Revisão e Notas sobre Vetores

Pedro Rupf Pereira Viana

10 de novembro de 2025

1 Introdução e Objetivos

Trata-se de uma revisão teórica e expositiva sobre vetores e suas definições fundamentais no plano cartesiano, e suas aplicações na matemática, física e na programação.

2 Desenvolvimento

Consideremos o plano cartesiano em um sistema de coordenadas em $\mathbb{R}^2$, formado por um par de retas ortogonais. Fixada uma unidade de comprimento, qualquer ponto P do plano pode ser identificado pelo par ordenado $(a, b) \in \mathbb{R}^2$, onde a é a abscissa e b é a ordenada.

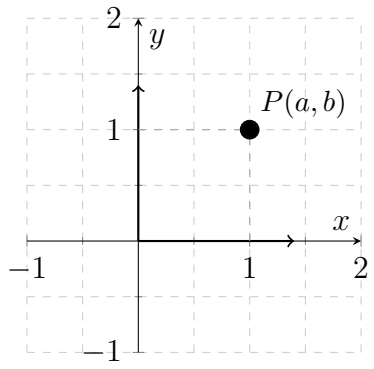

Figura 1: Representação do ponto $P(a, b)$ no primeiro quadrante do plano cartesiano.

Desta forma, dados dois pontos P e Q do plano, adotando $Q(0, 0)$ (isto é, a origem do plano cartesiano), podemos considerar que o segmento de reta $\overrightarrow{QP}$, com ponto inicial Q e ponto final P. Note que embora como conjunto de pontos os segmentos $\overrightarrow{QP}$ e $\overrightarrow{PQ}$ sejam iguais, como segmentos orientados eles são distintos, onde chamamos estes vetores de segmentos opostos.

Passemos a considerar, à partir de agora, apenas segmentos orientados com ponto inicial na origem, denominados *vetores no plano*. É importante notar que vetores no plano são determinados exclusivamente pelo seu ponto final, pois o ponto inicial é fixo na origem. Assim, para cada ponto no plano $P(a, b)$, está associado um único vetor $\mathbf{v} = \overrightarrow{OP}$. Usando esta correspondência entre pontos e vetores, podemos representar o vetor $\mathbf{v} = \overrightarrow{OP}$ pela identificação $\mathbf{v} = (1, 3)$, ou ainda pela notação matriz-coluna $\mathbf{v} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$. Observe que, deste modo, à origem do plano, ficará associado um vetor que têm os pontos inicial e final coincidentes com esta. Denominaremos este vetor (que, na verdade, é apenas um ponto) de *vetor nulo*, sendo representado por $(0, 0)$.

O oposto de um vetor $\mathbf{v} = \overrightarrow{OP}$ é o vetor $\mathbf{w} = \overrightarrow{OQ}$, que possui o mesmo comprimento que $\mathbf{v}$, porém com direção oposta. Em termos de coordenadas, se $\mathbf{v} = (a, b)$, então $\mathbf{w} = (-a, -b)$ e, por essa razão, denota-se que $\mathbf{w} = -\mathbf{v}$.

2.1 Operações com vetores no plano

2.1.1 Multiplicação de um vetor por um número

Multiplicar um vetor $\mathbf{v}$ por um número real $k > 0$ é considerar um novo vetor $k\mathbf{w} = kv$, que possui a mesma direção de $\mathbf{v}$ e têm como comprimento k vezes o comprimento de

$\mathbf{v}$. Se $k < 0$, o vetor $\mathbf{w} = kv$ terá direção oposta à de $\mathbf{v}$ e comprimento $|k|$ vezes o comprimento de $\mathbf{v}$. Se $k = 0$, o vetor resultante $\mathbf{w} = kv$ será o vetor nulo.

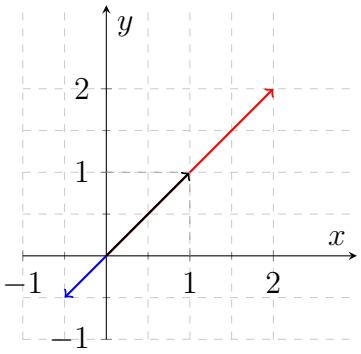

Figura 2: Representação dos vetores $\mathbf{v}$, $\mathbf{w} = 2\mathbf{v}$, e $\mathbf{t} = -0.5\mathbf{v}$ no plano cartesiano.

2.1.2 Adição de dois vetores

Para introduzir a soma de dois vetores, consideremos um exemplo de duas forças atuando sobre um corpo, representadas pelos vetores $\mathbf{F}_1$ e $\mathbf{F}_2$, conforme podemos ver na Figura 3 abaixo:

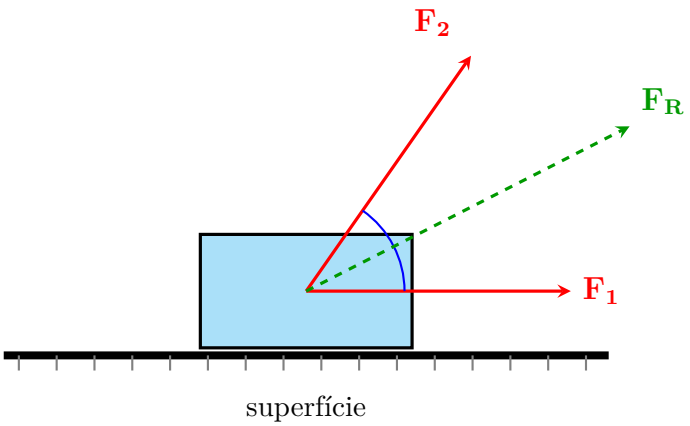

Figura 3: Bloco sobre uma superfície sujeito a duas forças: $\mathbf{F_1}$ paralela ao plano e $\mathbf{F_2}$ formando ângulo $\alpha < 90°$ com $\mathbf{F_1}$. A força resultante $\mathbf{F_R}$ é obtida pela regra do paralelogramo.

Uma força que atua num ponto pode ser representada por um vetor, de comprimento igual à intensidade da força, com a mesma direção e sentido desta força. Supondo agora que as duas forças $\mathbf{F}_1$ e $\mathbf{F}_2$, representadas na FIgura 3, atuem simultaneamente sobre o corpo apresentado nesta mesma figura. Podemos representar o resultado destas duas forças por uma única força $\mathbf{F_R}$?

Ora, podemos representar que a força resultante $\mathbf{F_R}$ é obtida pelo vetor diagonal do paralelogramo construído a partir dos vetores $\mathbf{F_1}$ e $\mathbf{F_2}$, chamando esta operação de soma de vetores, onde escrevemos $\mathbf{F_R} = \mathbf{F_1} + \mathbf{F_2}$. De acordo com a Figura 4 abaixo, temos que $\vec{R} = \vec{v} + \vec{u}$.

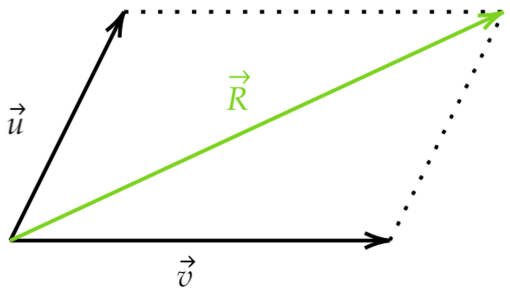

Figura 4: Regra do paralelogramo aplicada à soma de vetores.

2.1.3 Vetores no espaço

Da mesma forma que é definido vetores em um espaço em $\mathbb{R}^2$, podemos definir vetores em um espaço tridimensional, ou seja, em $\mathbb{R}^3$. Neste caso, um ponto P do espaço é identificado por um triplo ordenado (a, b, c), onde temos um sistema de coordenadas formado por três retas ortogonais entre si. Assim, um vetor são dados por um segmento orientado com ponto inicial na origem e ponto final em $P(a, b, c)$, representado por $\mathbf{v} = (a, b, c)$.

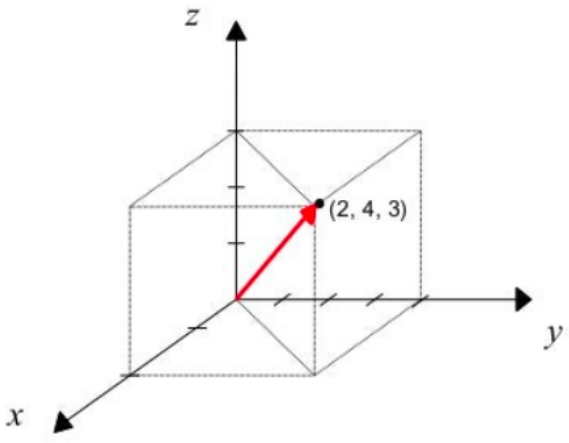

Figura 5: Vetor genérico $\mathbf{v} = (2, 4, 3)$, representado no espaço em $\mathbb{R}^3$.

2.1.4 Operações com vetores no espaço

A soma de dois vetores, e o produto de um vetor por um número real k em $\mathbb{R}^3$ também são da mesma forma que no plano. Isto é, se $\mathbf{u} = (x_1 + x_2 + x_3)$ e $\mathbf{w} = (y_1 + y_2 + y_3)$, então a soma dos vetores é dada por:

$$\mathbf{u} + \mathbf{w} = (x_1 + y_1, x_2 + y_2, x_3 + y_3) \tag{2.1}$$

$$k\mathbf{u} = (kx_1, kx_2, kx_3) \tag{2.2}$$

Por exemplo, se $\mathbf{u} = (2, -3, 5)$ e $\mathbf{v} = (1, 2, 0)$, então $\mathbf{u} + \mathbf{w} = (3, -1, 5)$, e $2\mathbf{u} = (4, -6, 10)$.

Como já observamos no caso do plano, estas operações correspondem exatamente às respectivas operações das matrizes linha que representam os vetores, e gozam de uma série de propriedades decorrentes daquelas relativas às operações com números reais.

2.1.5 Propriedades vetoriais

- $(\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$
- $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
- Existe $\mathbf{0} \in V$ tal que $\mathbf{u} + \mathbf{0} = \mathbf{u}$
- Existe $-\mathbf{u} \in V$ tal que $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$
- $a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$
- $(a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$
- $(ab)\mathbf{v} = a(b\mathbf{v})$
- $1\mathbf{u} = \mathbf{u}$

3 Ok...e onde isso se aplica na programação?

Um **vetor** é uma estrutura de dados que armazena uma coleção **ordenada** de elementos do mesmo tipo (ou tipos compatíveis), acessíveis por meio de um **índice numérico**. Em termos matemáticos, um vetor é uma grandeza com magnitude e direção; na programação, ele representa uma **sequência indexada** de valores.

Em linguagens de baixo nível como C ou C++, vetores são implementados como **arrays** estáticos (tamanho fixo na memória). Em C# e em Python, o conceito de vetor é mais flexível e geralmente representado por **Lists** (ou Arrays, no caso do C#), ou também por Lists e pela biblioteca **NumPy** (que oferece verdadeiros arrays n-dimensionais otimizados).

3.1 Uso de vetores em Python - Lists VS NumPy Array

Características	**List - Python Nativo**	**numpy.array**
Tipagem	heterogênea	Homogênea (mesmo tipo)
Performance	Lenta para op. numéricas	Muito rápida (Usam C/Fortran)
Tamanho fixo?	Dinâmico	Fixo após criação
Operações matemáticas	Manual (loops)	Vetorizadas (element-wise)
Consumo de memória	Alto	Baixo

Tabela 1: Resumo comparativo entre listas nativas do Python e arrays da biblioteca NumPy.

Tal como no C#, em Python podemos usar listas (list) para representar vetores, possuíndo operações básicas como acesso por índice, modificação de elementos, soma e multiplicação por escalar, além de apresentar uma alta e excelente versatilidade de uso (principalmente quando nos vemos em cenários onde um determinado objeto pode receber um número elevado de registros, como por exemplo, um paciente que possui em seu pedido médico 80 exames). Utilizo este exemplo pois é um caso real que já me deparei em meu dia a dia como desenvolvedor, além do fato de que não é possível "prever"quantos exames um paciente pode solicitar em um único pedido médico; e caso usemos um array estático, podemos acabar tendo problemas na montagem do pedido, bem como estouro de memória.

Já a biblioteca NumPy é ideal para cálculos numéricos e manipulação de grandes conjuntos de dados, sendo amplamente utilizada em ciência de dados, aprendizado de

máquina e computação científica. O numpy.array possui tamanho fixo após a criação, o que permite otimizações de desempenho significativas. Suas desvantages incluem a necessidade de instalação adicional e a limitação de tipos homogêneos (isto é, todos os elementos devem ser do mesmo tipo).

3.2 Exemplos Práticos em Python

3.2.1 Lista como vetor

```python
# ==================== EXEMPLO 1: Lista como vetor ====================
vetor_lista = [10, 20, 30, 40, 50]

# Acesso por indice
print(f"Primeiro elemento: {vetor_lista[0]}")
print(f"Ultimo elemento: {vetor_lista[-1]}")

# Modificacao
vetor_lista[2] = 999
print(f"Vetor modificado: {vetor_lista}")

# Soma manual
soma = sum(vetor_lista)
print(f"Soma (funcao sum): {soma}")

# Multiplicacao por escalar (list comprehension)
vetor_dobrado = [x * 2 for x in vetor_lista]
print(f"Vetor dobrado: {vetor_dobrado}")
```

Listing 1: Uso de list como vetor em Python

3.2.2 Vetorização Real com NumPy

```python
# ================ EXEMPLO 2: NumPy - Vetorizacao Real ================
import numpy as np

print("\n=== Usando numpy.ndarray (vetor de verdade) ===")
vetor_np = np.array([10, 20, 30, 40, 50], dtype=np.float64)

# OPERACOES VETORIZADAS (o grande diferencial!)
print(f"Vetor original: {vetor_np}")
print(f"Vetor + 100: {vetor_np + 100}")
print(f"Vetor * 3: {vetor_np * 3}")
print(f"Vetor ao quadrado: {vetor_np ** 2}")
print(f"Soma total: {vetor_np.sum()}")
print(f"Media: {vetor_np.mean():.2f}")
print(f"Desvio padrao: {vetor_np.std():.2f}")
print(f"Maior valor: {vetor_np.max()}, Menor: {vetor_np.min()}")
```

Listing 2: Vetorização Real com NumPy

3.2.3 Vetores bidimensionais (matrizes)

```python
# ================ Vetores bidimensionais (matrizes) ================
print("\n=== Matriz 3x3 com NumPy ===")
matriz = np.array([[1, 2, 3],
                   [4, 5, 6],
                   [7, 8, 9]])

print("Matriz:\n", matriz)

# Transposta
print("Transposta:\n", matriz.T)

# Multiplicacao elemento a elemento
print("Matriz ao quadrado (element-wise):\n", matriz ** 2)

# Produto matricial
print("Matriz x sua transposta:\n", matriz @ matriz.T)
```

Listing 3: Vetores bidimensionais (matrizes)

3.2.4 Performance entre lista e NumPy.Array

```python
# ============ Performance real (milhoes de elementos) ============
import time

tamanho = 10_000_000

# Lista Python
inicio = time.time()
lista = list(range(tamanho))
lista_dobro = [x * 2 for x in lista]
fim = time.time()
print(f"\nLista (10M elementos) - tempo: {fim - inicio:.3f} s")

# NumPy
inicio = time.time()
vetor = np.arange(tamanho)
vetor_dobro = vetor * 2
fim = time.time()
print(f"NumPy (10M elementos) - tempo: {fim - inicio:.3f} s")
# Resultado tipico: NumPy sendo 50 - 200x mais rapido!
```

Listing 4: Performance entre lista e NumPy.Array

3.2.5 Operações vetoriais comuns usando NumPy

```
# ============ Operacoes vetoriais ============
a = np.array([1, 2, 3, 4])
b = np.array([10, 20, 30, 40])

print(a + b)                        # [11 22 33 44]
print(a - b)                        # [-9 -18 -27 -36]
print(a * b)                        # [10 40 90 160]  (produto elemento a
    elemento)
print(a / b)                        # [0.1 0.1 0.1 0.1]
print(a.dot(b))                     # 300 (produto escalar)
print(np.cross([1,0,0], [0,1,0])) # vetor perpendicular (3D)
```

Listing 5: Operações vetoriais comuns usando NumPy

3.3 Aplicações reais de vetores

Área	Uso de vetores
Machine Learning	Vetor de 784 dimensões (i.e. imagem 28 x 28)
Física	posição, velocidade, aceleração, etcs.
Processamento de áudio	vetor 1D contendo milhares de pontos
Finanças	Série temporal de preços (i.e. vetor de retornos diários)
Visão computacional	Imagens (i.e. Tensores)

Tabela 2: Resumo comparativo entre listas nativas do Python e arrays da biblioteca NumPy.

3.4 Boas práticas

```
# 1. Sempre especifique o dtype quando for critico
vetor_preciso = np.array([1.1, 2.2, 3.3], dtype=np.float32)

# 2. Use funcoes universais (ufuncs) - sao rapidas e legiveis
np.sin(vetor) # seno de cada elemento
np.exp(vetor) # exponencial
np.log(vetor) # logaritmo

# 3. Evite loops explicitos em NumPy
# Exemplo ruim:
for i in range(len(v)): v[i] *= 2
# Exemplo bom:
v *= 2
```

Listing 6: Operações vetoriais comuns usando NumPy

3.5 Conclusão

Embora o Python nativo ofereça listas flexíveis que podem atuar como vetores, o verdadeiro poder dos **vetores na programação moderna** vem da vetorização proporcionada pelo **NumPy**. Ela permite:

- Código mais límpo e legível;
- Desempenho "próximo"de linguagens compiladas para operações numéricas;
- Base para bibliotecas como Pandas, Scikit-learn, TensorFlow e PyTorch