Desafio: Calcular o espaço gerado por um conjunto de vetores

Projete um algoritmo Python para calcular o **espaço vetorial gerado** por um conjunto de vetores, retornando uma **base** para esse espaço.

Declaração

Escreva uma função que receba uma coleção de vetores `generatedSubspaceBasis`,

S={v1,v2,...,vn}, onde todos os vetores pertencem ao mesmo espaço vetorial, e retorne um subconjunto de vetores que formam uma **base do subespaço gerado** por SSS.

Esse subconjunto:

- Não precisa conter todos os vetores de SSS
- Deve conter apenas vetores **linearmente independentes**
- Deve gerar o **mesmo subespaço** que o conjunto original

Exemplo

Considere o seguinte conjunto de vetores:

S={[1,2],[2,4],[3,6]}

S={[1,2],[2,4],[3,6]}

Todos os vetores estão no mesmo eixo (são múltiplos uns dos outros), então o subespaço gerado é uma **reta**.

Logo, uma base possível é:

{[1,2]}

{[1,2]}

Entradas e saídas de amostra

A tabela abaixo mostra exemplos de entrada e saída:

Entrada	Saída
S:[[1,2],[2,4],[3,6]]S: [[1,2],[2,4],[3,6]]	[[1,2]][[1,2]]
S:[[1,0],[0,1],[1,1]]S: [[1,0],[0,1],[1,1]]	[[1,0],[0,1]][[1,0],[0,1]]
S:[[1,2,3],[2,4,6],[1,0,0]]S: [[1,2,3],[2,4,6],[1,0,0]]	[[1,2,3],[1,0,0]][[1,2,3],[1,0,0]]

Solução

A ideia aqui é aplicar uma técnica como **eliminação de Gauss** ou **decomposição por rank** para identificar quais vetores são linearmente independentes e, portanto, podem ser usados para gerar o mesmo subespaço.

Ao aplicarmos a **eliminação de Gauss** ou uma fatoração como `np.linalg.matrix_rank`, conseguimos identificar quais linhas são dependentes (ou seja, que se anulam ou viram combinações lineares de outras).

O algoritmo abaixo:

- Cria uma matriz com os vetores
- Usa eliminação para identificar as **linhas que contribuem para o rank**
- Retorna apenas essas linhas, na ordem original

Dessa forma, temos uma **base para o subespaço gerado por S**.

```
import numpy as np

def generatedSubspaceBasis(S: np.array) → np.array:
    """
    Retorna uma base do subespaço vetorial gerado por um conjunto de vetore

    Parâmetro:
    - S: np.array, onde cada linha representa um vetor.

    Retorno:
    - np.array contendo apenas os vetores linearmente independentes que forr
    """

    # Transpomos a matriz para aplicar a eliminação nas colunas (vetores) se n
    # Mas como os vetores são passados como linhas, mantemos assim.
```

```
    # Aplicamos uma decomposição QR (ou eliminação de Gauss) para identific
    # O truque aqui é usar o rank e selecionar linhas "não-nulas" após eliminaç

    # Etapa 1: Obtemos o rank da matriz original
    rank = np.linalg.matrix_rank(S)

    # Etapa 2: Aplicamos eliminação de Gauss para triangularizar e encontrar b
    # A função np.linalg.qr pode nos ajudar a separar os vetores independentes
    # Mas aqui vamos usar SVD para identificar as linhas independentes mais d
    U, s, Vt = np.linalg.svd(S)

    # s contém os autovalores (singular values), e nos diz quais direções têm c
    # Precisamos identificar quais linhas de S estão associadas com essas dire

    # Usamos tolerância para descartar valores muito pequenos (máquina)
    tol = 1e-10
    independentes = []

    # Iteramos sobre as linhas da matriz original
    for i in range(len(S)):
        # Criamos um conjunto com os vetores acumulados
        if len(independentes) == 0:
            independentes.append(S[i])
        else:
            # Testamos se ao adicionar o novo vetor o rank muda
            teste = np.vstack(independentes + [S[i]])
            if np.linalg.matrix_rank(teste) > len(independentes):
                independentes.append(S[i])

    return np.array(independentes)
```

- Testamos **linha por linha** se o vetor atual **aumenta o rank** do conjunto.
- Se sim, significa que ele **não pode ser obtido** por combinação dos anteriores → **vetor independente**.
- Assim, acumulamos apenas os vetores **essenciais para formar a base** do subespaço.