FGV EMAp

João Pedro Jerônimo e Arthur Rabello

Algebra Linear Numérica

Revisão para A2

Rio de Janeiro

2025

Contents

1. Lecture 16 - Estabilidade da Triangularização de Householder 3
   1.1. O Experimento 3
   1.2. Teorema 4
   1.3. Algoritmo para resolver $Ax = b$ 4
2. Lecture 17 - Estabilidade da Back Substitution 6
   2.1. Teorema da Estabilidade Retroativa (Backward Stability) 7
3. Lecture 24 - Problemas de Autovalores 7
   3.1. Definições 7
   3.2. Decomposição em Autovalores 7
   3.3. Multiplicidades Algébrica e Geométrica 8
   3.4. Transformações Similares 9
   3.5. Diagonalização 9
   3.6. Autovalores e Matrizes Deficientes 9
   3.7. Determinante e Traço 9
   3.8. Diagonalização Unitária 9
   3.9. Forma de Schur 9
4. Lecture 25 - Algoritmos de Autovalores 9
   4.1. Ideia da Iteração de Potência 9
   4.2. A ideia dos Algoritmos de Autovalores 9
   4.3. Forma de Schur e Diagonalização 9
   4.4. As 2 fases do Cálculo de Autovalores, Forma de Hessenberg 9
5. Lecture 26 - Redução à forma de Hessenberg 9
   5.1. A Redução 9
   5.2. Redução à Hessenberg via Householder 9
   5.3. Custo Computacional 9
   5.4. O Caso Hermitiano 9
   5.5. Estabilidade do Algoritmo 9
6. Lecture 27 - Quociente de Rayleigh e Iteração Inversa 9
   6.1. Restrição à matrizzes reais e simétricas 9
   6.2. Quociente de Rayleigh 9
   6.3. Iteração de Potência com o Quociente de Rayleigh 9
   6.4. Iteração Inversa 9
7. Lecture 30 - Calculando a SVD 9

Disclaimer: Nesse resumo, quando falarmos de **computadores ideais**, estamos nos referindo a computadores que satisfaçam: $\mathbf{fl}(x) = x(1 + \varepsilon)$ e $x \odot y = (x \cdot y)(1 + \varepsilon)$. Se essa notação lhe é estranha, veja o resumo anterior, mais especificamente sobre a **Lecture 13**

1. Lecture 16 - Estabilidade da Triangularização de Householder

Nesse capítulo, a gente tem uma visão mais aprofundada da análise de **erro retroativo** (Backwards Stable). Dando uma breve recapitulada, para mostrar que um algoritmo $\tilde{f} : X \to Y$ é **backwards stable**, você tem que mostrar que, ao aplicar $\tilde{f}$ em uma entrada x, o resultado retornado seria o mesmo que aplicar o problema original $f : X \to Y$ em uma entrada levemente perturbada $x + \Delta x$, de forma que $\Delta x = O(\varepsilon_{\text{machine}})$.

1.1. O Experimento

O livro nos mostra um experimento no matlab para demonstrar a estabilidade em ação e alguns conceitos importantes, irei fazer o mesmo experimento, porém, utilizarei código em python e mostrarei meus resultados aqui.

Primeiro de tudo, mostraremos na prática que o algoritmo de **Householder** é **backwards stable**. Vamos criar uma matriz A com a fatoração QR conhecida, então vamos gerar as matrizes Q e R. Aqui, temos que $\varepsilon_{\text{machine}} = 2.220446049250313 \times 10^{-16}$:

Python

```python
import numpy as np
np.random.seed(0)  # Ter sempre os mesmos resultados
# Crio R triangular superior (50 x 50)
R_1 = np.triu(np.random.random_sample(size=(50, 50)))
# Crio a matriz Q a partir de uma matriz aleatória
Q_1, _ = np.linalg.qr(np.random.random_sample(size=(100, 50)), mode='reduced')
# Crio a minha matriz com fatoração QR conhecida (A = Q_1 R_1)
A = Q_1 @ R_1
# Calculo a fatoração QR de A usando Householer
Q_2, R_2 = householder_qr(A)
```

Sabemos que, por conta de erros de aproximação, a matriz A que temos no código não é **exatamente** igual a que obteríamos se tivéssemos fazendo $Q_1 R_1$ na mão, mas é preciso o suficiente. Podemos ver aqui que elas são diferentes:

CÓDIGO (Python)

```python
print(np.linalg.norm(Q_1 - Q_2))
print(np.linalg.norm(R_1 - R_2))
```

SAÍDA

```
7.58392995752057e-8
8.75766271246312e-9
```

Perceba que é um erro muito grande, não é tão próximo de 0 quanto eu gostaria, se eu printasse as matrizes Q_2 e R_2 eu veria que, as entradas que deveriam ser 0, tem erro de magnitude $\approx 10^{17}$. Bem, se ambas tem um erro tão grande, então o resultado da multiplicação delas em comparação com A também vai ser grande, correto?

CÓDIGO (Python)

```python
print(np.linalg.norm(A - Q_2 @ R_2))
```

SAÍDA

```
3.8022328832723555e-14
```

Veja que, mesmo minhas matrizes Q_2 e R_2 tendo erros bem grandes com relação às matrizes Q_1 e R_2, conseguimos uma aproximação de A bem precisa com ambas. Vamos agora dar um destaque nessa acurácia de $Q_2 R_2$:

CÓDIGO (Python)

```python
delta_Q_1 = np.random.random_sample(size=Q_1.shape)
delta_R_1 = np.random.random_sample(size=R_1.shape)
Q_3 = Q_1 + delta_Q_1 * 1e-4
R_3 = R_1 + delta_R_1 * 1e-4
print(np.linalg.norm(A - Q_3 @ R_3))
```

SAÍDA

```
0.05197521348918455
```

Perceba o quão grande é esse erro, é **enorme**, então: Q_2 não é melhor que Q_3, R_2 não é melhor que R_3, mas $Q_2 R_2$ é muito mais preciso do que $Q_3 R_3$

1.2. Teorema

Vamos ver que, de fato, o algoritmo de **Householder** é **backwards stable** para toda e qualquer matriz A. Fazendo a análise de backwards stable, nosso resultado precisa ter esse formato aqui:

$$\tilde{Q}\tilde{R} = A + \delta A \tag{1}$$

com $\|\delta A\| \ / \ \|A\| = O(\varepsilon_{\text{machine}})$. Ou seja, calcular a QR de A pelo algoritmo é o mesmo que calcular a QR de $A + \delta A$ da forma matemática. Mas aqui temos uns adendos.

A matriz $\tilde{R}$ é como imaginamos, a matriz triangular superior obtida pelo algoritmo, onde as entradas abaixo de 0 podem não ser exatamente 0, mas **muito próximas**.

Porém, $\tilde{Q}$ **não é aproximadamente** ortogonal, ela é **perfeitamente** ortogonal, mas por quê? Pois no algoritmo de Householder, não calculamos essa matriz diretamente, ela fica "*implícita*" nos cálculos, logo, podemos assumir que ela é perfeitamente ortogonal, já que o computador não a calcula, ou seja, não há erros de arredondamento. Vale lembrar também que $\tilde{Q}$ é definido por:

$$\tilde{Q} = \tilde{Q}_1 \tilde{Q}_2 ... \tilde{Q}_n \tag{2}$$

De forma que $\tilde{Q}$ é perfeitamente unitária e cada matriz $\tilde{Q}_j$ é definida como o refletor de householder no vetor de floating point $\tilde{v_k}$ (Olha a página 73 do livro pra você relembrar direitinho o que é esse vetor $\tilde{v_k}$ no algoritmo). Lembrando que $\tilde{Q}$ é perfeitamente ortogonal, já que eu não calculo ela no computador diretamente, se eu o fizesse, então ela não seria perfeitamente ortogonal, teriam pequenos erros.

Teorema 1.2.1 (Householder's Backwards Stability): Deixe que a fatoração QR de $A \in \mathbb{C}^{m \times n}$ seja dada por $A = QR$ e seja computada pelo algoritmo de **Householder**, o resultado dessa computação são as matrizes $\tilde{Q}$ e $\tilde{R}$ definidas anterioremente. Então temos:

$$\tilde{Q}\tilde{R} = A + \delta A \tag{3}$$

Tal que:

$$\frac{\|\delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \tag{4}$$

para algum $\delta A \in \mathbb{C}^{m \times n}$

1.3. Algoritmo para resolver $Ax = b$

Vimos que o algoritmo de householder é backwards stable, show! Porém, sabemos que não costumamos fazer essas fatorações só por fazer né, a gente faz pra resolver um sistema $Ax = b$, ou outros tipos de problemas. Certo, mas, se fizermos um algoritmo que resolve $Ax = b$ usando a fatoração QR obtida com householder, a gente precisa que Q e R sejam precisos? Ou só precisamos que QR seja preciso? O bom é que precisamos apenas que QR seja precisa! Vamos mostrar isso para a resolução de sistemas $m \times m$ não singulares.

```
function ResolverSistema(A ∈ ℂ^{m×n}, b ∈ ℂ^{m×1}) {
    QR = Householder(A)
    y = Q*b
    x = R^{-1}y
    return x
}
```

Algoritmo 1: Algoritmo para calcular $Ax = b$

Esse algoritmo é **backwards stable**, e é bem passo-a-passo já que cada passo dentro do algoritmo é **backwards stable**.

Teorema 1.3.1: O Algoritmo 1 para solucionar $Ax = b$ é **backwards stable**, satisfazendo

$$(A + \Delta A)\tilde{x} = b \tag{5}$$

com

$$\frac{\|\Delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \tag{6}$$

para algum $\Delta A \in \mathbb{C}^{m \times n}$

Demonstração: Quando computamos $\tilde{Q}^* b$, por conta de erros de aproximação, não obtemos um vetor y, e sim $\tilde{y}$. É possível mostrar (Não faremos) que esse vetor $\tilde{y}$ satisfaz:

$$\left(\tilde{Q} + \delta Q\right)\tilde{y} = b \tag{7}$$

satisfazendo $\frac{\|\delta Q\|}{\|\tilde{Q}\|} = O(\varepsilon_{\text{machine}})$

Ou seja, só pra esclarecer, aqui (nesse passo de y) a gente ta tratando o problema f de calcular $Q^* b$, ou seja $f(Q) = Q^* b$, então usamos um algoritmo comum $\tilde{f}(Q) = Q^* b$ (Não matematicamente, mas usando as operações de um computador), daí reescrevemos isso como $\tilde{f}(Q) = (Q + \delta Q)^* b$, por isso podemos reescrever como a equação que falamos anteriormente.

No último passo, a gente usa **back substitution** pra resolver o sistema $x = R^{-1} y$ e esse algoritmo é **backwards stable** (Isso vamos provar na próxima lecture). Então temos que:

$$(\tilde{R} + \delta R)\tilde{x} = \tilde{y} \tag{8}$$

satisfazendo $\frac{\|\delta R\|}{\|\tilde{R}\|} = O(\varepsilon_{\text{machine}})$

Agora podemos ir pro algoritmo em si, temos um problema $f(A)$: Resolver $Ax = b$, daí usamos $\tilde{f}(A)$: Usando householder, resolve $Ax = b$. Então, se o algoritmo nos dá as matrizes perturbadas que citei anteriormente ($Q + \delta Q$ e $R + \delta R$), ao substituir isso por A, eu tenho que ter um resultado $A + \Delta A$ com $\frac{\|\Delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}})$, vamos ver:

$$b = \left(\tilde{Q} + \delta Q\right)(\tilde{R} + \delta R)\tilde{x} \tag{9}$$

$$b = \left(A + \delta A + \tilde{Q}(\delta R) + (\delta Q)\tilde{R} + (\delta Q)(\delta R)\right)\tilde{x} \tag{10}$$

$$b = (A + \Delta A)\tilde{x} \Leftrightarrow \Delta A = \delta A + \tilde{Q}(\delta R) + (\delta Q)\tilde{R} + (\delta Q)(\delta R) \tag{11}$$

Como ΔA é a soma de 4 termos, temos que mostrar que cada um desses termos é pequeno com relação a A (Ou seja, mostrar que $\frac{\|X\|}{\|A\|} = O(\varepsilon_{\text{machine}})$ onde X é um dos 4 termos de ΔA).

- δA: Pela própria definição que o algoritmo de householder é backwards stable nós sabemos que δA satisfaz a condição de $O(\varepsilon_{\text{machine}})$
- $(\delta Q)\tilde{R}$:

$$\frac{\|(\delta Q)\tilde{R}\|}{\|A\|} \leq \|(\delta Q)\| \frac{\|\tilde{R}\|}{\|A\|} \tag{12}$$

Perceba que

$$\frac{\|\tilde{R}\|}{\|A\|} \leq \frac{\|\tilde{Q}^*(A + \delta A)\|}{\|A\|} \leq \|\tilde{Q}^*\| \frac{\|A + \delta A\|}{\|A\|} \tag{13}$$

Lembra que, quando trabalhamos com $O(\varepsilon_{\text{machine}})$, a gente ta trabalhando com um limite implícito que, no caso, aqui é $\varepsilon_{\text{machine}} \to 0$. Ou seja, se temos que $\varepsilon_{\text{machine}} \to 0$, o erro de arredondamento diminui cada vez mais, certo? Então $\delta A \to 0$ ou seja:

$$\frac{\|\tilde{R}\|}{\|A\|} = O(1) \tag{14}$$

O que nos indica que

$$\|\delta Q\| \frac{\|\tilde{R}\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \tag{15}$$

- $\tilde{Q}(\delta R)$: Provamos de uma forma similar

$$\frac{\|\tilde{Q}(\delta R)\|}{\|A\|} \leq \|\tilde{Q}\| \frac{\|\delta R\|}{\|A\|} = \|\tilde{Q}\| \frac{\|\delta R\|}{\|\tilde{R}\|} \frac{\|\tilde{R}\|}{\|A\|} \leq \|\tilde{Q}\| \frac{\|\delta R\|}{\|\tilde{R}\|} = O(\varepsilon_{\text{machine}}) \tag{16}$$

- $(\delta Q)(\delta R)$: Por último:

$$\frac{\|(\delta Q)(\delta R)\|}{\|A\|} \leq \|\delta Q\| \frac{\|\delta R\|}{\|A\|} = O(\varepsilon_{\text{machine}}^2) \tag{17}$$

Ou seja, todos os termos de ΔA são da ordem $O(\varepsilon_{\text{machine}})$, ou seja, provamos que resolver $Ax = b$ usando householder é um algoritmo **backwards stable**. Se a gente junta alguns teoremas e temos que:

Teorema 1.3.2: A solução $\tilde{x}$ computada pelo algoritmo satisfaz:

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O(\kappa(A)\varepsilon_{\text{machine}}) \tag{18}$$

□

2. Lecture 17 - Estabilidade da Back Substitution

Só para esclarecer, o termo **back substitution** se refere ao algoritmo de resolver um sistema triangular superior

$$\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1m} \\ & r_{22} & \cdots & r_{2m} \\ & & \ddots & \vdots \\ & & & r_{mm} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_m \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \tag{19}$$

E é aquele esquema, a gente vai resolvendo de baixo para cima, o que resulta nesse algoritmo (A gente escreve como uma sequência de fórmulas por conveniência, mas é o mesmo que escrever um loop):

function BackSubstitution($R \in \mathbb{C}^{m \times m}$, $b \in \mathbb{C}^{m \times 1}$) {

$x_m = b_m / r_{mm}$

$x_{m-1} = (b_{m-1} - x_m r_{m-1,m}) / r_{m-1,m-1}$

$x_{m-2} = (b_{m-2} - x_{m-1} r_{m-2,m-1} - x_m r_{m-2,m}) / r_{m-2,m-2}$

$\vdots$

$x_j = \left(b_j - \sum_{k=j+1}^{m} x_k r_{jk}\right) / r_{jj}$

}

Algoritmo 2: Algoritmo de **Back Substitution**

2.1. Teorema da Estabilidade Retroativa (Backward Stability)

A gente viu no último tópico (Estabilidade de Householder) que a **back substitution** era um dos passos para chegar no resultado final, porém, nós apenas assumimos que ela era **backward stable**, mas a gente **não** provou isso! Porém, antes de provarmos isso, vamos estabelecer que as subtrações serão feitas da esquerda para a direita (Sim, isso pode influenciar). Mas, como o livro não explica muito bem o porquê de isso influenciar, vou dar uma breve explicação e exemplificação:

Quando realizamos uma sequência de subtrações pela **direita**, caso os números sejam muito próximos, pode ocorrer o chamado **cancelamento catastrófico**, que é a perca de muitos dígitos significativos, veja um exemplo:

CÓDIGO — Python

```python
a = 1e16
b = 1e16
c = 1
print((a-b)-c)
```

SAÍDA

```
-1.0
```

O que parece correto! Mas veja o que acontece se invertermos a ordem e executarmos $a - (b - c)$

CÓDIGO — Python

```python
a = 1e16
b = 1e16
c = 1
print(a-(b-c))
```

SAÍDA

```
0.0
```

Veja que houve um problema no arredondamento! Então os sistemas, por convenção, utilizam o esquema de subtrações pela esquerda.

Voltando ao algoritmo de **back substitution**, temos o seguinte teorema:

Teorema 2.1.1: Deixe o Algoritmo 2 ser aplicado a um problema de $Rx = b$ com R triangular superior.

3. Lecture 24 - Problemas de Autovalores

Esse capítulo nada mais é do que uma revisão de resultados da A2 de álgebra linear.

3.1. Definições

Dada uma matriz $A \in \mathbb{C}^{m \times n}$, pela decomposição SVD $A = U\Sigma V^*$ sabemos que A é uma transformação que **estica** e **rotaciona** vetores. Por isso, estamos interessados em subespaços de $\mathbb{C}^m$ nos quais a matriz age como uma multiplicação escalar, ou seja, estamos interessados nos $x \in \mathbb{C}^n$ que são somente esticados pela matriz. Como $Ax \in \mathbb{C}^m$ e $\lambda x \in \mathbb{C}^n$, concluimos que $m = n$: A matriz **deve ser quadrada**. Afinal, não faz sentido se λx e Ax estiverem em conjuntos distintos. Com isso, prosseguimos com a definição:

Definição 3.1.1: (Autovalores e Autovetores) Dada $A \in \mathbb{C}^{m \times m}$, um *autovetor* de A é $x \in \mathbb{C}^m \setminus \{0\}$ que satisfaz:

$$Ax = \lambda x \tag{20}$$

$\lambda \in \mathbb{C}$ é dito *autovalor* associado a x.

3.2. Decomposição em Autovalores

Uma **decomposição em autovalores** de uma matriz $A \in \mathbb{C}^{m \times n}$ é uma fatoração:

$$A = X\Lambda X^{-1} \tag{21}$$

Onde Λ é diagonal e $\det(X) \neq 0$.

Isso é equivalente a:

$$\underbrace{\begin{pmatrix} & & \\ & A & \\ & & \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & ... & x_n \\ | & | & | & | \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} \lambda_1 & 0 & ... & 0 \\ 0 & \lambda_2 & ... & 0 \\ 0 & 0 & ... & 0 \end{pmatrix}}_{\Lambda} \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & ... & x_n \\ | & | & | & | \end{pmatrix}}_{X} \quad (22)$$

Da eq. (22) e da Definição 3.1.1, decorre que $Ax_i = \lambda_i x_i$, então a i-ésima coluna de X é um autovetor de A e λ_i é o autovalor associado (a x_i).

A decomposição apresentada pode representar uma mudança de base: Considere $Ax = b$ e $A = X\Lambda X^{-1}$, então:

$$Ax = b \Leftrightarrow X\Lambda X^{-1}x = b \Leftrightarrow \Lambda(X^{-1}x) = X^{-1}b \quad (23)$$

Então para calcular Ax, podemos expandir x como combinação das colunas de X e aplicar Λ. Como Λ é diagonal, o resultado ainda vai ser uma combinação das colunas de X.

3.3. Multiplicidades Algébrica e Geométrica

Como mencionado anteriormente, definimos os conjuntos nos quais a matriz atua como multiplicação escalar:

Definição 3.3.1: (Autoespaço) Dada $A \in \mathbb{C}^{m \times n}, \lambda \in \mathbb{C}$, o definimos $S_\lambda \in \mathbb{C}^m$ como sendo o **autoespaço** gerado por todos os $v_\in \mathbb{C}^m$ tais que $Av = \lambda v$

Interpretaremos $\dim(S_\lambda)$ como a maior quantidade de autovetores L.I associados a um único λ, e chamaremos isso de *multiplicidade geométrica* de λ. Então temos:

Definição 3.3.2: (Multiplicidade Geométrica) A multiplicidade geométrica de λ é $\dim(S_\lambda)$

Note que da eq. (20):

$$Ax = \lambda x \Leftrightarrow Ax - \lambda x = 0 \Leftrightarrow (A - \lambda I)x = 0 \quad (24)$$

Mas como $x \neq 0$ e $x \in N(A - \lambda I)$, $(A - \lambda I)$ não é injetiva. Logo não é inversível:

$$\det(A - \lambda I) = 0 \quad (25)$$

A eq. (25) se chama *polinômio característico* de A e é um polinômio de grau m em λ. Pelo teorema fundamental da Álgebra, se $\lambda_1, ..., \lambda_n$ são raízes de eq. (25), então podemos escrever isso como:

$$p(\varphi) = (\varphi - \lambda_1)(\varphi - \lambda_2)...(\varphi - \lambda_n) \quad (26)$$

Com isso, prosseguimos com:

Definição 3.3.3: (Multiplicidade Algébrica) A multiplicidade algébrica de λ é a multiplicidade de λ como raiz do polinômio característico de A

3.4. Transformações Similares

3.5. Diagonalização

3.6. Autovalores e Matrizes Deficientes

3.7. Determinante e Traço

3.8. Diagonalização Unitária

3.9. Forma de Schur

4. Lecture 25 - Algoritmos de Autovalores

4.1. Ideia da Iteração de Potência

4.2. A ideia dos Algoritmos de Autovalores

Escrever pqq tem q ser iterativo. (pag 192 trefethen)

4.3. Forma de Schur e Diagonalização

4.4. As 2 fases do Cálculo de Autovalores, Forma de Hessenberg

5. Lecture 26 - Redução à forma de Hessenberg

5.1. A Redução

5.2. Redução à Hessenberg via Householder

5.3. Custo Computacional

5.4. O Caso Hermitiano

5.5. Estabilidade do Algoritmo

6. Lecture 27 - Quociente de Rayleigh e Iteração Inversa

6.1. Restrição à matrizzes reais e simétricas

6.2. Quociente de Rayleigh

6.3. Iteração de Potência com o Quociente de Rayleigh

6.4. Iteração Inversa

7. Lecture 30 - Calculando a SVD

Calcular autovalores da matriz:

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \tag{27}$$

Retorna os valores singulares de A com $\kappa(A)$, e não $\kappa^2(A)$ PQ CARALHOS