

Algebra Linear Numérica A2 Recap

20/05/2025

Contents

1. Lecture 16 - Estabilidade da Triangularização de Householder	2
1.1. O Experimento	2
1.2. Teorema	3
1.3. Algoritmo para resolver $Ax = b$	3
2. Lecture 24 - Problemas de Autovalores	5
2.1. Definições	5
2.2. Decomposição em Autovalores	5
2.3. Multiplicidades Algébrica e Geométrica	6
2.4. Transformações Similares	7
2.5. Diagonalização	7
2.6. Autovalores e Matrizes Deficientes	7
2.7. Determinante e Traço	7
2.8. Diagonalização Unitária	7
2.9. Forma de Schur	7
3. Lecture 25 - Algoritmos de Autovalores	7
3.1. Ideia da Iteração de Potência	7
3.2. A ideia dos Algoritmos de Autovalores	7
3.3. Forma de Schur e Diagonalização	7
3.4. As 2 fases do Cálculo de Autovalores, Forma de Hessenberg	7
4. Lecture 26 - Redução à forma de Hessenberg	7
4.1. A Redução	7
4.2. Redução à Hessenberg via Householder	7
4.3. Custo Computacional	7
4.4. O Caso Hermitiano	7
4.5. Estabilidade do Algoritmo	7
5. Lecture 27 - Quociente de Rayleigh e Iteração Inversa	7
5.1. Restrição à matrizes reais e simétricas	7
5.2. Quociente de Rayleigh	7
5.3. Iteração de Potência com o Quociente de Rayleigh	7
5.4. Iteração Inversa	7
6. Lecture 30 - Calculando a SVD	7

Olha só, agora é em português! Coisa boa. Esse documento se refere aos capítulos 16 e posteriores.

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 $\tilde{f} : X \rightarrow Y$ 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 $f : X \rightarrow Y$ 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 $\varepsilon_{\text{machine}} = 2.220446049250313 \times 10^{-16}$:

```
1 import numpy as np
2 np.random.seed(0) # Ter sempre os mesmos resultados
3 # Crio R triangular superior (50 x 50)
4 R_1 = np.triu(np.random.random_sample(size=(50, 50)))
5 # Crio a matriz Q a partir de uma matriz aleatória
6 Q_1, _ = np.linalg.qr(np.random.random_sample(size=(100, 50)), mode='reduced')
7 # Crio a minha matriz com fatoração QR conhecida (A = Q_1 R_1)
8 A = Q_1 @ R_1
9 # Calculo a fatoração QR de A usando Householder
10 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:

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

```
SAÍDA
1 7.58392995752057e-8
2 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?

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

```
SAÍDA
1 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_1 , conseguimos uma aproximação de A bem precisa com ambas. Vamos agora dar um destaque nessa acurácia de $Q_2 R_2$:

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

```
SAÍDA
1 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:

$$\tilde{Q}\tilde{R} = A + \delta A \quad (1)$$

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:

$$\tilde{Q} = \tilde{Q}_1 \tilde{Q}_2 \dots \tilde{Q}_n \quad (2)$$

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 anteriormente. Então temos:

$$\tilde{Q}\tilde{R} = A + \delta A \quad (3)$$

Tal que:

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

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.

Algoritmo para resolver $Ax = b$

```
1 function ResolverSistema( $A \in \mathbb{C}^{m \times n}$ ,  $b \in \mathbb{C}^{m \times 1}$ ) {  
2    $QR = \text{Householder}(A)$   
3    $y = Q^*b$   
4    $x = R^{-1}y$   
5   return  $x$   
6 }
```

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 descrito anteriormente para solucionar $Ax = b$ é **backwards stable**, satisfazendo

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

com

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

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:

$$(\tilde{Q} + \delta Q)\tilde{y} = b \quad (7)$$

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:

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

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:

$$b = (\tilde{Q} + \delta Q)(\tilde{R} + \delta R)\tilde{x} \quad (9)$$

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

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

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 $O(\varepsilon_{\text{machine}})$
- $(\delta Q)\tilde{R}$:

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

Perceba que

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

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

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

O que nos indica que

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

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

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

- $(\delta Q)(\delta R)$: Por último:

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

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:

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

□

2. Lecture 24 - Problemas de Autovalores

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

2.1. 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 $\lambda x \in \mathbb{C}^n$, concluímos 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 2.1.1: (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:

$$Ax = \lambda x \quad (19)$$

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

2.2. Decomposição em Autovalores

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

$$A = X\Lambda X^{-1} \quad (20)$$

Onde Λ é diagonal e $\det(X) \neq 0$.

Isso é equivalente a:

$$\underbrace{\begin{pmatrix} & & & \\ & A & & \\ & & & \end{pmatrix}}_A \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & \dots & x_n \\ | & | & | & | \end{pmatrix}}_X = \underbrace{\begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \end{pmatrix}}_\Lambda \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & \dots & x_n \\ | & | & | & | \end{pmatrix}}_X \quad (21)$$

Da eq. (21) e da Definição 2.1.1, 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:

$$Ax = b \Leftrightarrow X\Lambda X^{-1}x = b \Leftrightarrow \Lambda(X^{-1}x) = X^{-1}b \quad (22)$$

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 .

2.3. Multiplicidades Algébrica e Geométrica

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

Definição 2.3.1: (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 2.3.2: (Multiplicidade Geométrica) A multiplicidade geométrica de λ é $\dim(S_\lambda)$

Note que da eq. (19):

$$Ax = \lambda x \Leftrightarrow Ax - \lambda x = 0 \Leftrightarrow (A - \lambda I)x = 0 \quad (23)$$

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

$$\det(A - \lambda I) = 0 \quad (24)$$

A eq. (24) se chama *polinômio característico* de A e é um polinômio de grau m em λ . Pelo teorema fundamental da Álgebra, se $\lambda_1, \dots, \lambda_n$ são raízes de eq. (24), então podemos escrever isso como:

$$p(\varphi) = (\varphi - \lambda_1)(\varphi - \lambda_2) \dots (\varphi - \lambda_n) \quad (25)$$

Com isso, prosseguimos com:

Definição 2.3.3: (Multiplicidade Algébrica) A multiplicidade algébrica de λ é a multiplicidade de λ como raiz do polinômio característico de A

2.4. Transformações Similares

2.5. Diagonalização

2.6. Autovalores e Matrizes Deficientes

2.7. Determinante e Traço

2.8. Diagonalização Unitária

2.9. Forma de Schur

3. Lecture 25 - Algoritmos de Autovalores

3.1. Ideia da Iteração de Potência

3.2. A ideia dos Algoritmos de Autovalores

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

3.3. Forma de Schur e Diagonalização

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

4. Lecture 26 - Redução à forma de Hessenberg

4.1. A Redução

4.2. Redução à Hessenberg via Householder

4.3. Custo Computacional

4.4. O Caso Hermitiano

4.5. Estabilidade do Algoritmo

5. Lecture 27 - Quociente de Rayleigh e Iteração Inversa

5.1. Restrição à matrizes reais e simétricas

5.2. Quociente de Rayleigh

5.3. Iteração de Potência com o Quociente de Rayleigh

5.4. Iteração Inversa

6. Lecture 30 - Calculando a SVD

Calcular autovalores da matriz:

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \quad (26)$$

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