Análise Numérica: Trabalho prático 3

André Cerqueira - up201804991
Maria Leite - up201905503
Margarida Miranda - up201909858
Simão Lopes - up201805175

Maio 2021

Introdução

Este trabalho foi realizado no âmbito da UC Análise Numérica com objetivo de fazer interpolação polinomial de uma função.
Para fazer esta interpolação, construímos um polínomio interpolador e splines cúbicos naturais, de forma a permitem avaliar a sua aproximação à função dada. Além disso, foi feita a majoração dos erros segundo os polinómios e os splines calculados quando conhecida a função.

Exercício 1

Neste exercício apenas conhecemos 12 pontos de uma função evaporação desconhecida. A partir deles calculamos o polinómio interpolador e o spline cúbico que interpola os pontos apresentados.

Alínea a

Dados 12 pontos de abcissas distintas existe um e um só polinómio de grau menor ou igual a 11 que os contém. Assim, para calcular o polinómio interpolador, usamos o método de lagrange. Implementamos o programa que imprime o polinómio gerado.

```
#include<stdio.h>
#define L 12
#define C 2


double matrix[L][C] =
{
  {1.0 , 8.6},
  {2.0 , 7.0},
  {3.0 , 6.4},
  {4.0 , 4.0},
```

```
  {5.0 , 2.8},
  {6.0 , 1.8},
  {7.0 , 1.8},
  {8.0 , 2.3},
  {9.0 , 3.2},
  {10.0 , 4.7},
  {11.0 , 6.2},
  {12.0 , 7.9},
};

int main() {

  printf("(");
  for (int i = 1 ; i <= L; i++) {
    printf("(");
    for (int x = 1 ; x <= L; x++) {
      if (x != i)
        printf("(x-%d)",x);
    }
    printf(")/(");
    for (int k = 1 ; k <= L; k++) {
      if (k != i)
        printf("(%d-%d)",i,k);
    }
    if ( i == L)
      printf(")*%f",matrix[i-1][1]);
    else
      printf(")*%f+",matrix[i-1][1]);
  }
  printf(")");

  return 0;
}
```

Note-se que este programa apenas faz o print segundo a fórmula do método de lagrange.

$$p_n(x) = f_0 I_0(x) + ... + f_n I_n(x)$$

O polinómio gerado é:

$$p_{11}(x) =$$

$$\frac{(((x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8)(x-9)(x-10)(x-11)(x-12))}{((1-2)(1-3)(1-4)(1-5)(1-6)(1-7)(1-8)(1-9)(1-10)(1-11)(1-12))} \times 8.600000 +$$

$$\frac{((x-1)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8)(x-9)(x-10)(x-11)(x-12))}{((2-1)(2-3)(2-4)(2-5)(2-6)(2-7)(2-8)(2-9)(2-10)(2-11)(2-12))} \times 7.000000 +$$

$$\frac{((x-1)(x-2)(x-4)(x-5)(x-6)(x-7)(x-8)(x-9)(x-10)(x-11)(x-12))}{((3-1)(3-2)(3-4)(3-5)(3-6)(3-7)(3-8)(3-9)(3-10)(3-11)(3-12))} \times 6.400000 +$$

$$\frac{((x-1)(x-2)(x-3)(x-5)(x-6)(x-7)(x-8)(x-9)(x-10)(x-11)(x-12))}{((4-1)(4-2)(4-3)(4-5)(4-6)(4-7)(4-8)(4-9)(4-10)(4-11)(4-12))}\times 4.000000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-6)(x-7)(x-8)(x-9)(x-10)(x-11)(x-12))}{((5-1)(5-2)(5-3)(5-4)(5-6)(5-7)(5-8)(5-9)(5-10)(5-11)(5-12))}\times 2.800000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-7)(x-8)(x-9)(x-10)(x-11)(x-12))}{((6-1)(6-2)(6-3)(6-4)(6-5)(6-7)(6-8)(6-9)(6-10)(6-11)(6-12))}\times 1.800000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-8)(x-9)(x-10)(x-11)(x-12))}{((7-1)(7-2)(7-3)(7-4)(7-5)(7-6)(7-8)(7-9)(7-10)(7-11)(7-12))}\times 1.800000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-9)(x-10)(x-11)(x-12))}{((8-1)(8-2)(8-3)(8-4)(8-5)(8-6)(8-7)(8-9)(8-10)(8-11)(8-12))}\times 2.300000+$$

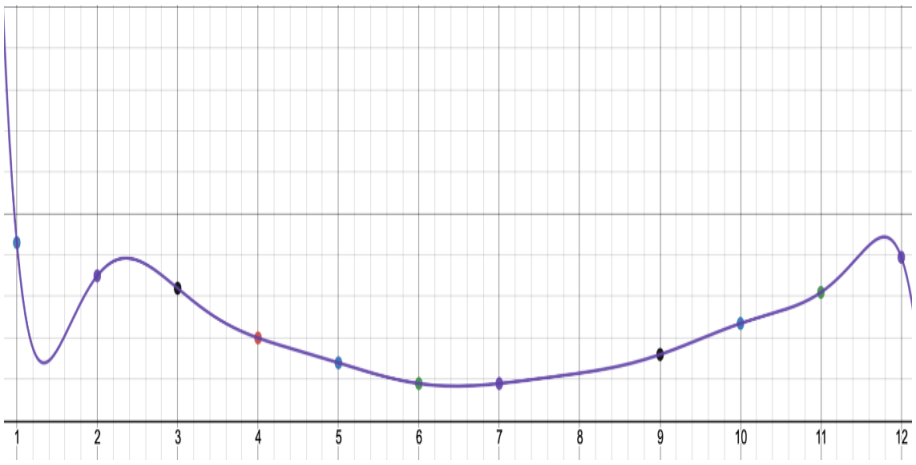
$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8)(x-10)(x-11)(x-12))}{((9-1)(9-2)(9-3)(9-4)(9-5)(9-6)(9-7)(9-8)(9-10)(9-11)(9-12))}\times 3.200000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8)(x-9)(x-11)(x-12))}{((10-1)(10-2)(10-3)(10-4)(10-5)(10-6)(10-7)(10-8)(10-9)(10-11)(10-12))}\times 4.700000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8)(x-9)(x-10)(x-12))}{((11-1)(11-2)(11-3)(11-4)(11-5)(11-6)(11-7)(11-8)(11-9)(11-10)(11-12))}\times 6.200000+$$

$$\frac{((x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7)(x-8)(x-9)(x-10)(x-11))}{((12-1)(12-2)(12-3)(12-4)(12-5)(12-6)(12-7)(12-8)(12-9)(12-10)(12-11))}\times 7.900000+$$

Partindo do polinómio construido, desenhamos graficamente a aproximação da função evaporação, como se pode ver pelo gráfico:



Os pontos marcados representam a tabela de pontos dada no enunciado.

Além da construção do polinómio interpolador, também construimos o spline cúbico natural. Assim, dados 12 pontos com abcissas igualmente espaçadas e por isso h=1. Note-se que o h representa o intervalor entre as abcissas. Pelo que o spline vai ser de grau menor ou igual que 11, bem como o spline tem 11 ramos. Pelo que podemos observar pelo seguinte:

$$s(x)\begin{cases} 0.166667 \times M0 + 0.666667 \times M1 + 0.166667 \times M2 = 1.0 \\ 0.166667 \times M1 + 0.666667 \times M2 + 0.166667 \times M3 = -1.8 \\ 0.166667 \times M2 + 0.666667 \times M3 + 0.166667 \times M4 = 1.2 \\ 0.166667 \times M3 + 0.666667 \times M4 + 0.166667 \times M5 = 0.2 \\ 0.166667 \times M4 + 0.666667 \times M5 + 0.166667 \times M6 = 1.0 \\ 0.166667 \times M5 + 0.666667 \times M6 + 0.166667 \times M7 = 0.5 \\ 0.166667 \times M6 + 0.666667 \times M7 + 0.166667 \times M8 = 0.4 \\ 0.166667 \times M7 + 0.666667 \times M8 + 0.166667 \times M9 = 0.6 \\ 0.166667 \times M8 + 0.666667 \times M9 + 0.166667 \times M10 = 0.0 \\ 0.166667 \times M9 + 0.666667 \times M10 + 0.166667 \times M11 = 0.2 \\ M_0 = 0 \\ M_{11} = 0 \end{cases}$$

Esta função por ramos foi feita através de um programa que implementamos da seguinte forma:

```c
#include<stdio.h>
#include<math.h>

int main(){
    int m,i;
    printf("Enter the number of points:\n");
    scanf("%d",&m);

    double x[m];
```

```
    double y[m];
    double h[m-1];
    printf("Enter the x values:\n");
    for(i=0;i<m;i++){
        scanf("%lf",&x[i]);
    }
    printf("Enter the y values:\n");
    for(i=0;i<m;i++){
        scanf("%lf",&y[i]);
    }
    for(i=0;i<m;i++){
        //h[i]=x[i+1]-x[i];
        h[i]=1;
    }

    double matriz[m-2][8];

    for(i=1;i<m-1;i++){

        matriz[i][1]=h[i]/6;
        matriz[i][2]= i-1;
        matriz[i][3]= (h[i]+h[i+1] )/3;
        matriz[i][4]= i;
        matriz[i][5]= h[i+1]/6;
        matriz[i][6]= i+1;
        matriz[i][7]= ((y[i+1]-y[i] )/h[i+1] ) - ((y[i]-y[i-1] )/h[
    i] );

    }

    for (int i = 1; i <m-1; i++) {

  printf("s(%d) = (%.2f* M%.0f) + (%.2f *M%.0f) + (%.2f * M%.0f) =
    %.2f\n",i, matriz[i][1], matriz[i][2],
  matriz[i][3], matriz[i][4], matriz[i][5], matriz[i][6], matriz[i
    ][7]);

      }

return 0;
}
```

Este programa imprime a função por ramos apresentada em cima. Assim, através de um software que resolve sistemas encontramos os sguintes valores dos M:

$$M0 = 0$$

$$M1 = 2.52278$$

$$M2 = -4.09112$$

$$M3 = 3.04171$$

$$M4 = -0.87572$$

$$M5 = 1.653618$$

$$M6 = 0.25922$$

$$M7 = 0.30949$$

$$M8 = 0.902814$$

$$M9 = -0.320751$$

$$M10 = 0.380187$$

$$M11 = 0$$

É de realçar que a condição M0=0 e M11=0, vem pelo facto de querermos um spline natural. Com estes valores de M conseguimos construir o spline apresentado abaixo da função evaporação:

$$s(x)\begin{cases} 2.52278 \times \frac{(x-1)^3}{6} + 8.60000 \times \frac{(2-x)}{1} + 6.57953 \times \frac{(x-1)}{1}, 1 < x < 2 \\ 2.52278 \times \frac{(3-x)^3}{6} + -4.09112 \times \frac{(x-2)^3}{6} + 6.57953 \times \frac{(3-x)}{1} + 7.08187 \times \frac{(x-2)}{1}, 2 < x < 3 \\ -4.09112 \times \frac{(4-x)^3}{6} + 3.04171 \times \frac{(x-3)^3}{6} + 7.08187 \times \frac{(4-x)}{1} + 3.49304 \times \frac{(x-3)}{1}, 3 < x < 4 \\ 3.04171 \times \frac{(5-x)^3}{6} + -0.87572 \times \frac{(x-4)^3}{6} + 3.49304 \times \frac{(5-x)}{1} + 2.94596 \times \frac{(x-4)}{1}, 4 < x < 5 \\ -0.87572 \times \frac{(6-x)^3}{6} + 1.65362 \times \frac{(x-5)^3}{6} + 2.94596 \times \frac{(6-x)}{1} + 1.52439 \times \frac{(x-5)}{1}, 5 < x < 6 \\ 1.65362 \times \frac{(7-x)^3}{6} + 0.25922 \times \frac{(x-6)^3}{6} + 1.52439 \times \frac{(7-x)}{1} + 1.75680 \times \frac{(x-6)}{1}, 6 < x < 7 \\ 0.25922 \times \frac{(8-x)^3}{6} + 0.30949 \times \frac{(x-7)^3}{6} + 1.75680 \times \frac{(8-x)}{1} + 2.24842 \times \frac{(x-7)}{1}, 7 < x < 8 \\ 0.30949 \times \frac{(9-x)^3}{6} + 0.90281 \times \frac{(x-8)^3}{6} + 2.24842 \times \frac{(9-x)}{1} + 3.04953 \times \frac{(x-8)}{1}, 8 < x < 9 \\ 0.90281 \times \frac{(10-x)^3}{6} + -0.32075 \times \frac{(x-9)^3}{6} + 3.04953 \times \frac{(10-x)}{1} + 4.75346 \times \frac{(x-9)}{1}, 9 < x < 10 \\ -0.32075 \times \frac{(11-x)^3}{6} + 0.38019 \times \frac{(x-10)^3}{6} + 4.75346 \times \frac{(11-x)}{1} + 6.13663 \times \frac{(x-10)}{1}, 10 < x < 11 \\ 0.38019 \times \frac{(12-x)^3}{6} + 6.13663 \times \frac{(12-x)}{1} + 7.90000 \times \frac{(x-11)}{1}, 11 < x < 12 \end{cases}$$

Para conseguirmos este sistema s(x), implementamos o seguinte programa através da fórmula que devolve-nos o sistema acima apresentado.

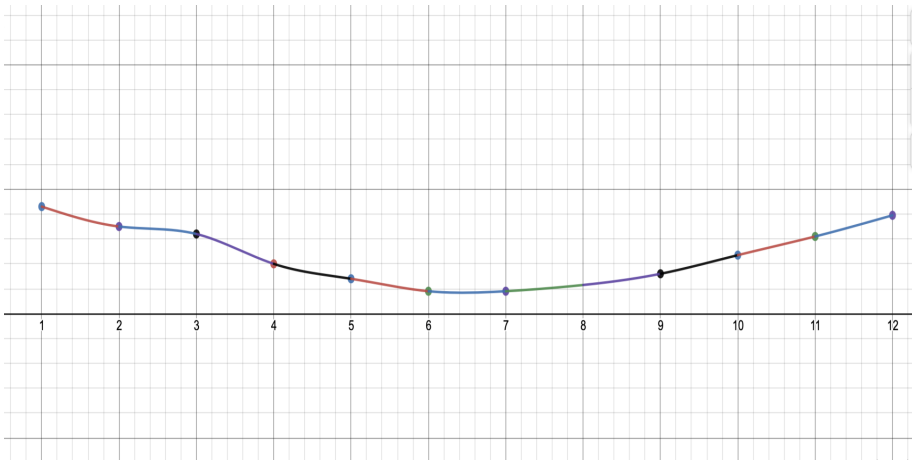
$$M_{i-1} \times \frac{(xi - x)^3}{6 \times h_i} + M_i \times \frac{(x - x_{i-1})^3}{6 \times h_i} + (f_{i-1} - M_{i-1} \times \frac{(h_i)^2}{6}) \times \frac{x_i - x}{hi} + (f_i - M_i \times \frac{(h_i)^2}{6}) \times \frac{x - x_{i-1}}{hi}$$

```c
#include<stdio.h>
#include<math.h>

int main(){
  int m,i;
  printf("Enter the number of points:\n");
  scanf("%d",&m);

  double x[m];
  double y[m];
  double mi[m];
  double h[m-1];
  printf("Enter the xs values:\n");

```

```
  for(i=0;i<m;i++){
    scanf("%lf",&x[i]);
  }

  printf("Enter the y values:\n");

  for(i=0;i<m;i++){
    scanf("%lf",&y[i]);
  }

  for(i=0;i<m-1;i++){
    h[i]=1; // porque as abcissas estao igualmente espa adas

  }

  printf("Enter the mi values:\n");

  mi[0]=0;
  mi[m-1]=0;

  for(i=1;i<m-1;i++){
    scanf("%lf",&mi[i]);
  }

#ifdef DEBUG
  for (int i = 1 ; i < m-1 ; i++) {
    printf("%d: %lf\n",i,mi[i]);
  }
#endif


  for (int i = 1; i < m; i++) {

    double f =  y[i]-(mi[i]*((0.16667))); //
    double f2= y[i-1]-(mi[i-1]*(0.16667));
    printf(" %.5lf* (%d-x)^3/ %d + %.5lf *(x-%d)^3/ %d + %.5lf* (%d
    -x)/%d  + %.5lf*(x-%d)/%d \n",
        mi[i-1], i+1 ,6*1 , mi[i], i, 6*1, f2,i+1,1*1, f,i,1*1 );
  }
  return 0;
}
```

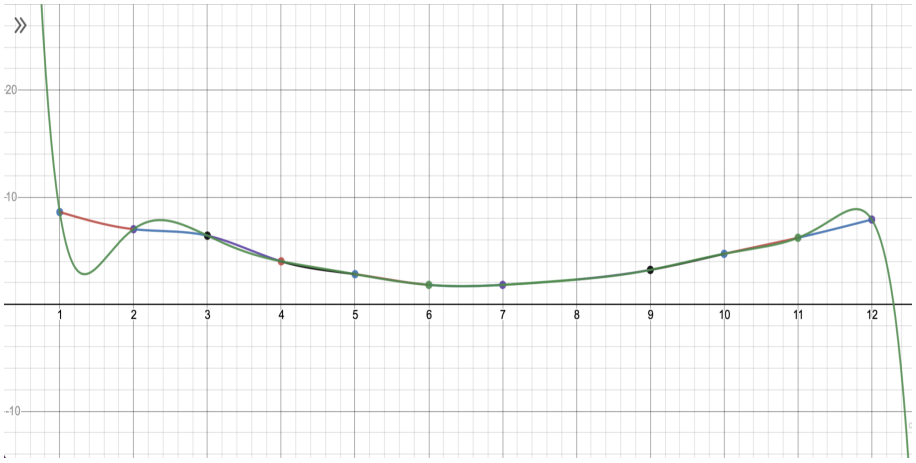
Note-se que o sistema s(x) é um polinómio de grau ¡= 11, para além de ser continuo em[1,12] e as suas derivadas até à décima segunda também são contínuas.

Partindo do spline construido, desenhamos graficamente as a aproximações da função evaporação, como podemos ver pelo gráfico abaixo representado:



Alínea b

O grafico abaixo representado mostra os pontos dados no enunciado juntamente com o spline e o polinomio interpolador construidos na alínea anterior.



Partindo da análise do gráfico podemos perceber que tanto polinómio interpolador como o spline passam nos pontos referidos. Tal como era de esperar visto que são aproximações à função.

Podemos ainda dizer que o spline se aproxima melhor da função e desta forma será a aproximação mais aceitável, na medida em que passa de uma forma mais "reta"nos pontos não dispersando tanto.

Exercício 2

Alínea a

Dada a função $f(x) = x - cos(x)^3$ construímos um conjunto de 7 pontos de abcissas igualmente espaçadas.

O calculo usado foi : $h = \frac{a-b}{n}$

Sendo o intervalo [a,b] considerado igual a [-3,3] e n=6.

Logo: $h = \frac{3-(-3)}{6} = 1$

x	-3	-2	-1	0	1	2	3
f(x)	-2.02972	-1.92793	-1.15772	-1	0.84227	2.07206	3.97027

Alínea b

Partindo dos pontos calculados na alínea anterior construimos o polinómio interpolador e o spline cúbico.

Dados 7 pontos de abcissas distintas existe um e um só polinómio de grau menor ou igual a 6 que os contém.

Assim, calcular o polinómio interpolador usamos o método de lagrange.

Implementamos o programa a seguir apresentado segundo a fórmula do método mencionado.

```
#include<stdio.h>
#define L 7
#define C 2
double matrix[L][C] =
{
  {-3.0 , -2.02972},
  {-2.0 , -1.92793},
  {-1.0 , -1.15772},
  { 0.0 , -1.0},
  {-1.0 , 0.84227},
  {-2.0 , 2.07206},
  {-3.0 , 3.97027},
};

int main() {

  printf("(");
  int a = 1;
  int h = L+-3;
  for (int i = -3 ; i < h ; i++) {
    printf("(");
    for (int x = -3 ; x <= L; x++) {
      if (x != i)
        printf("(x-%d)",x);
    }
    printf(")/(");
    for (int k = -3 ; k <= L; k++) {
      if (k != i)
```

```
29          printf("(%d-%d)",i,k);
30      }
31      if ( a == L)
32        printf(")*(%f)",matrix[a-1][1]);
33      else
34        printf(")*(%f)+",matrix[a-1][1]);
35      a++;
36    }
37    printf(")");
38
39    return 0;
40  }
```

Sendo que é de notar que o programa imprime o polinómio que é :

$$p_6(x) =$$

$$\frac{(((x+2)(x+1)(x-0)(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7))}{((-3+2)(-3+1)(-3-0)(-3-1)(-3-2)(-3-3)(-3-4)(-3-5)(-3-6)(-3-7))}\times(-2.029720)$$

$$+\frac{((x+3)(x+1)(x-0)(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7))}{((-2+3)(-2+1)(-2-0)(-2-1)(-2-2)(-2-3)(-2-4)(-2-5)(-2-6)(-2-7))}\times(-1.927930)$$

$$+\frac{((x+3)(x+2)(x-0)(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7))}{((-1+3)(-1+2)(-1-0)(-1-1)(-1-2)(-1-3)(-1-4)(-1-5)(-1-6)(-1-7))}\times(-1.157720)$$

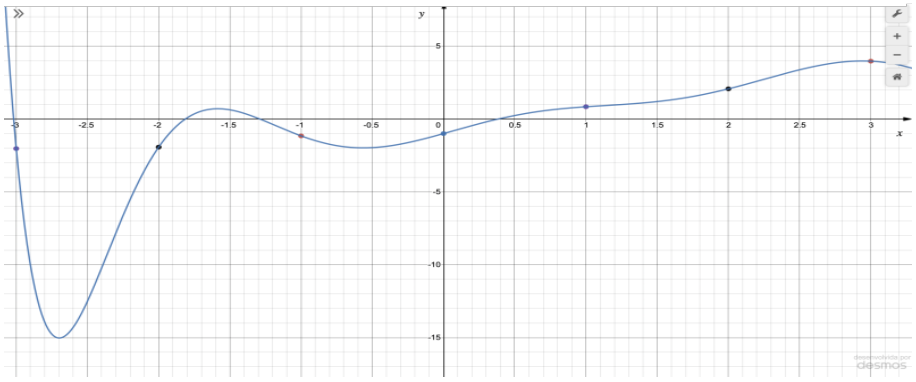
$$+\frac{((x+3)(x+2)(x+1)(x-1)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7))}{((0+3)(0+2)(0+1)(0-1)(0-2)(0-3)(0-4)(0-5)(0-6)(0-7))}\times(-1.000000)$$

$$+\frac{((x+3)(x+2)(x+1)(x-0)(x-2)(x-3)(x-4)(x-5)(x-6)(x-7))}{((1+3)(1+2)(1+1)(1-0)(1-2)(1-3)(1-4)(1-5)(1-6)(1-7))}\times 0.842270$$

$$+\frac{((x+3)(x+2)(x+1)(x-0)(x-1)(x-3)(x-4)(x-5)(x-6)(x-7))}{((2+3)(2+2)(2+1)(2-0)(2-1)(2-3)(2-4)(2-5)(2-6)(2-7))}\times 2.072060$$

$$+\frac{((x+3)(x+2)(x+1)(x-0)(x-1)(x-2)(x-4)(x-5)(x-6)(x-7))}{((3+3)(3+2)(3+1)(3-0)(3-1)(3-2)(3-4)(3-5)(3-6)(3-7))}\times 3.970270$$

No gráfico abaixo, podemos observar o polinómio interpolador no intervalo considerado:



Além da construção do polinómio interpolador, construimos o spline cúbico natural. Assim, dados 7 pontos com abcissas igualmente espaçadas e por isso h=1. Note-se que o h representa o intervalor entre as abcissas. Vemos, desta forma que o spline vai ser de grau menor ou igual que 6, bem como vamos ter 6 ramos no spline. Tal como podemos observar:

$$s(x) = \begin{cases} 0.166667 \times M0 + 0.666667 \times M1 + 0.166667 \times M2 = 0.668 \\ 0.166667 \times M1 + 0.666667 \times M2 + 0.166667 \times M3 = -0.610000 \\ 0.166667 \times M2 + 0.666667 \times M3 + 0.166667 \times M4 = 1.680000 \\ 0.166667 \times M3 + 0.666667 \times M4 + 0.166667 \times M5 = -0.610000 \\ 0.166667 \times M4 + 0.666667 \times M5 + 0.166667 \times M6 = -7.060000 \\ M_0 = 0 \\ M_6 = 0 \end{cases}$$

Esta função por ramos foi feita através de um programa que implementamos:

```c
#include<stdio.h>
#include<math.h>

int main(){
    int m,i;
    printf("Enter the number of points:\n");
    scanf("%d",&m);

    double x[m];
    double y[m];
    double h[m-1];
    printf("Enter the x values:\n");
    for(i=0;i<m;i++){
        scanf("%lf",&x[i]);
    }
    printf("Enter the y values:\n");
    for(i=0;i<m;i++){
        scanf("%lf",&y[i]);
    }
```

```
    for(i=0;i<m;i++){
        //h[i]=x[i+1]-x[i];
        h[i]=1;
    }

    double matriz[m-2][8];

    for(i=1;i<m-1;i++){

        matriz[i][1]=h[i]/6;
        matriz[i][2]= i-1;
        matriz[i][3]= (h[i]+h[i+1] )/3;
        matriz[i][4]= i;
        matriz[i][5]= h[i+1]/6;
        matriz[i][6]= i+1;
        matriz[i][7]= ((y[i+1]-y[i] )/h[i+1] ) - ((y[i]-y[i-1] )/h[
    i] );

    }

    for (int i = 1; i <m-1; i++) {

  printf("s(%d) = (%.2f* M%.0f) + (%.2f *M%.0f) + (%.2f * M%.0f) =
    %.2f\n",i, matriz[i][1], matriz[i][2],
  matriz[i][3], matriz[i][4], matriz[i][5], matriz[i][6], matriz[i
    ][7]);

      }

return 0;
}
```

Este programa imprime a função por ramos apresentada em cima. Assim, através de um software que resolve sistemas encontramos os sguintes valores dos M:

$$M0 = 0$$

$$M1 = 1.49961650$$

$$M2 = -1.9808415$$

$$M3 = 2.7189942$$

$$M4 = 1.1471584$$

$$M5 = -1.08283$$

$$M6 = 0$$

É de realçar que a condição M0=0 e M6=0, vem pelo facto de querermos um spline natural. Com estes valores de M conseguimes construir o spline apresentado abaixo da função considerada:

$$s(x) = \begin{cases} 1.49962 \times \frac{(x+3)^3}{6} + -2.02972 \times \frac{(-2-x)}{1} + -2.17787 \times \frac{(x+3)}{1}, -3 < x < -2 \\ 1.49962 \times \frac{(-1-x)^3}{6} + -1.98084 \times \frac{(x+2)^3}{6} + -2.17787 \times \frac{(-1-x)}{1} + -0.82757 \times \frac{(x+2)}{1}, -2 < x < -1 \\ -1.98084 \times \frac{(0-x)^3}{6} + 2.71899 \times \frac{(x+1)^3}{6} + -0.82757 \times \frac{(0-x)}{1} + -1.45317 \times \frac{(x+1)}{1}, -1 < x < 0 \\ 2.71899 \times \frac{(1-x)^3}{6} + 1.14716 \times \frac{(x-0)^3}{6} + -1.45317 \times \frac{(1-x)}{1} + 0.65107 \times \frac{(x-0)}{1}, 0 < x < 1 \\ 1.14716 \times \frac{(2-x)^3}{6} + -10.82838 \times \frac{(x-1)^3}{6} + 0.65107 \times \frac{(2-x)}{1} + 3.87683 \times \frac{(x-1)}{1}, 1 < x < 2 \\ -1.082838 \times \frac{(3-x)^3}{6} + 3.87683 \times \frac{(3-x)}{1} + 3.97027 \times \frac{(x-2)}{1}, 2 < x < 3 \end{cases}$$

Para conseguirmos este sistema s(x) implementamos o seguinte programa através da formula que nos devolve o sistema em cima apresentado. formula

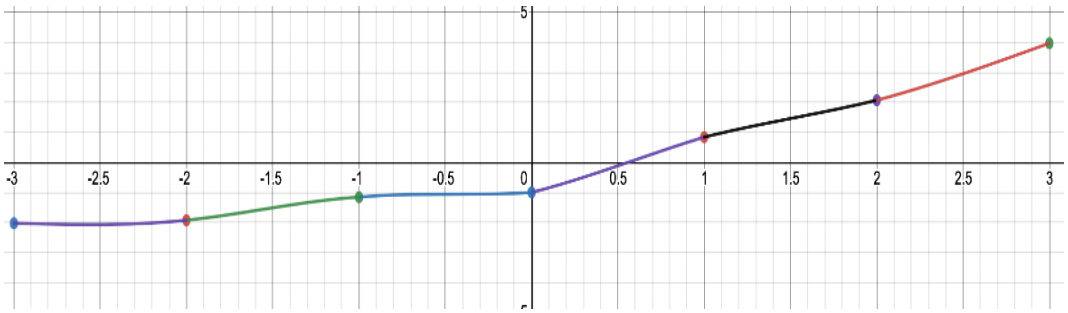
$$M_{i-1} \times \frac{(xi - x)^3}{6 \times h_i} + M_i \times \frac{(x - x_{i-1})^3}{6 \times h_i} + (f_{i-1} - M_{i-1} \times \frac{(h_i)^2}{6}) \times \frac{x_i - x}{hi} + (f_i - M_i \times \frac{(h_i)^2}{6}) \times \frac{x - x_{i-1}}{hi}$$

Note-se que o sistema s(x) é um polinómio de grau ¡= 6, é continuo em [-3,3] e as suas derivadas até à sétima também são contínuas.

```c
#include<stdio.h>
#include<math.h>

int main(){
  int m,i;
  printf("Enter the number of points:\n");
  scanf("%d",&m);

  double x[m];
  double y[m];
  double mi[m];
  double h[m-1];
  printf("Enter the xs values:\n");
  for(i=0;i<m;i++){
      scanf("%lf",&x[i]);
  }
  printf("Enter the y values:\n");
  for(i=0;i<m;i++){
      scanf("%lf",&y[i]);
  }
  for(i=0;i<m-1;i++){
    h[i]=1; // porque as abcissas estao iugualmente espa adas

  }

  printf("Enter the mi values:\n");
  mi[0]=0;
  mi[m-1]=0;
  for(i=1;i<m-1;i++){
      scanf("%lf",&mi[i]);
  }


  for (int i = 1; i < m; i++) {
    double f =  y[i]-(mi[i]*((0.16667))); //
     double f2= y[i-1]-(mi[i-1]*(0.16667));
```

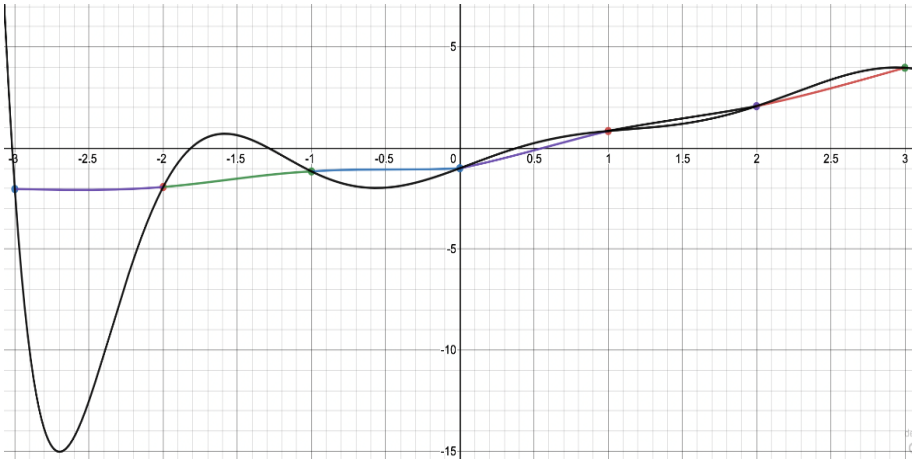
```
37      printf(" %.5lf* (%d-x)^3/ %d + %.5lf *(x-%d)^3/ %d + %.5lf* (%
    d-x)/%d  + %.5lf*(x-%d)/%d \n",
38          mi[i-1], i-3 ,6*1 , mi[i], i-4, 6*1, f2,i-3,1*1, f,i-4,1*1
     );
39
40  }
41
42 return 0;
43 }
```

No gráfico abaixo podemos observar o spline cúbico natural no intervalo considerado:



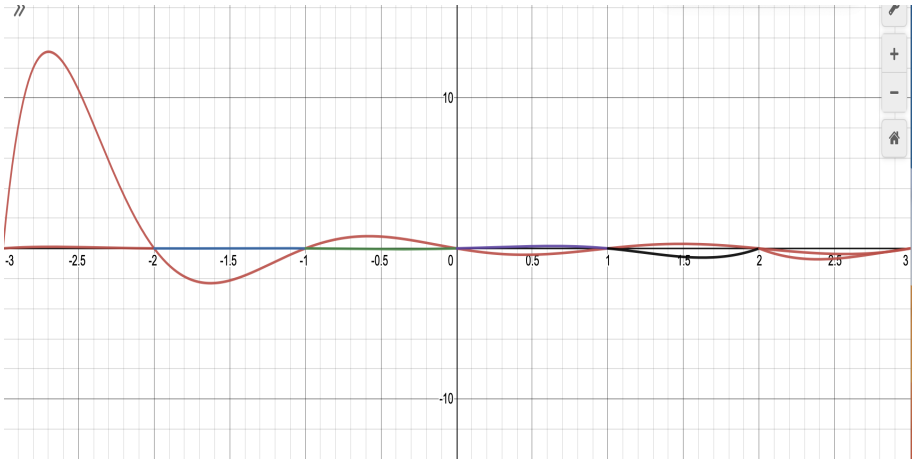
Alínea c

Partindo do polinómio e do spline construídos, desenhamos graficamente as duas aproximações da função $f(x) = x - cos(x)^3$.



Note-se ainda o gráfico apresentado abaixo que mostra as funções erro do polinómio e do spline construidos anteriormente.
A vermelho:$|x - cos(x)^3 - p(x)|$.

Com múltiplas cores que determinam os diferentes ramos: $|x - cos(x)^3 - s(x)|$.



É de realçar que ambos os gráficos fazem uma aproximação à função. Tal como é comprovado pelo primeiro gráfico em que tanto o spline como o polinómio interpolador passam nos pontos considerados.
Relativamente ao segundo gráfico, podemos concluir que o erro do spline é muito menor. Percebemos isso já que a função erro no intervalo a avaliar encontra-se sempre muito perto de 0(quase que se anula). Logo, uma aproximação por splines é melhor dado que não está sujeito a tantos erros de arredondamento como o polinómio interpolador.
Isto deve-se ao facto de dividirmos o intervalo em vários subintervalos e aplicar interpolação polinomial em cada um dos subintervalos, logo fazemos interpolação usando polinómios de grau baixo.
Assim, no polinómio interpolador como podemos confirmar pelo gráfico apresentado, a função distancia-se várias vezes de 0, levando-nos a dizer que os erros são maiores que no spline.

Alínea d

Nesta alínea calculamos os majorantes dos erros ao estimar f(0.1) e f(2.6). Assim, para isso, começamos por calcular o valor exato de

$$f(0.1) = -0.88508$$

$$f(2.6) = 3.22917$$

Neste seguimento implementamos um programa que calcula o polinómio inerpolador segundo o método de lagrange num determinado ponto.

```
1
2 #include<stdio.h>
3
```

```
4  double I(int i, int n, double x[n+1], double X){
5      int j;
6      double calc=1;
7      for(j=0;j<=n;j++){
8          if(j!=i)
9              calc=calc*(X-x[j])/(x[i]-x[j]);
10     }
11         return calc;
12 }
13 double Polinomio(int n, double x[n+1], double y[n+1], double X){
14     double soma=0;
15     int i;
16     for(i=0;i<=n;i++){
17         soma=soma+I(i,n,x,X)*y[i];
18     }
19     return soma;
20 }
21 int main(){
22     int i,n;
23     printf("Introduza n mero de pontos:\n");
24     scanf("%d",&n);
25     n=n-1;
26
27     double x[n+1];
28     double y[n+1];
29     printf("Introduza valores de x:\n");
30     for(i=0;i<n+1;i++){
31       scanf("%lf",&x[i]);
32     }
33
34     printf("Introduza valores de y:\n");
35     for(i=0;i<n+1;i++){
36         scanf("%lf",&y[i]);
37     }
38
39     double X;
40     printf("Introduza o valor de x que quer interpolar:\n");
41     scanf("%lf",&X);
42     printf("O valor interpolado    %lf\n",Polinomio(n,x,y,X));
43     return 0;
44   }
```

O resultado de $p(0.1) = -0.888786$ e o resultado de $p(2.6) = 2.279877$.

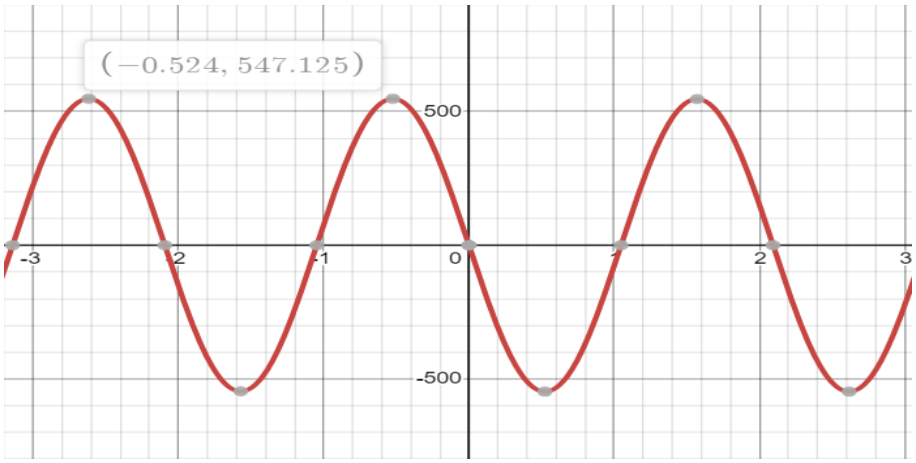
Para majorar o erro do polinómio interpolador usamos a fórmula:

$$|f(x) - p_6(x)| <= \frac{M}{7!}|(x-x0)(x-x1)(x-x2)(x-x3)(x-x4)(x-x5)(x-x6)|$$

Sendo M= $max|f_7(x)|$
Podemos ver então que $f_7(x) = 3\left(182\sin(x) - 729\cos^2(x)\sin(x)\right)$

O grafico abaixo representa a sétima derivada da função, podemos assim

verificar que o maximo entre [-3,3] é 547.125 que representa o M.



Para 0.1 :

$$|-0.88508-(-0.888786)| <=$$

$$\frac{547.125}{7!}|(0.1+3))(0.1+2))(0.1+1)(0.1-0)(0.1-1)(0.1-2)(0.1-3)| <= 3.3*10^1$$

Para 2.6 :

$$|3.22917-3.279877| <=$$

$$\frac{547.125}{7!}|(2.6-(-3))(2.6-(-2))(2.6-(-1))(2.6-0)(2.6-1)(2.6-2)(2.6-3)| <= 8.5*10^2$$

Para majorar o erro para o spline cúbico a fórmula é diferente como se pode verficar com:

$$|f(x)-s(x)| <= \frac{5}{384}M \times h^4$$

Sendo $M = max|f_4(x)|$ e h é o máximo dos h entre 1 e 7.

Calculamos assima quarta devivada da função:

$$3\left(20\cos\left(x\right)-27\cos^3\left(x\right)\right)$$

Tal como podemos ver pelo grafico o maximo é atingido no ponto identificado.

Logo, o $M = 21$.

Além disso, o h = 1, pois como as abcissas estão igualmente espaçadas o intervalo entre elas é sempre o mesmo.

Note-se que temos de calcular o spline em 0.1 e 2.6 e para isso usamos a linha do sistema que contém cada um dos pontos, ou seja:

$$0 < 0.1 < 1$$

$$2 < 2.6 < 3$$

Assim, obtemos os resultados seguintes:

$$s(0.1) = -0.91219$$
$$s(2.6) = 3.31739128$$

Por isso obtivemos:

$$|f(x) - s(x)| <= \frac{5}{384} 21 \times 1^4$$

Para 0.1 :

$$| - 0.88508 - (-0.91219)| <=$$
$$\frac{5}{384} 21 \times 1^4 <= 2.7 * 10^{-1}$$

Sendo o resultado : $0.89 \pm 2.7 * 10^{-1}$

Para 2.6 :

$$|3.22917 - 3.32134| <=$$

$$\frac{5}{384} 21 \times 1^4 <= 2.7 * 10^{-1}$$

Sendo o resultado : $3.23 \pm 2.7 * 10^{-1}$

Alínea e

Partindo da análise dos resultados obtidos no calculo dos erros, podemos dizer que nos splines os erros são iguais para todos os pontos, na medida em que a formula a aplicar não necessita de algo particular de cada ponto.
Ao contrário do que acontece com o polinómio interpolador que cada ponto vai ter o seu erro específico.
Note-se s os majorantes dos erros são muito grandes, não sendo possível escrever da forma habitual os erros. Contudo é visível para os splines que para 0.1:

$$2.7 * 10^{-2} <= 7.1$$

. Bem como para 2.6:

$$9.2 * 10^{-2} <= 7.1$$

.

Para o polinómio interpolador também podemos observar que para 0.1:

$$3.7 * 10^{-3} <= 3.3 * 10^{1}$$

Tal como no ponto 2.6:

$$5.1 * 10^{-2} <= 8.5 * 10^{2}$$

Ou seja comparamos os majorantes com o erro efetivo e observamos que o majorante é maior.
Por fim, verificamos ainda que o majorante do erro do spline é menor do que o majorante do polinómio interpolador.

Conclusão

Em suma, conseguimos realizar todas as tarefas de forma clara. É de notar que usamos uma ferramenta que gera gráficos de forma a facilitar a interpretação dos resultados.
Além disso, para a construção dos splines e do polinómio interpolador tentamos usar a nossa máquina, isto é, implementamos o que conseguimos para nos facilitar os calculos.
É e realçar que tivemos alguma dificuldade na construção dos splines, uma vez que não estavamos a obter um sistema que garantisse a passagem pelos pontos considerados.

Referências

- Criação de gráficos/funções:
  - `https://www.desmos.com/calculator`
  - (último acesso em: Maio)
- Calculos auxiliares:
  - `https://www.symbolab.com/`
  - (último acesso em: Maio)
- Calculo de sistemas:
  - `http://www.calculadoraonline.com.br/sistemas-lineares`
  - (último acesso em: Maio)