Operações com matrizes no computador

Francisco N. C. Sobral   Júlia Demori Guizardi
Universidade Estadual de Maringá

XXXII Semana da Matemática
UEM, 2022

Por que reinventar a roda?

- Entender o funcionamento da linguagem
- Entender o funcionamento da memória
- Adquirir conhecimento mais profundo de programação
- Em Julia: `A * B` e `A * v` calculam eficientemente os produtos matriz-matriz e matriz-vetor

Tópicos

1. Notação
2. Vetores, matrizes e memória em Julia
3. Multiplicação de matrizes e vetores
4. Multiplicação de matrizes
5. Problema prático: matrizes esparsas

Table of Contents

Notação

- Vetor

$$v \in \mathbb{R}^n \Rightarrow v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix}$$

- Vetor transposto

$$v^T = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$$

- Produto escalar de vetores x e y

$$x^T y = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i$$

Notação

- Matriz

$$A \in \mathbb{R}^{m\times p} \Rightarrow A = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ a_{21} & \cdots & a_{2p} \\ & \vdots & \\ a_{m1} & \cdots & a_{mp} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} A^1 & \cdots & A^p \end{bmatrix}$$

- i-ésima linha de A

$$A_i = \begin{bmatrix} a_{i1} & \cdots & a_{ip} \end{bmatrix}$$

- j-ésima coluna de A

$$A^j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

Notação

- Matriz

$$A \in \mathbb{R}^{m \times p} \Rightarrow A = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ a_{21} & \cdots & a_{2p} \\ & \vdots & \\ a_{m1} & \cdots & a_{mp} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} A^1 & \cdots & A^p \end{bmatrix}$$

- i-ésima linha de A

$$A_i = \begin{bmatrix} a_{i1} & \cdots & a_{ip} \end{bmatrix}$$

- j-ésima coluna de A

$$A^j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

Notação

- Matriz

$$A \in \mathbb{R}^{m\times p} \Rightarrow A = \begin{bmatrix} a_{11} & \cdots & a_{1p} \\ a_{21} & \cdots & a_{2p} \\ & \vdots & \\ a_{m1} & \cdots & a_{mp} \end{bmatrix} = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{bmatrix} = \begin{bmatrix} A^1 & \cdots & A^p \end{bmatrix}$$

- i-ésima linha de A

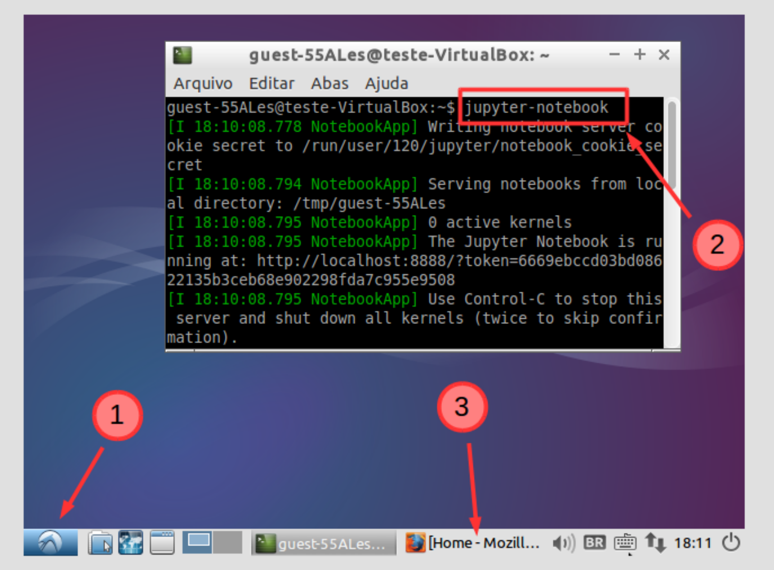
$$A_i = \begin{bmatrix} a_{i1} & \cdots & a_{ip} \end{bmatrix}$$

- j-ésima coluna de A

$$A^j = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}$$

Table of Contents

Abrindo o *notebook*



Vetores

- Comando `Vector{Float64}(undef, n)`
- Reserva um espaço na memória de tamanho n·`sizeof(Float64)` e retorna o seu **endereço**
- Checamos o endereço com `pointer`

- `zeros(n)`: vetor de zeros
- `ones(n)`: vetor de uns
- `rand(n)`: vetor aleatório com números entre 0 e 1

- i-ésima posição de um vetor v: `v[i]`
  - `v[2] = 10`: atribuição
  - `v[2] + 10`: conteúdo

Matrizes

- Comando `Array{Float64}(undef, m, n)`
- Reserva um espaço na memória de tamanho mn·`sizeof(Float64)` e retorna o seu **endereço**
- Checamos o endereço com `pointer`

- `zeros(m, n)`: matriz de zeros
- `ones(m, n)`: matriz de uns
- `rand(m, n)`: matriz aleatória com números entre 0 e 1

- Elemento (i, j) da matriz M: `M[i, j]`
  - `M[3, 2] = 10`: atribuição
  - `M[3, 2] + 10`: conteúdo

Percorrendo vetores e matrizes

- Utilizamos laços do tipo `for` para percorrer um vetor

```
s = 0.0
for i = 1:n
    s = s + v[i]
end
```

- Para percorrer uma matriz por inteiro, precisamos de dois laços

```
s = 0.0
for j = 1:n
    # Aqui dentro o j está fixo
    for i = 1:m
        # Aqui dentro o i está fixo
        # j também
        s = s + M[i, j]
    end
end
```

Memória

- Ao criarmos um vetor, um espaço da memória do computador é reservado
- Quando o vetor não é mais usado, o GC (*Garbage Colector*) entra em ação
- Criação $>$ Destruição pelo GC $\Rightarrow$ Travamento
- Ordem de velocidade de acesso

  HD $<$ Memória RAM $<$ *Cache* do processador
- Ordem de espaço disponível

  HD (TB) $>$ Memória RAM (GB) $>$ *Cache* do processador (MB)

Memória

- Matriz 1000×1000 do tipo `Float64` ocupa

$$\frac{1000 \times 1000 \times 8}{1024} \approx 8\text{MB}$$

- Matriz 10000×10000: 800MB

Acessando posições próximas

Ao acessar a posição i de um vetor ou (i, j) de uma matriz o processador também acessa posições vizinhas com maior velocidade!

- Uso do *cache*
- Se acessou `v[5]`, então `v[4]` e `v[6]` são acessadas mais rapidamente que `v[20]`
- Se acessou `M[3, 3]`, então `M[2, 3]` e `M[4, 3]` são acessados mais rapidamente que `M[3, 4]`

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 4 & 7 & 2 & 5 & 9 & 3 & 6 & 9 \end{bmatrix}$$

- Julia é Orientada a colunas

Table of Contents

Duas interpretações

Sejam $A \in \mathbb{R}^{m\times n}$ e $v \in \mathbb{R}^n$. Existem duas formas de interpretarmos $u = Av$:

- Produto escalar com as **linhas** de A:

$$u = \begin{bmatrix} a_{11}v_1 + \cdots + a_{1n}v_n \\ a_{21}v_1 + \cdots + a_{2n}v_n \\ \vdots \\ a_{m1}v_1 + \cdots + a_{mn}v_n \end{bmatrix} = \begin{bmatrix} A_1 \cdot v \\ \vdots \\ A_m \cdot v \end{bmatrix}$$

- Combinação linear das **colunas** de A:

$$u = \begin{bmatrix} a_{11}v_1 + \cdots + a_{1n}v_n \\ a_{21}v_1 + \cdots + a_{2n}v_n \\ \vdots \\ a_{m1}v_1 + \cdots + a_{mn}v_n \end{bmatrix} = A^1 v_1 + \cdots + A^n v_n$$

Duas interpretações

Sejam $A \in \mathbb{R}^{m\times n}$ e $v \in \mathbb{R}^n$. Existem duas formas de interpretarmos $u = Av$:

- Produto escalar com as **linhas** de A:

$$u = \begin{bmatrix} a_{11}v_1 + \cdots + a_{1n}v_n \\ a_{21}v_1 + \cdots + a_{2n}v_n \\ \vdots \\ a_{m1}v_1 + \cdots + a_{mn}v_n \end{bmatrix} = \begin{bmatrix} A_1 \cdot v \\ \vdots \\ A_m \cdot v \end{bmatrix}$$

- Combinação linear das **colunas** de A:

$$u = \begin{bmatrix} a_{11}v_1 + \cdots + a_{1n}v_n \\ a_{21}v_1 + \cdots + a_{2n}v_n \\ \vdots \\ a_{m1}v_1 + \cdots + a_{mn}v_n \end{bmatrix} = A^1 v_1 + \cdots + A^n v_n$$

Duas interpretações

- Produto escalar

$$u_i = a_{i1} v_1 + \cdots + a_{in} v_n$$

Os elementos de A não estão próximos uns dos outros!

- Combinação linear

$$u_1 = u_1 + a_{1j} v_j$$

$$\vdots$$

$$u_m = u_m + a_{mj} v_j$$

Os elementos de A estão próximos uns dos outros!

Table of Contents

Três interpretações

Sejam $A \in \mathbb{R}^{m\times p}$ e $B \in \mathbb{R}^{p\times n}$. Existem três formas de interpretarmos $C = A \cdot B$:

- Produto escalar

$$C = \begin{bmatrix} a_{11}b_{11} + \cdots + a_{1p}b_{p1} & \cdots & a_{11}b_{1n} + \cdots + a_{1p}b_{pn} \\ & \vdots & \\ a_{m1}b_{11} + \cdots + a_{mp}b_{p1} & \cdots & a_{m1}b_{1n} + \cdots + a_{mp}b_{pn} \end{bmatrix}$$
$$= \begin{bmatrix} A_1B^1 & \cdots & A_1B^n \\ & \vdots & \\ A_mB^1 & \cdots & A_mB^n \end{bmatrix}$$

Sejam $A \in \mathbb{R}^{m \times p}$ e $B \in \mathbb{R}^{p \times n}$. Existem três formas de interpretarmos $C = A \cdot B$:

- Combinação linear de linhas ou colunas

$$
\begin{aligned}
C &= \begin{bmatrix} a_{11}b_{11} + \cdots + a_{1p}b_{p1} & \cdots & a_{11}b_{1n} + \cdots + a_{1p}b_{pn} \\ & \vdots & \\ a_{m1}b_{11} + \cdots + a_{mp}b_{p1} & \cdots & a_{m1}b_{1n} + \cdots + a_{mp}b_{pn} \end{bmatrix} \\
&= \begin{bmatrix} A_1 B \\ \vdots \\ A_m B \end{bmatrix} = \begin{bmatrix} AB^1 & \cdots & AB^n \end{bmatrix}
\end{aligned}
$$

Três interpretações

Sejam $A \in \mathbb{R}^{m\times p}$ e $B \in \mathbb{R}^{p\times n}$. Existem três formas de interpretarmos $C = A \cdot B$:

- Combinação linear de linhas ou colunas

$$C = \begin{bmatrix} a_{11}b_{11} + \cdots + a_{1p}b_{p1} & \cdots & a_{11}b_{1n} + \cdots + a_{1p}b_{pn} \\ & \vdots & \\ a_{m1}b_{11} + \cdots + a_{mp}b_{p1} & \cdots & a_{m1}b_{1n} + \cdots + a_{mp}b_{pn} \end{bmatrix}$$
$$= \begin{bmatrix} A_1 B \\ \vdots \\ A_m B \end{bmatrix} = \begin{bmatrix} AB^1 & \cdots & AB^n \end{bmatrix}$$

Três interpretações

Sejam $A \in \mathbb{R}^{m \times p}$ e $B \in \mathbb{R}^{p \times n}$. Existem três formas de interpretarmos $C = A \cdot B$:

- Soma de matrizes de posto 1 (*outer product*)

$$C = \begin{bmatrix} a_{11}b_{11} + \cdots + a_{1p}b_{p1} & \cdots & a_{11}b_{1n} + \cdots + a_{1p}b_{pn} \\ & \vdots & \\ a_{m1}b_{11} + \cdots + a_{mp}b_{p1} & \cdots & a_{m1}b_{1n} + \cdots + a_{mp}b_{pn} \end{bmatrix}$$
$$= \begin{bmatrix} a_{11}b_{11} & \cdots & a_{11}b_{1n} \\ & \vdots & \\ a_{m1}b_{11} & \cdots & a_{m1}b_{1n} \end{bmatrix} + \cdots + \begin{bmatrix} a_{1p}b_{1p} & \cdots & a_{1p}b_{pn} \\ & \vdots & \\ a_{mp}b_{1p} & \cdots & a_{mp}b_{pn} \end{bmatrix}$$
$$= A^1 B_1 + \cdots + A^p B_p$$

Table of Contents

Matrizes diagonais

Seja $D \in \mathbb{R}^{n \times n}$ uma matriz diagonal

$$D = \begin{bmatrix} d_{11} & 0 & 0 & \cdots & 0 \\ 0 & d_{22} & 0 & \cdots & 0 \\ & & \ddots & & \\ & & & \ddots & \\ 0 & 0 & 0 & \cdots & d_{nn} \end{bmatrix}$$

- D pode ser armazenada no computador em um vetor

$$d = \begin{bmatrix} d_{11} & d_{22} & \cdots & d_{nn} \end{bmatrix}$$

- Economizamos $n^2 - n$ posições na memória!

Multiplicação

- $D \in \mathbb{R}^{n\times n}$ diagonal e $v \in \mathbb{R}^n$

$$Dv = \begin{bmatrix} d_{11}v_1 \\ \vdots \\ d_{nn}v_n \end{bmatrix}$$

- $D \in \mathbb{R}^{n\times n}$ diagonal e $A \in \mathbb{R}^{n\times n}$ qualquer

$$D \cdot A = \begin{bmatrix} d_{11}a_{11} & \cdots & d_{11}a_{1n} \\ d_{22}a_{21} & \cdots & d_{22}a_{2n} \\ & \vdots & \\ d_{nn}a_{n1} & \cdots & d_{nn}a_{nn} \end{bmatrix}$$

Matriz esparsa qualquer

Considere a matriz

$$A = \begin{bmatrix} 4 & 0 & 2 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Como A tem muitos zeros, podemos armazená-la de uma forma inteligente através de 3 vetores:

- `vi[k]` guarda a linha do k-ésimo elemento não nulo
- `vj[k]` guarda a coluna do k-ésimo elemento não nulo
- `v[k]` guarda o valor do k-ésimo elemento não nulo

$$\begin{aligned} \texttt{vi} &= [1, 2, 1] \\ \texttt{vj} &= [1, 2, 3] \\ \texttt{v} &= [4, 5, 2] \end{aligned}$$

Matriz esparsa qualquer

No caso de $A \in \mathbb{R}^{m \times n}$ geral, temos que, se `vi[k]` $= i$, `vj[k]` $= j$ e `v[k]` $= x$, então

$$a_{ij} = x$$

`length(v)` indica quantos elementos não nulos há em A.

Como implementar os produtos?

Obrigado!

fncsobral@uem.br