Resolvendo Sistemas Lineares — Métodos Iterativos

Métodos Diretos $\times$ Iterativos

Métodos Diretos

- Eliminação Gaussiana: sequência finita de operações ($\mathcal{O}(n^3)$ operações de ponto flutuante), resultando numa solução.

Métodos Diretos $\times$ Iterativos

Métodos Diretos

- Eliminação Gaussiana: sequência finita de operações ($\mathcal{O}(n^3)$ operações de ponto flutuante), resultando numa solução.
- Este tipo de método é chamado *direto*: em teoria chegam na solução exata em um número finito de passos (um computador com precisão finita chega a uma aproximação, cuja precisão está associada com o número de condicionamento).

Métodos Diretos $\times$ Iterativos

Métodos Iterativos

- Assim como no caso da busca por raízes, há *Métodos Iterativos* para resolver sistemas: começando com uma estimativa, vamos refinando a solução a cada passo (esperando que convirja...)

Métodos Diretos $\times$ Iterativos

Métodos Iterativos

- Assim como no caso da busca por raízes, há *Métodos Iterativos* para resolver sistemas: começando com uma estimativa, vamos refinando a solução a cada passo (esperando que convirja...)
- Métodos iterativos são especialmente úteis em:

Métodos Diretos × Iterativos

Métodos Iterativos

- Assim como no caso da busca por raízes, há *Métodos Iterativos* para resolver sistemas: começando com uma estimativa, vamos refinando a solução a cada passo (esperando que convirja...)
- Métodos iterativos são especialmente úteis em:
  - Problemas muito grandes (mas com A esparsa): Eliminação tende a densificar matrizes. Iterativos preservam sua estrutura.

Métodos Diretos $\times$ Iterativos

Métodos Iterativos

- Assim como no caso da busca por raízes, há *Métodos Iterativos* para resolver sistemas: começando com uma estimativa, vamos refinando a solução a cada passo (esperando que convirja...)
- Métodos iterativos são especialmente úteis em:
  - Problemas muito grandes (mas com A esparsa): Eliminação tende a densificar matrizes. Iterativos preservam sua estrutura.
  - Soluções múltiplas do mesmo problemas (mesmo A), b parecidos: convergência rápida para nova estimativa.

Método de Jacobi

- O método de Jacobi é um tipo de iteração de ponto fixo para sistemas.

Método de Jacobi

- O método de Jacobi é um tipo de iteração de ponto fixo para sistemas.
- Basicamente, isolamos a i-ésima incógnita na i-ésima equação, resolvendo iterativamente a partir de um chute inicial

Método de Jacobi

- O método de Jacobi é um tipo de iteração de ponto fixo para sistemas.
- Basicamente, isolamos a i-ésima incógnita na i-ésima equação, resolvendo iterativamente a partir de um chute inicial
- Exemplo:

$$\begin{cases} 3u + v = 5 \\ u + 2v = 5 \end{cases}$$

Método de Jacobi

- O método de Jacobi é um tipo de iteração de ponto fixo para sistemas.
- Basicamente, isolamos a i-ésima incógnita na i-ésima equação, resolvendo iterativamente a partir de um chute inicial
- Exemplo:

$$\begin{cases} 3u + v = 5 \\ u + 2v = 5 \end{cases}$$

- Começamos resolvendo a 1a. equação para u, e a 2a. para v:

$$u = \frac{5 - v}{3}$$

$$v = \frac{5 - u}{2}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-\frac{5}{2}}{3} \\ \frac{5-\frac{5}{3}}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix}$$

Métodos de Jacobi

- Basta então iterar. Seja o chute inicial $[u_0, v_0] = [0, 0]$:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-0}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{2} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-\frac{5}{2}}{3} \\ \frac{5-\frac{5}{3}}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{6} \\ \frac{5}{3} \end{bmatrix}$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-v_2}{3} \\ \frac{5-u_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-\frac{5}{3}}{3} \\ \frac{5-\frac{5}{6}}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{25}{12} \end{bmatrix}$$

- Continuado, vemos que converge para $x = [1, 2]$

Método de Jacobi

- Vamos tomar o mesmo sistema do exemplo anterior, mas invertendo as equações:

$$\begin{cases} u + 2v = 5 \\ 3u + v = 5 \end{cases}$$

Método de Jacobi

- Vamos tomar o mesmo sistema do exemplo anterior, mas invertendo as equações:

$$\begin{cases} u + 2v = 5 \\ 3u + v = 5 \end{cases}$$

- Exatamente como antes: começamos resolvendo a 1a. equação para a 1a. variável, e a 2a. equação para a 2a. variável:

$$u = 5 - 2v$$

$$v = 5 - 3u$$

Método de Jacobi

- Iterando:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Método de Jacobi

- Iterando:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

Método de Jacobi

- Iterando:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$$

Método de Jacobi

- Iterando:

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} 5 - 2v_0 \\ 5 - 3u_0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} 5 - 2v_1 \\ 5 - 3u_1 \end{bmatrix} = \begin{bmatrix} -5 \\ -10 \end{bmatrix}$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} 5 - 2(-10) \\ 5 - 3(-5) \end{bmatrix} = \begin{bmatrix} 25 \\ 20 \end{bmatrix}$$

- Neste caso (apenas invertemos as equações do exemplo anterior!), **diverge**.

Método de Jacobi: Convergência

- A convergência do método de Jacobi depende criticamente da matriz A.

Método de Jacobi: Convergência

- A convergência do método de Jacobi depende criticamente da matriz A.
- Ele só convergirá se A for **estritamente diagonal dominante**

Método de Jacobi: Convergência

- A convergência do método de Jacobi depende criticamente da matriz A.
- Ele só convergirá se A for **estritamente diagonal dominante**
- Isso siginifica que, para cada linha de A, o valor absoluto do elemento na diagonal deve ser maior do que a soma dos valores absolutos de todos os outros elementos. Ou seja, para a linha i:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

Método de Jacobi: Convergência

- A convergência do método de Jacobi depende criticamente da matriz A.
- Ele só convergirá se A for **estritamente diagonal dominante**
- Isso siginifica que, para cada linha de A, o valor absoluto do elemento na diagonal deve ser maior do que a soma dos valores absolutos de todos os outros elementos. Ou seja, para a linha i:

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

- Nos exemplos anteriores, tínhamos

$$\underbrace{\begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}}_{3>1,2>1 \implies \text{EDD}} \qquad \underbrace{\begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}}_{1<2,1<3 \implies \text{não é EDD}}$$

Método de Jacobi: Forma Matricial

- O método de Jacobi é um tipo de IFP.

Método de Jacobi: Forma Matricial

- O método de Jacobi é um tipo de IFP.
- Seja D uma matriz cuja diagonal principal é a mesma da de A, e todos os outros elementos nulos.

Método de Jacobi: Forma Matricial

- O método de Jacobi é um tipo de IFP.
- Seja D uma matriz cuja diagonal principal é a mesma da de A, e todos os outros elementos nulos.
- Seja L uma matriz com todos os elementos *abaixo* da diagonal iguais aos de A, e o resto nulo.

Método de Jacobi: Forma Matricial

- O método de Jacobi é um tipo de IFP.
- Seja D uma matriz cuja diagonal principal é a mesma da de A, e todos os outros elementos nulos.
- Seja L uma matriz com todos os elementos *abaixo* da diagonal iguais aos de A, e o resto nulo.
- Seja U uma matriz com todos os elementos *acima* da diagonal iguais aos de A, e o resto nulo.

Método de Jacobi: Forma Matricial

- O método de Jacobi é um tipo de IFP.
- Seja D uma matriz cuja diagonal principal é a mesma da de A, e todos os outros elementos nulos.
- Seja L uma matriz com todos os elementos *abaixo* da diagonal iguais aos de A, e o resto nulo.
- Seja U uma matriz com todos os elementos *acima* da diagonal iguais aos de A, e o resto nulo.
- Temos então

$$\begin{aligned} Ax &= b \\ (D + L + U)x &= b \\ Dx &= b - (L + U)x \\ x &= D^{-1}(b - (L + U)x) \end{aligned}$$

Método de Jacobi: Forma Matricial

- Temos então a seguinte IPF:

$$x_0 = \text{estimativa inicial}$$
$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}) \text{ para } n = 0, 1, 2, \ldots$$

Método de Jacobi: Forma Matricial

- Temos então a seguinte IPF:

$$x_0 = \text{estimativa inicial}$$
$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}) \text{ para } n = 0, 1, 2, \ldots$$

- Note que, como D é diagonal, sua inversa também o é, sendo cada elemento da diagonal o recíproco do correspondente em D

Método de Jacobi: Forma Matricial

- Temos então a seguinte IPF:

$$x_0 = \text{estimativa inicial}$$
$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)}) \text{ para } n = 0, 1, 2, \ldots$$

- Note que, como D é diagonal, sua inversa também o é, sendo cada elemento da diagonal o recíproco do correspondente em D
- (ao implementar o método, não é necessário inverter D)

Método de Jacobi — Elemento a elemento

- Vejamos como fica o método para cada um dos elementos do vetor x:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

Método de Jacobi — Elemento a elemento

- Vejamos como fica o método para cada um dos elementos do vetor x:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

- Cada elemento $x_i^{(k+1)}$ depende de todos os outros elementos de $x^{(k)}$, exceto dele mesmo.

Método de Jacobi — Elemento a elemento

- Vejamos como fica o método para cada um dos elementos do vetor x:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right)$$

- Cada elemento $x_i^{(k+1)}$ depende de todos os outros elementos de $x^{(k)}$, exceto dele mesmo.
- É possível então calcular todos os elementos em paralelo! Linguagens como *Python* (numpy) e *Matlab* são especialmente eficientes neste tipo de operação.

Método de Gauss-Seidel

- O método de Gauss-Seidel é bastante similar ao de Jacobi

Método de Gauss-Seidel

- O método de Gauss-Seidel é bastante similar ao de Jacobi
- A única diferença é que os valores de $x_i(k+1)$ recém calculados vão sendos usados nos elementos subsequentes.

$$\begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} \frac{5-v_0}{3} \\ \frac{5-u_1}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-0}{3} \\ \frac{5-\frac{5}{3}}{2} \end{bmatrix} = \begin{bmatrix} \frac{5}{3} \\ \frac{5}{3} \end{bmatrix}$$

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} \frac{5-v_1}{3} \\ \frac{5-u_2}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-\frac{5}{3}}{3} \\ \frac{5-\frac{10}{9}}{2} \end{bmatrix} = \begin{bmatrix} \frac{10}{9} \\ \frac{35}{18} \end{bmatrix}$$

$$\begin{bmatrix} u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} \frac{5-v_2}{3} \\ \frac{5-u_3}{2} \end{bmatrix} = \begin{bmatrix} \frac{5-\frac{35}{18}}{3} \\ \frac{5-\frac{55}{54}}{2} \end{bmatrix} = \begin{bmatrix} \frac{55}{54} \\ \frac{215}{108} \end{bmatrix}$$

Método de Gauss-Seidel

- Gauss Seidel costuma convergir mais rápido do que Jacobi

Método de Gauss-Seidel

- Gauss Seidel costuma convergir mais rápido do que Jacobi
- Também tem convergência garantida se A for estritamente diagonal dominante

Método de Gauss-Seidel

- Gauss Seidel costuma convergir mais rápido do que Jacobi
- Também tem convergência garantida se A for estritamente diagonal dominante
- Para a forma matricial, isolamos $(L + D + U)x = b$ desta forma:

$$(L + D)x^{(k+1)} = -Ux^{(k)} + b$$

Método de Gauss-Seidel

- Gauss Seidel costuma convergir mais rápido do que Jacobi
- Também tem convergência garantida se A for estritamente diagonal dominante
- Para a forma matricial, isolamos $(L + D + U)x = b$ desta forma:

$$(L + D)x^{(k+1)} = -Ux^{(k)} + b$$

- Colocando a matriz triangular inferior L do lado direito, fazemos com que os elementos recém calculados $x^{(k+1)}$ sejam usados na estimativa dos próximos.

Método de Gauss-Seidel

- Da expressão anterior, obtemos a IPF:

$x_0 =$ estimativa inicial

$x^{(k+1)} = D^{-1}(b - Ux^{(k)} - Lx^{(k+1)})$ para $n = 0, 1, 2, \ldots$

Método de Gauss-Seidel

- Da expressão anterior, obtemos a IPF:

  $x_0 =$ estimativa inicial

  $x^{(k+1)} = D^{-1}(b - Ux^{(k)} - Lx^{(k+1)})$ para $n = 0, 1, 2, \ldots$

- Ou elemento a elemento (n é o número de equações):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} \right)$$

Método de Gauss-Seidel

- Da expressão anterior, obtemos a IPF:

  $x_0 =$ estimativa inicial

  $x^{(k+1)} = D^{-1}(b - Ux^{(k)} - Lx^{(k+1)})$ para $n = 0, 1, 2, \ldots$

- Ou elemento a elemento (n é o número de equações):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} \right)$$

- Ao contrário de Jacobi, não é possível calcular GS em paralelo.

Método de Gauss-Seidel

- Da expressão anterior, obtemos a IPF:

  $x_0 =$ estimativa inicial

  $x^{(k+1)} = D^{-1}(b - Ux^{(k)} - Lx^{(k+1)})$ para $n = 0, 1, 2, \ldots$

- Ou elemento a elemento (n é o número de equações):

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} \right)$$

- Ao contrário de Jacobi, não é possível calcular GS em paralelo.
- Mas GS permite sobrescrever os elementos de $x_i^{(k)}$, economizando memória.

Método de Gauss-Seidel — Exemplo

- Vamos resolver o sistema:

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

Método de Gauss-Seidel — Exemplo

- Vamos resolver o sistema:

$$\begin{bmatrix} 3 & 1 & -1 \\ 2 & 4 & 1 \\ -1 & 2 & 5 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 1 \end{bmatrix}$$

- A iteração de GS fica

$$u_{k+1} = \frac{4 - v_k + w_k}{3}$$

$$v_{k+1} = \frac{1 - 2u_{k+1} - w_k}{4}$$

$$w_{k+1} = \frac{1 + u_{k+1} - 2v_{k+1}}{5}$$

Método de Gauss-Seidel — Exemplo

- Começando de $[u_0, v_0, w_0] = [0, 0, 0]$ temos

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 4 - 0 - 0 = \frac{4}{3} \\ \frac{1 - \frac{8}{3} - 0}{4} = -\frac{5}{12} \\ \frac{1 + \frac{4}{3} + \frac{5}{6}}{5} = -\frac{19}{30} \end{bmatrix} \approx \begin{bmatrix} 1.3333 \\ -0.4167 \\ 0.6333 \end{bmatrix}$$

Método de Gauss-Seidel — Exemplo

- Começando de $[u_0, v_0, w_0] = [0, 0, 0]$ temos

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 4 - 0 - 0 = \frac{4}{3} \\ \frac{1 - \frac{8}{3} - 0}{4} = -\frac{5}{12} \\ \frac{1 + \frac{4}{3} + \frac{5}{6}}{5} = -\frac{19}{30} \end{bmatrix} \approx \begin{bmatrix} 1.3333 \\ -0.4167 \\ 0.6333 \end{bmatrix}$$

- Iterando mais uma vez

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{101}{60} \\ -\frac{3}{4} \\ -\frac{251}{300} \end{bmatrix} \approx \begin{bmatrix} 1.6822 \\ -0.7500 \\ 0.8367 \end{bmatrix}$$

Método de Gauss-Seidel — Exemplo

- Começando de $[u_0, v_0, w_0] = [0, 0, 0]$ temos

$$\begin{bmatrix} u_1 \\ v_1 \\ w_1 \end{bmatrix} = \begin{bmatrix} 4 - 0 - 0 = \frac{4}{3} \\ \frac{1 - \frac{8}{3} - 0}{4} = -\frac{5}{12} \\ \frac{1 + \frac{4}{3} + \frac{5}{6}}{5} = -\frac{19}{30} \end{bmatrix} \approx \begin{bmatrix} 1.3333 \\ -0.4167 \\ 0.6333 \end{bmatrix}$$

- Iterando mais uma vez

$$\begin{bmatrix} u_2 \\ v_2 \\ w_2 \end{bmatrix} = \begin{bmatrix} \frac{101}{60} \\ -\frac{3}{4} \\ -\frac{251}{300} \end{bmatrix} \approx \begin{bmatrix} 1.6822 \\ -0.7500 \\ 0.8367 \end{bmatrix}$$

- A matriz é estritamente diagonal dominante, logo o método converge para a solução $[2, -1, 1]$

Métodos Iterativos: Tamanho do Problema

Tempo

- Eliminação Gaussiana para um sistema de n equações requer da ordem de n^3 operações de ponto flutuante.

Métodos Iterativos: Tamanho do Problema

Tempo

- Eliminação Gaussiana para um sistema de n equações requer da ordem de n^3 operações de ponto flutuante.
- Consideremos um problema de tamanho $n \approx 10^6$

Métodos Iterativos: Tamanho do Problema

Tempo

- Eliminação Gaussiana para um sistema de n equações requer da ordem de n^3 operações de ponto flutuante.
- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$.

Métodos Iterativos: Tamanho do Problema

Tempo

- Eliminação Gaussiana para um sistema de n equações requer da ordem de n^3 operações de ponto flutuante.
- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$.
- O processador de um computador moderno "típico" opera em GHz (10^9 ciclos por segundo)

Métodos Iterativos: Tamanho do Problema

Tempo

- Eliminação Gaussiana para um sistema de n equações requer da ordem de n^3 operações de ponto flutuante.
- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$.
- O processador de um computador moderno "típico" opera em GHz (10^9 ciclos por segundo)
  - $\approx 10^8$ operações de ponto flutuante por segundo (FLOPS).

Métodos Iterativos: Tamanho do Problema

Tempo

- Eliminação Gaussiana para um sistema de n equações requer da ordem de n^3 operações de ponto flutuante.
- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$.
- O processador de um computador moderno "típico" opera em GHz (10^9 ciclos por segundo)
  - $\approx 10^8$ operações de ponto flutuante por segundo (FLOPS).
- Nosso computador levaria da ordem de $\frac{(10^6)^3}{10^8} = 10^{10}$ segundos ($\approx$ 3 anos) para resolver o problema.

Métodos Iterativos: Tamanho do Problema

Armazenamento

- Consideremos um problema de tamanho $n \approx 10^6$

Métodos Iterativos: Tamanho do Problema

Armazenamento

- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$).

Métodos Iterativos: Tamanho do Problema

Armazenamento

- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$).
- Além disso, supondo que armazenamos cada entrada da matriz em um *double* (8 bytes)

Métodos Iterativos: Tamanho do Problema

Armazenamento

- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$).
- Além disso, supondo que armazenamos cada entrada da matriz em um *double* (8 bytes)
  - a matriz A tem da ordem de n^2 entradas

Métodos Iterativos: Tamanho do Problema

Armazenamento

- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$).
- Além disso, supondo que armazenamos cada entrada da matriz em um *double* (8 bytes)
  - a matriz A tem da ordem de n^2 entradas
  - precisaremos de $(10^6)^2 \times 8 = 8\,\text{TB}$ de memória RAM

Métodos Iterativos: Tamanho do Problema

Armazenamento

- Consideremos um problema de tamanho $n \approx 10^6$
  - EDP numa grade tridimensional $100 \times 100 \times 100$).
- Além disso, supondo que armazenamos cada entrada da matriz em um *double* (8 bytes)
  - a matriz A tem da ordem de n^2 entradas
  - precisaremos de $(10^6)^2 \times 8 = 8\,\text{TB}$ de memória RAM
  - um computador típico tem da ordem de 10 GB de RAM, ou seja, 800 vezes *menos*.

Métodos Iterativos: Tamanho do Problema

- Mas esperem! Muitos problemas práticos (por exemplo, discretização de EDPs) produzem matrizes *esparsas* (com muitos elementos nulos). (Matrizes que não são esparsas são chamadas de *densas*).

Métodos Iterativos: Tamanho do Problema

- Mas esperem! Muitos problemas práticos (por exemplo, discretização de EDPs) produzem matrizes *esparsas* (com muitos elementos nulos). (Matrizes que não são esparsas são chamadas de *densas*).
- Várias linguagens de programação têm estruturas de dados específicas para armazenar matrizes esparsas. A grosso modo, armazenam apenas os elementos não nulo e suas posições.

Métodos Iterativos: Tamanho do Problema

- Mas esperem! Muitos problemas práticos (por exemplo, discretização de EDPs) produzem matrizes *esparsas* (com muitos elementos nulos). (Matrizes que não são esparsas são chamadas de *densas*).
- Várias linguagens de programação têm estruturas de dados específicas para armazenar matrizes esparsas. A grosso modo, armazenam apenas os elementos não nulo e suas posições.
- Será que podemos nos aproveitar disso ao resolver sistemas?

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).
  - cada *double* tem 8 bytes, os dois inteiros perfazem 1 byte

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).
  - cada *double* tem 8 bytes, os dois inteiros perfazem 1 byte
  - Para $n = 10^6$, temos $4 \times 10^6 \cdot (8 + 1) = 36\,\text{MB}$

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).
  - cada *double* tem 8 bytes, os dois inteiros perfazem 1 byte
  - Para $n = 10^6$, temos $4 \times 10^6 \cdot (8 + 1) = 36\,\text{MB}$
  - na verdade, implementações reais são ainda mais eficientes!

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).
  - cada *double* tem 8 bytes, os dois inteiros perfazem 1 byte
  - Para $n = 10^6$, temos $4 \times 10^6 \cdot (8 + 1) = 36\,\text{MB}$
  - na verdade, implementações reais são ainda mais eficientes!
- Infelizmente, Eliminação Gaussiana tende a "densificar" algumas matrizes, mesmo que sejam esparsas (pense em LU). Desta forma, não podemos utilizar estruturas esparsas.

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).
  - cada *double* tem 8 bytes, os dois inteiros perfazem 1 byte
  - Para $n = 10^6$, temos $4 \times 10^6 \cdot (8 + 1) = 36\,\text{MB}$
  - na verdade, implementações reais são ainda mais eficientes!
- Infelizmente, Eliminação Gaussiana tende a "densificar" algumas matrizes, mesmo que sejam esparsas (pense em LU). Desta forma, não podemos utilizar estruturas esparsas.
- Mas métodos iterativos podem!

Métodos Iterativos: Tamanho do Problema

- Exemplo: discretização da Eq. de Laplace (segundo trabalho)
  - matriz com aproximadamente 4 elementos não nulos por linha.
  - ao invés de n^2 números, precisamos de $\approx 4n$ (mais dois inteiros para cada índice).
  - cada *double* tem 8 bytes, os dois inteiros perfazem 1 byte
  - Para $n = 10^6$, temos $4 \times 10^6 \cdot (8 + 1) = 36\,\text{MB}$
  - na verdade, implementações reais são ainda mais eficientes!
- Infelizmente, Eliminação Gaussiana tende a "densificar" algumas matrizes, mesmo que sejam esparsas (pense em LU). Desta forma, não podemos utilizar estruturas esparsas.
- Mas métodos iterativos podem!
- Inverter uma matriz esparsa costuma produzir uma matriz densa!