Aproximação por Splines e Polinómios Interpoladores

FCUP
Análise Numérica (M2018) 2018/2019
Trabalho de Grupo 3
Ângelo Gomes – 201703990 – MIERSI
Eduardo Morgado – 201706894 – MIERSI
Simão Cardoso – 201604595 – MIERSI
Sónia Rocha – 201704679 – MIERSI

1. Considerações Iniciais:

Na realização deste trabalho, utilizamos como linguagem de implementação Python(3.7.2), tendo dividido o trabalho em 4 grandes classes, *Main.py*, *Spline.py* (classe de cálculo de spline), *NewtonDiferences.py* (classe de cálculo de diferenças divididas e aplicação do método de Newton para criação do polinómio interpolador) e *Interpolation.py* (classe de cálculo do polinómio interpolador).

Todos estes programas podem ser encontrados no Github no repositório da equipa, https://github.com/thejoblessducks/Trabalho3AN, iremos mostrar as classes em geral bem como as funções dos exercícios 1 e 2, no entanto, recomendamos que consulte o repositório, uma vez que, será mais fácil analisar o código e torna possível o seu teste.

Para algumas das classes utilizamos bibliotecas específicas do Python para facilitar a resolução do problema, essas bibliotecas foram, *matplotlib*, *numpy* e *scipy*, caso queira testar o problema, estas bibliotecas devem estar instaladas, a instalação poderá ser feita em linha de comandos UNIX/Windows/Mac através dos comandos: *pip install matplotlib*, *pip install numpy* e *pip install scipy* respetivamente.

As *Figuras 1*, *2*, *3* e *4-6* apresentam os códigos de *Spline.py*, *NewtonDiferences.py*, *Interpolation.py* e *Main.py* respetivamente.

É também importante referir que no Python os dados são tratados em formato *double*, ou seja, todos os resultados, das tabelas, do cálculo dos splines e dos erros estará em formato *double* sendo assim, por uma questão de facilidade de visualização, iremos apresentar os resultados, com 4 casas decimas (apesar de poder existir um certo erro quando comparado com os resultados do programa em si), isto se a tabela de dados permitir. Todos os resultados, desde a tabela de pontos, os gráficos, erros e os sistemas de equações de spline são fornecidos pelo programa.

```python
import numpy as np
from scipy.interpolate import CubicSpline as cs

'''--------------------------------------------------------------------------
Class to represent a normal spline
    Calculates the natural cubic spline
    Calculates the spline aproximation for a given value
    Displays the equations for each composing spline
--------------------------------------------------------------------------'''
class Spline():
    def __init__(self,x_points,y_points):
        self.x_points = x_points
        self.y_points = y_points
    def buildNormalSpline(self):
        x = self.x_points
        y = self.y_points
        spline = cs(x,y,bc_type='natural')
        return spline
    def showEquations(self,spline=None):
        if not spline:
            spline = self.buildNormalSpline()
        x = self.x_points
        coef = spline.c
        for i in range(len(x)-1):
            val = "{:0.4f}".format(x[i])
            s = "S"+str(i)+"("+val+"<=x<="+"{:0.4f}".format(x[i+1])+") = "
            a = "{:0.4f}".format(coef.item(3,i))
            b = "{:0.4f}".format(coef.item(2,i))
            c = "{:0.4f}".format(coef.item(1,i))
            d = "{:0.4f}".format(coef.item(0,i))

            s2 = a+" + "+b+"(x-"+val+") + "+c+"(x-"+val+")^2 + "+d+"(x-"+val+")^3;"
            print( s + s2)
        return
    def calc(self,value,spline=None):
        if not spline:
            spline = self.buildNormalSpline()
        return spline(value)
```

Figura 1-Spline.py

```
from __future__ import division
import numpy as np
'''-------------------------------------------------------------------------
Newton diferences class
    Calculates the divided diferences
    Calculates the polinomial aproximation for a value
-------------------------------------------------------------------------'''
class NewtonDiferences():
    def __init__(self,x,y):
        self.x = x
        self.y = y
    def dividedDiferences(self):
        #Calculates all the divided diferences for the list of points
        x = self.x
        y = self.y
        n = len(x)
        a = []
        for i in range(n):
            a.append(y[i])
        for j in range(1,n):
            for i in range(n-1,j-1,-1):
                a[i] = (a[i]-a[i-1])/(x[i]-x[i-j])

        return np.array(a)
    def calInterpolation(self,a,x,value_to_interpolate):
        #given a value_to_interpolate aproximates the value through the polinomial
        n = len(a)-1
        tmp = a[n]
        for i in range(1,n+1):
            tmp = tmp*(value_to_interpolate-x[n-i])+a[n-i]
        return tmp
```

Figura 2- NewtonDiferences.py

```
import numpy as np
import NewtonDiferences as ndiferences
'''-------------------------------------------------------------------------
Interpolation class
    Calculates the interpolating polinomial using newton method
    Calculates the aproximation of a certain value
-------------------------------------------------------------------------'''
class Interpolation():
    def __init__(self,x,y):
        self.x = x
        self.y = y #table images of x
        #calculate the list of divided diferences
        newton = ndiferences.NewtonDiferences(x,y)
        divided = newton.dividedDiferences()
        self.divided = divided
        self.newton = newton
    def interpolateData(self):
        divided = self.divided
        newton = self.newton
        x = self.x
        a = np.arange(x[0],x[-1]+0.0001,0.0001)
        y = []
        for i in a:
            #calculates the aproximation for each value
            y.append(newton.calInterpolation(divided,x,i))
        return a,y
    def calc(self,val):
        #given a value of x, val, calculates its aproximation
        divided = self.divided
        newton = self.newton
        x = self.x
        return newton.calInterpolation(divided,x,val)
```

Figura 3- Interpolation.py

```
from __future__ import division
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import fminbound as fmin

#imports for auxiliar classes
import Spline as spl
import Interpolation as inter

'''functions, f, f(2),f(4) f(9)'''
f = lambda x: (4*np.power(x,2)+np.sin(9*x))
f2 = lambda x: (8-8*np.cos(9*x))
f4 = lambda x: 6561*np.cos(9*x)
f9 = lambda x: -386239509*np.sin(9*x)

def graphicTable(x,y,func=False):
    #Spline and Interpolating polinomial calculation
    #initiates classes
    data = spl.Spline(x,y)
    data2 = inter.Interpolation(x,y)

    spline = data.buildNormalSpline()   #calculates the spline for the dataset
    data2_x,data2_y= data2.interpolateData() #calculates the interpolating pol
    xs = np.arange(x[0],x[-1]+0.0001,0.0001)

    plt.plot(x,y,"o",label="Data")
    if func:
        #displays the actual function f
        plt.plot(xs,f(xs),label='True Value')
    plt.plot(xs,spline(xs),label="Spline Trace")
    plt.plot(data2_x,data2_y,label="Interpolation")
    plt.legend(loc='lower left',ncol=2)
    plt.grid()
    plt.show()

    return data,spline,data2

def pointSet(lower,upper,n):
    #Creates the liniar spaced values in [lower,upper]
    values = np.linspace(lower,upper,num=n,endpoint=True)
    a = []
    for val in values:
        a.append(f(val))
    return values,np.array(a)
```

Figura 4- Main.py

```
def maxH(x):
    #Calculates the max h value in the data_set
    max_h = None
    for i in range(len(x)-1):
        h = x[i+1]-x[i]
        if not max_h:
            max_h = h
        max_h = max(max_h,h)
    return max_h

def M(lower,upper,interpolation=False):
    #Calculates the maximizer  of f4 or f9 in [lower,upper]
    if interpolation:
        return fmin(lambda x : -f9(x),lower,upper)
    return fmin(lambda x : -f4(x),lower,upper)

def interpolationError(x,value,lower,upper):
    #Calculates the major error in interpolating polinomial
    m = M(lower,upper,interpolation=True)
    m = abs(f9(m))
    print("     Max|f9(x)|="+str(m)+" ,"+str(lower)+"<=x<="+str(upper))
    n1 = len(x)+1
    b = np.arange(1,n1)
    fac = b.prod()
    mult = 1
    for i in x:
        mult *= (value-i)
    return (m/fac)*abs(mult)
def splineError(x,lower,upper):
    #Calculates the major error in the spline
    m = M(lower,upper,interpolation=False)
    m = abs(f4(m))
    h = maxH(x)
    print("     Max|f4(x)|="+str(m)+" ,"+str(lower)+"<=x<="+str(upper))
    print("     Max(h)="+str(h))
    return (5/384)*m*np.power(h,4)


def ex1():
    #exercise 1
    x = np.array([0,1,2,2.5,3,4])
    y = np.array([1.4,0.6,1.0,0.6,0.6,1.0])
    spline,natural,interpolator = graphicTable(x,y,func=False)
    print("Spline equations:")
    spline.showEquations(natural)
```

Figura 5- Main.py

```
def ex2():
    #exercise 2
    x,y = pointSet(-1,1,9)
    spline,natural, interpolator = graphicTable(x,y,func=True)

    print("Data Table:") #Show points and images in table
    for i in range(len(x)):
        print("("+str(x[i])+","+str(y[i])+")")

    print("\n\nSpline equations:") #Show Spline equations
    spline.showEquations(natural)
    print("\n\nx=0.3: ")
    print("     S(0.3)="+str(spline.calc(0.3,spline=natural)))
    print("     |f(0.3)-S(0.3)|<="+str(splineError(x,-1,1)))

    print("\n     p(0.3)="+str(interpolator.calc(0.3)))
    print("     |f(0.3)-p(0.3)|<="+str(interpolationError(x,0.3,-1,1)))

    print("\n\nx=0.83:")
    print("     S(0.83)="+str(spline.calc(0.83,spline=natural)))
    print("     |f(0.83)-S(0.83)|<="+str(splineError(x,-1,1)))

    print("\n     p(0.83)="+str(interpolator.calc(0.83)))
    print("     |f(0.83)-p(0.83)|<="+str(interpolationError(x,0.83,-1,1)))

# Starter-------------------------------------------------------------------------
print("Exercise 1:")
ex1()
print("\nExercise2:")
ex2()
```

Figura 6- Main.py

2. Exercício 2:

2.1. Alínea a:

Para esta secção iremos aplicar as classes representadas anteriormente, no ficheiro *Main.py* iremos chamar a função *ex1()*, apresentada na *Figura 5*, para esta função, a tabela considerada está representada na *Figura 7*, a *Figura* 8 apresenta o sistema de equações do spline cúbico natural gerado.

x_i	0	1	2	2.5	3	4
f_i	1.4	0.6	1.0	0.6	0.6	1.0

Figura 7- Tabela de valores para exercício 2.a

$$S(x) = \begin{cases} S_0(x) = 1.40 - 1.2614x + 0.4614x^3 & \mathbf{0 \le x \le 1} \\ S_1(x) = 0.60 + 0.1228(x-1) + 1.3842(x-1)^2 + 1.1071(x-1)^3 & \mathbf{1 \le x \le 2} \\ S_2(x) = 1 - 0.4299(x-2) - 1.9369(x-2)^2 + 2.3934(x-2)^3 & \mathbf{2 \le x \le 2.5} \\ S_3(x) = 0.6 - 0.5718(x-2.5) + 1.6531(x-2.5)^2 + 1.0191(x-2.5)^3 & \mathbf{2.5 \le x \le 3} \\ S_4(x) = 0.6 + 0.3170(x-3) + 0.1245(x-3)^2 + 0.0415(x-3)^3 & \mathbf{3 \le x \le 4} \end{cases}$$

Figura 8- Sistema de equações para exercício 2.a)

A *Figura 9* apresenta os gráficos do spline cúbico natural e do polinómio interpolador para a tabela (azul estão os pontos da tabela, x, a linha a verde representa o polinómio interpolador e a linha a laranja, o spline).

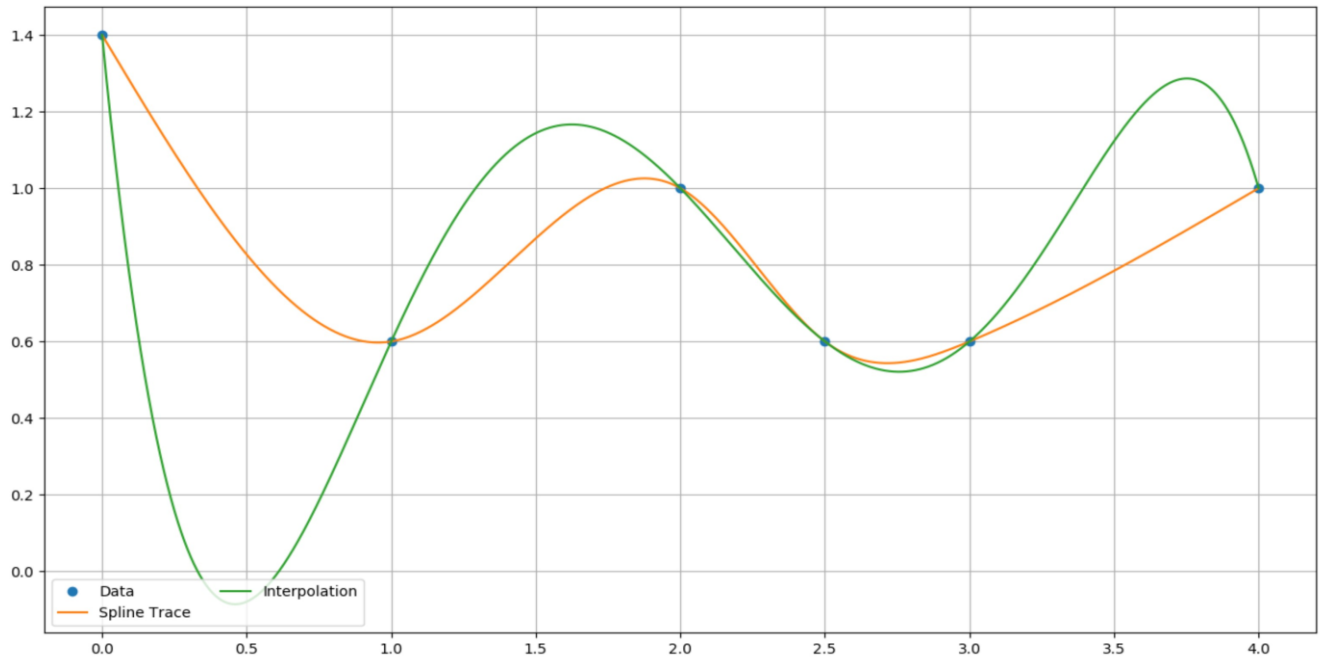

Figura 9- Gráficos de spline e polinómio interpolador para exercício 2.a)

Para este exercício não existe uma forma de comparar os gráficos quanto ao quão próximo estes aproximam a função $f(x)$, no entanto, a partir da observação dos gráficos, podemos concluir que o spline apresenta um gráfico mais suave para a representação de $f(x)$, observamos também que existe uma grande discrepância de valores entre os gráficos do spline e do polinómio, são poucos os pontos onde os gráficos se intersectam (quase todos os pontos, são pontos da tabela). Sem a função original torna-se difícil escolher qual dos métodos aproxima melhor a função, no entanto, a interpolação polinomial parece aproximar os dados de forma mais drástica e o spline uma aproximação mais estável.

2.2. Alínea b: $\boldsymbol{f(x) = 4x^2 + sin(9x)}$

Para esta secção iremos aplicar as classes representadas anteriormente, no ficheiro *Main.py* iremos chamar a função *ex2()*, apresentada na *Figura 5*, para esta função, a tabela considerada está

representada na *Figura 10* (tabela de pontos de abcissas igualmente espaçadas no intervalo [-1,1]), a *Figura* 11 apresenta o sistema de equações do spline cúbico natural gerado. É importante voltar a referir que os valores apresentados nas tabelas não são os valores fornecidos pelo programa, no programa os valores apresentam 16 casas decimais, no entanto, para uma facilidade de leitura, iremos apresentar todos os valores com 4 casas decimais apenas.

x_i	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1
f_i	3.5879	1.8000	1.9775	-0.5281	0	1.0281	0.0225	2.700	4.4121

Figura 10-Tabela de valores para exercício 2.b) i

$$S(x) = \begin{cases} S_0(x) = 3.5875 - 10.2350(\mathrm{x}+1) + 49.3330(x+1)^3 \ \boldsymbol{x} \in [-\mathbf{1}, -\mathbf{0.75}] \\ S_1(x) = 1.8959 - 0.9851(x+0.75) + 36.9998(x+0.75)^2 - 120.8732(x+0.75)^3 \ \boldsymbol{x} \in [-\mathbf{0.75}, -\mathbf{0.5}] \\ S_2(x) = 1.9775 - 5.1489(x+0.5) - 53.6551(x+0.5)^2 + 136.6444(x+0.5)^3 \ \boldsymbol{x} \in [-\mathbf{0.5}, -\mathbf{0.25}] \\ S_3(x) = -0.5281 - 6.3556(x+0.25) + 48.8282(x+0.25)^2 - 59.8259(x+0.25)^3 \ \boldsymbol{x} \in [-\mathbf{0.25}, \mathbf{0}] \\ S_4(x) = 6.8411x + 3.9588x^2 - 59.4960x^3 \ \boldsymbol{x} \in [\mathbf{0}, \mathbf{0.25}] \\ S_5(x) = 1.0281 - 2.3350(x-0.25) - 40.6633(x-0.25)^2 + 135.6548(x-0.25)^3 \ \boldsymbol{x} \in [\mathbf{0.25}, \mathbf{0.5}] \\ S_6(x) = 0.0225 + 2.7686(x-0.5) + 61.0778(x-0.5)^2 - 117.2443(x-0.50)^3 \ \boldsymbol{x} \in [\mathbf{0.5}, \mathbf{0.75}] \\ S_7(x) = 2.7000 + 11.3242(x-0.75) - 26.8554(x-0.75)^2 + 35.8073(x-0.75)^3 \ \boldsymbol{x} \in [\mathbf{0.75}, \mathbf{1}] \end{cases}$$

Figura 11-Sistema de Equações de Spline Cúbico Natural para tabela da Figura 10

A *Figura 12* apresenta os gráficos do spline cúbico natural do polinómio interpolador para a tabela e da função (azul estão os pontos da tabela, x, a linha a laranja representa a função $f(x)$, a vermelho está representado o polinómio interpolador e a linha a verde, o spline cúbico natural).

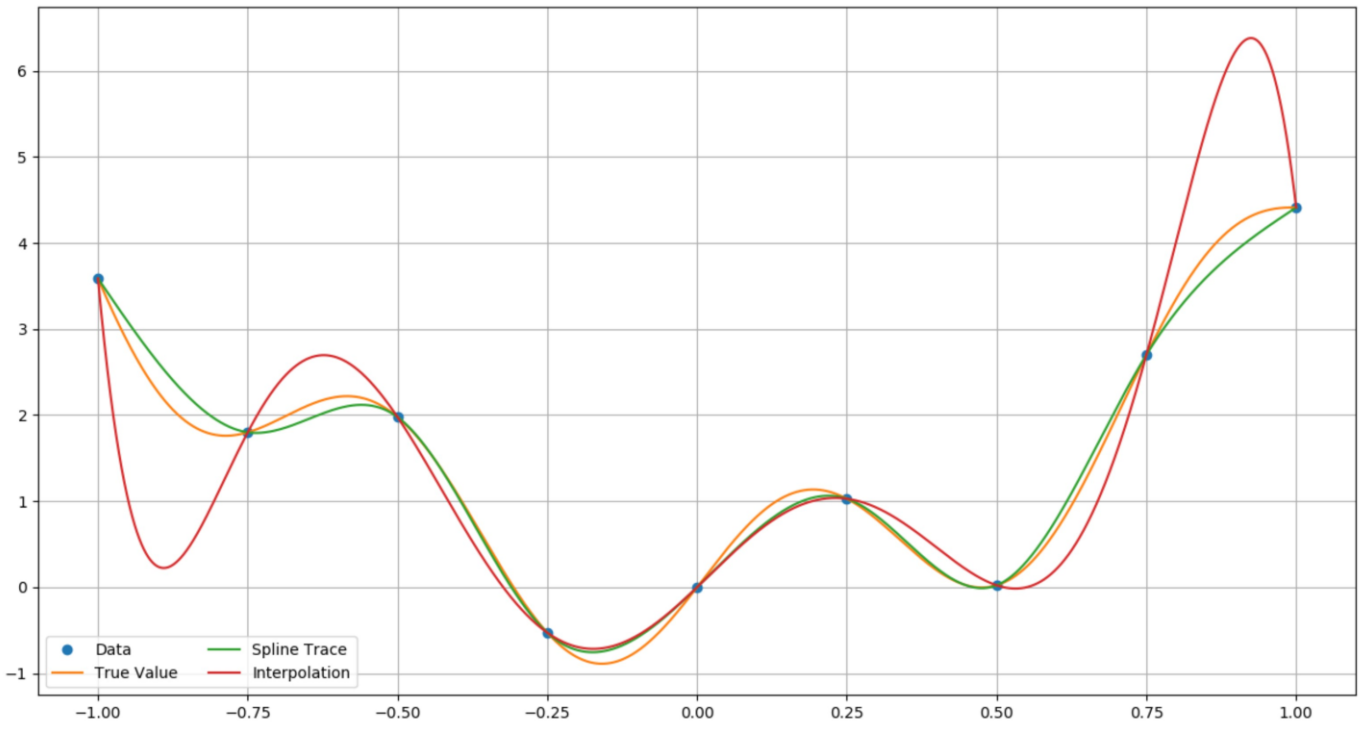

Figura 12-Gráfico para o exercício 2.b) ii intervalo [-1,1]

Numa primeira observação podemos concluir que o spline segue mais proximamente o traço de $f(x)$, já o polinómio interpolador apresenta maiores discrepâncias, especialmente para valores próximos dos extremos do intervalo [-1,1], esta diferença é normal, uma vez que, o polinómio interpolador é um polinómio que aproxima toda a função num intervalo, enquanto que o spline é composto por vários polinómios interpoladores num intervalo, ou seja, o spline vai adaptar-se melhor aos dados, daí seguir melhor o traço da função que está a ser aproximada.

No entanto, tal como a *Figura 13* mostra, existem várias instâncias onde o polinómio interpolador aproxima melhor os dados que o spline.

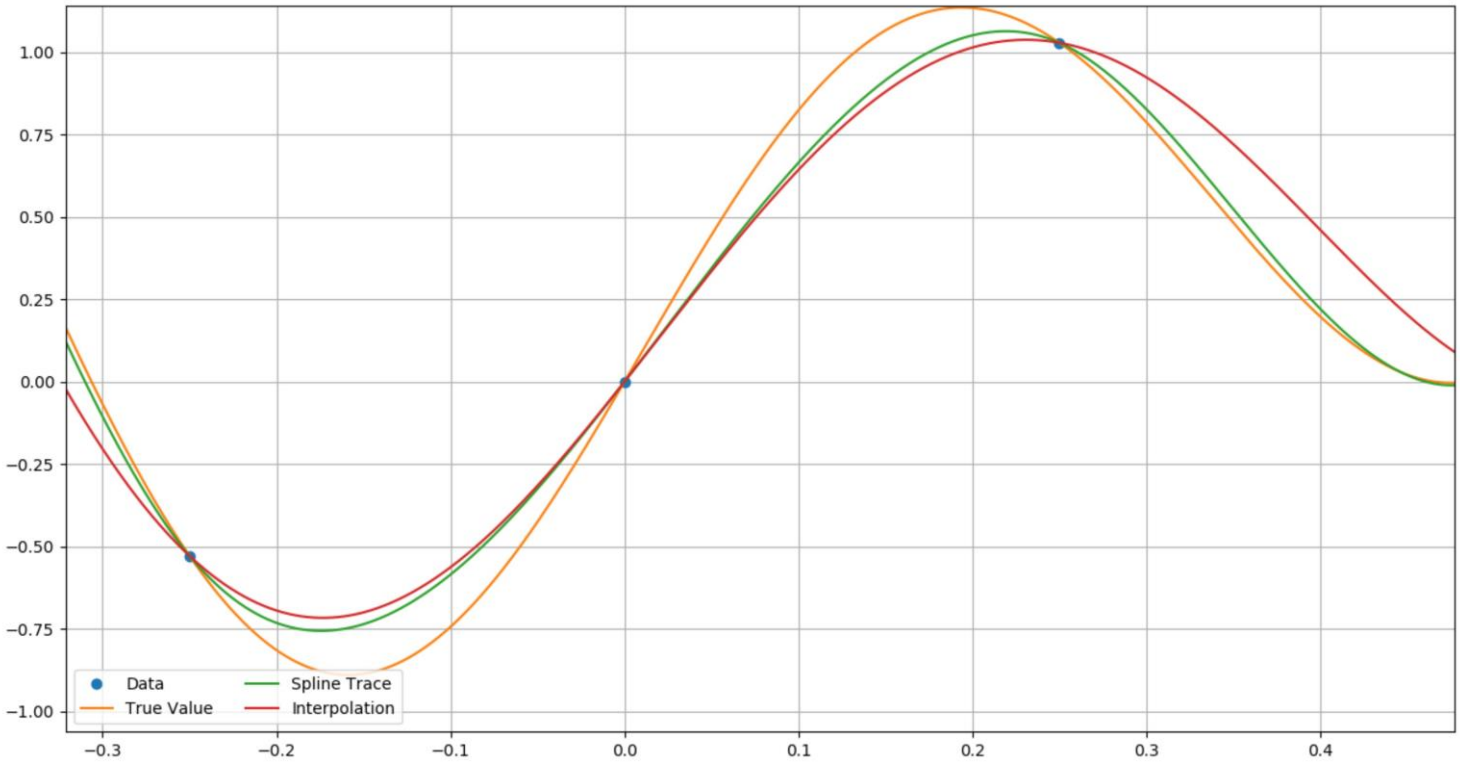

Figura 13- Gráfico para exercício 2.b) ii intervalo [-0.3,0.4]

A seguir apresentamos os valores de $f(0.3)$ e $f(0.83)$ obtidos por aproximação por polinómio interpolador e por spline cúbico natural. Mais uma vez, os valores fornecidos pelo programa têm 16 casas decimais, no entanto, para facilitar a apresentação e, uma vez que, os dados na tabela já restringem as casas decimais, iremos apresentar os valores com 4 casas decimais.

	Aproximação por spline cúbico natural	Aproximação por polinómio interpolador
$f(0.3)$	0.8266	0.9233
$f(0.83)$	3. 4524	4.8342

Figura 14- Valores de exercício 2.b.iii)

Para majorar o erro do polinómio interpolador, podemos aplicar a seguinte fórmula:

$$|f(x) - p(x)| \leq \frac{M}{(n+1)!} \cdot |\prod_{n+1}(x)|, \qquad \prod_{n+1}(x) = (x - x_0) \dots (x - x_n),$$
$$M = max_{x \in [a,b]} |f^{n+1}(x)|$$

Para este problema $n + 1 = 9$, logo $f^9(x) = -386239509 \cdot \sin(9x)$ e $M = 386239509$ (o valor de M também é retornado pelo programa).

Para majorar o erro do spline cúbico natural, uma vez que $f(x) \in C^4[-1,1]$, podemos aplicar a seguinte fórmula: $|f(x) - S(x)| \leq \frac{5}{384} \cdot Mh^4$, onde $M = max_{x \in [-1,1]} |f^4(x)|$ e $h = \max h_i,\ i = 1(1)n$, $f^4(x) = 6561\cos(9x)$, para este intervalo ([-1,1]) $M = 6561$ e $h = 0.25$. A *Figura 15* apresenta os valores majorados dos erros.

	Erro por spline cúbico natural	Erro por polinómio interpolador
$f(0.3)$	$3.3 * 10^{-1}$	$6.0 * 10^{-1}$
$f(0.83)$	$3.3 * 10^{-1}$	$0.96 * 10^{1}$

Figura 15- Erros majorados

Pelo que podemos observar, em geral a aproximação por spline apresenta erros mais baixos em comparação com a aproximação por polinómio interpolador, para os dois valores, 0.3 e 0.83, o erro do spline mantêm-se enquanto que o do polinómio interpolador cresce.

Se observarmos o gráfico do polinómio interpolador da *Figura* 12, podemos observar que as aproximações pelo polinómio interpolador vão sendo menos precisas para valores próximos dos extremos do intervalo]-1,1[, essa falta de precisão é também observável no erro da aproximação de 0.83 pelo polinómio, no entanto, o mesmo não acontece para o spline, este mantém uma certa estabilidade de precisão ao longo de todo o intervalo. A partir da *Figura 13*, podemos verificar que, para o ponto 0.3, tanto o spline como o polinómio interpolador estão muito próximos de f(0.3).

Sendo assim, considerando todos os dados, podemos concluir que, quanto maior for o conjunto de pontos da tabela de valores, melhor será a aproximação de f pelo polinómio interpolador, no entanto, esta observação não acontece para os splines, estes mantêm uma estabilidade/suavidade de precisão, uma vez que, para cada dois pontos igualmente distantes, existe um polinómio que os interpola, dessa forma, o spline irá enquadrar melhor cada sub-intervalo (irá adaptar-se melhor ao conjunto de dados).

Podemos então concluir que a aproximação por spline cúbico natural aproxima melhor a função da tabela de pontos