TRABALHO 3

May 10, 2020

Análise Numérica (M2018)

Francisco Gonçalves - 201604505
Márcia Dias - 201704466

Departamento de Ciência de Computadores
Faculdade de Ciências de Universide do Porto

Contents

0.1 ReadMe . . . 2
0.2 Primeiro Exercício . . . 3
0.2.1 Alínea a) . . . 3
0.2.2 Alínea b) . . . 4
0.3 Segundo Exercício . . . 5
0.3.1 Alínea a) . . . 5
0.3.2 Alínea b) . . . 6
0.3.2.1 Polinómio Interpolador . . . 6
0.3.2.2 Spline Cúbico . . . 8
0.3.3 Alínea c) . . . 9
0.3.3.1 Representações Gráficas . . . 9
0.3.4 Alínea d) . . . 10
0.3.4.1 a) . . . 10
0.3.4.2 b) . . . 10
0.3.4.2.1 Polinómio Interpolador . . . 10
0.3.4.2.2 Spline Cúbico . . . 11
0.3.4.3 c) . . . 12
0.3.5 Alínea e) . . . 13

0.1 ReadMe

Este trabalho é composto pelos seguintes programas:

- *problem2a.py* - [Python]
- *problem2bd.c* - [C]
- *problem2bd.py* - [Python]

Para testar o programa *problem2bd.c* basta fazer:

1. *gcc problem2bd.c*
2. *./a.out*

Para testar os programas *problem2a.py* e *problem2bd.py* basta fazer:

- *python3 problem2a.py*
- *python3 problem2bd.py*

⚠ **NOTA** ⚠

Por favor assegure-se que instalou todos os módulos e bibliotecas necessárias ao bom funcionamento dos programas, especialmente para Python 3:

numpy - *mathplotlib*

Os programas aqui exibidos assim como os resultados obtidos podem ser consultados em detalhe aqui: `https://github.com/1Skkar1/Numerical-Analysis/tree/master/Trabalho3`

0.2 Primeiro Exercício

Na seguinte tabela apresenta-se o número de animais de uma determinada espécie, $p(t)$, medido em vários tempos t:

t (em anos)	0	5	10	15	20	25	30
$p(t)$ (em milhões)	100	89.5560	78.4905	67.2706	56.3897	46.2842	37.2687

0.2.1 Alínea a)

Calculem um valor aproximado de p(9), número de animais dessa espécie ao fim de 9 anos, por interpolação polinomial sobre os dados conhecidos, usando toda a informação fornecida:

Temos 7 pontos do tipo (t_i, p_i), logo o polinómio tem grau menor ou igual a $n = 6$.

$$I_0 = \frac{(t-5)(t-10)(t-15)(t-20)(t-25)(t-30)}{(0-5)(0-10)(0-15)(0-20)(0-25)(0-30)} \tag{1}$$

$$I_1 = \frac{(t-0)(t-10)(t-15)(t-20)(t-25)(t-30)}{(5-0)(5-10)(5-15)(5-20)(5-25)(5-30)} \tag{2}$$

$$I_2 = \frac{(t-0)(t-5)(t-15)(t-20)(t-25)(t-30)}{(10-0)(10-5)(10-15)(10-20)(10-25)(10-30)} \tag{3}$$

$$I_3 = \frac{(t-0)(t-15)(t-20)(t-25)(t-30)}{(15-0)(15-5)(15-10)(15-20)(15-25)(15-30)} \tag{4}$$

$$I_4 = \frac{(t-0)(t-5)(t-10)(t-15)(t-25)(t-30)}{(20-0)(20-5)(20-10)(20-15)(20-25)(20-30)} \tag{5}$$

$$I_5 = \frac{(t-0)(t-5)(t-10)(t-15)(t-20)(t-30)}{(25-0)(25-5)(25-10)(25-15)(25-20)(25-30)} \tag{6}$$

$$I_6 = \frac{(t-0)(t-5)(t-10)(t-15)(t-20)(t-25)}{(30-0)(30-5)(30-10)(30-15)(30-20)(30-25)} \tag{7}$$

$$P_6(t) = I_0 \cdot 100 + I_1 \cdot 89.5560 + I_2 \cdot 78.4905 + I_3 \cdot 67.2706 + I_4 \cdot 56.3897 + I_5 \cdot 46.2842 + I_6 \cdot 37.2687$$

Substituindo t por 9, obtivemos:

$P_6(9) \approx 80.73186813$

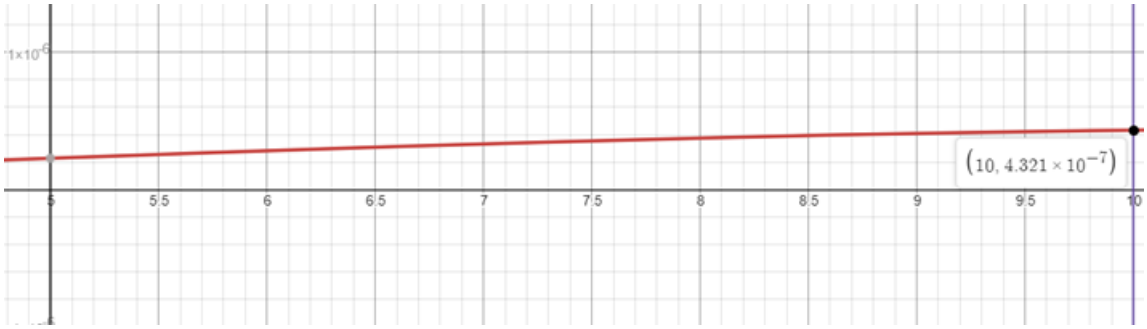
0.2.2 Alínea b)

Calculem um majorante do erro cometido sabendo que a população da espécie pode ser modelada por $p(t) = \frac{300}{2+e^{0.06t}}$

$$|\mathrm{p(t)} - \mathrm{p_6}(t)| \leq \frac{M}{(n+1)!}|(t-t_0)(t-t_1)(t-t_2)(t-t_3)(t-t_4)(t-t_5)(t-t_6)|$$

$$\mathrm{M} = \max_{t\in I}|p^{(7)}(t)|$$

Para cálculo do M, recorremos ao gráfico de $\frac{d_7}{dx_7}\frac{300}{2+e^{0.06x}}$, ou seja, a sétima derivada de $p(t)$. Obtivemos o seguinte gráfico:



graph1b: Representação gráfica da sétima derivada de $p(t)$ no intervalo $[5,10]$

Concluímos que o valor máximo é $4.321 \cdot 10^{-7}$.

$$|80.73182078 - 80.73186813| \leq \frac{4.321\cdot 10^{-7}}{7!}|9\cdot 4\cdot(-1)\cdot(-6)\cdot(-11)\cdot(-16)\cdot(-21)|$$

$$\iff 4.7\cdot 10^{-5} \leq 6.3\cdot 10^{-5}$$

Logo, o erro cometido é: $80.73182 \pm 6.3\cdot 10^{-5}$

0.3 Segundo Exercício

Considerem a função:

$$f(x) = e^{-\frac{1}{1+x^2}}, para - 4 \leq x \leq 4 \tag{8}$$

0.3.1 Alínea a)

Construam um conjunto de $n+1=7$ *pontos,* $(x_i, f_i = f(x_i))_i^n = 0$*, de abcissas* $x_i, i = 0, ..., n$ *igualmente espaçadas no intervalo* $[4,4]$*:*

A seguinte implementação permite construir o conjunto de pontos desejados:

createPoints( ) in *problem2a.py* (PYTHON)

```
import math as math
import numpy as np

def main():
   f = lambda x: np.power(math.e, -1/(1+x**2))
   nPoints = 7 # 7 for problem 2a) Or 8 for problem 2d) [change accordingly]
   createPoints(f, nPoints)

def createPoints(f, nPoints):
   xPoints = np.around(np.linspace(-4,4,nPoints),3)
   yPoints = []

   for i in xPoints:
      yPoints.append(round(f(i),3))

   points = zip(xPoints,yPoints)
   print ("CONJUNTO DE PONTOS: \n", tuple(points))

main()
```

Caso se deseje alterar o número de pontos a criar, como será posteriormente necessário para o exercício *2.d)*, basta alterar a variável ***nPoints*** para refletir a quantidade desejada.

Após executar a função ***createPoints***(), a variável ***xPoints*** cria (para este exercício) 7 pontos igualmente espaçados, no intervalo $[-4,4]$.

O loop da função permite calcular as imagens desses pontos relativamente à função f previamente definida.

x_i	-4	-2.667	-1.333	0	1.333	2.667	4
f_i	0.943	0.884	0.698	0.368	0.698	0.884	0.943

0.3.2 Alínea b)

Construam o polinómio interpolador, p, e o spline cúbico natural, s, de f naquele conjunto de pontos:

O mesmo programa *problem2bd.c* constrói o interpolador e o spline em simultâneo, imprimindo os resultados que permitem fazer as respetivas representações gráficas.

0.3.2.1 Polinómio Interpolador

Newton( ) / DiffTable( ) in *problem2b.c* (C)

```c
double Newton(int n, double x[], double diffTable[][10]){
    printf("P%d(x) = %f + ", n-1, diffTable[0][0]);
    for(int i = 1; i < n; i++) {
        printf("%f * ", diffTable[0][i]);
        for(int j = 0; j < i; j++) {
            if(j != i - 1)
                printf("(x - %f) * ", x[j]);
            else
                printf("(x - %f) ", x[j]);
        }
        if(i != n - 1)
            printf("+ ");
        printf("\n\t");
    }
    printf("\n");
}

void DiffTable(int n, double x[], double fx[], double diffTable[][10]) {
    for(int i = 0; i < n; i++) {
        diffTable[i][0] = fx[i];
    }
    for(int i = 1; i < n; i++) {
        for(int j = 0; j < n - i; j++) {
            diffTable[j][i] = (diffTable[j][i-1] - diffTable[j+1][i-1]) / (x[j] -
                x[i+j]);
        }
    }
}
```

⚠ **NOTA** ⚠

Por razões de formatação e estética foi adotada uma precisão de 3 casas decimais nos resultados aqui exibidos. Para melhores resultados, a precisão dos prints pode ser aumentada nos $printfs$ *em* $problem2bd.c$

x_i	$f(x_i)$	$f[,]$	$f[,,]$	$f[,,,]$	$f[,,,,]$	$f[,,,,,]$	$f[,,,,,,]$
-4	0.943	-0.044	-0.036	-0.001	0.011	-0.005	0.001
-2.667	0.884	-0.140	-0.041	0.057	-0.021	0.005	
-1.333	0.698	-0.248	0.186	-0.057	0.011		
0	0.368	0.248	-0.041	0.001			
1.333	0.698	0.140	-0.036				
2.667	0.884	0.044					
4	0.943						

$P_6(x) =$

$0.943 - 0.044 * (x+4) +$

$-0.036 * (x+4) * (x+2.667) +$

$-0.001 * (x+4) * (x+2.667) * (x+1.333) +$

$0.011 * (x+4) * (x+2.667) * (x+1.333) * (x-0) +$

$-0.005 * (x+4) * (x+2.667) * (x+1.333) * (x-0) * (x-1.333) +$

$0.001 * (x+4) * (x+2.667) * (x+1.333) * (x-0) * (x-1.333) * (x-2.667)$

De forma a obter o polinómio interpolador, foi usado o método de Newton, embora houvesse outras possibilidades, como o Método de Lagrange. No entanto, este método estima as raízes de uma dada função, escolhendo uma aproximação inicial. Posteriormente, calcula a reta tangente nesse ponto assim como a sua interseção com o eixo das abcissas, permitindo encontrar a melhor aproximação possível para a raíz. Este processo é repetido para os diversos pontos a considerar. A partir desta expressão final de $P_6(x)$ será possível representar graficamente o polinómio interpolador, demonstrado na alínea seguinte.

0.3.2.2 Spline Cúbico

CubicSpline() in *problem2bd.c* (C)

```c
void cubicSpline(int n, double xi[], double fxi[]) {
    double h[n], A[n], l[n+1], u[n+1], z[n+1], c[n+1], b[n], d[n];
    for(int i = 0; i < n; i++)
        h[i] = xi[i+1] - xi[i];
    for(int i = 1; i < n; i++)
        A[i] = 3 * (fxi[i+1] - fxi[i]) / h[i] - 3 * (fxi[i] - fxi[i-1]) / h[i-1];
    l[0] = 1;
    u[0] = 0;
    z[0] = 0;
    for (int i = 1; i < n; i++) {
        l[i] = 2 * (xi[i+1] - xi[i-1]) - h[i-1] * u[i-1];
        u[i] = h[i] / l[i];
        z[i] = (A[i] - h[i-1] * z[i-1]) / l[i];
    }
    l[n] = 1;
    z[n] = 0;
    c[n] = 0;
    for (int j = n - 1; j >= 0; j--) {
        c[j] = z[j] - u[j] * c[j+1];
        b[j] = (fxi[j+1] - fxi[j]) / h[j] - h[j] * (c[j+1] + 2 * c[j]) / 3;
        d[j] = (c[j+1] - c[j]) / (3 * h[j]);
    }
    printf("\nCubic Spline:\n");
    printf("%2s %8s %8s %8s %8s\n", "i", "x^3", "x^2", "x", "fxi");
    for (int i = 0; i < n; i++)
        printf("%2d %8.3f %8.3f %8.3f %8.3f\n", i, d[i], c[i], b[i], fxi[i]);
}
```

$$S(x) = \begin{cases} -4.3755 \cdot 10^{-3} \cdot x^3 - 5.2506 \cdot 10^{-2} \cdot x^2 - 2.4651 \cdot 10^{-1} \cdot x + 5.1702 \cdot 10^{-1}, & -4 \le x \le -2.667 \\ -3.1625 \cdot 10^{-2} \cdot x^3 - 2.7053 \cdot 10^{-1} \cdot x^2 - 8.2797 \cdot 10^{-1} \cdot x + 1.0589 \cdot 10^{-4}, & -2.667 \le x \le -1.333 \\ 1.2370 \cdot 10^{-1} \cdot x^3 + 3.5061 \cdot 10^{-1} \cdot x^2 + 6.2400 \cdot 10^{-64} \cdot x + 3.6800 \cdot 10^{-1}, & -1.333 \le x \le 0 \\ -1.2370 \cdot 10^{-1} \cdot x^3 + 3.5061 \cdot 10^{-1} \cdot x^2 + 6.2400 \cdot 10^{-64} \cdot x + 3.6800 \cdot 10^{-1}, & 0 \le x \le 1.333 \\ 3.1625 \cdot 10^{-2} \cdot x^3 - 2.7053 \cdot 10^{-1} \cdot x^2 + 8.2797 \cdot 10^{-1} \cdot x + 1.0589 \cdot 10^{-4}, & 1.333 \le x \le 2.667 \\ 4.3755 \cdot 10^{-3} \cdot x^3 - 5.2506 \cdot 10^{-2} \cdot x^2 + 2.4651 \cdot 10^{-1} \cdot x + 5.1702 \cdot 10^{-1}, & 2.667 \le x \le 4 \end{cases}$$

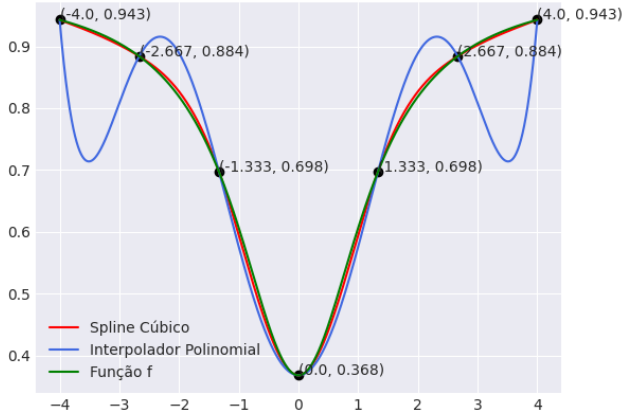
Relativamente à construção do spline cúbico, o programa permite resolver o sistema de equações de forma a obter a solução. Posteriormente, basta considerar cada função para o respetivo intervalo de tempo, obtendo a sua representação gráfica.

0.3.3 Alínea c)

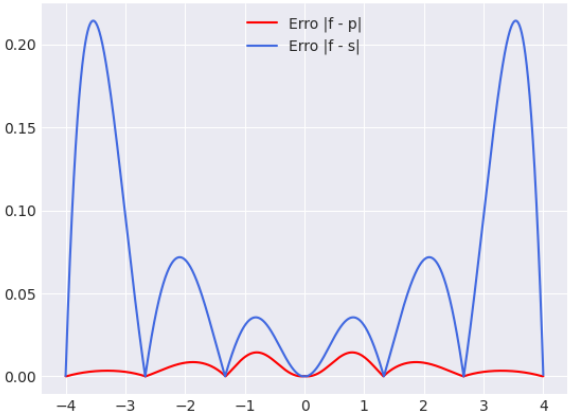
Comparem os gráficos das aproximações p e s com o de f no intervalo dado, e comparem os gráficos das funções erro $|f-p|$ e $|f-s|$:

0.3.3.1 Representações Gráficas

Os seguintes gráficos são obtidos ao executar o programa *problem2bd.py*, guardando as seguintes imagens na mesma localização da sua implementação. Foi recorrida à biblioteca *matplotlib* para desenhar os gráficos pedidos. Este programa será usado posteriormente para o exercício 2.*d*):



graph2b_Lines: Comparação gráfica entre o polinómio interpolador, o spline cúbico e a função original f, tendo em conta a distribuição de pontos



graph2b_Errors: Comparação gráfica entre as funções erro $|f-p$ e $|f-s|$

0.3.4 Alínea d)

Repitam as alíneas anteriores para $n+1=8$*:*

0.3.4.1 a)

Repetindo o processo do exercício 2.*a*) para obter um conjunto de 8 pontos no mesmo intervalo $[-4,4]$. O programa é o mesmo que o apresentado anteriormente, alterando apenas o valor da variável $nPoints$ para 8. Após correr o programa $problem2a.py$ com a alteração mencionada, obtêm-se os seguintes resultados:

x_i	-4	-2.857	-1.714	-0.571	0.571	1.714	2.857	4
f_i	0.943	0.897	0.776	0.470	0.470	0.776	0.897	0.943

0.3.4.2 b)

A implementação usada para resolver o exercício 2.*b*) permanece quase inalterada com a exceção de algumas variáveis. Dado que o programa é interativo, basta escolher a segunda opção para obter os resultados estimados:

0.3.4.2.1 Polinómio Interpolador

x_i	$f(x_i)$	$f[,]$	$f[,,]$	$f[,,,]$	$f[,,,,]$	$f[,,,,,]$	$f[,,,,,,]$	$f[,,,,,,,]$
-4	0.943	-0.040	-0.029	-0.012	0.015	-0.005	0.001	-0.000
-2.857	0.897	-0.106	-0.071	0.055	-0.012	-0.000	0.001	
-1.714	0.776	-0.268	0.117	-0.000	-0.012	0.005		
-0.571	0.470	-0.000	0.117	-0.055	0.015			
0.571	0.470	0.268	-0.071	0.012				
1.714	0.776	0.106	-0.029					
2.857	0.897	0.040						
4	0.943							

$P_7(x) =$

$0.943 - 0.040 * (x+4) +$

$-0.029 * (x+4) * (x+2.857) +$

$-0.012 * (x+4) * (x+2.857) * (x+1.714) +$

$0.015 * (x+4) * (x+2.857) * (x+1.714) * (x+0.571) +$

$-0.005 * (x+4) * (x+2.857) * (x+1.714) * (x+0.571) * (x-0.571) +$

$0.001 * (x+4) * (x+2.857) * (x+1.714) * (x+0.571) * (x-0.571) * (x-1.714) +$

$-0.000 * (x+4) * (x+2.857) * (x+1.714) * (x+0.571) * (x-0.571) * (x-1.714) * (x-2.857)$

0.3.4.2.2 Spline Cúbico

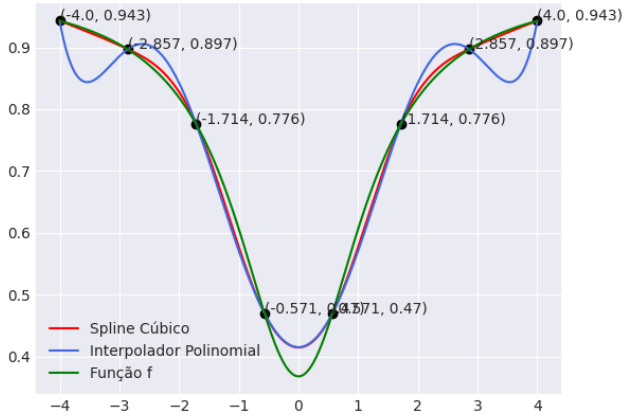
$$S(x) = \begin{cases} -1.8280 \cdot 10^{-3} \cdot x^3 - 2.1936 \cdot 10^{-2} \cdot x^2 - 1.2560 \cdot 10^{-1} \cdot x + 6.7458 \cdot 10^{-1}, & -4 \leq x \leq -2.857 \\ -4.1085 \cdot 10^{-2} \cdot x^3 - 3.5841 \cdot 10^{-1} \cdot x^2 - 1.0869 \cdot x - 2.4091 \cdot 10^{-1}, & -2.857 \leq x \leq -1.714 \\ 9.2506 \cdot 10^{-2} \cdot x^3 + 3.2852 \cdot 10^{-1} \cdot x^2 + 9.0482 \cdot 10^{-2} \cdot x + 4.3178 \cdot 10^{-1}, & -1.714 \leq x \leq -0.571 \\ 5.1180 \cdot 10^{-64} \cdot x^3 + 1.7005 \cdot 10^{-1} \cdot x^2 - 4.7770 \cdot 10^{-64} \cdot x + 4.1456 \cdot 10^{-1}, & -0.571 \leq x \leq 0.571 \\ -9.2506 \cdot 10^{-2} \cdot x^3 + 3.2852 \cdot 10^{-1} \cdot x^2 - 9.0482 \cdot 10^{-2} \cdot x + 4.3178 \cdot 10^{-1}, & 0.571 \leq x \leq 1.714 \\ 4.1085 \cdot 10^{-2} \cdot x^3 - 3.5841 \cdot 10^{-1} \cdot x^2 + 1.0869 \cdot x - 2.4091 \cdot 10^{-1}, & 1.714 \leq x \leq 2.857 \\ 1.8280 \cdot 10^{-3} \cdot x^3 - 2.1936 \cdot 10^{-2} \cdot x^2 + 1.2560 \cdot 10^{-1} \cdot x + 6.7458 \cdot 10^{-1}, & 2.857 \leq x \leq 4 \end{cases}$$

main( ) in *problem2bd.c* (C)

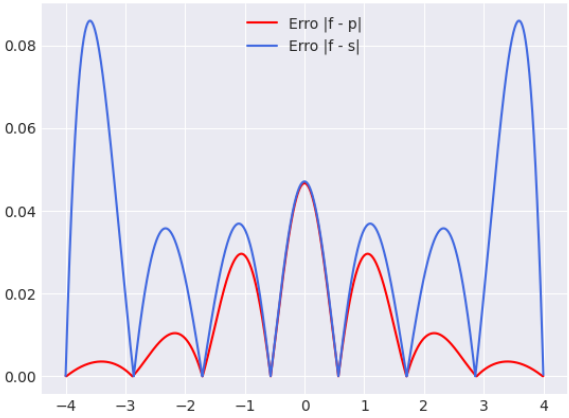
```
int main() {
    int option, nPoints;
    double xb[] = {-4,-2.667,-1.333,0,1.333,2.667,4};
    double fxb[] = {0.943,0.884,0.698,0.368,0.698,0.884,0.943};
    double xd[] = {-4,-2.857,-1.714,-0.571,0.571,1.714,2.857,4};
    double fxd[] = {0.943,0.897,0.776,0.470,0.470,0.776,0.897,0.943};
    double diffTable[10][10];

    printf("Exercise to be solved:\n1) Exercise 2.b) [nPoints = 7]\n2) Exercise 2.d)
        [nPoints = 8]\n");
    scanf("%d", &option);
    if (option == 1) {    // Exercise 2.b)
        int nPoints = 7;
        splitDiffTable(nPoints, xb, fxb, diffTable);
        printDiffTable(nPoints, xb, diffTable);
        Newton(nPoints, xb, diffTable);
        cubicSpline(nPoints, xb, fxb);
        }
    else if (option == 2) { // Exercise 2.d)
        int nPoints = 8;
        splitDiffTable(nPoints, xd, fxd, diffTable);
        printDiffTable(nPoints, xd, diffTable);
        Newton(nPoints, xd, diffTable);
        cubicSpline(nPoints, xd, fxd);
    }
    return 0;
}
```

0.3.4.3 c)

Para ilustrar um novo gráfico representativo da nova alínea *d*) basta escolher a segunda opção ao correr o programa *problem2bd.py*. Visto que este é interativo não é preciso modificar nada no código do mesmo para obter os seguintes gráficos:



graph2d_Lines: Comparação gráfica entre o polinómio interpolador, o spline cúbico e a função original f, tendo em conta a distribuição de pontos



graph2d_Errors: Comparação gráfica entre as funções erro $|f - p$ e $|f - s|$

0.3.5 Alínea e)

Observem e comentem os resultados obtidos:

Comparando primeiro as funções obtidas do spline cúbico com as do interpolador polinomial e tendo em consideração a função original f, é possível verificar, tanto para $n = 7$ como para $n = 8$ que o spline cúbico consegue obter valores mais aproximados daqueles de f. Contrariamente, a função do interpolador polinomial afasta-se mais da função original enquanto se desloca de um ponto para o próximo.

Estas observações são comprovadas pelos gráficos das funções erro $|f - p|$ e $|f - s|$ mostrando que os valores da segunda, que aborda a diferença absoluta entre a função original e o spline cúbico natural, se aproximam mais de 0 durante a a grande maioria do intervalo. Por outro lado, a função que representa a diferença absoluta entre f e o interpolador polinomial tem picos de valores mais acentuados, que se referem aos momentos entre os diferentes pontos em que a função se afasta.

Sabe-se que para as alíneas *b*) e *d*), o polinómio interpolador tem grau 6 e 7, respetivamente. Isso causa uma oscilação justificadamente elevada entre os diversos pontos de f, levando a erros de arredondamento inevitáveis.

Quanto ao spline cúbico natural, trata-se de um polinómio de grau inferior ou igual a 3 em cada ramo e, como tal, apresenta oscilações menos protuberadas, diminuindo os seus erros de aproximação.