PRACTICE 3

1 Interpolation

1.1 Introduction

Interpolation is a method for finding a function $f(x)$ that passes exactly through a given set of points (x_i, y_i), where $i = 0, 1, \ldots, n$. Unlike **fitting methods**, which aim to minimize the error concerning the given data, interpolation ensures an exact match at the given points.

Given a set of data points:

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n), \tag{1}$$

the goal is to find a function $P(x)$ such that:

$$P(x_i) = y_i, \quad \forall i = 0, 1, \ldots, n. \tag{2}$$

The interpolating function $P(x)$ can be constructed using different approaches depending on the choice of basis functions and the degree of the interpolating polynomial.

1.2 General Formulation of Interpolation

In general terms, the interpolation problem can be formulated within the framework of a **functional approximation model**, where the interpolating function is constructed as a linear combination of basis functions $\phi_k(x)$:

$$P(x) = \sum_{k=1}^{n} a_k \phi_k(x), \tag{3}$$

where:

- $\{a_1, a_2, \ldots, a_n\}$ are the **unknown parameters** to be determined.
- $\{\phi_1(x), \phi_2(x), \ldots, \phi_n(x)\}$ are the **basis functions** that determine the shape of the interpolant.

To ensure that $P(x)$ passes exactly through the given data points, the following system of equations must be satisfied:

$$P(x_i) = \sum_{k=1}^{n} a_k \phi_k(x_i) = y_i, \quad \forall i = 0, 1, \ldots, n. \tag{4}$$

Different interpolation methods use different choices for the basis functions $\phi_k(x)$, such as:

- **Polynomial interpolation**: $\phi_k(x) = x^{k-1}$.
- **Trigonometric interpolation**: $\phi_k(x) = \cos(kx), \sin(kx)$ (useful for periodic data).

- **Spline interpolation**: Piecewise polynomials ensuring smoothness between segments.

The choice of method depends on the nature of the data and the required smoothness and numerical stability.

- If the data is smooth and without oscillations, a low-degree polynomial is sufficient.
- If the data is periodic, trigonometric interpolation is the best option.
- If the data is irregular or has abrupt changes, splines are more appropriate.
- If better numerical stability is required, orthogonal polynomials (Chebyshev, Legendre) can be used.

2 Polynomial Interpolation

2.1 Lagrange Interpolation

Given a set of $n+1$ data points

$$\{(x_1, y_1), (x_2, y_2), \ldots, (x_{n+1}, y_{n+1})\}$$

with $x_i \neq x_j$, for $i \neq j$, there exists a unique polynomial of degree $\leq n$ that passes through these $n+1$ points, we call it ***the interpolating polynomial***.

The Lagrange interpolating polynomial is defined as:

$$P(x) = \sum_{i=0}^{n} y_i L_i(x), \tag{5}$$

where $L_i(x)$ are the Lagrange basis polynomials given by:

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{6}$$

Polynomials in MATLAB: The polynomial $p(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ can be defined in Matlab through the a vector of it's coefficients:

$$\texttt{p=[a}_\texttt{n}\ \texttt{a}_{\texttt{n}-1}\ \cdots\ \texttt{a}_\texttt{1}\ \texttt{a}_\texttt{0}\texttt{]}$$

- `polyval(p,t)`: returns the value of a polynomial of degree n evaluated at `t`. The input argument `p` is a vector of length $n+1$ whose elements are the coefficients in descending powers of the polynomial
- `polyder(p)`: returns the derivative of the polynomial `p`.
- `conv(p,q)`: returns a row vector whose elements are the coefficients of the polynomial `p`·`q` (polynomial multiplication).
- `roots(p)`: returns a column vector whose elements are the roots of the polynomial `p`.

- `poly([r1 r2 ··· rn])`: returns a row vector whose elements are the coefficients of the polynomial `(x-r1)(x-r2)···(x-rn)`.

Apply these commands to the polynomial $x^2 + x + 2$.

Now programme a Matlab function which returns the lagrange basis polynomials.

```
function L=lagrange(x)
%     Function L=lagrange(x) which constructs the lagrange basis polynomials
%     defined by the components of the vector x
% INPUT ARGUMENT:
%   x ..... a vector with the points which define the polynomials
% OUTPUT ARGUMENT:
%   L ..... a matrix whose rows are the Lagrange basis polynomials l_j(x)
m=length(x); L=zeros(m);
for j=1:m
r=x([1:j-1 j+1:m]); l=poly(r)/prod(x(j)-r);
L(j,:)=l;
end
```

therefore, once we obtain the matrix with the Lagrange basis polynomials as

$$\begin{pmatrix} L_1(x) \\ L_2(x) \\ \vdots \\ L_{n+1}(x) \end{pmatrix}$$

to obtain the interpolating polynomial, it suffices to multiply these two matrices:

$$(y_1\ y_2\ \ldots\ y_{n+1}) \begin{pmatrix} L_1(x) \\ L_2(x) \\ \vdots \\ L_{n+1}(x) \end{pmatrix} = y_1L_1(x) + y_2L_2(x) + \cdots + y_{n+1}L_{n+1}(x)$$

Example 2.1 *Given the set of data points,*

x	0	1	2	3	4	5
y	1.1	1.5	2.4	2	3	1

find the interpolating polynomial and plot the graph of this polynomial and the interpolation data points.

Solution: we define the interpolation data points as the vectors `x` and `y`, and we apply the previous function `lagrange` for the input `x`.

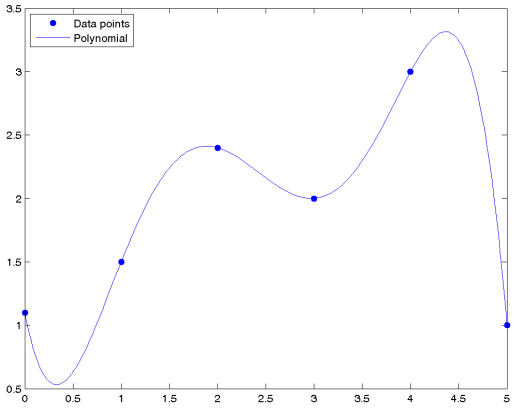
```
>> x=0:5, y=[1.1 1.5 2.4 2 3 1]

>> L=lagrange(x)

>> p=y*L % p is the interpolating polynomial
```

and, finally, we plot the graphs

```
>> plot(x,y,'.','markersize',20);
>> hold on; fplot(@(x) polyval(p,x),[0 5]);
>> legend('Data points','Polynomial','location','northwest');
```



Example 2.2 *Write a script that computes Lagrange's fundamental polynomials. Create a function:*

`lagrange_fundamental(k, x, z)`

- `Input:`
  - k: *Index of the fundamental polynomial.*
  - x: *The x-coordinates of the nodes (or points the polynomial passes through).*
  - z: Point (or array of points) where we will evaluate the polynomial.
- **Output:** *The value of the k-th fundamental polynomial at the point z.*

Plot the Lagrange's Fundamental Polynomials. Then implement a script that computes and plots the fundamental polynomials of Lagrange. Use the following set of nodes to test the created function:

$$x = \texttt{[-1., 0, 2, 3, 5]}$$

2.1.1 Errors in Polynomial Interpolation

In many cases, the data y_i are the values of a function $f(x)$ that we want to approximate ($y_i = f(x_i)$). Obviously, if we approximate $f(x)$ with $p(x)$, we get the error $E(x) = f(x) - p(x)$. So, if $f \in \mathcal{C}^{n+1}[a,b]$ and $x_1, x_2, \ldots, x_{n+1} \in [a,b]$, for any $x \in [a,b]$, there exists $c \in [a,b]$, such that:

$$E(x) = \frac{f^{(n+1)}(c)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) \tag{7}$$

Example 2.3 *Approximate the function* $f(x) = x\sin(x)$ *at the interval* $[0, 3]$*, as follows: define a vector of* 5 *elements,* x_i*, linearly spaced at the previous interval. Find the interpolating polynomial* $p(x)$ *defined by the data interpolating points* $(x_i, f(x_i))$*. Then, bound the absolute error we get when we approximate* $f(x)$ *with* $p(x)$ *and plot the graphs of the function, the interpolating polynomial and the interpolation data points at the interval* $[0, 3]$*. Observe the extrapolation problem plotting the graphs at the interval* $[0, 4]$*.*

Solution: we define $f(x)$ and find the interpolating polynomial

```
>> f=@(x) x.*sin(x)

>> x=linspace(0,3,5), y=f(x)

>> L=lagrange(x)

>> p=y*L
```

and, if we approximate $f(x)$ with $p(x)$, we get the error

$$E(x) = \frac{1}{5!}\prod_{i=1}^{5}(x - x_i)\, f^{(v)}(c) = \frac{1}{5!}q(x)f^{(v)}(c), \text{ with } q(x) = \prod_{i=1}^{5}(x - x_i)$$

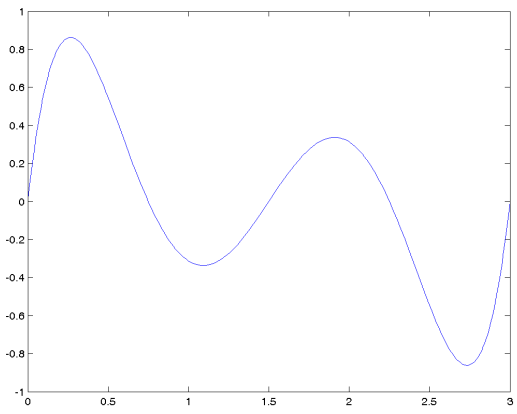
therefore, a bound of the error may be $\frac{1}{5!}K \cdot M$, being:

$$K = \max\{|q(x)| : \ 0 \le x \le 3\}, \ M = \max\{|f^{(v)}(x)| : \ 0 \le x \le 3\}$$

To find K, we define the polynomial $q(x)$ and we plot its graph

```
>> q=poly(x)

>> fplot(@(x) polyval(q,x),[0 3])
```



Then, we find the relative maxima and minima of $q(x)$. For that, we need the points such that $q'(x) = 0$, that is

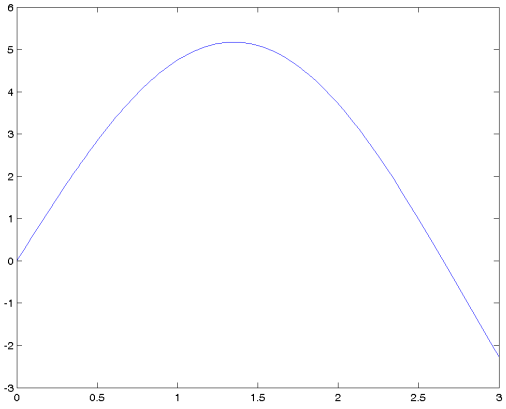
```
>> r=roots(polyder(q))
```

and, thus, the maximum of $|q(x)|$ at the interval $[0, 3]$, is:

```
>> K=max(abs(polyval(q,r)))
```

To calculate the maximum of the function $|f^{(v)}(x)|$, we denote $f_5(x) = f^{(v)}(x)$ and we plot it at the interval $[0, 3]$

```
>> syms x, f_sym(x)=f(x)
>> f5=diff(f_sym,5)
>> fplot(matlabFunction(f5),[0 3])
```



and, to obtain the relative maximum, we observe in the figure that we must find the root of $f_5'(x) = 0$. We can do it using the command **fzero**

```
>> c=fzero(matlabFunction(diff(f5)),1)
```

then, the maximum of the function $|f^{(v)}(x)|$ is

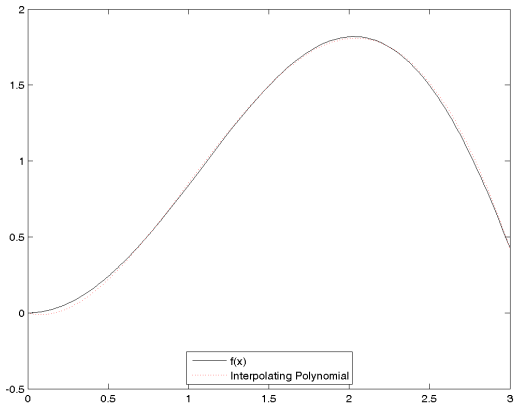
```
>> M=double(abs(f5(c)))
```

and a bound of the absolute error is

```
>> error_bound=K*M/factorial(5)
```

so, we can consider that the approximation is reasonably good, as we can see below:

```
>> fplot(f,[0 3],'k')
>> hold on, fplot(@(x) polyval(p,x),[0 3],'r:')
>> legend('f(x)','Interpolating Polynomial','location','south')
```



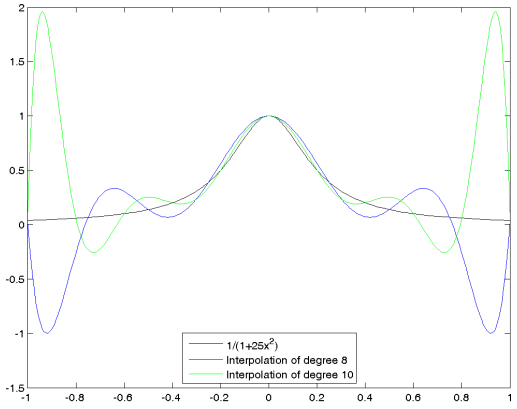
Limitations of Lagrange Interpolation:

One of the main limitation is Runge's phenomenon, which refers to the large oscillations that appear near the edges of the interval, when increasing the degree of the interpolation polynomial. In fact, higher-degree polynomials tend to exhibit even more severe oscillations.

Although it seems reasonable to construct the interpolating polynomial from equally spaced nodes, it is not always the best solution for obtaining good approximations. For example, if we try to approximate the function

$$f(x) = \frac{1}{1 + 25x^2}$$

at the interval $[-1, 1]$ using interpolating polynomials of degrees 8 and 10, it results



and it is observed that, at the endpoints of the interval, the approximation gets worse when the number of nodes increases.

2.1.2 Chebyshev Nodes to Minimize Error

To reduce the effects of Runge's phenomenon, Chebyshev nodes are used instead of equally spaced nodes. These nodes are given by:

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{(2i+1)\pi}{2(n+1)}\right), \quad i = 0, 1, \ldots, n. \tag{8}$$

Chebyshev nodes cluster more densely near the endpoints, which helps control large oscillations and improves numerical stability in polynomial interpolation.

So, in case we use n points, at the interval $[-1, 1]$, it is better to use the Chebyshev points defined as follows

$$x_i^{(n)} = \cos\frac{(2i-1)\pi}{2n}\ ,\ i = 1, 2, \ldots, n$$

For a general interval $[a, b]$ we apply the affine transformation that maps $[-1, 1]$ onto $[a, b]$ to shift and scale the Chebyshev points.

$$x = \frac{a+b}{2} + \frac{b-a}{2}t$$

we redefine the interpolation abscissae as

$$x_i^{(n)} = \frac{a+b}{2} + \frac{b-a}{2}\cos\frac{(2i-1)\pi}{2n}\ ,\ i = 1, 2, \ldots, n$$

Let us apply this to the previous function

$$f(x) = \frac{1}{1+25x^2}$$

at the interval $[-1, 1]$ using interpolating polynomials of degrees 8 and 10 with the Chebyshev points and plot the graphs of the function.

Solution: Given that we have to use 9 and 11 Chebyshev points, respectively, we proceed as follows

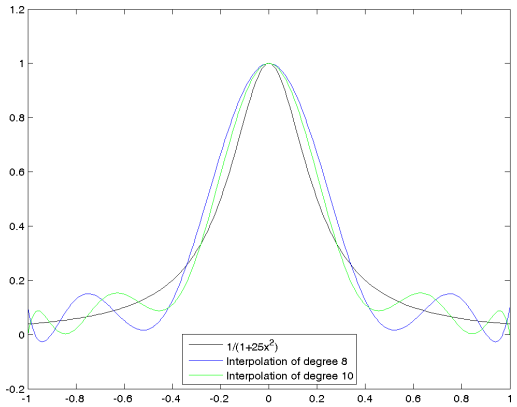
```
>> n=9; i=1:n; x=cos((2*i-1)*pi/(2*n)), y=f(x)

>> L=lagrange(x); p8=y*L

>> n=11; i=1:n; x=cos((2*i-1)*pi/(2*n)), y=f(x)

>> L=lagrange(x); p10=y*L

>> fplot(f,[-1 1],'k')
>> hold on, fplot(@(x) polyval(p8,x),[-1 1],'b')
>> fplot(@(x) polyval(p10,x),[-1 1],'g')
>> legend('1/(1+25x^2)','Interpolation of degree 8',...
'Interpolation of degree 10','location','south')
```



Check that the interpolation also works with 15 nodes and the function:

$$f_2(x) = e^{-20x^2}$$

Follow the steps:

1. **Build the Nodes** The n nodes will be stored in an n-element vector x in both cases. For the equispaced nodes, the functions `np.linspace` or `np.arange` may be used. The corresponding y values are calculated using f_1 and f_2.

2. **Build the Interpolant Polynomials** To interpolate the functions, use `pol.polyfit` and `pol.polyval`.

3. **Plot the Graphs** In both cases, plot the nodes, the function, and the interpolant polynomial in the interval `[-1.5,1.5]`.
