Exercise 1. Spline interpolation is often preferred over polynomial interpolation because the interpolation error can be made small even when using low degree polynomials for the spline. Spline interpolation avoids the problem of Runge's phenomenon, in which oscillation can occur between points when using high degree polynomials. Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called a spline. Most popular are cubic splines, which are known to minimise the bending under the constraint that passes through all knots.

To illustrate this, select 13 points on the well-known serpentine curve given by

$$y = \frac{x}{1/4 + x^2}$$

So that the knots will not be equally spaced, write the curve in parametric form

$$x = \frac{1}{2}\tan\theta, \qquad y = \sin 2\theta.$$

We want to restrict the x-knots within an interval, $\left[-a, a\right]$, with a reasonable value, let's say $a = 3$. To this end, take the θ-knots in the form

$$\theta_i = \frac{i}{6}\arctan(2a), \qquad i = -6,-5,\ldots,5,6$$

Then, x- and y-knots are given by

$$x_i = \frac{1}{2}\tan\theta_i, \qquad y_i = \sin 2\theta_i.$$

This form guarantees that $-a \le x_i \le a$

a) Plot the serpentine curve within the limits, $-4 \le x \le 4$, together with the knots, $\left(x_i, y_i\right)$ and label them as o.
b) Using the MATLAB built-in function, `spline`, or the Python built-in function, `CubicSpline`, plot not-a-knot cubic spline also within limit, $-4 \le x \le 4$.
c) Plot the interpolation polynomial of a suitable degree in order to compare the curves. It is convenient to restrict the plot in the y-axis as $-8 \le y \le 8$.

Exercise 2. Use cubic spline functions to produce the curve for the following data:

x	0	1	2	3	4	5	6	7
y	1.0	1.5	1.6	1.5	0.9	2.2	2.8	3.1

It is known that the curve is continuous, but its slope is not. **Hint:** find where the slope is discontinuous and plot the curve with two spline functions.

Exercise 3. Do there exist a, b, c and d such that the function

$$S(x) = \begin{cases} ax^3 + x^2 + cx & \left(-1 \le x \le 0\right) \\ bx^3 + x^2 + dx & \left(0 \le x \le 1\right) \end{cases}$$

is a natural cubic spline function that agrees with absolute value function $\left|x\right|$ at the knots -1, 0, 1? Plot this function in case it does exist.

Exercise 4. Parametrisation in mathematics and particularly in geometry is a very useful concept. Generally, it is a process of finding parametric equations of a curve, surface etc. To parameterise by itself means to express in terms of parameters. For

instance, function parametrisation consists of substituting a given function with a pair of functions

$$y = f(x) \in C^1[a,b] \longrightarrow \begin{cases} x = g(t) \in C^1[\alpha,\beta] \\ y = f(g(t)) \end{cases}$$

Hence we treat the variable x as a dependent variable thus equating its role to that of y. In this case, the resulting plot y vs. x can wrap on itself. Very often, the problem in question suggests a suitable choice of $g(t)$. However, if we don't know much about the underlying problem, we can opt for the so called "natural" parametrisation where the parameter t is the arc length along the given curve.

Similar ideas may be applied in spline interpolation when the spline function does not respect the monotonicity property of the original data or it does not preserve geometric invariance etc. Based on the natural parametrisation concept, a very simple method of building a parametric spline interpolation is described below. We begin with initial data

$$(x_0, y_0), (x_1, y_1), \cdots, (x_n, y_n)$$

It defines a broken straight line that connects data knots in 2D. Each line segment is of length

$$l_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \qquad i = 1,2,\cdots,n$$

Next, we define a finite sequence (independent discrete variable/parameter), $\{s_k\}$

$$s_0 = 0, \quad s_k = \sum_{i=1}^{k} l_i \quad k = 1,2,\cdots,n$$

Finally, we build 2 cubic splines based on the knots, (s_i, x_i) and (s_i, y_i) thus treating x and y as dependent on s.

Another example, where parametric splines naturally appear, is parametric curves, i.e. the curves that are originally given in parametric form as in the example below.

Archimedean spiral in polar coordinates is given by:

$$r = a + b \cdot \theta$$

Reasonable parameters for plotting would be $a = 1$, $b = 1$, and $0 \le \theta \le 2 \cdot 2\pi$ that gives a spiral with 2 full revolutions. The built-in MATLAB/Python function `plot` requires Cartesian coordinates

$$x = r\cos\theta, \qquad y = r\sin\theta$$

Draw the spiral and reproduce it by way of parametric spline functions. To this end, select and plot n knots, (x_i, y_i), on the Archimedean spiral

$$\theta_i = (i-1)\frac{4\pi}{n-1} \qquad (i = 1,2,\ldots,n)$$

The built-in MATLAB/Python function `linspace` could be helpful here. Repeat your calculations for $n = 5$, 6, 8 and 12. Organise your drawings as a 2×2 matrix using the built-in MATLAB/Python function `subplot`.

Exercise 5. Describe explicitly the natural cubic spline that interpolates a table with only two entries:

x	x_0	x_1
y	y_0	y_1

Exercise 6. Derive the approximation formula

$$f'(x) \approx \frac{4f(x+h) - 3f(x) - f(x+2h)}{2h}$$

and show that its error term is of the form $\frac{1}{3}h^2 f'''(\xi)$

Exercise 7. Criticise the following analysis. By Taylor's formula, we have

$$f(x+h) - f(x) = hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi)$$

$$f(x-h) - f(x) = -hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi)$$

So, by adding, we obtain an *exact* expression for $f''(x)$

$$f(x+h) + f(x-h) - 2f(x) = h^2 f''(x)$$

Exercise 8. The values of a function *f* are given at three points x_0, x_1, and x_2. If a quadratic interpolating polynomial is used to estimate $f'(x)$ at $\left(x_0 + x_1\right)/2$, what formula will result?

Exercise 9. Show how Richardson extrapolation would work on the formula

$$y''\left(x\right) \approx \frac{y\left(x+h\right) - 2y\left(x\right) + y\left(x-h\right)}{h^2}$$

Exercise 10. Approximate the derivative of $f(x) = e^x$ at $x = 0$ by the forward finite difference

$$f'(x) \approx \nabla_h^+ f(x,h) \equiv \frac{f(x+h) - f(x)}{h}$$

By definition, the approximation error is

$$Err(x,h) \equiv \nabla_h^+ f(x,h) - f'(x)$$

Assuming exact arithmetic, a simple error analysis gives

$$Err(x,h) = \frac{1}{2}f''(x)h + O(h^2)$$

However, in computer (finite) arithmetic, we have $f^{\mathrm{M}}(x)$ instead of $f(x)$ where the superscript M symbolizes the machine representation of the exact value $f(x)$ that is in reality, it holds

$$\nabla_h^+ f(x,h) = \frac{f^{\mathrm{M}}(x+h) - f^{\mathrm{M}}(x)}{h}$$

We know that

$$f^{\mathrm{M}}(x+h) = f(x+h)\cdot\left(1+\delta_1\right) \text{ and } f^{\mathrm{M}}(x) = f(x)\cdot\left(1+\delta_2\right)$$

Assuming the (default) rounding to the nearest guarantees

$$\left|\delta_{1,2}\right| \le \varepsilon_M / 2$$

(a) Show the following representation in computer arithmetic

$$Err(x,h) = \frac{f''(x)}{2}h + O\left(h^2\right) + \frac{f(x+h)\delta_1 - f(x)\delta_2}{h}.$$

(b) Ignore $O(h^2)$, assume that $f(x+h) \approx f(x)$ for a sufficiently small h and derive the upper bound for the approximation error (using $|a-b| \le |a| + |b|$):

$$\left|Err(x,h)\right| \le \frac{h}{2}\left|f''(x)\right| + \frac{\varepsilon_M}{h}\left|f(x)\right| \equiv E(h).$$

(c) Find an "optimal" discretization step h_{opt} that minimizes the error bound $E(h)$.

(d) Write a computer code that calculates the exact approximation error $Err(x,h)$ for $f(x) = e^x$ at $x = 0$ using the discretization steps 10^{-1} through 10^{-16} and compare your results with h_{opt}.