implementacoes

Fábio José Dos Santos Sousa

August 5, 2020

1 Implemetações de Matemática Computacional

2 Polinômio de Lagrange

[1]:
```
#!pip3 install PTable
#!pip install --upgrade
#!pip install sympy
import sympy

import numpy as np
import math
import mpmath as mp
from scipy.integrate import quad
import sys
from scipy import integrate

from prettytable import PrettyTable
```

[2]:
```
def polinomio_lagrange(X, Y):

    if len(X) != len(Y): raise ValueError("Dimensões diferentes em X e Y.")

    pares = list(zip(X, Y))
    pares.sort(key = lambda x: x[0])
    X, Y  = zip(*pares)
    def polinomio(x):
        L=[]
        poli=0
        for i in range(len(X)):
            numerador=1
            denominador=1
            for j in range(len(X)):
                if j != i:
                    numerador=numerador*(x-X[j])
                    denominador=denominador*(X[i]-X[j])
            L.append(numerador/denominador)
```

```
        for index in range(len(X)):
            poli+=(Y[index]*L[index])
        return poli
    return polinomio


if __name__ == '__main__':

    X=[-2,0,4]
    Y=[2,-2,1]
    polinomio=polinomio_lagrange(X,Y)
    p=polinomio(3)
    print('Solução numerica',p)
```

```
Solução numerica -1.125
```

3 Polinômio de Taylor

[3]:

```
def derivada(f, h = 0.01):
      def _(x):
        return (f(x + h) - f(x))/h
      return _


def polinomio_taylor(f, x0, n):

    aux=0
    aux2=0
    def polinomio(x):
        for i in range(n):
            if i==0:
                aux=f(x0)
                f2=f
            else:
                aux2=derivada(f2)
                f2=aux2
                aux3=aux2(x0)
                aux= aux+(((aux3*((x-x0)**i)))/math.factorial(i))
        return aux
    return polinomio

if __name__ == '__main__':

    f=lambda x: math.tan(x)
    p=polinomio_taylor(f,0,4)
    x=0.30
```

```
    print('Solução numerica',p(x))
```

```
Solução numerica 0.3099191848184239
```

4 Método da Posição Falsa

[31]:
```
# Função
def f(x):
    return math.exp(-x**2) - math.cos(x)
def posFalsa(f,a,b,eps=10**(-4)):
    #criando tabela
    xt = PrettyTable(["n","Posição falsa","a","b"])
    xt.align["n"] = "l"
    xt.align["Newton"] = "l"
    xt.align["a"] = "l"
    xt.align["b"] = "l"
    # Número de Interações
    #função para estimativa do número de iterações
    #link ---> https://www1.univap.br/spilling/CN/CN_Capt2.pdf    #baseado nessa␣
↪formula#
    inter = lambda a,b,erro: 1 + int((np.log(b-a)-np.log(erro))//np.log(2))
    # Pontos iniciais e erro

    k = inter(a,b,eps)
    for i in range(k):
        x = (a*f(b)-b*f(a))/(f(b)-f(a))
        xt.add_row([i,x,a,b])
        if np.abs(f(x)) >= eps:
            if f(a)*f(x)<0:
                b = x
            else:
                a = x
        else:
            break
        if abs(b - a) <= eps:
            break

    return xt
print(posFalsa(f,1,2))
```

```
+---+--------------------+--------------------+---+
| n |   Posição falsa    | a                  | b |
+---+--------------------+--------------------+---+
| 0 | 1.2841111051702645 | 1                  | 2 |
| 1 | 1.4075491905404016 | 1.2841111051702645 | 2 |
```

```
| 2 | 1.4393199026619496 | 1.4075491905404016 | 2 |
| 3 | 1.4458493491370066 | 1.4393199026619496 | 2 |
| 4 | 1.4471147784471927 | 1.4458493491370066 | 2 |
| 5 | 1.4473570678005703 | 1.4471147784471927 | 2 |
+---+--------------------+--------------------+---+
```

5 Método de Newton

$f(x) = x - 9x + 3$

solução numerica : 0.3376089559653128

Número de interações : 3

$x_0 = 0.5$

Tolerancia = $1x10$

[52]:
```python
def f(x):
    return x**2 - 4
def newton(f,x0,erro=0.0001, n=100):
    #Tabela
    xt = PrettyTable(["n","Newton","Xi"])
    xt.align["n"] = "l"
    xt.align["Newton"] = "l"
    xt.align["Xi"] = "l"



    #mp.dps = 15; mp.pretty = True



    for i in np.arange(n):
        x = x0-(float(f(x0)/mp.diff(f,x0)))
        valor_f = f(x0)
        valor_dx = mp.diff(f,x0)

        xt.add_row([i,x,x0])

        if np.abs(float(valor_f/valor_dx))<erro:
            break
        x0 = x
    return xt
print(newton(f,3))
```

```
+---+--------------------+--------------------+
| n | Newton             | Xi                 |
+---+--------------------+--------------------+
| 0 | 2.1666666666666665 | 3                  |
```

```
| 1 | 2.0064102564102564 | 2.1666666666666665 |
| 2 | 2.0000102400262145 | 2.0064102564102564 |
| 3 | 2.000000000026214  | 2.0000102400262145 |
+---+--------------------+--------------------+
```

[ ]:

```

```

[6]:

```
from sympy import *
init_printing()
var('f(x)=x³-9x+3')
```

[6]:

$f(x) = x - 9x + 3$

$f(x) = x - 9x + 3$

Solução numerica : 0.33760897287513775

Numero de interação: 4

6 Método Híbrido

[7]:

```
def f(x):
    return x**3 - 9*x + 3

def hibrido(f,x0,x1,n=100,e=0.0005):
    #criando tabela
    t = PrettyTable(["n","Métod da secante","X0","X1"])
    t.align["n"] = "l"
    t.align["Métod da secante"] = "l"
    t.align["X0"] = "l"
    t.align["X1"] = "l"


    for i in range(n):
        x = x1 -f(x1)*(x0-x1)/(f(x0)-f(x1))
        t.add_row([i,x,x0,x1])
        if np.abs((x-x1)/x1)<=e:
            break
        x0 = x1
        x1 = x
    return t

print(hibrido(f,0,1))
```

```
+---+---------------------+---------------------+---------------------+
| n | Métod da secante    | X0                  | X1                  |
+---+---------------------+---------------------+---------------------+
| 0 | 0.375               | 0                   | 1                   |
| 1 | 0.33194154488517746 | 1                   | 0.375               |
```

```
| 2 | 0.33763462072303707 | 0.375               | 0.331941544885177466 |
| 3 | 0.33760897287513775 | 0.33194154488517746 | 0.33763462072303707 |
+---+---------------------+---------------------+---------------------+
```

7 Método do trapézio

```
[32]: def trapezio_composto(f, a, b, n,eps):
          h = (b-a)/float(n)
          result = f(a) + f(b)


          for i in np.arange(1,n,1):
              result = result + (2*(f(a + i*h)))
          result *= (h/2.0)

          return result
      ################################################
      def trapezio_simples(f, a, b):

          return float(b-a)*((f(a)+f(b))/2)

      #################################################
      def answerQuestion(f, a, b, eps=0.0000001):
          x = PrettyTable(["n","Regra do Trapezio composto","a","b","Regra do Trapezio␣
       ↪simples"])

          x.align["n"] = "l"
          x.align["Trapezoid"] = "l"
          x.align["a"] = "l"
          x.align["b"] = "l"
          x.align["Regra do Trapezio simples"] = "l"


          i = 0
          aux = 0
          erro = 0
          while True:


              n = 2**i

              t =  trapezio_composto(f, a, b, n,eps)

              ts = trapezio_simples(f, a, b)
              if (abs(t - aux) < eps):
                  break
```

```
        aux = t
        x.add_row([n,aux,a,b,ts])
        i = i + 1




    print(x)
################################################
def f(x):
    return math.log(x+9)

if __name__ == '__main__':
    answerQuestion(f,1,7)
```

```
+------+---------------------------+---+---+--------------------------+
| n    | Regra do Trapezio composto | a | b | Regra do Trapezio simples |
+------+---------------------------+---+---+--------------------------+
| 1    |     15.22552144570148      | 1 | 7 | 15.22552144570148        |
| 2    |     15.307608795235351     | 1 | 7 | 15.22552144570148        |
| 4    |     15.328547924811275     | 1 | 7 | 15.22552144570148        |
| 8    |     15.333811476453443     | 1 | 7 | 15.22552144570148        |
| 16   |     15.335129214271696     | 1 | 7 | 15.22552144570148        |
| 32   |     15.335458765209516     | 1 | 7 | 15.22552144570148        |
| 64   |     15.335541160237913     | 1 | 7 | 15.22552144570148        |
| 128  |     15.335561759451094     | 1 | 7 | 15.22552144570148        |
| 256  |     15.335566909282909     | 1 | 7 | 15.22552144570148        |
| 512  |     15.33556819674265      | 1 | 7 | 15.22552144570148        |
| 1024 |     15.33556851860767      | 1 | 7 | 15.22552144570148        |
+------+---------------------------+---+---+--------------------------+
```

[ ]:

8 Método de Simpson

[74]:

```
#simpson composto 1/3
from prettytable import PrettyTable

def simpson_c_1_3(f,a, b, n,eps):
    h = (b - a)/float(n)
    result = 0.0
    if abs(b - a) < eps:
        return 0, False
```

```
    for i in range(1,int(n/2 + 1)):
        result += 4*f(a + (-1+2*i)*h)


    for i in range(1, int(n/2)):
        result += 2*f(a + 2*i*h)

    return (h/3) * ((f(a) + f(b))+ result),True

#simpson simples 1/3
def simpson_s_1_3(f,a,b):
    return (((b - a) / 6) * (f(a) + 4 * f((a + b) / 2) + f(b)))
#simpson simples 3/8
def simpson_s_3_8(f,a,b):
    return (f(a) + 3 * f((2 * a + b) / 3) + 3 * f((2 * b + a) / 3) + f(b)) * ((b␣
 ↪- a) /8.0)
#simpson composto 3/8
def simpson_c_3_8(f,a, b, n,eps):
    h = ((b - a) / n)
    result = f(a) + f(b);
    if abs(b - a) < eps:
        return 0,False

    for i in range(1,n):
        if (i % 3 == 0):

            result +=  2 * f(a + i * h)

        else:
            result+= 3 * f(a + i * h)


    return ((( 3 * h) / 8) * result),True
def tableSimpson(f, a, b, eps=10**(-7)):
    x = PrettyTable(["n","Regra de Simpson's composta 1/3 ","simples 1/3 ","␣
 ↪composta 3/8","Simples 3/8"])
    x.align["n"] = "l"
    x.align["Regra de Simpson's composta 1/3"] = "l"
    x.align["simples 1/3"] = "l"
    x.align["composta 1/3"] = "l"
    x.align["composta 3/8"] = "l"
    x.align["simples 3/8"] = "l"
    aux = 0
    i = 0
    aux2 = 0
    while True:
```

```python
        n = 2**i

        t,d = simpson_c_1_3(f, a, b, n,eps)
        s = simpson_s_1_3(f, a, b)
        h,er = simpson_c_3_8(f, a, b, n,eps)
        v = simpson_s_3_8(f, a, b)

        if (abs(t - aux) < eps)or d == False:
            break
        aux = t
        aux2 = h
        x.add_row([n,aux,s,aux2,v])
        i = i + 1
    print(x)
################################################
def f(x):
    return math.log(x + 9)



if __name__ == '__main__':
    tableSimpson(f,1,7)
```

```
+----+----------------------------------+--------------------+------------------
--+--------------------+
| n  | Regra de Simpson's composta 1/3  |    simples 1/3     |    composta 3/8
|    Simples 3/8     |
+----+----------------------------------+--------------------+------------------
--+--------------------+
| 1  |         10.150347630467653       | 15.334971245079974 | 11.41914108427611
| 15.335299315082702 |
| 2  |         15.334971245079974       | 15.334971245079974 |
14.366274623570742 | 15.335299315082702 |
| 4  |         15.335527634669916       | 15.334971245079974 |
14.313015164575747 | 15.335299315082702 |
| 8  |         15.335565993667498       | 15.334971245079974 |
15.080058115813014 | 15.335299315082702 |
| 16 |         15.33556846021111        | 15.334971245079974 |
15.076732205205357 | 15.335299315082702 |
| 32 |         15.33556861552213        | 15.334971245079974 |
15.270861236357348 | 15.335299315082702 |
+----+----------------------------------+--------------------+------------------
--+--------------------+
```

[ ]:

9 Gauss Legendre

[10]:
```
# Definição da função e do valor do intervalo :
def g(x):
    return math.exp(x) + math.sin(x) + 2

a = 0
b = math.pi
f = np.vectorize(g)
# Gauss-Legendre intervalo -1, 1
deg = 2 #grau
x, w = np.polynomial.legendre.leggauss(deg)
#generalizando a quadratura [-1, 1] para   [a, b]
t = 0.5*(x + 1)*(b - a) + a
gauss = sum(w * f(t)) * 0.5*(b - a)

#integral exata da função
quad, quad_err = quad(f, a, b)
print ('Quadratura : {0:.12} O erro da quatradura: {1:.12}'.format(quad,␣
 ↪quad_err))
print ('Gauss-Legendre aproximação: {0:.12}'.format(gauss))
print ('Erro cometido ao aproximar o valor da integral e gauss-legendre: ',␣
 ↪abs(gauss - quad))
```

```
Quadratura : 30.42387794 O erro da quatradura: 3.37772897873e-13
Gauss-Legendre aproximação: 29.984162615
Erro cometido ao aproximar o valor da integral e gauss-legendre:
0.43971532498408905
```

10 Integral Dupla

[14]:
```
from scipy import integrate
def trapezioAdaptado(f=f,a=0,b=2,c=-np.pi,d=np.pi,ny=1000,nx=1000):
    hx = (b - a)/float(nx)
    hy = (d - c)/float(ny)
    result = f(a,c)+f(b,d)
    for i in range(ny):
        yi = a + i*hx
        result += 2*(f(c, yi) + f(d, yi))
    result *= (hy/2)
    return result
def simpsonAdaptado(f=f,a=0,b=2,c=-np.pi,d=np.pi,ny=1000,nx=1000):
    hx = (b - a)/float(nx)
```

```python
    hy = (d - c)/float(ny)
    result = f(a,c) + f(b,d)

    for j in range(1,int(nx/2 + 1)):
        xj = c + (-1+2*j)*hy
        result += 4*(f(xj,a) + f(xj,b))
    for j in range(1,int(nx/2)):
        xj = c + 2*j*hy
        result += 2*(f(xj, a) + f(xj,b))


    result *= (hx/3)
    return result
def integralDupla(f,a,b,c,d,trapezioAdaptado,simpsonAdaptado,nx,ny):
    hx = (b - a)/float(nx)
    hy = (d - c)/float(ny)
    I = 0.166666667*(f(a, c) + f(a, d) + f(b, c) + f(b, d))
    Ixy = 0
    I += trapezioAdaptado(f,a,b,c,d,ny,nx)
    I += simpsonAdaptado(f,a,b,c,d,ny,nx)
    for i in range(1, nx):
        for j in range(1, ny):
            xi = a + i*hx
            yj = c + j*hy
            Ixy += f(xi, yj)

    I += Ixy
    I *= hx*hy
    return I
#função
def f(x, y):
    return x + np.sin(y) + 1
def integrand(y, x):
    return x + np.sin(y) + 1
#Intervalos
a = 0;  b = 2;  c = -np.pi;  d = np.pi
#Integral exata da função
I, err = integrate.dblquad(integrand,0,2,-np.pi,np.pi)
#Usando quadraturas
I_aproxi = integralDupla(f,a,b,c,d,trapezioAdaptado,simpsonAdaptado,1000,1000)
print('Valor da aproximação da integral dupla:',I_aproxi)
print('Valor da integral dupla exata:',I)
print('Erro cometido da aproximação',abs(I - I_aproxi))
```

```
Valor da aproximação da integral dupla: 25.082860551770896
Valor da integral dupla exata: 25.13274122871834
Erro cometido da aproximação 0.04988067694744558
```

[2]:

```

```

2

[18]:

```
from math import sin

def f(x):
        return sin(x) -np.cos(x)

def calc(a,b,fa,fb):
        return a+(fa*(a-b))/(fb-fa)

def pos_false(f,a,b):
        if(abs(b-a)<=epsilon or abs(f(a))<=epsilon or abs(f(b))<=epsilon):
             return b
        for i in range(0,n_iter):

                p = calc(a,b,f(a),f(b))

                if(abs(f(p))<=epsilon):
                    return p
                if(f(a)*f(p)>0):
                        a = p
                else:
                        b = p
                if(abs(b-a)<=epsilon):
                        return a
        return p

n_iter = 100
epsilon=0.01
print(pos_false(f,1,2))
```

0.7857834392883747

[ ]:

```

```