```
import math
import copy

def jacobi(A, n, TOLm, IDET = 0, itermax = 10000):
    for i in range(n):
        for j in range(n):
            if A[i][j] != A[j][i]:
                print("Erro: A matriz deve ser simétrica
positiva.\n\n")
                output = open("output.txt", "a")
                output.write("Erro: A matriz deve ser simétrica
positiva.\n\n")
                output.close()
                return

    eigenvectors = []
    for i in range(n):
        eigenvectors.append([0]*n)
        eigenvectors[i][i] = 1

    identity = copy.deepcopy(eigenvectors)

    for k in range(itermax):
        largest, posLargest, p = 0, [], copy.deepcopy(identity)

        for i in range(n):
            for j in range(n):
                if i == j: continue
                if abs(A[i][j]) > abs(largest): largest, posLargest =
A[i][j], [i, j]

        if abs(largest) <= TOLm or k == itermax - 1:
            eigenvalues = [0] * n
            det = 1
            for i in range(n): eigenvalues[i] = A[i][i]
            for i in range(len(eigenvalues)): det *= eigenvalues[i]
            return eigenvectors, eigenvalues, k, det


        differenceLineColumn = A[posLargest[0]][posLargest[0]] -
A[posLargest[1]][posLargest[1]]

        if differenceLineColumn == 0: angle = math.pi / 4
```

```
        else: angle = math.atan(2 * largest / differenceLineColumn) / 2

        sin, cos = math.sin(angle), math.cos(angle)

        p[posLargest[0]][posLargest[0]],
p[posLargest[0]][posLargest[1]] = cos, - sin
        p[posLargest[1]][posLargest[0]],
p[posLargest[1]][posLargest[1]] = sin, cos

        pTranspose = transpose(p, n)
        eigenvectors = matrix_multiplication(eigenvectors, p, n)
        pTransposeA = matrix_multiplication(pTranspose, A, n)
        A = matrix_multiplication(pTransposeA, p, n)

def potencia(A, n, TOLm, IDET = 0, path = "", itermax = 10000):
    eigenvector, previousEigenvalue, olderEigenvalues = [0] * n, 1, [1]
* n
    for k in range(itermax):
        for i in range(n):
            sum = 0
            for j in range(n): sum += A[i][j] * olderEigenvalues[j]
            eigenvector[i] = sum
        currentEigenvalue = eigenvector[0]
        #eigenvector /= currentEigenvalue
        eigenvector = [eigen / currentEigenvalue for eigen in
eigenvector]
        error = abs((currentEigenvalue - previousEigenvalue) /
currentEigenvalue)


        if (error <= TOLm or k == itermax - 1):
            det = eigenvector[0] + eigenvector[1] + eigenvector[2]
            print("\nFIM\n")
            print("Iteração " + str(k) + ":")
            print("Erro é de " + str(error))
            print("Autorvalor: " + str(currentEigenvalue))
            print("Autovetor: " + str(eigenvector))
            output = open(path + "output.txt", "a")
            output.write("\nFIM\n")
            output.write("Iteração " + str(k) + ":" + "\n")
            output.write("Erro é de " + str(error) + "\n")
```

```
            output.write("Autorvalor: " + str(currentEigenvalue) +
"\n")
            output.write("Autovetor: " + str(eigenvector)+ "\n")
            output.write("\n")
            if IDET:
                print("A determinante da matrix é: " + str(det))
                output.write("A determinante da matrix é: " + str(det))

            print("\n")
            output.write("\n\n\n")
            output.close()

            return

        print("Iteração " + str(k) + ":")
        print("Erro é de " + str(error))
        print("Autorvalor: " + str(currentEigenvalue))
        print("Autovetor: " + str(eigenvector))
        print("\n")
        output = open(path + "output.txt", "a")
        output.write("Iteração " + str(k) + ":" + "\n")
        output.write("Erro é de " + str(error) + "\n")
        output.write("Autorvalor: " + str(currentEigenvalue) + "\n")
        output.write("Autovetor: " + str(eigenvector)+ "\n")
        output.write("\n")
        output.close()


        previousEigenvalue = currentEigenvalue
        olderEigenvalues = copy.deepcopy(eigenvector)

    return


def transpose(matrix, n):
    result = []
    for i in range(n): result.append([0]*n)
    for i in range(n):
       for j in range(n):
           result[j][i] = matrix[i][j]
    return result

def matrix_multiplication(matrixA, matrixB, n):
```

```python
    result = []
    for i in range(n): result.append([0]*n)
    for i in range(n):
       for j in range(n):
           for k in range(n):
               result[i][j] += matrixA[i][k] * matrixB[k][j]
    return result

path = ""
with open(path + 'input.csv', 'r') as entrada:
    lineCounter = 0
    matrizA, vectorB = [], []
    for line in entrada:
        if lineCounter == 0:
            n = int(line)
            lineCounter += 1
            continue
        elif lineCounter == 1:
            ICOD = int(line)
            lineCounter += 1
            continue
        elif lineCounter == 2:
            IDET = float(line)
            lineCounter += 1
            continue
        elif lineCounter == 3:
            Tolm = float(line)
            lineCounter += 1
            continue
        else:
            matrizA.append([float(x) for x in line.split(',')])
            lineCounter += 1

entrada.close()


print("Arquivos lidos, iniciando algoritmo selecionado")

if ICOD == 1:
    out = jacobi(matrizA, n, Tolm, IDET)
    print("Método de Jacobi:\nAutovetores estimados: " + str(out[0]))
    print("Autovalores encontrados: " + str(out[1]))
    print("Iterações para convergência: " + str(out[2]))
```

```
    output = open(path + "output.txt", "a")
    output.write("Método de Jacobi:\nAutovetores estimados: " +
str(out[0]) + "\n")
    output.write("Autovalores encontrados: " + str(out[1]) + "\n")
    output.write("Iterações para convergência: " + str(out[2]) + "\n")
    if IDET:
        print("A determinante da matriz é " + str(out[3]) + "\n")
        output.write("A determinante da matriz é " + str(out[3]) +
"\n")
    output.write("\n\n")
    output.close()

elif ICOD == 2:
    print("Método da Potência")
    output = open(path + "output.txt", "a")
    output.write("Método de Potência")
    output.close()
    potencia(matrizA, n, Tolm, IDET)
```