Daniel Herreros| 540002818
David Miranda | 540001818
Nestor Uriarte| 540000817
MAT300-SP23
Project 1

Interpolation

Octave Project 1

Description of the problem:

This project's goal is to compute polynomial curves which traverse all the points given as input. The user will supply the points to interpolate, their dimension, the type of mesh the program must compute, the method it will use to resolve the interpolation problem and how many output nodes should be interpolated.

Explanation of the methods:

In this section all three different methos used to compute the curves will be explained; Gauss-Jordan, Lagrange, and Newton.

Gauss-Jordan

Gauss-Jordan method takes advantage of the Vandermonde's matrix properties. It forms a Vandermonde basis for P_n based on the given points and a parameter t. The process is as follows:

1. Define a parametric curve where x, y and z are polynomials as such: $\gamma(t) = (p(t), q(t)$. Each polynomial will define a coordinate of the final curve.
2. After that, take a mesh a = t0 < t1 < t2 < t3 < t4 < … < tn = b
3. Build a system of equations with each of the given points, convert it to matrix form and solve it using the Reduced Row Echelon Form. This will give the coefficients of the interpolant polynomial respect to t which passes through all the given points. Knowing that all the points are computed with respect to the same t, we can compute a matrix M and augment it to compute the solution for all the coordinates at the same time.

```
function c = computeCoefficients(PX, PY, PZ, t, n, dimension)

    matrix = zeros(n, n + dimension);

    % Fill the matrix with the given points
    % computing the rows and columns based on the slides
    for i = 1 : n
      for j = 1 : n
        matrix(i,j) = t(i)^(n-j);
      endfor
    endfor

    % Add the augmented matrix's column
    matrix(:,n+1) = PX';
    matrix(:,n+2) = PY';

    if(dimension > 2)
      matrix(:,n+3) = PZ';
    endif

    % Compute the result and extract it
    matrix = rref(matrix);

    c = zeros(n, dimension);
    c(:,1) = matrix(:,n+1);
    c(:,2) = matrix(:,n+2);

    if(dimension > 2)
      c(:,3) = matrix(:,n+3);
    endif

endfunction
```

$$\begin{cases} a_0 + a_1 t_0 + a_2 t_0^2 \cdots a_n t_0^n \\ a_0 + a_1 t_1 + a_2 t_1^2 \cdots a_n t_1^n \\ \vdots \\ a_0 + a_1 t_n + a_2 t_n^2 \cdots a_n t_n^n \end{cases} \rightarrow \left[\begin{array}{ccccccc} 1 & t_0 & t_0^2 & \cdots & t_0^n & p_0 & q_0 \\ 1 & t_1 & t_1^2 & \cdots & t_1^n & p_1 & q_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n & p_n & q_n \end{array}\right]$$

$$\underline{REEF} \left[\begin{array}{ccccccc} 1 & t_0 & t_0^2 & \cdots & t_0^n & p_0 & q_0 \\ 1 & t_1 & t_1^2 & \cdots & t_1^n & p_1 & q_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & t_n & t_n^2 & \cdots & t_n^n & p_n & q_n \end{array}\right] = \left[\begin{array}{ccccccc} 1 & 0 & 0 & \cdots & 0 & s_0 & r_0 \\ 0 & 1 & 0 & \cdots & 0 & s_1 & r_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & s_n & r_n \end{array}\right]$$

4. Finally, once we have all the coefficients for all the coordinates, compute the polynomials and plot the resulting curve.

```
% Compute x and y interpolated values inside the interval
xResults = polyval(c(:,1)', steps);
yResults = polyval(c(:,2)', steps);

% Compute z values only when needed
if(dimension > 2)
  zResults = polyval(c(:,3)', steps);
  plot3(xResults, yResults, zResults, 'r');
  hold on;
  plot3(PX, PY, PZ, 'ok');
  hold on;
else
  plot(xResults, yResults, 'r');
  hold on;
  plot(PX, PY, 'ok');
  hold on;
endif
```

Lagrange

Lagrange polynomials of degree n given by $x_0, x_1, x_2, \dots, x_n$ where $x_i \neq x_j$ for $i \neq j$, the Ith polynomial is:

$$L_i^n(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \quad i = 0,1,2, \dots, n$$

We will compute a Lagrange basis for a polynomial of degree n where n is the number of points minus one. So, for points $(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n), (x_{n+1}, y_{n+1})$

We create a mesh with a specific distribution $\{t_0, t_1, t_2, \dots, t_n, t_{n+1}\}$

We will create two polynomials p(t), q(t) where:

$$p(t_0) = x_0, p(t_1) = x_1, p(t_2) = x_2, \dots, p(t_n) = x_n, p(t_{n+1}) = x_{n+1}$$

$$p(t_0) = y_0, p(t_1) = y_1, p(t_2) = y_2, \dots, p(t_n) = y_n, p(t_{n+1}) = y_{n+1}$$

We know use Lagrange interpolation to create basis $\{L_0^n, L_1^n, L_2^n, \dots, L_{n-1}^n, L_n^n\}$.

$$L_0^n(x) = \frac{x - t_1}{t_0 - t_1} * \frac{x - t_2}{t_0 - t_2} * \dots * \frac{x - t_{n-1}}{t_0 - t_{n-1}} * \frac{x - t_n}{t_0 - t_n}$$

$$L_1^n(x) = \frac{x - x_0}{x_1 - x_0} * \frac{x - x_2}{x_1 - x_2} * \dots * \frac{x - t_{n-1}}{t_1 - t_{n-1}} * \frac{x - t_n}{t_1 - t_n}$$

$$L_2^n(x) = \frac{x - x_0}{x_2 - x_0} * \frac{x - x_1}{x_2 - x_1} * \dots * \frac{x - t_{n-1}}{t_2 - t_{n-1}} * \frac{x - t_n}{t_2 - t_n}$$

$$\vdots$$

$$L_{n-1}^n(x) = \frac{x - x_0}{x_{n-1} - x_0} * \frac{x - x_1}{x_{n-1} - x_1} * \frac{x - t_2}{t_{n-1} - t_2} * \dots * \frac{x - t_n}{t_{n-1} - t_n}$$

$$L_n^n(x) = \frac{x - x_0}{x_n - x_0} * \frac{x - x_1}{x_n - x_1} * \frac{x - t_2}{t_n - t_2} * \dots * \frac{x - t_{n-1}}{t_n - t_{n-1}}$$

```
%go thought the points in the mesh
for i = 1 : points
  l=1;
  for j = 1 : points
    if j!=i
      #compute each of the Lagrange polynomial terms: x-x_j/x_i-x_j for each of the x in our mesh
      l*=(outnodes(n)-mesh(j))/(mesh(i)-mesh(j));
    endif
  endfor
```

Lagrange 1

In the figure *Lagrange 1* above we can see in code how to compute each of the terms of the Lagrange polynomials for the Lagrange base.

Once the basis is computed the can compute each of the polynomials

$$q(x) = x_0 * L_0^n(x) + x_1 * L_1^n(x) + x_2 * L_2^n(x) + \cdots + x_{n-1} * L_{n-1}^n(x) + x_n * L_n^n(x)$$
$$p(x) = y_0 * L_0^n(x) + y_1 * L_1^n(x) + y_2 * L_2^n(x) + \cdots + y_{n-1} * L_{n-1}^n(x) + y_n * L_n^n(x)$$

In the figure *Lagrange 2* we compute the value of the Lagrange term and add it to the already calculated ones.

```
finalValueX+=_Px(i)*l;
finalValueY+=_Py(i)*l;
if _dimension == 3
  finalValueZ+=_Pz(i)*l;
```

Lagrange 2

Newton

Similarly, to the Neville's method to obtain Lagrange interpolant polynomials, this method will also use a recursion, in this case to compute the **Divided Differences** that will give the vector of coordinates in **Newton basis**, which later will be used alongside **Newton polynomials** to construct the final interpolant polynomial.

1. In order to compute the divided differences, we will follow this **recursion** rule:

$$f[x_i\ ,\ x_{i+1},\ \ldots\ ,\ x_{i+k}] = \frac{f[x_{i+1},\ \ldots\ ,\ x_{i+k}]\ -\ f[x_i\ ,\ x_{i+1},\ \ldots\ ,\ x_{i+k-1}]}{x_{i+k}\ -\ x_i}$$

```
function d = divideddiferences(points, mesh)

    #geting the point count and creating the matrix which will store the diferences
    pointcount = columns(points);
    diferences = zeros(pointcount);
    #setting the first column
    diferences(:, 1) = points;
    #computing the rest of the diferences
    loops = 1;
    prev_values = pointcount;
    for i=2:pointcount
        count = 0;
        for j=i:pointcount
            diferences(j, i) = (diferences(j, i - 1) - diferences(j - 1, i - 1)) / (mesh(j) - mesh(j - loops));
            count++;
        endfor
            loops++;
        prev_values = count;
    endfor

    #setting the diferences into the vector
    for i=2:pointcount
        d(i- 1) = diferences(i,i);
    endfor
endfunction
```

2. Then what we need is to compute the Newton polynomial of the desired degree, defined by the formula below:

$$N_j(t) = \prod_{i=0}^{j-1}(x - x_i)$$

```
#function that computes the newton polynomial of the given degree
function out = ComputePolynomial(degree, value, mesh)
    #the staring value for the output
    out = 1;
    #accumulating the multiplications of the polynoimials to generate the wanted one
    for i=1:degree
        out *= value - mesh(i);
    endfor
endfunction
```

3. Now that we have both parts, to construct the interpolant polynomial what we must do is a **summation** of each divided difference multiplied by their respective newton polynomial, this is the defined in the formula below alongside the code implementation:

$$P(t_i) = \sum_{j=0}^{n}\left(DD(j) \cdot N_j(t_i)\right)$$

```
#function to evaluate the NewtonPolynomial
function out = Evaluate(mesh, diferences, nodeVal, value, pointcount)
    #the staring value for the output
    out = value;
    #making the sumatory of the computation
    for i=2:pointcount
        out += diferences(i - 1) * ComputePolynomial(i - 1, nodeVal, mesh);
    endfor
endfunction
```

Note that instead of computing every Newton polynomial and storing them, we are constructing them on run time, then being evaluated to enhance the efficiency of the code.

4. Finally, once the polynomial is evaluated at the value of every node, we just need to plot it. For that, we must store the results of the evaluations for each axis and then call **plot()** or **plot3()** to see the resulting curve.

```
#based on the different dimension configuration plot properly
switch (dimensions)
case 2
    plot(outX, outY, 'r');
    hold on;
    plot(pointsX, pointsY, 'ok');
    hold on;
case 3
    plot3(outX, outY, outZ, 'r');
    hold on;
    plot3(pointsX, pointsY, pointsZ, 'ok');
    hold on;
endswitch
```

Examples:

Interpolant polynomial through $(1,-1), (2,2)$, $(3,6), (1,0)$ using Chebyshev method and standard basis.

$$x_0 = 0, x_1 = 1, x_2 = 2, x_3 = 3$$

$$L_0^3 = \frac{x-x_1}{x_0-x_1} * \frac{x-x_2}{x_0-x_2} * \frac{x-x_3}{x_0-x_3} = \frac{x-1}{0-1} * \frac{x-2}{0-2} * \frac{x-3}{0-3} = -\frac{x^3}{6} + x^2 - \frac{11x}{6} + 1$$

$$L_1^3 = \frac{x-x_0}{x_1-x_0} * \frac{x-x_2}{x_1-x_2} * \frac{x-x_3}{x_1-x_3} = \frac{x-0}{1-0} * \frac{x-2}{1-2} * \frac{x-3}{1-3} = \frac{x^3}{2} - \frac{5x^2}{2} + 3x$$

$$L_2^3 = \frac{x-x_0}{x_2-x_0} * \frac{x-x_1}{x_2-x_1} * \frac{x-x_3}{x_2-x_3} = \frac{x-0}{2-0} * \frac{x-1}{2-1} * \frac{x-3}{2-3} = -\frac{x^3}{2} + 2x^2 - \frac{3x}{2}$$

$$L_3^3 = \frac{x-x_0}{x_3-x_0} * \frac{x-x_1}{x_3-x_1} * \frac{x-x_2}{x_3-x_2} = \frac{x-0}{3-0} * \frac{x-1}{3-1} * \frac{x-2}{3-2} = \frac{x^3}{6} - \frac{x^2}{2} + \frac{x}{3}$$

$$P_x = 1\left(-\frac{x^3}{6} + x^2 - \frac{11x}{6} + 1\right) + 2\left(\frac{x^3}{2} - \frac{5x^2}{2} + 3x\right) + 3\left(-\frac{x^3}{2} + 2x^2 - \frac{3x}{2}\right) + 1\left(\frac{x^3}{6} - \frac{x^2}{2} + \frac{x}{3}\right)$$

$$= -\frac{x^3}{2} + \frac{3x^2}{2} + 1$$

$$P_y = -1\left(-\frac{x^3}{6} + x^2 - \frac{11x}{6} + 1\right) + 2\left(\frac{x^3}{2} - \frac{5x^2}{2} + 3x\right) + 6\left(-\frac{x^3}{2} + 2x^2 - \frac{3x}{2}\right)$$

$$+ 0\left(\frac{x^3}{6} - \frac{x^2}{2} + \frac{x}{3}\right) x22 + \frac{x}{3} = -\frac{11x^3}{6} + 6x^2 - \frac{7x}{6} - 1$$

Now we can evaluate x and obtain a value for x and y.

Value x	Numerical	Gauss-Jordan	Lagrange	Newton
0.7	X=1.5635 Y=0.4945	X=1.5635 Y=0.4945	X=1.5635 Y=0.4945	X=1.5635 Y=0.4945
1.4	X=2.568 Y=4.096	X=2.568 Y=4.096	X=2.568 Y=4.096	X=2.568 Y=4.096
1.8	X=2.944 Y=5.648	X=2.944 Y=5.648	X=2.944 Y=5.648	X=2.944 Y=5.648
2.2	X=2.936 Y=5.952	X=2.936 Y=5.952	X=2.936 Y=5.952	X=2.936 Y=5.952
2.7	X=2.0935 Y=3.5045	X=2.0935 Y=3.5045	X=2.0935 Y=3.5045	X=2.0935 Y=3.5045
2.9	X=1.4205 Y=1.3635	X=1.4205 Y=1.3635	X=1.4205 Y=1.635	X=1.4205 Y=1.3635

As we can see in the table above the values of the different methods are consistent with the numerical one.

Observations:

Through the process of evaluating the correctness of the codes we have seen a pattern among all three methods, the more output nodes we have the smother the resulting curve. Which makes sense since having more samples implies that we have more detailed information of where the curve passes through, thus resulting in a smoother graph. Also, we checked the breaking point of each method by randomly generating a set of values. We noticed that methods break when over 40 points are given, this can vary based on the points that are introduced and how each method is implemented. About time of computation, Gauss-Jordan seems very consistent independently of the number of points. However, Lagrange and Newton scale linearly with the number of input points. In terms of speed, we can say that Gauss-Jordan is the fastest one, then Newton and lastly Lagrange.

Bibliography:

MAT300 Lecture Notes: lecture4, lecture5, lecture6.