main.py

```
import os
from newton import newton_method
from broyden import broyden_method
import math

def main():

    icod = int(input("ICOD: "))
    teta1 = float(input("teta1: "))
    teta2 = float(input("teta2: "))
    tol = int(input("Potência da tolerância: "))
    tol = math.pow(10,tol)

    outputText = ""

    outputText += "ICOD: " + str(icod) + "\n"
    outputText += "Teta1: " + str(teta1) + "\n"
    outputText += "Teta2: " + str(teta2) + "\n"
    outputText += "Potência: " + str(tol) + "\n"

    fileName = "method_"
    if(icod == 1):
        fileName += "newton"
        outputText += newton_method(teta1, teta2, tol)
    elif (icod == 2):
        fileName += "broyden"
        outputText += broyden_method(teta1, teta2, tol)

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

main();
```

newton.py

```
import numpy as np
import math

def jacobian(c2, c3, c4):
```

```
    return [
        [2*c2, 4*c3, 12*c4],
        [12*c2*c3 + 36*c3*c4, 24*math.pow(c3,2) + 6*math.pow(c2,2) +
36*c2*c4 + 108*math.pow(c4,2), 36*c3*c2 + 216*c3*c4],
        [120*math.pow(c3,2)*c2 + 576*math.pow(c3,2)*c4 +
504*math.pow(c4,2)*c2 + 1296*math.pow(c4,3) + 72*math.pow(c2,2)*c4 + 3,
        240*math.pow(c3,3) + 120*c3*math.pow(c2,2) + 1152*c3*c2*c4 +
4464*c3*math.pow(c4,2),
        576*math.pow(c3,2)*c2 + 4464*math.pow(c3,2)*c4 +
504*c4*math.pow(c2,2) + 3888*math.pow(c4,2)*c2 + 13392*math.pow(c4,3) +
24*math.pow(c2,3)]
    ]

def getPossibleValues(c2, c3, c4, teta1, teta2):
    return [
    2*math.pow(c3,2) + math.pow(c2,2) + 6*math.pow(c4,2) - 1,
    8*math.pow(c3,3) + 6*c3*math.pow(c2,2) + 36*c3*c2*c4 +
108*c3*math.pow(c4,2) - teta1,
    60*math.pow(c3,4) + 60*math.pow(c3,2)*math.pow(c2,2) +
576*math.pow(c3,2)*c2*c4 + 2232*math.pow(c3,2)*math.pow(c4,2) +
    252*math.pow(c4,2)*math.pow(c2,2) + 1296*math.pow(c4,3)*c2 +
3348*math.pow(c4,4) + 24*math.pow(c2,3)*c4 + 3*c2 - teta2
 ]


def getError(delta_x,new_x):
    a = 0
    b = 0
    for i in range(len(delta_x)):
        a += math.pow(delta_x[i],2);
        b += math.pow(new_x[i],2);

    return (a/b)


def createOutput(c2, c3, c4, counter, error):
    output = 'Iteração N°' + str(counter)
    output += '\n' + ' ' + 'c2: ' + str(c2)
    output += '\n' + ' ' + 'c3: ' + str(c3)
    output += '\n' + ' ' + 'c4: ' + str(c4)
    output += '\n' + ' ' + 'Error: ' + str(error)
    output += '\n\n'
    return output
```

```
def newton_method(teta1, teta2, tol):

   error = 1.1
   counter = 0
   c2 = 1
   c3 = 0
   c4 = 0

   outputText = createOutput(c2, c3, c4, counter, 'não definido');

   while error > tol:
       counter += 1

       try:
           j = jacobian(c2, c3, c4)

           possibleValues = getPossibleValues(c2, c3, c4, teta1, teta2)

           newJ = (-1)*(np.linalg.inv(j))

           deltaX = np.matmul(newJ, possibleValues)

           newX = np.zeros(3)

           newX[0] = c2 + deltaX[0]
           newX[1] = c3 + deltaX[1]
           newX[2] = c4 + deltaX[2]

           error = getError(deltaX,newX)

           [c2, c3, c4] = newX

           outputText += createOutput(c2, c3, c4, counter, error)


       except OverflowError as e:
           return outputText + "\n\n\n" + "Ocorreu Oveflow devido a
alguma dificuldade de convergência!"

   return outputText
```

broyden.py

```
from newton import getError, getPossibleValues, createOutput
import numpy as np

def broyden_method(teta1, teta2, tol):

   error = 1.1
   counter = 0
   c2 = 1
   c3 = 0
   c4 = 0

   b = np.identity(3);

   outputText = createOutput(c2, c3, c4, counter, 'não definido')

   while error > tol:

       j = b
       possibleValues = getPossibleValues(c2, c3, c4, teta1, teta2)

       try:
           newJ = np.linalg.inv(j)
       except:
           return "Matrix singular!"

       deltaX = ((-1)*np.matmul(newJ,possibleValues))

       newX = np.zeros(3)

       newX[0] = c2 + deltaX[0]
       newX[1] = c3 + deltaX[1]
       newX[2] = c4 + deltaX[2]

       newPossibleValues = getPossibleValues(newX[0], newX[1], newX[2],
teta1, teta2)

       y = np.zeros(3)
       for i in range(3):
           y[i] = possibleValues[i] + newPossibleValues[i]

       error = getError(deltaX, newX)
```

```
        [c2, c3, c4] = newX

        outputText += createOutput(c2, c3, c4, counter, error)

        newBPart1 = (y-(np.matmul(b,deltaX))*np.transpose(deltaX))
        newBPart2 = (np.matmul(np.transpose(deltaX),deltaX))

        b = b + ( newBPart1/newBPart2 )

    return outputText
```