Atividade Avaliativa Bim 1 de Cálculo Numérico

Aluno: Pedro Henrique Souza Cravo
Turma: CC5mb
Código:
https://drive.google.com/file/d/1_ZQlw0-ynd-1Yr780cWU18iU5U4ll1-9/view?usp=sharing

```
def print_matrix(M, decimals=3):
    for row in M:
        print([round(x, decimals) + 0 for x in row])

def zeros_matrix(rows, cols):
    M = []
    while len(M) < rows:
        M.append([])
        while len(M[-1]) < cols:
            M[-1].append(0.0)
    return M

def resolver_sistema(augMat, sistema_num):
    n = len(augMat)
    m = len(augMat[0])
    M = [row[:] for row in augMat]

    print(f"\nSistema {sistema_num}:")
    print("Matriz original:")
    print_matrix(M)

    rows = n
    cols = m - 1
    MC = zeros_matrix(rows, cols)
    for i in range(rows):
        for j in range(cols):
            MC[i][j] = M[i][j]
    print("\nMatriz de coeficientes:")
    print_matrix(MC)

    A = [row[:] for row in MC]
    for fd in range(n):
        if A[fd][fd] == 0:
            for j in range(fd + 1, n):
                if A[j][fd] != 0:
                    A[fd], A[j] = A[j], A[fd]
                    break
            else:
                print(f"\nDeterminante: 0")
                print("Matriz é não singular: False")
                print("\nMatriz após Gauss-Jordan:")
                print("Sistema singular ou inconsistente")
```

```
                return
            for i in range(fd + 1, n):
                scaler = A[i][fd] / A[fd][fd]
                for j in range(n):
                    A[i][j] -= scaler * A[fd][j]
        product = 1.0
        for i in range(n):
            product *= A[i][i]
        print(f"\nDeterminante: {product}")

        print(f"Matriz é não singular: {product != 0}")

        for i in range(n):
            if M[i][i] == 0:
                for j in range(i + 1, n):
                    if M[j][i] != 0:
                        M[i], M[j] = M[j], M[i]
                        break
                else:
                    print("\nMatriz após Gauss-Jordan:")
                    print("Sistema singular ou inconsistente")
                    return
            pivot = M[i][i]
            for k in range(m):
                M[i][k] /= pivot
            for j in range(n):
                if i != j:
                    coef = M[j][i]
                    for k in range(m):
                        M[j][k] -= coef * M[i][k]

        print("\nMatriz após Gauss-Jordan:")
        print_matrix(M)

if __name__ == "__main__":
    matrizes = [
        [[2.0, 3.0, 5.0], [4.0, -1.0, 7.0]],
        [[2.0, 3.0, 1.0, 0.0], [1.0, -2.0, -1.0, 1.0], [1.0, 4.0,
1.0, 2.0]],
        [[2.0, 2.0, -3.0, 4.0], [1.0, 3.0, 1.0, 11.0], [2.0, 5.0,
-4.0, 13.0]],
        [[1.0, 1.0, 1.0, 1.0], [1.0, -1.0, 1.0, 2.0], [2.0, 2.0,
2.0, 5.0]],
        [[1.0, 1.0, 1.0, 1.0], [1.0, -1.0, 1.0, 2.0], [2.0, 0.0,
2.0, 3.0]],
        [[2.0, 1.0, -2.0, 10.0], [3.0, 2.0, 2.0, 1.0], [5.0, 4.0,
3.0, 4.0]]
    ]
```

```
for idx, matrix in enumerate(matrizes):
    resolver_sistema(matrix, idx + 1)
```