

Pedro Pacheco Mendes Filho

Trabalho de Modelagem Computacional

Santarém, Pará

2018

Pedro Pacheco Mendes Filho

Trabalho de Modelagem Computacional

Esse trabalho foi criado como avaliador na obtenção de nota para a disciplina de Modelagem Computacional.

Universidade Federal do Oeste do Pará

Instituto de Engenharia e Geociência

Programa de Ciência e Tecnologia

Professora: Dra. Marciana Lima Góes

Santarém, Pará

2018

Pedro Pacheco Mendes Filho

Trabalho de Modelagem Computacional/ Pedro Pacheco Mendes Filho. – Santarém, Pará, 2018.

25 p.

Professora: Dra. Marciana Lima Góes

Trabalho Acadêmico – Universidade Federal do Oeste do Pará
Instituto de Engenharia e Geociência
Programa de Ciência e Tecnologia, 2018.

Resumo

Esse trabalho tem como objetivo demonstrar o uso de métodos iterativos, no caso, de Gauss-Jacobi e Gauss-Seidel, para a resolução de sistemas lineares propostos e relacionando a problemas comuns, para fins de demonstrar suas diferenças enquanto desempenho e precisão. Além disso, mostrar como a Modelagem Computacional é um ótimo recurso para resolução de problemas de cálculo como operações matriciais, sistemas, somatória e equações lineares.

Palavras-chave: Gauss-Jacobi. Gauss-Seidel. Métodos Iterativos. Sistemas Lineares. Modelagem Computacional.

Sumário

	Introdução	5
1	MÉTODO DE GAUSS-JACOB	7
1.1	Valores Iniciais	7
1.2	CrITÉRIOS de Convergência	9
1.3	Algoritmo de Gauss-Jacob	11
1.4	Algoritmo Completo	12
2	RESOLUÇÃO DE PROBLEMAS	17
2.1	Transferência de calor em uma placa	18
2.2	Vitaminas equilibradas diariamente	21
3	CONCLUSÃO	25

Introdução

A resolução de sistemas lineares é muito importante para a resolução de problemas reais, para isso usamos a solução numérica com auxílio computacional para resolver e solucionar problemas de maneira rápida e precisa. O sistema linear é descrito por m equações com n incógnitas x_i como em: $a_{i1}x_1 + a_{i2}x_2 + \dots + a_{in}x_n = b_i$, e é usualmente representado na forma matricial $Ax = b$, que facilita a visualização computacional.

Os métodos iterativos usados para achar o valor aproximado, partem de um valor inicial e calculando a discretização numérica se aproximando cada vez do resultado, até o erro ser inferior à tolerância, significando assim a convergência do método; ou quando o número de iterações exceder o limite definido, portanto, não convergindo a um resultado. É possível encontrar um resultado tão aproximado quanto queira, mas a baixa tolerância pode significar um número de iterações muito grande e talvez desnecessário.

O método de Gauss-Jacob e de Gauss-Seidel usados na resolução dos problemas necessitam de alguns parâmetros, são eles, o sistema linear, o chute inicial, a tolerância desejada e também o limite de iterações.

Será apresentado o algoritmo desenvolvido em Python, tanto para o teste de critérios de convergência: critério de linha, diagonal dominante e critério de Sassenfeld; e também a resolução do métodos numéricos.

Os resultados serão demonstrados aos seus respectivos métodos, será também apresentado algumas iterações, o gráfico de rapidez de convergência e os parâmetros usados na resolução.

1 Método de Gauss-Jacob

O método de Gauss-Jacobi permite obter-se uma solução única para um sistema $Ax = b$, $A = (a_{ij})$ com $i, j = 1, \dots, n$ e $\det(A) \neq 0$. É denominado iterativo porque fornece uma sequência de raízes aproximadas obtidas através dos resultados anteriores. A construção do método é descrito como a transformação do sistema $Ax = b$ para forma $x = Cx + d$, que a partir dela, lança-se um valor inicial para o sistema, para encontrar a primeira solução aproximada, dando início ao processo iterativo do método, porém só será executado se os critérios de convergência forem atendidos.

1.1 Valores Iniciais

Primeiramente, para a execução de um sistema pelo método, é necessário definir os valores iniciais para aquele específico problema. São parâmetros como as funções do sistema, chute inicial, tolerância, limite de iterações e número de variáveis.

Esses parâmetros tem como características:

a) **Funções do sistema:**

As funções são colocadas em forma de matriz para que analise e resolva de forma rápida através de operações matriciais:

$$\left. \begin{array}{l} A = a_{1A}x + a_{2A}y \\ B = a_{1B}x + a_{2B}y \end{array} \right\} \rightarrow \begin{bmatrix} a_{1A} & a_{2A} \\ a_{1B} & a_{2B} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A \\ B \end{bmatrix}$$

$$Ax = b$$

No algoritmo, elas são definidas e armazenadas em variáveis como:

$$A \rightarrow \begin{bmatrix} a_{1A} & a_{2A} \\ a_{1B} & a_{2B} \end{bmatrix} \quad b \rightarrow \begin{bmatrix} A \\ B \end{bmatrix}$$

Em exemplo, ele se apresenta dessa forma:

$$A \rightarrow \begin{bmatrix} 1 & 2 & 3 \\ 2 & 5 & 6 \\ 2 & 3 & 4 \end{bmatrix} \quad b \rightarrow \begin{bmatrix} 23,5 \\ 50 \\ 36 \end{bmatrix}$$

Agora no Python,

```

1      A = numpy.array ([[1 , 2 , 3] ,
2      [2 , 5 , 6] ,
3      [2 , 3 , 4]] , dtype=float )
4
5      b = [23.5 , 50 , 36]
6

```

A biblioteca *numpy* servirá para auxiliar nas operações matriciais.

b) Chute Inicial

Em valores iniciais você define os valores *b* para cada equação do sistema. No caso, esses valores vem de análise previamente feitas do sistema. O chute inicial vai ser determinante para definir a quantidade de iterações necessárias para que se encontre o valor aproximado, de acordo com a tolerância definida, de cada raiz.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

E no Python,

```

1      chute_inicial = [0 , 0 , 0]

```

c) Tolerância:

A tolerância é o valor menor valor de aproximação da raiz aceito pelo algoritmo. Quanto menor o valor, maior será a precisão do método, porém, aumentará também o número de iterações. Nos problemas, definimos como padrão a tolerância de 1×10^{-3} . No python, fica:

```

1      tolerancia= 1E-3

```

d) Limite de iterações:

O limite de iterações é necessário para o caso de houver erros programas e levar ao programa executar inúmeras iterações e entrando em um *loop infinito*, fazendo o código executar cálculos desnecessário e ocupando muita memória.

Definimos por padrão, o número limite de iterações como 1×10^3 . No Python, fica:

```
1 iterMax= 1E3
```

e) **Números de Variáveis:**

No caso do número de variáveis é necessário para alguns cálculos. No caso, fazemos com que Python detecte de forma automática o tamanho da primeira linha da matriz A , e defina como número de coeficientes, logo, também o número de variáveis. No algoritmo, fica:

```
1 coef = A[0, :].size
```

f) **Tratamento da Matriz:**

1.2 Critérios de Convergência

Os critérios de convergência que envolve o método iterativo de Gauss-Jacobi, são esses:

a) **Critério de Linha:**

$$\max_{1 \leq i \leq n} \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}^*| < 1$$

Em Python,

```
1 max_somatoria = 0
2 for i in range(A[:,0].size):
3     somatoria = 0
4     for j in range(n):
5         somatoria += abs(A[i,j])
6
7     if(somatoria > max_somatoria):
8         max_somatoria = somatoria
9
10    if(max_somatoria < 2):
11        print("Critério de Linha foi Atendido!")
12    return True
```

b) **Se a matriz for diagonal dominante.**

Em Python,

```
1 diagonais = 0
2 for i in range(A[:,0].size):
3     if(A[i,i] > A[:,i].sum()-A[i,i] and A[i,i] > A[i,:].sum()-
A[i,i]):
```

```

4         diagonais +=1
5
6         if diagonais == n:
7             print('Critério da Diagonal Dominante Atendido!')
8             return True
9

```

c) **Critério de Sassenfeld:**

$$\max_{1 \leq i \leq n} \beta_i < 1$$

, onde

$$\beta_i = \sum_{j=1}^{i-1} |a_{ij}^*| \beta_j + \sum_{j=i+1}^n |a_{ij}^*|$$

Em Python,

```

1         max_sassenfeld = 0
2         bj = [0]
3         for i in range(A[:,0].size):
4             bi = 0
5             if i > 0:
6                 for j in range(A[:,0].size - 1):
7                     bi += abs(A[i,j]) * bj[j]
8
9                 for j in range(i + 1, n):
10                    bi += abs(A[i,j])
11
12                bj.append(bi)
13
14                if(bi > max_sassenfeld):
15                    max_sassenfeld = bi
16
17                if(max_sassenfeld < 1):
18                    print('Critério de Sassenfeld foi Atendido!')
19                    return True

```

No algoritmo, para executar os critérios do sistema, apresenta primeiramente um tratamento, que é a divisão de cada equação pela sua diagonal, para facilitar o algoritmo na verificação do critério e para fins de análise. Nos problemas, todos os critérios são testados respectivamente como condição para execução do método, se um desses critérios forem atendido o método é executado. Se nenhum deles retornar um valor verdadeiro, o algoritmo cessará pois não irá convergir.

1.3 Algoritmo de Gauss-Jacob

O método, de forma geral, consiste em dado x_0 , aproximação inicial, obter aproximações através da relação recursiva $x(k+1) = Cx^k + d$.

Para $i = 1, 2, \dots, n$, podemos reescrever o sistema na (2.2) seguinte forma:

$$x_i = \frac{b_i - \sum_{j=1, j \neq i}^n a_{ij}x_j}{a_{ii}}, i = 1, 2, \dots, n.$$

até que o critério de parada seja satisfeito. Definiremos agora, o critério de parada e o algoritmo de Gauss-Jacob.

- a) **Critério de Parada** Os critérios de parada são usados para fim de não manter o algoritmo rodando quando o resultado já é o bastante preciso e quando, por algum motivo, não alcançar a convergência. Há dois critérios que encerraram a execução do método, são eles:

Máxima Iterações:

Será quando o limite de iterações forem atingidas, que poderá ser o caso de não convergência (serve para não sobrecarregar o computador quando não houver convergência).

Critério de distância:

Critério de distância avalia a precisão que o algoritmo chegou a partir do módulo da diferença da iteração atual e da anterior se o valor, porém o erro de cada variável do sistema tem que estar abaixo da tolerância para ser satisfeito:

$$d^{(k)} = \max_{1 \leq i \leq n} |x_i^{(k)} - x_i^{(k-1)}| < \varepsilon_1$$

Então a condicional para execução do *loop iterativo* do método, se torna em Python:

```

1      iterAtual = 0 #Zerando as Iteracoes
2      erro = 2      #Erro Inicial
3      while (k < iterMax and erro >= tol):
4          ###Execucao do metodo
5  
```

Enquanto, que o condicional presente dentro de *while* resultar verdadeiro (*True*), o método iterativo irá executar.

Obs: A variável 'erro' é definido 2 apenas para fim que o condicional da iteração inicial resulte em verdadeiro, se não fosse definido, o método nunca iria executar.

b) Algoritmo de Gauss Jacobi:

Agora, com todas as variáveis definidas, definiremos o método iterativo usado. Todos os valores entram no algoritmo do método e são executadas até que ache os valores das raízes, ou seja, a convergência. O algoritmo é auto-regulável, então, executará sistemas bidimensionais de tamanhos totalmente distintos. Seu desenvolvimento foi feito em Python, como apresentado logo abaixo:

```

1  while(k < iterMax and erro >= tol):
2      erro = 0      #Redefine o erro para iteracao atual
3
4      xant = x.copy() #Copia os valores de 'x' antigos
5
6      for i in range(n):
7          soma = 0 #Redefine a somatoria para Iteracao atual
8          for j in range(n):
9              if(j != i):
10                 soma = soma + float(A[i,j])*xant[j] #Metodo de Gauss-
Jacob
11                 x[i] = (b[i] - soma)/float(A[i,i])      #Metodo de Gauss
Jacob(se j = i)
12                 if(abs(x[i]-xant[i])>erro): #Acha o Maximo erro das raizes
13                     erro = abs(x[i] - xant[i])
14                 k +=1 #Incremento da Iteracao
15

```

1.4 Algoritmo Completo

Apresentado abaixo o algoritmo completo com todas as definições e algumas ferramentas para análise como o *matplotlib* (para plotagem de gráfico).

```

1  #-*- coding: utf-8 -*-
2  """
3  Nome = Pedro Pacheco Mendes Filho
4  Turma = CTEC2017
5  Matricula = 201701029
6  Professora = Marciana Lima Goes
7  Metodo = Metodo iterativo de Gauss Jacobi
8  """
9  import numpy as np
10 import matplotlib.pyplot as plt
11
12 def jacobi(n, A, b, iterMax, tol, x):
13
14
15     array.append(x.copy())

```

```

16     print(x)
17     k = 0
18     erro = 2
19
20     while(k < iterMax and erro >= tol):
21
22         erro = 0
23
24         xant = x.copy()
25
26         for i in range(n):
27             soma = 0
28             for j in range(n):
29                 if(j != i):
30                     soma = soma + float(A[i,j])*xant[j]
31             x[i] = (b[i] - soma)/float(A[i,i])
32             if(abs(x[i]-xant[i])>erro):
33                 erro = abs(x[i] - xant[i])
34         k +=1
35
36         array.append(x.copy())
37
38
39     if(erro < tol):
40         print('iteracoes = ', k)
41         print('erro = ', erro)
42         print("x = " , x)
43         sist_array = np.array(array, dtype=float)
44         return k,sist_array
45
46     else:
47         print('Nao houve convergencia')
48         return 0, 0
49
50
51
52
53 def crit_convergencia(n):
54
55     #Simplificando o sistema
56     for i in range(A[:,0].size):
57         b[i] = b[i]/A[i,i]
58         A[i] = A[i]/A[i,i]
59
60
61     #####Criterio de Linha
62     max_somatoria = 0

```

```

63     for i in range(A[:,0].size):
64         somatoria = 0
65         for j in range(n):
66             somatoria += abs(A[i,j])
67
68         if(somatoria > max_somatoria): max_somatoria = somatoria
69
70     if(max_somatoria < 2):
71         print("Critério de Linha foi Aprovado!")
72         return True
73
74 #####Critério de Diagonal dominante
75     diagonais = 0
76     for i in range(A[:,0].size):
77         if(A[i,i]>A[:,i].sum()-A[i,i] and A[i,i]>A[i,:].sum()-A[i,i]):
78             diagonais +=1
79     if diagonais == n:
80         print('Critério da Diagonal Dominante Aprovado!')
81
82 #####Critério de Sassenfeld
83     max_sassenfeld = 0
84     bj = [0]
85     for i in range(A[:,0].size):
86         bi = 0
87         if i > 0:
88             for j in range(A[:,0].size - 1):
89
90                 bi += abs(A[i,j]) * bj[j]
91
92         for j in range(i + 1, n):
93             bi += abs(A[i,j])
94
95         bj.append(bi)
96
97         if(bi > max_sassenfeld):
98             max_sassenfeld = bi
99
100     if(max_sassenfeld < 1):
101         print('Critério de Sassenfeld foi Aprovado!')
102         return True
103     print("Nenhum critério foi atendido!")
104
105 A = np.array([[1,2,3],
106               [2,5,6],
107               [2,3,4]], dtype=float)
108
109

```



```
110 chute_inicial = [0,0,0]
111 b = [23.5,50,36]
112 array = []
113 iterMax = 1000
114 tol = 1E-3
115 n = A[0,:].size
116 if(crit_convergencia(n)):
117     total_iter, sstm = jacobi(n,A,b,iterMax,tol,chute_inicial)
118
119     for i in range(n):
120         plt.plot(range(tot_iter+1),sstm[:,i], '-o', label='X{}'.format(i+1))
121     plt.xlabel("Iteracoes")
122     plt.ylabel("Aproximacao")
123     plt.title("Gauss Jacob")
124     plt.grid(True, linestyle='-.')
125     plt.legend()
126     plt.savefig("gaus.png")
```


2 Resolução de Problemas

Resolveremos problemas do cotidiano para testar a veracidade do método, desempenho e precisão. Os parâmetros de entrada do problema serão:

$$\left. \begin{array}{l} A = a_{1A}x + a_{2A}y \\ B = a_{1B}x + a_{2B}y \end{array} \right\} \rightarrow \begin{bmatrix} a_{1A} & a_{2A} \\ a_{1B} & a_{2B} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} A \\ B \end{bmatrix}$$

$$A \rightarrow \begin{bmatrix} a_{1A} & a_{2A} \\ a_{1B} & a_{2B} \end{bmatrix} \quad b \rightarrow \begin{bmatrix} A \\ B \end{bmatrix}$$

a) **Matriz de A:**

A matriz dos coeficientes das variáveis:

$$A \rightarrow \begin{bmatrix} a_{1A} & a_{2A} \\ a_{1B} & a_{2B} \end{bmatrix}$$

b) **Matriz de b:**

$$b \rightarrow \begin{bmatrix} A \\ B \end{bmatrix}$$

c) **Chute Inicial:** $x_1, x_2, x_3, \dots, x_n = 0$

d) **Tolerância:** 1×10^{-3}

e) **Iterações Máxima:** 1×10^3

E como resultado terá valores como:

a) **Critério de Convergência:**

Em qual critério o sistema passou.

b) **Número de Iterações:**

Quantas iterações para achar o resultado com a tolerância desejada.

c) **Primeiras Iterações:**

O resultado das 3 primeiras iterações.

d) **Resultado Obtido:**

Resultado obtido pelo método com a tolerância desejada.

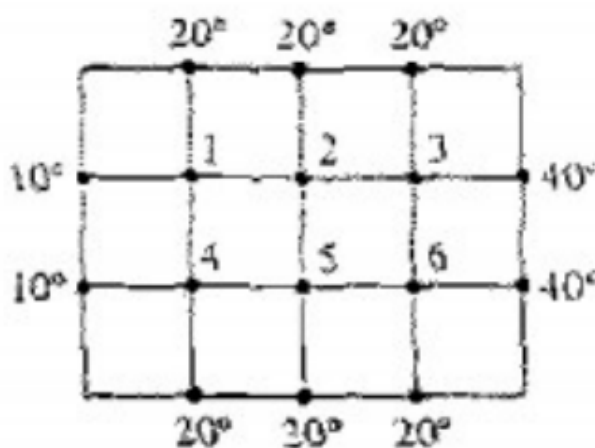
- e) **Gráfico de Convergência:** É o gráfico que demonstra qual a rapidez que o método convergiu para o resultado.

2.1 Transferência de calor em uma placa

Uma consideração importante no estudo da transferência de calor é a de se determinar a distribuição de temperatura assintótica de uma placa fina quando a temperatura em seu bordo é conhecida. Suponha que a placa represente uma seção transversal de uma barra de metal, com fluxo de calor desprezível na direção perpendicular à placa. Sejam T_1, \dots, T_6 as temperaturas em seis vértices interiores do reticulado da figura. A temperatura num vértice é aproximadamente igual à média dos quatro vértices vizinhos mais próximos - à esquerda, acima, à direita, e abaixo. Por exemplo,

$$T_1 = \frac{(10 + 20 + T_2 + T_4)}{4}, \text{ ou } 4T_1 - T_2 - T_4 = 30$$

Escreva um sistema de seis equações cuja solução fornece estimativas para as temperaturas T_1, \dots, T_6 .



Resolução:

A solução desse problema primeiramente devemos encontrar o sistema. O próprio problema dá um exemplo para seguir, então foi feito como demonstrado a definição do problema

abaixo:

$$\begin{cases} -4T_1 + T_2 + T_4 = -30 \\ T_1 - 4T_2 + T_3 + T_5 = -20 \\ T_2 - 4T_3 + T_6 = -60 \\ T - 1 - 4T_4 + T_5 = -30 \\ T_2 + T_4 - 4T_5 + T_6 = -20 \\ T_3 + T_5 - 4T_6 = -60 \end{cases}$$

criando agora as matrizes A e b :

$$A \rightarrow \begin{bmatrix} -4 & 1 & 0 & 1 & 0 & 0 \\ 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 1 & -4 & 0 & 0 & 1 \\ 1 & 0 & 0 & -4 & 1 & 0 \\ 0 & 1 & 0 & 1 & -4 & 1 \\ 0 & 0 & 1 & 0 & 1 & -4 \end{bmatrix} \quad b \rightarrow \begin{bmatrix} -30 \\ -20 \\ -60 \\ -30 \\ -20 \\ 60 \end{bmatrix}$$

Colocando-a no algoritmo ela retornará os resultados:

- a) **Critério de Convergência:** *Critério de linha.*
- b) **Número de Iterações:** 20.
- c) **Precisão:** $7,14 \times 10^{-3}$
- d) **Primeiras Iterações:**

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{1} \begin{bmatrix} 7,5 \\ 5 \\ 15 \\ 7,5 \\ 5 \\ 15 \end{bmatrix} \xrightarrow{2} \begin{bmatrix} 10,625 \\ 11,875 \\ 20 \\ 10,625 \\ 11,875 \\ 20 \end{bmatrix} \xrightarrow{3} \begin{bmatrix} 13,125 \\ 15,625 \\ 22,96875 \\ 13,125 \\ 15,625 \\ 22,96875 \end{bmatrix}$$

- e) **Resultado Obtido:**

$$\begin{bmatrix} 17,142 \\ 21,427 \\ 27,142 \\ 17,142 \\ 21,427 \\ 27,142 \end{bmatrix}$$

f) Gráfico de Convergência:

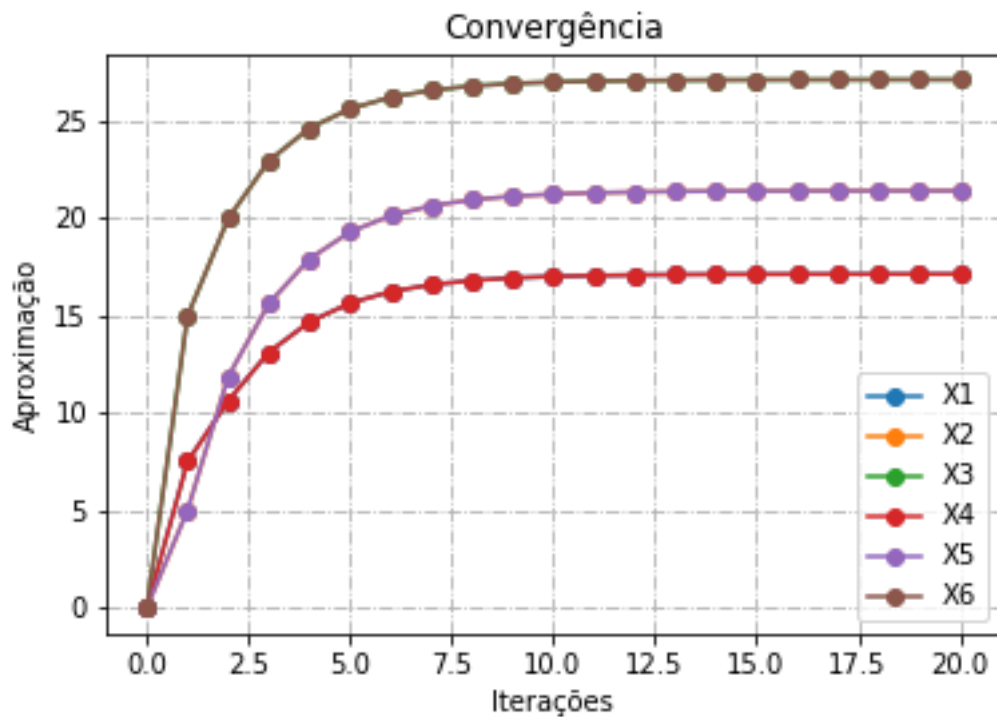


Gráfico de convergência de Gauss Jacob no sistema da placa transversal

2.2 Vitaminas equilibradas diariamente

Sabe-se que uma alimentação diária equilibrada em vitaminas deve constar de 170 unidades de vitamina A, 180 unidades de vitamina C, 140 unidades de vitamina C, 180 unidades de vitamina D e 350 unidades de vitamina E.

Com o objetivo de descobrir como deverá ser uma refeição equilibrada, foram estudados cinco alimentos. Fixada na mesma quantidade (1g) de cada alimento, determinou-se:

1. O alimento I tem 1 unidade de vitamina A, 10 unidades de vitamina B, 1 unidade de vitamina C, 2 unidades de vitamina D e 2 unidades de vitamina E.
2. O alimento II tem 9 unidades de vitamina A, 1 unidade de vitamina B, 0 unidade de vitamina C, 1 unidade de vitamina D e 1 unidade de vitamina E.
3. O alimento III tem 2 unidades de A, 2 unidades de B, 5 unidades de C, 1 unidade de D e 2 unidades de E.
4. O alimento IV tem 1 unidade de A, 1 unidade de B, 1 unidade de C, 2 unidades de D e 13 unidades de E.

5. O alimento V tem 1 unidade de A, 1 unidade de B, 1 unidade de C, 9 unidades de D e 2 unidades de E.

Quanto gramas de cada um dos alimentos I, II, III, IV e V devemos ingerir diariamente para que nossa alimentação seja equilibrada?

Resolução:

A solução desse problema primeiramente devemos encontrar o sistema. O sistema encontrado a partir da análise do problema foi esse:

$$\begin{cases} 1I + 9II + 2III + IV + V = 170 \\ 10I + II + 2III + IV + V = 180 \\ 1I + 0II + 5III + IV + V = 140 \\ 2I + II + III + 2IV + 9V = 180 \\ 2I + II + 2III + 13IV + 2V = 350 \end{cases}$$

criando agora as matrizes A e b , já ajustadas, de modo de tentar a diagonal dominante, apenas trocando as posições da equações:

$$A \rightarrow \begin{bmatrix} 10 & 1 & 2 & 1 & 1 \\ 1 & 9 & 2 & 1 & 1 \\ 1 & 0 & 5 & 1 & 1 \\ 2 & 1 & 2 & 13 & 2 \\ 2 & 1 & 1 & 2 & 9 \end{bmatrix} \quad b \rightarrow \begin{bmatrix} 180 \\ 170 \\ 140 \\ 350 \\ 180 \end{bmatrix}$$

Colocando-a no algoritmo ela retornará os resultados:

- a) **Critério de Convergência:** *Critério de linha.*
- b) **Número de Iterações:** 20.
- c) **Precisão:** $6,05 \times 10^{-4}$
- d) **Primeiras Iterações:**

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \xrightarrow{1} \begin{bmatrix} 18 \\ 18,888 \\ 28 \\ 26,923 \\ 20 \end{bmatrix} \xrightarrow{2} \begin{bmatrix} 5,818 \\ 5,452 \\ 15,015 \\ 15,316 \\ 4,807 \end{bmatrix} \xrightarrow{3} \begin{bmatrix} 12,439 \\ 12,669 \\ 22,811 \\ 22,558 \\ 13,029 \end{bmatrix}$$

e) **Resultado Obtido:**

$$\begin{bmatrix} 9,9998 \\ 9,9998 \\ 19,9997 \\ 19,9998 \\ 9,9997 \end{bmatrix}$$

f) **Gráfico de Convergência:**

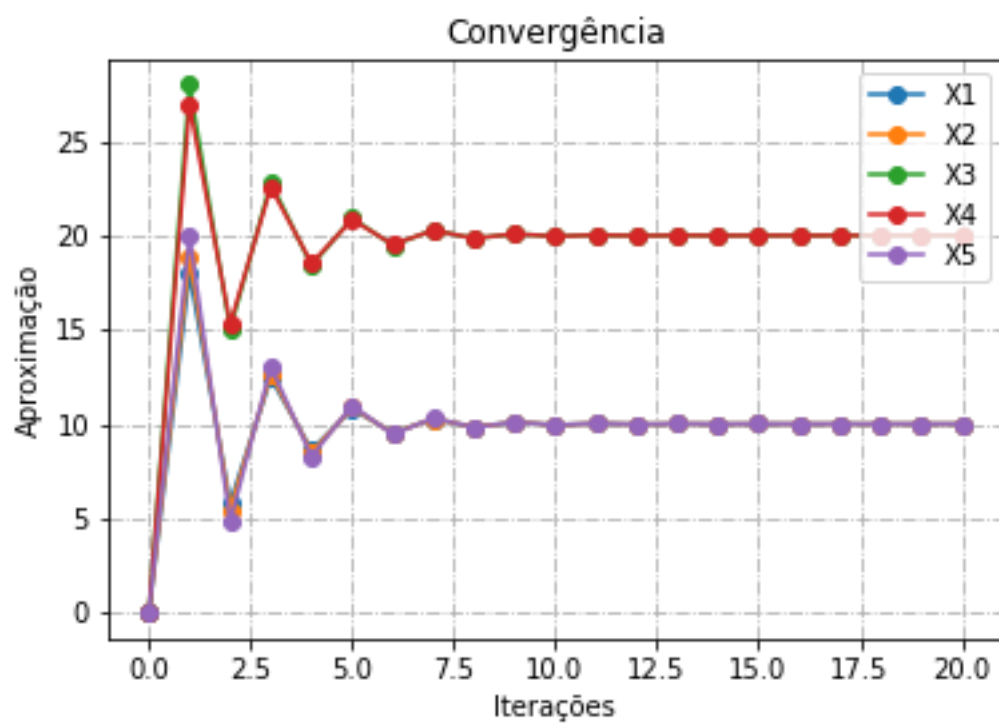


Gráfico de convergência de Gauss Jacob no sistema da vitaminas diárias

3 Conclusão

O método numérico de Gauss-Jacob é preciso e rápido para resolver sistemas, porém é limitado em questão de abrangência de sistema, pois, somente convergem sistemas se, e somente se, passar pelos seus *Crítérios de Convergência*. Por isso, não poderá ser usado para resolução para qualquer problema. A convergência de sistemas é rápida, por isso, é usual tratando sistemas grandes, pois pelo método convencional é inviável resolver número grande de equações e variáveis, o que torna o método eficaz. Há também a possibilidade de paralelização de seu algoritmo, o que aumenta mais ainda a rapidez do código.