t UFABC

PROCESSAMENTO DA INFORMAÇÃO
CENTRO DE MATEMÁTICA, COMPUTAÇÃO E COGNIÇÃO
UNIVERSIDADE FEDERAL DO ABC
PROF. MAYCON SAMBINELLI

Lista 3: Comandos de repetição

1. Qual o resultado da chamada `Magic(10, 14)`?

```
def Magic(x, y):
    if x > y:
        r = x
        x = y
        y = r
    r = 0
    while x <= y:
        r = r + x
        x = x + 1
    return r
```

2. Faça um algoritmo que imprima as n primeiras potências de 2, em ordem crescente, para $n \geq 0$.

3. Faça um algoritmo que calcule a média, o maior número e o menor número de n números inteiros.

4. Faça um algoritmo que recebe n inteiros e decide se a sequência dada é crescente ou não.

5. Crie uma função em que, dado um inteiro $n > 0$, seja realizada o seguinte somatório:

$$1 - 2 + 3 - 4 + 5 - 6 + 7 ... \pm n$$

6. Na Mega-Sena, um jogo consiste de 6 números distintos com valores entre 1 e 60. Faça um algoritmo que imprime todos os jogos possíveis da Mega-Sena, sem repetição.

7. Faça um algoritmo que imprime os primeiros n números da série de Fibonacci.

8. Faça um algoritmo que calcule o maior divisor em comum entre dois números **usando a definição**: é o maior número inteiro positivo que é divisor desses números.

9. Faça um algoritmo que calcule o maior divisor em comum entre dois números **usando o algoritmo de Euclides**. A ideia básica do algoritmo é a seguinte:

   - Divida o maior número pelo menor. Anote o resto dessa divisão.
   - Substitua o número maior pelo número menor e o número menor pelo resto obtido na divisão anterior.
   - Continue dividindo o número maior pelo menor, substituindo os valores a cada passo, até que o resto da divisão seja zero.
   - Quando o resto se tornar zero, o último divisor não-nulo será o maior divisor em comum dos dois números iniciais.

10. Faça um algoritmo que calcule uma aproximação da área sob a curva da função $f(x) = 40x^3 - 6x^2 + 4x + 12$ entre x_{ini} e x_{fim} usando a técnica de aproximação por retângulos. O intervalo

$[x_{ini}, x_{fim}]$ será dividido em n subintervalos, onde x_{ini}, x_{fim} e n são fornecidos como entrada pelo usuário. A figura a seguir ilustra a ideia da aproximação por retângulos para uma outra função [1].

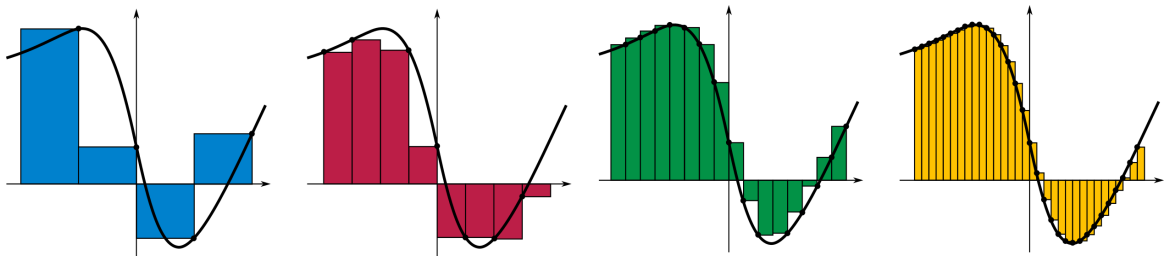


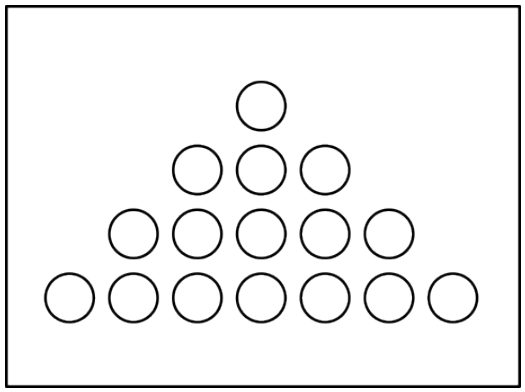
Figura 1:

11. Faça um programa que ilustre a curva e os retângulos sob a curva como na Figura 1. Além do usuário fornecer o intervalo no qual o mesmo deseja o visualizar a função, permita que o usuário também forneça o número de retângulos no qual deseja particionar o intervalo.

12. Escreva uma função em Python chamada *meuseno*(x) que compute $\sin(x)$ usando a expansão da Série de Taylor:
$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

13. Escreva um programa que plota a função $\sin(x)$ em um dado intervalo $[x_{ini}, x_{fim}]$. Use a biblioteca `stddraw`.

14. Escreva um programa que, dado um valor n, imprime um padrão em tela usando a biblioteca `stddraw`. A figura a seguir mostra o padrão quando $n = 4$.



15. **Conjectura de Goldbach** A Conjectura de Goldbach, proposta pelo matemática Prussiano Chistian Goldbach, é um dos problemas não resolvidos da matemática mais antigos. O problema foi proposto em 1742 e afirma que *"todo número par maior do que* 2 *pode ser escrito como a soma de dois primos (não necessariamente distintos)"*. Veja abaixo que a conjectura é válida para os primeiros números pares:

[1] Note que, na realidade, estou pedindo para vocês escreverem um programa para computar $\int_a^b (40x^3 - 6x^2 + 4x + 12)$, onde $a = x_{ini}$ e $b = x_{fim}$. Ao substituir a função $40x^3 - 6x^2 + 4x + 12$ por uma outra função contínua $f(x)$, o programa se torna capaz de resolver a integral $\int_a^b f(x)$. Dessa forma, vocês terão uma ferramenta útil para verificar as integrais definidas que resolverem ao longo do curso de cálculo!.

$$
\begin{aligned}
4 &= 2+2 \\
6 &= 3+3 \\
8 &= 5+3 \\
10 &= 5+5 \\
12 &= 5+7 \\
&\vdots
\end{aligned}
$$

Usando os seus conhecimentos de programação, escreva um programa que verifique se a Conjectura de Goldbach é verdadeira para os primeiros n números naturais![2][3]

16. **Algoritmo de Newton-Raphson.** Existe uma fórmula bem conhecida para resolver equações quadráticas de segundo grau (ax2+bx+c=0)(ax2+bx+c=0). Para equações de terceiro e quarto grau, também existem fórmulas fechadas, mas estas são extremamente complexas. No entanto, para polinômios de grau cinco ou superior, foi demonstrado que não há uma solução geral por meio de expressões algébricas, conforme provado pelo teorema de Abel-Ruffini.

    Neste exercício, exploraremos um algoritmo amplamente utilizado por calculadoras e computadores para encontrar aproximações das raízes de uma equação. O método empregado é conhecido como o *Algoritmo de Newton-Raphson*.

    Queremos utilizar o Algoritmo de Newton-Raphson para determinar um valor r tal que $f(r) = 0$.

    A ideia central do método é que a reta tangente a um ponto de uma curva pode ser usada para obter aproximações sucessivamente melhores para uma raiz da equação $f(x)$. Esse princípio geométrico permite refinarmos iterativamente uma estimativa inicial até alcançarmos uma solução com a precisão desejada.

    Para ilustrar o método, considere a função

    $$f(x) = x^2 - 16.$$

    Da mesma forma que o método de Heron, começamos com uma estimativa inicial para a raiz. Aqui, esse valor inicial será fornecido pelo usuário do programa, que pode obter esse palpite através da inspeção do gráfico da função. Quanto melhor a escolha inicial, maior a chance de sucesso do algoritmo. Infelizmente, o *Algoritmo de Newton-Raphson* pode falhar e não convergir para uma solução caso a estimativa inicial seja inadequada.

    Nosso objetivo é encontrar r. Vamos chamar a estimativa inicial de x_1. Note que a reta tangente ao ponto $(x_1, f(x_1))$ intersecta o eixo x no ponto x_2 (veja a Figura 2). Esse novo valor x_2 fornece uma aproximação melhor para a raiz r.

    Agora, aplicando o mesmo raciocínio ao ponto x_2:

    - Analisamos a reta tangente à curva no ponto $(x_2, f(x_2))$.
    - Definimos x_3 como sendo o ponto no qual a reta tangente ao ponto $(x_2, f(x_2))$ intersecta o eixo x.

[2] Se o seu programa encontrar um número par maior do que dois que não pode ser escrito como a soma de dois primos, então: (i) o seu programa está errado; ou (ii) você vai ficar muito famoso por ter resolvido um dos problemas mais antigos da Matemática!

[3] Qual o maior n que o seu programa (computador) consegue verificar a conjectura?

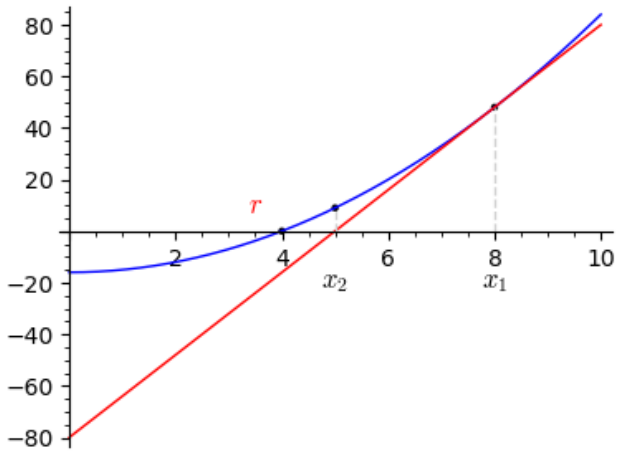


Figura 2:

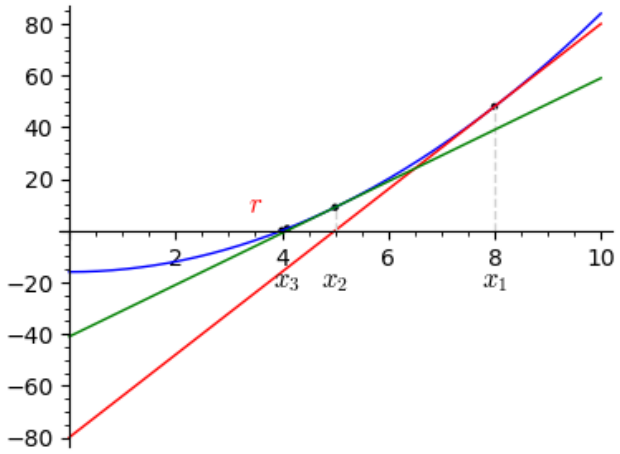


Figura 3:

Note que o valor x_3 é uma aproximação para r ainda melhor do que x_2 (veja a Figura 3). Podemos aplicar esse raciocínio repetidamente, gerando uma sequência $x_1, x_2, \ldots, x_\ell$ de valores que representam aproximações sucessivamente melhores.

Como estamos interessados apenas na melhor aproximação, podemos armazenar apenas a última aproximação (ou as duas últimas). O processo continua até que a diferença entre dois valores consecutivos, x_i e x_{i+1} seja menor do que um erro pré-estabelecido. Assim, obtemos x_ℓ como uma aproximação para a raiz r.

Agora vamos descrever como obter x_{i+1} de x_i. Primeiro, lembre-se da equação fundamental da reta:

$$y - y_i = m(x - x_i) \quad ,$$

onde m é a inclinação da reta e (x_i, y_i) é um ponto na reta. Todos os pares de valores (x, y) que satisfazem a equação acima pertencem a reta.

Dada a curva contínua $f(x)$, sabemos que a inclinação da reta tangente à f no ponto $(x_i, f(x_i))$ é $f'(x_i)$. Substituindo na equação fundamental da reta, temos

$$y - f(x_i) = f'(x_i)(x - x_i) \quad .$$

Essa equação nos dá todos os pontos sobre a reta tangente, mas estamos interessados no ponto que que intersecta o eixo x, ou seja, onde $y = 0$. Denotaremos a coordenada x desse ponto por x_{i+1}. Assim,

$$0 - f(x_i) = f'(x_i)(x_{i+1} - x_i) \Leftrightarrow x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad .$$

Assim, se tivermos sorte e $f'(x_i) \neq 0$, podemos obter o próximo valor da aproximação, x_{i+1}, pela fórmula

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \quad .$$

(a) Usando a sua implementação do Algoritmo de Newton-Raphson, encontre as raízes de

$$f(x) = x^3 - 6x^2 + 4x + 12$$

(b) Faça um programa que ilustre o Método de Newton-Rapson gerando uma imagem semelhante à Figura 3 (você consegue animar o método?).

17. **Ruína do Jogador.** A simulação é um recurso fundamental na ciência, pois permite validar empiricamente resultados teóricos e obter, de forma mais acessível, respostas que seriam difíceis de alcançar por meio de análises matemáticas puramente analíticas.

Neste exercício, exploraremos um problema clássico denominado *"A Ruína do Jogador"*. Suponha que um jogador vá a um cassino com um montante inicial M. No cassino, ele realiza uma série de apostas justas de \$1, onde a probabilidade de ganhar ou perder cada aposta é de 50%. Se o jogador continuar apostando indefinidamente, inevitavelmente irá à ruína. Entretanto, se adotar a estratégia de "*cair fora*" ao atingir um valor alvo G, a probabilidade de sucesso (ou seja, de evitar a ruína e obter lucro) torna-se maior que zero.

No problema da Ruína do Jogador, dado um montante inicial M e um valor objetivo G, buscamos determinar a probabilidade de sucesso. Você conseguiria calcular esse valor analiticamente? Provavelmente não, mas já é capaz de escrever um programa que forneça uma boa estimativa! Para isso, você irá simular T rodadas (idas ao cassino) e calcular a razão entre o número de vitórias e T. Quanto maior for T, melhor será a aproximação da probabilidade real.

Para simular uma ida ao cassino, comece inicializando uma variável, por exemplo, `money`, com o valor inicial M. Para simular uma aposta, utilize a função `random()` do Python. Esta função retorna um número pseudo-aleatório[4] no intervalo $[0,1)$ a cada chamada. Assim, se `random() < 0.5`, podemos considerar que o jogador ganhou a aposta. Para utilizar a função `random()` é necessário escrever a seguinte linha no início do programa: `from random import random`.

Caso o jogador ganhe a aposta, incremente a variável `money` em 1; caso contrário, decremente `money` em -1. Continue a simulação até que o jogador vá à ruína (`money = 0`) ou atinja o seu objetivo (`money = G`). Ao final de cada simulação, contabilize se o jogador foi à ruína ou saiu vitorioso.

Em seguida, inicie uma nova simulação até atingir o número total de simulações especificado por T. Com o total de vitórias e o número total de simulações, será possível determinar a probabilidade de vitória no cassino sem precisar fazer cálculos analíticos.

[4]Aqui usamos o termo pseudo-aleatório, pois produzir números verdadeiramente aleatórios é um problema muito complexo. O que a função `random()` e similares fazem é gerar números que parecem aleatórios, mas que não são verdadeiramente aleatórios. A seguir indico um vídeo com uma discussão interessante sobre o problema de gerar números aleatórios https://www.youtube.com/watch?v=1cUUfMeOijg.

18. **Lançamento Livre.** Vamos criar um programa que simule o lançamento de um projétil. Nesta simulação, representaremos o projétil por um círculo e a base de lançamento por um retângulo. Além disso, desprezaremos a influência do vento na simulação. O seu simulador deverá gerar uma animação similar à mostrada no link: http://professor.ufabc.edu.br/~m.sambinelli/courses/2025Q1-PI/figs/canon.gif.

    O programa receberá como entrada a velocidade escalar inicial v_0 (m/s) e o ângulo de lançamento θ, em graus, com relação ao solo. O projétil começa na posição $(0,0)$ no nosso sistema de coordenadas. Para calcular a posição (x,y) do projétil em um dado tempo t (s), você pode usar as seguintes equações:

$$x = v_0 t \cos(\theta), \tag{1}$$

$$y = v_0 t \sin(\theta) - \frac{1}{2} g t^2, \tag{2}$$

    onde g é a aceleração devido à gravidade, com valor de $g = 9.81\,\text{m/s}^2$.

    Você pode utilizar as funções $\sin(x)$ e $\cos(x)$ do Python para calcular o seno e o cosseno, respectivamente. Para utilizá-las, deve escrever as seguintes linhas no início do seu programa:

```
from math import sin
from math import cos
```

    As funções $\sin(x)$ e $\cos(x)$ esperam que o valor x do ângulo seja dado em radianos. Como é mais natural para o usuário do seu programa fornecer o ângulo em graus, será necessário fazer a conversão. A relação para converter graus em radianos é dada por:

$$\text{radianos} = \text{graus} \times \frac{\pi}{180}.$$