Como Resolver Sistemas Lineares?

Prof. Americo Cunha

Universidade do Estado do Rio de Janeiro – UERJ

americo.cunha@uerj.br

www.americocunha.org

@AmericoCunhaJr

@AmericoCunhaJr @AmericoCunhaJr

Possíveis abordagens para sistemas lineares

$$A\mathbf{x} = \mathbf{b}$$

Métodos Diretos: uma solução do sistema linear é obtida após um número finito de etapas, onde cada passo envolve operações com os elementos de A e $\mathbf{b}$, i.e.,

$$\mathbf{x} = T\,(A, \mathbf{b})$$

Métodos Iterativos: uma solução do sistema linear é obtida após um número infinito de etapas, como o limite de uma sequência de aproximações que envolve operações com os elementos de A e $\mathbf{b}$, i.e.,

$$\mathbf{x} = \lim_{n \to \infty} \mathbf{x}_n\,, \qquad \text{onde} \qquad \mathbf{x}_n = T_n\,(A, \mathbf{b})$$

Métodos diretos

- Em *teoria* calculam uma *solução exata*;
- Na *prática* calculam uma *solução aproximada* devido aos erros da aritmética de ponto flutuante;
- Em geral, são *mais precisos* que os métodos iterativos.

Exemplos de métodos diretos:

- Eliminação gaussiana
- Fatoração LU
- Fatoração Cholesky
- Fatoração QR
- Fatoração SVD
- Regra de Cramer
- Gradiente Conjugado
- etc

Métodos iterativos

- Em *teoria* calculam uma *solução exata* no limite;
- Na *prática* a iteração é interrompida após um número finito de etapas, de modo que se obtém uma *solução aproximada*;
- A solução aproximada está sujeita a *duas fontes de erro*:
  - ponto flutuante;
  - parada do processo iterativo;
- Em geral, são *mais rápidos* que os métodos diretos.

Exemplos de métodos iterativos:

- Método de Jacobi
- Método de Gauss-Siedel
- Sobre-Relaxação Sucessiva
- Gradiente Conjugado
- Gradiente Bi-conjugado
- GMRES
- etc

ATENÇÃO: métodos iterativos só devem ser utilizados quando métodos diretos não forem viáveis!

Podemos usar qualquer método numérico para resolver um sistema linear?

Podemos usar qualquer método numérico para resolver um sistema linear?

Veremos a seguir que não!

A regra de Cramer

$$\underbrace{\begin{bmatrix} | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_n \\ | & & | \end{bmatrix}}_{A} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} | \\ \mathbf{b} \\ | \end{bmatrix}$$

Os *componentes do vetor solução* são dados por

$$x_1 = \frac{\det A_1}{\det A}, \ \cdots, \ x_j = \frac{\det A_j}{\det A}, \ \cdots, \ x_n = \frac{\det A_n}{\det A}$$

$$A_j = \begin{bmatrix} | & & | & | & | & & | \\ \mathbf{a}_1 & \cdots & \mathbf{a}_{j-1} & \mathbf{b} & \mathbf{a}_{j+1} & \cdots & \mathbf{a}_n \\ | & & | & | & | & & | \end{bmatrix} \ j = 1, \cdots, n$$

Aplicável quando o sistema for não singular, i.e. $\det A \neq 0$**.**

Exemplo de aplicação da regra de Cramer

$$\begin{bmatrix} 1 & 3 & -2 \\ 3 & 5 & 6 \\ 2 & 4 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 7 \\ 8 \end{bmatrix}$$

$$x_1 = \frac{\det \begin{pmatrix} \color{red}{5} & 3 & -2 \\ \color{red}{7} & 5 & 6 \\ \color{red}{8} & 4 & 3 \end{pmatrix}}{\det \begin{pmatrix} 1 & 3 & -2 \\ 3 & 5 & 6 \\ 2 & 4 & 3 \end{pmatrix}} = \frac{+60}{-4} = -15$$

$$x_2 = \frac{\det \begin{pmatrix} 1 & \color{red}{5} & -2 \\ 3 & \color{red}{7} & 6 \\ 2 & \color{red}{8} & 3 \end{pmatrix}}{\det \begin{pmatrix} 1 & 3 & -2 \\ 3 & 5 & 6 \\ 2 & 4 & 3 \end{pmatrix}} = \frac{-32}{-4} = 8$$

$$x_3 = \frac{\det \begin{pmatrix} 1 & 3 & \color{red}{5} \\ 3 & 5 & \color{red}{7} \\ 2 & 4 & \color{red}{8} \end{pmatrix}}{\det \begin{pmatrix} 1 & 3 & -2 \\ 3 & 5 & 6 \\ 2 & 4 & 3 \end{pmatrix}} = \frac{-8}{-4} = 2$$

Algoritmo da regra de Cramer

Input: A, $\mathbf{b}$

Compute the length of $\mathbf{b}$
Allocate memory for $\mathbf{x}$
Compute the matrix determinant $\det A$
if $\det A = 0$ **then**
    return
end if
for j=1:n **do**
    Construct the modified matrix A_j
    Compute the solution j-th component $x_j = \det A_j / \det A$
end for
return

Output: $\mathbf{x}$

Implementação em GNU Octave

```
function x = cramer(A,b)
    n    = length(b);
    x    = zeros(n,1);
    detA = myDet(A);
    if detA == 0.0
        error('matriz singular')
    end
    for j = 1:n
        Aj      = A;
        Aj(:,j) = b;
        x(j)    = myDet(Aj)/detA;
    end
end
```

Implementação em GNU Octave

```
function detA = myDet(A)
    if isscalar(A)
        detA = A;
        return
    end
    detA   = 0.0;
    toprow = A(1,:);
    A(1,:) = [];
    for i = 1:size(A,2)
        Ai      = A;
        Ai(:,i) = [];
        detA    = detA +(-1)^(i+1)*toprow(i)*myDet(Ai);
    end
end
```

Experimento Computacional 1

$$\begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \implies \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

```
>> n = 5; A = eye(n,n); b = ones(n,1);
>> tic; x = cramer(A,b); toc
>> disp(x)
```

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\,(\textbf{Cramer}) = (n+1) \times \texttt{flops}\,(\text{det}\ (\texttt{n} \times \texttt{n})) + n$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\left(\textbf{Cramer}\right) = (n+1) \times \texttt{flops}\left(\text{det}\ \texttt{(n} \times \texttt{n)}\right) + n$$

$$\texttt{flops}\left(\text{det}\ \texttt{(n} \times \texttt{n)}\right) = n \times \texttt{flops}\left(\text{det}\ \texttt{(n-1} \times \texttt{n-1)}\right) + 2n - 1$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\left(\textbf{Cramer}\right) = (n+1) \times \texttt{flops}\left(\text{det}\ (\texttt{n} \times \texttt{n})\right) + n$$

$$\texttt{flops}\left(\text{det}\ (\texttt{n} \times \texttt{n})\right) = n \times \texttt{flops}\left(\text{det}\ (\texttt{n-1} \times \texttt{n-1})\right) + 2n - 1$$

$$\texttt{flops}\left(\text{det}\ (\texttt{n} \times \texttt{n})\right) \sim n \times \texttt{flops}\left(\text{det}\ (\texttt{n-1} \times \texttt{n-1})\right)$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\,(\textbf{Cramer}) = (n+1) \times \texttt{flops}\,(\text{det}\,(\texttt{n} \times \texttt{n})) + n$$

$$\texttt{flops}\,(\text{det}\,(\texttt{n} \times \texttt{n})) = n \times \texttt{flops}\,(\text{det}\,(\texttt{n-1} \times \texttt{n-1})) + 2n - 1$$

$$\begin{aligned} \texttt{flops}\,(\text{det}\,(\texttt{n} \times \texttt{n})) &\sim n \times \texttt{flops}\,(\text{det}\,(\texttt{n-1} \times \texttt{n-1})) \\ &\sim n \times (n-1) \times \texttt{flops}\,(\text{det}\,(\texttt{n-2} \times \texttt{n-2})) \end{aligned}$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\,(\textbf{Cramer}) = (n+1) \times \texttt{flops}\,(\text{det}\,(\texttt{n} \;\times\; \texttt{n})) + n$$

$$\texttt{flops}\,(\text{det}\,(\texttt{n} \;\times\; \texttt{n})) = n \times \texttt{flops}\,(\text{det}\,(\texttt{n-1} \;\times\; \texttt{n-1})) + 2n - 1$$

$$\begin{aligned}\texttt{flops}\,(\text{det}\,(\texttt{n} \;\times\; \texttt{n})) &\sim n \times \texttt{flops}\,(\text{det}\,(\texttt{n-1} \;\times\; \texttt{n-1})) \\ &\sim n \times (n-1) \times \texttt{flops}\,(\text{det}\,(\texttt{n-2} \;\times\; \texttt{n-2})) \\ &\vdots\end{aligned}$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\,(\textbf{Cramer}) = (n+1) \times \texttt{flops}\,(\text{det}\,(\texttt{n} \times \texttt{n})) + n$$

$$\texttt{flops}\,(\text{det}\,(\texttt{n} \times \texttt{n})) = n \times \texttt{flops}\,(\text{det}\,(\texttt{n-1} \times \texttt{n-1})) + 2n - 1$$

$$\begin{aligned}
\texttt{flops}\,(\text{det}\,(\texttt{n} \times \texttt{n})) &\sim n \times \texttt{flops}\,(\text{det}\,(\texttt{n-1} \times \texttt{n-1})) \\
&\sim n \times (n-1) \times \texttt{flops}\,(\text{det}\,(\texttt{n-2} \times \texttt{n-2})) \\
&\qquad \vdots \\
&\sim n \times (n-1) \times \cdots \times 2 \times 1
\end{aligned}$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\left(\textbf{Cramer}\right) = (n+1) \times \texttt{flops}\left(\text{det}\,(\texttt{n} \times \texttt{n})\right) + n$$

$$\texttt{flops}\left(\text{det}\,(\texttt{n} \times \texttt{n})\right) = n \times \texttt{flops}\left(\text{det}\,(\texttt{n-1} \times \texttt{n-1})\right) + 2n - 1$$

$$\begin{aligned}
\texttt{flops}\left(\text{det}\,(\texttt{n} \times \texttt{n})\right) &\sim n \times \texttt{flops}\left(\text{det}\,(\texttt{n-1} \times \texttt{n-1})\right)\\
&\sim n \times (n-1) \times \texttt{flops}\left(\text{det}\,(\texttt{n-2} \times \texttt{n-2})\right)\\
&\qquad\vdots\\
&\sim n \times (n-1) \times \cdots \times 2 \times 1\\
&\sim n\,!
\end{aligned}$$

Qual o custo computacional da regra de Cramer?

Uma medida de *custo computacional* da execução de um algoritmo é dada pelo número de *operações de ponto flutuante* (`flops`).

$$\texttt{flops}\left(\textbf{Cramer}\right) = (n+1) \times \texttt{flops}\left(\text{det}\ (\texttt{n} \times \texttt{n})\right) + n$$

$$\texttt{flops}\left(\text{det}\ (\texttt{n} \times \texttt{n})\right) = n \times \texttt{flops}\left(\text{det}\ (\texttt{n-1} \times \texttt{n-1})\right) + 2n - 1$$

$$\begin{aligned}
\texttt{flops}\left(\text{det}\ (\texttt{n} \times \texttt{n})\right) &\sim n \times \texttt{flops}\left(\text{det}\ (\texttt{n-1} \times \texttt{n-1})\right) \\
&\sim n \times (n-1) \times \texttt{flops}\left(\text{det}\ (\texttt{n-2} \times \texttt{n-2})\right) \\
&\quad \vdots \\
&\sim n \times (n-1) \times \cdots \times 2 \times 1 \\
&\sim n\,!
\end{aligned}$$

$$\boxed{\texttt{flops}\left(\textbf{Cramer}\right) \sim e\,(n+1)\,!}$$

Tempo de processamento da regra de Cramer

Intel Core i7 em 2011: 12×10^9 flops/sec
Intel Core i7 em 2021: 52×10^9 flops/sec

n	flops	Tempo de CPU 2011	Tempo de CPU 2021
5	2×10^3	166 ns	38 ns
10	108×10^6	9 ms	2 ms
15	56×10^{12}	79 min	18 min
20	138×10^{18}	367 anos	85 anos

https://en.wikipedia.org/wiki/FLOPS https://www.cpubenchmark.net

Tempo de processamento da regra de Cramer

Intel Core i7 em 2011: 12×10^9 flops/sec
Intel Core i7 em 2021: 52×10^9 flops/sec

n	flops	Tempo de CPU	
		2011	2021
5	2×10^3	166 ns	38 ns
10	108×10^6	9 ms	2 ms
15	56×10^{12}	79 min	18 min
20	138×10^{18}	367 anos	85 anos

Não é viável usar a regra de Cramer em sistemas lineares do mundo real!

https://en.wikipedia.org/wiki/FLOPS https://www.cpubenchmark.net

Experimento Computacional 2

$$\begin{bmatrix} 1+\varepsilon & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} (1+\varepsilon)^2+1 \\ -2-\varepsilon \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ -1 \end{bmatrix}$$

```
>>  A = [1+eps -1; -1 1]
>>  b = A*[1+eps; -1]
>>  x = cramer(A,b)
```

Experimento Computacional 2

$$\begin{bmatrix} 1+\varepsilon & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} (1+\varepsilon)^2+1 \\ -2-\varepsilon \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ -1 \end{bmatrix}$$

```
>> A = [1+eps -1; -1 1]
>> b = A*[1+eps; -1]
>> x = cramer(A,b)
```

```
x =

   2
   0
```

Experimento Computacional 2

$$\begin{bmatrix} 1+\varepsilon & -1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} (1+\varepsilon)^2+1 \\ -2-\varepsilon \end{bmatrix} \longrightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ -1 \end{bmatrix}$$

```
>> A = [1+eps -1; -1 1]
>> b = A*[1+eps; -1]
>> x = cramer(A,b)
```

```
x =

   2
   0
```

Além de ser muito cara computacionalmente, a regra de Cramer é instável!

Fatos sobre sistemas lineares

1. A escolha do método numérico tem grande influência na acurácia e eficiência computacional do cálculo da solução de um sistema linear;

2. O método numérico a ser utilizado na solução do sistema linear deve ser escolhido com sabedoria!

Como citar esse material?

A. Cunha, *Como Resolver Sistemas Lineares?*,
Universidade do Estado do Rio de Janeiro – UERJ, 2021.

@AmericoCunhaJr

@AmericoCunhaJr @AmericoCunhaJr