ESTUDO DE CASO: MÉTODO DE JACOBI E GAUSS-SEIDEL IMPLEMENTADOS EM LINGUAGEM C E APLICADOS NA ENGENHARIA DE BIOSSISTEMAS

Bruna Carolina Perama Marson[1]
Guilherme Pinto Andrade[1]
Inelia Albino Ribeiro[1]
Lara De Oliveira Lima[1]
Lucas Katsumata Negrão Durço[1]
Luiza Helena Sampaio Moro[1]
Luiz Gustavo Coelho Pedro[1]
Maria Eduarda Muniz De Souza[1]
Rafael Martins Christino[1]
Victor De Oliveira Rodrigues Valentim[1]
Luciane de Fatima Rodrigues de Souza[2]
[1]Discente da área – Instituto Federal de Educação, Ciência e Tecnologia – Campus Avaré
[2]Docente da área – Instituto Federal de Educação, Ciência e Tecnologia – Campus Avaré

Resumo

Este trabalho apresenta o estudo e aplicação de métodos numéricos para resolução de sistemas lineares por meio das técnicas iterativas de Jacobi e Gauss-Seidel, com foco na modelagem de um problema de planejamento semanal de produção industrial. Inicialmente, são abordados conceitos fundamentais de matrizes e sistemas lineares, suas classificações e propriedades, bem como métodos de resolução analítica e numérica. A metodologia baseia-se na implementação dos algoritmos em linguagem C, permitindo a personalização de parâmetros como chutes iniciais, tolerância e número máximo de iterações. A matriz inicial foi reorganizada para garantir dominância diagonal, condição necessária para a convergência do método de Gauss-Seidel. Os resultados demonstraram que esse método convergiu de forma eficiente ao passo que o método de Jacobi apresentou divergência no mesmo sistema. Conclui-se que a dominância diagonal exerce papel determinante na estabilidade dos métodos iterativos, sendo recomendável sua verificação prévia. Como sugestões futuras, propõe-se a integração do programa com ferramentas de análise como planilhas eletrônicas e o estudo de técnicas complementares de aceleração da convergência.

Palavras-chave: Matrizes. Sistemas Lineares. Métodos Iterativos. Gauss-Seidel. Jacobi. Planejamento de Produção.

1. Introdução

Para o cálculo, sistemas lineares são conjuntos de equações que, quando resolvidas ao mesmo tempo, ajudam a encontrar valores para variáveis desconhecidas. Eles aparecem em várias áreas das ciências exatas e da engenharia, onde a modelagem de situações do mundo real exige lidar com um grande número de variáveis. A resolução desses sistemas pode ser feita por métodos diretos ou iterativos, sendo os últimos especialmente úteis para sistemas grandes ou com estruturas que favorecem aproximações sucessivas (Amaral, 2018). Entre os métodos iterativos

mais utilizados estão os de Jacobi e Gauss-Seidel, que são bastante aplicados em contextos computacionais devido à sua simplicidade e eficiência.

O método de Jacobi é um processo iterativo que começa com valores iniciais escolhidos aleatoriamente para as variáveis e gera aproximações sucessiva, repetindo o processo até que a solução convirja na precisão desejada (Ruggiero; Lopes, 1996). A principal característica desse método é que o valor de cada variável na nova iteração depende apenas dos valores da iteração anterior, sem levar em conta as atualizações simultâneas das outras variáveis.

Por outro lado, o método de Gauss-Seidel também utiliza aproximações sucessivas, mas com a diferença de que, à medida que os valores das variáveis são recalculados, eles são imediatamente aplicados nas outras equações da mesma iteração. Isso faz com que, em muitos casos, o Gauss-Seidel convirja mais rapidamente do que o Jacobi, desde que algumas condições sejam atendidas, como a dominância diagonal da matriz dos coeficientes. (Chapra; Canale, 2007; Ruggiero; Lopes, 1996).

A linguagem C, uma das mais antigas e robustas linguagens de programação, é frequentemente utilizada em aplicações científicas e de engenharia por oferecer alto desempenho e controle preciso dos recursos computacionais. Criada por Dennis Ritchie, em 1972, no centro de Pesquisas da Bell Laboratories, tem como objetivo ser uma linguagem de propósito geral, sendo adequada à programação estruturada (UNICAMP, 2011). Seu uso é particularmente vantajoso na implementação de métodos iterativos, já que permite manipular diretamente vetores, matrizes e laços de repetição, elementos essenciais para a resolução de sistemas lineares.

Nesse contexto, o objetivo deste trabalho é analisar como os métodos iterativos de Jacobi e Gauss-Seidel podem ser aplicados, utilizando ferramentas computacionais desenvolvidas na linguagem C, para resolver um sistema linear que foi extraído de um problema apresentado no livro Métodos Numéricos Aplicados com MATLAB® para Engenheiros e Cientistas (Chapra, 2013).

2. Referencial teórico

2.1. Matrizes, sistemas lineares e métodos numéricos

Matrizes são, de forma simplificada, uma tabela de elementos dispostos em linhas e colunas. (Boldrini et al., 1980). Elas são capazes de organizar diversos problemas matemáticos, nos auxiliando a fornecer resoluções alternativas e simplificadas de grandes problemas.

Sejam *m* e *n* números naturais, polinômios e/ou funções, chama-se matriz *m* por *n*, naturalmente escrito como (*m* X *n*), distribuídos em linhas *m*, e colunas *n*.

Ela sempre deve ser representada por uma letra maiúscula.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

A matriz acima é representada por A_{ij}, onde o índice *i* expõe o número de linhas, e o índice *j*, as colunas.

Se a matriz A é de ordem *m* por *n*, costuma-se escrever simplesmente $A_{(m,n)}$. Assim, se uma matriz A tiver 3 linhas e 4 colunas, escreve-se $A_{(3,4)}$ e dizemos que matriz é de ordem 3 por 4.

Após analisar sua composição básica, a forma das matrizes se altera conforme suas diferentes especialidades, como a quantidade de linhas e colunas, ou até mesmos seus elementos. Com

essas variações, surgem diversos tipos de matrizes, cada uma com propriedades e aplicações específicas.

Matriz Quadrada é aquela cujo número de linhas é igual ao número de colunas (m = n).

Matriz Nula é aquela em que $A_{ij} = 0$, para todo *i* e *j*.

Matriz-Coluna é aquela que possui uma única coluna (n = 1).

Matriz-Linha é aquela onde m = 1.

Matriz Diagonal é uma matriz quadrada $_{(m = n)}$ onde A*ij* = 0, para $i \neq j$, isto é, os elementos que não estão na "diagonal" são nulos.

Matriz Identidade Quadrada é aquela em que A*ii* = 1 e A*ij* = 0, para $i \neq j$.

Matriz Triangular Superior é uma matriz quadrada onde todos os elementos abaixo da diagonal principal são nulos, isto é, $m = n$ e A*ij* = 0, para $i > j$.

Matriz Triangular Inferior é aquela em que $m = n$ e A*jj* = 0, para $i < j$.

Matriz Simétrica é aquela onde $m = n$ e A*ij* = A*ji*.

Sistemas lineares são conjuntos de equações lineares com *n* incógnitas. Eles são representados por expressões da forma $a_1x_1+a_2x_2+\cdots+a_nx_n = b$, onde os coeficientes a_i, e o termo independente *b* são números reais.

A resolução desses sistemas pretende encontrar os valores das variáveis que simultaneamente satisfazem todas as equações. Essa busca é estruturada em torno da representação matricial $Ax = b$, em que *A* é a matriz dos coeficientes, *x* o vetor das incógnitas e *b* o vetor dos termos independentes.

Na prática, esses sistemas aparecem com frequência em diversas áreas da engenharia, economia e nas ciências exatas e aplicadas, como em cálculos de circuitos elétricos e análise estrutural. Desse jeito, as variáveis podem ser organizadas em forma de matriz e solucionadas com métodos específicos que garantem agilidade do processo.

Os sistemas lineares têm várias classificações de acordo com o número de soluções que ele pode admitir. Um sistema *S* é possível e determinado quando há apenas uma única solução. Caso existam infinitas soluções, ele vai ser chamado de possível indeterminado. Agora, quando S não admite nenhuma solução, ele é classificado como impossível. (Amaral, 2018).

Existem diversas maneiras de resolver sistemas lineares. A regra de Cramer, por exemplo, funciona para sistemas quadrados. Essa regra usa os determinantes para encontrar soluções, desde que o determinante da matriz dos coeficientes seja não nulo. Outra técnica muito utilizada é o escalonamento, que transforma a matriz aumentada em uma forma triangular, permitindo

resolver o sistema pela substituição regressiva. Esses dois métodos são específicos para sistemas menores.

Para sistemas maiores, geralmente lidamos com métodos iterativos. Esses métodos são extremamente usados em cálculos computacionais por serem executados em uma sequência de instruções, onde cada valor obtido depende do valor anterior. Desse jeito, chegam a uma

solução quase sempre aproximada, e se existir erro, ele vai ter uma margem muito pequena. (Amaral, 2018).

Usar esses métodos é vantajoso por economizar tempo na hora da execução, assim é um ótimo modelo para resolver sistemas de equações lineares de grandes dimensões.

2.2. Os métodos numéricos na engenharia

Os métodos numéricos existem desde a era pré-computacional, porém seu potencial e utilização prática foram limitados nesse período, devido a necessidade de extensas operações e iterações. Por conta disso, os métodos numéricos começaram a ser aplicados em problemas da engenharia amplamente a partir do surgimento e potencialização dos computadores.

Dessa forma, a capacidade de resolver problemas matemáticos é aumentada, visto que os métodos possibilitam resolução de sistemas não lineares e geometrias mais complexas nas quais métodos analíticos não podem ser aplicados.

De acordo com o que foi dito, a utilização dos métodos numéricos começou a ser mais frequente devido a era dos computadores, isso se deve à existência de softwares, que passaram a ser explorados na engenharia, pois aplicam os métodos numéricos na resolução de problemas, geralmente com pacotes e bibliotecas prontas, que funcionam melhor, conforme o conhecimento do usuário sobre os métodos. Porém, não são todos os problemas que são possíveis de resolver utilizando as bibliotecas prontas, sendo necessário programar um novo pacote que seja capaz de resolver o que foi proposto.

Por conta disso, é de suma importância que um engenheiro esteja familiarizado com os métodos numéricos, para que seja capaz de utilizar das ferramentas computacionais de sua forma mais produtiva alcançando resultados desejados com uma melhor compreensão e precisão (Chapra, 2013).

2.3. Método de Gauss-Jacobi

Gauss-Jacobi é um método iterativo que transforma o sistema linear $Ax = b$ em $x = Cx + g$, através um processo de convergência iniciado com uma aproximação $x^{(0)}$ para obter uma sequência de soluções (resolução de um sistema linear) (Ruggiero; Lopes, 1996).

Tomar o sistema inicial:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

tendo $a_{ii} \neq 0, i = 1, \ldots, n$ e isolando o vetor x na separação diagonal:

$$\begin{cases} x_1 = \dfrac{1}{a_{11}}(b_1 - a_{12}x_2 - a_{12}x_3 - \cdots - a_{1n}x_n) \\ x_2 = \dfrac{1}{a_{22}}(b_2 - a_{21}x_1 - a_{23}x_3 - \cdots - a_{2n}x_n) \\ \vdots \\ x_n = \dfrac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots - a_{n,n-1}x_{n-1}) \end{cases}$$

Portanto, tendo $x = Cx + g$, onde:

$$C = \begin{bmatrix} 0 & \dfrac{-a_{12}}{a_{11}} & \dfrac{-a_{13}}{a_{11}} & \cdots & \dfrac{-a_{1n}}{a_{11}} \\ \dfrac{-a_{21}}{a_{22}} & 0 & \dfrac{-a_{23}}{a_{22}} & \cdots & \dfrac{-a_{2n}}{a_{11}} \\ \vdots & \vdots & \vdots & & \vdots \\ \dfrac{-a_{n1}}{a_{nn}} & \dfrac{-a_{n2}}{a_{nn}} & \dfrac{-a_{n3}}{a_{nn}} & \cdots & 0 \end{bmatrix}$$

$$g = \begin{bmatrix} \dfrac{b_1}{a_{11}} \\ \dfrac{b_2}{a_{22}} \\ \vdots \\ \dfrac{b_n}{a_{nn}} \end{bmatrix}$$

O método pretende isolar cada variável da equação correspondente e atualizar seu valor a partir das estimativas anteriores. Dado $x^{(0)}$ como aproximação ideal, obter $x^{(1)} \ldots, x^{(k)} \ldots$ por meio da relação $x^{(k+1)} = Cx^{(k)} + g$:

$$\begin{cases} x_1^{(k+1)} = \dfrac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k)} - a_{12}x_3^{(k)} - \cdots - a_{1n}x_n^{(k)}\right) \\ x_2^{(k+1)} = \dfrac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)}\right) \\ \vdots \\ x_n^{(k+1)} = \dfrac{1}{a_{nn}}\left(b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \cdots - a_{n,n-1}x_{n-1}^{(k)}\right) \end{cases}$$

Na prática, significa que todos os novos valores da solução $x^{(k+1)}$ são calculados simultaneamente, utilizando apenas os valores da iteração anterior $x^{(k)}$. O processo é repetido até que a solução convirja na precisão desejada (Ruggiero; Lopes, 1996).

2.4. Método de Gauss-Seidel

Em Gauss-Seidel, a diferença fundamental em relação ao método de Jacobi está no uso dos valores atualizados das variáveis dentro de uma mesma iteração. Assim, à medida que uma nova estimativa $x_i^{(k+1)}$ é calculada, é utilizada imediatamente nos próximos cálculos da mesma iteração. Assim como Jacobi, o sistema linear $Ax = b$ é equivalente a $x = Cx + g$ por separação da diagonal (Ruggiero; Lopes, 1996).

Limitado um conjunto com 3×3 de equações, se os elementos de diagonal forem todos diferentes de zero, é possível reorganizar a equação de modo que a variável x_1 seja isolada na primeira equação, x_2 na segunda e x_3 na terceira.

$$\begin{cases} x_1 = \dfrac{b_1 - a_{12}x_2 - a_{13}x_3}{a_{11}} \\ x_2 = \dfrac{b_2 - a_{23}x_1 - a_{23}x_3}{a_{22}} \\ x_3 = \dfrac{b_3 - a_{31}x_1 - a_{32}x_2}{a_{33}} \end{cases}$$

Posteriormente, serão escolhidos os valores de aproximação para os $x's$. Um método simples consiste em supor todas as aproximações iguais a zero. Substituindo na primeira equação, é possível obter um novo valor para $x_1 = \frac{b_1}{a_{11}}$. Portanto, substitui-se o novo valor de $x_1$para a aproximação anterior nula de x_3 para calcular um novo valor de x_2. O processo é repetido para calcular uma nova aproximação para x_3. Retorna-se a primeira equação e repete-se o procedimento até que a solução convirja para valores próximos aos desejados (Chapra; Canale, 2007).

Segundo a literatura de Ruggiero e Lopes (1996), o sistema de equações geral é entendido como:

$$\begin{cases} x_1^{(k+1)} = \dfrac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k)} - a_{12}x_3^{(k)} - \cdots - a_{1n}x_n^{(k)}\right) \\ x_2^{(k+1)} = \dfrac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - \cdots - a_{2n}x_n^{(k)}\right) \\ x_3^{(k+1)} = \dfrac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - a_{34}x^{(k)} \ldots - a_{3n}x^{(k)}\right) \\ \vdots \\ x_n^{(k+1)} = \dfrac{1}{a_{nn}}\left(b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - \cdots - a_{n,n-1}x_{n-1}^{(k+1)}\right) \end{cases}$$

Portanto, no momento de se calcular $x_j^{(k+1)}$ usamos todos os valores $x_j^{(k+1)}, \ldots, x_{j-1}^{(k+1)}$ que já foram calculados e os valores $x_{j+1}^{(k)} \ldots, x_n^{(k)}$ restantes.

2.5. Linguagem C

A linguagem C é uma das mais tradicionais e eficientes ferramentas para o desenvolvimento de algoritmos numéricos. Criada por Dennis Ritchie, em 1972, no centro de Pesquisas da Bell Laboratories, tem como objetivo ser uma linguagem de propósito geral, sendo adequada à programação estruturada. No entanto é mais utilizada escrever compiladores, analisadores léxicos, bancos de dados, editores de texto, etc. (UNICAMP, 2011).

A característica principal de uma linguagem estruturada é a compartimentalização do código e dos dados, sendo a capacidade de seccionar as informações necessárias para realizar uma tarefa específica. Portanto, é possível que programas em C compartilhem facilmente seções de um código, apenas descrevendo o que a função faz. Ademais, uma linguagem estruturada permite diversas possibilidades de programação.

O principal componente estrutural da C é a função, que admite a definição e codificação separada de diferentes tarefas de um programa (Schildt, 1996). Seu uso é particularmente vantajoso na implementação de métodos iterativos como Jacobi e Gauss-Seidel, já que permite manipular diretamente vetores, matrizes e laços de repetição, elementos essenciais para a resolução de sistemas lineares.

3. Metodologia

Este trabalho foi desenvolvido com objetivo de aplicar e comparar os métodos iterativos de resolução de sistemas lineares, sendo ele Gauss-Jacobi e Gauss-Seidel, e posteriormente sua

aplicação em problemas matemáticos. Foi utilizado a linguagem de programação C++ e foi desenvolvido na plataforma Dev-C++.

A princípio, foram estudadas as formulações matemáticas dos métodos de Gauss Jacobi e Gauss Seidel, com ênfase em sua aplicabilidade para matrizes diagonais dominantes em problemas. Em seguida, foi elaborado um algoritmo em C++ para cada método, utilizando a estrutura de vetores e laços de repetição.

Ambos os programas foram escritos de forma modular, com funções específicas para leitura de dados, cálculo das iterações, exibição dos resultados e comparação de eficiência.

A entrada de dados é feita manualmente pelo usuário, via console, permitindo a inserção de valores em uma matriz e vetores. Os critérios de parada considerados foram a tolerância do erro (definida previamente no console) e o número máximo de iterações, utilizando-se para ter uma base de eficiência do código.

O Desempenho dos métodos foi comparado quanto à sua eficiência computacional, como número máximo e mínimo de iterações, destacando os contextos nos quais cada abordagem apresenta melhores resultados. E por fim, foi realizado a aplicação de ambos os códigos na resolução do problema de reposição de estoque.

3.1. Apresentação do problema

O problema foi retirado do livro Cálculo Numérico: *Aprendizagem com apoio de software*, de Arenales e Darezzo (2018).

Uma indústria produz quatro tipos de produtos (1), (2), (3) e (4), os quais são processados e produzidos no decorrer da semana. Para a produção de cada unidade desses produtos necessitam-se de quatro diferentes tipos de matérias-primas (A), (B), (C) e (D) conforme a tabela 1:

Tabela 1. Tipos de produtos e matérias-primas.

	A	B	C	D
(1)	1	2	4	1
(2)	2	0	1	0
(3)	4	2	3	1
(4)	3	1	2	1

Fonte: *Arenales; Darezzo, 2018*.

Por exemplo, para produzir uma unidade do produto (1) necessita-se de 1 unidade de (A), 2 unidades de (B), 4 unidades de (C) e 1 unidade de (D).

A indústria possui um planejamento de reposição de matérias-primas que varia semanalmente conforme a tabela 2 que mostra a disponibilidade dos estoques:

Tabela 2. Disponibilidade de estoques.

Estoque	A	B	C	D
1ª semana	16	13	27	7
2ª semana	18	12	25	7
3ª semana	20	20	40	11
4ª semana	27	9	23	5

Fonte: *Arenales; Darezzo, 2018.*

Formule um modelo matemático para determinar a manufatura dos produtos, de modo que os estoques de matérias-primas sejam esgotados semanalmente. Resolva o problema dado utilizando o Método de Decomposição LU no Software Numérico (Arenales, Darezzo, 2018).

3.2. Desenvolvimento do modelo matemático:

Sejam $xi, i = 1, 2, 3, 4$ as quantidades dos produtos (1), (2), (3), (4) a serem fabricados semanalmente pela indústria.

A fabricação dos produtos na 1ª semana do processo produtivo satisfaz o seguinte sistema de equações lineares:

$$\begin{cases} x_1 + 2x_2 + 4x_3 + 3x_4 \\ 2x_1 + 0x_2 + 2x_3 + x_4 \\ 4x_1 + x_2 + 3x_3 + 2x_4 \\ x_1 + 0x_2 + x_3 + x_4 \end{cases}$$

Utilizando o Software Numérico, resolvemos o sistema obtido na 1ª semana, selecionando o Módulo Sistemas Lineares, com a opção de Método de Decomposição LU, seguidamente das condições de aplicabilidade do método (Menores principais ≠ 0 e det(A) ≠ 0).

Assim, temos a solução do sistema $\overline{x} = (5, 2, 1, 1)^t$, ou seja, devemos produzir na 1ª semana 5 produtos do tipo (1), 2 produtos do tipo (2), 1 produto do tipo (3) e 1 produto do tipo (4).

Observe que nas demais semanas, a matriz *A* do sistema obtido permanece inalterada, enquanto que o vetor *b* se modifica pelas quantidades em estoque conforme a disponibilidade nas respectivas semanas, ou seja, devemos tomar nas próximas 3 semanas os vetores $b = (18,12,25,7)^t$, $b = (20,20,40,11)^t$, $b = (27,9,23,5)^t$.

Desta forma, para resolver o problema temos 4 sistemas de equações lineares para serem resolvidos. Esta é uma situação problema, em que devemos escolher um método de decomposição, por exemplo, o método de decomposição LU. A matriz *A* é decomposta uma única vez na 1ª semana e deve ser armazenada. Nas demais semanas, trocamos apenas os vetores *b*, resolvemos os sistemas triangulares obtidos e determinamos o vetor-solução dos produtos nas respectivas semanas.

Seguindo este procedimento obtemos os vetores-solução para os produtos na 2ª semana $\overline{x} = (4,2,1,2)^t$, na 3ª semana $\overline{x} = (8,1,1,2)^t$ e na 4ª semana $\overline{x} = (2,7,2,1)^t$ (Arenales, Darezzo, 2018).

4. Resultados e Discussões

4.2. Implementação da Linguagem C:

Neste trabalho, foi implementado em C dois métodos iterativos Jacobi e Gauss-Seidel para resolver o sistema linear que modela a reposição semanal de matérias-primas em uma indústria. A partir das disponibilidades de cada matéria-prima (A, B, C e D) e das quantidades exigidas por produto (1, 2, 3 e 4), foi montado um sistema de quatro equações e aplicamos ambos os métodos até atingir uma tolerância E = 0,0001.

Inicialmente o sistema não possuía dominância diagonal, o que impedia a convergência de Gauss-Seidel. Após reordenar as equações para garantir dominância, o método convergiu em 27 iterações para a solução aproximada:

x1 = 5
x2 = 2
x3 = 1
x4 = 1

Esses resultados significam produzir, semanalmente, 5 unidades do produto 1, 2 unidades do produto 2, 1 unidade do produto 3 e 1 unidade do produto 4.

A importância da dominância diagonal se deve ao fato de que ela é fundamental para garantir a convergência dos métodos iterativos, especialmente do método de Gauss-Seidel. Sem essa condição, o método não converge para a solução correta, como foi observado nos testes iniciais.

Com a flexibilidade no código o programa permite ao usuário escolher:

- Os valores iniciais de chute para cada variável (em nosso teste, utilizamos 1, 1, 1 e 1).
- O número máximo de iterações a serem executadas.
- A tolerância E para a parada automática: o cálculo é interrompido quando a diferença entre os valores das iterações sucessivas fica menor que E, mesmo que o número máximo de iterações ainda não tenha sido atingido.

4.2.1. Código utilizado para o método de Gauss-Seidel:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define EPSILON 0.0001f

//dominância diagonal
int dominancia(float** a, int N) {
    int i, j;
    for (i = 0; i < N; i++) {
        float soma = 0;
        for (j = 0; j < N; j++) {
            if (i != j) soma += fabsf(a[i][j]);

        }
        if (fabsf(a[i][i]) < soma) return 0; // não dominante
    }
    return 1; // dominante
}

//reorganizar para dominância diagonal(swap de linhas)
void reorganizar(float** a, int N) {
    int i, k;
    for (i = 0; i < N; i++) {
        int melhor = i;
        float max_val = fabsf(a[i][i]);
        for (k = i + 1; k < N; k++) {
            if (fabsf(a[k][i]) > max_val) {
                melhor = k;
                max_val = fabsf(a[k][i]);
            }
        }
        if (melhor != i) {
            float* temp = a[i];
            a[i] = a[melhor];
            a[melhor] = temp;
        }
    }
}
```

```
int main() {
    int N, i, j, iter, max_iter;
    float sum, error;

    // Escolher tamanho da matriz
    printf("Digite o numero de variaveis (N): ");
    scanf("%d", &N);

    //Alocação dinâmica da matriz aumentada A[N][N+1]
    float** a = (float**)malloc(N * sizeof(float*));
    for (i = 0; i < N; i++) {
        a[i] = (float*)malloc((N + 1) * sizeof(float));
    }

    //Vetores de solução e chute inicial
    float* x = (float*)malloc(N * sizeof(float));      // vetor solução
atual
    float* old_x = (float*)malloc(N * sizeof(float));  // vetor solução
anterior

    //Leitura dos coeficientes
    printf("\nDigite os coeficientes da matriz aumentada (Ax = b):\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j <= N; j++) {
            printf("a[%d][%d]: ", i+1, j+1);
            scanf("%f", &a[i][j]);
        }
    }

    // Verificação e tentativa de reorganizar
    if (!dominancia(a, N)) {

        printf("\nMatriz nao e diagonalmente dominante. Tentando
reorganizar...\n");
        reorganizar(a, N);

        if (dominancia(a, N)) {
            printf("Reorganizacao bem-sucedida. Matriz agora e
diagonalmente dominante.\n");
        } else {
            printf("Nao foi possível tornar a matriz dominante.
Resultados podem divergir.\n");
        }
    }

    //Leitura do chute inicial
    printf("\nDigite o chute inicial para cada variável:\n");
    for (i = 0; i < N; i++) {
        printf("x[%d]_0 = ", i+1);
        scanf("%f", &x[i]);
    }

    //Numero de iteracoes desejado
    printf("\nDigite o numero de iteracoes desejado: ");
    scanf("%d", &max_iter);
```

```
    //Metodo de Gauss-Seidel
    for (iter = 0; iter < max_iter; iter++) {
        for (i = 0; i < N; i++) {
            old_x[i] = x[i]; //valor anterior
        }

        // atualiza iterativamente cada xi
        for (i = 0; i < N; i++) {
            if (a[i][i] == 0.0f) {
                printf("Erro: divisao por zero na linha %d.
Abortando.\n", i+1);
                return 1;
            }
            sum = a[i][N];
            for (j = 0; j < N; j++) {
                if (j != i) {
                    sum -= a[i][j] * x[j];
                }
            }
            x[i] = sum / a[i][i];
        }

        //vai mostrando valores da iteração
        printf("\nIteracao %d:\n", iter+1);
        for (i = 0; i < N; i++) {
            printf("x[%d] = %.6f\n", i+1, x[i]);
        }

        // verificação da convergencia
        error = 0.0f;
        for (i = 0; i < N; i++) {
            error += fabsf(x[i] - old_x[i]);

        }
        if (error < EPSILON) {
            printf("Convergencia alcancada (erro < %.6f).
Parando...\n", EPSILON);
            break;
        }
    }

    //Resultado final
    printf("\nSolucao final apos %d iteracoes:\n", iter+1);
    for (i = 0; i < N; i++) {
        printf("x[%d] = %.6f\n", i+1, x[i]);
    }

    //Liberacao de memoria
    for (i = 0; i < N; i++) free(a[i]);
    free(a);
    free(x);
    free(old_x);

    //esperar para sair
    printf("\nPressione Enter para sair...");
    getchar();
    getchar();
```

```
    return 0;
}
```

4.2.2. Código utilizado para o método de Jacobi:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define EPSILON 0.0001

int main() {
    int N, i, j, iter, max_iter;
    float sum, error;

    // Poder escolher o tamanho da matriz 2 = 2x2, 3 = 3x3 etc...
    printf("Digite o numero de variaveis (N): ");
    if (scanf("%d", &N) != 1 || N <= 0) {
        printf("Valor de N invalido.\n");
        return 1;
    }

    //Alocacao dinamica da matriz aumentada A[N][N+1]
    float **a = (float**) malloc(N * sizeof(float*));
    for (i = 0; i < N; i++) {
        a[i] = (float*) malloc((N + 1) * sizeof(float));
    }

    //Vetores de chute inicial *calloc -> malloc
    float *x      = (float*) malloc(N * sizeof(float));    // solucao
atual
    float *x_old = (float*) malloc(N * sizeof(float));    // iteracao
anterior

    float *x_new  = (float*) malloc(N * sizeof(float));    // proximo
valor

    //Leitura dos coeficientes da matriz aumentada
    printf("\nDigite os coeficientes da matriz aumentada (Ax = b):\n");
    for (i = 0; i < N; i++) {
        for (j = 0; j <= N; j++) {
            printf("a[%d][%d]: ", i+1, j+1);
            scanf("%f", &a[i][j]);
        }
    }

    //Leitura do chute inicial
    printf("\nDigite o chute inicial para cada variavel:\n");
    for (i = 0; i < N; i++) {
        printf("x[%d]_0 = ", i+1);
        scanf("%f", &x[i]);
    }

    //Numero de iteracoes desejado
    printf("\nDigite o numero de iteracoes desejado: ");
    scanf("%d", &max_iter);
```

```
    //Metodo de Jacobi
    printf("\nIniciando o metodo de Jacobi...\n");
    for (iter = 0; iter < max_iter; iter++) {
        // copia solucao atual para anterior
        for (i = 0; i < N; i++) {
            x_old[i] = x[i];
        }

        // calcula novos valores em x_novo usando somente x_prev
        for (i = 0; i < N; i++) {
            sum = a[i][N];
            for (j = 0; j < N; j++) {
                if (j != i) {
                    sum -= a[i][j] * x_old[j];
                }
            }
            x_new[i] = sum / a[i][i];
        }
        // copia x novo para x
        for (i = 0; i < N; i++) {
            x[i] = x_new[i];
        }

        // exibe valores da iteracao
        printf("\nIteracao %d:\n", iter+1);
        for (i = 0; i < N; i++) {
            printf("x[%d] = %.6f\n", i+1, x[i]);
        }

        // verifica convergencia
        error = 0.0f;
        for (i = 0; i < N; i++) {
            error += fabsf(x[i] - x_old[i]);

        }
        if (error < EPSILON) {
            printf("Convergencia alcancada (erro < %.6f).
Parando...\n", EPSILON);
            break;
        }
    }

    //Resultado
    printf("\nSolucao final apos %d iteracao(oes):\n", iter+1);
    for (i = 0; i < N; i++) {
        printf("x[%d] = %.6f\n", i+1, x[i]);
    }

    //Libera de memoria (desnecessario mas né eficiencia)
    for (i = 0; i < N; i++) free(a[i]);
    free(a);
    free(x);
    free(x_old);
    free(x_new);

    //Espera para sair
    printf("\nPressione Enter para sair...");
```

```
    getchar(); getchar();
    return 0;
}
```

4.3. Comparação entre métodos:

O método de Gauss-Seidel convergiu em 27 iterações após criar uma dominância diagonal, demonstrando consistência e eficiência para este problema.

Jacobi não convergiu. Os valores cresceram sem estabilizar. Isso evidencia que, em sistemas que não são naturalmente dominantes, Jacobi pode falhar enquanto Gauss-Seidel, com atualizações imediatas dos valores, alcança estabilidade.

Em resumo, o método de Gauss-Seidel mostrou-se adequado e seguro para resolver nosso sistema de planejamento semanal, enquanto o método de Jacobi requer cautela quanto às propriedades da matriz associada. A capacidade de definir chutes iniciais e limites de iteração torna a ferramenta flexível para diferentes configurações de uso.

5. Conclusão

Tendo em vista o propósito inicial deste trabalho, os resultados mostraram que o método de Jacobi não conseguiu encontrar a solução correta na ausência de dominância diagonal. Dessa forma, não houve convergência dos valores, que continuaram crescendo sem estabilizar. Por outro lado, o método de Gauss-Seidel convergiu com eficiência após o reordenamento das equações e a implementação da dominância diagonal, obtendo resultado satisfatório em 27 iterações. Isso evidencia como essa propriedade é essencial para garantir soluções estáveis.

A variedade de códigos utilizados e o desenvolvimento de funcionalidades que permitem a alteração dos chutes iniciais, da tolerância e do número máximo de iterações tornaram-se ferramentas úteis e fáceis de aplicar em diferentes cenários.

Observa-se também que a comparação com a abordagem baseada na decomposição LU, conforme demonstrado no problema, reforça a relevância de se utilizar diferentes métodos numéricos, dependendo da natureza dos problemas e dos recursos disponíveis.

Portanto, conclui-se que o método de Gauss-Seidel foi mais eficiente por convergir após a criação de uma dominância diagonal. Sua flexibilidade para diferentes configurações e sua segurança na obtenção dos resultados demonstram que ele é uma escolha adequada para a resolução de sistemas lineares.

Referências Bibliográficas

AMARAL, W. M. *Comparativo entre o método de Gauss-Seidel e os métodos de sub e sobre relaxação sucessiva*. 2018. Trabalho de Conclusão de Curso (Bacharelado em Ciência e Tecnologia) – Universidade Federal Rural do Semiárido, Mossoró, 2018. Disponível em:

https://repositorio.ufersa.edu.br/server/api/core/bitstreams/bb59bf82-d8a0-4b98-a028-ef664870f9d3/content. Acesso em: 10 jun. 2025.

ARENALES, S.; DAREZZO, A. Cálculo Numérico: aprendizagem com apoio de software. 2. ed. São Paulo: Cengage Learning, 2018.

BOLDRINI, J. L. *et al*. *Álgebra linear*. 3. ed. São Paulo: Harper & Row do Brasil, 1980.

CHAPRA, S. C. *Métodos Numéricos Aplicados com MATLAB® para Engenheiros e Cientistas*. 3. ed. [S. l.]: AMGH, 2013. 655 p.

CHAPRA, S. C.; CANALE, R. P. *Métodos Numéricos Para Engenharia*. [*S. l.*]: Mcgraw-Hill, 2007.

RUGGIERO, M. G. A.; LOPES, V. L. R. *Cálculo Numérico*: *Aspectos teóricos e computacionais*. 2. ed. São Paulo: Pearson, 1996.

SILVA, J. V. S. *Aplicações de matrizes no ensino médio*. 2014. 48 f. Trabalho de Conclusão de Curso (Graduação em Matemática) – Departamento de Matemática, Centro de Ciências e Tecnologia, Universidade Estadual da Paraíba, Campina Grande, 2014. Disponível em: https://dspace.bc.uepb.edu.br/jspui/bitstream/123456789/9493/1/PDF%20-%20Jos%C3%A9%20Valber%20Silvino%20da%20Silva.pdf. Acesso em: 10 jun. 2025.

SCHILDT, H. *C, Completo e Total*. 3. ed. São Paulo: Makron Books, 1996.

UNIVERSIDADE ESTADUAL DE CAMPINAS. *Introdução à linguagem C*. Instituto de Computação. Disponível em: https://www.inf.ufrgs.br/~binsely/tutorialc.pdf. Acesso em: 22 jun. 2025.