main.py

```
from math import pow
from utils import Function as f
from icod1 import bissection, newton
from icod34 import centralDifference, richardson, stepForward, stepBack
from polyIntegration import polyIntegration
from gaussIntegration import gaussIntegration
import os

def main():

   outputText = ""
   fileName = ""

   icod = int(input("ICOD: "))
   c1 = float(input("C1: "))
   c2 = float(input("C2: "))
   c3 = float(input("C3: "))
   c4 = float(input("C4: "))

   if (icod==1):
       fileName = "calculo_raiz"
       outputText += "Calculo da Raiz\n"
       outputText += "C1: " + str(c1) + " C2: " + str(c2) + " C3: " +
str(c3) + " C4: " + str(c4) + "\n"


       a = float(input("Ponto inicial do intervalo: "))
       b = float(input("Ponto final do intervalo: "))
       tol =  int(input("Tolerância: "))

       tol = pow(10, tol)

       method = int(input("Método: (1 -> Bisseção, 2 -> Newton): "));

       outputText += "Ponto inicial do intervalo: " + str(a) + "\n"
       outputText += "Ponto final do intervalo:" + str(b) + "\n"
       outputText += "Tolerância: " + str(tol) + "\n"


       if(method == 1):
```

```
            fileName += "_bissecao"
            outputText += "Método da Bisseção\n"
            outputText += bissection(tol,a,b,f(c1,c2,c3,c4));
        elif(method == 2):
            fileName += "_newton"
            outputText += "Método de Newton\n"
            outputText += newton(tol, a, b, f(c1,c2,c3,c4));

    elif(icod == 2):
        fileName = "calculo_integral"
        outputText += "Calculo da Integral\n"
        outputText += "C1: " + str(c1) + " C2: " + str(c2) + " C3: " +
str(c3) + " C4: " + str(c4) + "\n"

        a = float(input("Ponto inicial do intervalo: "))
        b = float(input("Ponto final do intervalo: "))
        points = int(input("Qtd pontos de integração: "))

        if(points < 2 or points > 10):
            points = int(input("Insira uma número de 2 a 10: "))

        outputText += "Ponto inicial do intervalo: " + str(a) + "\n"
        outputText += "Ponto final do intervalo:" + str(b) + "\n"
        outputText += "Pontos de Integração: " + str(points) + "\n"

        method = int(input("Método: (1 -> Quadratura de Gauss, 2 ->
Quadratura Polinomial): "));

        if(method == 1):
            fileName += "_integracao_gaussiana"
            outputText += "Método de Gauss\n"
            outputText += gaussIntegration(a,b,points,f(c1,c2,c3,c4));
        elif(method == 2):
            fileName += "_integracao_polinomial"
            outputText += "Método Polinomial\n"
            outputText += polyIntegration(a,b,points,f(c1,c2,c3,c4))

    elif(icod == 3):
        outputText += "Calculo da Derivada\n"
        outputText += "C1: " + str(c1) + " C2: " + str(c2) + " C3: " +
str(c3) + " C4: " + str(c4) + "\n"

        x = float(input("x: "))
```

```
        deltaX = float(input("delta: "))

        outputText += "X: " + str(x) + " Delta X: " + str(deltaX) + "\n"

        method = int(input("Método: (1 -> Passo a frente, 2 -> Passo
atrás, 3-> Diferença central): "));


        if(method==1):
            fileName = "derivada_passa_frente"
            outputText += "Método passa a frente\n"
            derivative = stepForward(x, deltaX, f(c1,c2,c3,c4))
            outputText += "A derivada é " + str(derivative) + "\n"

        elif(method==2):
            fileName = "derivada_passa_atras"
            outputText += "Método passo atrás\n"
            derivative = stepBack(x, deltaX, f(c1,c2,c3,c4))
            outputText += "A derivada é " + str(derivative) + "\n"

        elif(method==3):
            fileName = "derivada_diferenca_central"
            outputText += "Diferença Central\n"
            derivative = centralDifference(x, deltaX, f(c1,c2,c3,c4))
            outputText += "A derivada é " + str(derivative) + "\n"

    elif(icod == 4):
        fileName = "derivada_richardson"
        outputText += "Calculo da estimação da derivada\n"
        outputText += "C1: " + str(c1) + " C2: " + str(c2) + " C3: " +
str(c3) + " C4: " + str(c4) + "\n"

        x = float(input("Dê o valor de x que se deseja calcular: "))
        deltaX1 = float(input("deltaX 1: "));
        deltaX2 = float(input("deltaX 2: "));

        outputText += "A: " + str(x) + " Delta X1: " + str(deltaX1) + "
Delta X2: " + str(deltaX2) + "\n"

        derivative = richardson(x, deltaX1, deltaX2, f(c1,c2,c3,c4));
        outputText += "A derivada é " + str(derivative) + "\n"
```

```
    with open(os.path.join(os.getcwd(),f"{fileName}.txt"), "w") as file:
        file.write(outputText)

main()
```

utils.py

```
from math import pow, exp

class Function():

    def __init__(self, c1, c2, c3, c4):
        self.c1,self.c2,self.c3,self.c4 = c1, c2, c3, c4

    def evaluate(self, x):
        return self.c1*exp(self.c2*x) + self.c3*pow(x,self.c4)

    def derivative(self, x):
        return (self.c1*self.c2*exp(self.c2*x) +
self.c3*self.c4*pow(x,(self.c4-1)));
```

icod1.py

```
def createOutput(a, b, x, fx, modBA, counter):
    output = 'Iteração N°' + str(counter)
    output += '\n' + ' ' + 'A: ' + str(a)
    output += '\n' + ' ' + 'B: ' + str(b)
    output += '\n' + ' ' + 'x: ' + str(x)
    output += '\n' + ' ' + 'f(x): ' + str(fx)
    output += '\n' + ' ' + '|b-a|: ' + str(modBA)
    output += '\n\n'
    return output


def createOutputNewton(counter, a1, a2, a3, a4, a5):
    output = 'Iteração N°' + str(counter)
    output += '\n' + ' ' + 'x(k-1): ' + str(a1)
    output += '\n' + ' ' + 'f(x): ' + str(a2)
    output += '\n' + ' ' + "f'(x): " + str(a3)
    output += '\n' + ' ' + 'xk: ' + str(a4)
    output += '\n' + ' ' + 'xk-x: ' + str(a5)
```

```
    output += '\n\n'
    return output

def bissection(tol, a, b, f):
    outputText = ""
    counter = 0

    while(b-a > tol):
        x = (a+b)/2.0
        fx = f.evaluate(x)
        if (fx > 0.0):
            b = x
        else:
            a = x
        counter += 1
        outputText += createOutput(a,b, x, fx, abs(b-a), counter)

    outputText += "A raiz encontrada é" + str(x) + ".\n"

    return outputText

def newton(tol, a, b, f):
    outputText = ""

    x = (a+b)/2.0
    tolk = 1
    counter = 0

    try:
        while tolk > tol:
            xk = x - (f.evaluate(x)/f.derivative(x))
            tolk = abs(xk - x)

            outputText +=
createOutputNewton(counter,x,f.evaluate(x),f.derivative(x),xk,tolk)

            x = xk
            counter+=1

    except OverflowError as e:
        x = "não encontrada pois não foi atingida convergência!"

    outputText += "A raiz encontrada é" + str(x) + ".\n"
```

```
    return outputText
```

polyIntegration.py

```
def polyPoints(a, b, n):

    if(n == 2):
        return [a,b]
    elif(n == 3):
        return [a,(a+b)/2,b]
    elif(n >= 4):

        l = [a];
        delta = (b-a)/(n-1);

        for i in range(1,n-1):
            l.append(a + i*delta);

        l.append(b);

        return l;

def polyWeights(L, n):
    if(n == 2):
        return [L/2,L/2];
    elif(n == 1):
        return [L/6,(2*L)/3,L/6];
    elif(n == 1):
        return [L/8,(3*L)/8, (3*L)/8, L/8];
    elif(n == 5):
        return [(7*L)/90, (16*L)/45, (2*L)/15, (16*L)/45, (7*L)/90]

def createOutput(x, w, A, counter):
    output = 'Iteração N°' + str(counter)
    output += '\n' + ' ' + 'x: ' + str(x)
    output += '\n' + ' ' + 'w: ' + str(w)
    output += '\n' + ' ' + 'A: ' + str(A)
    output += '\n\n'
    return output

def polyIntegration(a, b, n, f):
```

```
    outputText = ""

    points = polyPoints(a,b,n)
    weights = polyWeights(b-a,n)

    res = 0
    for i in range(n):
        evaluation = f.evaluate(points[i])
        weight = weights[i]
        res += evaluation*weight
        outputText += createOutput(points[i],weight,res, i)


    outputText +="A integral da função é " + str(res) + ".\n"
    return outputText
```

gaussIntegration.py

```
def gaussPoints(n):
    if(n == 2):
        return [-0.5773502691896257, 0.5773502691896257];
    if(n == 3):
        return [0, -0.7745966692414834, 0.7745966692414834];
    if(n == 4):
        return [-0.3399810435848563, 0.3399810435848563,
-0.8611363115940526, 0.8611363115940526];
    if(n == 5):
        return [0, -0.5384693101056831, 0.5384693101056831,
-0.9061798459386640, 0.9061798459386640];
    if(n == 6):
        return [0.6612093864662645, -0.6612093864662645,
-0.2386191860831969, 0.2386191860831969, -0.9324695142031521,
0.9324695142031521];
    if(n == 7):
        return [0, 0.4058451513773972, -0.4058451513773972,
-0.7415311855993945, 0.7415311855993945, -0.9491079123427585,
0.9491079123427585];
    if(n == 8):
```

```
        return [-0.1834346424956498, 0.1834346424956498,
-0.5255324099163290, 0.5255324099163290, -0.7966664774136267,
0.7966664774136267, -0.9602898564975363, 0.9602898564975363];
    if(n == 9):
        return [0, -0.8360311073266358, 0.8360311073266358,
-0.9681602395076261, 0.9681602395076261, -0.3242534234038089,
0.3242534234038089, -0.6133714327005904, 0.6133714327005904];
    if(n == 10):
        return [-0.1488743389816312, 0.1488743389816312,
-0.4333953941292472, 0.4333953941292472, -0.6794095682990244,
0.6794095682990244, -0.8650633666889845, 0.8650633666889845,
-0.9739065285171717, 0.9739065285171717];

def gaussWeights(n):
    if(n == 2):
        return [1,1]
    if(n == 3):
        return [0.8888888888888888, 0.5555555555555556,
0.5555555555555556]
    if(n == 4):
        return [0.6521451548625461, 0.6521451548625461,
0.3478548451374538, 0.3478548451374538]
    if(n == 5):
        return [0.5688888888888889, 0.4786286704993665,
0.4786286704993665, 0.2369268850561891, 0.2369268850561891]
    if(n == 6):
        return [0.3607615730481386, 0.3607615730481386,
0.4679139345726910, 0.4679139345726910, 0.1713244923791704,
0.1713244923791704]
    if(n == 7):
        return
[0.4179591836734694,0.3818300505051189,0.3818300505051189,0.27970539148
92766,0.2797053914892766, 0.1294849661688697,0.1294849661688697]
    if(n == 8):
        return [0.3626837833783620,0.3626837833783620,
0.3137066458778873, 0.3137066458778873, 0.2223810344533745,
0.2223810344533745, 0.1012285362903763,  0.1012285362903763]
    if(n == 9):
        return [0.3302393550012598, 0.1806481606948574,
0.1806481606948574, 0.0812743883615744, 0.0812743883615744,
0.3123470770400029, 0.3123470770400029, 0.2606106964029354,
0.2606106964029354]
    if(n == 10):
```

```
        return [0.2955242247147529,0.2955242247147529,
0.2692667193099963,
0.2692667193099963,0.2190863625159820,0.2190863625159820,
0.1494513491505806,0.1494513491505806,
0.0666713443086881,0.0666713443086881]

def createOutput(z, w, x, A, counter):
    output = 'Iteração N°' + str(counter)
    output += '\n' + ' ' + 'z: ' + str(z)
    output += '\n' + ' ' + 'w: ' + str(w)
    output += '\n' + ' ' + 'x: ' + str(x)
    output += '\n' + ' ' + 'A: ' + str(A)
    output += '\n\n'
    return output

def gaussIntegration(a, b, n, f):

    outputText = ""

    zPoints = gaussPoints(n)
    weights = gaussWeights(n)
    L = b-a;

    res = 0;
    for i in range(n):
        x = (1/2)*(a+b+zPoints[i]*L)
        res += f.evaluate(x)*weights[i];
        outputText += createOutput(zPoints[i],weights[i],x,res, i)

    outputText +="A integral da função é " + str(res) + ".\n"
    return outputText
```

icod34.py

```
from math import pow

def stepForward(x, deltaX, f):
    fx = f.evaluate(x)
    f_next_delta = f.evaluate(x+deltaX)
    return ((f_next_delta - fx)/deltaX)
```

```
def stepBack(x, deltaX, f):
    fx = f.evaluate(x)
    f_prev_delta = f.evaluate(x-deltaX)
    return ((fx - f_prev_delta)/deltaX)

def centralDifference(x, deltaX, f):
    f_next_delta = f.evaluate(x+deltaX)
    f_prev_delta = f.evaluate(x-deltaX)
    return ((f_next_delta - f_prev_delta)/(2*deltaX))

def richardson(x, delta1, delta2, f):

    d1 = stepForward(x,delta1,f)
    d2 = stepForward(x, delta2,f)

    q = (delta1/delta2)

    return (d1 + ((d1-d2)/(pow(q,-1)-1)))
```