FGV EMAp

João Pedro Jerônimo e Arthur Rabello Oliveira

# Algebra Linear Numérica

## Revisão para A2

Rio de Janeiro

2025

# Contents

**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 $\tilde{f} : X \to Y$ é **backwards stable**, você tem que mostrar que, ao aplicar $\tilde{f}$ em uma entrada $x$, o resultado retornado seria o mesmo que aplicar o problema original $f : X \to Y$ em uma entrada levemente perturbada $x + \Delta x$, de forma que $\Delta x = O(\varepsilon_{\text{machine}})$.

## 1.1 O Experimento

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

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

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

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

**CÓDIGO**

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

**SAÍDA**

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

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

**CÓDIGO**

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

**SAÍDA**

```
3.8022328832723555e-14
```

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

**CÓDIGO**

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

**SAÍDA**

```
0.05197521348918455
```

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

## 1.2 Teorema

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

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

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

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

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

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

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

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

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

Tal que:

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

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

## 1.3 Algoritmo para resolver $Ax = b$

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

```
1 function ResolverSistema(A ∈ ℂ^{m×n}, b ∈ ℂ^{m×1}) {
2     QR = 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

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

com

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Perceba que

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

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

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

O que nos indica que

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

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

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

- $(\delta Q)(\delta R)$: Por último:

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

Ou seja, todos os termos de $\Delta A$ 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:

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

□

# 2 Lecture 17 - Estabilidade da Back Substitution

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

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

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

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

$$x_m = b_m / r_{mm}$$
$$x_{m-1} = \left(b_{m-1} - x_m r_{m-1,m}\right)/r_{m-1,m-1}$$
$$x_{m-2} = \left(b_{m-2} - x_{m-1} r_{m-2,m-1} - x_m r_{m-2,m}\right)/r_{m-2,m-2}$$
$$\vdots$$
$$x_j = \left(b_j - \sum_{k=j+1}^{m} x_k r_{jk}\right)/r_{jj}$$

}

Algoritmo 2: Algoritmo de **Back Substitution**

## 2.1 Teorema da Estabilidade Retroativa (Backward Stability)

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

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

**CÓDIGO** Python

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

**SAÍDA**

```
-1.0
```

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

**CÓDIGO** Python

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

**SAÍDA**

```
0.0
```

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

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

**Teorema 2.1.1**: Deixe o Algoritmo 2 ser aplicado a um problema de $Rx = b$ com $R$ triangular superior em um **computador ideal**. Esse algoritmo é **backward stable**, ou seja, a solução $\tilde{x}$ computada satisfaz:

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

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

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

*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:

  $$\widetilde{x_1} = b_1 \oslash r_{11} \tag{22}$$

  E nós **já sabemos** que essa divisão é backward stable, mas vamos analisar melhor. Queremos manter $b$ fixo, então temos que expressar $\widetilde{x_1}$ como o $r_{11}$ original vezes uma leve perturbação. Expressamos então

  $$\widetilde{x_1} = \frac{b_1}{r_{11}}(1 + \varepsilon_1) \tag{23}$$

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

  $$\begin{gathered} \widetilde{x_1} = \frac{b_1}{r_{11}(1+\varepsilon_1')} \Leftrightarrow \widetilde{x_1} = \frac{b_1}{r_{11}\left(1 - \frac{\varepsilon_1}{1+\varepsilon_1}\right)} \Leftrightarrow \widetilde{x_1} = \frac{b_1}{r_{11}\frac{1+\varepsilon_1-\varepsilon_1}{1+\varepsilon_1}} \\ \Leftrightarrow \widetilde{x_1} = \frac{b_1}{r_{11}\frac{1}{1+\varepsilon_1}} \Leftrightarrow \widetilde{x_1} = \frac{b_1}{r_{11}}(1+\varepsilon_1) \end{gathered} \tag{24}$$

  Se fizermos a expansão de taylor de $\varepsilon_1'$, conseguimos ver:

  $$-\frac{\varepsilon_1}{1+\varepsilon_1} = -\varepsilon_1 + \varepsilon_1^2 - \varepsilon_1^3 + \varepsilon_1^4 - \ldots \tag{25}$$

  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

  $$(r_{11} + \delta r_{11})\widetilde{x_1} = b_1 \tag{26}$$

  Com

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

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

  $$\widetilde{x_1} = (b_1 \ominus (\widetilde{x_2} \otimes r_{12})) \oslash r_{22} \tag{28}$$

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

  $$\widetilde{x_1} = \frac{(b_1 - \widetilde{x_2} r_{12}(1+\varepsilon_2))(1+\varepsilon_3)}{r_{22}}(1+\varepsilon_4) \tag{29}$$

  Aqui eu não iniciei os epsilons em $\varepsilon_1$ porque eu estou tomando intrínseco que esse $\varepsilon_1$ ta no $\widetilde{x_2}$ que a gente computa antes de computar o $\widetilde{x_1}$ (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:

$$\widetilde{x_1} = \frac{b_1 - \widetilde{x_2} r_{12}(1+\varepsilon_2)}{r_{22}(1+\varepsilon_3')(1+\varepsilon_4')} \tag{30}$$

(Mesmo racicocínio 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

$$\widetilde{x_1} = \frac{b_1 - \widetilde{x_2} r_{12}(1+\varepsilon_2)}{r_{22}(1+2\varepsilon_5')} \tag{31}$$

Esse $2\varepsilon_5$ se dá pois, como vimos no caso $1 \times 1$:

$$\begin{aligned} 1 + \varepsilon_3' &= 1 - \varepsilon_3 + O(\varepsilon_3^2) \\ 1 + \varepsilon_4' &= 1 - \varepsilon_4 + O(\varepsilon_4^2) \\ \Rightarrow (1+\varepsilon_3')(1+\varepsilon_4') &= (1 - \varepsilon_3 + O(\varepsilon_3^2))(1 - \varepsilon_4 + O(\varepsilon_4^2)) \\ \Rightarrow 1 - \varepsilon_4 + O(\varepsilon_4^2) - \varepsilon_3 + \varepsilon_3\varepsilon_4 - \varepsilon_3 O(\varepsilon_4^2) + O(\varepsilon_3^2) - \varepsilon_4 O(\varepsilon_3^2) + O(\varepsilon_4^2)O(\varepsilon_3^2) & \end{aligned} \tag{32}$$

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

$$1 - \varepsilon_4 - \varepsilon_3 = 1 - 2\varepsilon_5 \tag{33}$$

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

$$(R + \delta R)\widetilde{x_1} = b_1 \tag{34}$$

De forma que

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

- **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:

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

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

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

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

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

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 Quadrados

**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?

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

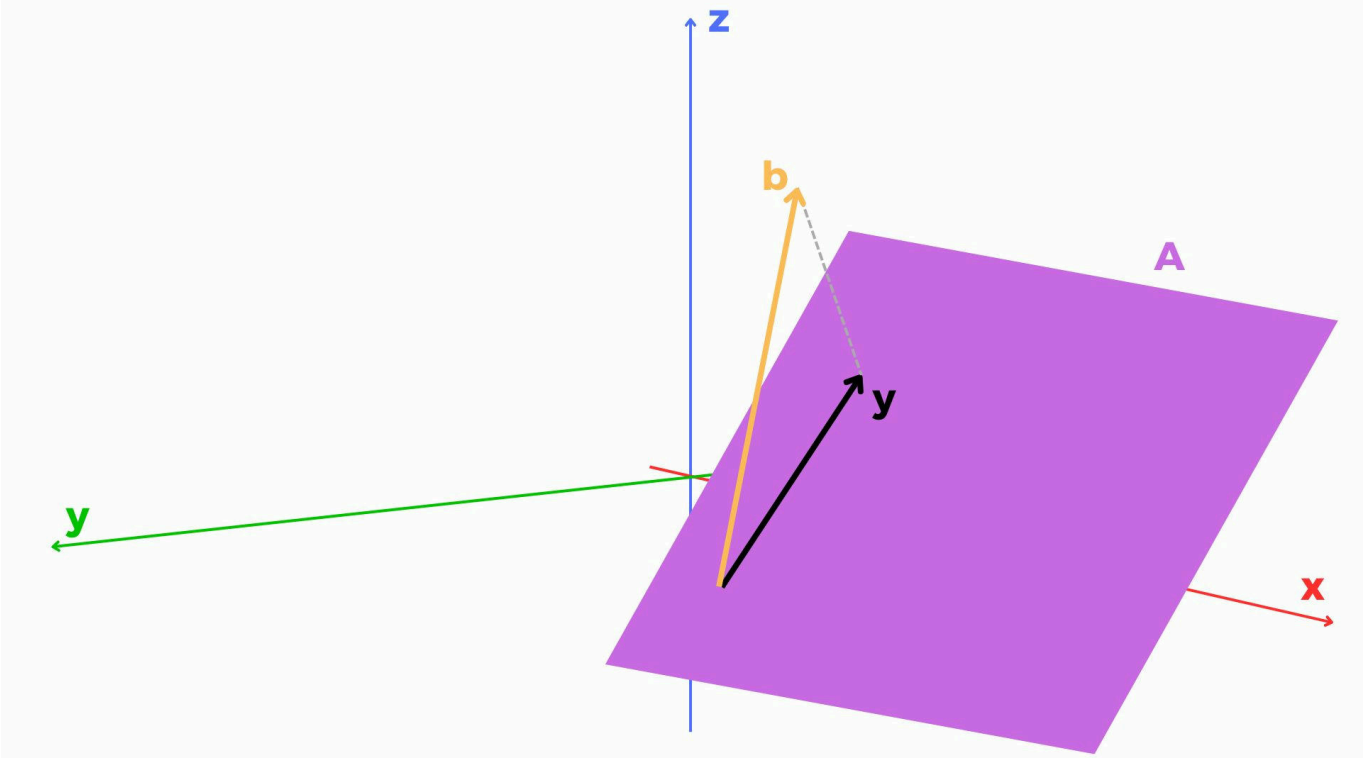
No resumo passado, vimos que o $x$ que satisfaz esse problema é

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

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

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

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

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

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

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

$$\eta = \frac{\|A\|\ \|x\|}{\|y\|} = \frac{\|A\|\ \|x\|}{\|Ax\|} \tag{43}$$

Show! E esses parâmetros tem esses domínios:

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

**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: Sesibilidade 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 perca 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)

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

Reescrevendo os outros termos, temos:

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

- **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:

  $$\frac{\|A\|}{\|x\|/\|b\|} \tag{47}$$

  Então, fazendo simples substituições:

  $$\frac{\|P\|}{\|y\|/\|b\|} = \frac{1}{\cos\theta} \tag{48}$$

  O que até que faz sentido na intuição. Se fazemos com que $b$ fique muito próximo a um ângulo de 90° 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:

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

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 pouquinha 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:

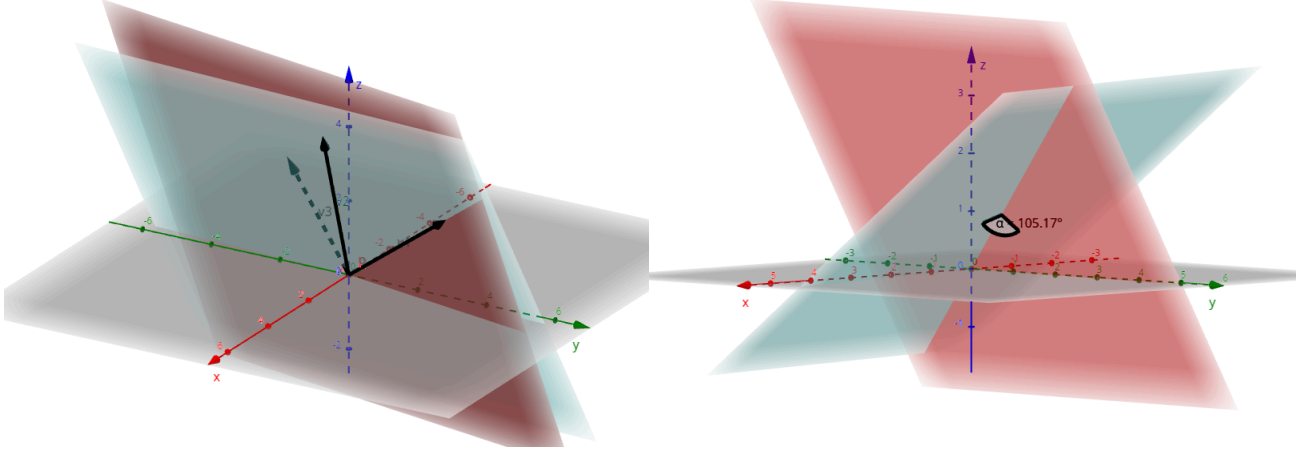



Figura 1: Perturbação em $C(A)$. $v_1$ é o vetor que está na divisão entre o plano azul e o vermelho, $v_2$ é o vetor mais destacado no plano azul e $v_3$ é o vetor pontilhado

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:

$$A = uv^* \Rightarrow \|Ax\| = \|uv^*x\| \leq \|u\|\|v\|\ x\| \tag{50}$$

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 valor singular de $A$ e $u_n$ a $n$-ésima coluna de $U$. Se tomarmos $A = \Sigma$, $p$ é a última coluna de $A$, $v^* = e_n^* = (0, 0, ..., 1)$ 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:

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

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 $y - b$.

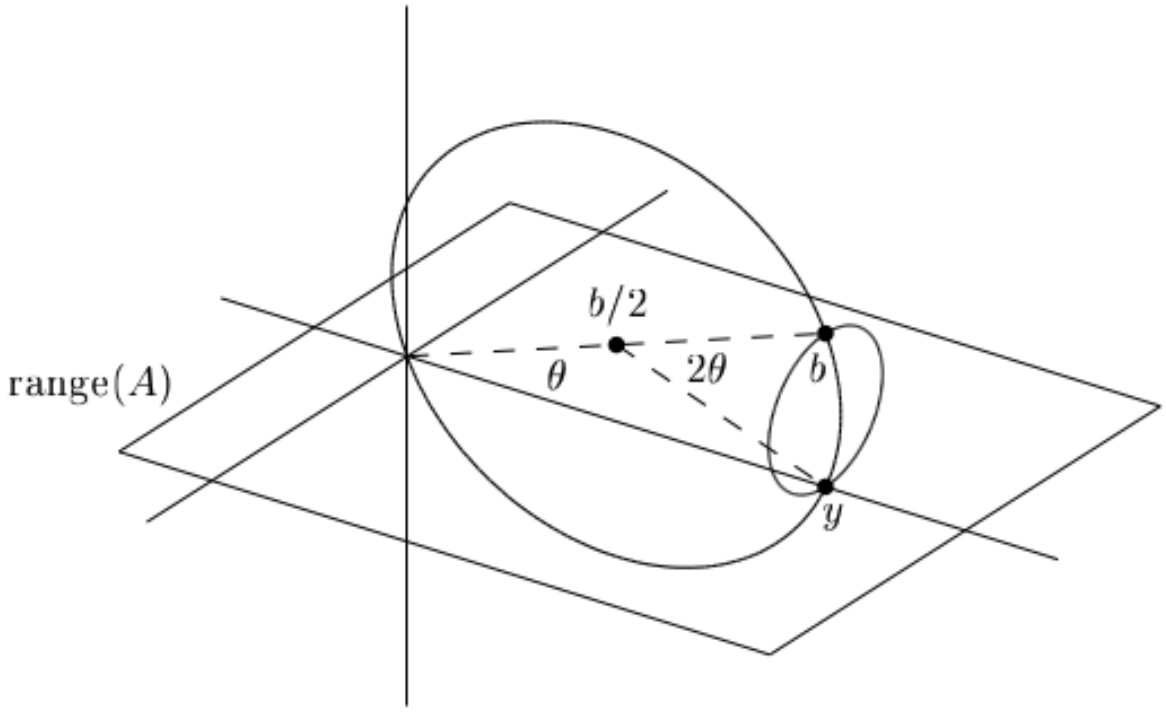



Figura 2: Círculo de projeção de $y$. O círculo maior representa a inclinação de $C(A)$ no plano $0yb$ e o círculo menor é quando inclinamos $C(A)$ em uma direção ortogonal a ele

Como eu posso rotacionar $C(A)$ em 360°, 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 isóceles.

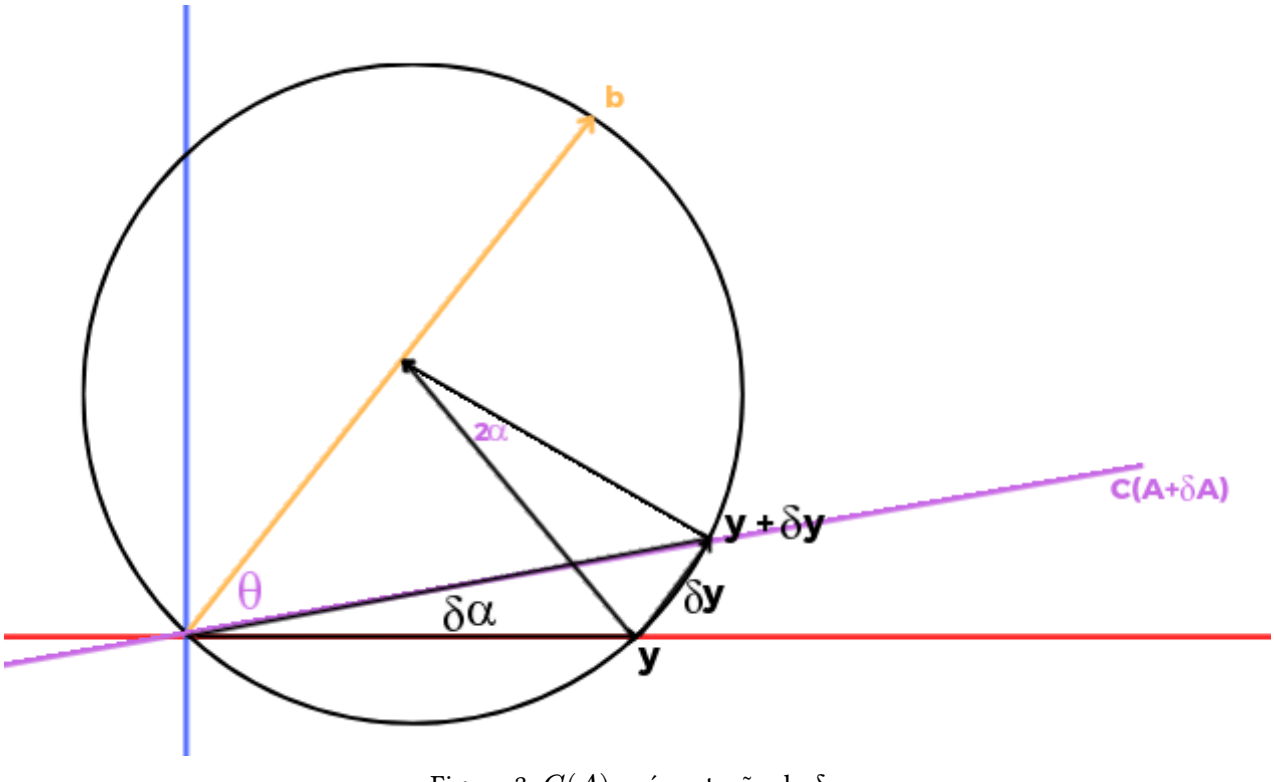



Figura 3: $C(A)$ após rotação de $\delta\alpha$

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

$$\|\delta y\| \leq \|b\| \sin(\delta\alpha) \leq \|b\|(\delta\alpha) \leq \|b\| \frac{\|\delta A\|}{\|A\|} \kappa(A)$$

$$\cos(\theta) = \frac{\|y\|}{\|b\|} \Leftrightarrow \|b\| = \frac{\|y\|}{\cos(\theta)} \tag{52}$$

$$\Rightarrow \|\delta y\| \leq \frac{\|y\|}{\cos(\theta)} \frac{\|\delta A\|}{\|A\|} \kappa(A) \Leftrightarrow \frac{\frac{\|\delta y\|}{\|y\|}}{\frac{\|\delta A\|}{\|A\|}} = \frac{\kappa(A)}{\cos(\theta)}$$

Concluímos 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, consequentemente, 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 $0by$, $\delta y$ 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 $0by$ (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)

$$\delta\alpha \leq \frac{\|\delta A_2\|}{\|A\|}\kappa(A) \Leftrightarrow \frac{\|\delta b_1\|}{\|b_1\|} \leq \frac{\|\delta A_2\|}{\|A\|}\kappa(A)\tan(\theta)$$

$$\left(\frac{\|\delta x\|}{\|x\|}\right) / \left(\frac{\|\delta b_1\|}{\|b_1\|}\right) \leq \frac{\kappa(A)}{\eta} \Leftrightarrow \frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{\eta}\frac{\|\delta b_1\|}{\|b_1\|} \Leftrightarrow \frac{\|\delta x\|}{\|x\|} \leq \frac{\kappa(A)}{\eta}\frac{\|\delta A_2\|}{\|A\|}\kappa(A)\tan(\theta) \tag{58}$$

$$\Leftrightarrow \left(\frac{\|\delta x\|}{\|x\|}\right) / \left(\frac{\|\delta A_2\|}{\|A\|}\right) \leq \frac{\kappa(A)^2\tan(\theta)}{\eta}$$

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

```python
import numpy as np
m = 100
n = 15
t = np.linspace(0, 1, m)
A = np.vander(t, n, True)
b = np.exp(np.sin(4*t))/2.00678728e+03
```

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 `1.42775025e+07`, 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

```python
Q, R = householder_qr(A)
x = np.linalg.solve(R, Q.T @ b)
print(1-x[-1])  # Erro relativo
```

**SAÍDA**

```
1.9845992627054443e-09
```

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

```python
Q, R = householder_qr(np.c_[A, b])
print(R.shape)
Qb = R[0:n, n]
R = R[0:n, 0:n]
x = np.linalg.solve(R, Qb)
print(1-x[-1])
```

**SAÍDA**

```
1.989168163518684e-09
```

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:

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

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

```python
Q, R = modified_gram_schmidt(A)
x = np.linalg.solve(R, Q.T @ b)
print(1-x[-1])
```

**SAÍDA**

```
-0.01726542
```

Meu amigo, esse erro é **terrível**. O resultado obtido é tenebroso de ruim. O livro comenta também de outro método que involve fazer umas manipulações em $Q$, mas como o próprio diz que involve 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

```python
Q, R = modified_gram_schmidt(np.c_[A, b])
Qb = R[0:n, n]
R = R[0:n, 0:n]
x = np.linalg.solve(R, Qb)
print(1-x[-1])
```

**SAÍDA**

```
-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

```python
x = np.linalg.solve(A.T @ A, A.T @ b)
print(1-x[-1])
```

**SAÍDA**

```
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:

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

Suponha que $A$ é mal-condicionada. Dependendo dos valores dos híperparâmetros, 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 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

```
7    U, S, Vh = np.linalg.svd(A, full_matrices=False)
8    S = np.diag(S)
9    x = (Vh.T * 1/S) @ (U.T @ b)
10   print(1-x[-1])
```

**SAÍDA**

```
1    -2.3301211e-07
```

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

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

## 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 $\lambda x \in \mathbb{C}^n$, concluimos 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:

$$Ax = \lambda x \tag{61}$$

$\lambda \in \mathbb{C}$ 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:

$$A = X\Lambda X^{-1} \tag{62}$$

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

Isso é equivalente a:

$$\underbrace{\begin{pmatrix} & A & \end{pmatrix}}_{A} \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & \dots & x_n \\ | & | & | & | \end{pmatrix}}_{X} = \underbrace{\begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \end{pmatrix}}_{\Lambda} \cdot \underbrace{\begin{pmatrix} | & | & | & | \\ x_1 & x_2 & \dots & x_n \\ | & | & | & | \end{pmatrix}}_{X} \tag{63}$$

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:

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

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):

$$Ax = \lambda x \Leftrightarrow Ax - \lambda x = 0 \Leftrightarrow (A - \lambda I)x = 0 \tag{65}$$

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

$$\det(A - \lambda I) = 0 \tag{66}$$

**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 $\lambda_1, ..., \lambda_n$ são raízes de (66), então podemos escrever isso como:

$$p(\lambda) = (\lambda - \lambda_1)(\lambda - \lambda_2)...(\lambda - \lambda_n) \tag{67}$$

(Nota: $\lambda$ é uma variável, enquanto $\lambda_j$ é uma raíz 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:

$$X^{-1}AX = X^{-1}BX \tag{68}$$

**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*:

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

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 □

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 gemétrica 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 $\tilde{V}$ para uma matriz ortogonal quadrada, temos:

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

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

$$\det(\mu I - B) = \det(\mu I - \lambda I)\det(\mu I - D) = (\mu - \lambda)^n \det(\mu I - D) \tag{71}$$

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

## 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 $A = X\Lambda X^{-1}$ □

## 5.7 Determinante e Traço

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

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

*Demonstração*:

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

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, $-\operatorname{tr}(A)$, ou seja, $\operatorname{tr}(A) = \sum_{j=1}^{m} \lambda_j$ □

## 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 Álebra 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:

$$A = QTQ^* \tag{74}$$

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:

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

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

$$Q = U\begin{pmatrix} 1 & 0 \\ 0 & V \end{pmatrix} \tag{76}$$

$Q$ é uma matriz unitária e temos que

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

Essa era a fatoração de Schur que procurávamos

□

# 6 Lecture 25 - Algoritmos de Autovalores

## 6.1 Ideia da Iteração de Potência

## 6.2 A ideia dos Algoritmos de Autovalores

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

## 6.3 Forma de Schur e Diagonalização

## 6.4 As 2 fases do Cálculo de Autovalores, Forma de Hessenberg

# 7 Lecture 26 - Redução à forma de Hessenberg

## 7.1 A Redução

## 7.2 Redução à Hessenberg via Householder

## 7.3 Custo Computacional

## 7.4 O Caso Hermitiano

## 7.5 Estabilidade do Algoritmo

# 8 Lecture 27 - Quociente de Rayleigh e Iteração Inversa

## 8.1 Restrição à matrizes reais e simétricas

## 8.2 Quociente de Rayleigh

## 8.3 Iteração de Potência com o Quociente de Rayleigh

## 8.4 Iteração Inversa

# 9 Lecture 31 - Calculando a SVD

## 9.1 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

$$H = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \tag{78}$$

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

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \cdot \begin{pmatrix} V & V \\ U & -U \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \cdot \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \tag{79}$$

Ou:

$$H = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \cdot \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \cdot \begin{pmatrix} V & V \\ U & -U \end{pmatrix}^{-1} \tag{80}$$

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

$$\kappa(H) = \|H\|_2 \cdot \left\|H^{-1}\right\|_2 = \frac{\sigma_1(H)}{\sigma_m(H)} = \frac{\sigma_1(A)}{\sigma_m(A)} = \kappa(A). \tag{81}$$