MAP3121 - Métodos Numéricos e Aplicações

Escola Politécnica da Universidade de São Paulo

Exercício Programa 2

Autovalores e Autovetores de Matrizes Reais Simétricas

Gabriel Macias de Oliveira, NUSP 11260811, Eng. Elétrica

Rodrigo Ryuji Ikegami, NUSP 10297265, Eng. Elétrica

São Paulo,

2021.

Sumário

1 Introdução

1.1 Ferramentas Utilizadas

Foram utilizadas as seguintes ferramentas para construção do código:

- Linguagem de Programação: *Python 3.7.9+*
- Bibliotecas Externas:
  - `numpy`, para trabalhar com aritmética de vetores
  - `matplotlib`, para produção de gráficos e animações
- *IDE*: *Visual Studio Code*
- Desenvolvimento Paralelo: *Git*

Além das bibliotecas externas, utilizaram-se as bibliotecas nativas `math`, para funções matemáticas básicas, `typing`, para utilizar tipos estáticos em *Python*, e `sys` para personalização da CLI.

Todos os testes em que são envolvidas métricas de tempo / número de iterações foram executados com base em um AMD Ryzen 5 3600X @ 4.2 GHz, portanto sendo suscetíveis a variações.

Todo o código está concentrado no arquivo `main.py`, cujos detalhes de execução se encontram em sequência e no arquivo `LEIA-ME.txt`.

Este relatório foi tipografado em LaTeX.

1.2 Execução dos ***Scripts***

Estando o *Python* atualizado para uma versão compatível, isto é, 3.7.9 ou mais recente, deve-se certificar que ambas bibliotecas `numpy` e `matplotlib` estejam instaladas. Caso contrário, basta executar `pip install -r requirements.txt` em um terminal, para recebê-las.

O arquivo principal deve ser executado no mesmo diretório em que foi descompactado, utilizando o comando `python main.py`. A exibição do terminal deve ser da CLI que acompanha o programa, conforme a Figura **??**.

2 Implementação

2.1 As Transformações de Householder

Conforme [1], as *Transformações de Householder* são transformações lineares ortogonais $H_w : \mathbb{R}^n \to \mathbb{R}^n$ da forma $H_w = I - \frac{2ww^T}{w \cdot w}$ que operam sobre o espaço de vetores como uma reflexão em relação ao espaço $w^\perp$. Dado um vetor de interesse x, se $y = H_w x$, então:

$$y = x - 2\frac{w \cdot x}{w \cdot w} w\,.$$

Dados dois vetores x e y, não nulos em $\mathbb{R}^n$, é possível definir uma transformação de Householder tal que $H_w x = \lambda y$, com $\lambda \in \mathbb{R}$. Para tanto basta se definir $w = x \pm \frac{||x||}{||y||} y$. (1)

Esta propriedade se torna extremamente útil para a tridiagonalização de matrizes reais simétricas. Para cada coluna i de uma matriz dada A, podemos definir uma transformação de Householder com:

$$w_i = \tilde{a}_i + \delta \frac{||\tilde{a}_i||}{||e_{i+1}||} e_{i+1} = \tilde{a}_i + \delta ||\tilde{a}_i|| e_{i+1}\,.$$

sendo $\delta = \pm 1$, e_{i+1} o i+1-ésimo versor da base canônica de $\mathbb{R}^n$ e $\tilde{a}_i$ composta pelos elementos da coluna i de A, exceto os pertencentes à diagonal principal e acima dela. Isto é,

$$\tilde{a}_i = (0, 0, ..., 0, A_{i+1,i}, A_{i+2,i}, ..., A_{n-1,i}, A_{n,i})^T\,.$$

De acordo com o proposto em [1], utilizamos δ com sinal igual ao de $A_{i+1,i}$ para cada w_i.

Utilizando a propriedade em (1), podemos provar que

$$H_{w_i} \tilde{a}_i = \tilde{a}_i - \delta w_i H_{w_i} \tilde{a}_i = (0, 0, ..., 0, -\delta ||\tilde{a}_i||, 0, ..., 0)^T\,.$$

Ou seja, a coluna i, após a transformação H_{w_i}, possui como único elemento não nulo o módulo de $\tilde{a}_i$ na posição i, com sinal oposto ao de $A_{i+1,i}$.

Assim, temos que

$$H_{w_1} A = \begin{bmatrix} x & x & x & ... & x \\ x & x & x & ... & x \\ 0 & x & x & ... & x \\ ... & ... & ... & ... & ... \\ 0 & x & x & ... & x \end{bmatrix}$$

onde os x representam valores quaisquer.

E, como H_{w_1} e A são simétricas, podemos fazer

$$H_{w_1} A H_{w_1} = \begin{bmatrix} x & x & 0 & ... & 0 \\ x & x & x & ... & x \\ 0 & x & x & ... & x \\ ... & ... & ... & ... & ... \\ 0 & x & x & ... & x \end{bmatrix}$$

para zerar os elementos à direita da sobrediagonal na primeira coluna.

A matriz resultante, pelo mesmo motivo, também é simétrica. Assim, podemos aplicar sucessivas transformações de Householder à direita e à esquerda de A de forma a obter uma matriz semelhante a A, T, tridiagonal.

Vale ressaltar que, para cada H_{w_i} multiplicado à esquerda de A, apenas as linhas $i+1$ a n têm seus elementos modificados. Simetricamente, quando se multiplica H_{w_i} à direita de A, apenas as colunas $i+1$ a n têm seus elementos modificados. (2)

Ao fim das transformações, obteremos a expressão $T = HAH^T$, com:

$$H^T = H_{w_1} H_{w_2} ... H_{w_{n-1}} H_{w_n} .$$

Para economizar tempo e memória, definimos $\bar{w}_i$ e $\bar{a}_i$, que são equivalentes a w_i e $\tilde{a}_i$, respectivamente, mas sem os i primeiros valores, pois são todos zero.

2.1.1 Função de Implementação da Tridiagonalização de uma Matriz Real Simétrica

Feitas as considerações acima, criamos a função `tridiagonalization`, que recebe uma matriz simétrica `A` como entrada e devolve a matriz T tridiagonal, representada por dois vetores, `alphas` e `betas`, que representam sua diagonal principal e sua sobrediagonal, respectivamente, e a matriz `Ht`, que representa H^T, descrita anteriormente.

A implementação feita se utilizada das propriedades descritas em (1) e (2) para aumentar a eficiência do código. Em cada iteração, podemos trabalhar sobre uma submatriz de A, sobre a qual faremos as contas, já que valores fora dela não são modificados, exceto a coluna/linha cuja maioria dos valores serão zerados. Os único valor que não são zero pertencem à diagonal principal e sua sobre/subdiagonal. Para os valores da diagonal, basta tomar $A_{i,i}$ na iteração i, pois seu valor não é afetado por nenhuma das transformações de Householder subsequentes. Para os valores da sobrediagonal, basta tomar $-\delta||\tilde{a}_i||$, como demonstrado anteriormente.

A implementação está no Código 2.1.1, abaixo, do qual se retiraram os comentários, mantidos no arquivo original do *script*.

```
def tridiagonalization(A: np.array) → Tuple[np.array, np.array, np.array]:
```

```
2       A = A.copy()
3       alphas = []
4       betas = []
5
6       H = np.identity(np.size(A, 0))
7
8       for m in reversed(range(2, np.size(A, 0))):
9           w_i = A[1:, 0]
10
11          alphas.append(A[0, 0])
12          betas.append(-sgn(w_i[0]) * np.sqrt(np.dot(w_i, w_i)))
13
14          w_i[0] -= betas[-1]
15          w_i2 = np.dot(w_i, w_i)
16
17          A = A[1:, 1:]
18
19          for Acol, Arow, Hrow in zip(np.transpose(A), A, H[:, -m:]):
20              Acol -= 2 * np.dot(w_i, Acol) / w_i2 * w_i
21              Arow -= 2 * np.dot(w_i, Arow) / w_i2 * w_i
22              Hrow -= 2 * np.dot(w_i, Hrow) / w_i2 * w_i
23
24      alphas.extend(np.diag(A))
25      betas.append(A[1, 0])
26
27      return (np.array(alphas), np.array(betas), H)
```

Código 2.1.1: Função que implementa a tridiagonalização de uma matriz dada, `A`.

O código segue a descrição formal apresentada anteriormente. A linha 9 define o vetor $\bar{w}_i$ da Tranformação de Householder, $H_{\bar{w}_i}$, de uma dada iteração `i` e o inicializa com $\bar{a}_i$. Sa linhas 11 e 12 adicionam os elementos calculados da diagonal principal e da sua sobrediagonal aos vetores `alphas` e `betas`, respectivamente. A linha 14 modifica o w_i de acordo com a expressão $\bar{w}_i = \bar{a}_i + ||\bar{a}_i||\delta e_1$. A linha 15 define uma variável auxiliar `w_i2`, equivalente a $\bar{w}_i \cdot \bar{w}_i$. A linha 17 atualiza a variável `A` para armazenar a submatriz de uma dada iteração. As linhas 19 a 22 executam as multiplicações $H_{\bar{w}_i}\bar{A}H_{\bar{w}_i}$ e $H^T H_{\bar{w}_i}$. As linhas 24 e 25 adicionam os últimos elementos da diagonal principal e da sobrediagonal da matriz resultante em `alphas` e `betas`, respectivamente.

2.2 O Algoritmo QR

Após se obter a matriz T, tridiagonal, a partir das transformações de Householder, podemos obter seus autovetores e autovalores utilizando o *Algoritmo QR*. Apesar de A e T serem matrizes semelhantes, isto é, possuem mesmos autovalores, seus autovetores são distintos. Ou seja, para se obter os autovalores de A, precisamos que $V^{(0)}$ seja equivalente a H^T, pois, utilizando-se a matriz identidade como $V^{0)}$, obteríamos os autovetores de T.

Para a implementação do Algoritmo QR, foi utilizada a mesma função, `qr_algorithm`, do EP anterior. A única modificação feita sobre ela foi a adição de um parâmetro de entrada, `V0`, que é utilizado ao invés da identidade para o cálculo dos autovetores.

3 Construção dos Testes

Os aspectos matemáticos da construção de cada teste serão apresentados na Seção **??** junto aos resultados. Nesta seção, deseja-se detalhar a implementação *em código* de tais testes.

3.1 Função Principal

A função principal do programa segue uma interface simples, com opções numeradas que o usuário pode selecionar.

Referências

[1] MAP3121. **EP2: Autovalores e Autovetores de Matrizes Reais Simétricas**. Disponível em: <https://edisciplinas.usp.br/pluginfile.php/6318233/mod_resource/content/2/ep2_2021.pdf>. Acesso em: 30 jun. 2021.