Cálculo Numérico

Trabalho 1

Aline Werner

3 de abril de 2018

Introdução

Este trabalho tem como objetivo colocar em prática os conceitos aprendidos durante as aulas de Cálculo Numérico a respeito de erros e aproximações, principalmente através da programação na linguagem Python.

Questão 1

Pergunta: O que é o Épsilon da Máquina? Apresente um programa em Python para obter o épsilon do computador que você usa. Qual o valor obtido com seu programa?

Resposta: Em um sistema de ponto flutuante, o Épsilon da Máquina, aqui representado por *eps*, corresponde ao menor número que, somado ao número 1, resulta em um número diferente de 1, ou seja, não é arredondado. Em outras palavras, *eps* é o menor float tal que $(1 + eps) > 1$. Esse valor também é chamado de unidade de arredondamento e representa a exatidão relativa da aritmética do computador.

Para obter o épsilon de meu computador, utilizei o programa em Python abaixo. Ele executa um loop que divide o valor do épsilon definido no início do programa pela metade enquanto o valor resultante desta divisão, somado a 1, continuar maior que 1. Quando isso não mais acontece, o loop é encerrado e o último valor é multiplicado por 2, para que o épsilon seja obtido.

```
eps = 1
while ((eps+1)>1):
        eps = eps/2

eps = eps*2

print(eps)
```

Figura 1: Programa que calcula o épsilon da máquina

O valor encontrado pelo programa foi $2.220446049250313e^{-16}$, ou seja, um número inimaginavelmente pequeno, mas que, como foi discutido em aula, pode levar a alguns resultados "estranhos" ao executarmos certas operações aritméticas.

Questão 2

Pergunta: Considere as expressões

$$\frac{e^{1/x}}{1+e^{1/x}} \quad e \quad \frac{1}{e^{-1/x}+1}$$

Verifique que, para $x > 0$, são funções idênticas, então, use um programa em Python para testar o valor de cada uma para alguns valores de x entre 0.1 e 0.001. Qual dessas expressões é mais adequada quando x é um número pequeno? Explique.

Resposta: Multiplicando a expressão $\frac{e^{1/x}}{1+e^{1/x}}$ por $\frac{e^{-1/x}}{e^{-1/x}}$, obtemos a expressão $\frac{1}{e^{-1/x}+1}$ que, para $x > 0$, são funções idênticas.

Para testar o valor das duas funções para alguns números entre 0.1 e 0.001, utilizei o programa em Python abaixo, no qual $f(x)$ corresponde à primeira e $g(x)$ à segunda função.

```
import math
e = math.e

f = lambda x: (e**(1/x))/(1+(e**(1/x)))
g = lambda x: 1/((e**(-1/x))+1)

x = 0.1
while(x>0.001):
        print(x, g(x))
        x = x - 0.0001

x = 0.1
while(x>0.001):
        print(x, f(x))
        x = x - 0.0001
```

Figura 2: Programa que calcula o valor das funções para alguns números

Ao executar o programa, a função $g(x)$, testada por primeiro, correu normalmente. Os últimos valores de x e $g(x)$ mostrados foram, respectivamente, 0.001099999999998172 e 1.0. A função $f(x)$, porém, apresentou um erro de Overflow para valores abaixo de $x = 0.0014999999999981723$, indicando que $f(x)$ não é representável para esses valores no sistema de ponto flutuante em questão.

Questão 3

Pergunta: O número e pode ser definido pela série $e = \sum_{n=0}^{\infty} \left(\frac{1}{n!}\right)$. Apresente um programa em Python para obter uma aproximação para e com erro relativo menor do que 0.0001.

Resposta: O programa em Python que utilizei para aproximar e está representado na Figura 3. Em primeiro lugar, há uma função que calcula o fatorial de um dado número, que será utilizada na série. Em seguida, são definidos os valores iniciais do erro, do n do somatório e da variável e-ant, que é utilizada para armazenar o valor de e que será usado na iteração seguinte. Após isso, há um loop que aproxima o valor de e através da série dada e apresenta na tela o valor de e calculado nessa iteração, na iteração anterior e o erro relativo entre elas. Ele é executado enquanto o erro é maior ou igual a 0.0001. Por fim,

fora do loop há mais uma aproximação, que encontra o próximo valor de e e tem erro relativo menor do que 0.0001. Os resultados obtidos com o programa podem ser vistos na Figura 4.

```
def fat(n):
        if n==0 or n==1:
                return 1
        else:
                return n*fat(n-1)

err = 100.0
n = 0
e_ant = 0

while(err>=0.0001):
        e = e_ant + 1/fat(n)
        print(e, e_ant, err)

        err = (e - e_ant)/e
        e_ant = e
        n = n+1

e = e_ant + 1/fat(n)
print(e, e_ant, err)
```

Figura 3: Programa que aproxima o valor de e

```
1.0 0 100.0
2.0 1.0 1.0
2.5 2.0 0.5
2.6666666666666665 2.5 0.2
2.708333333333333 2.6666666666666665 0.062499999999999944
2.7166666666666663 2.708333333333333 0.015384615384615332
2.7180555555555554 2.7166666666666663 0.0030674846625766768
2.7182539682539684 2.7180555555555554 0.0005109862033725888
2.71827876984127 2.7182539682539684 7.299270073000844e-05
```

Figura 4: Resultados obtidos com o programa acima

Questão 4

Pergunta: A fórmula de Leibniz para o número π é dada pela série infinita $\frac{\pi}{4} = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$. Apresente um programa em Python para obter uma aproximação para π usando 50 termos da série.

Resposta: Para obter uma aproximação de π através da série acima, utilizei o seguinte programa em Python. No início dele, são definidos oa valores iniciais para n e para a variável pi-ant, que armazena o valor aproximado para π da iteração anterior. Em seguida, há um loop, executado 50 vezes, que aproxima π e o mostra na tela usando, a cada iteração, um termo a mais da série, totalizando 50 termos. O último valor aproximado para π obtido com o programa foi 3.121594652591011.

Questão 5

Pergunta: Apresente um programa em Python para obter aproximações para o valor da função $f(x) = ln(1+x)$ usando expansões em séries de Taylor em torno do ponto $x = 0$. Descubra quantos termos da série precisam ser retidos para calcular $ln(0.8)$ com erro absoluto inferior a 0.0001.

```
n = 0
pi_ant = 0

for x in range(50):
        pi = pi_ant + 4*(((-1)**n)/(2*n+1))
        print(pi)

        pi_ant = pi
        n = n+1
```

Figura 5: Programa que aproxima π

Resposta: A série de Taylor para $f(x) = ln(1+x)$ em torno do ponto $x = 0$ pode também ser escrita como $\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n} x^n$, somatório que pode ser encontrado após fazer as derivadas e perceber o padrão existente ao longo da série encontrada. Esse somatório foi utilizado no programa em Python a seguir, que inicia com a atribuição de valores para as variáveis "ln-ant", que irá armazenar o valor aproximado na iteração anterior, para "n" e para "x", sendo este último um valor arbitrário, utilizado apenas para teste em um primeiro momento. Em seguida, foi criado um loop que aproxima o valor de $ln(1+x)$ utilizando um termo a mais em cada iteração, totalizando 10, também para fins de teste.

```
ln_ant = 0
n = 1
x = 0

for n in range(1, 11, 1):
    ln = ln_ant + (((-1)**(n+1))/n)*x**n
    print('Valor com ' + str(n) + ' termos da série = ', ln)
    ln_ant = ln
```

Figura 6: Série de Taylor para $f(x) = ln(x+1)$

```
Valor com 1 termos da série =  0.0
Valor com 2 termos da série =  0.0
Valor com 3 termos da série =  0.0
Valor com 4 termos da série =  0.0
Valor com 5 termos da série =  0.0
Valor com 6 termos da série =  0.0
Valor com 7 termos da série =  0.0
Valor com 8 termos da série =  0.0
Valor com 9 termos da série =  0.0
Valor com 10 termos da série =  0.0
```

Figura 7: Resultados obtidos com o programa acima

Para a segunda parte da questão, ou seja, aproximar $ln(0.8)$ com erro absoluto inferior a 0.0001, foram feitas algumas modificações no programa. A váriável "x" foi definida como -0.2, já que o programa serve para calcular $ln(1+x)$ e é necessário calcular $ln(0.8)$. O loop anterior foi substituído por um que se encerra ao atingir um valor para o erro inferior a 0.0001, sendo que o erro foi encontrado através da subtração do valor aproximado do $ln(0.8)$ do seu valor real. O programa pode ser conferido na Figura 8 e os resultados obtidos na Figura 9. De acordo com o programa, foram retidos apenas 4 termos da série para calcular $ln(0.8)$ com erro absoluto inferior a 0.0001.

```python
import math

ln_ant = 0
n = 1

err = 100
x = -0.2

while(err>=0.0001):
    ln = ln_ant + (((-1)**(n+1))/n)*x**n
    err = ln - math.log(0.8)
    print('Valor com ' + str(n) + ' termos = ', ln)
    print('Erro = ', err)

    ln_ant = ln
    n = n+1
```

Figura 8: Programa que aproxima $ln(0.8)$

```
Valor com 1 termos =  -0.2
Erro =  0.023143551314209698
Valor com 2 termos =  -0.22000000000000003
Erro =  0.0031435513142096805
Valor com 3 termos =  -0.2226666666666667
Erro =  0.0004768846475430022
Valor com 4 termos =  -0.22306666666666672
Erro =  7.688464754299074e-05
```

Figura 9: Resultados obtidos com o programa acima