Lista Prática: Métodos Iterativos para Sistemas Lineares e Aplicação de Decomposições Matriciais

1) Crie algoritmos em Python para os métodos de Jacobi e Gauss-Seidel, e resolva os sistemas lineares abaixo:

a.
$$\begin{aligned} 3x_1 - x_2 + x_3 &= 1, \\ 3x_1 + 6x_2 + 2x_3 &= 0, \\ 3x_1 + 3x_2 + 7x_3 &= 4. \end{aligned}$$

b.
$$\begin{aligned} 10x_1 - x_2 &= 9, \\ -x_1 + 10x_2 - 2x_3 &= 7, \\ - 2x_2 + 10x_3 &= 6. \end{aligned}$$

c.
$$\begin{aligned} 10x_1 + 5x_2 &= 6, \\ 5x_1 + 10x_2 - 4x_3 &= 25, \\ - 4x_2 + 8x_3 - x_4 &= -11, \\ - x_3 + 5x_4 &= -11. \end{aligned}$$

d.
$$\begin{aligned} 4x_1 + x_2 - x_3 + x_4 &= -2, \\ x_1 + 4x_2 - x_3 - x_4 &= -1, \\ -x_1 - x_2 + 5x_3 + x_4 &= 0, \\ x_1 - x_2 + x_3 + 3x_4 &= 1. \end{aligned}$$

e.
$$\begin{aligned} 4x_1 + x_2 + x_3 + x_5 &= 6, \\ -x_1 - 3x_2 + x_3 + x_4 &= 6, \\ 2x_1 + x_2 + 5x_3 - x_4 - x_5 &= 6, \\ -x_1 - x_2 - x_3 + 4x_4 &= 6, \\ 2x_2 - x_3 + x_4 + 4x_5 &= 6. \end{aligned}$$

f.
$$\begin{aligned} 4x_1 - x_2 - x_4 &= 0, \\ -x_1 + 4x_2 - x_3 - x_5 &= 5, \\ - x_2 + 4x_3 - x_6 &= 0, \\ -x_1 + 4x_4 - x_5 &= 6, \\ - x_2 - x_4 + 4x_5 - x_6 &= -2, \\ - x_3 - x_5 + 4x_6 &= 6. \end{aligned}$$

Use o vetor chute inicial como $x^{(0)} = 0$, ou seja, um vetor com todas as componentes iguais a zero, e uma tolerância para o erro de 10^{-3}. Para cada sistema linear, crie uma tabela mostrando para os dois métodos o valor da solução do sistema, do erro e o número de iterações utilizadas.

2) Use o código abaixo para gerar 6 matrizes de tamanho ***n,*** com ***n*** à sua escolha (n escolhido deve ser maior que 10).

```
def gerarMatrix(n):
   x = np.linspace(0,1, n)
   x_, y_ = np.meshgrid(x, x)
   A = (2*n * np.random.rand(n,n))/(n**(2.6)*(x_ - y_)**2 + 1)
   return A
```

Gere o vetor independente b, assumindo que todos os sistemas criados terão o vetor solução $x = 1$ (vetor preenchido de 1's) como resposta. Use os métodos de Jacobi e Gauss-Seidel implementados no exercício 1) para resolver os sistemas lineares. Use o vetor chute inicial como $x^{(0)} = 0$, e uma tolerância para o erro de 10^{-3}. Para cada sistema linear, crie uma tabela mostrando para os dois métodos, o erro e o número de iterações utilizadas, ou caso a matriz gerada não for ideal para os métodos (resultado não convergir), verificar a condição desrespeitada pela matriz e registrar na tabela também a falha.

3) Instruções para o exercício:

1. Decompor uma matriz usando decomposição SVD do pacote numpy.linalg (numpy.linalg.svd)
   1.1 Entender como recompor as matrizes decompostas
   1.2 Verificar como podemos reduzir as dimensões das matrizes e mesmo assim obter uma matriz resultante de mesma dimensão original. $A_{\mathrm{m\times n}} = U_{m\times m}\Sigma_{\mathrm{m\times n}}V_{\mathrm{n\times n}}^{\mathrm{T}}$
2. Carregar uma imagem no formato numpy.array usando python (carregar em escala cinza)
3. Aplicar o conceito do item 1. Na imagem carregada.
4. Analisar a relação entre o *n* escolhido, qualidade final (comparação visual e numérica), e a taxa de compressão (valores a serem armazenados no formato original e depois de aplicar o método),
5. Aplicar em uma imagem de baixa resolução (<300p) e em uma de alta resolução (>HD)

Sugestões para o exercício 3):

- A biblioteca **cv2** já instalada no GoogleColab carrega imagens em formato numpy
- O comando sys.getsizeof() da biblioteca sys calcula o tamanho em memória(Ram) de um objeto.