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

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

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

Perceba que

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

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

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

O que nos indica que

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

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

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

- $(\delta Q)(\delta R)$: Por último:

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

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

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

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

□

2 Lecture 17 - Estabilidade da Back Substitution

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

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

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

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

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

}

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 1×1, 2×2 e 3×3, de forma que o raciocínio que aplicarmos poderá ser aplicado para matrizes de tamanhos maiores.

- **1×1**: 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 ε_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

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

- **2×2**: Beleza, no caso 2×2, 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 ε_1 porque eu estou tomando intrínseco que esse ε_1 ta no $\widetilde{x_2}$ que a gente computa antes de computar o $\widetilde{x_1}$ (A gente computa igual o caso 1×1)

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 1×1). 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 1×1:

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

Os termos diferentes de 1, ε_3 e ε_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 $2\varepsilon_5$, ε_2 e ε_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 ε_{m+1}' e ε_{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?

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

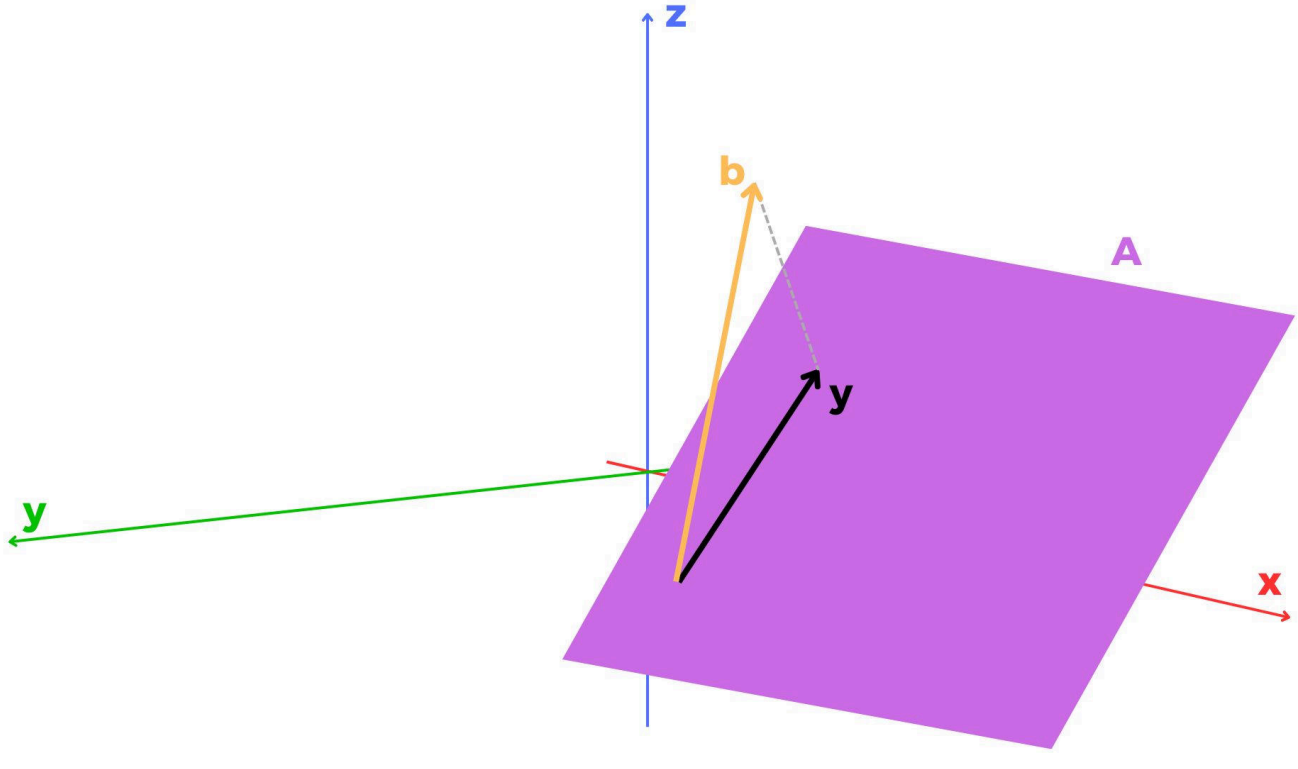
No resumo passado, vimos que o x que satisfaz esse problema é

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

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 θ entre b e y

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

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

E a segunda medida é η, 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: 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 perca da generalidade, lidar diretamente com Σ, 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 δA? Aí a gente pode seguir do seguinte modo:

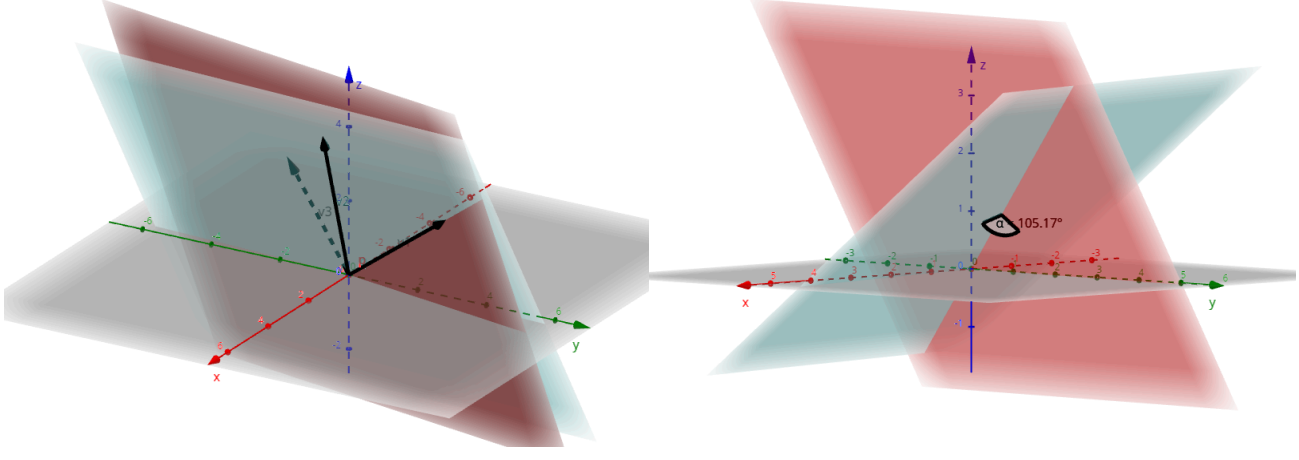


Figura 1: Perturbação em $C(A)$. v_1 é o vetor que está na divisão entre o plano azul e o vermelho, v_2 é o vetor mais destacado no plano azul e v_3 é o vetor pontilhado

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 δp 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 σ_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 δA são perturbações na entrada de A. Essa perturbação inclina $C(A)$ pelo ângulo δA 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 $y - b$.

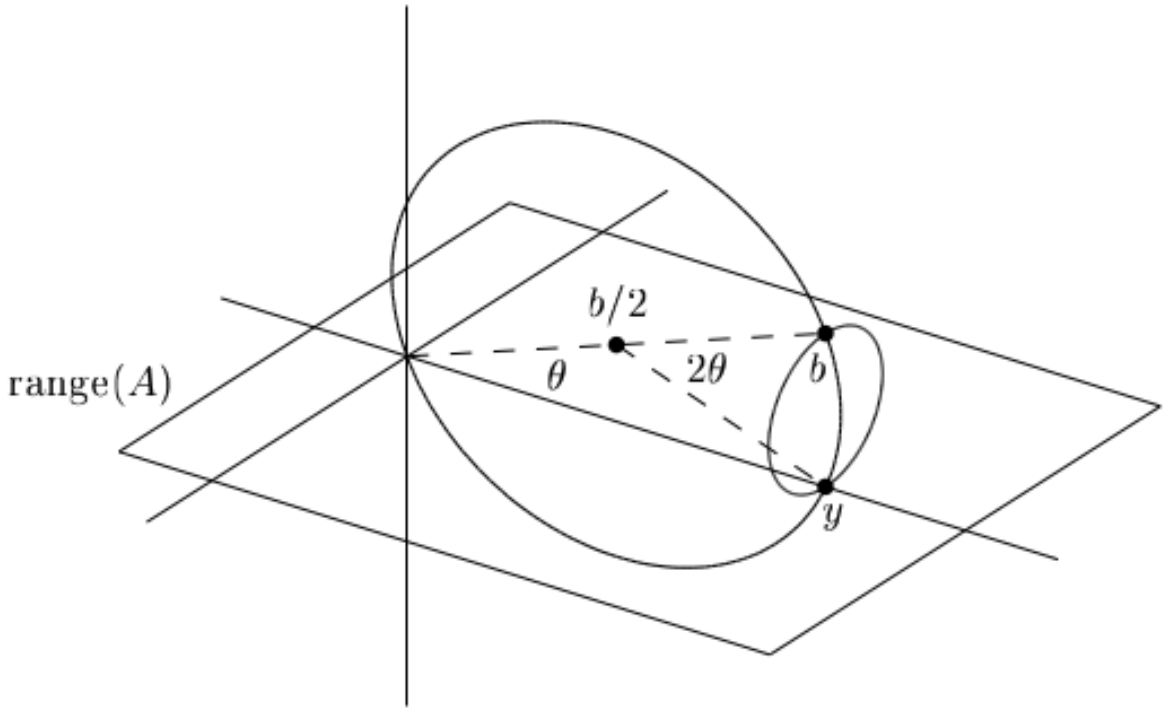


Figura 2: Círculo de projeção de y. O círculo maior representa a inclinação de $C(A)$ no plano $0yb$ e o círculo menor é quando inclinamos $C(A)$ em uma direção ortogonal a ele

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 2θ vai ser alterado. Mais especificamente, vai ser alterado em $2\delta\alpha$. Ou seja, a perturbação δy que eu vou obter ao inclinar $C(A)$ será a base de um triângulo isóceles.

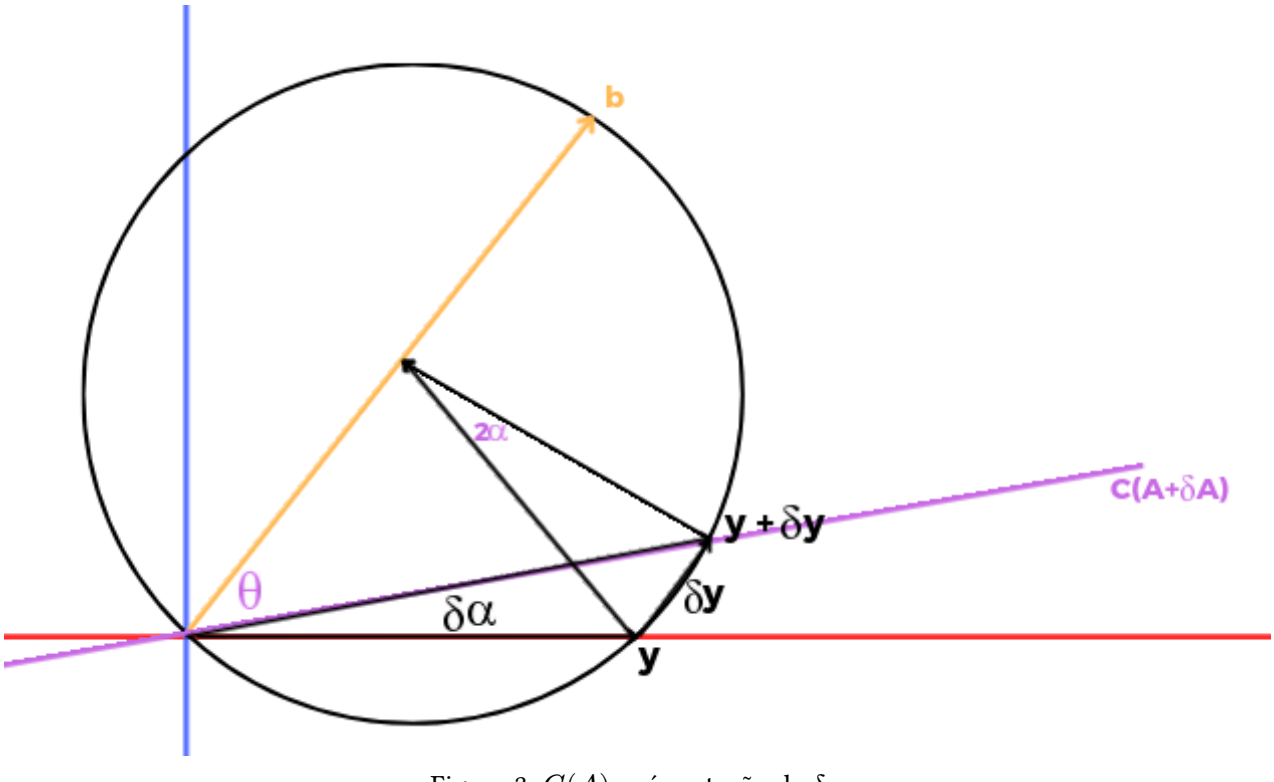


Figura 3: $C(A)$ após rotação de $\delta\alpha$

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 δA em A, podemos separar essa perturbação em duas outras: δA_1 que ocorre nas primeiras n linhas de A e δA_2 que ocorre nas $m - n$ linhas restantes.

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

Vamos ver δA_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 δA_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 δA_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 δb_1 com δA_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 $0by$, δy 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 `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

```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 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

```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 δA 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 η é próximo de κ, 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

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 $\lambda x \in \mathbb{C}^n$, concluimos que $m = n$: A matriz **deve ser quadrada**. Afinal, não faz sentido se λx e Ax estiverem em conjuntos distintos. Com isso, prosseguimos com a definição:

Definição 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 Λ é diagonal e $\det(X) \neq 0$.

Isso é equivalente a:

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

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 λ_i é o autovalor associado a x_i.

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

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

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

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 λ, e chamaremos isso de *multiplicidade geométrica* de λ. Então temos:

Definição 5.3.2: (Multiplicidade Geométrica) A multiplicidade geométrica de λ 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 λ. 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: λ é uma variável, enquanto λ_j é uma raíz do polinômio, fique atento)

Com isso, prosseguimos com:

Definição 5.3.4 (Multiplicidade Algébrica): A multiplicidade algébrica de λ é a multiplicidade de λ 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:

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

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_λ 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 λ é sempre maior ou igual a sua multiplicidade geométrica

Demonstração: Deixe n ser a multiplicidade gemétrica de λ 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:

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

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 λ 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 Λ sendo similar a A, logo, A tem os mesmos autovalores, MA e MG de Λ. Como Λ é diagonal, eu tenho que Λ 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 λ_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 λ^{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 Λ 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 quadrada** 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 λ. 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

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

$$\frac{x}{\|x\|}, \frac{Ax}{\|Ax\|}, \frac{A^2x}{\|A^2x\|}, ..., \frac{A^nx}{\|A^nx\|} \tag{78}$$

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 issos 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:

$$Q_j^*...Q_2^*Q_1^*AQ_1Q_2...Q_j \tag{79}$$

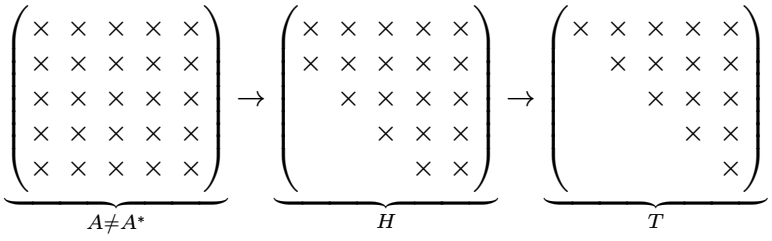
Converja para uma matriz triangular superior T conforme $j \to \infty$

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

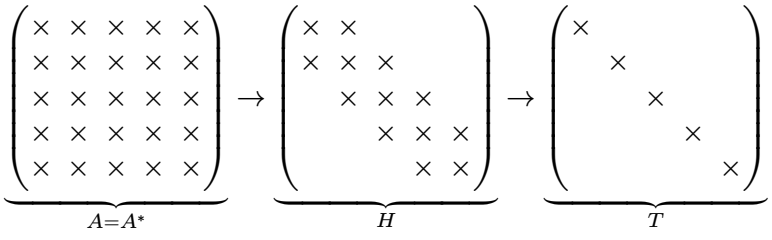
A sendo Hermitiana ou não, a gente separa a sequência (79) em duas partes.

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:



(80)

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



(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:

$$Q_1^* A Q_1 \tag{82}$$

Isso faz com que os zeros que a gente colocou antes se percam, e a gente obtem 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**:

$$\underbrace{\begin{pmatrix} x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \end{pmatrix}}_{A} \rightarrow \underbrace{\begin{pmatrix} x & x & x & x & x \\ \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ 0 & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ 0 & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ 0 & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \end{pmatrix}}_{Q_1^* A} \rightarrow \underbrace{\begin{pmatrix} x & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ x & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \\ & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} & \boldsymbol{x} \end{pmatrix}}_{Q_1^* A Q_1} \tag{83}$$

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

```
1 function HessenbergReduction(A ∈ ℂ^{m×m}) {
2   for k = 1 to m − 2
3     x = A_{k+1:m,k}
4     v_k = 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 QAQ^* é hermitiana. Inclusive, essa propriedade pose 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 $A = QTQ^*$ 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:

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

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

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

Perceba que se x é um autovetor de A com autovalor λ associado, então $r(x) = \lambda$. Uma motivação para essa fórmula é pensarmos no seguinte: Dado um x, qual escalar α "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:

$$x^T \alpha x = x^T A x \Rightarrow \alpha = r(x) \tag{86}$$

A gente pode fazer essas ideias mais quantitativas se tomarmos $r(x) : \mathbb{R}^m \to \mathbb{R}$, 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:

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

Podemos então expressar o gradiente como:

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

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.

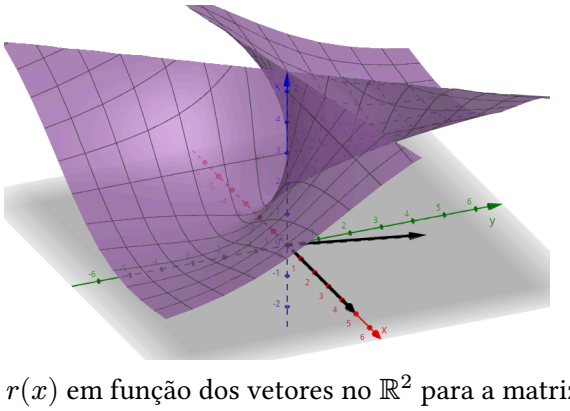


Figura 4: $r(x)$ em função dos vetores no $\mathbb{R}^2$ para a matriz $\begin{pmatrix} 5 & 4 \\ 0 & 3 \end{pmatrix}$

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 μx de x?

$$A(\mu x) = \lambda(\mu x) \tag{89}$$

μx **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 αx também o fará $\forall \alpha \in \mathbb{R}$. Não acredita em mim? Veja por conta própria:

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

Mas podemos contornar isso **limitando** o domínio de $r(x)$. Podemos fazer isso fazendo com $r(x) : \mathbb{R}^m \to \mathbb{R}$ 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.

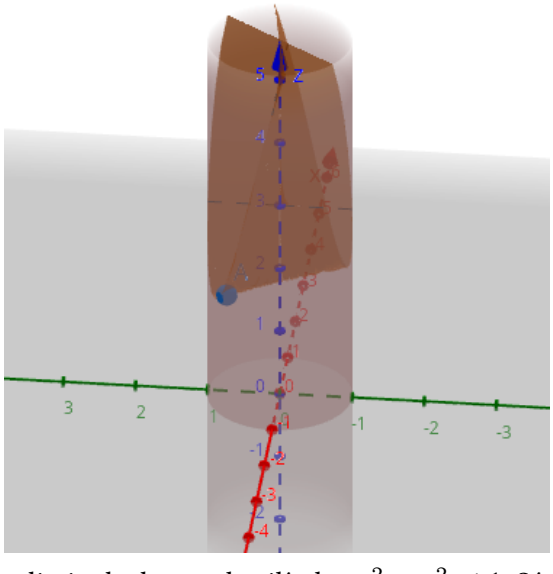


Figura 5: Mesma função da Figura 4 limitada dentro do cilíndro $x^2 + y^2 \leq 1$. Só não coloquei $= 1$ pois o Geogebra não conseguia fazer a plotagem

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

$$r(x) - r(q_j) = O(\|x - q_j\|^2), x \to q_j \tag{91}$$

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

```
function PowerIteration(A ∈ ℂ^{m×m}, v^(0) com ‖v^(0)‖ = 1) {
    for k = 1, 2, 3, ...
        w = A v^(k−1)
        v^(k) = w/‖w‖
        λ^(k) = (v^(k))^T A v^(k)
}
```

Algoritmo 4: Iteração por potências

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

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

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

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

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

A primeira equação se da ao fato de que, quando $\lim_{k\to\infty} \left(\frac{\lambda_j}{\lambda_1}\right)^k = 0$, porém, como λ_2 é o maior entre λ_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 μ **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 λ_j são os autovalores de A

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

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

□

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)

```
function ReverseIteration(A ∈ ℂ^{m×m}, v^(0) com ‖v^(0)‖ = 1) {
    for k = 1, 2, 3, ...
        Resolva (A − μI)w = v^(k−1) para w
        v^(k) = w/‖w‖
        λ^(k) = (v^(k))^T A v^(k)
}
```

Algoritmo 5: Iteração Inversa

Você pode estar se perguntando: "Mas e se μ for um autovalor de A? Isso vai fazer com que $A - \mu I$ não seja inversível! Ou de μ 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 μ que escolhermos, podemos encontrar um autovalor diferente, ou seja, podemos escolher qual autovalor encontrar se fizermos a escolha certa de μ

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

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

Conforme $k \to \infty$ 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 (μ converge para o mais próximo)

8.5 Iteração do Quociente de Rayleigh

Beleza, a gente ja bisoiou 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**.

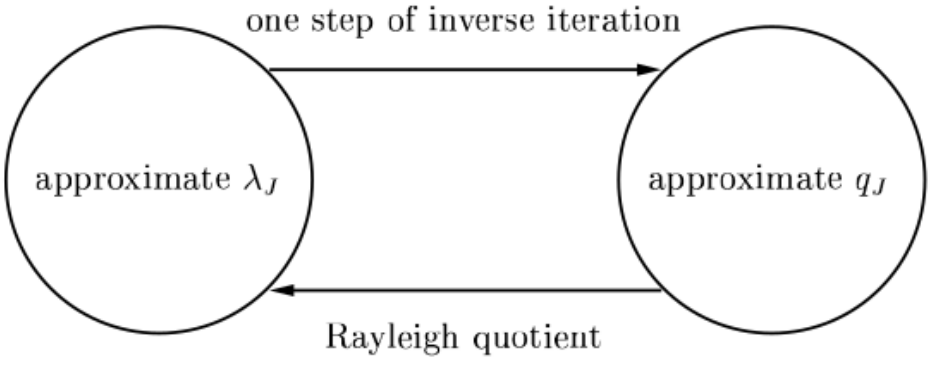


Figura 6: Iteração do Quociente de Rayleigh

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 ∈ ℂ^{m×m}) {
2   v^(0) com ‖v^(0)‖ = 1
3   λ^(0) = (v^(0))^T A v^(0)
4   for k = 1, 2, 3, ...
5     Resolva (A − λ^(k−1) I) w = v^(k−1) para w
6     v^(k) = w/‖w‖
7     λ^(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 λ_J e um autovetor q_J de A de forma que:

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

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.

```
function QRIteration(A ∈ ℂ^{m×m}) {
    A^(0) = A
    for k = 1, 2, 3, ...
        Q^(k), R^(k) = qr(A^(k−1))
        A^(k) = R^(k) Q^(k)
}
```

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 $\left(Q^{(k)}\right)^T A^{(k-1)}$, ou seja: $A^{(k)} = \left(Q^{(k)}\right)^T A^{(k-1)} Q^{(k)}$ (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

```
function ShiftedQR(A ∈ ℂ^{m×m}) {
    (Q^(0))^T A^(0) Q^(0) = A
    for k = 1, 2, 3, ...
        Escolha um shift μ^(k)
        Q^(k), R^(k) = qr(A^(k−1) − μ^(k) I)
        A^(k) = R^(k) Q^(k) + μ^(k) I
        Se qualquer elemento A^(k)_{j,j+1} fora da diagonal é suficientemente próximo de 0
            A_{j,j+1} = A_{j+1,j} = 0 para obter
            [A_1  0; 0  A_2] = A^(k)
            e agora aplicamos o algoritmo em A_1 e A_2
}
```

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. Vamo supor que a gente tem n vetores LI iniciais $v_1^{(0)}, ..., v_n^{(0)}$. Se a gente aplica $A^k v_1^{(0)}$, conforme $k \to \infty$, isso converge para o autovetor correspondente ao autovalor de maior valor absoluto (Com algumas condições adequadas), meio que parece plausível que $\text{span}\left\{A^k v_1^{(0)}, A^k v_2^{(0)}, ..., A^k v_n^{(0)}\right\}$ converge para $\text{span}\{q_1, ..., q_n\}$ 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:

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

E definimos

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

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 ta interessado em $C\left(V^{(k)}\right)$. 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 $C\left(V^{(k)}\right)$

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

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 ($\pm q_1, ..., \pm q_n$). 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

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

Mas não precisamos entrer 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

$$|\lambda_1| > |\lambda_2| > ... > |\lambda_n| > |\lambda_{n+1}| \geq |\lambda_{n+2}| \geq ... \geq |\lambda_m| \tag{101}$$

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

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

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 $k \to \infty$, as colunas da matriz $Q^{(k)}$ vão convergindo linearmente para os autovetores de A:

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

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 Λ 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 Λ com os autovalores associados a matriz $\hat{Q}$.

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

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

$$V^{(k)} = \Big(\hat{Q}\Lambda^k + O(|\lambda_{k+1}|)\big(\hat{Q}^T V^{(0)}\big)^{-1}\Big)\hat{Q}^T V^{(0)} \tag{105}$$

Como $\hat{Q}^T V^{(0)}$ é inversível, $C\big(V^{(k)}\big) = C\Big(\hat{Q}\Lambda^k + O(|\lambda_{k+1}|)\big(\hat{Q}^T V^{(0)}\big)^{-1}\Big)$. 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 □

9.3 Iteração Simultânea

Conforme $k \to \infty$, 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 $C\big(Z^{(k)}\big) = C\big(V^{(k)}\big)$

```
1 function SimultaniousAlgorithm(A ∈ ℂ^{m×m}) {
2 |  Escolha Q̂^(0) ∈ ℝ^{m×n} com colunas ortonormais
3 |  for k = 1, 2, 3, ...
4 |  |  Z = AQ̂^(k−1)
5 |  |  Q̂^(k), R̂^(k) = qr(Z) # Fatoração Reduzida
6 }
```

Algoritmo 9: Iteração Simultânea

Assim é mais tranquilo de ver que $C\big(Z^{(k)}\big) = C\big(\hat{Q}^{(k)}\big) = C\big(A^k \hat{Q}^{(0)}\big)$. 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 ⇔ 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 tudos, temos um algoritmo de iteração simultânea com uma leve adaptação, mostraremos que ele e o algoritmo qr são equivalentes

```
function ModifiedSimultaniousAlgorithm(A ∈ ℂ^{m×m}) {
    Q̲^(0) = I
    for k = 1, 2, 3, ...
        Z = A Q̲^(k−1)
        Q̲^(k), R^(k) = qr(Z)
        A^(k) = (Q̲^(k))^T A Q̲^(k)
        R̲^(k) = R^(k) R^(k−1) ... R^(1)
}
```

Algoritmo 10: Iteração Simultânea Modificada

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

```
function UnshiftedQRAlgorithm(A ∈ ℂ^{m×m}) {
    A^(0) = A
    for k = 1, 2, 3, ...
        Q^(k), R^(k) = qr(A^(k−1))
        A^(k) = R^(k) Q^(k)
        Q̲^(k) = Q^(1) Q^(2) ... Q^(k)
        R̲^(k) = R^(k) R^(k−1) ... R^(1)
}
```

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 $A^{(k)}$, de tal forma que:

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

junto da projeção

$$A^{(k)} = \left(\underline{Q}^{(k)}\right)^T A \underline{Q}^{(k)} \tag{107}$$

Demonstração: Vamos fazer indução em k

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

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

Agora, faremos o mesmo assumindo o algoritmo QR

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

Então verificamos que

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

□

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^TAQ, 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 $|\lambda_1| > |\lambda_2| > ... > |\lambda_m|$ e que a matriz de autovetores correspondente Q não tem blocos singulares (Todos os blocos da matriz formam matrizes inversíveis). Então, conforme $k \to \infty$, $A^{(k)}$ 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 $A \to A - \mu I$ e discutir por que que essa ideia nada intuitiva funciona e leva a uma convergência cúbica.

10.1 Conexao 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:

$$\underline{Q}^{(k)} = \prod_{j=1}^{k} Q^{(j)} = \left(q_1^{(k)} \;\middle|\; \dots \;\middle|\; q_m^{(k)} \right) \tag{111}$$

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

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

Se a gente inverte essa fórmula, temos

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

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:

$$P = \begin{pmatrix} & & 1 \\ & \iddots & \\ 1 & & \end{pmatrix} \tag{114}$$

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

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

Perceba que $\underline{Q}^{(k)} P$ é ortogonal ($\underline{Q}^{(k)}$ é ortogonal e P também) e $P\left(\underline{R}^{(k)}\right)^{-T} P$ é triangular superior ($\left(\underline{R}^{(k)}\right)^{-T}$ é triangular inferior, daí eu inverto 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 agloritmo 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 mistureba 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):

```
function ShiftedQR(A ∈ ℂ^{m×m}) {
    (Q^(0))^T A^(0) Q^(0) = A
    for k = 1, 2, 3, ...
        Escolha um shift μ^(k)
        Q^(k), R^(k) = qr(A^(k-1) − μ^(k) I)
        A^(k) = R^(k) Q^(k) + μ^(k) I
        ...
}
```

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:

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

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

$$A^{(k)} = \left(Q^{(k)}\right)^T A^{(k-1)} Q^{(k)} \tag{117}$$

Aí se a gente aplica uma indução, temos:

$$A^{(k)} = \left(\underline{Q}^{(k)}\right)^T A \underline{Q}^{(k)} \tag{118}$$

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 substituida por:

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

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 da uma ênfase na primeira e na última coluna, onde cada uma é equivalente a apliar 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?

$$\mu^{(k)} = \frac{\left(q_m^{(k)}\right)^T A q_m^{(k)}}{q_m^{((k))^T} q_m^{(k)}} = \left(q_m^{(k)}\right)^T A q_m^{(k)}) \tag{120}$$

Se escolhermos esse valor, as estimativas $\mu^{(k)}$ (Estimativa de autovalor) e $q_m^{(k)}$ estimativa de autovetor são identicos 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 $r\left(q_m^{(k)}\right)$ (r sendo a função do quociente de rayleigh), a gente pode visualizar assim:

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

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 $r\left(q_m^{(k)}\right)$ como os shifts para convergência funciona. Um exemplo disso é a matriz:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{122}$$

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 ta igualmente distante dos dois). A gente precisa de uma estimativa que quebre a simetria, vamo fazer o seguinte então:

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

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

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:

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

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 partiular, 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:

$$\tilde{Q}\tilde{\Lambda}\tilde{Q} = A + \delta A \tag{125}$$

onde

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

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

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 λ_j, o autovalor computado $\tilde{\lambda_j}$ satisfaz:

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

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 $A \in \mathbb{C}^{3\times3}$, 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:

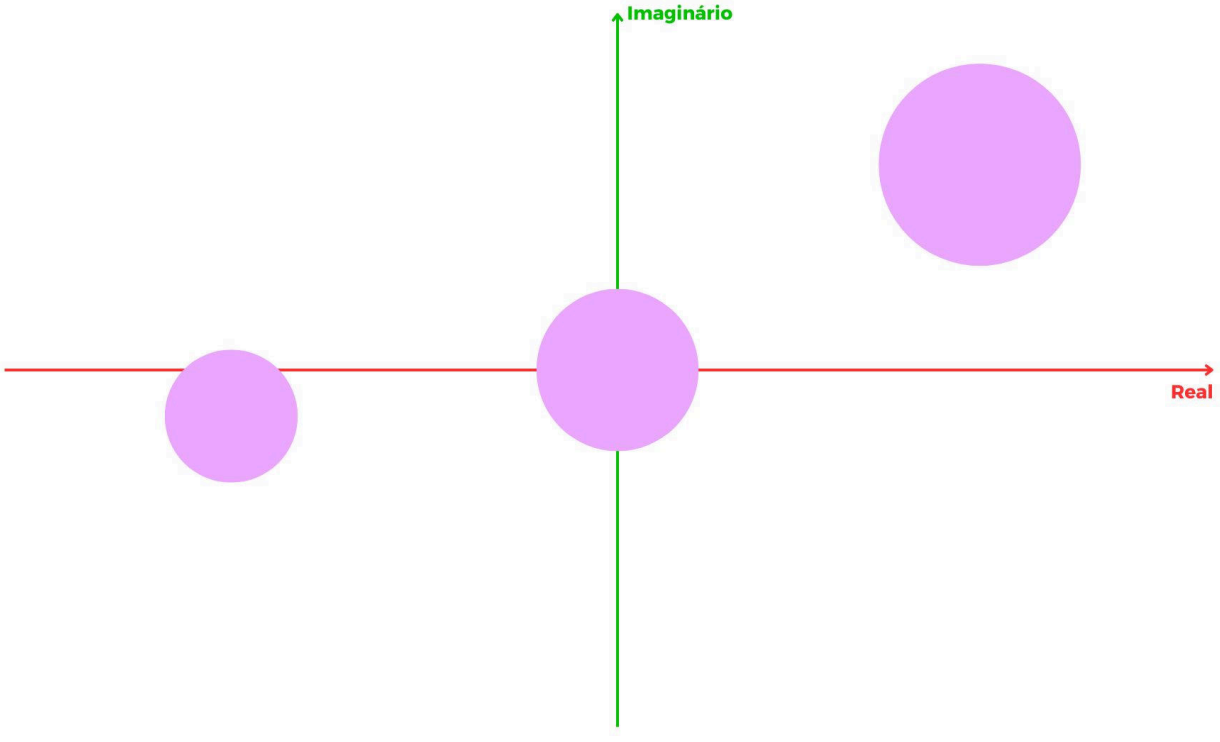


Figura 7: Ilustração dos Discos de Gershgorin de uma matriz 3×3

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 $A = \left[a_{ij}\right] \in \mathbb{C}^{m\times m}$ estão contidos na união dos discos:

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

Demonstração: Dada uma matriz $A \in \mathbb{C}^{m\times m}$ e um autovetor v de A tal que $Av = \lambda v$ e seja v_i a entrada de maior magnitude de v, temos:

$$\begin{aligned}
\sum_{j=1}^{m} A_{ij}v_j &= \lambda v_i \\
A_{ii}v_i + \sum_{j\neq i}^{m} A_{ij}v_j &= \lambda v_i \\
\sum_{j\neq i}^{m} A_{ij}v_j &= \lambda v_i - A_{ii}v_i \\
\frac{1}{v_i}\sum_{j\neq i}^{m} A_{ij}v_j &= \lambda - A_{ii} \\
\left|\frac{1}{v_i}\sum_{j\neq i}^{m} A_{ij}v_j\right| &= |\lambda - A_{ii}|
\end{aligned} \tag{129}$$

Por desigualdade triangular, reescrevemos como:

$$\sum_{j\neq i}^{m}|A_{ij}\frac{v_j}{v_i}| \geq |\lambda - A_{ii}| \tag{130}$$

Veja que, como v_i é a entrada de maior magnitude de v, temos que $|\frac{v_j}{v_i}| \leq 1 \ \forall j$. Isso quer dizer que:

$$\sum_{j\neq i}^{m}|A_{ij}\frac{v_j}{v_i}| \leq \sum_{j\neq i}^{m}|A_{ij}| \tag{131}$$

Ou seja, podemos reescrever como:

$$\sum_{j\neq i}^{m}|A_{ij}| \geq |\lambda - A_{ii}| \tag{132}$$

Isso quer dizer que o autovalor λ está localizado dentro de um disco com centro A_{ii} e raio $\sum_{j\neq i}^{m}|A_{ij}|$ $\square$

12 Lecture 30 - Outros algoritmos de Autovalores

12.1 Algoritmo de Jacobi

A gente pode imaginar uma matriz A como uma representação de um elipsóide num plano. Tente imaginar no plano 3D. Se a gente conseguir rotacionar essa matriz A (Rotacionar a elipe) até o ponto de que os eixos da elipse se alinhem com os eixos do plano, então A seria diagonal (Ou seja, a gente obteria uma diagonalização de A).

Para uma melhor vizualisação desse conceito, veja esse vídeo

Ok, então o que podemos fazer pra ir fazendo isso? Vamos aplicando pequenas rotações 2×2 até obter o resultado designado, as rotações são do tipo:

$$\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \tag{133}$$

onde $c = \cos(\theta)$ e $s = \sin(\theta)$ para algum θ. Vale dizer também que essa rotação é para o caso de $A \in \mathbb{R}^{2\times 2}$. Se A é uma matriz de dimensão maior, então a matriz de rotação é a identidade com um bloco do tipo que apresentei antes em algum lugar (Qualquer lugar da matriz). No final a gente teria uma matriz J tal que:

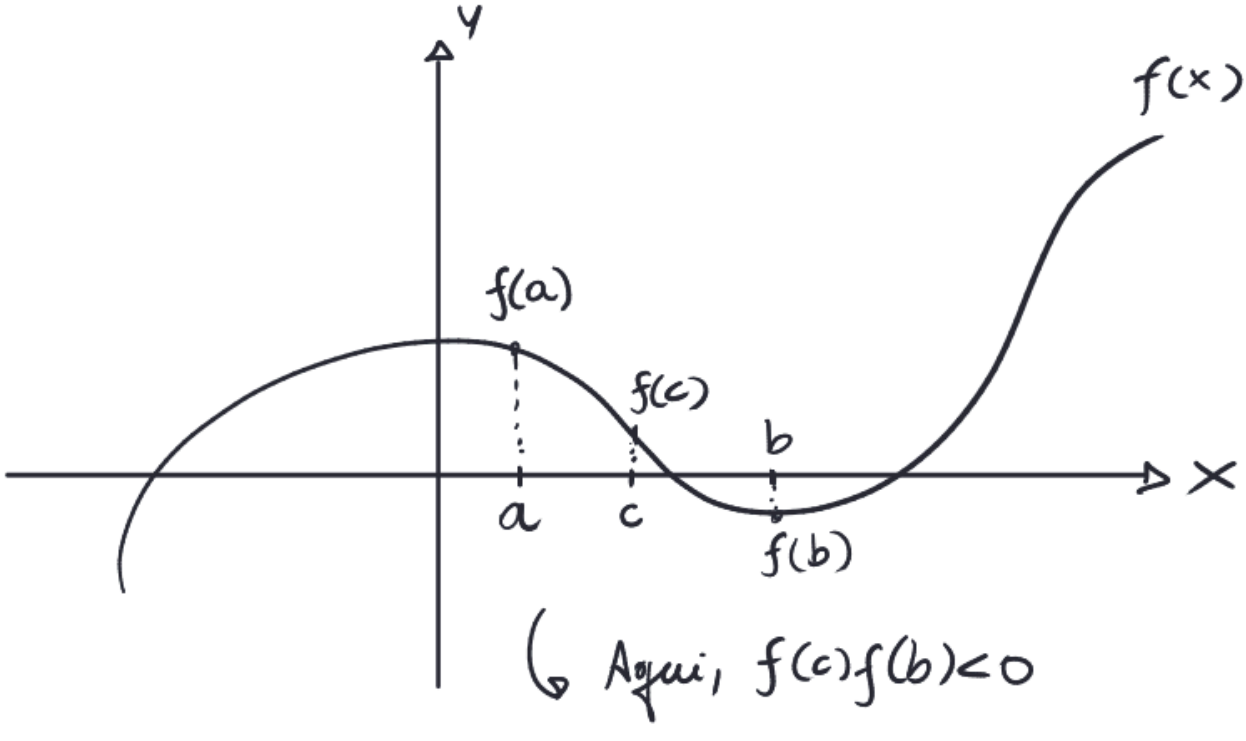
$$J^T A J = \text{Diagonal} \tag{134}$$

12.2 Bisection

Antes de tudo, vou explicar o que é o algoritmo de Bisection. Ele é um algoritmo pra estimar as raízes de uma função.

Temos uma função $f(x)$ e queremos estimar suas raízes. Então pegamos uma região $[a, b]$ de forma que $f(a) < 0$ e $f(b) > 0$ (Ou o contrário). Pelo TVI, isso significa que uma raíz $f(\delta) = 0$ é tal que $\delta \in [a, b]$. Pegamos então o ponto médio do intervalo $c = \frac{a+b}{2}$ e calculamos $f(c)$. Daí, fazemos a seguinte análise:

- Se $f(a)f(c) < 0$, então a raíz δ está a esquerda de c, então eu vou fazer o processo novamente no intervalo $[a, c]$. Se não, então a raíz não está no intervalo $[a, c]$
- Se $f(b)f(c) < 0$, então a raíz δ está a direita de c, então eu vou fazer o processo novamente no intervalo $[c, b]$. Se não, então a raíz não está no intervalo $[c, b]$



Essa é a ideia para achar os autovalores, aplicamos isso no polinômio característico. Ué, mas usar o polinômio não era uma ideia ruim de autovalor? Na real que a ideia ruim é achar a raíz do polinômio pelos seus **coeficientes**, isso sim é instável. No método de bisection a gente não precisa calcular isso.

Vamos definir algumas coisas antes de continuar com o algoritmo.

Chame de $A^{(j)}$ a submatriz principal de A com tamanho $j \times j$ e tenha que $A \in \mathbb{R}^{m\times m}$ é tridiagonal, simétrica e não-redutível (0 fora da diagonal, com exceção das diagonais superior e inferior)

$$A = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{m-1} \\ & & & b_{m-1} & a_m \end{pmatrix} \tag{135}$$

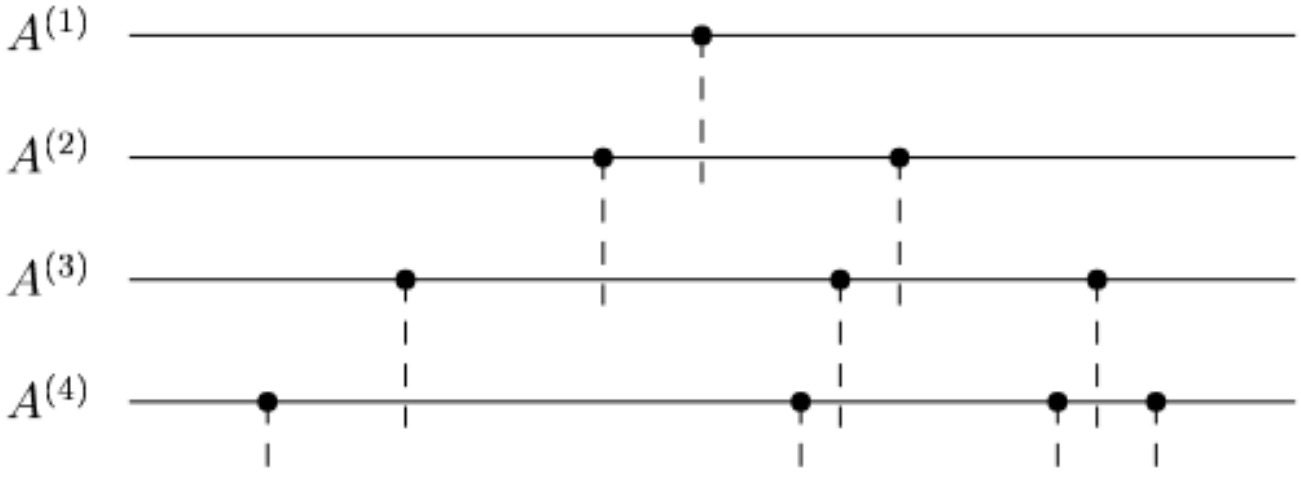
Tenha também o seguinte teorema (É um exercício do livro)

Teorema 12.2.1: Se $A \in \mathbb{C}^{m\times m}$ é tridiagonal, hermitiana e as entradas acima e abaixo da diagonal são diferentes de 0, então os autovalores de A são todos distintos

Esse ponto é muito importante. Por conta dele, podemos organizar os autovalores de $A^{(k)}$ como:

$$\lambda_1^{(k)} < \lambda_2^{(k)} < ... < \lambda_m^{(k)} \tag{136}$$

Então é possível provar que $\lambda_j^{(k+1)} < \lambda_j^{(k)} < \lambda_{j+1}^{(k+1)}$, veja a figura para ter uma noção visual:



Por conta disso eu consigo dizer quantos autovalores de uma matriz são positivos ou negativos, etc. Imagina a seguinte matriz:

$$\begin{pmatrix} 1 & 1 & & \\ 1 & 0 & 1 & \\ & 1 & 2 & 1 \\ & & 1 & -1 \end{pmatrix} \tag{137}$$

E vamos analisar a seguinte sequência (Lembre-se que $\det(A) = \prod_{i=1}^{m} \lambda_i$):
- $\det\left(A^{(1)}\right) = 1 \Rightarrow$ 0 autovalores negativos
- $\det\left(A^{(2)}\right) = -1 \Rightarrow$ 1 autovalor negativo
- $\det\left(A^{(3)}\right) = -3 \Rightarrow$ 1 autovalor negativo
- $\det\left(A^{(4)}\right) = 4 \Rightarrow$ 2 autovalores negativos

Definição 12.2.1 (Sequência de Sturm): A sequência de Sturm é definida por:

$$1, \det\left(A^{(1)}\right), \det\left(A^{(2)}\right), ..., \det\left(A^{(m)}\right) \tag{138}$$

É fácil notar que a quantidade de autovalores negativos de A está ligado a quantas vezes o sinal do determinante muda na sequência de Sturm (de 0 e + para - ou de - para 0 ou +). Mas por que isso é interessante? Eu falei e falei mas não estou vendo muito a utilidade disso.

Por conta dessas estimações, conseguimos, por exemplo, saber quantos autovalores estão dentro de um intervalo $[a, b]$. Vamos fazer a inserção de um shift xI. Vamos calcular os autovalores de $A - xI$, mas por quê? Acontece que os autovalores de $A - xI$ são $\lambda - x$, ou seja, se eu pegar todos os valores de $\lambda - x < 0 \Leftrightarrow \lambda < x$, logo, eu consigo estimar a quantidade de autovalores de A no intervalo $[-\infty, x]$

Também podemos fazer uma pequena troca e fazer um passo-a-passo mais conciso. Se vermos como A é formada, temos que:

$$\det\big(A^{(k)}\big) = a_k \det\big(A^{(k-1)}\big) - b_{k-1}^2 \det\big(A^{(k-2)}\big) \tag{139}$$

Se trocarmos $\det\big(A^{(k)}\big)$ por $\det\big(A^{(k)} - xI\big) = p^{(k)}(x)$

$$p^{(k)}(x) = (a_k - x)p^{(k-1)}(x) - b_{k-1}^2 p^{(k-2)}(x) \tag{140}$$

E se definirmos $p^{(-1)}(x) = 0$ e $p^{(0)}(x) = 1$, conseguimos, uma fórmula de recorrência para $k = 1, 2, ..., m$

12.3 Dividir para Conquistar

Esse método consiste em pegar a matriz tridiagonal e subdividi-la em matrizes menores que são mais fáceis de se trabalhar. A ideia principal é a seguinte. Temos $T \in \mathbb{R}^{m \times m}$ com $m \geq 2$ simétrica, tridiagonal e irredutível (No sentido que os valores fora da diagonal são diferentes de 0). Então podemos dividir a matriz T da seguinte forma:

$$T = \begin{bmatrix} T_1 & \beta \\ \beta & T_2 \end{bmatrix} = \begin{bmatrix} \hat{T}_1 & \\ & \hat{T}_2 \end{bmatrix} + \begin{bmatrix} \beta & \beta \\ \beta & \beta \end{bmatrix}.$$

Fazemos essa divisão em que T_1 é $n \times n$ e T_2 é $m - n \times m - n$. A diferença de T_1 para $\hat{T}_1$ é que o elemento t_{nn} foi substituido por $t_{nn} - \beta$ e a diferença de T_2 para $\hat{T}_2$ é que o elemento t_{n+1n+1} foi trocado por $t_{n+1n+1} - \beta$. Após fazer essa divisão, a gente vai subdividindo $\hat{T}_j$ da mesma forma que fizemos com T, de forma que no final vamos ter uma matriz 1×1 (A qual sabemos com certeza quem é o autovalor).

Beleza, mas como que isso vai me ajudar? É possível mostrar que, sabendo os autovalores de $\hat{T}_j$, conseguimos achar os autovalores de T. Mostrando isso, acaba que isso vira um caso de recursão, já que, ao acharmos o autovalor da matriz 1×1 (Trivial), vamos subindo até o caso $m \times m$ (T).

Vamos supor que conhecemos os autovalores de $\hat{T}_j$. Vamos então supor a seguinte diagonalização: $\hat{T} = Q_j D_j Q_j^T$. Podemos então fazer a seguinte transformação de similaridade:

$$T = \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} \left(\begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix} + \beta z z^T \right) \begin{pmatrix} Q_1^T & \\ & Q_2^T \end{pmatrix} \tag{141}$$

Onde $z^T = \begin{pmatrix} q_1^T & q_2^T \end{pmatrix}$, onde q_1^T é a última linha de Q_1 e q_2^T é a última linha de Q_2. O segundo termo do interior da matriz pode ser dificil de visualizar. O primeiro é bem intuitivo de que vai se transformar em $\begin{pmatrix} \hat{T}_1 & \\ & \hat{T}_2 \end{pmatrix}$, mas o segundo não é tão intuitivo. Vou tentar explicar melhor.

Pelo que definimos antes, podemos visualizar Q_1 e Q_2 como

$$Q_1 = \begin{pmatrix} \vdots \\ - \ q_1 \ - \end{pmatrix} \text{ e } Q_2 = \begin{pmatrix} - \ q_2 \ - \\ \vdots \end{pmatrix} \tag{142}$$

Vamos ver o que acontece com a multiplicação:

$$
\begin{gathered}
\begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} \beta \begin{pmatrix} q_1 \\ q_2 \end{pmatrix} (q_1^T \;\; q_2^T) \begin{pmatrix} Q_1^T & \\ & Q_2^T \end{pmatrix} \\
\beta \begin{pmatrix} Q_1 q_1 & \\ & Q_2 q_2 \end{pmatrix} \begin{pmatrix} q_1^T Q_1^T & \\ & q_2^T Q_2^T \end{pmatrix} \\
\beta \begin{pmatrix} 0 \\ \vdots \\ 1 \\ 1 \\ \vdots \\ 0 \end{pmatrix} (0.... \;\; 1 \;\; 1 \;\; ... \;\; 0) \\
\begin{pmatrix} 0 & & & & \\ & \ddots & & & \\ & & \beta & \beta & \\ & & \beta & \beta & \\ & & & \ddots & \\ & & & & 0 \end{pmatrix}
\end{gathered}
\tag{143}
$$

Que é justamente a matriz que tinhamos antes, então a fatoração está correta! Beleza, então eu só preciso achar os autovalores de

$$
\begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix} + \beta z z^T \tag{144}
$$

já que ela é similar a T ($\begin{pmatrix} Q_1^T & \\ & Q_2^T \end{pmatrix} \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} = I$). Mas como que eu faço isso? Em vez de trabalhar com o caso especifico de z, vamos generalizar para qualquer vetor w.

Teorema 12.3.1 (Equação Secular): Queremos achar os autovalores de $D + ww^T$ onde D é uma matriz diagonal com entradas distintas (Teorema 12.2.1), então esses autovalores são as raízes da função

$$
f(\lambda) = 1 + \sum_{j=1}^{m} \frac{w_j^2}{d_j - \lambda} \tag{145}
$$

Onde d_j são as entradas de D e w_j as entradas de w

Demonstração: Vamos supor que q é um autovetor de $D + ww^T$, então temos:

$$
\begin{gathered}
(D + ww^T)q = \lambda q \\
Dq - \lambda q + ww^T q = 0 \\
(D - \lambda I)q + ww^T q = 0 \\
q + (D - \lambda I)^{-1} ww^T q = 0 \\
w^T q + w^T (D - \lambda I)^{-1} w (w^T q) = 0 \\
(1 + w^T (D - \lambda I)^{-1} w)(w^T q) = 0
\end{gathered}
\tag{146}
$$

Se abrirmos $1 + w^{T(D-\lambda I)^{-1}} w$ na mão vamos obter $f(\lambda)$ que falei anteriormente, ou seja, a expressão total fica $f(\lambda)(w^T q) = 0$, porém, se $w^T q = 0$, então q seria autovetor de D (Só olhar a primeira equação), ou seja, $f(\lambda) = 0$ □

13 Lecture 31 - Calculando a SVD

13.1 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. O algoritmo padrão seria:

1. Calcule A^*A
2. Calcular $A^*A = V\Lambda V$
3. Defina Σ como a matriz $m \times n$ não-negativa que é a raíz de Λ
4. Resolva $U\Sigma = AV$ para uma U unitária

Só que a gente pode mostrar que esse algoritmo não é ideal é instável. Pelo Exercício 26.3 (b) do livro, temos o seguinte:

> **Teorema 13.1.1**: Suponha que A é normal. Para cada autovalor $\tilde{\lambda}_j$ de $A + \delta A$, existe um autovalor λ_j de A tal que
>
> $$|\tilde{\lambda}_j - \lambda_j| < \|\delta A\|_2 \tag{147}$$

Usando esse teorema, fazemos uma perturbação δB em A^*A, de forma que:

$$|\lambda_k(A^*A + \delta B) - \lambda_k(A^*A)| \leq \|\delta B\|_2 \tag{148}$$

Agora vamos supor um algoritmo **backward stable** que calcula os valores singulares de A. Esse algoritmo vai retornar valores $\tilde{\sigma}$ tais que:

$$\tilde{\sigma}_k = \sigma_k(A + \delta A),\ \frac{\|\delta A\|}{\|A\|} = O(\varepsilon_{\text{machine}}) \tag{149}$$

ou seja, temos que

$$|\tilde{\sigma}_k - \sigma_k| = O(\varepsilon_{\text{machine}} \cdot \|A\|) \tag{150}$$

Porém, a gente também pode supor um algoritmo **backward stable** para calcular os autovalores de A^*A, então esse algoritmo nos daria valores $\tilde{\lambda}$ tais que:

$$|\tilde{\lambda}_k - \lambda_k| = O(\varepsilon_{\text{machine}} \cdot \|A^*A\|) = O(\varepsilon_{\text{machine}} \cdot \|A\|^2) \tag{151}$$

Então a gente pode tirar a raíz desses valores computados, correto?

$$|\widetilde{\sigma_k} - \sigma_k| = O\big(|\tilde{\lambda}_k - \lambda_k|/\sqrt{\lambda_k}\big) = O(\varepsilon_{\text{machine}}\ \|A\|^2\ /\sigma_k) \tag{152}$$

E isso é pior do que antes, ou seja, mesmo que utilizemos algoritmos estáveis para calcular os autovalores de A^*A e tirar sua raíz quadrada, ainda teríamos erros maiores do que algoritmos diretos para calcular os valores singulares.

13.2 Redução para um problema de Autovalores

Por conta disso, 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

$$H = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \tag{153}$$

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

$$\begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} \cdot \begin{pmatrix} V & V \\ U & -U \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \cdot \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \tag{154}$$

Ou:

$$H = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \cdot \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \cdot \begin{pmatrix} V & V \\ U & -U \end{pmatrix}^{-1} \tag{155}$$

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

$$\kappa(H) = \|H\|_2 \cdot \left\|H^{-1}\right\|_2 = \frac{\sigma_1(H)}{\sigma_m(H)} = \frac{\sigma_1(A)}{\sigma_m(A)} = \kappa(A). \tag{156}$$

13.3 Divisão em duas fases

Porém, nós vimos algoritmos de autovalores para matrizes tridiagonais, e H não é tridiagonal, como podemos ver. Então o que fazemos? Nós dividimos o processo de achar a SVD em duas etapas, uma de tridiagonalização (Ou bidiagonalização, como veremos), e uma de diagonalização (Achar os autovalores da matriz bidiagonalizada)

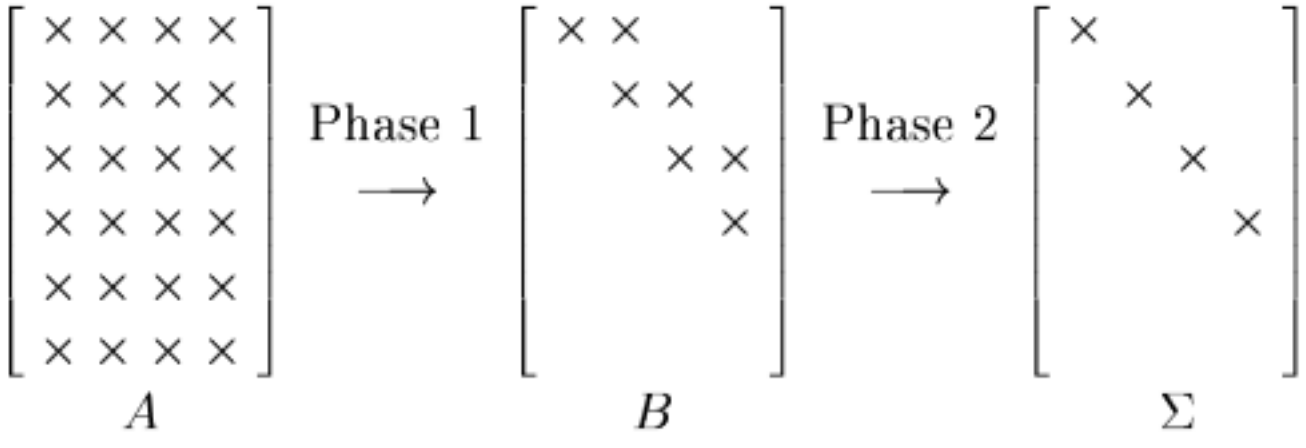


Figura 8: As fases de um algoritmo de SVD

13.4 Bidiagonalização de Galub-Kahan

A ideia é aplicar matrizes unitárias distintas na esquerda de A e na sua direita, e advinha que tipo de matrizes usamo? Exatamente: **Refletores de Householder**. A ideia é aplicar refletores a esquerda de A para colocar zeros abaixo da diagonal principal e a direita para aplicar zeros após a diagonal superior de A:

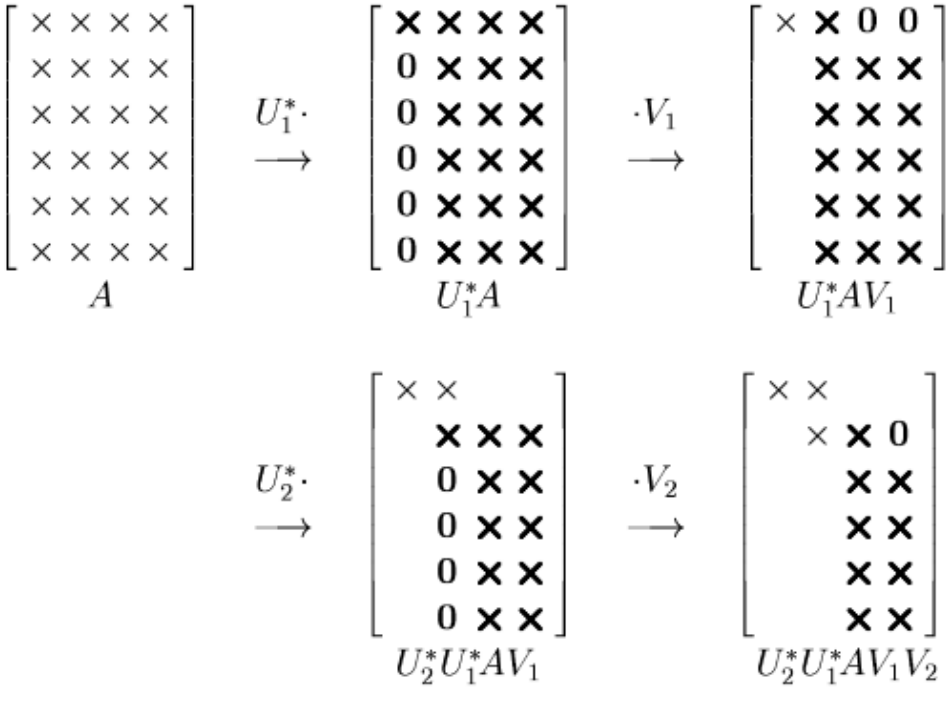


Figura 9: Bidiagonalização de Galub-Kahan exemplificada

13.5 Métodos de Bidiagonalização mais eficientes

Um método mais rápido que podemos aplicar quando $m > n$ é a *Bidiagonalização de Lawson-Hanson-Chan*, que consiste em aplicar a bidiagonalização de Galub-Kahan em R da fatoração QR de A. Pois assim reduzimos o problema para uma bidiagonalização numa matriz triangular, veja:

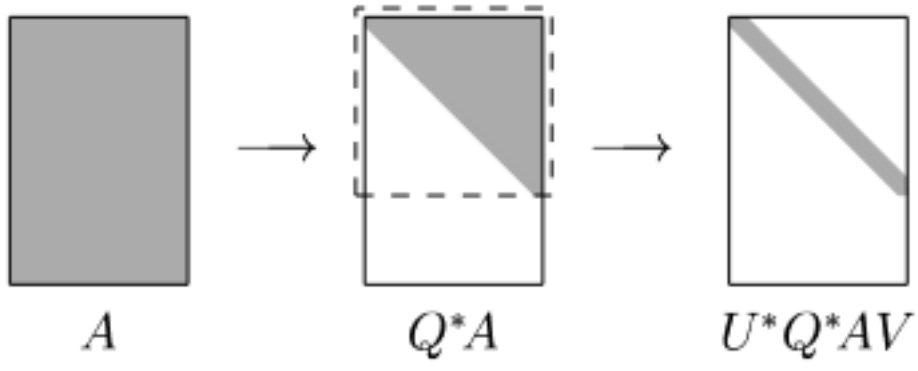


Figura 10: Bidiagonalização LHC exemplificada

Isso gera uma redução na quantidade de operações gastas para fazer o algoritmo. O problema é que, de acordo com o livro, isso só vale a pena quando $m > \frac{5}{3}n$. O interessante seria generalizar isso para o caso $m > n$. E isso é possível!

A ideia para essa generalização é não fazer a fatoração QR no inicio do algoritmo, mas em pontos adequados do algoritmo. Mas que pontos são esses? Conforme vamos fazendo a bidiagonalização, a proporção de m e n vai alterando a cada passo do algoritmo, como assim? Imagine que estamos aplicando o algoritmo numa matriz 10000×30, no segundo passo do algoritmo, perceba que vamos aplicar na matriz 9999×29, se fizermos a proporção de ambos:

$$\begin{aligned} \frac{10000}{30} &\approx 333,33 \\ \frac{9999}{29} &\approx 344,79 \\ \frac{9998}{28} &\approx 357,07 \end{aligned} \tag{157}$$

Perceba que a proporção só aumenta pois eu estou sempre aplicando em matrizes com m muito grande. O livro fala que o que fazemos é aplicar a fatoração QR no k-ésimo passo quando:

$$\frac{m-k}{n-k} = 2 \tag{158}$$

Veja a ilustração do processo:

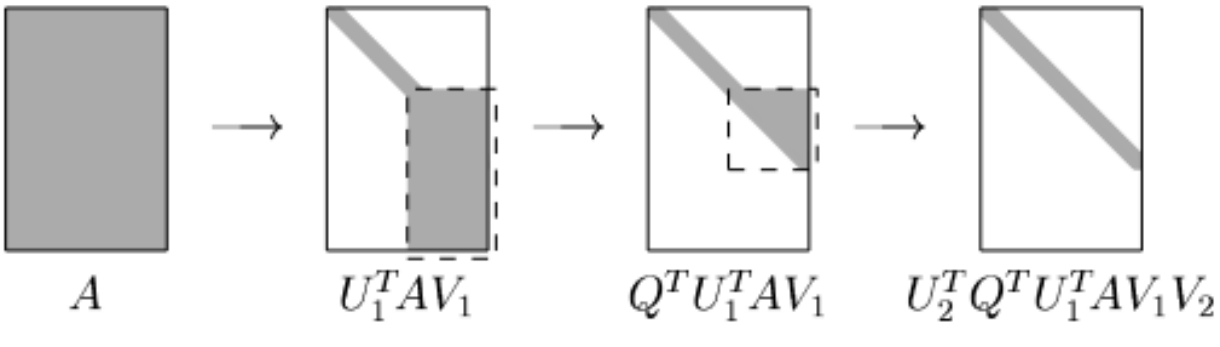


Figura 11: Aplicação da QR em pontos-chave da iteração

13.6 Fase 2

A fase 2 é aplicar algum algoritmo de autovalores na matriz que encontramos. Os dois principais algoritmos que são utilizados é uma versão modificada do algoritmo QR e o dividir e conquistar