Alunas: Júlia Gabriela Gomes da Silva e Lara Ewellen De Carvalho Rocha

Algoritmo de Resolução de Sistemas Lineares pelo Método de Gauss (com Pivoteamento Parcial)

Sumário

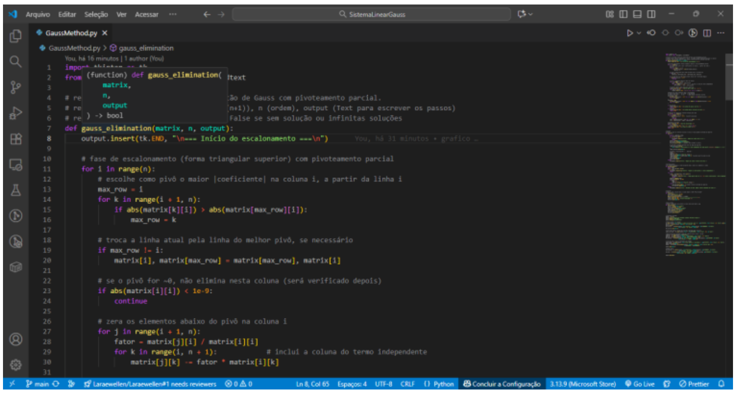
Introdução ao método de Gauss(com Pivoteamento Parcial)

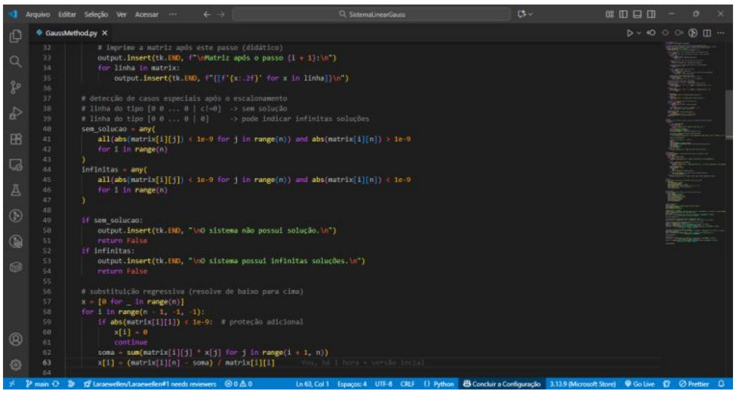
O método da Eliminação de Gauss, também conhecido como escalonamento, é um procedimento utilizado para resolver sistemas lineares de equações. Seu objetivo é transformar o sistema original em outro equivalente, porém mais simples, por meio de operações elementares entre as linhas da matriz aumentada que representa o sistema. O processo consiste em aplicar operações até que todos os elementos abaixo da diagonal principal se tornem nulos, obtendo uma matriz triangular superior. A partir dessa forma escalonada, as incógnitas são determinadas de forma sequencial, por meio da substituição regressiva.

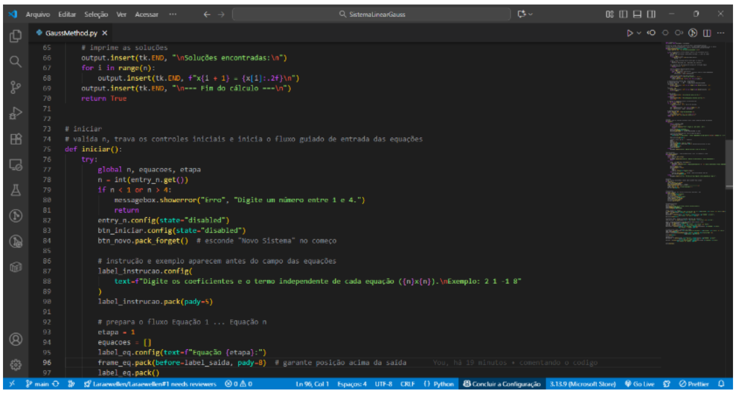
Para aumentar a precisão dos cálculos e evitar divisões por zero, é utilizado o **pivoteamento parcial**, que consiste em escolher, em cada etapa, o maior elemento em valor absoluto da coluna do pivô e trocá-lo com a linha atual. Isso reduz erros numéricos e torna o algoritmo mais estável e confiável. Além disso, o programa foi desenvolvido para detectar automaticamente **sistemas sem solução** (inconsistentes) e **sistemas com infinitas soluções** (dependentes), exibindo mensagens claras ao usuário.

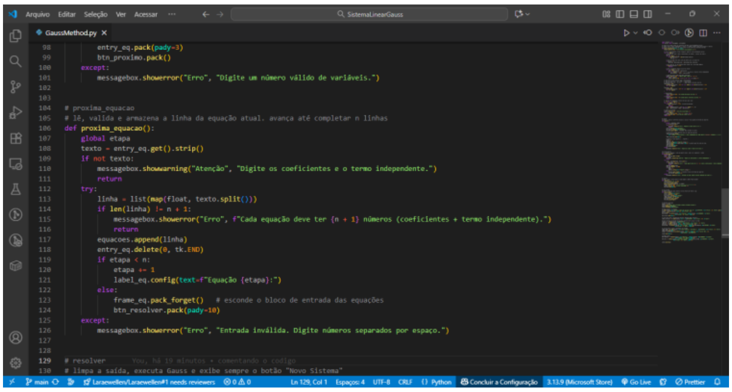
O método de Gauss é amplamente aplicado por sua simplicidade, eficiência e facilidade de implementação computacional, sendo a base para diversos métodos numéricos modernos. Neste trabalho, ele foi escolhido por permitir visualizar claramente o processo de resolução passo a passo e compreender o funcionamento interno de um sistema linear, desde o escalonamento até o resultado final.

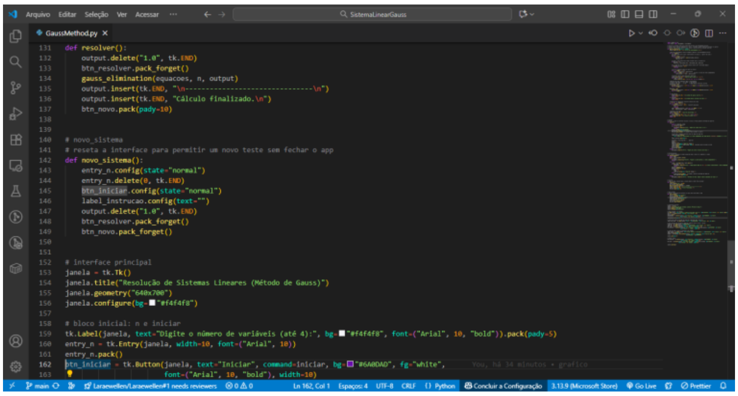
Fotos do código

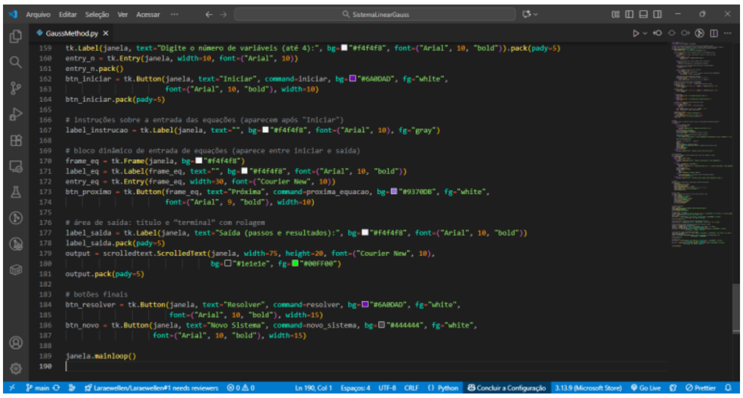












Código para copiar (já comentado)

```
import tkinter as tk
from tkinter import messagebox, scrolledtext

# resolve um sistema linear por eliminação de Gauss com pivoteamento
parcial.
# recebe: matrix (matriz aumentada n x (n+1)), n (ordem), output (Text
para escrever os passos)
# retorna: True se houve solução única; False se sem solução ou infinitas
soluções
def gauss_elimination(matrix, n, output):
```

```
    output.insert(tk.END, "\n=== Início do escalonamento ===\n")

    # fase de escalonamento (forma triangular superior) com pivoteamento
parcial
    for i in range(n):
        # escolhe como pivô o maior |coeficiente| na coluna i, a partir
da linha i
        max_row = i
        for k in range(i + 1, n):
            if abs(matrix[k][i]) > abs(matrix[max_row][i]):
                max_row = k

        # troca a linha atual pela linha do melhor pivô, se necessário
        if max_row != i:
            matrix[i], matrix[max_row] = matrix[max_row], matrix[i]

        # se o pivô for ~0, não elimina nesta coluna (será verificado
depois)
        if abs(matrix[i][i]) < 1e-9:
            continue

        # zera os elementos abaixo do pivô na coluna i
        for j in range(i + 1, n):
            fator = matrix[j][i] / matrix[i][i]
            for k in range(i, n + 1):            # inclui a coluna do
termo independente
                matrix[j][k] -= fator * matrix[i][k]

        # imprime a matriz após este passo (didático)
        output.insert(tk.END, f"\nMatriz após o passo {i + 1}:\n")
        for linha in matrix:
            output.insert(tk.END, f"{[f'{x:.2f}' for x in linha]}\n")

    # detecção de casos especiais após o escalonamento
    # linha do tipo [0 0 ... 0 | c!=0]  -> sem solução
    # linha do tipo [0 0 ... 0 | 0]     -> pode indicar infinitas
soluções
    sem_solucao = any(
        all(abs(matrix[i][j]) < 1e-9 for j in range(n)) and
abs(matrix[i][n]) > 1e-9
        for i in range(n)
    )
    infinitas = any(
        all(abs(matrix[i][j]) < 1e-9 for j in range(n)) and
abs(matrix[i][n]) < 1e-9
        for i in range(n)
    )

    if sem_solucao:
```

```
        output.insert(tk.END, "\nO sistema não possui solução.\n")
        return False
    if infinitas:
        output.insert(tk.END, "\nO sistema possui infinitas soluções.\n")
        return False

    # substituição regressiva (resolve de baixo para cima)
    x = [0 for _ in range(n)]
    for i in range(n - 1, -1, -1):
        if abs(matrix[i][i]) < 1e-9:  # proteção adicional
            x[i] = 0
            continue
        soma = sum(matrix[i][j] * x[j] for j in range(i + 1, n))
        x[i] = (matrix[i][n] - soma) / matrix[i][i]

    # imprime as soluções
    output.insert(tk.END, "\nSoluções encontradas:\n")
    for i in range(n):
        output.insert(tk.END, f"x{i + 1} = {x[i]:.2f}\n")
    output.insert(tk.END, "\n=== Fim do cálculo ===\n")
    return True


# iniciar
# valida n, trava os controles iniciais e inicia o fluxo guiado de
entrada das equações
def iniciar():
    try:
        global n, equacoes, etapa
        n = int(entry_n.get())
        if n < 1 or n > 4:
            messagebox.showerror("Erro", "Digite um número entre 1 e 4.")
            return
        entry_n.config(state="disabled")
        btn_iniciar.config(state="disabled")
        btn_novo.pack_forget()  # esconde "Novo Sistema" no começo

        # instrução e exemplo aparecem antes do campo das equações
        label_instrucao.config(
            text=f"Digite os coeficientes e o termo independente de cada
equação ({n}x{n}).\nExemplo: 2 1 -1 8"
        )
        label_instrucao.pack(pady=5)

        # prepara o fluxo Equação 1 ... Equação n
        etapa = 1
        equacoes = []
        label_eq.config(text=f"Equação {etapa}:")
```

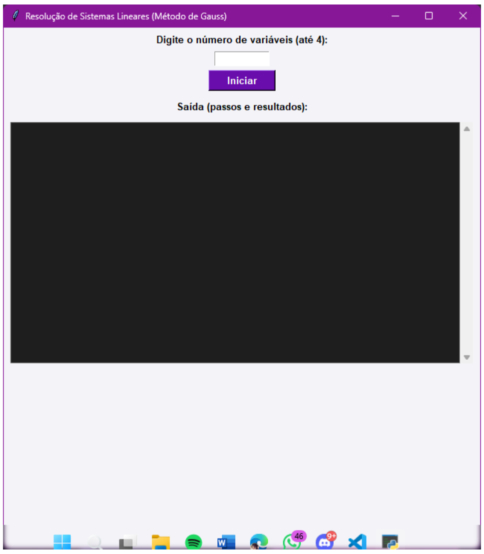
```
        frame_eq.pack(before=label_saida, pady=8)  # garante posição
acima da saída
        label_eq.pack()
        entry_eq.pack(pady=3)
        btn_proximo.pack()
    except:
        messagebox.showerror("Erro", "Digite um número válido de
variáveis.")


# proxima_equacao
# lê, valida e armazena a linha da equação atual. avança até completar n
linhas
def proxima_equacao():
    global etapa
    texto = entry_eq.get().strip()
    if not texto:
        messagebox.showwarning("Atenção", "Digite os coeficientes e o
termo independente.")
        return
    try:
        linha = list(map(float, texto.split()))
        if len(linha) != n + 1:
            messagebox.showerror("Erro", f"Cada equação deve ter {n + 1}
números (coeficientes + termo independente).")
            return
        equacoes.append(linha)
        entry_eq.delete(0, tk.END)
        if etapa < n:
            etapa += 1
            label_eq.config(text=f"Equação {etapa}:")
        else:
            frame_eq.pack_forget()   # esconde o bloco de entrada das
equações
            btn_resolver.pack(pady=10)
    except:
        messagebox.showerror("Erro", "Entrada inválida. Digite números
separados por espaço.")


# resolver
# limpa a saída, executa Gauss e exibe sempre o botão "Novo Sistema"
def resolver():
    output.delete("1.0", tk.END)
    btn_resolver.pack_forget()
    gauss_elimination(equacoes, n, output)
    output.insert(tk.END, "\n-------------------------------\n")
    output.insert(tk.END, "Cálculo finalizado.\n")
    btn_novo.pack(pady=10)
```

```python
# novo_sistema
# reseta a interface para permitir um novo teste sem fechar o app
def novo_sistema():
    entry_n.config(state="normal")
    entry_n.delete(0, tk.END)
    btn_iniciar.config(state="normal")
    label_instrucao.config(text="")
    output.delete("1.0", tk.END)
    btn_resolver.pack_forget()
    btn_novo.pack_forget()


# interface principal
janela = tk.Tk()
janela.title("Resolução de Sistemas Lineares (Método de Gauss)")
janela.geometry("640x700")
janela.configure(bg="#f4f4f8")

# bloco inicial: n e iniciar
tk.Label(janela, text="Digite o número de variáveis (até 4):",
bg="#f4f4f8", font=("Arial", 10, "bold")).pack(pady=5)
entry_n = tk.Entry(janela, width=10, font=("Arial", 10))
entry_n.pack()
btn_iniciar = tk.Button(janela, text="Iniciar", command=iniciar,
bg="#6A0DAD", fg="white",
                        font=("Arial", 10, "bold"), width=10)
btn_iniciar.pack(pady=5)

# instruções sobre a entrada das equações (aparecem após "Iniciar")
label_instrucao = tk.Label(janela, text="", bg="#f4f4f8", font=("Arial",
10), fg="gray")

# bloco dinâmico de entrada de equações (aparece entre iniciar e saída)
frame_eq = tk.Frame(janela, bg="#f4f4f8")
label_eq = tk.Label(frame_eq, text="", bg="#f4f4f8", font=("Arial", 10,
"bold"))
entry_eq = tk.Entry(frame_eq, width=30, font=("Courier New", 10))
btn_proximo = tk.Button(frame_eq, text="Próxima",
command=proxima_equacao, bg="#9370DB", fg="white",
                        font=("Arial", 9, "bold"), width=10)

# área de saída: título e “terminal” com rolagem
label_saida = tk.Label(janela, text="Saída (passos e resultados):",
bg="#f4f4f8", font=("Arial", 10, "bold"))
label_saida.pack(pady=5)
output = scrolledtext.ScrolledText(janela, width=75, height=20,
font=("Courier New", 10),
```

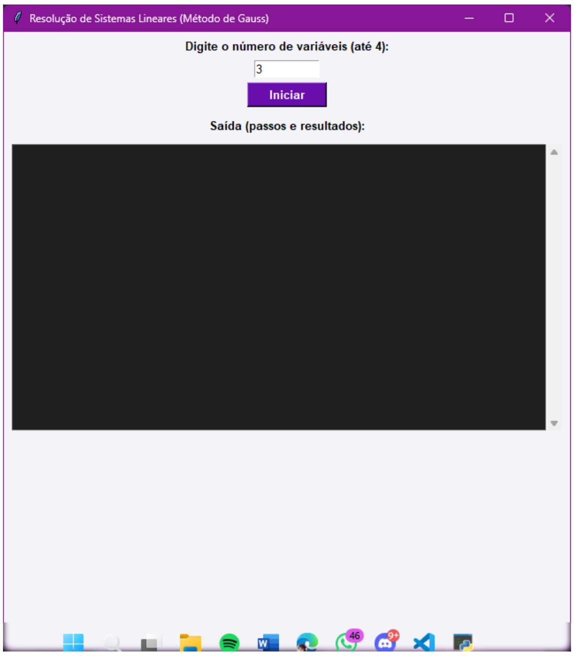
```
                                   bg="#1e1e1e", fg="#00FF00")
output.pack(pady=5)

# botões finais
btn_resolver = tk.Button(janela, text="Resolver", command=resolver,
bg="#6A0DAD", fg="white",
                         font=("Arial", 10, "bold"), width=15)
btn_novo = tk.Button(janela, text="Novo Sistema", command=novo_sistema,
bg="#444444", fg="white",
                     font=("Arial", 10, "bold"), width=15)

janela.mainloop()
```

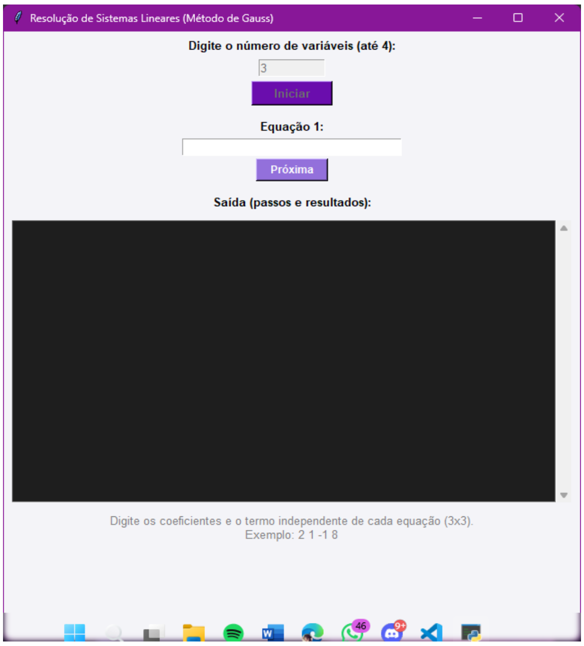
Passo a passo de utilização do programa

1. **Abrir o programa**
   Ao executar o arquivo Python, a interface principal será exibida. A tela possui um fundo claro, um campo de entrada superior e uma área preta na parte inferior onde aparecerão os passos e resultados.

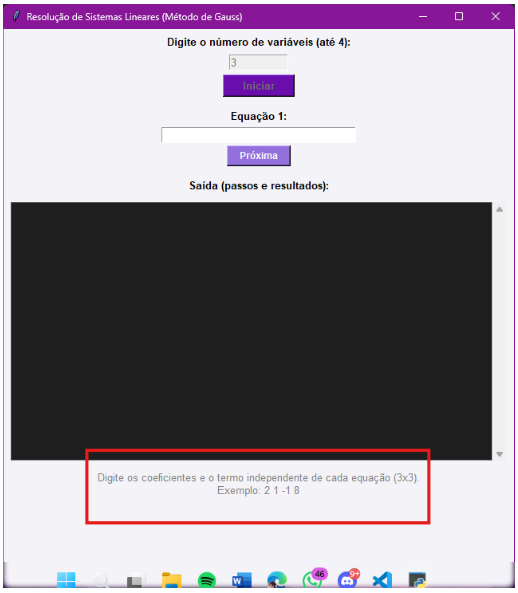


2. **Informar o número de variáveis**
   No campo indicado, o usuário deve digitar quantas incógnitas o sistema possui (por exemplo, 2, 3 ou 4).
   Em seguida, deve clicar no botão **“Iniciar”**.



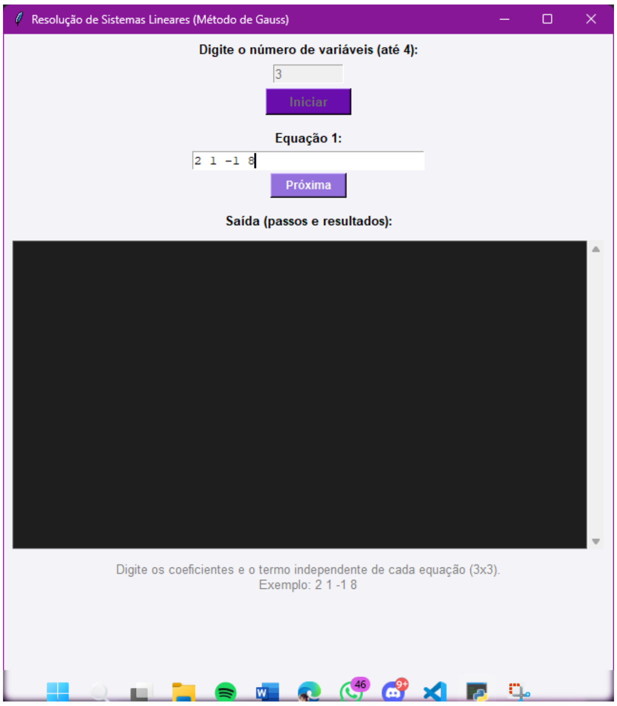


3. **Ler as instruções exibidas**
   O programa mostrará uma mensagem explicativa indicando o que deve ser digitado em cada linha, como:



4. Digite os coeficientes e o termo independente de cada equação (3x3).
   Exemplo: 2 1 -1 8

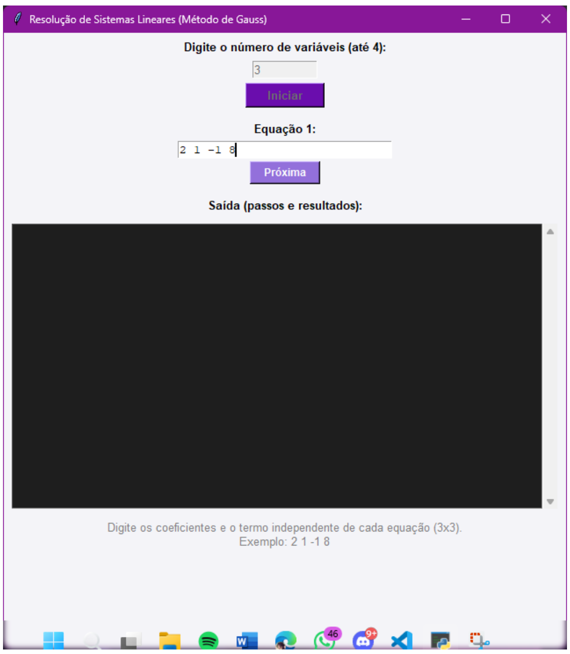
Esse exemplo indica que o usuário deve digitar os números **separados por espaço**, sendo os primeiros os coeficientes das variáveis e o último o termo independente (lado direito da igualdade).



5. **Digitar as equações**
   O campo abaixo das instruções será liberado para digitar as equações.
   - Por exemplo, se o sistema for:
   - 2x + y - z = 8
   - -3x - y + 2z = -11
   - -2x + y + 2z = -3

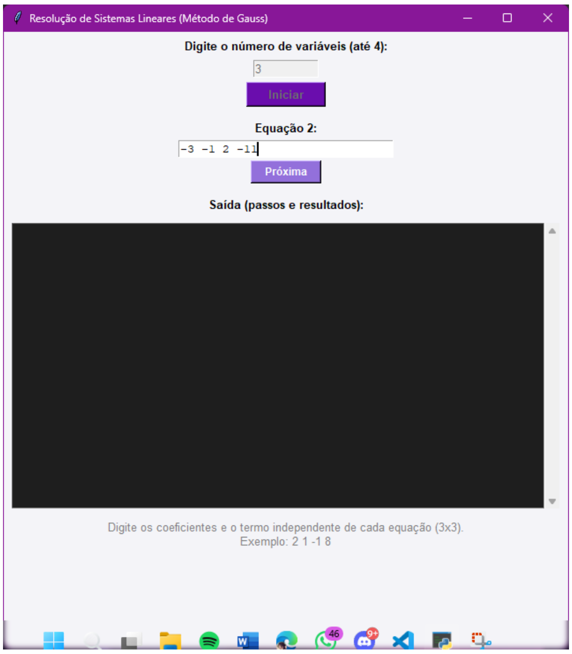
O usuário deve digitar:

2 1 -1 8



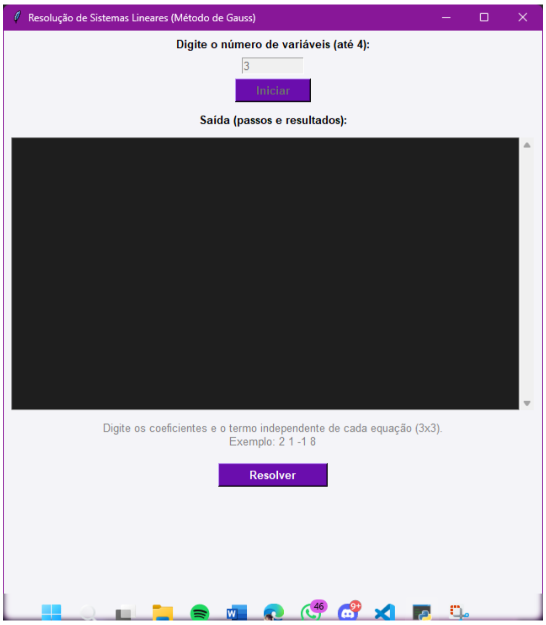
depois clicar em **“Próxima”**,
e digitar:

-3 -1 2 -11

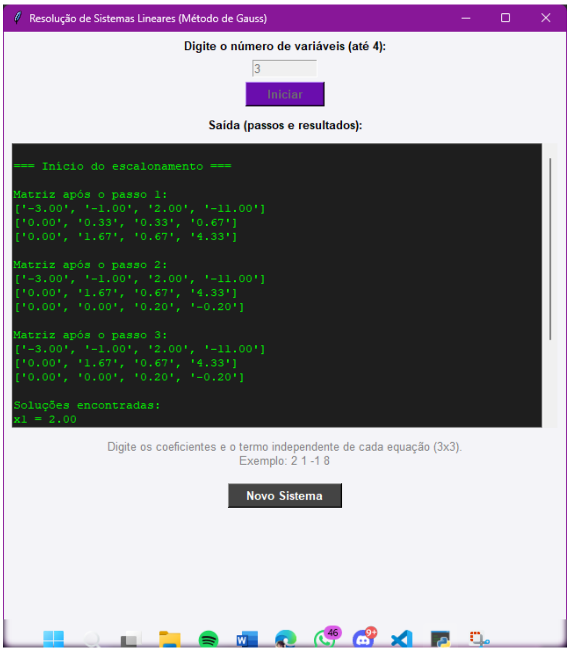


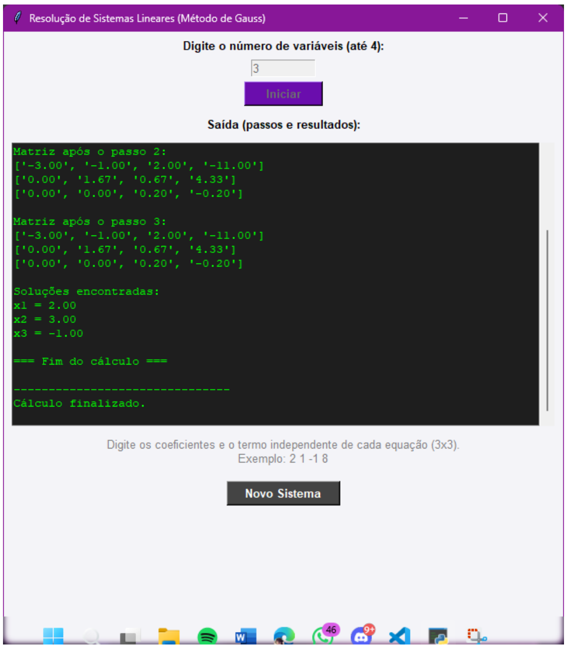
e novamente **“Próxima”**,
e assim por diante até inserir todas as equações.

6. **Executar a resolução**
   Após digitar todas as equações, o botão **“Resolver”** será exibido.
   O usuário deve clicar nele para que o programa realize o escalonamento de Gauss com pivoteamento parcial.

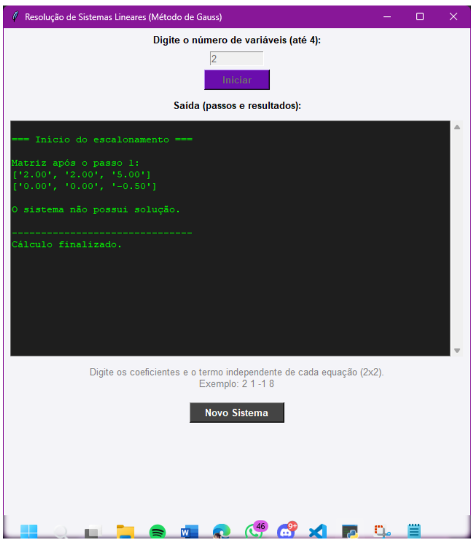


7. **Acompanhar os passos no painel**
   O processo será mostrado na área preta inferior, chamada de “Saída (passos e resultados)”.
   Ali o usuário poderá visualizar:
   - As matrizes escalonadas após cada passo;
   - As mensagens indicando se o sistema possui solução única, infinitas soluções ou nenhuma solução;
   - E, em caso de solução única, os valores de cada variável calculada.

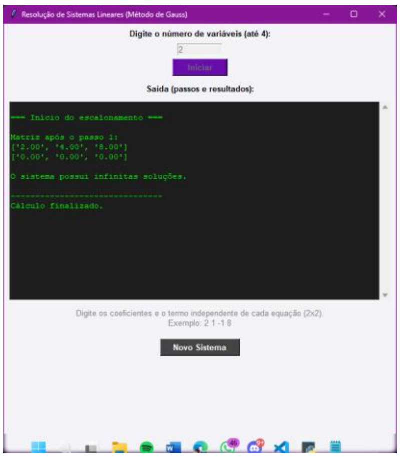




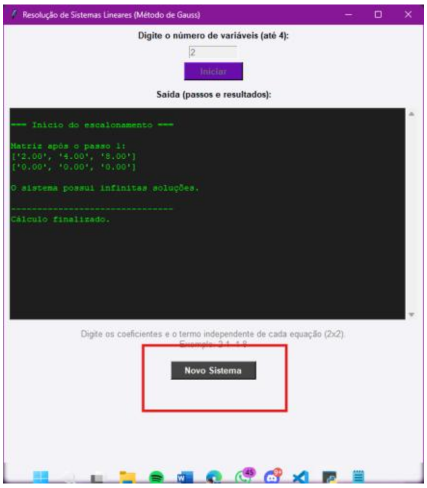
8. **Interpretar os resultados especiais**
   Caso o sistema não possua solução ou possua infinitas soluções, o programa exibirá mensagens específicas:
   - “O sistema não possui solução.”



- “O sistema possui infinitas soluções.”



9. **Iniciar um novo teste**
   Após o cálculo, aparecerá o botão **“Novo Sistema”**.



Ao clicar nele, todos os campos serão limpos e o usuário poderá inserir um novo sistema para resolver sem precisar fechar o programa.

Explicação Detalhada do código por Partes

Visão Geral

O programa desenvolvido resolve sistemas lineares de tamanho 1 a 4 pelo método de Gauss com pivoteamento parcial. A interface gráfica, criada com a biblioteca Tkinter, permite ao usuário inserir as equações, acompanhar o processo de escalonamento e visualizar o resultado final. O código apresenta um diferencial importante: além de aplicar o pivoteamento, ele também reconhece automaticamente quando o sistema não tem solução ou possui infinitas soluções, informando o caso detectado de maneira didática.

Função gauss_elimination

O processo de escalonamento com pivoteamento parcial escolhe em cada coluna o maior valor absoluto como pivô, reduzindo erros numéricos e evitando divisões por zero. Quando necessário, as linhas são trocadas e os elementos abaixo do pivô são eliminados, formando uma matriz triangular superior. Após cada passo de eliminação, a matriz aumentada é exibida no painel de saída para facilitar a visualização do progresso.

Durante o cálculo, o programa identifica casos especiais. Se houver uma linha do tipo [0 … 0 | c ≠ 0], o sistema é inconsistente e, portanto, **não possui solução**. Se surgir uma linha [0 … 0 | 0], o sistema é dependente e, portanto, **possui infinitas soluções**. Nessas situações, a função interrompe o processo de substituição regressiva e exibe a mensagem correspondente.

Quando todos os pivôs são válidos, ocorre a substituição regressiva: as variáveis são calculadas de baixo para cima, isolando cada incógnita a partir da matriz triangular, e os valores finais são exibidos no terminal visual.

Função iniciar

Valida o número de variáveis (entre 1 e 4), desabilita os controles iniciais e apresenta as instruções. Em seguida, exibe os campos para digitação das equações, mostrando um exemplo de formato correto para orientar o usuário.

Função proxima_equacao

Lê a linha digitada, converte os valores em números reais e valida a quantidade correta de elementos (n coeficientes mais um termo independente). Armazena a linha e avança o rótulo para a próxima equação. Quando todas as equações são inseridas, o programa mostra o botão **Resolver**.

Função resolver

Executa a função principal gauss_elimination, mostra cada etapa do escalonamento e as soluções finais. Independentemente do resultado (única, infinita ou inexistente), exibe uma linha informando que o cálculo foi finalizado e libera o botão **Novo Sistema**, permitindo que o usuário inicie um novo teste sem fechar o programa.

Função novo_sistema

Restaura os campos iniciais, limpa a saída e reabilita o botão de início, tornando o processo contínuo e intuitivo.

Diferenciais Implementados

O código possui três diferenciais principais em relação a uma versão tradicional do método de Gauss:

1. **Pivoteamento Parcial**: garante maior precisão e estabilidade nos cálculos, evitando divisões por números pequenos ou nulos.

2. **Verificação Automática de Solução**: o programa detecta e informa se o sistema não possui solução (inconsistente) ou possui infinitas soluções (dependente).

3. **Interface Gráfica Interativa**: o uso do Tkinter transforma o código em uma calculadora visual, exibindo a evolução da matriz após cada passo e permitindo novos testes de forma rápida e intuitiva.

Complexidade e Limites

O escalonamento possui complexidade $O(n^3)$, e a substituição regressiva, $O(n^2)$. Para sistemas de até 4 equações, o desempenho é praticamente instantâneo. O programa assume sistemas quadrados (mesmo número de equações e incógnitas), mas pode ser expandido para casos retangulares com pequenas modificações.

Como o Usuário Opera

O usuário inicia o programa, digita o número de variáveis, clica em **Iniciar**, insere os coeficientes e termos independentes de cada equação, clica em **Resolver** e acompanha o passo a passo no painel de saída. Caso deseje testar outro sistema, basta clicar em **Novo Sistema** e repetir o processo.