www.estudar.com.vc

Lista de Exercícios

P3 2016

Programação em Python Poli

USP

1. Tema

Podemos calcular uma aproximação da raiz quadrada de um número através de uma série. A ideia desse método é devida a *Isaac Newton*, que propôs um método geral para encontrar zeros de funções, ou seja, números que quando aplicados na função levam ao resultado zero.

O primeiro termo da série para achar a raiz quadrada de um número x é o próprio x. Dado um termo x_i qualquer, podemos calcular o próximo através da fórmula abaixo:

$$x_{i+1} = \frac{x_i + \frac{x}{x_i}}{2}$$

Por exemplo, se aplicarmos o método para $x = 4$, vamos obter os seguintes termos:

$$x_0 = 4$$

$$x_1 = 2.5$$

$$x_2 = 2.05$$

$$x_3 = 2.000609756097561$$

$$x_4 = 2.000000929222947 \ldots$$

Escreva uma função, cujo protótipo está definido a seguir, que recebe um número real x e um número inteiro $n \geq 0$ e retorna como aproximação da raiz quadrada de x o número real x_n obtido pelo método definido acima.

```
def raiz_quadrada(x, n):
    '''
    (float, int) -> float
    Recebe um real x e um inteiro n e retorna o valor de
    x_n como aproximação para a raiz quadrada de x.
    '''
```

2. Tema

Nessa questão vamos implementar uma classe para números complexos. Um número complexo é representado por um par (a, b) de números reais: parte real e parte imaginária. O número complexo representado é $a + bi$ onde $i = \sqrt{-1}$. Considere abaixo a descrição da classe em Python:

```
class Complexo:
    '''
    Classe que representa números complexos. Cada
    objeto da classe tem dois atributos p_real e p_imag,
    onde p_real tem a parte real e p_imag a parte
    imaginária do número.
    '''

    def __init__(self, p_real = 0, p_imag = 0):
        '''
```

```
        Construtor da classe. Recebe dois números reais
        e constrói um objeto da classe Complexo, com o
        par de números como atributos.
        '''
        self.p_real = p_real
        self.p_imag = p_imag

    def __str__(self):
        '''
        Retorna um string para impressão de um número
        complexo.
        '''
        if self.p_imag > 0:
            texto = "%.2f + %.2fi" %(self.p_real,
            self.p_imag)
        elif self.p_imag < 0:
            texto = "%.2f - %.2fi" %(self.p_real,
            -self.p_imag)
        else:
            texto = "%.2f" %(self.p_real)
        return texto
```

a. Implemente os métodos a seguir:

```
def __add__(self, other):
    '''
    (Complexo, Complexo) -> Complexo
    Retorna a soma dos números complexos self e other.
    Exemplos: (4 - 3i) + (2 + 4i) = 6 + i, (12 - 5i) +
    (0 + 3i) = 12 - 2i, (2 - 1i) + (-2 + 0i) = 0 - 1i
    '''
```

```
def __mul__(self, other):
    '''
    (Complexo, Complexo) -> Complexo
    Retorna o produto dos números complexos self e other.
    Exemplos: (3 - 2i) * (5 + 4i) = 23 + 2i,
    (12i) * (3 - i) = 12 + 36i
    '''

def modulo(self):
    '''
    (Complexo) -> float
    Retorna o módulo do número complexo self. Exemplos: O
    módulo de 4 - 3i é 5. O módulo de 3i é 3. O módulo de 1
    + i é 1.4142... Use a função raiz_quadrada(x, n), que
    recebe um real x e um inteiro n e retorna o valor de x_n
    como aproximação para a raiz quadrada de x, com n = 100.
    '''

def prod_escalar (self, c):
    '''
    (Complexo, float) -> Complexo
    Retorna o produto do complexo self com o escalar c.
    Exemplos: Para o complexo (2 + 3i) e c = -2, deve
    retornar -4 - 6i. Para o complexo (10 + 2i) e c = 0,
    deve retornar 0 + 0i.
    '''
```

b. Implemente o método exponencial:

```
def exponencial(self):
    '''
    (Complexo) -> Complexo
```

```
    Retorna uma aproximação para e^self usando a expansão
    de Taylor: e^self = 1 + self + self^2/2! + self^3/3! +
    self^4/4! + ... + self^n/n! + ...
    Inclua na aproximação todos os termos até o primeiro que
    seja, em módulo, menor ou igual a 10^{-6}.
    '''
```

3. TEMA

Dizemos que uma matriz A com m linhas e n colunas tem banda nula $k \leq m$ se as k diagonais de A, começando a partir do canto inferior esquerdo, são nulas. Veja os exemplos abaixo:

$$\begin{pmatrix} 1 & 2 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$ tem banda nula 3

$$\begin{pmatrix} 4 & 2 & 1 & 5 \\ 0 & 3 & 2 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$ tem banda nula 2

$$\begin{pmatrix} 2 & 1 \\ 0 & 2 \\ 2 & 1 \\ 0 & 0 \end{pmatrix}$$ tem banda nula 1

$\begin{pmatrix} 2 & 1 \\ 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}$ tem banda nula 3

$\begin{pmatrix} -1 & 2 & 1 \\ 0 & 3 & 2 \\ 0 & 4 & 1 \\ 3 & 0 & 0 \end{pmatrix}$ tem banda nula 0

$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$ tem banda nula 2

Escreva um programa que lê inteiros positivos m e n e uma matriz A_{mxn}, e imprime o maior k tal que $0 \leq k \leq m$ e a matriz A tem banda nula k.

Gabarito

1.
```
def raiz_quadrada(x,n):
    i = 0
    xi = x
    while i < n:
        xi = (xi + (x/xi))/2
        i += 1
    return xi
```

2.

a.
```
def __add__(self, other):
    return Complexo(self.p_real + other.p_real, self.p_imag
    + other.p_imag)


def __mul__(self, other):
    a = self.p_real
    b = self.p_imag
    c = other.p_real
    d = other.p_imag
    return Complexo(a*c - b*d, a*d + b*c)

def modulo(self):
    return  raiz_quadrada(self.p_real**2  +  self.p_imag**2,
    100)

def prod_escalar (self, c):
    return Complexo(c*self.p_real, c*self.p_imag)
```

b.

```
def fat(n): #essa declaração é feita fora da classe
     r = 1
     while n > 1:
          r *= n
          n -= 1
      return r

def pot(self, n):
     i = 0
     res = Complexo(1,0)
     while i < n:
          res *= self
          i += 1
     return res

def exponencial(self):
     Parar = False
     aprox = Complexo(1, 0) + self
     i = 2
     while not Parar:
          termo = (self.pot(i)).prod_escalar(1/fat(i))
          aprox += termo
          i += 1
          if termo.modulo()<0.000001:
               Parar = True
     return aprox
```

3.

```
def banda_nula(M):
     i = 0
     while(i<len(M)):
```

```
            i += 1
            for j in range(i):
                if j >= len(M[0]):
                    break
                if M[-i+j][j] != 0:
                    return i - 1
        return i


def main()
        m = int(input('Digite m: '))
        n = int(input('Digite n: '))
        A = []
        for i in range(m):
            linha = []
            for j in range(n):
                x = int(input())
                linha.append(x)
            A.append(linha)
        print(banda_nula(A))

main()
```