Métodos Numéricos para Engenharia

MÓDULO 9 - ALGEBRA LINEAR NUMERICA – JACOBI - SEIDEL

PROFESSOR LUCIANO NEVES DA FONSECA

Problema $Ax = b$ Bem-Condicionado

Um problema $Ax = b$ é considerado **Bem-Condicionado** quando apresenta uma Solução única

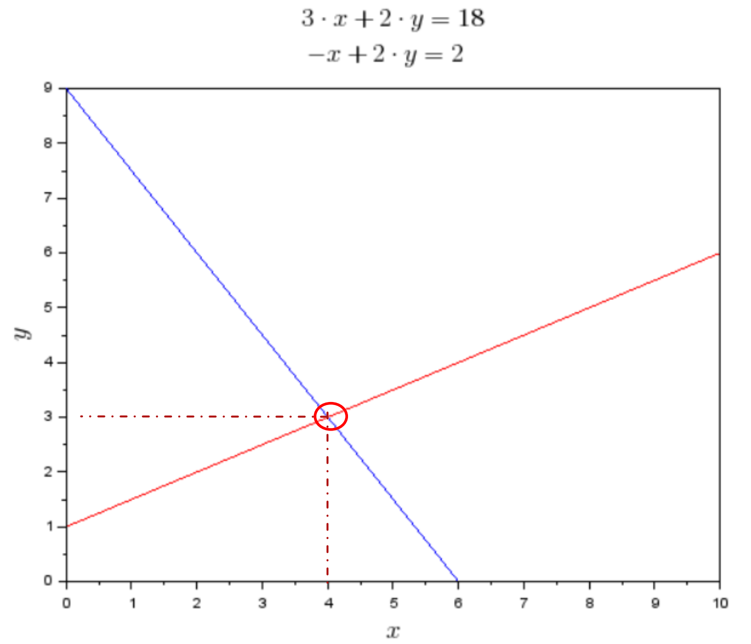
$$A = \begin{bmatrix} 3 & 2 \\ -1 & 2 \end{bmatrix}$$

$$b = \begin{bmatrix} 18 \\ 2 \end{bmatrix}$$

```
--> plotimplicit("3*x+2*y=18",0:10,0:9,'b')

--> plotimplicit("-x+2*y=2",0:10,0:9,'r')
```

```
--> A=[3,2;-1,2]
   3.    2.
  -1.    2.

--> b=[18;2]
   18.
   2.

--> x=inv(A)*b
   4.
   3.
```

```
--> det(A)
 ans  =

    8.
```



Condições Necessárias para uma solução única.

- O número de equações deve ser igual ao número de incógnitas, isto é, o número de linhas deve ser igual ao número de colunas da matriz A.
- Todas as equações devem ser linearmente independentes. Nenhuma linha pode ser uma combinação linear de outras linhas.
- O Determinante da Matriz Característica (A) não pode ser nulo.
- Um determinante característico (determinante da matriz a) muito pequeno pode causar erros de arredondamento e instabilidade na solução.

Um problema $Ax = b$ é considerado Mal-Condicionado quando apresenta infinitas soluções ou quando não apresenta soluções.

- 1) Determinante Nulo – Equações idênticas

infinitas soluções

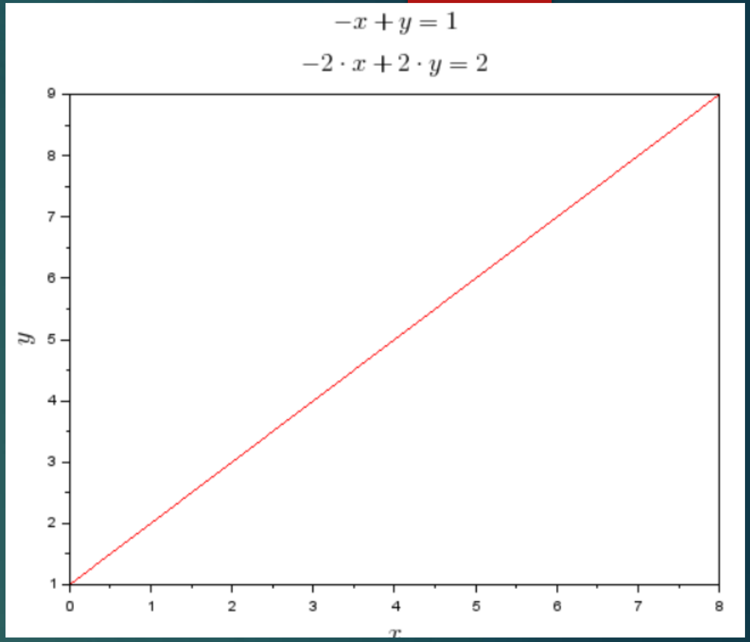
```
--> plotimplicit("-2*x+2*y=2",0:10,0:9,'b')

--> plotimplicit("-2*x+2*y=2",0:10,0:9,'r')
```

$$A = \begin{bmatrix} -1 & 1 \\ -2 & 2 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

```
--> A=[-1,1;-2,2];

--> b=[1;2];

--> x=inv(A)*b

inv: Problem is singular.
```

```
--> det(A)
 ans  =

   0.
```



- 2) Determinante Nulo – Equações Paralelas

não há solução

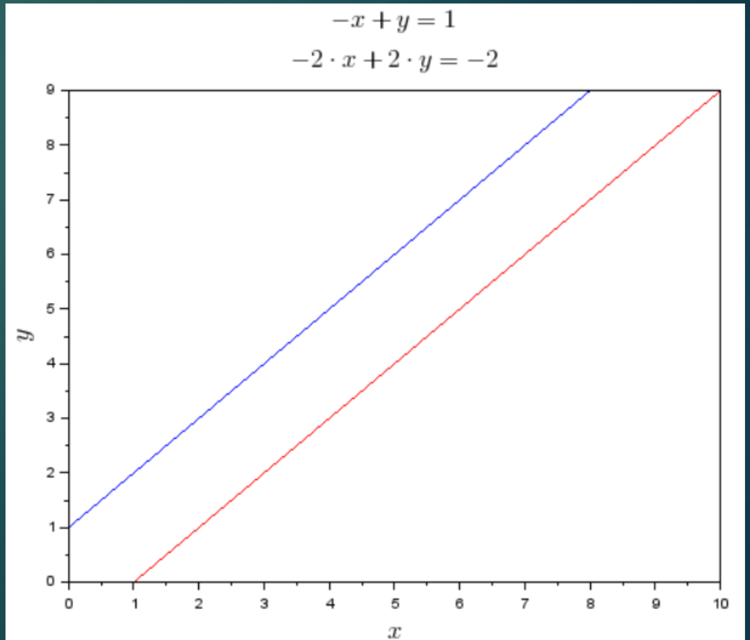
```
--> plotimplicit("-x+y=1",0:10,0:9,'b')

--> plotimplicit("-2*x+2*y=-2",0:10,0:9,'r')
```

$$A = \begin{bmatrix} -1 & 1 \\ -2 & 2 \end{bmatrix}$$

$$b = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$

```
--> A=[-1,1;-2,2];

--> b=[1;-2];

--> x=inv(A)*b

inv: Problem is singular.
```

```
--> det(A)
 ans  =

   0.
```



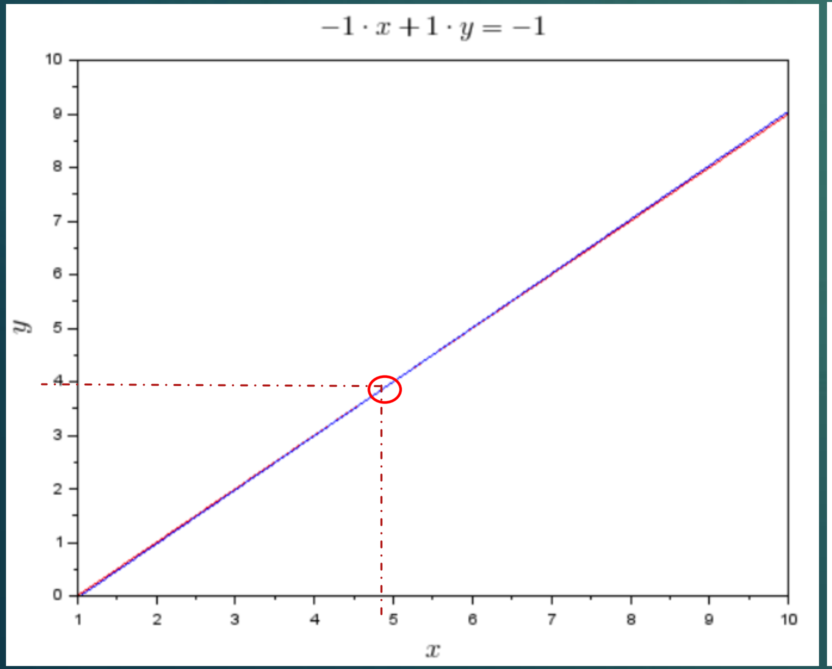
3) Determinante de A muito pequeno

- Equações com inclinações muito próximas, causando erros de arredondamento.
- Solução única mal definida
- Notar que pequenas mudanças nos coeficientes de [A] ou [b] alteram significativamente o resultado final x

$$A = \begin{bmatrix} -1.01 & 1 \\ -1 & 1 \end{bmatrix} \quad b = \begin{bmatrix} -1.05 \\ -1 \end{bmatrix}$$

```
--> A=[-1.01,1;-1,1];
--> det(A)
  -0.010000000
--> b=[-1.05;-1];
--> x=inv(A)*b
    5.
    4.
```

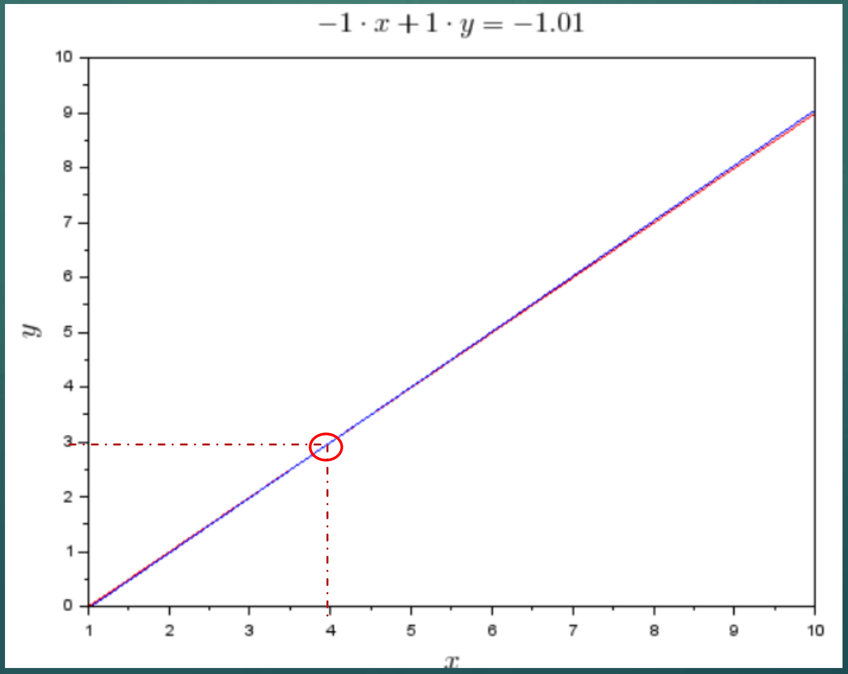
```
--> plotimplicit("-1.01*x+y=-1.05",0:10,0:10,'b')

--> plotimplicit("-1*x+y=-1",0:10,0:10,'r')
```



$$A = \begin{bmatrix} -1.01 & 1 \\ -1 & 1 \end{bmatrix} \quad b = \begin{bmatrix} -1.05 \\ -1.01 \end{bmatrix}$$

```
--> A=[-1.01,1;-1,1];
--> det(A)
  -0.010000000
--> b=[-1.05;-1.01];
--> x=inv(A)*b
    4.
    2.99
```

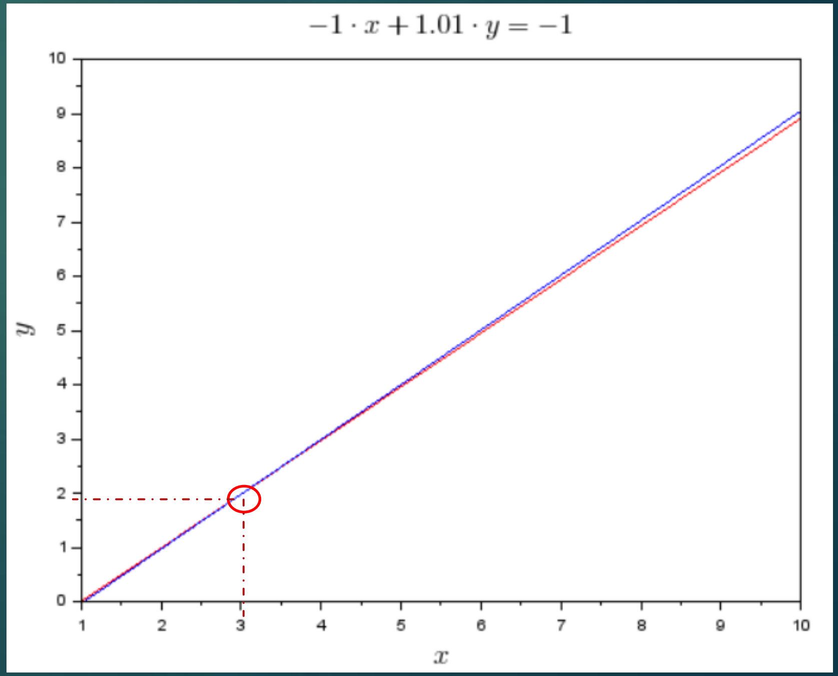
```
--> plotimplicit("-1.01*x+y=-1.05",0:10,0:10,'b')

--> plotimplicit("-1*x+y=-1.01",0:10,0:10,'r')
```



$$A = \begin{bmatrix} -1.01 & 1 \\ -1 & 1.01 \end{bmatrix} \quad b = \begin{bmatrix} -1.05 \\ -1 \end{bmatrix}$$

```
--> A=[-1.01,1;-1,1.01];
--> det(A)
  -0.020100000
--> b=[-1.05;-1];
--> x=inv(A)*b
    3.009950249
    1.990049751
```

```
--> plotimplicit("-1.01*x+y=-1.05",0:10,0:10,'b')

--> plotimplicit("-1*x+1.01*y=-1",0:10,0:10,'r')
```



4) Det(A) muito grande $A^{-1}A$ muito diferente da Matriz identidade I

```
--> A=testmatrix('hilb',7)
 A  =

   49.     -1176.      8820.      -29400.      48510.     -38808.      12012.
  -1176.    37632.    -317520.     1128960.   -1940400.    1596672.    -504504.
   8820.   -317520.    2857680.   -10584000.   18711000.  -15717240.    5045040.
  -29400.   1128960.  -10584000.   40320000.  -72765000.   62092800.  -20180160.
   48510.  -1940400.   18711000.  -72765000.   1.334D+08  -1.153D+08   37837800.
  -38808.   1596672.  -15717240.   62092800.  -1.153D+08   1.006D+08  -33297264.
   12012.  -504504.    5045040.   -20180160.   37837800.  -33297264.   11099088.

--> det_gauss(A,%F)
 ans  =

   2.068D+24

--> inv_gauss(A,%F)*A
 ans  =

   1.          -5.053D-11   4.471D-10  -1.323D-09   4.343D-09  -2.928D-09   3.550D-10
   1.728D-12   1.           7.721D-10  -2.157D-09   4.103D-09  -3.908D-09   2.234D-09
   1.494D-13   3.101D-11   1.           3.674D-10  -1.650D-10   1.188D-09  -8.566D-11
   6.236D-13  -1.664D-11   2.090D-11   1.           1.234D-09   8.516D-10   3.370D-10
   5.451D-14  -2.048D-11  -1.348D-11   5.394D-11   1.           7.293D-10  -5.535D-10
   7.589D-13  -2.255D-11   2.255D-10  -9.020D-10   1.517D-09   1.           1.469D-10
  -7.866D-14   2.849D-12   4.427D-11  -1.771D-10   3.320D-10  -7.578D-10   1.
```

```
--> A
 A  =

   49.     -1176.      8820.      -29400.      48510.      -38808.      12012.
  -1176.    37632.    -317520.     1128960.   -1940400.     1596672.    -504504.
   8820.   -317520.    2857680.   -10584000.   18711000.   -15717240.    5045040.
  -29400.   1128960.  -10584000.   40320000.  -72765000.    62092800.  -20180160.
   48510.  -1940400.   18711000.  -72765000.   133402500.  -115259760.   37837800.
  -38808.   1596672.  -15717240.   62092800.  -115259760.   100590336.  -33297264.
   12012.  -504504.    5045040.   -20180160.   37837800.   -33297264.    11099088.

--> b=[1;2;3;4;5;6;7];

--> inv(A)*b
Warning :
matrix is close to singular or badly scaled. rcond = 1.0150E-09
 ans  =

   7.
   5.282142857
   4.342063492
   3.713095238
   3.253823954
   2.900613276
   2.619197469

--> A(1,1)=50
 A  =

   50.     -1176.      8820.      -29400.      48510.      -38808.      12012.
  -1176.    37632.    -317520.     1128960.   -1940400.     1596672.    -504504.
   8820.   -317520.    2857680.   -10584000.   18711000.   -15717240.    5045040.
  -29400.   1128960.  -10584000.   40320000.  -72765000.    62092800.  -20180160.
   48510.  -1940400.   18711000.  -72765000.   133402500.  -115259760.   37837800.
  -38808.   1596672.  -15717240.   62092800.  -115259760.   100590336.  -33297264.
   12012.  -504504.    5045040.   -20180160.   37837800.   -33297264.    11099088.

--> inv(A)*b
Warning :
matrix is close to singular or badly scaled. rcond = 2.0301E-09
 ans  =

   3.5
   3.532142857
   3.175396825
   2.838095238
   2.553823954
   2.317279942
   2.119197469
```

Detectamos que a Matriz Característica A está mal condicionada, por o seu determinante ser muito grande ($2.068\ 10^{24}$) e por $A^{-1}A \neq I$ (quadro acima)

No quadro da direita, vemos que na resolução do sistema, a mudança de um único elemento $A(1,1)$ de 49 para 50, alterou de maneira significativa a solução.

1) Métodos Iterativos - Gauss Jacobi

Neste método, consideramos que cada uma das linhas do nosso sistema de equações Ax=b é, na verdade, uma equação para a qual queremos encontrar a raiz.
Para o cálculo das raízes usamos exatamente o método do Ponto Fixo, aplicado a cada uma das linhas.
Como o método é iterativo, começamos com uma vetor solução inicial x^0 qualquer e calculamos a solução x^1, com a solução x^k calculamos a solução x^{k+1}e assim iterativamente, parando quando o erro relativo entre soluções subsequentes for menor que uma certa tolerância.

$$\begin{array}{cccccc} a_{11}x_1 & +a_{12}x_2 & +a_{13}x_3 & = & b_1 \\ a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & = & b_2 \\ a_{31}x_1 & +a_{32}x_2 & +a_{33}x_3 & = & b_3 \end{array}$$

Isolando uma variável por linha temos:

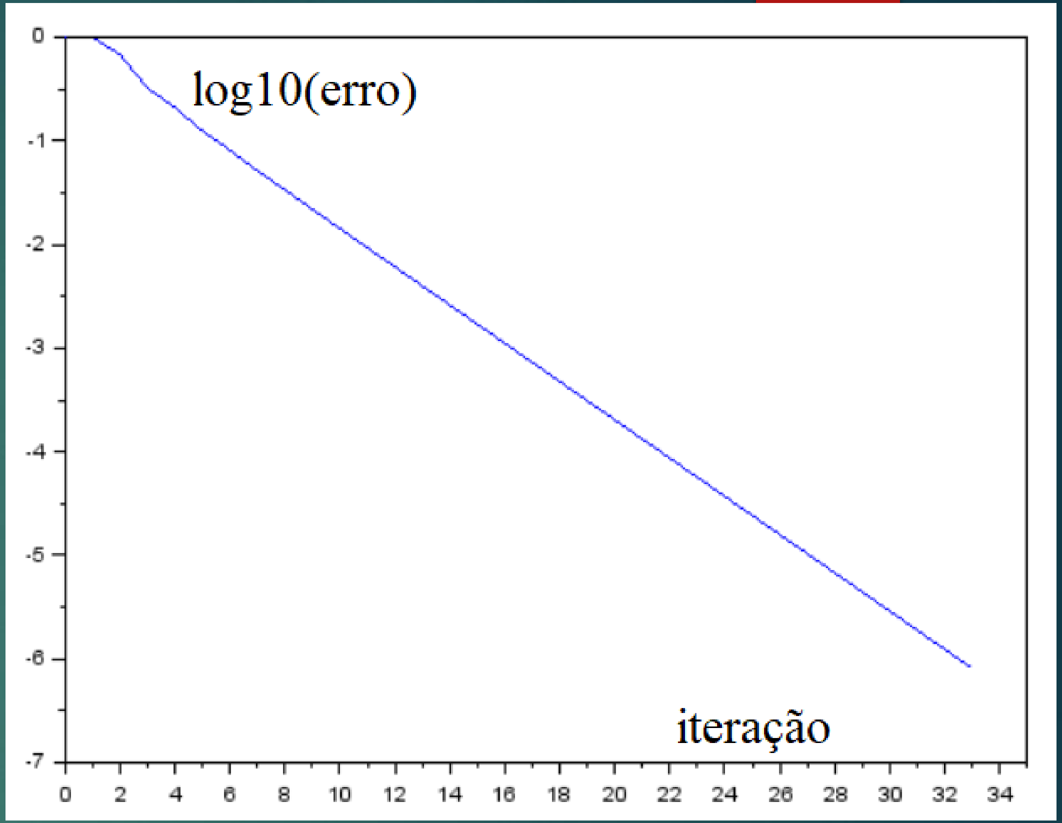
$$\boxed{\begin{aligned} x_1^{k+1} &= \frac{1}{a_{11}}(b_1 - a_{12}x_2^k - a_{13}x_3^k) \\ x_2^{k+1} &= \frac{1}{a_{22}}(b_2 - a_{21}x_1^k - a_{23}x_3^k) \\ x_3^{k+1} &= \frac{1}{a_{33}}(b_3 - a_{31}x_1^k - a_{32}x_2^k) \end{aligned}}$$

```
function x0=GaussJacobi(A,b,x_ini,tol,prt)
    N = length(b);
    erro = 1;
    x0=x_ini
    sol(:,1)=x0
    erro_vector(1,1)=erro
    for k=2:500
        for j=1:N
            x1(j)=(b(j)-A(j,:)*x0 + A(j,j)*x0(j))/A(j,j);
        end
        if(norm(x1)<>0)  erro=norm(x1-x0)/norm(x1)  end
        x0=x1;
        sol(:,k)=x0
        erro_vector(1,k)=erro
        if  erro<tol   break   end
    end
    ⋮
endfunction
```

Algoritmo para o Método Iterativo de Gauss Jacobi

```
--> A=[8,2,4;1,11,3;5,3,10]
 A  =


   8.    2.    4.
   1.    11.   3.
   5.    3.    10.

--> b=[9;18;72];

--> x=inv(A)*b
 x  =


  -3.2311475
  -0.5163934
   8.9704918
```

```
--> x=GaussJacobi(A,b,[0;0;0],1e-6,%T)

   8.    2.    4.    9.
   1.    11.   3.    18.
   5.    3.    10.   72.

  "erro"
  "x(1)"
  "x(2)"
  "x(3)"

         column 1 to 6
   1.   1.         0.6807807   0.3232971   0.2095781   0.124572
   0.   1.125     -2.8840909  -1.8409091  -3.3160072  -2.7548089
   0.   1.6363636 -0.4295455   0.2222107  -0.5883471  -0.2586697
   0.   7.2        6.1465909   8.7709091   8.0537913   9.0345077
         column 7 to 12
   0.0815145   0.051581    0.0337982   0.021842    0.0143004   0.0093114
  -3.3275864  -3.0582137  -3.2880739  -3.1652319  -3.2595842  -3.2050605
  -0.5771559  -0.4215845  -0.5502369  -0.4798793  -0.5329175  -0.5018323
   8.6550054   9.03694     8.8555822   9.009108    8.9265797   8.9896673
         column 13 to 18
   0.0060918   0.0039774   0.0026012   0.0017      0.0011116   0.0007267
  -3.2443756  -3.2205413  -3.2370826  -3.2267558  -3.233758   -3.2293072
  -0.5239947  -0.5104422  -0.5197834  -0.5139207  -0.5178794  -0.5153549
   8.9530799   8.9793862   8.9634033   8.9744763   8.9675541   8.9722428
         column 19 to 24
   0.0004752   0.0003107   0.0002031   0.0001328   0.0000869   0.0000568
  -3.2322827  -3.2303705  -3.2316378  -3.2308179  -3.2313585  -3.2310073
  -0.5170383  -0.5159543  -0.5166716  -0.516207   -0.516513   -0.5163141
   8.9692601   8.9712528   8.9699715   8.9708204   8.970271    8.9706331
         column 25 to 30
   0.0000371   0.0000243   0.0000159   0.0000104   0.0000068   0.0000044
  -3.2312381  -3.2310877  -3.2311863  -3.231122   -3.2311641  -3.2311366
  -0.5164447  -0.5163596  -0.5164154  -0.516379   -0.5164029  -0.5163873
   8.9703978   8.9705525   8.9704517   8.9705178   8.9704747   8.9705029
         column 31 to 34
   0.0000029   0.0000019   0.0000012   0.0000008
  -3.2311546  -3.2311429  -3.2311506  -3.2311456
  -0.5163975  -0.5163908  -0.5163952  -0.5163923
   8.9704845   8.9704966   8.9704887   8.9704938

após 34 iterações, erro=8.1e-07
 x  =

  -3.2311456
  -0.5163923
   8.9704938
```



O log decimal do erro diminui linearmente a cada iteração

2) Métodos Iterativos - Gauss Seidel

Neste método, também consideramos que cada uma das linhas do nosso sistema de equações Ax=b é, na verdade, uma equação para a qual queremos encontrar a raiz.

Para o cálculo das raízes usamos exatamente o método do Ponto Fixo, aplicado a cada uma das linhas.

Como o método é iterativo, começamos com uma vetor solução inicial x^0 qualquer e calculamos a solução x^1, com a solução x^k calculamos a solução x^{k+1}e assim iterativamente, parando quando o erro relativo entre soluções subsequentes for menor que uma certa tolerância.

Como no método de Gauss Jacobi, na iteração k+1, usamos todos os elemento da iteração k para calcular x_1^{k+1}. No entanto, para calcular x_2^{k+1}, usamos x_1^{k+1} (que já foi calculado), e assim sucessivamente.

$$
\begin{array}{lllll}
a_{11}x_1 & +a_{12}x_2 & +a_{13}x_3 & = & b_1 \\
a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & = & b_2 \\
a_{31}x_1 & +a_{32}x_2 & +a_{33}x_3 & = & b_3
\end{array}
$$

Isolando uma variável por linha temos:

$$
\boxed{
\begin{array}{lll}
\mathbf{x_1^{k+1}} & = & \dfrac{1}{a_{11}}(b_1 - a_{12}x_2^k - a_{13}x_3^k) \\
\mathbf{x_2^{k+1}} & = & \dfrac{1}{a_{22}}(b_2 - a_{21}\mathbf{x_1^{k+1}} - a_{23}x_3^k) \\
x_3^{k+1} & = & \dfrac{1}{a_{33}}(b_3 - a_{31}\mathbf{x_1^{k+1}} - a_{32}\mathbf{x_2^{k+1}})
\end{array}
}
$$

```
function x1=GaussSeidel(A,b,x_ini,tol,prt)
    N = length(b);
    erro = 1;
    x0=x_ini
    sol(:,1)=x0
    erro_vector(1,1)=erro
    for k=2:500
        x1(1)=(b(1)-A(1,2:N)*x0(2:N))/A(1,1);
        for j=2:N-1
            x1(j)=(b(j)-A(j,1:j-1)*x1(1:j-1)-A(j,j+1:N)*x0(j+1:N))/A(j,j);
        end
        x1(N)=(b(N)-A(N,1:N-1)*(x1(1:N-1)))/A(N,N);
        if(norm(x1)<>0)  erro=norm(x1-x0)/norm(x1)  end
        x0=x1;
        sol(:,k)=x0
        erro_vector(1,k)=erro
        if  erro<tol    break   end
    end
        ⋮
endfunction
```

Comparação Guass-Jacobi e Gauss Seidel

```
function x1=GaussSeidel(A,b,x_ini,tol,prt)
    N = length(b);
    erro = 1;
    x0=x_ini
    sol(:,1)=x0
    erro_vector(1,1)=erro
    for k=2:500
        x1(1)=(b(1)-A(1,2:N)*x0(2:N))/A(1,1);
        for j=2:N-1
            x1(j)=(b(j)-A(j,1:j-1)*x1(1:j-1)-A(j,j+1:N)*x0(j+1:N))/A(j,j);
        end
        x1(N)=(b(N)-A(N,1:N-1)*(x1(1:N-1)))/A(N,N);
        if(norm(x1)<>0) erro=norm(x1-x0)/norm(x1) end
        x0=x1;
        sol(:,k)=x0
        erro_vector(1,k)=erro
        if erro<tol break end
    end
    ⋮
endfunction
```

```
function x0=GaussJacobi(A,b,x_ini,tol,prt)
    N = length(b);
    erro = 1;
    x0=x_ini
    sol(:,1)=x0
    erro_vector(1,1)=erro
    for k=2:500
        for j=1:N
            x1(j)=(b(j)-A(j,:)*x0 + A(j,j)*x0(j))/A(j,j);
        end
        if(norm(x1)<>0) erro=norm(x1-x0)/norm(x1) end
        x0=x1;
        sol(:,k)=x0
        erro_vector(1,k)=erro
        if erro<tol break end
    end
    ⋮
endfunction
```

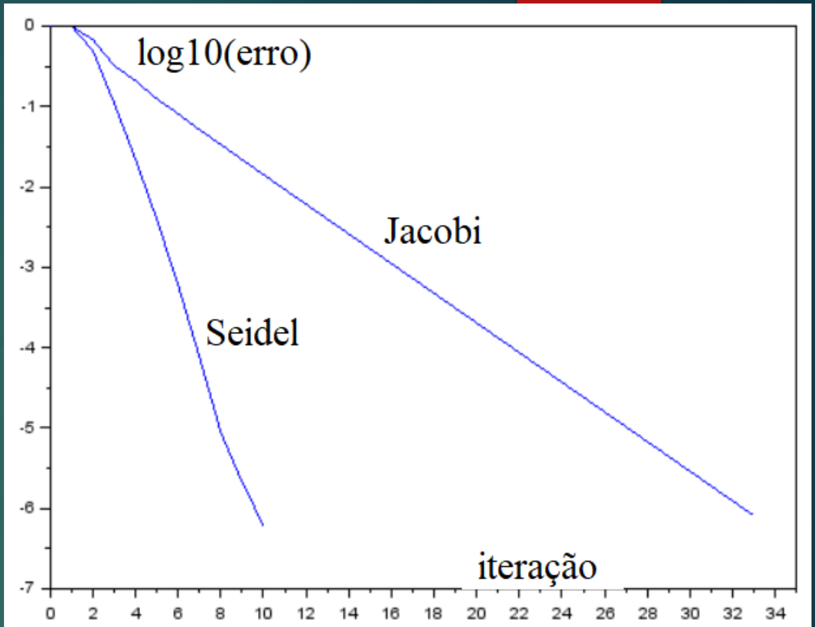
$$x_1^{k+1} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^k - a_{13}x_3^k)$$
$$x_2^{k+1} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^{k+1} - a_{23}x_3^k)$$
$$x_3^{k+1} = \frac{1}{a_{33}}(b_3 - a_{31}x_1^{k+1} - a_{32}x_2^{k+1})$$

$$x_1^{k+1} = \frac{1}{a_{11}}(b_1 - a_{12}x_2^k - a_{13}x_3^k)$$
$$x_2^{k+1} = \frac{1}{a_{22}}(b_2 - a_{21}x_1^k - a_{23}x_3^k)$$
$$x_3^{k+1} = \frac{1}{a_{33}}(b_3 - a_{31}x_1^k - a_{32}x_2^k)$$

Algoritmo para o Método Iterativo de Gauss Seidel

```
--> A=[8,2,4;1,11,3;5,3,10]
 A  =


   8.    2.     4.
   1.    11.    3.
   5.    3.     10.

--> b=[9;18;72];

--> x=inv(A)*b
 x  =


  -3.2311475
  -0.5163934
   8.9704918
```

```
--> x=GaussSeidel(A,b,[0;0;0],1e-6,%T)

   8.    2.     4.     9.
   1.    11.    3.     18.
   5.    3.     10.    72.

  "erro"
  "x(1)"
  "x(2)"
  "x(3)"

         column 1 to 6
   1.    1.           0.497754     0.1106598    0.0222884    0.0039487
   0.    1.125       -2.3471591   -3.0782929   -3.209174    -3.2288675
   0.    1.5340909    0.165031    -0.3539925   -0.4842599   -0.5109752
   0.    6.1772727    8.3240702    8.8453442    8.9498649    8.9677263
         column 7 to 11
   0.0006037    0.0000766    0.0000089    0.0000021    0.0000006
  -3.2311194   -3.2312157   -3.2311717   -3.2311535   -3.2311487
  -0.5156418   -0.5163219   -0.5163947   -0.5163963   -0.5163944
   8.9702522    8.9705044    8.9705043    8.9704956    8.9704927

após 11 iterações, erro=6.2e-07
 x  =

  -3.2311487
  -0.5163944
   8.9704927
```



O log decimal do erro em Gauss-Seidel também diminui linearmente a cada iteração, mas com um inclinação maior, isto é: muito mais rapidamente que com o método de Gauss-Jacobi.

Convergência de Gauss-Jacobi

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & | & b_1 \\ a_{21} & a_{22} & a_{23} & | & b_2 \\ a_{31} & a_{32} & a_{33} & | & b_3 \end{bmatrix} \begin{matrix} (I) \\ (II) \\ (III) \end{matrix}$$

Vamos definir a medida α_i para a linha i:

$$\alpha_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{|a_{ij}|}{a_{jj}}$$

O algoritmo de Gauss-Jacobi irá convergir se: $\max(\alpha_k) < 1$

Convergência de Gauss-Seidel

Vamos definir a medida β_i para a linha i:

$$\beta_1 = \sum_{j=2}^{N} \frac{|a_{1j}|}{a_{1j}}$$

$$\beta_i = \sum_{j=1}^{i-1} \frac{\beta_{i-1}|a_{1j}|}{a_{1j}} + \sum_{j=i+1}^{N} \frac{|a_{1j}|}{a_{1j}}$$

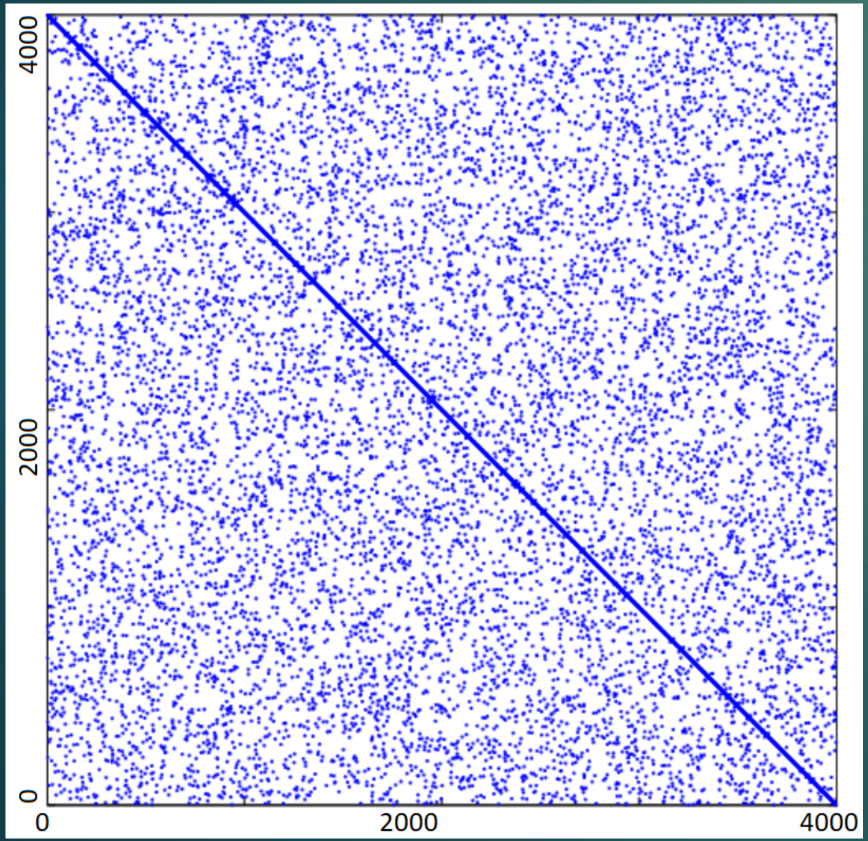
$$\beta_N = \sum_{j=1}^{N-1} \frac{\beta_{i-1}|a_{1j}|}{a_{1j}}$$

O algoritmo de Gauss-Seidel i irá convergir se: $\max(\beta_k) < 1$

Notar que se $\max(\beta_k) < 1$ então $\max(\alpha_k) < 1$, isto é: se Gauss_Seidel converge, então Gausss_Jacobi converge. A recíproca não é necessariamente verdadeira.

Comparação Métodos Diretos e Métodos Iterativos

- O métodos diretos sempre convergem, desde que a Matriz Característica do Sistema A não tenha determinante nulo.
- Os métodos iterativos dependem de um critério de convergência.
- Os métodos diretos usam todos os elementos da matriz para fazer os cálculos, de modo que os elementos nulos podem ser modificados durante os cálculos, o que é um problema com matrizes esparsas.
- No caso de matrizes esparsas os métodos iterativos são vantajosos, pois não modificam e pode não levar em conta os elementos nulos de uma matriz esparsa.



Exemplo de matriz esparsa
16 000 000 elementos (4000 linhas por 4000 colunas)
Somente 15 000 elementos não-nulos.