Algebra Linear Numérica A2 Recap

15/05/2025

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

1. Lecture 16 - Estabilidade da Triangularização de Householder

Nesse capítulo, a gente tem uma visão mais aprofundada da análise de **erro retroativo** (Backwards Stable). Dando uma breve recapitulada, para mostrar que um algoritmo $\tilde{f} : X \to Y$ 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_2R_2 é muito mais preciso do que Q_3R_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:

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

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:

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

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:

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

Tal que:

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

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

1.3. Algoritmo para resolver $Ax = b$

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

Algoritmo para resolver $Ax = b$

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

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

Teorema 1.3.1: O algoritmo descrito anterioremente para solucionar $Ax = b$ é **backwards stable**, satisfazendo

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

com

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

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:

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

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:

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

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:

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

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

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

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

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

Perceba que

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

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:

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

O que nos indica que

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

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

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

- $(\delta Q)(\delta R)$: Por último:

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

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:

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

□