Engenharia de Computação
Métodos numéricos - 2018.2

Trabalho extra: Interpolação e integração numérica

João Luca Ripardo Teixeira Costa – 399951

Casos de teste adotados

Foram utilizados os seguintes casos de teste:

Cenário 1 para teste dos métodos de interpolação numérica

x	0	0.2618	0.5234	0.7854	1.0472	1.3090
$f(x)$	0	1.0353	2	2.8284	3.4641	3.8637

- $f(0.6) = ?$
- Usar polinômios de ordem 1, 2, 3, 4 e 5.

Cenário 2 para teste dos métodos de interpolação numérica inversa

x	0	0.1	0.2	0.3	0.4	0.5
$f(x)$	1	1.1052	1.2214	1.3499	1.4918	1.6487

- $f(x) = 1.3165?$
- Usar polinômios de ordem 1, 2, 3, 4 e 5.

Cenário 3 para teste dos métodos de integração numérica

$$\int_2^{14} \frac{1}{\sqrt{x}} \mathrm{d}x \approx 4.6549$$

$$\int_0^{\pi/2} \cos(x) \mathrm{d}x = 1$$

$$\int_0^1 (\exp(x) + x^2) \mathrm{d}x \approx 2.05162$$

- Nos métodos com repetição, utilize $m = 2, 4, 10$ repetições.

Obs.: apenas a primeira equação do 3° cenário foi utilizada.

Métodos de interpolação

- Sistemas lineares

Interpolação de polinômio de 1º grau:

```
Digite o vetor de pontos x0;x1;x1;{...};xn : "[[0.5234],[0.7854]]"
Digite o vetor de pontos f(x0);f(x1);{...};f(xn) :"[[2],[2.8284]]"
Digite um valor de x para ser testado no polinômio da interpolação: 0.6
y(x) = 3.16183206107*x + 0.345097099237
y(0.6): 2.24219633587900
Tempo de execucao total: 7.739067e-04 segundos
```

Interpolação de polinômio de 2º grau:

```
Digite o vetor de pontos x0;x1;x1;{...};xn : "[[0.2618],[0.5234],[0.7854]]"
Digite o vetor de pontos f(x0);f(x1);{...};f(xn) :"[[1.0353],[2],[2.8284]]"
Digite um valor de x para ser testado no polinômio da interpolação: 0.6
y(x) = -1.00431449662*x**2 + 4.47627887424*x - 0.0677548569578
y(0.6): 2.25645924880300
Tempo de execucao total: 1.168013e-03 segundos
```

Interpolação de polinômio de 3º grau:

```
Digite o vetor de pontos x0;x1;x1;{...};xn : "[[0.2618],[0.5234],[0.7854],[1.0472]]"
Digite o vetor de pontos f(x0);f(x1);{...};f(xn) :"[[1.0353],[2],[2.8284],[3.4641]]"
Digite um valor de x para ser testado no polinômio da interpolação: 0.6
y(x) = -0.50458542644*x**3 - 0.211812625853*x**2 + 4.09596163654*x - 0.0134512145975
y(0.6): 2.25888276990838
Tempo de execucao total: 1.770973e-03 segundos
```

Interpolação de polinômio de 4º grau:

```
Digite o vetor de pontos x0;x1;x1;{...};xn : "[[0],[0.2618],[0.5234],[0.7854],[1.0472]]"
Digite o vetor de pontos f(x0);f(x1);{...};f(xn) :"[[0],[1.0353],[2],[2.8284],[3.4641]]"
Digite um valor de x para ser testado no polinômio da interpolação: 0.6
y(x) = 0.119354173677*x**4 - 0.817030782291*x**3 + 0.0744529314836*x**2 + 3.98891070563*x
y(0.6): 2.25913913064578
Tempo de execucao total: 1.458168e-03 segundos
```

Interpolação de polinômio de 5º grau:

```
Digite o vetor de pontos x0;x1;x1;{...};xn : "[[0],[0.2618],[0.5234],[0.7854],[1.0472],[1.3090]]"
Digite o vetor de pontos f(x0);f(x1);{...};f(xn) :"[[0],[1.0353],[2],[2.8284],[3.4641],[3.8637]]"
Digite um valor de x para ser testado no polinômio da interpolação: 0.6
y(x) = -0.021862186332*x**5 + 0.176585005057*x**4 - 0.869466241916*x**3 + 0.094061524351*x**2 + 3.98644683737*x
y(0.6): 2.25911095598072
Tempo de execucao total: 3.476143e-03 segundos
```

Código de interpolação por sistemas lineares:

```
# -*- coding: utf-8 -*-


import numpy as np
from sympy import *
from math import *

from timeit import default_timer as timer

def resolveTriangular(A):

    R = np.asmatrix([0]*A.shape[0]) # matriz de valores das icógnitas
    R = R.astype(float)

    for i in range(A.shape[0]-1,-1,-1):

        R[0,i] = np.copy(R[0,i]) + np.copy(A[i,A.shape[0]])

        for j in range(A.shape[0]-1,i,-1):

            R[0,i] = np.copy(R[0,i]) - np.copy(A[i,j])

        R[0,i] = (np.copy(R[0,i])/np.copy(A[i,i]))

        A[:,i] = np.copy(A[:,i])*np.copy(R[0,i])


    return [A,R]
```

```
def pivotacaoParcial(A):

    p = 0
    m = None

    for i in range(0,A.shape[0]):
        for j in range(i+1,A.shape[0]):
            if A[i,i] < A[j,i]:
                p += 1
                Temp = np.copy(A[i])
                A[i] = np.copy(A[j])
                A[j] = np.copy(Temp)

        for k in range(i+1,A.shape[0]):
            d = np.copy(A[k,i])
            d = d/np.copy(A[i,i])
            A[k,:] = np.copy(A[k,:]) - (np.copy(A[i,:])*d)

    return [A,p]

start = None
end = None

x = np.matrix(eval(input("\n\n   Digite o vetor de pontos x0;x1;x1;{...};xn : ")))
x = x.astype(float)
y = np.matrix(eval(input("   Digite o vetor de pontos f(x0);f(x1);{...};f(xn) :")))
x = x.astype(float)
n = x.shape[0] # quantidade de linhas de x = quantidade de pontos

v = np.zeros([n,n])

v[:,0] = 1

```

```
for i in range(1,n):

    v[:,i] = np.transpose(np.power(np.copy(x),i))


v = np.hstack((np.copy(v),np.copy(y)))

start = timer()

r = resolveTriangular(pivotacaoParcial(v)[0])[1]

end = timer()

st = ""

for j in range(0,r.shape[1]):

    if(j == 0): # se for o termo independente
        if(r[0,j] > 0):
            st += " + "+str(r[0,j])
        elif(r[0,j] < 0):
            st += " "+str(r[0,j])

    elif(j != r.shape[0]-1):

        if(r[0,j] > 0):
            st += " + "+str(r[0,j])+"*x**"+str(j)
        elif(r[0,j] < 0):
            st += " "+str(r[0,j])+"*x**"+str(j)

x = symbols('x')
st = sympify(str(st))

x0 = float(input("   Digite um valor de x para ser testado no polinômio da interpolação: "))

print("   y(x) = "+str(st))
print("   y("+str(x0)+"): "+str(st.subs(x,x0)))
print("   Tempo de execucao total: %e segundos\n\n" % (end - start))


```

- Lagrange

Interpolação de polinômio de 1º grau:

```
Digite os valores de x -> [[x0,x1,{...},xn]]: "[[0.5234,0.7854]]"
Digite os valores de y -> [[y0,y1,{...},yn]]: "[[2,2.8284]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 3.1618320610687*x + 0.345097099236642
y(0.6) = 2.24219633587786
Tempo de execucao total: 3.174686e-02 segundos
```

Interpolação de polinômio de 2º grau:

```
Digite os valores de x -> [[x0,x1,{...},xn]]: "[[0.2618,0.5234,0.7854]]"
Digite os valores de y -> [[y0,y1,{...},yn]]: "[[1.0353,2,2.8284]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 7.55838198498749*(x - 0.7854)*(x - 0.5234) - 29.1803814459463*(x - 0.7854)*(x - 0.2618) + 20.6176849643397*(x - 0.5234)*(x - 0.2618)
y(0.6) = 2.25645924880563
Tempo de execucao total: 4.654598e-02 segundos
```

Interpolação de polinômio de 3º grau:

```
Digite os valores de x -> [[x0,x1,{...},xn]]: "[[0.2618,0.5234,0.7854,1.0472]]"
Digite os valores de y -> [[y0,y1,{...},yn]]: "[[1.0353,2,2.8284,3.4641]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = -9.6236083333174*(x - 1.0472)*(x - 0.7854)*(x - 0.5234) + 55.7090138334217*(x - 1.0472)*(x - 0.7854)*(x - 0.2618) - 78.7535712923593*
(x - 1.0472)*(x - 0.5234)*(x - 0.2618) + 32.1635803658151*(x - 0.7854)*(x - 0.5234)*(x - 0.2618)
y(0.6) = 2.25888276990828
Tempo de execucao total: 8.588910e-02 segundos
```

Interpolação de polinômio de 4º grau:

```
Digite os valores de x -> [[x0,x1,{...},xn]]: "[[0,0.2618,0.5234,0.7854,1.0472]]"
Digite os valores de y -> [[y0,y1,{...},yn]]: "[[0,1.0353,2,2.8284,3.4641]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = -36.7593901196234*x*(x - 1.0472)*(x - 0.7854)*(x - 0.5234) + 106.436786078375*x*(x - 1.0472)*(x - 0.7854)*(x - 0.2618) - 100.27192677
9169*x*(x - 1.0472)*(x - 0.5234)*(x - 0.2618) + 30.7138849940939*x*(x - 0.7854)*(x - 0.5234)*(x - 0.2618)
y(0.6) = 2.25913913064322
Tempo de execucao total: 8.805299e-02 segundos
```

Interpolação de polinômio de 5º grau:

```
Digite os valores de x -> [[x0,x1,{...},xn]]: "[[0,0.2618,0.5234,0.7854,1.0472,1.3090]]"
Digite os valores de y -> [[y0,y1,{...},yn]]: "[[0,1.0353,2,2.8284,3.4641,3.8637]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 35.1025497704578*x*(x - 1.309)*(x - 1.0472)*(x - 0.7854)*(x - 0.5234) - 135.484707329908*x*(x - 1.309)*(x - 1.0472)*(x - 0.7854)*(x -
0.2618) + 191.504825781453*x*(x - 1.309)*(x - 1.0472)*(x - 0.5234)*(x - 0.2618) - 117.318124499977*x*(x - 1.309)*(x - 0.7854)*(x - 0.5234)*(x
- 0.2618) + 26.1735940916411*x*(x - 1.0472)*(x - 0.7854)*(x - 0.5234)*(x - 0.2618)
y(0.6) = 2.25911095597962
Tempo de execucao total: 1.606190e-01 segundos
```

Código de interpolação por fórmula de Lagrange:

```python
# -*- coding:utf-8 -*-

import numpy as np
from sympy import *
from math import *

from timeit import default_timer as timer

def lagrange(X,Y,x):
    Z = zeros(X.shape[0],X.shape[1])
    Pn = Z[:,:]
    n = 1
    d = 1
    Mxk = Z[:,:]
    for i in range(0,X.shape[1]):
        Pn[0,i] = 0
        n = 1
        d = 1
        for j in range(0,X.shape[1]):
            if(j != i):
                n *= x - X[0,j]
                d *= X[0,i] - X[0,j]
        Pn[0,i] = Y[0,i]*(n/d)
    return Pn

start = None
end = None

x = symbols('x')

X = Matrix(eval(input("\n\n   Digite os valores de x -> [[x0,x1,{...},xn]]: ")))
Y = Matrix(eval(input("   Digite os valores de y -> [[y0,y1,{...},yn]]: ")))

start = timer()

L = lagrange(X,Y,x)

end = timer()

LM = 0

```

```
for i in range(0,L.shape[1]):

    LM += L[0,i]


LM = sympify(LM)

x0 = float(input("   Digite um valor de x para ser testado no polinômio interpolado: "))
print("   y(x) = "+str(LM))
print("   y("+str(x0)+") = "+str(LM.subs(x,x0)))
print("   Tempo de execucao total: %e segundos\n\n" % (end - start))
```

- Newton

Interpolação de polinômio de 1º grau:

```
Digite o vetor x: "[[0.5234,0.7854]]"
Digite o vetor y: "[[2,2.8284]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 3.1618320610687*x + 0.345097099236642
y(0.6) = 2.24219633587786
Tempo de execucao total: 1.231599e-02 segundos
```

Interpolação de polinômio de 2º grau:

```
Digite o vetor x: "[[0.2618,0.5234,0.7854]]"
Digite o vetor y: "[[1.0353,2,2.8284]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 3.68769113149847*x - 1.00431449661912*(x - 0.5234)*(x - 0.2618) + 0.0698624617737006
y(0.6) = 2.25645924880563
Tempo de execucao total: 3.570986e-02 segundos
```

Interpolação de polinômio de 3º grau:

```
Digite o vetor x: "[[0.2618,0.5234,0.7854,1.0472]]"
Digite o vetor y: "[[1.0353,2,2.8284,3.4641]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 3.68769113149847*x - 0.504585426439899*(x - 0.7854)*(x - 0.5234)*(x - 0.2618) - 1.00431449661912*(x - 0.5234)*(x - 0.2618) + 0.069862
4617737006
y(0.6) = 2.25888276990828
Tempo de execucao total: 5.491304e-02 segundos
```

Interpolação de polinômio de 4º grau:

```
Digite o vetor x: "[[0,0.2618,0.5234,0.7854,1.0472]]"
Digite o vetor y: "[[0,1.0353,2,2.8284,3.4641]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = 0.119354173676912*x*(x - 0.7854)*(x - 0.5234)*(x - 0.2618) - 0.629573117114362*x*(x - 0.5234)*(x - 0.2618) - 0.509847770437495*x*(x -
0.2618) + 3.95454545454546*x
y(0.6) = 2.25913913064322
Tempo de execucao total: 9.660816e-02 segundos
```

Interpolação de polinômio de 5º grau:

```
Digite o vetor x: "[[0,0.2618,0.5234,0.7854,1.0472,1.3090]]"
Digite o vetor y: "[[0,1.0353,2,2.8284,3.4641,3.8637]]"
Digite um valor de x para ser testado no polinômio interpolado: 0.6
y(x) = -0.0218621863319944*x*(x - 1.0472)*(x - 0.7854)*(x - 0.5234)*(x - 0.2618) + 0.119354173676912*x*(x - 0.7854)*(x - 0.5234)*(x - 0.2618
) - 0.629573117114362*x*(x - 0.5234)*(x - 0.2618) - 0.509847770437495*x*(x - 0.2618) + 3.95454545454546*x
y(0.6) = 2.25911095597962
Tempo de execucao total: 1.201389e-01 segundos
```

Código de interpolação por fórmula de Newton:

```python
# -*- coding: utf-8 -*-

import numpy as np
from sympy import *
from math import *

from timeit import default_timer as timer

def diferencasDivididas(X,Y):
    if(Y.shape[1] == 1):

        return Y[0]

    else:

        X1 = X[:,1:]
        X0 = X[:,:(X.shape[1]-1)]
        Y1 = Y[:,1:]
        Y0 = Y[:,:(Y.shape[1]-1)]

        d = (diferencasDivididas(X1,Y1) - diferencasDivididas(X0,Y0))/(X[X.shape[1]-1] - X[0])

        return d

start = None
end = None

x = symbols('x')

X = Matrix(eval(input("\n\n   Digite o vetor x: ")))
Y = Matrix(eval(input("   Digite o vetor y: ")))

Xp = ones(X.shape[0],X.shape[1])*x - X[:,:]

Pn = zeros(X.shape[0],X.shape[1])

Pn[0,0] = Y[0,0]

Sn = Y[0,0]

start = timer()

```

```
for i in range(1,X.shape[1]):

    p = 1

    for j in range(0,i):

        p *= Xp[0,j]

    Pn[0,i] = p*diferencasDivididas(X[:,:(i+1)],Y[:,:(i+1)])

end = timer()

for i in range(1,Pn.shape[1]):

    Sn += Pn[0,i]

Sn = sympify(Sn)

x0 = float(input("   Digite um valor de x para ser testado no polinômio interpolado: "))
print("   y(x) = "+str(Sn))
print("   y("+str(x0)+") = "+str(Sn.subs(x,x0)))
print("   Tempo de execucao total: %e segundos\n\n" % (end - start))
```

- Interpolação inversa

Interpolação de polinômio de 1º grau:

```
Digite o vetor x: "[[0.2,0.3]]"
Digite o vetor y: "[[1.2214,1.3499]]"
Digite um valor de y para ser testado no polinômio interpolado: 1.3165
Digite a precisão do x estimado: 0.00001
Pn(x) = 1.285*x + 0.9644
f(xk) = 1.3165
xk = 0.274007782101167
Tempo de execucao total: 1.873612e-02 segundos
```

Interpolação de polinômio de 2º grau:

```
Digite o vetor x: "[[0.2,0.3,0.4]]"
Digite o vetor y: "[[1.2214,1.3499,1.4918]]"
Digite um valor de y para ser testado no polinômio interpolado: 1.3165
Digite a precisão do x estimado: 0.00001
Pn(x) = 1.285*x + 0.669999999999989*(x - 0.3)*(x - 0.2) + 0.9644
f(xk) = 1.3165
xk = 0.274985779191895
Tempo de execucao total: 3.047585e-02 segundos
```

Interpolação de polinômio de 3º grau:

```
  Digite o vetor x: "[[0.1,0.2,0.3,0.4]]"
  Digite o vetor y: "[[1.1052,1.2214,1.3499,1.4918]]"
  Digite um valor de y para ser testado no polinômio interpolado: 1.3165
  Digite a precisão do x estimado: 0.00001
  Pn(x) = 1.162*x + 0.183333333333298*(x - 0.3)*(x - 0.2)*(x - 0.1) + 0.615*(x
- 0.2)*(x - 0.1) + 0.989
  f(xk) = 1.3165
  xk = 0.274953136140996
  Tempo de execucao total: 4.850817e-02 segundos
```

Interpolação de polinômio de 4º grau:

```
  Digite o vetor x: "[[0,0.1,0.2,0.3,0.4]]"
  Digite o vetor y: "[[1,1.1052,1.2214,1.3499,1.4918]]"
  Digite um valor de y para ser testado no polinômio interpolado: 1.3165
  Digite a precisão do x estimado: 0.00001
  Pn(x) = -0.0833333333333705*x*(x - 0.3)*(x - 0.2)*(x - 0.1) + 0.21666666666666
47*x*(x - 0.2)*(x - 0.1) + 0.550000000000006*x*(x - 0.1) + 1.052*x + 1
  f(xk) = 1.3165
  xk = 0.274955734829894
  Tempo de execucao total: 8.948612e-02 segundos
```

Interpolação de polinômio de 5º grau:

```
Digite o vetor x: "[[0,0.1,0.2,0.3,0.4,0.5]]"
Digite o vetor y: "[[1,1.1052,1.2214,1.3499,1.4918,1.6487]]"
Digite um valor de y para ser testado no polinômio interpolado: 1.3165
Digite a precisão do x estimado: 0.00001
Pn(x) = 0.583333333333935*x*(x - 0.4)*(x - 0.3)*(x - 0.2)*(x - 0.1) - 0.08333
33333333705*x*(x - 0.3)*(x - 0.2)*(x - 0.1) + 0.216666666666647*x*(x - 0.2)*(x -
 0.1) + 0.550000000000006*x*(x - 0.1) + 1.052*x + 1
f(xk) = 1.3165
xk = 0.274950732895916
Tempo de execucao total: 1.486599e-01 segundos
```

Código de interpolação inversa com método de Newton-Raphson:

```
# -*- coding: utf-8 -*-


import numpy as np
from sympy import *
from math import *

from timeit import default_timer as timer

def phix(a, b, c, x0,x): # define função phi(x)

    return a - (b.subs(x,x0)/c.subs(x,x0))

def xProximo(y,Y):

    xR = Y[0,0]

    for i in range(1,Y.shape[1]):

        if(abs(Y[0,i] - y) < abs(xR - y)):
            xR = Y[0,i]
    return xR

```

```python
def newtonRaphson(fx,dfx,x0,e,c,x):

    phi = None
    d = 0

    while (d < c) :

        phi = phix(x0,fx,dfx,x0,x) # calcula phi(x0)

        if(abs(fx.subs(x,x0)) < e): # checa se a função em x0 é menor ou igual à precisão desejada

            return [phi,0,d]

        x0 = phi # caso f(x) não seja perto de 0 o suficiente, x0 recebe o valor de phi(x0) e segue no laço

        if(d == c-1):

            return [phi,1]

        d+=1


def diferencasDivididas(X,Y):
    if(Y.shape[1] == 1):

        return Y[0]

    else:

        X1 = X[:,1:]
        X0 = X[:,:(X.shape[1]-1)]
        Y1 = Y[:,1:]
        Y0 = Y[:,:(Y.shape[1]-1)]

        d = (diferencasDivididas(X1,Y1) - diferencasDivididas(X0,Y0))/(X[X.shape[1]-1] - X[0])

        return d
```

```python
start = None
end = None
x = symbols('x')
X = Matrix(eval(input("\n\n   Digite o vetor x: ")))
Y = Matrix(eval(input("   Digite o vetor y: ")))
Xp = ones(X.shape[0],X.shape[1])*x - X[:,:]
Pn = zeros(X.shape[0],X.shape[1])
Pn[0,0] = Y[0,0]
Sn = Y[0,0]
start = timer()
for i in range(1,X.shape[1]):
    p = 1
    for j in range(0,i):
        p *= Xp[0,j]
    Pn[0,i] = p*diferencasDivididas(X[:,:(i+1)],Y[:,:(i+1)])
end = timer()
for i in range(1,Pn.shape[1]):
    Sn += Pn[0,i]
Sn = sympify(Sn)
c = 30
y0 = float(input("   Digite um valor de y para ser testado no polinômio interpolado: "))
e = float(input("   Digite a precisão do x estimado: "))
Sn2 = Sn - y0
print("   Pn(x) = "+str(Sn))
print("   f(xk) = "+str(y0))
print("   xk = "+str(newtonRaphson(Sn2,diff(Sn2,x),xProximo(y0,Y),e,c,x)[0]))
print("   Tempo de execucao total: %e segundos\n\n" % (end - start))
```

Métodos de integração numérica

- Regra dos trapézios

m = 2

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de divisões: 2
Integral aproximada: 5.04442441285656
```

m = 4

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de divisões: 4
Integral aproximada: 4.76838702666144
```

m = 10

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de divisões: 10
Integral aproximada: 4.67452919165302
```

Código de integração por regra dos trapézios:

```python
# -*- coding: utf-8 -*-


import numpy as np
from sympy import *
from math import *

def regraDosTrapezios(fx,a,b,x):

    return ((b-a)*(fx.subs(x,a) + fx.subs(x,b)))/2


x = symbols('x')

fx = sympify(str(input("\n\n   Digite a função f(x): ")))
a = float(input("   Digite o começo do intervalo de integração: "))
b = float(input("   Digite o fim do intervalo de integração: "))
t = float(input("   Digite o modo de integração: (0 - sem repetição, 1 - com repetição): "))

if(t == 0):

    Ia = regraDosTrapezios(fx,a,b,x)
    print("   Integral aproximada: "+str(Ia)+"\n\n")
```

```python
if(t == 0):

    Ia = regraDosTrapezios(fx,a,b,x)
    print("   Integral aproximada: "+str(Ia)+"\n\n")

elif(t == 1):

    m = int(input("   Digite a quantidade m de divisões: "))

    h = abs(b-a)/m

    Et = (-h**3/12)*diff(diff(fx,x),x).subs(x,a)

    if(m*h < (b-a)):
        hEx = (b-a) - h*m
        Ia = 0
        xk = a
        for i in range(0,m+1):
            if(i == m):
                Et += -(h**3/12)*diff(diff(fx,x),x).subs(x,a+hEx)
                Ia += regraDosTrapezios(fx,a,(a+hEx),x)
                a += hEx
            else:
                Et += -(h**3/12)*diff(diff(fx,x),x).subs(x,a+h)
                Ia += regraDosTrapezios(fx,a,(a+h),x)
                a += h
        print("   Integral aproximada: "+str(Ia)+"\n\n")
    else:
        Ia = 0
        xk = a
        for i in range(0,m):
            Et += -(h**3/12)*(diff(diff(fx,x),x).subs(x,a+h))
            Ia += regraDosTrapezios(fx,a,(a+h),x)
            a += h

        print("   Integral aproximada: "+str(Ia)+"\n\n")

```

- 1/3 de Simpson

m = 2

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de intervalos: 2
Integral aproximada: 4.77716317094413
```

m = 4

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de intervalos: 4
Integral aproximada: 4.67637456459641
```

m = 10

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de intervalos: 10
Integral aproximada: 4.65614707122973
```

Código de integração por 1/3 de Simpson:

```
# -*- coding: utf-8 -*-

import numpy as np
from sympy import *
from math import *

def regra_1_3_Simpson(fx,a,h,x):

    return ((h)*(fx.subs(x,a) + 4*fx.subs(x,a+h) + fx.subs(x,a+2*h)))/3


x = symbols('x')

fx = sympify(str(input("\n\n   Digite a função f(x): ")))
a = float(input("   Digite o começo do intervalo de integração: "))
b = float(input("   Digite o fim do intervalo de integração: "))
t = float(input("   Digite o modo de integração: (0 - sem repetição, 1 - com repetição): "))

if(t == 0):

    Ir = integrate(fx,(x,a,b))
    Ia = regra_1_3_Simpson(fx,a,abs(b-a)/2,x)
    print("   Integral aproximada: "+str(Ia)+"\n\n")

elif(t == 1):

   m = int(input("   Digite a quantidade m de intervalos: "))

   h = float(abs(b-a)/m)

   Et = -(h**5/90)*diff(diff(diff(diff(fx,x),x),x),x)

   Es = Et.subs(x,a)

   if(m%2 == 0 and m*h == (b-a)):
        Ia = 0
        for i in range(0,m-1,2):
            Es += Et.subs(x,a+h)
            Ia += regra_1_3_Simpson(fx,a,h,x)
            a += 2*h

        print("   Integral aproximada: "+str(Ia)+"\n\n")
   else:
        print("   Erro: m não é múltiplo de 2\n\n")
```

- 3/8 de Simpson

m = 4

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de intervalos: 4
Integral aproximada: 6.14860630562027
```

m = 7

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de intervalos: 7
Integral aproximada: 5.52797832033612
```

m = 10

```
Digite a função f(x): "1/((x)**(1/2))"
Digite o começo do intervalo de integração: 2
Digite o fim do intervalo de integração: 14
Digite o modo de integração: (0 - sem repetição, 1 - com repetição): 1
Digite a quantidade m de intervalos: 10
Integral aproximada: 5.27323578902305
```

Obs.: A quantidade m de divisões do intervalo de integração no método de 3/8 de Simpson precisa obedecer a fórmula m = 4 + 3(n-1) onde n é um número natural maior do que zero. Por isso, o valor 2 não foi utilizado neste método.

Código do método de integração de 3/8 de Simpson:

```python
# -*- coding: utf-8 -*-


import numpy as np
from sympy import *
from math import *

def regra_3_8_Simpson(fx,a,h,x):

    return ((3*h)*(fx.subs(x,a) + 3*fx.subs(x,a+h) + 3*fx.subs(x,a+2*h) + fx.subs(x,a+3*h)))/8

x = symbols('x')

fx = sympify(str(input("\n\n   Digite a função f(x): ")))
a = float(input("   Digite o começo do intervalo de integração: "))
b = float(input("   Digite o fim do intervalo de integração: "))
t = float(input("   Digite o modo de integração: (0 - sem repetição, 1 - com repetição): "))

if(t == 0):

    Ir = integrate(fx,(x,a,b))
    Ia = regra_3_8_Simpson(fx,a,abs(b-a)/4,x)
    print("   Integral aproximada: "+str(Ia)+"\n\n")


elif(t == 1):

   m = int(input("   Digite a quantidade m de intervalos: "))

   h = float(abs(b-a)/m)

   if(m*h == (b-a) and (m-4)%3 == 0):
        Ia = 0
        for i in range(0,m,3):
            Ia += regra_3_8_Simpson(fx,a,h,x)
            a += 3*h

        print("   Integral aproximada: "+str(Ia)+"\n\n")
   else:
        print("   Erro: m não está no formato 4 + 3(n-1), onde n é um número inteiro maior do que zero.\n\n")
```

Todos os códigos podem ser acessados pelo link https://github.com/jlucartc/MetodosNumericosTrabalhoExtra20182