

- Dirichlet: two values for the solution, *y(x)*, are specified at *a* and *b*:
  $y(a) = BC_a$ and $y(b) = BC_b$ (this is the case for the example in Task 1)

- Neumann: two values for the derivative of the solution, $\frac{dy}{dx}$, are specified at *a* and *b*:
  $\frac{dy}{dx}(a) = BC_a$ and $\frac{dy}{dx}(b) = BC_b$

- Mixed (or Robin): two values for a combination of the solution, *y(x)*, and its derivative, $\frac{dy}{dx}$, are specified at *a* and *b*:
  $c_0\frac{dy}{dx}(a) + c_1 y(a) = BC_a$ and $c_2\frac{dy}{dx}(b) + c_3 y(b) = BC_b$

Note that to constrain the derivative at endpoint *a* it is necessary to implement the finite difference forward scheme, whilst at endpoint *b* the finite difference backward scheme is needed, instead.

Note also that by setting $c_0 = c_2 = 0$ the mixed boundary conditions become of Dirichlet type, and that by setting $c_1 = c_3 = 0$ the mixed boundary conditions become of Neumann type.

1. Modify the function, *myodebc*, to accommodate all the various types of boundary conditions. *myodebc* still receives the boundaries of the domain *a* and *b*, the boundary conditions at these points *BC(a)* and *BC(b)*, and the number *N* of desired intervals. In addition, it receives an array *c* of length 4, with the values of c_0, c_1, c_2, c_3, specifying the type of boundary conditions.

2. Solve the ODE $\frac{d^2y}{dx^2} + x\frac{dy}{dx} + y = Qx$, in the domain [0 2], with boundary conditions $\left.\frac{dy}{dx}\right|_{x=0} = 0$ and $\left.\frac{dy}{dx}\right|_{x=2} = -1$. Discretise the domain with 50 intervals. Plot *y(x)* as a set of parametric curves for value of Q = -5,0,5.

Task 4: Heat transfer in a nuclear fuel rod

The fuel rod of a nuclear reactor is a cylindrical structure with the fuel contained within a metal cladding. The heat is generated by the nuclear reaction in the fuel region and conducted, through the thickness of the cladding, to the outer surface of the cladding. Outside the cladding cooling occurs with flowing water at $T_w = 473K$ through convective heat transfer (heat transfer coefficient $h = 6 \cdot 10^4 \frac{W}{m^2K}$).

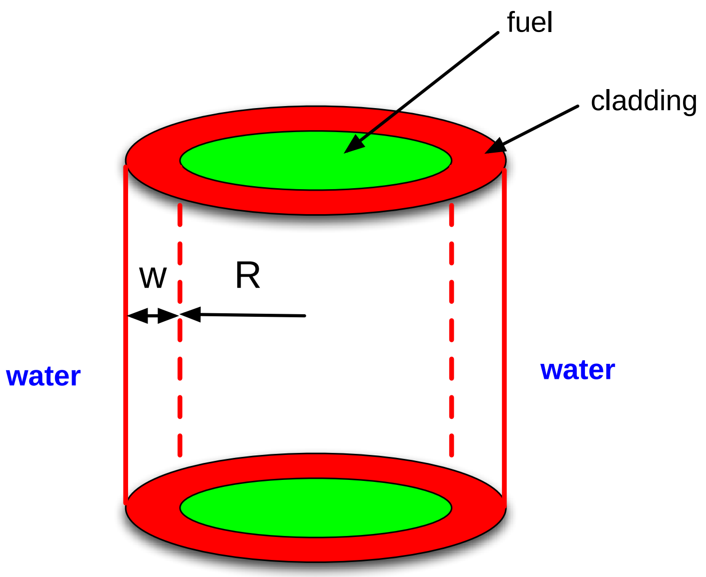
The temperature distribution within the cladding is determined by the ODE:

$$\frac{1}{r}\frac{d}{dr}\left(rk\frac{dT}{dr}\right) = -10^8\frac{e^{-r/R}}{r}$$

in the region of the cladding $R < r < R + w$, with boundary conditions

$$\left.\frac{dT}{dr}\right|_{r=R} = -\frac{6.32 \cdot 10^5}{k} \text{ and } \left.\frac{dT}{dr}\right|_{r=R+w} = -\frac{h}{k}(T_{r=R+w} - T_w).$$

The thermal conductivity of the metal is $k = 16.75 \frac{W}{mK}$. The dimensions of the rod are: $R = 15mm$ and $w = 3mm$.



1. Compute the temperature distribution within the metal cladding, with N = 50.

ME2 Computing- Tutorial 8: Transient analysis of ODEs: Heat Transfer

Learning outcomes:

- Being familiar with the finite difference scheme
- Being familiar with the heat conduction problem
- Being able to assess the stability of the numerical method

Before you start

In your H drive create a folder *H:\ME2CPT\Tutorial8* and work within it.

Introduction

In Tutorial 6 we have solved numerically ODEs with time marching solutions, after providing an initial condition. In Tutorial 7 we have solved steady state (no time varying) ODEs, with solutions restrained at the boundaries.
In this Tutorial, we join both the initial value and the boundary value problems, and will be solving ODEs with time marching solutions and restrains at the boundaries.

A very representative example of this case is the heat conduction within a body, which is described, in one-dimension, by the PDE:

$$\frac{dT}{dt} = \alpha \frac{d^2T}{dx^2}$$

where T is the temperature of the body, α is the thermal diffusivity of the material (in m^2/s) and *t* and *x* are the time and spatial independent variables, respectively.

To solve numerically the ODE we use the backward finite difference scheme for the temporal discretisation and the central difference scheme for the spatial discretisation:

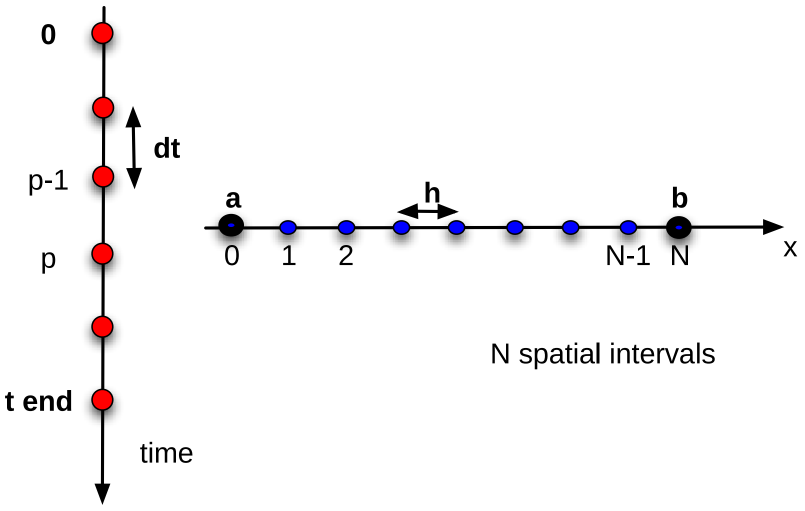
$\frac{dT}{dt} = \frac{T^p - T^{p-1}}{\Delta t}$	$\frac{d^2T}{dx^2} = \frac{T_{i+1} - 2T_i + T_{i-1}}{\Delta x^2}$

Therefore, the ODE is *discretised* as:

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

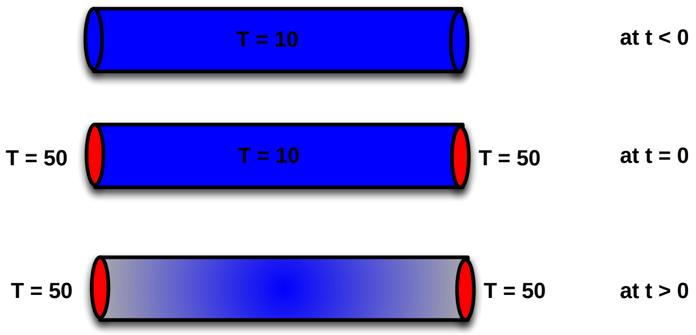
The superscript *p* refers to time, whilst the subscript *i* to space. So T_i^p indicates the Temperature at time step *p* and at node *i* of the domain. I.e., the temperature of spatial location $x = i \cdot \Delta x$ at time $t = p \cdot \Delta t$.
If we know the initial temperature of the body, at every node, we can compute the temperature at later times as:

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



Task A: Heat conduction in a bar

A steel bar (α = 1.172 × 10^{-5} m^2/s) of length 0.5 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.



1. Write a script to solve numerically the heat conduction equation.
   Input data for the problem are: the endpoints of the domain *a* and *b*, the values of the solution at these points, *T(a)* and *T(b)*, the initial uniform temperature T_0 for all spatial points, the time and spatial steps Δt and Δx, respectively, and the desired computational time span t_{end}.
   Output data for the problem are: the grid points *x*, the grid time *t* and the solution *T(x,t)* at every grid points for anytime between 0 and t_{end}.

2. Compute the temperature distribution with Δt = 1s and Δx = 0.01m.
   Plot *T(x,* t_{end}*)*.

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

Task B: Heat conduction in a bar with a heat source

Consider the same bar as in Task A, with an initial uniform temperature of 10°C. The temperature of the middle point is suddenly increased to T = 100°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°C and is subject to convective heat exchange with the water, i.e. $k\left.\frac{dT}{dx}\right|_a = h(T_a - T_w)$ and $k\left.\frac{dT}{dx}\right|_b = -h(T_b - T_w)$. The thermal conductivity for steel is k = 40 $\frac{W}{mK}$, and the heat transfer coefficient h = 500 $\frac{W}{m^2K}$.

1. Amend the script of Task A, to incorporate the mixed boundary conditions.
   A few hints:
   - Set the temperature in the middle point of the grid to be 100°C, irrespectively of the numerical formula, to simulate the source.
   - For the mixed boundary conditions, you need to write down the form of the numerical solution for the first and last points, $T_0^{p+1} = \cdots$, $T_N^{p+1} = \cdots$.

2. Compute the temperature distribution with Δt = 1s and Δx = 0.01m.
   Plot the spatial distribution of the temperature within the bar at *t = 0*, *t = 500s* and t_{end} *= 1200s*.

Task C: Stability of the finite difference numerical method

Solving numerically a time marching ODE with boundary conditions can be easily unstable and careful attention is needed.
Once the time step Δt becomes too large the last term in Equation 1 becomes negative. The same is true if Δx is reduced. When this term is negative the numerical computation is unstable: errors are introduced and amplified at every successive time step. Therefore, in order for the solution to converge it must be:

$\left(1 - \frac{2\alpha\Delta t}{\Delta x^2}\right) > 0 \qquad i.e.$	$\frac{\alpha\Delta t}{\Delta x^2} < \frac{1}{2}$

This is a stability criterion, also known as the Courant condition of stability, and pose a restrain between Δx and Δt. For a chosen Δx there must be a maximum Δt to ensure numerical stability.

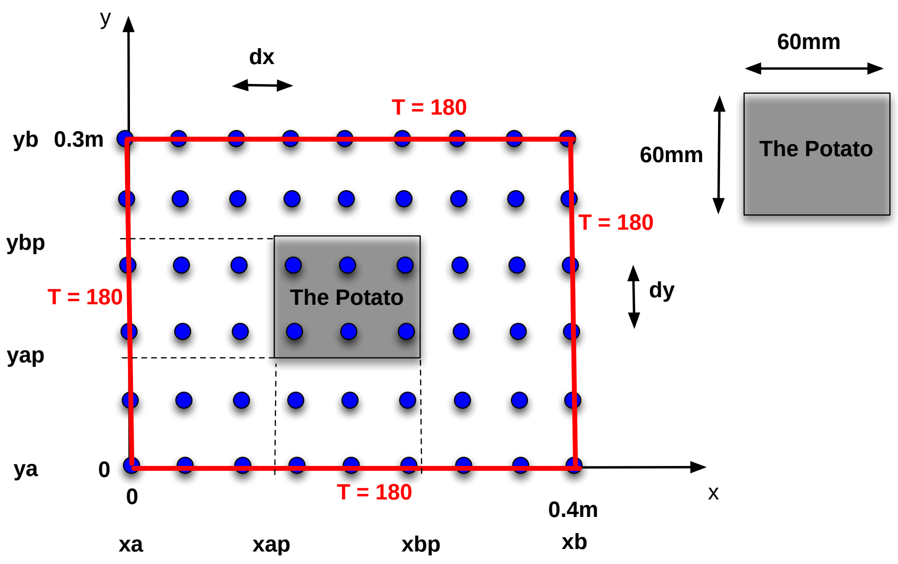
Use the script for Task A, with T(a) = 50 and T(b) = 50.

1. Compute *T(x,* t_{end}*)* with Δt = 1s and Δx = 0.01m, 0.005m and 0.001m. Calculate the Courant condition for all the three cases and plot *T(x,* t_{end}*)*.

2. Repeat the calculation with Δx = 0.001m and a reduced Δt = 0.04s.

Task D: From ODE to PDE: baking a freaking potato in the oven

A (two dimensional) traditional oven can be represented with the discretisation grid:



The heat equation within the oven is described by the Partial Differential Equation:

$$\frac{\partial T}{\partial t} = \alpha \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right)$$

The oven, inside, is initially at room temperature T_0 = 25°C. A frozen potato, with an initial temperature T_0 = -15°C, is positioned centrally in the middle of the oven. The walls of the oven are heat at a constant temperature of T = 180°C.

1. Write a script to solve numerically the heat conduction equation.
   [You need to derive the 2-D discretised equation for the interior points only, as at the boundaries the temperature is given].
   The thermal diffusivity of air in the oven is α = 1.9×10^{-5} m^2/s. The thermal diffusivity of the potato is α = 1.3×10^{-7} m^2/s.

2. Compute T(x,y,t) with $\Delta x = \Delta y = 0.01$m, $\Delta t = 1$s.

3. Plot the 2-D temperature profile at various time steps, to observe how the potato is baking.
   Enjoy the baked potato with a filling of your choice (I like cheese).

ME2 Computing- Tutorial 9: 2D interpolation with unstructured grids

Learning outcomes:

- Being able to interpolate over a triangle with various methods
- Being able to interpolate over triangulated grids
- Being able to refine triangulated mesh

Before you start

In your H drive create a folder *H:\ME2CPT\Tutorial9* and work within it.

Introduction

You need to refer to the slides and the video to be able to interpret what requested in these tasks.

Task A: Interpolation over a triangle: nearest neighbour method

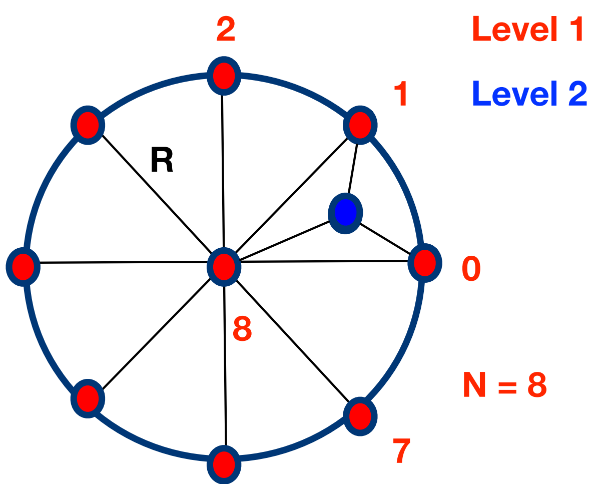
Write a function *TrNN*, to interpolate three points on a plane with the nearest neighbour methods. The function receives the coordinates of three points (r_1, r_2, r_3) and the values of the mathematical function at these three points (f_1, f_2, f_3), and the coordinates of a fourth point r_p, coplanar with the first three ones, and returns the interpolated value $f\left(r_p\right)$.

Task B: Interpolation over a triangle: barycentric coordinates

Write a function *TrBaryc*, to interpolate three points on a plane with the barycentric coordinates method. The function has same input and output arguments as in Task A.

Task C: Interpolation over a triangulated mesh and mesh refinement

1. Preparatory work, to generate a mesh with multiple triangles:
   Consider a circle of radius *R* and subdivide its perimeter in *N* points. Generate a triangulated mesh grid with *N+1* nodes, as depicted in the figure (red nodes):



   Assign at each node i the numerical values:

$$f[i] = i\frac{360}{N} \qquad i = 0,1,2,\ldots,N$$

   (You can assign any value that you prefer, this is only a suggestion, as we need to assign some values at nodal points).

2. For each triangle:

   i) establish a new node at the centroid of the triangle.
   Given three vertices, r_0, r_1, r_2, the centroid of a triangle is defined as a node r_p, with coordinates:

$$x_p = \frac{x_0 + x_1 + x_2}{3} \qquad\qquad y_p = \frac{y_0 + y_1 + y_2}{3}$$

   (you can choose to create any node you prefer inside the triangle, i.e., take the incenter or the circumcenter or the orthocentre, etc.)

   ii) interpolate the value of f at this new node r_p.

   Repeat the processes i) and ii) for a number of levels *L*.

ME2 Computing- Tutorial 10: Root finding

Learning outcomes:

- Being able to find roots of a function through bisection and Newton-Raphson methods
- Being able to solve numerically systems of non-linear equations

Before you start

In your H drive create a folder *H:\ME2CPT\Tutorial10* and work within it.

Task A: Bisection method

Write a Python function *mybisection*, to determine the root of the equation $f(x) = 0$, within a given interval $[a, b]$ and a specified accuracy ε, using the bisection method. $f(x)$ is a known function and might be implemented as a separate Python function.

Test your code by finding the root of the equation:

$$f(x) = x^2 + (x-2)^3 - 4 = 0$$

Find and compare the root with accuracies ε = 0.1, ε = 0.01 and ε = 0.001

Task B: Bisection method with a discrete function $f(x_n)$

Often the function $f(x)$ is available only in a discrete form, as a set of values $f(x_n)$, for a finite number of points. In such cases the evaluation of the function at any x, as requested by the bisection method, can be obtained through interpolation. Use a copy of the code in Task A and amend it to find roots from a discrete set of values.

Test your code with the discrete set of values stored in files *x.txt, fx.txt*, containing values of the independent variable x and the function $f(x)$, respectively.

Task C: Newton-Raphson method

Write a Python function *myNewton*, to determine the root of the equation $f(x) = 0$, given an initial guess x_0 and a specified accuracy ε, using the Newton-Raphson method. $f(x)$ is a known function and might be implemented as a separate Python function.

Task D: Newton-Raphson method for systems of non-linear equations

Write a Python function *mySystem*, to determine the solutions of system of non-linear equations, using the Newton-Raphson method.