MINISTÉRIO DA DEFESA
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
Seção de Engenharia de Defesa (SE/10)

# EE 600200: Álgebra Linear Computacional

Lista de Exercícios #3

Cap Filipe José Americano Albeny

Rio de Janeiro, RJ
Maio de 2025

# 1ª Questão

# Ford - 9.2

Give the flop count for each matrix operation.

## a) Multiplication of m × n matrix A by an n × 1 vector x.

Dada a multiplicação de uma matriz $A \in \mathbb{R}^{m\times n}$ por um vetor coluna $x \in \mathbb{R}^{n\times 1}$, queremos determinar o número de operações de ponto flutuante (flops) necessárias para computar o produto $y = Ax$, onde $y \in \mathbb{R}^{m\times 1}$.

Para cada uma das $m$ linhas da matriz $A$, o cálculo de uma entrada $y_i$ envolve:

- $n$ multiplicações: $A_{i1}x_1 + A_{i2}x_2 + \cdots + A_{in}x_n$
- $n - 1$ somas para acumular os resultados

Assim, o total de flops é dado por:

$$\text{Total de flops} = m \cdot n \text{ (multiplicações)} + m \cdot (n-1) \text{ (somas)} = m(2n-1)$$

**Resposta final:** $\boxed{m(2n-1)}$

## b) The product $xy^\top$ if $x$ is an $m \times 1$ vector and $y$ is a $p \times 1$ vector.

Considere os vetores $x \in \mathbb{R}^{m\times 1}$ e $y \in \mathbb{R}^{p\times 1}$. O produto $xy^\top$ resulta em uma matriz $m \times p$, onde cada entrada é dada por:

$$(xy^\top)_{ij} = x_i \cdot y_j$$

Para calcular todas as $m \cdot p$ entradas da matriz resultante, são necessárias:

- $m \cdot p$ multiplicações

Portanto, o número total de operações de ponto flutuante (flops) é:

$$\boxed{m \cdot p}$$

## c) If $u$ and $v$ are $n \times 1$ vectors, the computation of $\left(\langle v, u\rangle / \|u\|^2\right) u$.

Sejam $u, v \in \mathbb{R}^{n\times 1}$. Queremos calcular o número de flops necessários para computar:

$$\left(\frac{\langle v, u\rangle}{\|u\|^2}\right) u$$

Essa operação envolve os seguintes passos:

1. Cálculo do produto interno $\langle v, u\rangle$: $n$ multiplicações e $n-1$ somas, totalizando $2n-1$ flops.
2. Cálculo da norma ao quadrado $\|u\|^2 = \langle u, u\rangle$: também $2n-1$ flops.
3. Uma divisão escalar: 1 flop.
4. Multiplicação escalar do resultado por $u$: $n$ multiplicações, totalizando $n$ flops.

Somando tudo:

$$(2n-1) + (2n-1) + 1 + n = \boxed{5n-1}$$

## d) $\|A\|_\infty$ for an $m \times n$ matrix $A$

Seja $A \in \mathbb{R}^{m\times n}$. A norma infinito da matriz é definida por:

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}|$$

Para cada uma das $m$ linhas, a soma de $n$ elementos exige $n-1$ somas. Logo, o total de operações de ponto flutuante (flops) necessárias para calcular $\|A\|_\infty$ é:

$$\boxed{m(n-1)}$$

## e) $\|A\|_1$ for an $m \times n$ matrix $A$.

Seja $A \in \mathbb{R}^{m\times n}$. A norma 1 da matriz é definida por:

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}|$$

Para cada uma das $n$ colunas, a soma de $m$ elementos exige $m-1$ somas. Logo, o total de operações de ponto flutuante (flops) necessárias para calcular $\|A\|_1$ é:

$$\boxed{n(m-1)}$$

## f) trace (A),where A is an n $\times$ n matrix.

Seja $A \in \mathbb{R}^{n \times n}$. A traça da matriz é definida por:

$$\text{trace}(A) = \sum_{i=1}^{n} a_{ii}$$

Para calcular a soma dos $n$ elementos da diagonal principal, são necessárias $n-1$ somas. Portanto, o total de operações de ponto flutuante (flops) é:

$$\boxed{n-1}$$

## 2ª Questão

### a) [FORD] Exercício 9.18, utilizando Python ao invés de MATLAB

## Código Python Completo

Listing 1: Script Python para Produto Vetorial e Cálculos

```
import numpy as np

def crossprod(u, v):
    """
    Computes the cross product of two 3D vectors u and v
       .

    Args:
        u: A 1D NumPy array or list of length 3
           representing the first vector.
        v: A 1D NumPy array or list of length 3
           representing the second vector.

    Returns:
        A 1D NumPy array of length 3 representing the
           cross product u x v.

    Raises:
        ValueError: If the input vectors are not 3-
           dimensional.
    """
    u_arr = np.asarray(u)
    v_arr = np.asarray(v)

    if u_arr.shape != (3,) or v_arr.shape != (3,):
        raise ValueError("Both vectors must be 3-
           dimensional.")

    # u = [u1, u2, u3] -> u_arr[0], u_arr[1], u_arr[2]
    # v = [v1, v2, v3] -> v_arr[0], v_arr[1], v_arr[2]

    # c1 = u2*v3 - u3*v2
    c1 = u_arr[1] * v_arr[2] - u_arr[2] * v_arr[1]
    # c2 = u3*v1 - u1*v3
```

```python
    c2 = u_arr[2] * v_arr[0] - u_arr[0] * v_arr[2]
    # c3 = u1*v2 - u2*v1
    c3 = u_arr[0] * v_arr[1] - u_arr[1] * v_arr[0]

    return np.array([c1, c2, c3])

# For the dot product, we can use NumPy's np.dot
   function
# If you need to implement it:
# def dotprod(a, b):
#   return np.sum(np.asarray(a) * np.asarray(b))

# Define vectors u and v
u = np.array([1, 2, 3])
v = np.array([4, 5, 6])

print(f"Vector u = {u}")
print(f"Vector v = {v}")
print("-" * 30)

# 1. Compute u x v
u_cross_v = crossprod(u, v)
print(f"u x v = {u_cross_v}")

# 2. Compute v x u
v_cross_u = crossprod(v, u)
print(f"v x u = {v_cross_u}")
print("(Note: v x u should be -(u x v))")
print("-" * 30)

# 3. Compute (u x v) . u
# The dot product of (u x v) with u should be 0, as
   u_cross_v is orthogonal to u.
dot_ucrossv_u = np.dot(u_cross_v, u)
print(f"(u x v) . u = {dot_ucrossv_u}")
print("-" * 30)

# 4. Compute (v x u) . v
# The dot product of (v x u) with v should be 0, as
   v_cross_u is orthogonal to v.
dot_vcrossu_v = np.dot(v_cross_u, v)
print(f"(v x u) . v = {dot_vcrossu_v}")
```

## Saída do Código (Exemplo)

A execução do script Python acima produziu a seguinte saída:

```
Vector u = [1 2 3]
Vector v = [4 5 6]
------------------------------
u x v = [-3  6 -3]
v x u = [ 3 -6  3]
(Note: v x u should be -(u x v))
------------------------------
(u x v) . u = 0
------------------------------
(v x u) . v = 0
```

## Resultados Formatados Matematicamente

Os vetores utilizados foram $u = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ e $v = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}$. Os resultados são:

- $u \times v = \begin{bmatrix} -3 \\ 6 \\ -3 \end{bmatrix}$
- $v \times u = \begin{bmatrix} 3 \\ -6 \\ 3 \end{bmatrix}$
- $(u \times v) \cdot u = 0$
- $(v \times u) \cdot v = 0$

# 3ª Questão

## a) [FORD] Exercício 10.6

Seja $f(x) = \ln(x)$.

**a)** Mostrar que o número de condição de $f$ em $x$ é dado por $c(x) = \frac{1}{|\ln x|}$.

**Resolução:** O número de condição relativo de uma função escalar é definido por:

$$c(x) = \left| \frac{xf'(x)}{f(x)} \right|$$

Para $f(x) = \ln(x)$, temos:

$$f'(x) = \frac{1}{x}$$

Portanto:

$$c(x) = \left| \frac{x \cdot \frac{1}{x}}{\ln(x)} \right| = \left| \frac{1}{\ln(x)} \right| = \frac{1}{|\ln(x)|}$$

**b)** Usando esse resultado, mostrar que $\ln(x)$ é mal condicionado (ill-conditioned) próximo de $x = 1$.

**Resolução:**
Sabemos que:

$$\ln(1) = 0 \quad \Rightarrow \quad |\ln(x)| \to 0 \quad \text{quando } x \to 1$$

Logo:

$$c(x) = \frac{1}{|\ln(x)|} \to \infty \quad \text{quando } x \to 1$$

Isso significa que, para valores de $x$ próximos de 1, o número de condição cresce muito, indicando que **pequenas variações em $x$ provocam grandes variações relativas em $f(x)$**.

Portanto, a função $\ln(x)$ é **mal condicionada perto de $x = 1$**.

# 4ª Questão

## [FORD] Exercício 10.12

Seja a matriz

$$A = \begin{bmatrix} 1 & a \\ a & 1 \end{bmatrix}$$

1. Para quais valores de $a$ a matriz $A$ é **mal condicionada** (ill-conditioned)?
2. O que acontece com $A$ quando $a \to \infty$?

**Resolução:**
**1. Número de condicionamento:**
O número de condicionamento de uma matriz simétrica pode ser estimado por:

$$\kappa(A) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$$

Vamos calcular os autovalores de $A$.

$$\det(A - \lambda I) = \begin{vmatrix} 1-\lambda & a \\ a & 1-\lambda \end{vmatrix} = (1-\lambda)^2 - a^2 = 0$$

$$(1-\lambda)^2 = a^2 \Rightarrow 1 - \lambda = \pm a \Rightarrow \lambda = 1 \pm a$$

Portanto, os autovalores são:

$$\lambda_1 = 1 + a, \quad \lambda_2 = 1 - a$$

**Número de condicionamento:**

$$\kappa(A) = \left| \frac{1+a}{1-a} \right| \quad \text{(ou vice-versa, dependendo de qual for maior)}$$

A matriz é **mal condicionada** quando $\kappa(A)$ é muito maior que 1, ou seja, quando o denominador se aproxima de zero:

$$1 - a \approx 0 \Rightarrow |a| \approx 1 \Rightarrow kappa(A) \to \infty$$

Portanto, $A$ é mal condicionada quando $a$ está próximo de 1.

**2. Quando $a \to \infty$:**

$$\lambda_1 = 1 + a \to \infty, \quad \lambda_2 = 1 - a \to -\infty$$

Tomando os valores absolutos:

$$|\lambda_1| \approx |\lambda_2| \Rightarrow \kappa(A) \to 1$$

Assim, quando $a \to \infty$, a matriz $A$ torna-se **bem condicionada**.

## 5ª Questão

```python
    # Codigo disponivel em:
    # https://github.com/albenyfjaa/alc-albeny

# Funcao que gera a matriz bidiagonal de Wilkinson
def wilkinson_bidiagonal(n):
    A = np.diag(np.arange(n, 0, -1, dtype=float))  # for a tipo float
    A += np.diag([n] * (n - 1), k=1)
    return A

# Parte (a) e (b): calculo do numero de condicao para n = 1 ate 15
n_values = range(1, 16)
cond_numbers = [np.linalg.cond(wilkinson_bidiagonal(n)) for n in n_values]

plt.figure(figsize=(8, 5))
plt.plot(n_values, cond_numbers, marker='o')
plt.title("Numero de condicao da matriz bidiagonal de Wilkinson")
plt.xlabel("Ordem n")
plt.ylabel("Numero de condicao")
plt.grid(True)
plt.show()

# Parte (c): autovalores da matriz original e perturbada
n = 20
A = wilkinson_bidiagonal(n)

# Autovalores da matriz original
eigvals_original = np.linalg.eigvals(A)

# Criar perturbacao aleatoria pequena
np.random.seed(0)  # garante reprodutibilidade
perturbation = 1e-10 * np.random.randn(n, n)
A_perturbed = A + perturbation

# Autovalores da matriz perturbada
eigvals_perturbed = np.linalg.eigvals(A_perturbed)

# Impressao dos autovalores
```

```
print("Autovalores da matriz original:\n", np.sort(np.
    real(eigvals_original)))
print("\nAutovalores da matriz perturbada:\n", np.sort(
    np.real(eigvals_perturbed)))

# Grafico comparando os espectros
plt.figure(figsize=(9, 5))
plt.plot(np.sort(np.real(eigvals_original)), 'o-', label
    ='Original')
plt.plot(np.sort(np.real(eigvals_perturbed)), 'x--',
    label='Perturbada (1e-10)')
plt.title("Autovalores da matriz bidiagonal de Wilkinson
     (n=20)")
plt.xlabel("Indice ordenado")
plt.ylabel("Autovalores (parte real)")
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

## Letra (b) – Análise do número de condição

A Figura 1 mostra o comportamento do número de condição da matriz bidiagonal de Wilkinson para ordens de 1 a 15.

Observa-se que, para pequenas ordens, o número de condição permanece baixo, indicando estabilidade numérica. Entretanto, a partir de, há um crescimento exponencial, revelando que a matriz se torna rapidamente mal condicionada. Isso significa que, em ordens mais altas, pequenas perturbações nos dados podem provocar grandes erros na solução de problemas associados a essa matriz.

## Letra (c) – Sensibilidade dos autovalores

A Figura 2 apresenta os autovalores da matriz bidiagonal de Wilkinson de ordem 20, comparando os valores originais com aqueles obtidos após a adição de uma perturbação aleatória de magnitude $10^{-10}$.

Apesar da perturbação ser extremamente pequena, observa-se uma diferença significativa nos autovalores, reforçando o fato de que a matriz de Wilkinson apresenta comportamento altamente instável do ponto de vista espectral. Esse fenômeno é característico de problemas de autovalores mal condicionados.

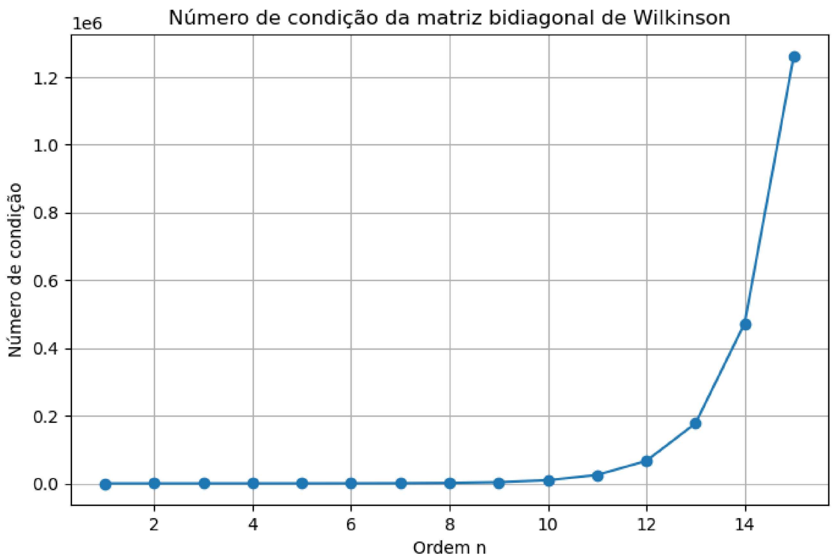


Figura 1: Questão 5 letra b

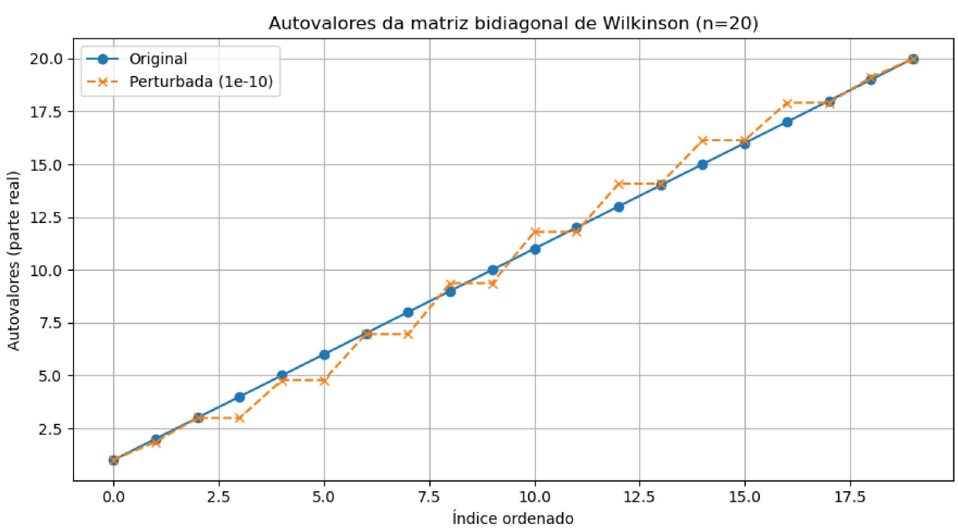


Figura 2: Questão 5 letra c