

FGV EMap

João Pedro Jerônimo e Arthur Rabello Oliveira

# Algebra Linear Numérica

Revisão para A2

Rio de Janeiro

2025

# Contents

1	Lecture 16 - Estabilidade da Triangularização de Householder .....	5
1.1	O Experimento .....	6
1.2	Teorema .....	6
1.3	Algoritmo para resolver $Ax = b$ .....	7
2	Lecture 17 - Estabilidade da Back Substitution .....	10
2.1	Teorema da Estabilidade Retroativa (Backward Stability) .....	11
3	Lecture 18 - Condicionando Problemas de Mínimos Quadrados .....	14
3.1	O Teorema .....	15
4	Lecture 19 - Estabilidade de Algoritmos de Mínimos Quadrados .....	20
4.1	Primeira Etapa .....	21
4.2	Householder .....	21
4.3	Ortogonalização de Gram-Schmidt .....	22
4.4	Equações Normais .....	22
4.5	SVD .....	23
4.6	Problemas de Mínimos Quadrados com Posto-Incompleto .....	23
5	Lecture 24 - Problemas de Autovalores .....	24
5.1	Definições .....	25
5.2	Decomposição em Autovalores .....	25
5.3	Multiplicidades Algébrica e Geométrica .....	25
5.4	Transformações Similares .....	26
5.5	Autovalores e Matrizes Deficientes .....	27
5.6	Diagonalizabilidade .....	27
5.7	Determinante e Traço .....	27
5.8	Diagonalização Unitária .....	27
5.9	Forma de Schur .....	28
6	Lecture 25 - Algoritmos de Autovalores .....	29
6.1	Algoritmos óbvios (Ou nem tanto) .....	30
6.2	Uma dificuldade fundamental .....	30
6.3	Fatoração e Diagonalização de Schur .....	30
6.4	Duas fases da computação de Autovalores .....	30
7	Lecture 26 - Redução à forma de Hessenberg .....	32
7.1	Uma ideia de Girico .....	33
7.2	Uma boa ideia .....	33
7.3	Hermitiana .....	33
7.4	Estabilidade .....	33
8	Lecture 27 - Quociente de Rayleigh e Iteração Inversa .....	34
8.1	Restrição à matrizes reais e simétricas .....	35
8.2	Quociente de Rayleigh .....	35
8.3	Iteração por Potências .....	36
8.4	Iteração Inversa .....	37
8.5	Iteração do Quociente de Rayleigh .....	38
9	Lecture 28 - Algoritmo QR sem Shift .....	40
9.1	O Algoritmo QR .....	41
9.2	Iterações Simultâneas Não-normalizadas .....	41
9.3	Iteração Simultânea .....	43
9.4	Iteração Simultânea $\Leftrightarrow$ Algoritmo QR .....	44
9.5	Convergência do algoritmo QR .....	45
10	Lecture 29 - Algoritmo QR com Shifts .....	46
10.1	Conexão com a Iteração Reversa .....	47
10.2	Conexão com o Algoritmo de Iteração Reversa com Shifts .....	47
10.3	Conexão com a Iteração do Quociente de Rayleigh .....	48

10.4	Wilkinson Shift .....	48
10.5	Estabilidade e Precisão .....	49
11	Discos de Gershgorin .....	50
12	Lecture 31 - Calculando a SVD .....	52
12.1	SVD de $A$ via autovalores de $A^*A$ .....	53

**Nota:** Os **computadores ideais** que mencionaremos, são computadores nos quais o *axioma fundamental da aritmética de ponto flutuante* é satisfeito. Convidamos o leitor a ler sobre isso no resumo anterior (A1), especificamente na **lecture 13**

Esse é um resumo feito por João Pedro Jerônimo (Ciência de Dados) e Arthur Rabello (Matemática Aplicada) com objetivo de traduzir os hieróglifos contidos no livro de Álgebra Linear Numérica do Trefthen e do Bau



## **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  $\Delta x = O(\epsilon_{\text{machine}})$ .

## 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  $\epsilon_{\text{machine}} = 2.220446049250313 \times 10^{-16}$ :

```
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  $R_2$ , 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.

---

```

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 }
```

---

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

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

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  $\Delta A$  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  $\Delta A$ ).

- $\delta A$ : Pela própria definição que o algoritmo de householder é backwards stable nós sabemos que  $\delta A$  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  $\Delta A$  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 17 - Estabilidade da Back Substitution**

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

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

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):

---

```

1 function BackSubstitution( $R \in \mathbb{C}^{m \times m}$ ,  $b \in \mathbb{C}^{m \times 1}$ ) {
2    $x_m = b_m / r_{mm}$ 
3    $x_{m-1} = (b_{m-1} - x_m r_{m-1,m}) / r_{m-1,m-1}$ 
4    $x_{m-2} = (b_{m-2} - x_{m-1} r_{m-2,m-1} - x_m r_{m-2,m}) / r_{m-2,m-2}$ 
5    $\vdots$ 
6    $x_j = (b_j - \sum_{k=j+1}^m x_k r_{jk}) / r_{jj}$ 
7 }
```

---

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 perda de muitos dígitos significativos, veja um exemplo:

CÓDIGO	Python
1 a = 1e16	
2 b = 1e16	
3 c = 1	
4 print((a-b)-c)	

SAÍDA
1 -1.0

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

CÓDIGO	Python
1 a = 1e16	
2 b = 1e16	
3 c = 1	
4 print(a-(b-c))	

SAÍDA
1 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 em um **computador ideal**. Esse algoritmo é **backward stable**, ou seja, a solução  $\tilde{x}$  computada satisfaz:

$$(R + \Delta R)\tilde{x} = b \quad (20)$$

para alguma triangular superior  $\Delta R \in \mathbb{C}^{m \times m}$  satisfazendo

$$\frac{\|\Delta R\|}{\|R\|} = O(\varepsilon_{\text{machine}}) \quad (21)$$

*Demonstração:* Essa prova não será muito rigorosa matematicamente, vamos montar a prova para matrizes  $1 \times 1$ ,  $2 \times 2$  e  $3 \times 3$ , de forma que o raciocínio que aplicarmos poderá ser aplicado para matrizes de tamanhos maiores.

- **$1 \times 1$ :** Nesse caso,  $R$  é um único número escalar e, pelo **Algoritmo 2**, temos que:

$$\tilde{x}_1 = b_1 \oplus r_{11} \quad (22)$$

E nós **já sabemos** que essa divisão é backward stable, mas vamos analisar melhor. Queremos manter  $b$  fixo, então temos que expressar  $\tilde{x}_1$  como o  $r_{11}$  original vezes uma leve perturbação. Expressamos então

$$\tilde{x}_1 = \frac{b_1}{r_{11}}(1 + \varepsilon_1) \quad (23)$$

Se definirmos  $\varepsilon'_1 = \frac{-\varepsilon_1}{1+\varepsilon_1}$ , podemos reescrever a equação assim:

$$\begin{aligned} \tilde{x}_1 = \frac{b_1}{r_{11}(1 + \varepsilon'_1)} &\Leftrightarrow \tilde{x}_1 = \frac{b_1}{r_{11}\left(1 - \frac{\varepsilon_1}{1+\varepsilon_1}\right)} \Leftrightarrow \tilde{x}_1 = \frac{b_1}{r_{11} \frac{1+\varepsilon_1-\varepsilon_1}{1+\varepsilon_1}} \\ &\Leftrightarrow \tilde{x}_1 = \frac{b_1}{r_{11} \frac{1}{1+\varepsilon_1}} \Leftrightarrow \tilde{x}_1 = \frac{b_1}{r_{11}}(1 + \varepsilon_1) \end{aligned} \quad (24)$$

Se fizermos a expansão de Taylor de  $\varepsilon'_1$ , conseguimos ver:

$$-\frac{\varepsilon_1}{1 + \varepsilon_1} = -\varepsilon_1 + \varepsilon_1^2 - \varepsilon_1^3 + \varepsilon_1^4 - \dots \quad (25)$$

Ou seja,  $-\varepsilon_1 + O(\varepsilon_1^2)$ , o que mostra que  $1 + \varepsilon'_1$  é uma perturbação válida para o teorema da estabilidade backwards, o que nos mostra também que

$$(r_{11} + \delta r_{11})\tilde{x}_1 = b_1 \quad (26)$$

Com

$$\frac{\|\delta r_{11}\|}{\|r_{11}\|} \leq \varepsilon_{\text{machine}} + O(\varepsilon_{\text{machine}}^2) \quad (27)$$

- **$2 \times 2$ :** Beleza, no caso  $2 \times 2$ , o primeiro passo do algoritmo nós já vimos que é **backwards stable**, vamos para o segundo passo:

$$\tilde{x}_1 = (b_1 \ominus (\tilde{x}_2 \otimes r_{12})) \oplus r_{22} \quad (28)$$

Ai meu Deus, fórmula grande do djabo :( Relaxa, vamo transformar em fórmulas normais com umas perturbações pra gente falar de matemática normal né

$$\tilde{x}_1 = \frac{(b_1 - \tilde{x}_2 r_{12}(1 + \varepsilon_2))(1 + \varepsilon_3)}{r_{22}}(1 + \varepsilon_4) \quad (29)$$

Aqui eu não iniciei os epsilons em  $\varepsilon_1$  porque eu estou tomando intrínseco que esse  $\varepsilon_1$  tá no  $\tilde{x}_2$  que a gente computa antes de computar o  $\tilde{x}_1$  (A gente computa igual o caso  $1 \times 1$ )

Podemos definir  $\varepsilon'_3 = -\frac{\varepsilon_3}{1+\varepsilon_3}$  e  $\varepsilon'_4 = -\frac{\varepsilon_4}{1+\varepsilon_4}$ , assim, podemos reescrever:

$$\tilde{x}_1 = \frac{b_1 - \tilde{x}_2 r_{12}(1 + \varepsilon_2)}{r_{22}(1 + \varepsilon'_3)(1 + \varepsilon'_4)} \quad (30)$$

(Mesmo raciocínio que usamos no caso  $1 \times 1$ ). A gente viu em alguns exercícios da lista que  $(1 + O(\varepsilon_{\text{machine}}))(1 + O(\varepsilon_{\text{machine}})) = 1 + O(\varepsilon_{\text{machine}})$ , com isso em mente, podemos reescrever a equação como

$$\tilde{x}_1 = \frac{b_1 - \tilde{x}_2 r_{12}(1 + \varepsilon_2)}{r_{22}(1 + 2\varepsilon'_5)} \quad (31)$$

Esse  $2\varepsilon'_5$  se dá pois, como vimos no caso  $1 \times 1$ :

$$\begin{aligned} 1 + \varepsilon'_3 &= 1 - \varepsilon_3 + O(\varepsilon_3^2) \\ 1 + \varepsilon'_4 &= 1 - \varepsilon_4 + O(\varepsilon_4^2) \\ \Rightarrow (1 + \varepsilon'_3)(1 + \varepsilon'_4) &= (1 - \varepsilon_3 + O(\varepsilon_3^2))(1 - \varepsilon_4 + O(\varepsilon_4^2)) \\ \Rightarrow 1 - \varepsilon_4 + O(\varepsilon_4^2) - \varepsilon_3 + \varepsilon_3\varepsilon_4 - \varepsilon_3O(\varepsilon_4^2) + O(\varepsilon_3^2) - \varepsilon_4O(\varepsilon_3^2) + O(\varepsilon_4^2)O(\varepsilon_3^2) \end{aligned} \quad (32)$$

Os termos diferentes de 1,  $\varepsilon_3$  e  $\varepsilon_4$  são irrelevantes, pois são **MUITO** pequenos, o que nos dá

$$1 - \varepsilon_4 - \varepsilon_3 = 1 - 2\varepsilon_5 \quad (33)$$

Voltando ao foco, acabamos de mostrar que, se  $r_{11}$ ,  $r_{12}$  e  $r_{22}$  fossem perturbados por fatores  $2\varepsilon_5$ ,  $\varepsilon_2$  e  $\varepsilon_1$  **respectivamente**, a conta feita para calcular  $b_1$ , no computador, seria **exata**. Podemos expressar isso na forma

$$(R + \delta R)\tilde{x}_1 = b_1 \quad (34)$$

De forma que

$$\delta R = \begin{pmatrix} 2|\varepsilon_5| & |\varepsilon_2| \\ & |\varepsilon_1| \end{pmatrix} \quad (35)$$

- **A Indução:** Suponha que, no  $(j - 1)$ -ésimo passo do algoritmo, eu sei que o  $\tilde{x}_{j-1}$  é gerado com um algoritmo backward stable. Nós já mostramos, pelos casos bases, que os primeiros dois passos são backward stable. Vamos relembrar o Algoritmo 2 para  $m$  colunas:

$$\tilde{x}_j = \left( b_j \ominus \sum_{k=j+1}^m x_k \otimes r_{jk} \right) \oplus r_{jj} \quad (36)$$

Usando o **Axioma Fundamental do Ponto Flutuante**:

$$\tilde{x}_j = \frac{(b_j - \sum_{k=j+1}^m x_k r_{jk}(1 + \varepsilon_k))(1 + \varepsilon_{m+1})}{r_{jj}}(1 + \varepsilon_{m+2}) \quad (37)$$

Definindo  $\varepsilon'_{m+1}$  e  $\varepsilon'_{m+2}$  de forma análoga a que fizemos anteriormente:

$$\tilde{x}_j = \frac{b_j - \sum_{k=j+1}^m x_k r_{jk}(1 + \varepsilon_k)}{r_{jj}(1 + \varepsilon'_{m+1})(1 + \varepsilon'_{m+2})} \quad (38)$$

Novamente, estamos expressando  $\tilde{x}_j$  como operações em  $x_k$  e  $b_j$  e com entradas **perturbadas** de  $R$ , mostrando que o algoritmo do **back substitution** é sim **backward stable**

□

### **3 Lecture 18 - Condicionando Problemas de Mínimos Cuadrados**

**Nota:** Nessa lecture, quando escrevemos  $\|\cdot\|$ , estamos nos referindo a norma 2, **não a qualquer norma**, logo,  $\|\cdot\| = \|\cdot\|_2$

Vamos relembrar o problema dos mínimos quadrados?

$$\begin{aligned} \text{Dada } A \in \mathbb{C}^{m \times n} \text{ de posto completo, } m \geq n \text{ e } b \in \mathbb{C}^m, \\ \text{ache } x \in \mathbb{C}^n \text{ tal que } \|b - Ax\|_2 \text{ seja a menor possível} \end{aligned} \quad (39)$$

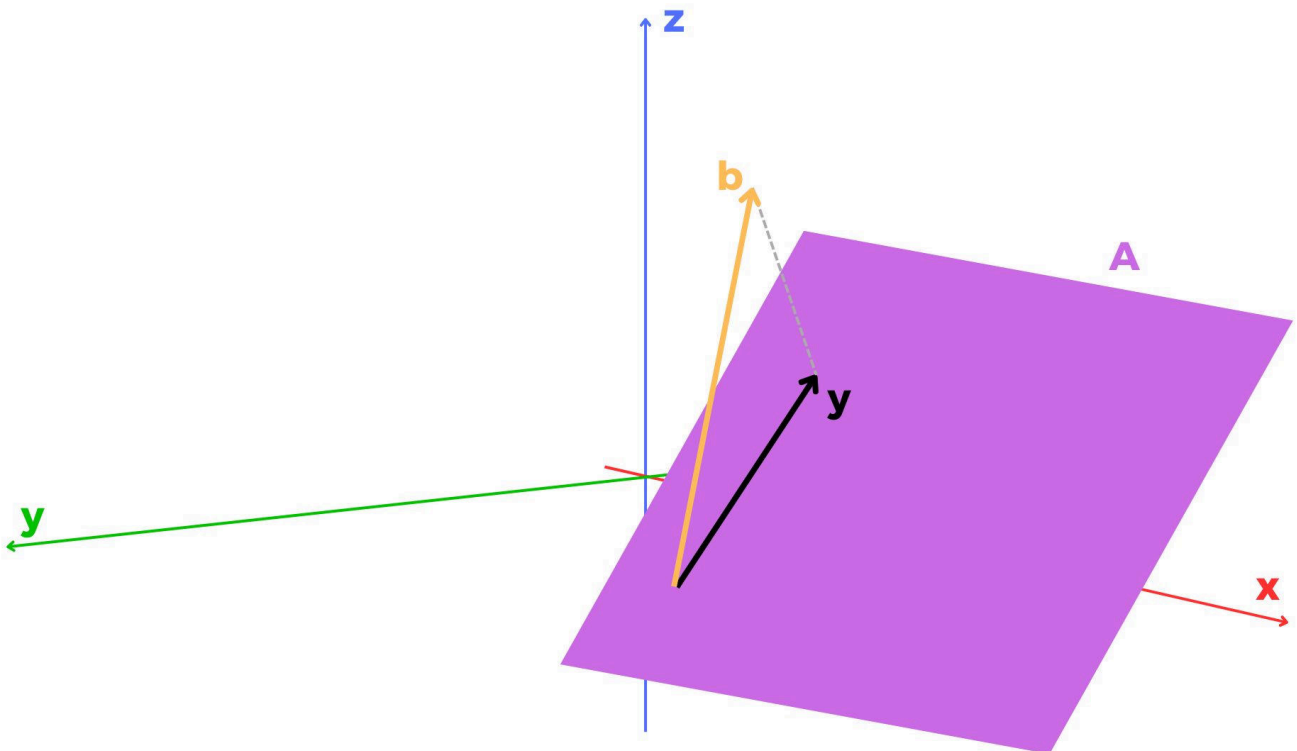
No resumo passado, vimos que o  $x$  que satisfaz esse problema é

$$x = (A^*A)^{-1}A^*b \Rightarrow y = A(A^*A)^{-1}A^*b \Leftrightarrow y = Pb \quad (40)$$

Ou seja, a projeção ortogonal de  $b$  em  $A$  resulta no vetor  $y$ . Queremos então saber o condicionamento de (39) de acordo com perturbações em  $b$ ,  $A$ ,  $y$  e  $x$ . Tenha em mente que o problema recebe dois parâmetros,  $A$  e  $b$  e retorna as soluções  $x$  e  $y$

### 3.1 O Teorema

Antes de estabelecer de fato o teorema, vamos relembrar alguns fatores-chave aqui. Vamos rever a imagem que representa o problema de mínimos quadrados visualmente (Mesma imagem do resumo anterior)



Vamos relembrar algumas coisas que já vimos antes e algumas novas. Primeiro é lembrar que, como  $A$  não é quadrada, definimos seu número de condicionamento como

$$\kappa(A) = \|A\| \|A^+\| = \|A\| \|(A^*A)^{-1}A^*\| \quad (41)$$

Não está explícito na imagem, mas podemos, também, definir o ângulo  $\theta$  entre  $b$  e  $y$

$$\theta = \arccos\left(\frac{\|y\|}{\|b\|}\right) \quad (42)$$

(A gente define assim pois  $b$  é a hipotenusa do triângulo retângulo formado por  $b$  e  $y - b$ )

E a segunda medida é  $\eta$ , que representa por quanto  $y$  não atinge seu valor máximo

$$\eta = \frac{\|A\| \|x\|}{\|y\|} = \frac{\|A\| \|x\|}{\|Ax\|} \quad (43)$$

Show! E esses parâmetros tem esses domínios:

$$\kappa(A) \in [1, \infty] \quad \theta \in \left[0, \frac{\pi}{2}\right] \quad \eta \in [1, \kappa(A)] \quad (44)$$

**Teorema 3.1.1** (Condicionamento de Mínimos Quadrados): Deixe  $b \in \mathbb{C}^m$  e  $A \in \mathbb{C}^{m \times n}$  de posto completo serem **fixos**. O problema de mínimos quadrados (39) possui a seguinte tabela de condicionamentos em norma-2:

	$y$	$x$
$b$	$\frac{1}{\cos(\theta)}$	$\frac{\kappa(A)}{\eta \cos(\theta)}$
$A$	$\frac{\kappa(A)}{\cos(\theta)}$	$\kappa(A) + \frac{\kappa(A)^2 \tan(\theta)}{\eta}$

Figura 1: Sensibilidade de  $x$  e  $y$  com relação a perturbações em  $A$  e  $b$

Vale dizer também que a primeira linha são igualdades exatas, enquanto a linha de baixo são arredondamentos para cima

*Demonstração:* Antes de provar para cada tipo de perturbação, temos em mente que estamos trabalhando com a norma-2, correto? Então nós vamos reescrever  $A$  para ter uma análise mais fácil. Seja  $A = U\Sigma V^*$  a decomposição S.V.D de  $A$ , sabemos que  $\|A\|_2 = \|\Sigma\|_2$  (As matrizes unitárias não afetam a norma), então podemos, sem perda da generalidade, lidar diretamente com  $\Sigma$ , então podemos assumir que  $A = \Sigma$  (Não literalmente, mas como vamos ficar analisando as normas, isso vai nos facilitar bastante)

$$A = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \end{pmatrix} = \begin{pmatrix} A_1 \\ 0 \end{pmatrix} \quad (45)$$

Reescrevendo os outros termos, temos:

$$b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \quad y = \begin{pmatrix} b_1 \\ 0 \end{pmatrix} \quad \begin{pmatrix} A_1 \\ 0 \end{pmatrix} x = \begin{pmatrix} b_1 \\ 0 \end{pmatrix} \Leftrightarrow x = A_1^{-1} b_1 \quad (46)$$

- **Sensibilidade de  $y$  com perturbações em  $b$ :** Vimos anteriormente na equação (40) que  $y = Pb$ , e podemos tirar o condicionamento disso se associarmos com a equação **genérica**  $Ax = b$ . Lembra que em estabilidade vimos que o condicionamento desse sistema genérico quando perturbamos  $x$  é:

$$\frac{\|A\|}{\|x\|/\|b\|} \quad (47)$$

Então, fazendo simples substituições:

$$\frac{\|P\|}{\|y\|/\|b\|} = \frac{1}{\cos \theta} \quad (48)$$

O que até que faz sentido na intuição. Se fazemos com que  $b$  fique muito próximo a um ângulo de  $90^\circ$  com  $C(A)$ , na hora que formos projetar, a projeção será minúscula, o que pode acarretar erros numéricos dependendo da precisão usada pelo computador

- **Sensibilidade de  $x$  com perturbações em  $b$ :** Também tem uma relação bem direta pela equação (40):  $x = A^+ b$ . Assim, temos o mesmo de antes:



$$\frac{\|A^+\|}{\|x\|/\|b\|} = \|A^+\| \frac{\|b\|}{\|y\|} \frac{\|y\|}{\|x\|} = \|A^+\| \frac{1}{\cos \theta} \frac{\|A\|}{\eta} = \frac{\kappa(A)}{\eta \cos \theta} \quad (49)$$

Antes de continuar o resto da demonstração, temos que entender um pouco como as perturbações em  $A$  podem afetar  $C(A)$ , porém, isso é um problema não-linear. Até daria pra fazer um monte de jacobiano algébrico, mas é melhor se manter numa pegada não muito formal e ter uma visão geométrica.

Primeiro, quando perturbamos  $A$ , isso afeta o problema de mínimos quadrados de dois modos: 1 - As perturbações afetam como vetores em  $\mathbb{C}^n$  ( $A \in \mathbb{C}^{m \times n}$ ) são mapeados em  $C(A)$ . 2 - Elas alteram  $C(A)$  em si. A gente pode imaginar as perturbações em  $C(A)$  como pequenas inclinações que a gente faz, coisa bem pouquinho mesmo. Então fazemos a pergunta: Qual é o maior ângulo de inclinação  $\delta\alpha$  (O quão inclinado eu deixei em comparação a como tava antes) que pode ser causado por uma pequena perturbação  $\delta A$ ? Aí a gente pode seguir do seguinte modo:

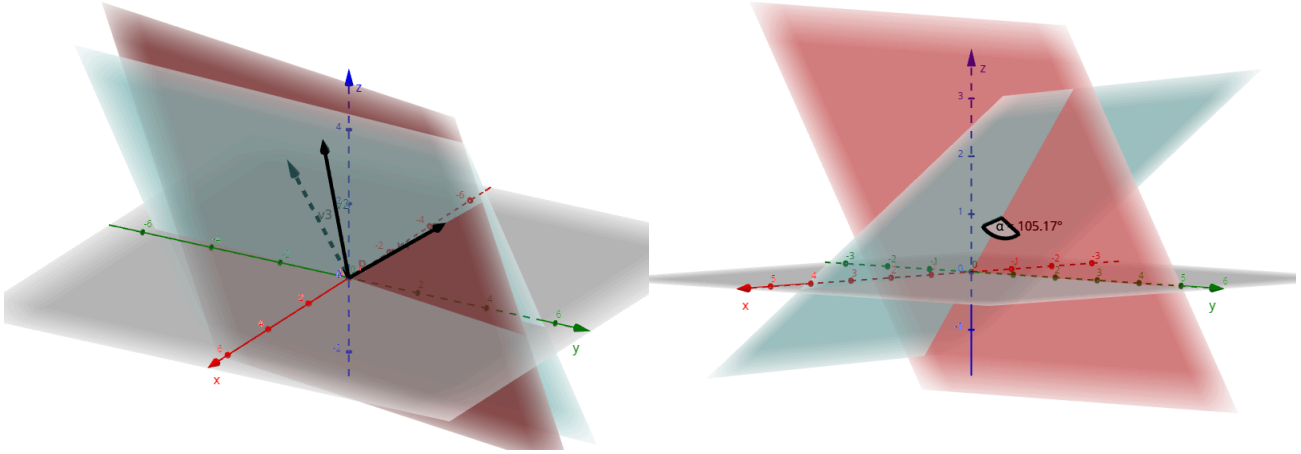


Figura 1: Perturbação em  $C(A)$ .  $v_1$  é o vetor que está na divisão entre o plano azul e o vermelho,  $v_2$  é o vetor mais destacado no plano azul e  $v_3$  é o vetor pontilhado

Na Figura 1, a gente consegue ver isso um pouco melhor. Nosso plano original é o **azul**, formado por  $v_1$  e  $v_2$ , enquanto o plano **vermelho** é formado por  $v_1$  e  $v_3$ , onde  $v_3$  é o  $v_2 + \delta v_2$ . Percebam que os planos tem uma abertura entre si, medimos aquela abertura por meio de  $\delta\alpha$  que mostra a diferença de inclinação entre os dois planos. A segunda mostra mais explicitamente esse ângulo aplicado a outros dois planos diferentes, eu aumentei a diferença entre um e outro apenas para ilustrar melhor a visualização do ângulo, mas normalmente queremos trabalhar com ângulos minúsculos.

Quando a gente projeta uma  $n$ -esfera unitária em  $C(A)$ , temos uma hiperelipse. Pra mudar  $C(A)$  da forma mais eficiente possível, pegamos um ponto  $p = Av$  que está na hiperelipse ( $\|v\| = 1$ ) e cutucamos ela em uma direção  $\delta p$  ortogonal a  $C(A)$ . A perturbação que melhor faz isso é  $\delta A = (\delta p)v^*$ , que resulta em  $(\delta A)v = \delta p \Rightarrow \|\delta A\| = \|\delta p\|$ . Essa perturbação é a melhor por conta da norma 2 de um produto externo:

$$A = uv^* \Rightarrow \|Ax\| = \|uv^*x\| \leq \|u\|\|v\|\|x\| \quad (50)$$

Daí **para ter a igualdade**, basta pegar  $x = v$ . Agora a gente pode perceber que, se a gente quer a maior inclinação possível dado uma perturbação  $\|\delta p\|$  a gente tem q fazer com que  $p$  fique perto da origem o máximo possível. Ou seja, queremos o menor  $p$  possível com base na definição, que seria  $p = \sigma_n u_n$  onde  $\sigma_n$  é o menor valor singular de  $A$  e  $u_n$  a  $n$ -ésima coluna de  $U$ . Se tomarmos  $A = \Sigma$ ,  $p$  é a última coluna de  $A$ ,  $v^* = e_n^* = (0, 0, \dots, 1)$  e  $\delta A$  são perturbações na entrada de  $A$ . Essa perturbação inclina  $C(A)$  pelo ângulo  $\delta A$  dado por  $\tan(\delta\alpha) = \|\delta p\|/\|\sigma_n\|$ , temos então:

$$\delta\alpha \leq \frac{\|\delta A\|}{\sigma_n} = \frac{\|\delta A\|}{\|A\|} \kappa(A) \quad (51)$$

Agora sim podemos continuar a demonstração

- **Sensibilidade de  $y$  com perturbações em  $A$ :** Podemos ver uma propriedades geométricas interessantes quando fixamos  $b$  e mexemos  $A$ . Lembra que  $y$  é a projeção **ortogonal** de  $b$  em  $C(A)$ , ou seja,  $y$  sempre é **ortogonal** a  $y - b$ .

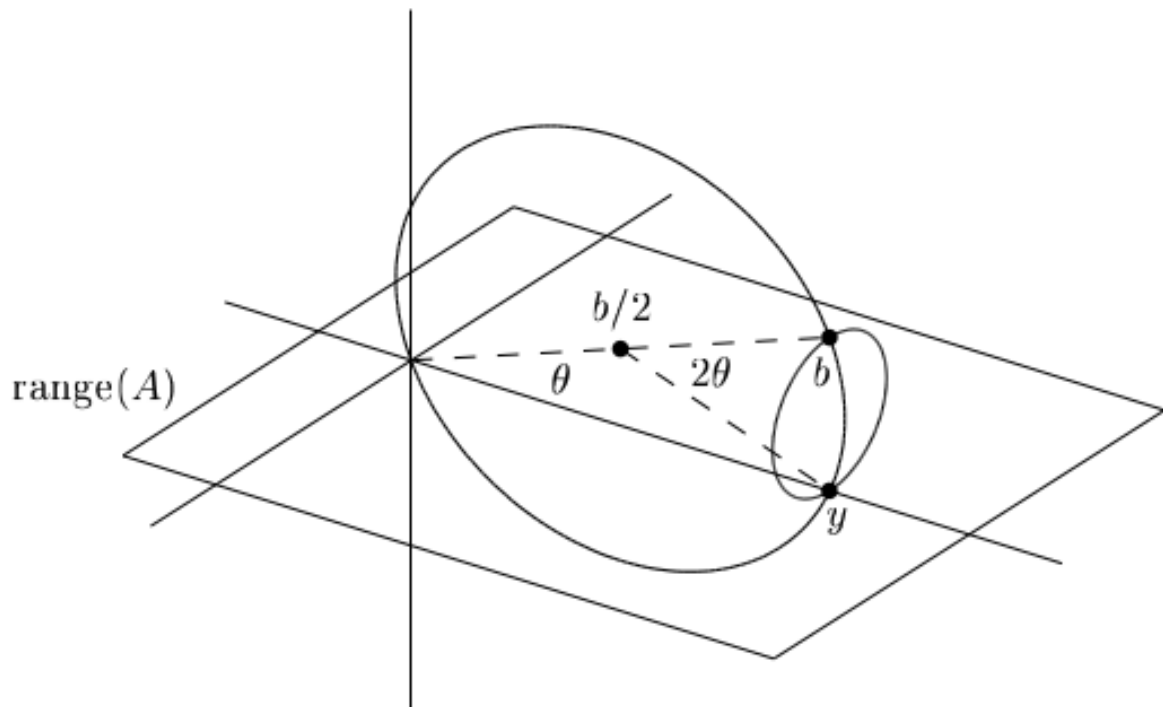


Figura 2: Círculo de projeção de  $y$ . O círculo maior representa a inclinação de  $C(A)$  no plano  $0yb$  e o círculo menor é quando inclinamos  $C(A)$  em uma direção ortogonal a ele

Como eu posso rotacionar  $C(A)$  em  $360^\circ$ , eu posso visualizar todos os possíveis locais de  $y$  estando nessa esfera. Quando eu inclino  $C(A)$  por um ângulo  $\delta\alpha$  no círculo maior, o meu ângulo  $2\theta$  vai ser alterado. Mais especificamente, vai ser alterado em  $2\delta\alpha$ . Ou seja, a perturbação  $\delta y$  que eu vou obter ao inclinar  $C(A)$  será a base de um triângulo isósceles.

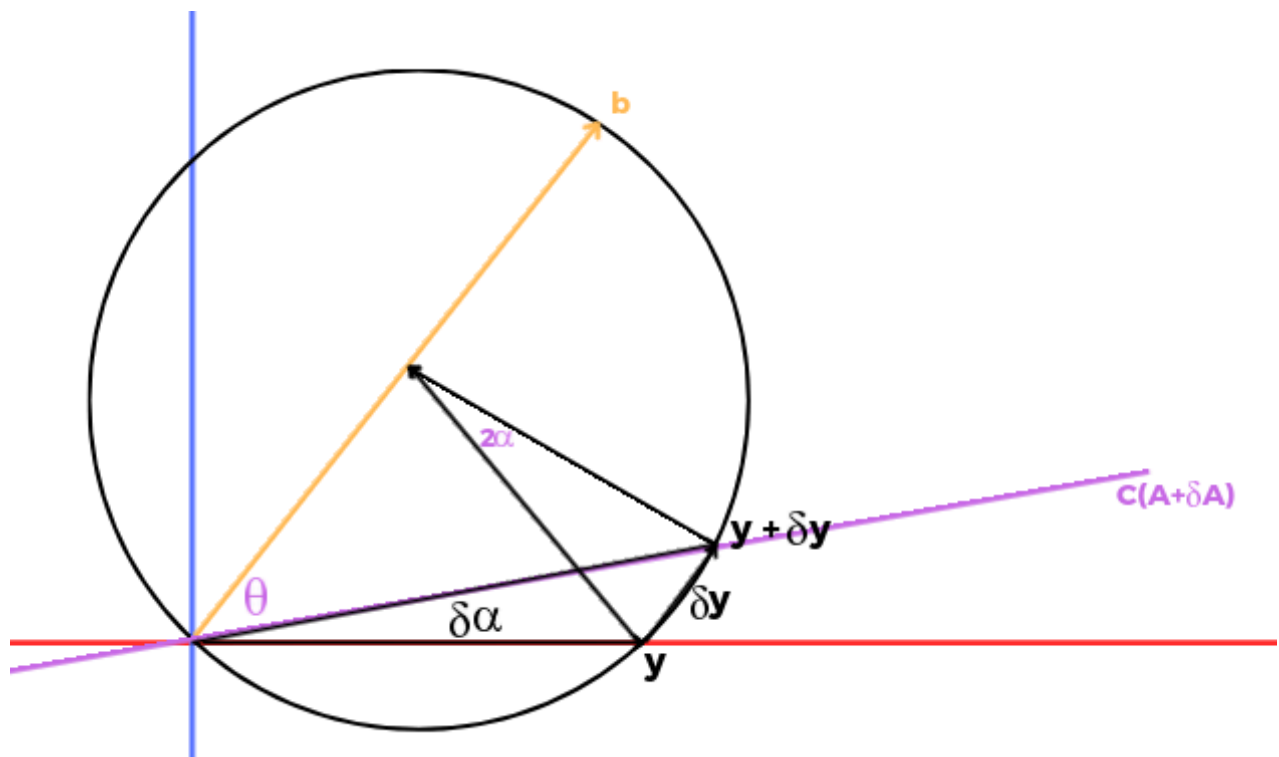


Figura 3:  $C(A)$  após rotação de  $\delta\alpha$

Podemos ver que o raio da esfera é  $\|b\|/2$ , ou seja, podemos chegar que:

$$\begin{aligned}
\|\delta y\| &\leq \|b\| \sin(\delta\alpha) \leq \|b\|(\delta\alpha) \leq \|b\| \frac{\|\delta A\|}{\|A\|} \kappa(A) \\
\cos(\theta) &= \frac{\|y\|}{\|b\|} \Leftrightarrow \|b\| = \frac{\|y\|}{\cos(\theta)} \\
\Rightarrow \|\delta y\| &\leq \frac{\|y\|}{\cos(\theta)} \frac{\|\delta A\|}{\|A\|} \kappa(A) \Leftrightarrow \frac{\frac{\|\delta y\|}{\|y\|}}{\frac{\|\delta A\|}{\|A\|}} = \frac{\kappa(A)}{\cos(\theta)}
\end{aligned} \tag{52}$$

Concluimos assim, o 3º condicionamento

- **Sensibilidade de  $x$  com perturbações em  $A$ :** Quando a gente faz uma perturbação  $\delta A$  em  $A$ , podemos separar essa perturbação em duas outras:  $\delta A_1$  que ocorre nas primeiras  $n$  linhas de  $A$  e  $\delta A_2$  que ocorre nas  $m - n$  linhas restantes.

$$A = \begin{pmatrix} \delta A_1 \\ \delta A_2 \end{pmatrix} \tag{53}$$

Vamos ver  $\delta A_1$  primeiro. Quando vemos essa perturbação específica, pelo que vimos em (46), temos que  $b$  não é alterado, então estamos mantendo  $b$  fixo e tentando calcular  $x$  com perturbação  $\delta A_1$  em  $A$ . Esse condicionamento já vimos no último resumo:

$$\left( \frac{\|\delta x\|}{\|x\|} \right) / \left( \frac{\|\delta A_1\|}{\|A\|} \right) \leq \kappa(A_1) = \kappa(A) \tag{54}$$

Já quando perturbamos por  $\delta A_2$  (Estamos perturbando  $C(A)$  por inteiro, não somente  $A_2$ ), acaba que o vetor  $y$  e, conseqüentemente, o vetor  $b_1$  são perturbados, porém, sem perturbação em  $A_1$ . Isso é a mesma coisa que a gente perturbar  $b_1$  sem perturbar  $A_1$ . O condicionamento disso é:

$$\left( \frac{\|\delta x\|}{\|x\|} \right) / \left( \frac{\|\delta b_1\|}{\|b_1\|} \right) \leq \frac{\kappa(A_1)}{\eta(A_1; x)} = \frac{\kappa(A)}{\eta} \tag{55}$$

Agora precisamos relacionar  $\delta b_1$  com  $\delta A_2$ . Sabemos que  $b_1$  é  $y$  expresso nas coordenadas de  $C(A)$ . Ou seja, as únicas mudanças em  $y$  que podem ser vistas como mudanças em  $b_1$  são aquelas paralelas a  $C(A)$ . Se  $C(A)$  é inclinado por um ângulo  $\delta\alpha$  no plano  $Ob_1y$ ,  $\delta y$  não está em  $C(A)$ , mas tem um ângulo de  $\frac{\pi}{2} - \theta$ . Ou seja, as mudanças em  $b_1$  satisfazem:

$$\|\delta b_1\| = \sin(\theta) \|\delta y\| \leq (\|b\| \delta\alpha) \sin(\theta) \tag{56}$$

Curiosamente se a gente inclina  $C(A)$  na direção ortogonal ao plano  $Ob_1y$  (Círculo menor na Figura 2) obtemos o mesmo resultado por motivos diferentes.

Como vimos antes:  $\cos(\theta) = \|y\|/\|b\| \Leftrightarrow \|b_1\| = \cos(\theta)\|b\|$ , então podemos reescrever (56) como:

$$\frac{\|\delta b_1\|}{\|b_1\|} \leq \frac{\|b\| \delta\alpha \sin(\theta)}{\|b\| \cos(\theta)} \Leftrightarrow \frac{\|\delta b_1\|}{\|b_1\|} \leq \delta\alpha \tan(\theta) \tag{57}$$

Assim, podemos relacionar  $\delta\alpha$  com  $\|\delta A_2\|$  da equação (51)

$$\begin{aligned}
\delta\alpha &\leq \frac{\|\delta A_2\|}{\|A\|} \kappa(A) \Leftrightarrow \frac{\|\delta b_1\|}{\|b_1\|} \leq \frac{\|\delta A_2\|}{\|A\|} \kappa(A) \tan(\theta) \\
\left( \frac{\|\delta x\|}{\|x\|} \right) / \left( \frac{\|\delta b_1\|}{\|b_1\|} \right) &\leq \frac{\kappa(A)}{\eta} \Leftrightarrow \frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{\eta} \frac{\|\delta b_1\|}{\|b_1\|} \Leftrightarrow \frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{\eta} \frac{\|\delta A_2\|}{\|A\|} \kappa(A) \tan(\theta) \\
&\Leftrightarrow \left( \frac{\|\delta x\|}{\|x\|} \right) / \left( \frac{\|\delta A_2\|}{\|A\|} \right) \leq \frac{\kappa(A)^2 \tan(\theta)}{\eta}
\end{aligned} \tag{58}$$

Combinando os condicionamentos de  $A_1$  e  $A_2$  temos  $\kappa(A) + \frac{\kappa(A)^2 \tan(\theta)}{\eta}$

□

## **4 Lecture 19 - Estabilidade de Algoritmos de Mínimos Quadrados**

A gente viu quem tem um monte de jeito de se resolver os problemas de mínimos quadrados (Resumo 1). Com isso, a gente pode calcular e estimar a estabilidade dos algoritmos que já vimos.

## 4.1 Primeira Etapa

Vamos fazer isso na prática. Vamos montar um cenário para a aplicação de cada um dos algoritmos. Vamos pegar  $m$  pontos igualmente espaçados entre 0 e 1, montamos a matriz de vandermonde desses pontos e aplicamos uma função que tentaremos prever com polinômios:

CÓDIGO	Python
1	<code>import numpy as np</code>
2	<code>m = 100</code>
3	<code>n = 15</code>
4	<code>t = np.linspace(0, 1, m)</code>
5	<code>A = np.vander(t, n, True)</code>
6	<code>b = np.exp(np.sin(4*t))/2.00678728e+03</code>

Oxe, por que que tem essa divisão esquisita no final? Quando a gente não faz essa divisão, ao fazer a previsão dos coeficientes que aproximam a função, temos que o último coeficiente previsto ( $x_{15}$ ) é igual a  $2.00678728e+03$ , então, nós dividimos  $b$  por esse valor para que o último coeficiente seja igual a 1 no caso matematicamente correto (Sem erros numéricos), assim poderemos fazer comparações apenas visualizando o último número dos coeficientes calculados.

## 4.2 Householder

O algoritmo padrão para problemas de mínimos quadrados. Vejamos:

CÓDIGO	Python	SAÍDA
7	<code>Q, R = householder_qr(A)</code>	1
8	<code>x = np.linalg.solve(R, Q.T @ b)</code>	1.9845992627054443e-09
9	<code>print(1-x[-1]) # Erro relativo</code>	

Temos um erro de grandeza  $10^9$ , porém, no Python, trabalhamos com precisão IEEE 754 ( $\varepsilon = 2.220446049250313e-16$ ), o que nos mostra um erro de precisão MUITO grande (Ordem de  $10^7$  de diferença). Porém, aqui nós calculamos  $Q$  explicitamente e, no resumo 1, foi comentado que isso normalmente não acontece, então vamos ver se o erro muda ao trocarmos  $Q$  por uma versão implícita

CÓDIGO	Python	SAÍDA
7	<code>Q, R = householder_qr(np.c_[A, b])</code>	1
8	<code>print(R.shape)</code>	1.989168163518684e-09
9	<code>Qb = R[0:n, n]</code>	
10	<code>R = R[0:n, 0:n]</code>	
11	<code>x = np.linalg.solve(R, Qb)</code>	
12	<code>print(1-x[-1])</code>	

Deu pra ver que da quase a mesma coisa do resultado anterior, ou seja, os erros da fatoração de  $A$  são maiores que os de  $Q$ . Pode ser provado que essas duas variações são **backward stable**. O mesmo vale para uma terceira variação que utiliza do **pivotamento** de colunas (Não é discutido nem no livro, tampouco nesse resumo)

**Teorema 4.2.1:** Deixe um problema de mínimos quadrados em uma matriz de posto completo  $A$  ser resolvida por fatoração **Householder** em um computador ideal. O algoritmo é **backward stable** tal que:

$$\|(A + \delta A)\tilde{x} - b\| = \min, \quad \frac{\|\delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \quad (59)$$

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

### 4.3 Ortogonalização de Gram-Schmidt

A gente também pode tentar resolver pelo método de Gram-Schmidt modificado, vamos ver o que a gente consegue:

CÓDIGO	Python
7	<code>Q, R = modified_gram_schmidt(A)</code>
8	<code>x = np.linalg.solve(R, Q.T @ b)</code>
9	<code>print(1-x[-1])</code>

SAÍDA
1    -0.01726542

Meu amigo, esse erro é **terrível**. O resultado obtido é tenebroso de ruim. O livro comenta também de outro método que envolve fazer umas manipulações em  $Q$ , mas como o próprio diz que envolve trabalho extra, desnecessário e não deveria ser usado na prática, nem vou comentar sobre aqui.

Mas a gente pode usar um método parecido com o que fizemos antes em unir  $A$  e  $b$  numa única matriz:

CÓDIGO	Python
7	<code>Q, R = modified_gram_schmidt(np.c_[A, b])</code>
8	<code>Qb = R[0:n, n]</code>
9	<code>R = R[0:n, 0:n]</code>
10	<code>x = np.linalg.solve(R, Qb)</code>
11	<code>print(1-x[-1])</code>

SAÍDA
1    -1.3274502852489434e-07

Olha só! Já deu uma melhorada no algoritmo!

**Teorema 4.3.1:** Solucionar o problema de mínimos quadrados de uma matriz  $A$  com posto completo utilizando o algoritmo de Gram-Schmidt (Fazendo de acordo como o código anterior mostra em que  $Q^*b$  é implícito) é **backward stable**

### 4.4 Equações Normais

A gente pode resolver por equações normais, que é o passo inicial para todos os outros métodos né? Vamos ver o que obtemos:

CÓDIGO	Python
7	<code>x = np.linalg.solve(A.T @ A, A.T @ b)</code>
8	<code>print(1-x[-1])</code>

SAÍDA
1    1.35207472

Meu amigo, esse erro é **TENEBROSO**, não chegou nem **PERTO** do resultado. Claramente as equações normais são um método **instável** de calcular mínimos quadrados. Vamos dar uma visualizada no porquê isso ocorre:

Suponha que nós temos um algoritmo **backward stable** para o problema de mínimos quadrados com uma matriz  $A$  de posto-completo que retorna uma solução  $\tilde{x}$  satisfazendo  $\|(A + \delta A)\tilde{x} - b\| = \min$  para algum  $\delta A$  com  $\|\delta A\|/\|A\| = O(\varepsilon_{\text{machine}})$ . Pelo teorema da acurácia de algoritmos backward stable (Resumo 1) e o Teorema 3.1.1 temos:

$$\frac{\|\tilde{x} - x\|}{\|x\|} = O\left(\left(\kappa + \frac{\kappa^2 \tan(\theta)}{\eta}\right) \varepsilon_{\text{machine}}\right) \quad (60)$$

Suponha que  $A$  é mal-condicionada. Dependendo dos valores dos hiperparâmetros, podem acontecer duas situações diferentes. Se  $\tan(\theta)$  for de ordem 1, então o lado direito da equação (60) troca e fica  $O(\kappa^2 \varepsilon_{\text{machine}})$ . Porém, se  $\tan(\theta)$  é próximo de 0, ou  $\eta$  é próximo de  $\kappa$ , então a equação muda para  $O(\kappa \varepsilon_{\text{machine}})$  (Usa um teorema mais la pra frente, mas é engraçado ver como tudo tá muito interconectado). Porém, a matriz  $A^*A$  tem número de condicionamento  $\kappa(A)^2$ , então o máximo que podemos esperar do problema é  $O(\kappa^2 \varepsilon_{\text{machine}})$

**Teorema 4.4.1:** A solução de um problema de mínimos quadrados com uma matriz  $A$  de posto-completo utilizando de equações normais é **instável**. Porém a estabilidade pode ser alcançada ao restringir para uma classe de problemas onde  $\kappa(A)$  é pequeno ou  $\frac{\tan(\theta)}{\eta}$  é pequeno.

## 4.5 SVD

O último algoritmo a ser mencionado foi utilizando a SVD de  $A$ , que nós vimos (no resumo 1) que parecia ser um algoritmo interessante:

CÓDIGO	Python	SAÍDA
7	<code>U, S, Vh = np.linalg.svd(A, full_matrices=False)</code>	1
8	<code>S = np.diag(S)</code>	-2.3301211e-07
9	<code>x = (Vh.T * 1/S) @ (U.T @ b)</code>	
10	<code>print(1-x[-1])</code>	

Olha só! Temos uma precisão ótima! (O algoritmo da SVD é o mais confiável e estável, mesmo que o erro mostrado seja maior do que alguns que obtivemos anteriormente)

**Teorema 4.5.1:** A solução do problema de mínimos quadrados com uma matriz  $A$  de posto-completo utilizando o algoritmo de SVD é **backward stable**.

## 4.6 Problemas de Mínimos Quadrados com Posto-Incompleto

A gente viu a aplicação de algoritmos em problemas de mínimos quadrados utilizando matrizes de posto-completo, mas pode ter outros casos de matrizes com posto  $< n$ , ou até  $m < n$ . Para essa classe de problemas, é necessário definirmos outro tipo de solução, já que nem todos tem o mesmo comportamento. As vezes precisamos restringir a solução com uma condição. Por conta disso, nem todo algoritmo que vimos ser estável até agora vai ser estável nesse tipo de problema, na verdade, apenas o de SVD será e o de Gram-Schmidt com pivotamento nas colunas.

## **5 Lecture 24 - Problemas de Autovalores**



**Nota:** Esse capítulo é uma revisão bem superficial sobre autovalores e autovetores, se quiser uma visão mais aprofundada sobre o tema, leia os resumos sobre Álgebra Linear do segundo período (Se já estiverem disponíveis)

## 5.1 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  $\lambda x$  e  $Ax$  estiverem em conjuntos distintos. Com isso, prosseguimos com a definição:

**Definição 5.1.1** (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:

$$Ax = \lambda x \quad (61)$$

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

## 5.2 Decomposição em Autovalores

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

$$A = X\Lambda X^{-1} \quad (62)$$

Onde  $\Lambda$  é diagonal e  $\det(X) \neq 0$ .

Isso é equivalente a:

$$\underbrace{\begin{pmatrix} & & & \\ & A & & \\ & & & \end{pmatrix}}_A \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & \dots & x_n \\ | & | & | & | \end{pmatrix}}_X = \underbrace{\begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \end{pmatrix}}_\Lambda \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & \dots & x_n \\ | & | & | & | \end{pmatrix}}_X \quad (63)$$

Da (63) e da Definição 5.1.1, decorre que  $Ax_i = \lambda_i x_i$ , então a  $i$ -ésima coluna de  $X$  é um autovetor de  $A$  e  $\lambda_i$  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:

$$Ax = b \Leftrightarrow X\Lambda X^{-1}x = b \Leftrightarrow \Lambda(X^{-1}x) = X^{-1}b \quad (64)$$

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

## 5.3 Multiplicidades Algébrica e Geométrica

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

**Definição 5.3.1** (Autoespaço): Dada  $A \in \mathbb{C}^{m \times n}$ ,  $\lambda \in \mathbb{C}$ , 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  $\lambda$ , e chamaremos isso de **multiplicidade geométrica** de  $\lambda$ . Então temos:

**Definição 5.3.2:** (Multiplicidade Geométrica) A multiplicidade geométrica de  $\lambda$  é  $\dim(S_\lambda)$

Note que da equação (61):

$$Ax = \lambda x \Leftrightarrow Ax - \lambda x = 0 \Leftrightarrow (A - \lambda I)x = 0 \quad (65)$$

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

$$\det(A - \lambda I) = 0 \quad (66)$$

**Definição 5.3.3** (Polinômio Característico): A equação (66) se chama **polinômio característico** de  $A$  e é um polinômio de grau  $m$  em  $\lambda$ . Pelo teorema fundamental da Álgebra, se  $\lambda_1, \dots, \lambda_n$  são raízes de (66), então podemos escrever isso como:

$$p(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2) \dots (\lambda - \lambda_n) \quad (67)$$

(Nota:  $\lambda$  é uma variável, enquanto  $\lambda_j$  é uma raiz do polinômio, fique atento)

Com isso, prosseguimos com:

**Definição 5.3.4** (Multiplicidade Algébrica): A multiplicidade algébrica de  $\lambda$  é a multiplicidade de  $\lambda$  como raiz do polinômio característico de  $A$

A definição de polinômio característico e de multiplicidade algébrica faz a gente ter um jeito muito fácil de contar a quantidade de autovalores de uma matriz

**Teorema 5.3.1:** Se  $A \in \mathbb{C}^{m \times m}$ , então  $A$  tem  $m$  autovalores, contando com a multiplicidade algébrica.

Isso mostra que **toda matriz** possui **pelo menos 1** autovalor

## 5.4 Transformações Similares

**Definição 5.4.1** (Transformação Similar): Se  $X \in \mathbb{C}^{m \times m}$  é inversível, então o mapeamento  $A \mapsto X^{-1}AX$  é chamado de **transformação similar** de  $A$ .

Dizemos que duas matrizes  $A$  e  $B$  são **similares** se existe uma matriz inversível  $X$  que relacione as transformações similares entre  $A$  e  $B$ , i.e:

$$X^{-1}AX = X^{-1}BX \quad (68)$$

**Teorema 5.4.1:** Se  $A \in \mathbb{C}^{m \times m}$  é inversível, então  $A$  e  $X^{-1}AX$  o mesmo polinômio característico, os mesmos autovalores e multiplicidades geométrica e algébrica.

*Demonstração:*

$$\begin{aligned} p_{X^{-1}AX}(z) &= \det(zI - X^{-1}AX) = \det(X^{-1}(zI - A)X) \\ &= \det(X^{-1}) \det(zI - A) \det(X) = \det(zI - A) = p_{A(z)} \end{aligned} \quad (69)$$

Suponha que  $E_\lambda$  é o autoespaço de  $A$ , então  $X^{-1}E_\lambda$  é autoespaço de  $X^{-1}AX$ , ou seja, ambos tem mesma multiplicidade geométrica  $\square$

Agora podemos correlacionar a multiplicidade geométrica e a algébrica

**Teorema 5.4.2:** A multiplicidade algébrica de um autovalor  $\lambda$  é sempre maior ou igual a sua multiplicidade geométrica

*Demonstração:* Deixe  $n$  ser a multiplicidade geométrica de  $\lambda$  para a matriz  $A$ . Forme uma matriz  $\hat{V} \in \mathbb{C}^{m \times n}$  de tal forma que as suas  $n$  colunas formam uma base ortonormal do autoespaço  $\{x : Ax = \lambda x\}$ . Se extendermos  $\hat{V}$  para uma matriz ortogonal quadrada, temos:

$$B = V^*AV = \begin{pmatrix} \lambda I & C \\ 0 & D \end{pmatrix} \quad (70)$$

Pela definição e propriedades do determinante (Não cabe mostrá-las aqui), temos que:

$$\det(\mu I - B) = \det(\mu I - \lambda I) \det(\mu I - D) = (\mu - \lambda)^n \det(\mu I - D) \quad (71)$$

Ou seja, a multiplicidade algébrica de  $\lambda$  como um autovalor de  $B$  é, no mínimo,  $n$ . Como transformações similares mantêm a multiplicidade, o mesmo vale para  $A$   $\square$

## 5.5 Autovalores e Matrizes Deficientes

Um autovalor é deficiente quando sua MA é maior que sua MG. Se uma matriz  $A$  tem autovalor deficiente, ela é uma matriz deficiente. Matrizes deficientes não podem ser diagonalizáveis (Próximo tópico)

## 5.6 Diagonalizabilidade

**Teorema 5.6.1** (Diagonalizabilidade): Uma matriz  $A \in \mathbb{C}^{m \times m}$  é não-deficiente  $\Leftrightarrow$  ela tem uma decomposição  $A = X\Lambda X^{-1}$

*Demonstração:*  $\Leftarrow$ ) Dada uma decomposição  $A = X\Lambda X^{-1}$ , sabemos, pelo Teorema 5.4.1, que  $\Lambda$  sendo similar a  $A$ , logo,  $A$  tem os mesmos autovalores, MA e MG de  $\Lambda$ . Como  $\Lambda$  é diagonal, eu tenho que  $\Lambda$  é não-deficiente, logo, o mesmo vale para  $A$

$\Rightarrow$ ) Uma matriz não-deficiente deve ter  $m$  autovetores linearmente independentes, pois autovetores com diferentes autovalores precisam ser L.I, e cada autovalor pode se associar com autovetores a quantidade de vezes que sua MA permitir. Se esses  $m$  autovetores independentes formam as colunas de uma matriz  $X$ , então  $X$  é inversível e  $A = X\Lambda X^{-1}$   $\square$

## 5.7 Determinante e Traço

**Teorema 5.7.1:** Seja  $\lambda_j$  um autovalor de  $A \in \mathbb{C}^{m \times m}$ :

$$\begin{aligned} \det(A) &= \prod_{j=1}^m \lambda_j \\ \text{tr}(A) &= \sum_{j=1}^m \lambda_j \end{aligned} \quad (72)$$

*Demonstração:*

$$\det(A) = (-1)^m \det(-A) = (-1)^m p_{A(0)} = \prod_{j=1}^m \lambda_j \quad (73)$$

Olhando a equação (67), podemos observar que o coeficiente do termo  $\lambda^{m-1}$  é igual a  $-\sum_{j=1}^m \lambda_j$  e na equação (66) o termo é o negativo da soma dos termos da diagonal, ou seja,  $-\text{tr}(A)$ , ou seja,  $\text{tr}(A) = \sum_{j=1}^m \lambda_j$   $\square$

## 5.8 Diagonalização Unitária

Acontece as vezes que, ao fazer a diagonalização de uma matriz, nós podemos cair com um conjunto de autovetores ortogonais entre si.

**Definição 5.8.1:**  $A$  é diagonalizável unitariamente quando  $A = Q\Lambda Q^*$  com  $Q$  ortogonal e  $\Lambda$  diagonal (Pode ter entradas complexas)

**Teorema 5.8.1** (Teorema Espectral): Uma matriz hermitiana é diagonalizável unitariamente e seus autovalores são reais.

Não cabe aqui a prova desse teorema, porém um resumo de Álgebra Linear do 2º período será feito e essa demonstração estará lá.

**Definição 5.8.2** (Matrizes Normais): Uma matriz  $A$  é normal se  $A^*A = AA^*$

**Teorema 5.8.2:** Uma matriz é diagonalizável unitariamente  $\Leftrightarrow$  ela é normal

## 5.9 Forma de Schur

Essa forma é **muito útil** em análise numérica tendo em vista que **toda matriz** pode ser fatorada assim

**Definição 5.9.1** (Fatoração de Schur): Dada uma matriz  $A \in \mathbb{C}^{m \times m}$ , sua fatoração de schur é tal que:

$$A = QTQ^* \quad (74)$$

onde  $Q$  é ortogonal e  $T$  é triangular superior

**Teorema 5.9.1:** Toda matriz quadrada  $A$  tem uma fatoração de Schur

*Demonstração:* Vamos fazer indução em  $m$ .

- **Casos base:**  $m = 1$  é trivial, então suponha que  $m \geq 2$ .
- **Passo Indutivo:** Deixe  $x$  ser um autovetor de  $A$  com autovalor  $\lambda$ . Normalize  $x$  e faça com que seja a primeira coluna de uma matriz ortogonal  $U$ . Então podemos fazer as contas e conferir que o produto  $U^*AU$  é tal que:

$$U^*AU = \begin{pmatrix} \lambda & B \\ 0 & C \end{pmatrix} \quad (75)$$

Pela hipótese indutiva, existe uma fatoração  $VTV^*$  de  $C$ , agora escrevemos:

$$Q = U \begin{pmatrix} 1 & 0 \\ 0 & V \end{pmatrix} \quad (76)$$

$Q$  é uma matriz unitária e temos que

$$Q^*AQ = \begin{pmatrix} \lambda & BV \\ 0 & T \end{pmatrix} \quad (77)$$

Essa era a fatoração de Schur que procurávamos

□

## **6 Lecture 25 - Algoritmos de Autovalores**

Essa Lecture é focada em mostrar a ideia geral dos algoritmos que são divididos em duas fases

1. Redução da forma completa para uma forma estrategicamente estruturada
2. Aplicação de um processo iterativo que leva à convergência dos autovalores

Ela também foca em explicar as vantagens desses métodos

## 6.1 Algoritmos óbvios (Ou nem tanto)

Por mais que os autovetores e autovalores tenham propriedades bonitas e simples, calcular eles de uma maneira numericamente estável não é algo tão simples e os algoritmos não são os mais óbvios. O mais óbvio que pensamos é calcular o polinômio característico da matriz e achar suas raízes, acontece que isso é uma péssima ideia, já que achar as raízes de um polinômio é um problema mal-condicionado.

Agora a gente pode tirar vantagem do fato que a sequência

$$\frac{x}{\|x\|}, \frac{Ax}{\|Ax\|}, \frac{A^2x}{\|A^2x\|}, \dots, \frac{A^nx}{\|A^nx\|} \quad (78)$$

converge, sobre certas condições, para o maior autovalor (Em valor absoluto) de  $A$ . Esse método é chamado de **Iteração sob Potências**, mas não é um método muito eficiente e não é utilizado em situações muito usuais.

Ao invés dessas ideias, é mais comum, para propósitos gerais, os algoritmos seguirem um princípio diferente: A computação de uma fatoração explícita de autovalores de  $A$ , onde um dos fatores da fatoração tem os autovalores de  $A$  como entradas. A gente viu 3 desses métodos na última lecture (Diagonalização, Diagonalização Unitária e Fatoração de Schur). Na prática, os algoritmos vão aplicando transformações em  $A$  de forma que eles inserem 0 nas colunas e entradas corretas (Tipo o que a gente viu no método de Householder)

## 6.2 Uma dificuldade fundamental

Acontece que **todo algoritmo para calcular autovalores deve ser iterativo**. Ué, por quê? Lembra que problemas de autovalores podem ser reduzidos a problemas de achar as raízes de um polinômio? Pois é, o inverso também é válido. O livro mostra isso criando um polinômio e expressando ele como o determinante de uma matriz e que as raízes do polinômio são os **autovalores** dessa matriz, mas isso não é o foco aqui. O foco é fazer a associação.

É bem conhecido o fato de que, para polinômios com grau maior ou igual a 5, não existe uma sequência de fórmulas com somas, subtrações, etc. (Fórmula fechada) que encontre suas raízes. O que isso quer dizer? Quer dizer que, se o problema de raízes de polinômios pode ser reduzido para um problema de autovalores, matrizes com dimensão maior ou igual a 5 não podem ter seus autovalores expressos em uma sequência finita de passos.

Por isso que os algoritmos de autovalores devem ser algoritmos iterativos que **convergem** para a solução

## 6.3 Fatoração e Diagonalização de Schur

A maioria dos algoritmos de fatoração atuais envolvem o uso da fatoração de Schur de uma matriz. A gente pega a matriz  $A$  e vai aplicando transformações nela com matrizes unitárias  $Q_j$  (Transformação  $X \mapsto Q_j^* X Q_j$ ) de forma que o produto:

$$Q_j^* \dots Q_2^* Q_1^* A Q_1 Q_2 \dots Q_j \quad (79)$$

Converja para uma matriz triangular superior  $T$  conforme  $j \rightarrow \infty$

O livro fala também que é possível utilizar de alguns truques para computar os autovalores complexos e que os algoritmos que veremos também podem ser usados, em matrizes Hermitianas, para obter sua diagonalização unitária.

## 6.4 Duas fases da computação de Autovalores

A sendo Hermitiana ou não, a gente separa a sequência (79) em duas partes.

1. A primeira fase consiste em produzir diretamente uma matriz **upper-Hessenberg**, isto é, uma matriz com zeros em baixo da primeira subdiagonal
2. Uma iteração é aplicada para que uma sequência formal de matrizes de Hessenberg converjam para uma matriz triangular superior. O processo se parece com isso:

$$\underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}_{A \neq A^*} \rightarrow \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}}_H \rightarrow \underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ & \times & \times & \times & \times \\ & & \times & \times & \times \\ & & & \times & \times \\ & & & & \times \end{pmatrix}}_T \quad (80)$$

Se  $A$  é hermitiana, isso fica ainda mais rápido já que vamos ter uma matriz tri-diagonal e, logo depois, uma diagonal

$$\underbrace{\begin{pmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{pmatrix}}_{A=A^*} \rightarrow \underbrace{\begin{pmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{pmatrix}}_H \rightarrow \underbrace{\begin{pmatrix} \times & & & & \\ & \times & & & \\ & & \times & & \\ & & & \times & \\ & & & & \times \end{pmatrix}}_T \quad (81)$$

## **7 Lecture 26 - Redução à forma de Hessenberg**



Beleza, vimos antes a importância da redução de Hessenberg, mas como ela funciona?

## 7.1 Uma ideia de Girico

A gente pode começar pensando “Macho, essa fatoração é mamão com açúcar, só eu multiplicar pelo refletor de Householder que eu vou ter 0 abaixo da diagonal que eu quiser”. Só que isso tem um problema, a gente precisa que o refletor multiplique de ambos os lados, ou seja:

$$Q_1^* A Q_1 \quad (82)$$

Isso faz com que os zeros que a gente colocou antes se percam, e a gente obtém uma matriz que a gente não queria :(.

## 7.2 Uma boa ideia

A gente vai fazer o seguinte: Vamos multiplicar  $A$  por um refletor de householder  $Q_1^*$  que mantém as duas primeiras linhas inalteradas, ou seja, vamos fazer combinações lineares das duas primeiras linhas de forma que todas as outras fiquem com 0 na primeira entrada, depois, ao multiplicar  $Q_1^* A$  por  $Q_1$ , a primeira coluna se mantém **inalterada**:

$$\underbrace{\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}}_A \rightarrow \underbrace{\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \\ 0 & x & x & x & x \end{pmatrix}}_{Q_1^* A} \rightarrow \underbrace{\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}}_{Q_1^* A Q_1} \quad (83)$$

Essa ideia continua a ser repetida para colunas subsequentes. Temos um algoritmo da forma:

---

```

1 function HessenbergReduction( $A \in \mathbb{C}^{m \times m}$ ) {
2   for  $k = 1$  to  $m - 2$ 
3      $x = A_{k+1:m,k}$ 
4      $v_k = \text{sign}(x_1) \|x\|_2 e_1 + x$ 
5      $v_k = v_k / \|v_k\|$ 
6      $A_{k+1:m,k:m} = A_{k+1:m,k:m} - 2v_k(v_k^* A_{k+1:m,k:m})$ 
7      $A_{1:m,k+1:m} = A_{1:m,k+1:m} - 2(A_{1:m,k+1:m} v_k) v_k^*$ 
8 }
```

---

Algoritmo 3: Redução de Householder para forma de Hessenberg

## 7.3 Hermitiana

É bem tranquilo de ver que o Algoritmo 3 gera uma matriz tri-diagonal no caso em que  $A$  é hermitiana, já que  $Q A Q^*$  é hermitiana. Inclusive, essa propriedade pode gerar uma redução de custo, tendo em vista que podemos realizar as operações apenas da diagonal para cima, ignorando a parte de baixo das operações.

## 7.4 Estabilidade

Assim como o algoritmo de Householder, para a fatoração QR, esse algoritmo é **backward stable**. Seja  $\tilde{H}$  a matriz de Hessenberg computada pelo computador ideal,  $\tilde{Q}$  seja a matriz exatamente unitária que reflete os vetores  $v_k$ , então o resultado a seguir pode ser demonstrado:

**Teorema 7.4.1:** Deixe a redução de Hessenberg  $A = Q T Q^*$  de uma matriz  $A$  ser computada pelo Algoritmo 3 em um computador ideal e sejam as matrizes  $\tilde{Q}$  e  $\tilde{H}$  definidas como falamos anteriormente, então:

$$\tilde{Q} \tilde{H} \tilde{Q}^* = A + \delta A, \text{ tal que } \frac{\|\delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \quad (84)$$

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

## **8 Lecture 27 - Quociente de Rayleigh e Iteração Inversa**

## 8.1 Restrição à matrizes reais e simétricas

Aqui nós iremos fazer essa restrição por questões que, ao compararmos os casos gerais e hermitianos, eles tem diferenças consideráveis, então por simplificação, falaremos apenas sobre o caso onde  $A$  é real e simétrica, ou seja:

- Autovalores reais
- Autovetores ortonormais

Isso vai continuar pelas próximas lectures até que se especifique que não vai mais continuar. Também vale ressaltar que a maioria das ideias descritas nas próximas lectures se referem a parte 2 das duas fases mencionadas na lecture 25. Ou seja, quando vamos aplicar as ideias que veremos aqui,  $A$  já terá sido transformada em uma tri-diagonal. Vale citar que também utilizaremos  $\|\cdot\| = \|\cdot\|_2$

## 8.2 Quociente de Rayleigh

**Definição 8.2.1** (Quociente de Rayleigh): O Quociente de Rayleigh de um vetor  $x \in \mathbb{R}^m$  é o escalar

$$r(x) = \frac{x^T A x}{x^T x} \quad (85)$$

Perceba que se  $x$  é um autovetor de  $A$  com autovalor  $\lambda$  associado, então  $r(x) = \lambda$ . Uma motivação para essa fórmula é pensarmos no seguinte: Dado um  $x$ , qual escalar  $\alpha$  “mais se comporta como um autovalor” no sentido de minimizar  $\|Ax - \alpha x\|$ ? Isso é um problema de mínimos quadrados  $m \times 1$  da forma  $\alpha x \approx Ax$ . Se escrevermos as equações normais:

$$x^T \alpha x = x^T A x \Rightarrow \alpha = r(x) \quad (86)$$

A gente pode fazer essas ideias mais quantitativas se tomarmos  $r(x) : \mathbb{R}^m \rightarrow \mathbb{R}$ , então podemos tomar interesse no comportamento local de  $r(x)$  quando  $x$  está perto de um autovalor. A gente pode calcular as derivadas parciais para isso:

$$\begin{aligned} \frac{\partial r(x)}{\partial x_j} &= \frac{\frac{\partial}{\partial x_j}(x^T A x)}{x^T x} - \frac{(x^T A x) \frac{\partial}{\partial x_j}(x^T x)}{(x^T x)^2} \\ &= \frac{2(Ax)_j}{x^T x} - \frac{(x^T A x) 2x_j}{(x^T x)^2} = \frac{2}{x^T x} (Ax - r(x)x)_j \end{aligned} \quad (87)$$

Podemos então expressar o gradiente como:

$$\nabla r(x) = \frac{2}{x^T x} (Ax - r(x)x) \quad (88)$$

É bem fácil de ver que, se a gente tem  $\nabla r(x) = 0$ , com  $x \neq 0$  então  $x$  é um autovetor de  $A$  (Tenta fazer mentalmente e lembra que  $r(x) \in \mathbb{R}$ ) e o inverso também, se  $x$  é autovetor de  $A$  então  $\nabla r(x) = 0$ .

Expressando geometricamente, os autovetores de  $A$  são pontos estacionários (pontos críticos) de  $r(x)$  e os autovalores de  $A$  são os valores de  $r(x)$  nesses pontos críticos.

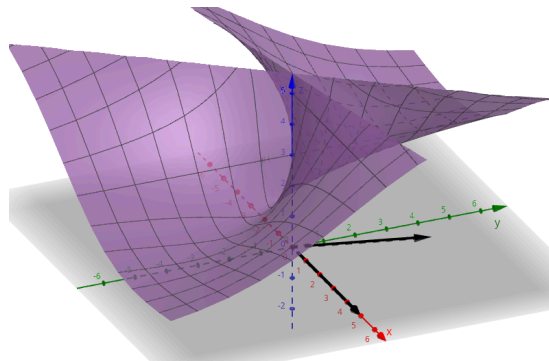


Figura 4:  $r(x)$  em função dos vetores no  $\mathbb{R}^2$  para a matriz  $\begin{pmatrix} 5 & 4 \\ 0 & 3 \end{pmatrix}$

Mas algo interessante que podemos perceber é que, aparentemente, não há apenas **um** ponto crítico nessa função, mas uma **reta**. Ué, mas por quê? O que acontece se, dado que  $Ax = \lambda x$ , eu pego um múltiplo  $\mu x$  de  $x$ ?

$$A(\mu x) = \lambda(\mu x) \quad (89)$$

$\mu x$  **ainda é autovetor de  $A$** . O que isso quer dizer? Quer dizer que, se um vetor  $x$  faz com que  $r(x)$  seja igual a um autovalor de  $A$ , então  $\alpha x$  também o fará  $\forall \alpha \in \mathbb{R}$ . Não acredita em mim? Veja por conta própria:

$$\begin{aligned} \text{Dado que } Ax &= \lambda x \\ r(\alpha x) &= \frac{(\alpha x)^T A(\alpha x)}{(\alpha x)^T (\alpha x)} = \frac{\alpha^2 x^T A x}{\alpha^2 x^T x} = \frac{\lambda x^T x}{x^T x} = \lambda \end{aligned} \quad (90)$$

Mas podemos contornar isso **limitando** o domínio de  $r(x)$ . Podemos fazer isso fazendo com  $r(x) : \mathbb{R}^m \rightarrow \mathbb{R}$  tal que  $\|x\| = 1$ , dessa forma, limitamos a  $m$ -esfera unitária em  $\mathbb{R}^m$  (No exemplo da Figura 4, seria uma circunferência em  $\mathbb{R}^2$ ). Dessa forma, em vez de serem retas com infinitos valores possíveis para zerar  $\nabla r(x)$ , temos pontos isolados **na** esfera.

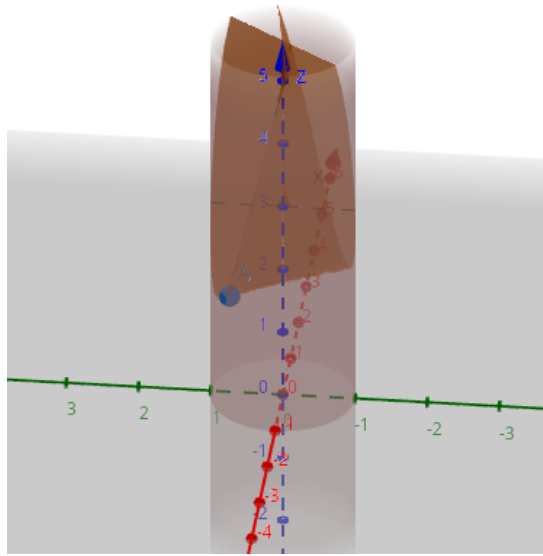


Figura 5: Mesma função da Figura 4 limitada dentro do cilindro  $x^2 + y^2 \leq 1$ . Só não coloquei  $= 1$  pois o Geogebra não conseguia fazer a plotagem

Seja  $q_j$  um autovetor de  $A$ , do fato que  $\nabla r(q_j) = 0$  nós chegamos que:

$$r(x) - r(q_j) = O(\|x - q_j\|^2), x \rightarrow q_j \quad (91)$$

Não precisamos entender o passo-a-passo até chegar nesse resultado, o importante dele é que o quociente de Rayleigh é uma **ótima** aproximação dos autovalores de  $A$ .

Um jeito mais explícito de vermos isso é expressar  $x$  como uma combinação linear dos autovetores de  $A$  (A gente pode fazer isso já que todos os autovetores de uma matriz são L.I), ou seja:  $x = \sum_{j=1}^m a_j q_j$ , o que significa que  $r(x) = \sum_{j=1}^m a_j^2 \lambda_j / \sum_{j=1}^m a_j^2$ , que é uma média ponderada dos autovalores de  $A$ .

### 8.3 Iteração por Potências

Agora nós invertemo as bola. Suponha que  $v^{(0)}$  é um vetor com  $\|v^{(0)}\| = 1$ . O processo de iteração por potência, citado antes como não muito bom, é esperado para convergir para o maior autovalor de  $A$

---

```

1 function PowerIteration( $A \in \mathbb{C}^{m \times m}$ ,  $v^{(0)}$  com  $\|v^{(0)}\| = 1$ ) {
2   for  $k = 1, 2, 3, \dots$ 
3      $w = Av^{(k-1)}$ 
4      $v^{(k)} = w/\|w\|$ 
5      $\lambda^{(k)} = (v^{(k)})^T Av^{(k)}$ 
6 }

```

---

Algoritmo 4: Iteração por potências

**Teorema 8.3.1:** Suponha que  $|\lambda_1| > |\lambda_2| \geq \dots \geq |\lambda_m| > 0$  e  $q_1^T v^{(0)} \neq 0$ . Então as iterações do Algoritmo 4 satisfazem:

$$\|v^k - (\pm q_1)\| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right), |\lambda^{(k)} - \lambda_1| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right) \quad (92)$$

Conforme  $k \rightarrow \infty$ . O sinal  $\pm$  significa que, a cada passo  $k$ , um dos dois sinais será escolhido para melhor estabilidade numérica

*Demonstração:* Escreva  $v^{(0)} = a_1 q_1 + \dots + a_m q_m$ . Como  $v^{(k)}$  é múltiplo de  $A^k v^{(0)}$  temos que, para algumas constantes  $c_k$

$$\begin{aligned} v^{(k)} &= c_k A^k v^{(0)} \\ &= c_k (a_1 \lambda_1^k q_1 + \dots + a_m \lambda_m^k q_m) \\ &= c_k \lambda_1^k (a_1 q_1 + \dots + a_m (\lambda_1/\lambda_m)^k q_m) \end{aligned} \quad (93)$$

A primeira equação se da ao fato de que, quando  $\lim_{k \rightarrow \infty} \left(\frac{\lambda_j}{\lambda_1}\right)^k = 0$ , porém, como  $\lambda_2$  é o maior entre  $\lambda_j$ , acaba que  $\left(\frac{\lambda_2}{\lambda_1}\right)^k$  domina o fator de erro  $v^{(k)} - (\pm q_1)$ .

A segunda envolve uma análise complicada que não há necessidade prática de visualizarmos □

O método de iteração por potências é bem ruim pois depende de alguns fatores específicos.

1. Só pode encontrar o maior autovalor de uma matriz
2. Se os dois maiores autovalores são próximos, a convergência demora muito
3. Se os dois maiores autovalores possuem mesmo valor, então o algoritmo não converge

## 8.4 Iteração Inversa

Antes de entendermos o que a iteração inversa faz, vamos conferir um teorema:

**Teorema 8.4.1:** Dado  $\mu \in \mathbb{R}$  tal que  $\mu$  **não** é autovalor de  $A$ , então os autovetores de  $(A - \mu I)^{-1}$  são os mesmos de  $A$ , onde os autovalores correspondentes são  $\left\{(\lambda_j - \mu)^{-1}\right\}$  de tal forma que  $\lambda_j$  são os autovalores de  $A$

*Demonstração:* Muito importante ressaltar que, como  $\mu$  **não** é autovalor de  $A$ , então  $A - \mu I$  é **invertível**.

$$\begin{aligned} Av &= \lambda v \\ Av - \mu Iv &= \lambda v - \mu Iv \\ (A - \mu I)v &= (\lambda - \mu)v \\ (A - \mu I)^{-1}(A - \mu I)v &= (A - \mu I)^{-1}(\lambda - \mu)v \\ \frac{1}{\lambda - \mu}v &= (A - \mu I)^{-1}v \end{aligned} \quad (94)$$

E isso nos dá uma ideia! Se aplicarmos a iteração de potências em  $(A - \mu I)^{-1}$ , o valor convergirá rapidamente para  $q_j$  (Autovetor de  $A$ )

---

```

1 function ReverseIteration( $A \in \mathbb{C}^{m \times m}$ ,  $v^{(0)}$  com  $\|v^{(0)}\| = 1$ ) {
2   for  $k = 1, 2, 3, \dots$ 
3     Resolva  $(A - \mu I)w = v^{(k-1)}$  para  $w$ 
4      $v^{(k)} = w / \|w\|$ 
5      $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$ 
6 }
```

---

Algoritmo 5: Iteração Inversa

Você pode estar se perguntando: “Mas e se  $\mu$  for um autovalor de  $A$ ? Isso vai fazer com que  $A - \mu I$  não seja inversível! Ou de  $\mu$  for muito próximo de um autovalor de  $A$ , se isso acontecer,  $A - \mu I$  vai ser **muito** mal-condicionada e vai ser quase impossível uma inversa precisa! Isso não vai quebrar o algoritmo?”. São perguntas válidas, mas não, isso não quebra o algoritmo! Há um exercício no livro que aborda isso (Se eu conseguir resolver antes da A2, eu coloco aqui).

Aqui o algoritmo também é um pouco mais interessante pois, dependendo do  $\mu$  que escolhermos, podemos encontrar um autovalor diferente, ou seja, podemos escolher qual autovalor encontrar se fizermos a escolha certa de  $\mu$

**Teorema 8.4.2:** Suponha que  $\lambda_J$  é o autovalor **mais próximo** de  $\mu$  e  $\lambda_K$  é o **segundo** mais próximo. Suponha então que  $q_J^T v^{(0)} \neq 0$ , então as iterações do Algoritmo 5 satisfazem:

$$\begin{aligned} \|v^{(k)} - (\pm q_J)\| &= O\left(\left|\frac{\mu - \lambda_J}{\mu - \lambda_K}\right|^k\right) \\ |\lambda^{(k)} - \lambda_J| &= O\left(\left|\frac{\mu - \lambda_J}{\mu - \lambda_K}\right|^{2k}\right) \end{aligned} \quad (95)$$

Conforme  $k \rightarrow \infty$  e  $\pm$  tem o mesmo significado que Teorema 8.3.1

Esse algoritmo, como mencionado, é muito útil se os autovalores são conhecidos ou se tem uma noção de quanto eles valem aproximadamente ( $\mu$  converge para o mais próximo)

## 8.5 Iteração do Quociente de Rayleigh

Beleza, a gente já biscoitou 2 métodos, um que a gente tem uma estimativa inicial de autovetor, e vai aproximando o autovalor, depois uma que a gente tem uma aproximação de um autovalor e vamos aproximando um autovetor, combinar as duas ideias me parece uma **boa ideia**.

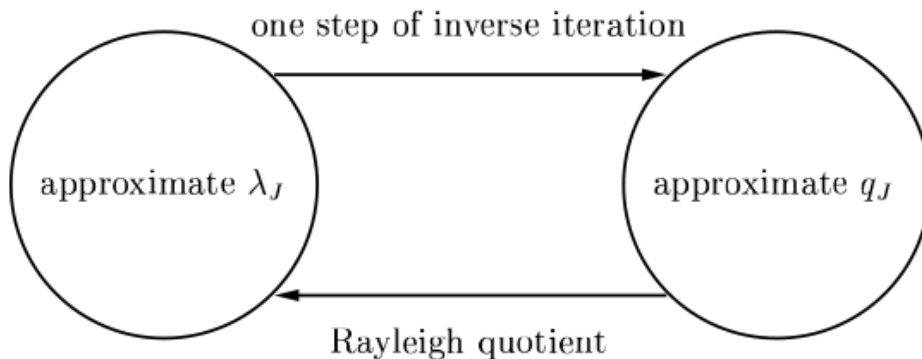


Figura 6: Iteração do Quociente de Rayleigh

A ideia é a gente ficar melhorando a estimativa de autovalores que temos pra que o algoritmo de **iteração reversa** tenha uma convergência muito mais rápida

---

```

1 function RayleighQuotientIteration( $A \in \mathbb{C}^{m \times m}$ ) {
2    $v^{(0)}$  com  $\|v^{(0)}\| = 1$ 
3    $\lambda^{(0)} = (v^{(0)})^T A v^{(0)}$ 
4   for  $k = 1, 2, 3, \dots$ 
5     Resolva  $(A - \lambda^{(k-1)} I)w = v^{(k-1)}$  para  $w$ 
6      $v^{(k)} = w / \|w\|$ 
7      $\lambda^{(k)} = (v^{(k)})^T A v^{(k)}$ 
8 }
```

---

Algoritmo 6: Iteração do Quociente de Rayleigh

A convergência do algoritmo é ótima, a cada iteração o valor de precisão triplica.

**Teorema 8.5.1:** Quando o algoritmo de iteração do quociente de rayleigh converge para um autovalor  $\lambda_J$  e um autovetor  $q_J$  de  $A$  de forma que:

$$\begin{aligned}
 \|v^{(k+1)} - (\pm q_J)\| &= O(\|v^{(k)} - (\pm q_J)\|^3) \\
 |\lambda^{(k+1)} - \lambda_J| &= O(|\lambda^{(k)} - \lambda_J|^3)
 \end{aligned} \tag{96}$$

Não há necessidade de uma demonstração formal, apenas a ideia de que há uma **ótima** conversão do algoritmo

## **9 Lecture 28 - Algoritmo QR sem Shift**



Agora vamos ver que o algoritmo de QR pode ser utilizado como um algoritmo estável para computar a fatoração QR de potências de  $A$

## 9.1 O Algoritmo QR

A versão mais simplificada parece coisa de doido.

---

```

1 function QRIteration( $A \in \mathbb{C}^{m \times m}$ ) {
2    $A^{(0)} = A$ 
3   for  $k = 1, 2, 3, \dots$ 
4      $Q^{(k)}, R^{(k)} = \text{qr}(A^{(k-1)})$ 
5      $A^{(k)} = R^{(k)}Q^{(k)}$ 
6 }
```

---

Algoritmo 7: Algoritmo QR

É um algoritmo estupidamente simples, mas sobre certas circunstâncias, esse algoritmo converge para a forma de Schur de uma matriz (Triangular superior se for arbitrária e diagonal se for simétrica). Por questão de simplicidade, vamos continuar assumindo que  $A$  é simétrica

Pra que a redução a forma diagonal seja útil pra achar autovalor, a gente precisa que transformações similares estejam envolvidas. “Oxe, daonde?”. Quando a gente faz  $A^{(k)} = R^{(k)}Q^{(k)}$ , a gente pode substituir  $R^{(k)}$  por  $(Q^{(k)})^T A^{(k-1)}$ , ou seja:  $A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)}$  (Mesmo que  $M^{-1}AM$ ). O Algoritmo 7 converge cubicamente assim como o do Algoritmo 6, porém, para o algoritmo ser prático, precisamos introduzir **shifts**. Introdução de **shifts** é 1 de 3 modificações que fazemos nesse algoritmo para que ele fique prático.

1. Antes de iniciar a iteração,  $A$  é reduzida a forma tridiagonal
2. Em vez de  $A^{(k)}$ , usamos uma matriz trocada  $A^{(k)} - \mu^{(k)}I$  que é fatorada a cada iteração e  $\mu^{(k)}$  é uma estimativa de autovalor
3. Quando possível (Especialmente quando um autovalor é encontrado) nós quebramos  $A^{(k)}$  em submatrizes

---

```

1 function ShiftedQR( $A \in \mathbb{C}^{m \times m}$ ) {
2    $(Q^{(0)})^T A^{(0)} Q^{(0)} = A$ 
3   for  $k = 1, 2, 3, \dots$ 
4     Escolha um shift  $\mu^{(k)}$ 
5      $Q^{(k)}, R^{(k)} = \text{qr}(A^{(k-1)} - \mu^{(k)}I)$ 
6      $A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}I$ 
7     Se qualquer elemento  $A_{j,j+1}^{(k)}$  fora da diagonal é suficientemente próximo de 0
8        $A_{j,j+1} = A_{j+1,j} = 0$  para obter
9        $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix} = A^{(k)}$ 
10      e agora aplicamos o algoritmo em  $A_1$  e  $A_2$ 
11 }
```

---

Algoritmo 8: Algoritmo QR com **shifts**

Esse é um algoritmo muito usado desde 1960. Mas perceba que precisamos ter uma noção prévia de quanto vale os autovalores da matriz, pois necessitamos ter aproximações particularmente boas de  $\mu^{(k)}$  para que o algoritmo tenha uma boa convergência. Porém, nos anos 1990 um competidor surgiu (Vai ser discutido na lecture 30 e a gente detalha o algoritmo com shifts na próxima lecture).

## 9.2 Iterações Simultâneas Não-normalizadas

A gente vai tentar relacionar (Eu vou tentar traduzir o que o livro fala né) o Algoritmo 7 com um algoritmo chamado **iterações simultâneas** que tem um comportamento mais simples de visualizar (De acordo com o livro, pq tudo pra ele é fácil né)

A ideia do algoritmo é aplicar o Algoritmo 4 (Iteração por Potências) para vários vetores simultaneamente. Vamos supor que a gente tem  $n$  vetores LI iniciais  $v_1^{(0)}, \dots, v_n^{(0)}$ . Se a gente aplica  $A^k v_1^{(0)}$ , conforme  $k \rightarrow \infty$ , isso converge para o autovetor correspondente ao autovalor de maior valor absoluto (Com algumas condições adequadas), meio que parece plausível que  $\text{span}\{A^k v_1^{(0)}, A^k v_2^{(0)}, \dots, A^k v_n^{(0)}\}$  converge para  $\text{span}\{q_1, \dots, q_n\}$  que é o espaço formado pelos autovetores associados aos  $n$  (Novamente com condições adequadas). Ué, mas quando eu aplico o método a um único vetor ele não converge pro maior? Como que aplicar a vários muda isso? Vou primeiro definir uma estrutura importante no algoritmo e depois faço uma explicação mais simplificada e uma analogia pra entender isso melhor

Na notação matricial, fazemos:

$$V^{(0)} = \begin{pmatrix} | & & | \\ v_1^{(0)} & \dots & v_n^{(0)} \\ | & & | \end{pmatrix} \quad (97)$$

E definimos

$$V^{(k)} = A^k V^{(0)} = \begin{pmatrix} | & & | \\ v_1^{(k)} & \dots & v_n^{(k)} \\ | & & | \end{pmatrix} \quad (98)$$

Vamos tentar entender a pergunta que fiz antes. Quando a gente aplica o algoritmo a um único vetor, ele vai se alinhando ao vetor dominante, porém, se a gente faz o mesmo com vários vetores **ao mesmo tempo**, ou seja, eu aplico na matriz, não faz muito sentido isso ocorrer. Pensa que se isso acontecesse, eu ia ter como resultado uma matriz que todas as colunas fossem iguais (Meio esquisito isso). O que acontece é que o espaço das colunas de  $V^{(0)}$  vai “girando” e se alinhando ao espaço que falei dos autovetores de  $A$

Imagine 3 agulhas em 3 direções diferentes (De forma que as agulhas representem vetores LI, e to falando apenas 3 pra representar  $\mathbb{R}^3$ , mas se aplica pra outros espaços). Aplicar o método de potência em um único vetor é como se aplicássemos um campo magnético que direciona todas as agulhas pra direção norte (Que seria a direção do autovetor associado ao maior autovalor). Aplicar na matriz  $V^{(0)}$  seria aplicar um campo magnético complexo, em que cada vetor  $v_j^{(0)}$  fica virado pra direção que ele “sente mais”

Beleza, vamos continuar então. A gente tá interessado em  $C(V^{(k)})$ . Que tal a gente pegar uma boa base desse espaço? Uma boa ideia é a fatoração QR dessa matriz né? Já que as colunas de  $Q$  são uma base ortonormal de  $C(V^{(k)})$

$$\hat{Q}^{(k)} \hat{R}^{(k)} = V^{(k)} \quad (99)$$

Aqui estamos vendo a fatoração reduzida, logo,  $\hat{Q}^{(k)}$  é  $m \times n$  e  $\hat{R}^{(k)}$  é  $n \times n$ . Bem, se as colunas de  $\hat{Q}^{(k)}$  vão formando uma base do span dos autovetores que eu comentei antes, então faz sentido elas irem convergindo para os próprios autovetores de  $A$  ( $\pm q_1, \dots, \pm q_n$ ). A gente pode argumentar melhor sobre isso fazendo uma expansão das colunas de  $V^{(0)}$  e  $V^{(k)}$  como combinação linear dos autovetores de  $A$  que nem a gente fez em uma lecture anterior

$$\begin{aligned} v_j^{(0)} &= a_{1j}q_1 + \dots + a_{mj}q_m \\ v_j^{(k)} &= \lambda_1^k a_{1j}q_1 + \dots + \lambda_m^k a_{mj}q_m \end{aligned} \quad (100)$$

Mas não precisamos entrar em detalhes mais aprofundados. Assim como na lecture anterior, resultados vão convergir quando satisfazemos duas condições.

1. A primeira é que, ao calcularmos  $n$  autovalores, todos tenham valor absoluto distintos

$$|\lambda_1| > |\lambda_2| > \dots > |\lambda_n| > |\lambda_{n+1}| \geq |\lambda_{n+2}| \geq \dots \geq |\lambda_m| \quad (101)$$

2. A segunda condição é que os valores  $a_{ij}$  na decomposição dos  $v_j^{(i)}$  que comentei antes sejam, de certa forma, não-singulares. O que isso quer dizer? Significa que eu preciso formar uma boa mistura dos meus autovetores originais. Tipo, se eu formar  $v_j^{(i)}$  ortogonal a algum autovetor, ele não vai ser muito bem aproximado pelo meu

algoritmo. Vou formalizar essa condição um pouco. Vamos definir  $\hat{Q}$  como a matriz  $m \times n$  que as colunas são os autovetores  $q_1, \dots, q_n$  de  $A$ . Então podemos formalizar isso escrevendo:

$$\text{Todas as submatrizes consequentes de } \hat{Q}^T V^{(0)} \text{ são inversíveis} \quad (102)$$

Eu posso definir como essa multiplicação pois eu vou ter que o elemento  $ij$  dessa matriz vai ser  $q_i^T v_j^{(0)}$ , que ao olharmos para a Equação (100), é igual a  $a_{ij}$

**Teorema 9.2.1:** Suponha que a iteração (97) e (99) é realizada e as condições (101) e (102) são satisfeitas. Conforme  $k \rightarrow \infty$ , as colunas da matriz  $Q^{(k)}$  vão convergindo linearmente para os autovetores de  $A$ :

$$\|q_j^{(k)} - \pm q_j\| = O(C^k) \quad (103)$$

para cada  $j$  com  $1 \leq j \leq n$  e  $C < 1$  é a constante  $\max_{1 \leq k \leq n} (|\lambda_{k+1}|/|\lambda_k|)$

*Demonstração:* Vamos transformar  $\hat{Q} \in \mathbb{R}^{m \times n}$  em  $Q \in \mathbb{R}^{m \times m}$ , de forma que  $Q$  tenha como colunas todos os autovetores de  $A$ . Definimos também  $\Lambda$  como a matriz de autovalores de  $A$  de tal forma que  $A = Q\Lambda Q^T$ . Defina também  $\hat{\Lambda}$  como sendo o bloco  $n \times n$  de  $\Lambda$  com os autovalores associados a matriz  $\hat{Q}$ .

$$V^{(k)} = A^k V^{(0)} = Q\Lambda^k Q^T V^{(0)} = \hat{Q}\hat{\Lambda}^k \hat{Q}^T V^{(0)} + O(|\lambda_{k+1}|) \quad (104)$$

Se a condição (102) for satisfeita, podemos fazer uma manipulação simples

$$V^{(k)} = \left( \hat{Q}\hat{\Lambda}^k + O(|\lambda_{k+1}|) \right) \left( \hat{Q}^T V^{(0)} \right)^{-1} \hat{Q}^T V^{(0)} \quad (105)$$

Como  $\hat{Q}^T V^{(0)}$  é inversível,  $C(V^{(k)}) = C\left(\hat{Q}\hat{\Lambda}^k + O(|\lambda_{k+1}|) \left(\hat{Q}^T V^{(0)}\right)^{-1}\right)$ . Ou seja, a gente consegue perceber que o espaço vai convergindo para o span dos autovetores de  $A$ . A gente pode até tentar quantificar a convergência, mas não tem necessidade  $\square$

### 9.3 Iteração Simultânea

Conforme  $k \rightarrow \infty$ , os vetores  $v_j^{(k)}$  vão convergindo para múltiplos do autovetor dominante (Associado ao autovalor). Quando eu digo múltiplos, eu quero dizer muito próximos. Por mais que o span deles converja para algo útil, eles em si formam uma base muito mal condicionada.

Vamos fazer uma alteração então, vamos construir uma sequência de matrizes  $Z^{(k)}$  tal que  $C(Z^{(k)}) = C(V^{(k)})$

---

```

1 function SimultaneousAlgorithm( $A \in \mathbb{C}^{m \times m}$ ) {
2   Escolha  $\hat{Q}^{(0)} \in \mathbb{R}^{m \times n}$  com colunas ortonormais
3   for  $k = 1, 2, 3, \dots$ 
4      $Z = A\hat{Q}^{(k-1)}$ 
5      $\hat{Q}^{(k)}, \hat{R}^{(k)} = \text{qr}(Z)$  # Fatoração Reduzida
6 }
```

---

Algoritmo 9: Iteração Simultânea

Assim é mais tranquilo de ver que  $C(Z^{(k)}) = C(\hat{Q}^{(k)}) = C(A^k \hat{Q}^{(0)})$ . Matematicamente falando, esse novo método converge igual o método anterior (Sob as mesmas circunstâncias)

**Teorema 9.3.1:** O Algoritmo 9 gera as mesmas matrizes  $\hat{Q}^{(k)}$  que os passos de iteração (97) ~ (99) considerados no Teorema 9.2.1 e sob as mesmas condições (101) e (102)

## 9.4 Iteração Simultânea $\Leftrightarrow$ Algoritmo QR

Beleza, agora a gente pode tentar entender o algoritmo QR (Não é um algoritmo pra calcular a fatoração QR, mas usa ela para calcular os autovalores e autovetores de  $A$ ). A gente vai aplicar a iteração simultânea na identidade, assim, a gente até remove os acentos de  $\hat{Q}^{(k)}$  e  $\hat{R}^{(k)}$ . A gente vai fazer umas substituições que eu vou explicar direitinho depois.

Primeiro de todos, temos um algoritmo de iteração simultânea com uma leve adaptação, mostraremos que ele e o algoritmo qr são equivalentes

---

```

1 function ModifiedSimultaneousAlgorithm( $A \in \mathbb{C}^{m \times m}$ ) {
2    $\underline{Q}^{(0)} = I$ 
3   for  $k = 1, 2, 3, \dots$ 
4      $Z = A\underline{Q}^{(k-1)}$ 
5      $\underline{Q}^{(k)}, \underline{R}^{(k)} = \text{qr}(Z)$ 
6      $\underline{A}^{(k)} = (\underline{Q}^{(k)})^T A \underline{Q}^{(k)}$ 
7      $\underline{R}^{(k)} = \underline{R}^{(k)} \underline{R}^{(k-1)} \dots \underline{R}^{(1)}$ 
8 }
```

---

Algoritmo 10: Iteração Simultânea Modificada

Aqui, a gente colocou  $\underline{Q}^{(k)}$  com esse traço em baixo só pra diferenciar o  $\underline{Q}$  do algoritmo de iteração simultânea e do algoritmo QR

---

```

1 function UnshiftedQRAlgorithm( $A \in \mathbb{C}^{m \times m}$ ) {
2    $A^{(0)} = A$ 
3   for  $k = 1, 2, 3, \dots$ 
4      $\underline{Q}^{(k)}, \underline{R}^{(k)} = \text{qr}(A^{(k-1)})$ 
5      $\underline{A}^{(k)} = \underline{R}^{(k)} \underline{Q}^{(k)}$ 
6      $\underline{Q}^{(k)} = \underline{Q}^{(1)} \underline{Q}^{(2)} \dots \underline{Q}^{(k)}$ 
7      $\underline{R}^{(k)} = \underline{R}^{(k)} \underline{R}^{(k-1)} \dots \underline{R}^{(1)}$ 
8 }
```

---

Algoritmo 11: Algoritmo QR sem Shift

Agora podemos visualizar a convergência de ambos os algoritmos.

**Teorema 9.4.1:** O Algoritmo 10 e Algoritmo 11 geram a mesma sequência de matrizes  $\underline{Q}^{(k)}$ ,  $\underline{R}^{(k)}$  e  $\underline{A}^{(k)}$ , de tal forma que:

$$\underline{A}^k = \underline{Q}^{(k)} \underline{R}^{(k)} \quad (106)$$

junto da projeção

$$\underline{A}^{(k)} = (\underline{Q}^{(k)})^T \underline{A} \underline{Q}^{(k)} \quad (107)$$

*Demonstração:* Vamos fazer indução em  $k$

- Caso base ( $k = 1$ ): Trivial, já que  $\underline{A}^{(0)} = \underline{Q}^{(0)} = \underline{R}^{(0)} = I$  e  $\underline{A}^{(0)} = A$
- Passo indutivo ( $k > 1$ ): A parte de que  $\underline{A}^{(k)} = (\underline{Q}^{(k)})^T \underline{A} \underline{Q}^{(k)}$  por definição de  $\underline{A}^k$  (Algoritmo 10). Então só precisamos conferir que  $\underline{A}^k = \underline{Q}^{(k)} \underline{R}^{(k)}$ , e fazemos isso, primeiro, considerando o algoritmo de iteração simultânea (Assumindo que isso é válido para  $\underline{A}^{k-1}$ ):

$$\underline{A}^k = \underline{A} \underline{Q}^{(k-1)} \underline{R}^{(k-1)} = \underline{Q}^{(k)} \underline{R}^{(k)} \underline{R}^{(k-1)} = \underline{Q}^{(k)} \underline{R}^{(k)} \quad (108)$$

Agora, faremos o mesmo assumindo o algoritmo QR

$$A^k = A\underline{Q}^{(k-1)}\underline{R}^{(k-1)} = \underline{Q}^{(k-1)}A^{(k-1)}\underline{R}^{(k-1)} = \underline{Q}^{(k)}\underline{R}^{(k)} \quad (109)$$

Então verificamos que

$$A^{(k)} = (\underline{Q}^{(k)})^T A^{(k-1)} \underline{Q}^{(k)} = (\underline{Q}^{(k)})^T A \underline{Q}^{(k)} \quad (110)$$

□

## 9.5 Convergência do algoritmo QR

Show, agora a gente pode entender melhor como que esse algoritmo acha os autovalores e autovetores. A parte dos autovetores a gente consegue visualizara pela equação (106), e pelo Teorema 9.4.1, já que, se o método de iteração simultânea converge para autovetores e tanto ele quanto o algoritmo QR geram as mesmas matrizes, obviamente ambos vão ter as matrizes  $Q$  convergindo para a matriz de colunas sendo os autovetores. Como  $Q$  converge pra matriz de autovetores, por consequência, se eu faço  $Q^T A Q$ , isso vai convergir pra matriz com os autovalores de  $A$  na diagonal (Diagonalização)

**Teorema 9.5.1:** Deixe que o Algoritmo 11 seja aplicado em uma matriz real simétrica  $A$  que os autovalores satisfazem  $|\lambda_1| > |\lambda_2| > \dots > |\lambda_m|$  e que a matriz de autovetores correspondente  $Q$  não tem blocos singulares (Todos os blocos da matriz formam matrizes inversíveis). Então, conforme  $k \rightarrow \infty$ ,  $A^{(k)}$  converge linearmente com constante  $\max_{j(|\lambda_{j+1}|/|\lambda_j|)}$  para a matriz com os autovalores na diagonal e  $Q^{(k)}$  converge na mesma velocidade para  $Q$

## **10 Lecture 29 - Algoritmo QR com Shifts**

A ideia aqui é a inserção dos shifts  $A \rightarrow A - \mu I$  e discutir por que que essa ideia nada intuitiva funciona e leva a uma convergência cúbica.

## 10.1 Conexão com a Iteração Reversa

A gente tinha visto que o algoritmo QR unshifted (Algoritmo 11) era a mesma coisa que aplicar a iteração reversa na matriz identidade. Tem um porém, o Algoritmo 11 também é equivalente a aplicar a iteração inversa simultânea numa matriz identidade “invertida”  $P$ . Vamo tentar desenvolver melhor essa ideia:

Seja  $Q^{(k)}$ , assim como na última lecture, o fator ortogonal no  $k$ -ésimo passo da iteração do algoritmo QR. Mostramos antes que o produto acumulado dessas matrizes forma:

$$\underline{Q}^{(k)} = \prod_{j=1}^k Q^{(j)} = \left( q_1^{(k)} \mid \dots \mid q_m^{(k)} \right) \quad (111)$$

É a mesma matriz ortogonal que aparece no  $k$ -ésimo passo do algoritmo de iteração simultânea.

$$A^k = \underline{Q}^{(k)} \underline{R}^{(k)} \quad (112)$$

Se a gente inverte essa fórmula, temos

$$A^{-k} = (\underline{R}^{(k)})^{-1} (\underline{Q}^{(k)})^T = \underline{Q}^{(k)} (\underline{R}^{(k)})^{-T} \quad (113)$$

Essa segunda igualdade a gente tira porque  $A^{-1}$  é simétrica (Ainda tamo usando que  $A$  é simétrica). Deixe  $P$  ser a matriz de permutação que troca a ordem de todas as linhas e colunas:

$$P = \begin{pmatrix} & & & 1 \\ & & \ddots & \\ & 1 & & \\ 1 & & & \end{pmatrix} \quad (114)$$

Bem, como  $P^2 = I$ , a gente pode reescrever a equação que tínhamos anteriormente como:

$$A^{-k} P = (\underline{Q}^{(k)} P) \left( P (\underline{R}^{(k)})^{-T} P \right) \quad (115)$$

Perceba que  $\underline{Q}^{(k)} P$  é ortogonal ( $\underline{Q}^{(k)}$  é ortogonal e  $P$  também) e  $P (\underline{R}^{(k)})^{-T} P$  é triangular superior ( $(\underline{R}^{(k)})^{-T}$  é triangular inferior, daí eu inverte a ordem das colunas, e depois a ordem das linhas, aí fica triangular superior), ou seja, a equação anterior pode ser interpretada como uma fatoração QR de  $A^{-k} P$ . Isso que fizemos é a mesma coisa que aplicar o algoritmo QR na matriz  $A^{-1}$  usando a matriz  $P$  como ponto de partida do algoritmo.

## 10.2 Conexão com o Algoritmo de Iteração Reversa com Shifts

Ok, a gente viu então que o algoritmo QR é tipo uma mistura da iteração reversa e da iteração simultânea reversa. O negócio é que a gente viu em umas lectures anteriores que o último que mencionei pode ser melhorado com o uso de shifts (Algoritmo 8). Isso é como inserir shifts nos dois algoritmos que comentei anterioremente. Vou escrever o algoritmo aqui novamente (Omiti a parte final de obter as submatrizes):

---

```

1 function ShiftedQR( $A \in \mathbb{C}^{m \times m}$ ) {
2    $(Q^{(0)})^T A^{(0)} Q^{(0)} = A$ 
3   for  $k = 1, 2, 3, \dots$ 
4     Escolha um shift  $\mu^{(k)}$ 
5      $Q^{(k)}, R^{(k)} = \text{qr}(A^{(k-1)} - \mu^{(k)} I)$ 
6      $A^{(k)} = R^{(k)} Q^{(k)} + \mu^{(k)} I$ 
7     ...
8 }
```

---

Deixe que  $\mu^{(k)}$  seja a aproximação de autovalor que a gente escolhe no  $k$ -ésimo passo do algoritmo QR. De acordo com o Algoritmo 8, a relação entre os passos  $k - 1$  e  $k$  do algoritmo é:

$$\begin{aligned} A^{(k-1)} - \mu^{(k)} I &= Q^{(k)} R^{(k)} \\ A^{(k)} &= R^{(k)} Q^{(k)} + \mu^{(k)} I \end{aligned} \quad (116)$$

Isso nos dá o seguinte (Só fazer umas substituições):

$$A^{(k)} = (Q^{(k)})^T A^{(k-1)} Q^{(k)} \quad (117)$$

Aí se a gente aplica uma indução, temos:

$$A^{(k)} = (\underline{Q}^{(k)})^T A \underline{Q}^{(k)} \quad (118)$$

Se você para pra olhar, é a mesma coisa que a gente definiu no Teorema 9.4.1 (Segunda equação). O problema é que a primeira equação não vale mais, ela vai ser substituída por:

$$\prod_{j=k}^1 (A - \mu^{(j)} I) = \underline{Q}^{(k)} \underline{R}^{(k)} \quad (119)$$

Aí a gente não precisa entrar em detalhes da prova dessa equivalência. Isso acarreta que as colunas de  $\underline{Q}^{(k)}$  aos poucos vão convergindo para autovetores de  $A$ . O livro dá uma ênfase na primeira e na última coluna, onde cada uma é equivalente a aplicar o algoritmo da iteração reversa com shifts nos vetores canônicos  $e_1$  e  $e_m$  respectivamente.

### 10.3 Conexão com a Iteração do Quociente de Rayleigh

Beleza, vimos que os shifts são bem poderosos para o cálculo das matrizes, mas aí tu pode tá se perguntando: “Q djabo eu faço pra escolher meus shift? Eu tenho q ser Mãe de Ná?”. E você está corretíssimo, precisamos de um método para escolher shifts interessantes para o algoritmo.

Faz sentido a gente tentar usar o quociente de Rayleigh pra isso. A gente quer tentar fazer com que a última coluna de  $\underline{Q}^{(k)}$  converja. Então faz sentido a gente usar o Quociente de Rayleigh com essa última coluna né?

$$\mu^{(k)} = \frac{(q_m^{(k)})^T A q_m^{(k)}}{q_m^{((k))^T} q_m^{(k)}} = (q_m^{(k)})^T A q_m^{(k)} \quad (120)$$

Se escolhermos esse valor, as estimativas  $\mu^{(k)}$  (Estimativa de autovalor) e  $q_m^{(k)}$  estimativa de autovetor são idênticos àqueles computados pela iteração do quociente de rayleigh com o vetor inicial sendo  $e_m$

Tem um negócio bem massa que a gente pode ver com isso. Que o valor  $A_{mm}^{(k)}$  é igual a  $r(q_m^{(k)})$  ( $r$  sendo a função do quociente de rayleigh), a gente pode visualizar assim:

$$A_{mm}^{(k)} = e_m^T A^{(k)} e_m = e_m^T (\underline{Q}^{(k)})^T A \underline{Q}^{(k)} e_m = q_m^{((k))^T} A q_m^{(k)} \quad (121)$$

Ou seja, escolher  $\mu^{(k)}$  como sendo o coeficiente de rayleigh de  $q_m^{(k)}$  é a mesma coisa que escolher ele como sendo a última entrada de  $A^{(k)}$ . A gente chama isso de **Shift do Quociente de Rayleigh**.

### 10.4 Wilkinson Shift

A gente tem um problema com o método anterior. Nem sempre escolhermos  $A_{mm}^{(k)}$  ou  $r(q_m^{(k)})$  como os shifts para convergência funciona. Um exemplo disso é a matriz:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad (122)$$

Isso ocorre porque temos uma simetria nos autovalores (1 e  $-1$ ) e  $A_{mm}^{(k)} = 0$ , o que acarreta que ao escolhermos esse valor como shift, o algoritmo tende a beneficiar ambos os autovalores igualmente (Ou seja, eu não tá mais próximo de nenhum, vou tá igualmente distante dos dois). A gente precisa de uma estimativa que quebre a simetria, vamos fazer o seguinte então:

Deixe  $B$  ser definida pelo bloco  $2 \times 2$  inferior direito da matriz  $A^{(k)}$



$$B = \begin{pmatrix} a_{m-1} & b_{m-1} \\ b_{m-1} & a_m \end{pmatrix} \quad (123)$$

O **Shift de Wilkinson** é definido como o autovalor mais próximo de  $a_m$ . Em caso de empate, eu seleciono qualquer um dos dois autovalores arbitrariamente. Aqui tem uma fórmula numericamente estável pra achar esses autovalores:

$$\mu = a_m - \frac{\text{sign}(\delta)b_{m-1}^2}{|\delta| + \sqrt{\delta^2 + b_{m-1}^2}} \quad (124)$$

onde  $\delta = \frac{a_{m-1} - a_m}{2}$ . Se  $\delta = 0$ , eu posso definir  $\text{sign}(\delta)$  como sendo 1 ou  $-1$  arbitrariamente. O **Shift de Wilkinson** também atinge convergência cúbica e, nos piores casos, pelo menos quadrática (Pode ser mostrado). Em partiuar, o algoritmo QR com shift de Wilkinson sempre converge.

## 10.5 Estabilidade e Precisão

Como esperado, os algoritmos vistos anteriormente são **backward stable**, ou seja, calcular os autovalores de uma matriz  $A$  com os algoritmos é o mesmo que calcular os autovalores de uma matriz levemente perturbada  $\tilde{A}$  do modo puramente matemático. O teorema a seguir pode ser provado, mas não é o intuito:

**Teorema 10.5.1:** Deixe uma matriz real, simétrica e tridiagonal  $A \in \mathbb{R}^{m \times m}$  ser diagonalizada pelo algoritmo QR (Algoritmo 8) em um computador ideal. Deixe  $\tilde{\Lambda}$  ser a matriz de autovalores de  $A$  computada por aritmética de ponto flutuante e  $\tilde{Q}$  a matriz exatamente ortogonal associada ao produto dos refletores de householder e rotações utilizadas nos algoritmos, temos que:

$$\tilde{Q}\tilde{\Lambda}\tilde{Q} = A + \delta A \quad (125)$$

onde

$$\frac{\|\delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \quad (126)$$

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

Isso mostra que temos resultados muito bom! Inclusive, juntando com alguns outros teoremas que vimos (Teorema 10.5.1 e Teorema 7.4.1), temos que, para todo autovalor  $\lambda_j$ , o autovalor computado  $\tilde{\lambda}_j$  satisfaz:

$$\frac{|\tilde{\lambda}_j - \lambda_j|}{\|A\|} = O(\varepsilon_{\text{machine}}) \quad (127)$$

## **11 Discos de Gershgorin**

Os Discos de Gershgorin é um método de estimar **onde** estão os autovalores de uma matriz complexa no plano de **Argand-Gauss** (Aquele plano que representa os complexos). Como assim? Vamos pegar uma matriz aleatória  $A \in \mathbb{C}^{3 \times 3}$ , eu sei que ela tem, no máximo, 3 autovalores. Aplicando o teorema dos discos (Vou explicar posteriormente como aplicar, vamos só entender a ideia) eu obtive o seguinte resultado:

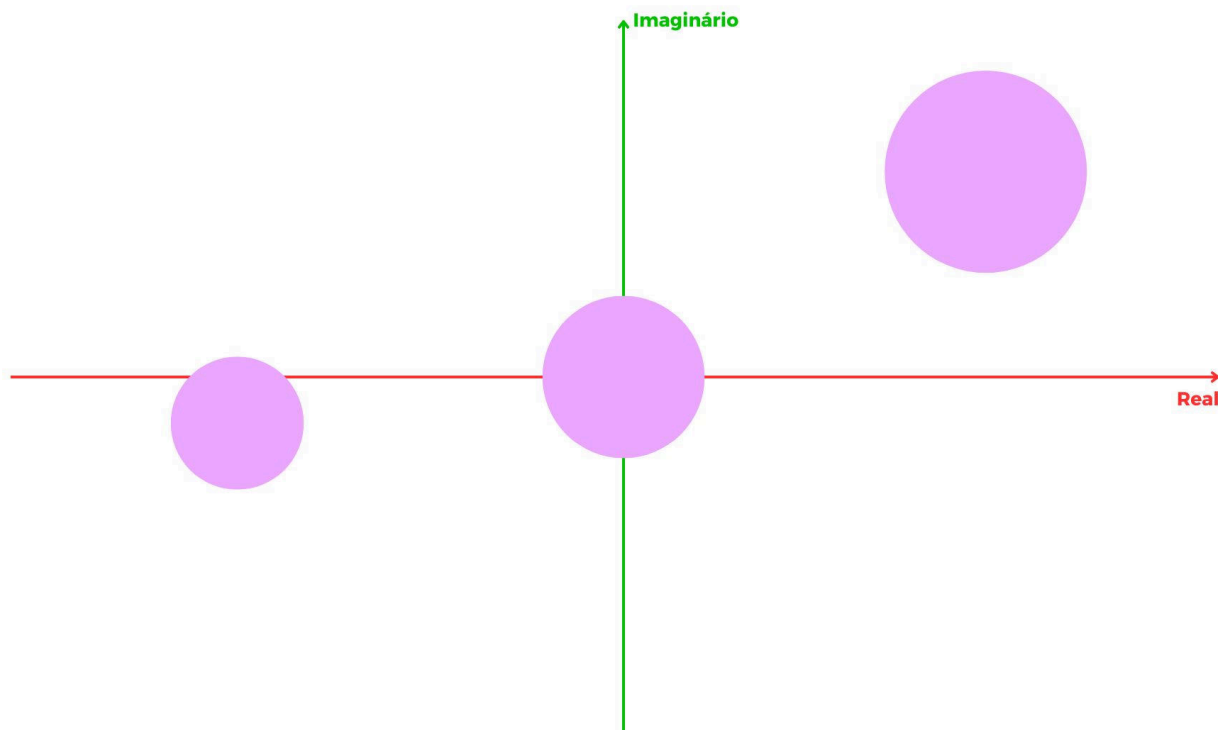


Figura 7: Ilustração dos Discos de Gershgorin de uma matriz  $3 \times 3$

Isso significa que os autovalores da minha matriz  $A$  estão em **algum lugar** dentro desses círculos roxos. Mas qual é a utilidade disso? Na verdade é muito útil, pois nos dá uma noção de **shifts** para utilizarmos em algoritmos

**Teorema 11.1:** Os autovalores de uma matriz complexa  $A = [a_{ij}] \in \mathbb{C}^{m \times m}$  estão contidos na união dos discos:

$$\bigcup_{i=1}^m \left\{ z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j \neq i} |a_{ij}| \right\} \quad (128)$$

*Demonstração:*

□

## **12 Lecture 31 - Calculando a SVD**

## 12.1 SVD de A via autovalores de $A^*A$

Calcular a SVD de  $A$  usando que  $A^*A = V\Sigma^*\Sigma V$  igual a um sagui disléxico não é a melhor ideia, pois reduzimos o problema de SVD a um problema de autovalores, que é sensível à perturbações.

Um algoritmo estável para calcular a SVD de  $A$ , usa a matriz

$$H = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \quad (129)$$

Se  $A = U\Sigma V^*$  é uma SVD de  $A$ , então  $AV = \Sigma U$  e  $A^*U = \Sigma^*V = \Sigma V$ , portanto

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \cdot \begin{pmatrix} V & V \\ U & -U \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \cdot \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \quad (130)$$

Ou:

$$H = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \cdot \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \cdot \begin{pmatrix} V & V \\ U & -U \end{pmatrix}^{-1} \quad (131)$$

É uma decomposição em autovalores de  $H$ , e fica claro que os autovalores de  $H$  são os valores singulares de  $A$ , em módulo.

Agora note que ao calcular os autovalores de  $H$ , pagamos  $\kappa(A)$ , e não  $\kappa^2(A)$ , Pois

$$\kappa(H) = \|H\|_2 \cdot \|H^{-1}\|_2 = \frac{\sigma_1(H)}{\sigma_m(H)} = \frac{\sigma_1(A)}{\sigma_m(A)} = \kappa(A). \quad (132)$$