PRACTICE 4

1 Divided Differences - Newton's Formula

In the context of interpolation, the divided differences are used to construct a polynomial that passes through a given set of points. The basic idea is to compute the differences between function values at different points, recursively.

Given the set of points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, a set of divided differences is defined for the values of the function $f(x)$ at the points $x_0, x_1, \ldots, x_n$, with $f(x_k) = y_k$. The recursive formula for the divided differences is:

$$f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{y_1 - y_0}{x_1 - x_0}$$

- **First-order divided difference** - similar to the slope between two points:

$$f[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$$

- **Second-order divided difference**

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

where:

$$f[x_1, x_2] = \frac{y_2 - y_1}{x_2 - x_1}$$

- **General form for higher-order differences**

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0}$$

These higher-order differences are calculated recursively, building up the coefficients for the Newton interpolating polynomial.

1.0.1 Newton's Interpolating Polynomial

The **Newton interpolation polynomial** $P(x)$ is defined incrementally using the divided differences. Given a set of data points $\{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)\}$, the polynomial can be written as:

$$P(x) = y_0 + (x - x_0)f[x_0] + (x - x_0)(x - x_1)f[x_0, x_1] + \cdots + (x - x_0)(x - x_1)\cdots(x - x_{n-1})f[x_0, x_1, \ldots, x_n]$$

where:

$$f[x_0] = y_0, \quad f[x_0, x_1] \text{ is the first divided difference,}$$

$$\ldots$$

$$f[x_0, x_1, \dots, x_n] = \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0}$$

The **coefficients** of the polynomial $P(x)$, denoted as a_k, are the **divided differences** of order k:

$$a_0 = f[x_0], \quad a_1 = f[x_0, x_1], \quad a_2 = f[x_0, x_1, x_2], \dots, a_n = f[x_0, x_1, \dots, x_n]$$

Thus, the Newton interpolation polynomial can also be written as:

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

This form makes it easy to add more points because the polynomial can be updated by just computing new divided differences without needing to recompute the entire polynomial.

Although the Lagrange form is algebraically different, both methods (Lagrange and Newton) produce the same interpolating polynomial. However, in the Newton method, the process is more convenient for adding new interpolation points since it only requires recalculating the divided differences without having to redo the entire polynomial.

Example 1.1 *Write a script that computes the interpolant polynomial using the divided differences. Create two functions* `divdef.m` *with*

Input: the coordinate coordinates of the nodes x and y. Output: a matrix that contains the divide differences.

and a function `newtonpolynomial.m` *with:*

Input: the coefficients and the coordinates x of the nodes. Output: the value of the polynomial in the point (or array or points) z.

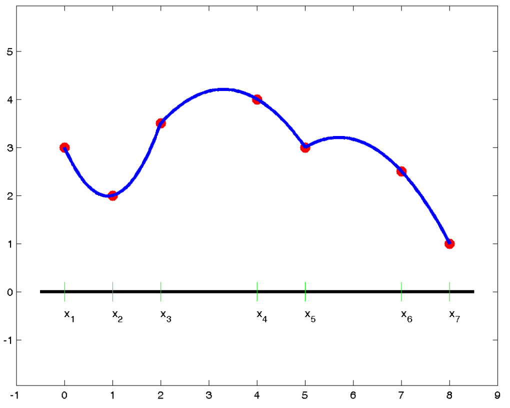
Use the nodes: $\{(-1, 1), (0, 3), (2, 4), (3, 3), (5, 1)\}$

2 Piecewise Polynomial Interpolation - Splines

Polynomial interpolation for a set of $n + 1$ points is frequently unsatisfactory. A polynomial of degree n can have $n - 1$ maxima and minima and the graph can wiggle in order to pass through the points. Polynomial interpolation may cause oscillations for large datasets. Splines offer a piecewise approach that ensures smooth transitions.

The piecewise polynomial interpolation is based on performing polynomial interpolation in *pieces*, rather than on the entire given interval, so interpolate between successive nodes (x_k, y_k) and (x_{k+1}, y_{k+1}) in the interval $[a, b]$:

$$a = x_1 < x_2 < \cdots < x_{n+1} = b.$$



The two adjacent portions of the curve $y = S_k(x)$ and $y = S_{k+1}(x)$ which lie above $[x_k, x_{k+1}]$ and $[x_{k+1}, x_{k+2}]$, respectively, pass through the common knot (x_{k+1}, y_{k+1}). The two portions of the graph are tied together at the knot and the set of functions $\{S_k(x)\}$ (where $S_k(x)$ is a polynomial of degree m) forms a piecewise polynomial curve, which is denoted by $S(x)$.

2.1 Linear Splines

A linear spline is a piecewise linear function that interpolates the data points. It is defined as:

$$S_i(x) = a_i + b_i(x - x_i), \quad x \in [x_i, x_{i+1}]. \tag{1}$$

Example 2.1 *Find out a linear spline interpolation using the following set of points:*

$$(x_0, y_0) = (0, 1), \quad (x_1, y_1) = (1, 3), \quad (x_2, y_2) = (2, 2)$$

Solution:

1. Define the Linear Spline Segments

   For each interval $[x_i, x_{i+1}]$, the linear spline is defined by the equation of the line connecting the two points:

$$S_i(x) = y_i + m_i(x - x_i), \quad \text{where} \quad m_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$

2. Compute the Slopes

   – For $[0, 1]$:

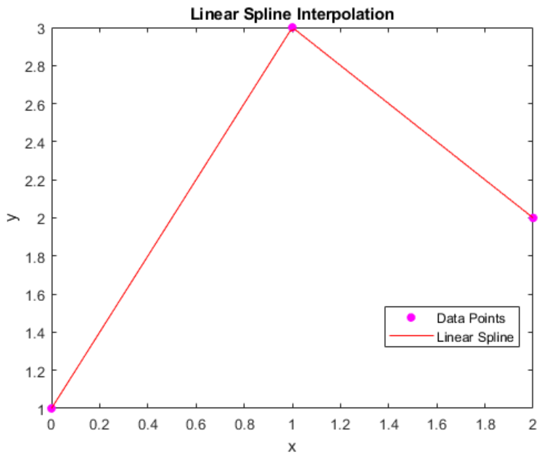
$$m_0 = \frac{3-1}{1-0} = 2 \quad \Longrightarrow \quad S_0(x) = 1 + 2(x - 0) = 1 + 2x$$

   – For $[1, 2]$:

$$m_1 = \frac{2-3}{2-1} = -1 \quad \Longrightarrow \quad S_1(x) = 3 - 1(x - 1) = 4 - x$$

3. Piecewise Function - the linear spline $S(x)$ is:

$$S(x) = \begin{cases} 1+2x, & \text{for } 0 \le x \le 1 \\ 4-x, & \text{for } 1 < x \le 2 \end{cases}$$



Matlab has the following commands to obtain that:

`interp1(x,y,xx)`: returns interpolated values of a 1-D function at specific query points using linear interpolation. Vector `x` contains the sample points, and `y` contains the corresponding values, `y(x)`. Vector `xx` contains the coordinates of the query points.

2.2 Cubic Splines

A cubic spline is a piecewise polynomial of degree 3 that satisfies continuity conditions for the function and its first and second derivatives.

The general form of a cubic spline is:

$$S_i(x) = a_{0i} + a_{1i}(x-x_i) + a_{2i}(x-x_i)^2 + a_{3i}(x-x_i)^3, \quad x \in [x_i, x_{i+1}]. \tag{2}$$

Matlab has the following commands to obtain that:

`spline(x,y,xx)`: uses a cubic spline interpolation to find `yy`, the values of the underlying function `y` at the values of the interpolant `xx`.

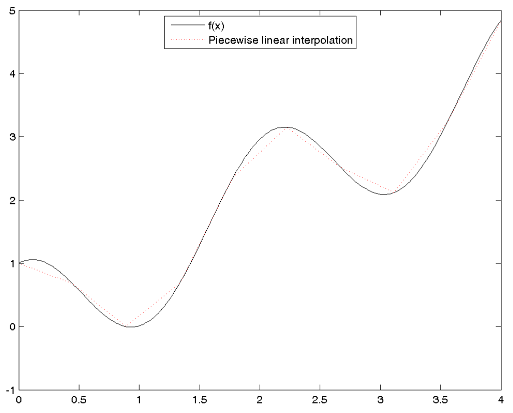
Example 2.2 *Given the function $f(x) = x + \cos 3x$, generate a row vector x of 10 points linearly spaced between and including 0 and 4 and plot, at the interval $[0,4]$, the function $f(x)$, the points $(x_i, f(x_i))$ and the piecewise linear interpolation. Find the absolute errors of the approximation at the points $x = 1$, $x = 2$ and $x = 3$.*

Solution: We generate the function and the data:

```
>> f=@(x) x+cos(3*x)
>> x=linspace(0,4,10)
>> y=f(x)
```

and plot the graphs:

```
>> fplot(f,[0 4],'k')
>> hold on, plot(x,y,'r:')
>> legend('f(x)','Piecewise linear interpolation','location','north')
```



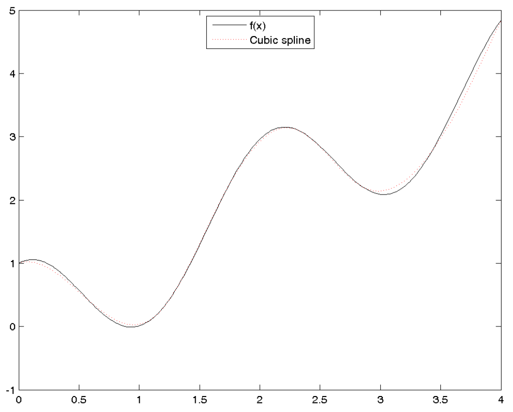
Finally, we find the absolute errors:

```
>> absolute_errors=abs(interp1(x,y,1:3)-f(1:3))
```

Example 2.3 *Repeat the previous exercise, dividing the interval in* 8 *points linearly spaced and and use the cubic spline interpolation to approximate the function.*

Solution:

```
>> x=linspace(0,4,8), y=f(x)
>> fplot(f,[0 4],'k')
>> hold on , fplot(@(t) spline(x,y,t),[0 4],'r:')
>> legend('f(x)','Cubic spline','location','north')
```



```
>> spline_errors=abs(spline(x,y,1:3)-f(1:3))
```