Universidade Federal de Minas Gerais
Instituto de Ciências Exatas – ICEx
Departamento de Matemática

Álgebra Linear Numérica
Trabalho Prático – Autovalores e SVD

Nesse trabalho, você implementará os algoritmos vistos na segunda parte do curso.

0.1 Dicas

Algumas recomendações podem facilitar o processo de implementação e *debugging*:

- Faça funções razoavelmente pequenas que fazem apenas uma tarefa.
- Não escreva muito código de uma vez sem parar para testar o que foi escrito. Se suas funções forem pequenas, será fácil verificar que elas fazem o trabalho corretamente antes de passar para escrever a próxima função. Escreva pequenos casos de teste para te ajudar! Julia tem excelente suporte a testes unitários (e os testes servirão de documentação).
- Se for difícil quebrar um pedaço de código razoavelmente grande em funções menores, sua vida ficará mais complicada. Em todo caso, ainda há salvação: Demarque partes da sua função grande e use comentários para explicar e delimitar cada parte.
- Use nomes de funções e de variáveis descritivos. Isso pode significar nomes mais longos, mas também pode significar usar as convenções matemáticas do livro-texto. Por exemplo, no contexto de refletores, τ e γ são nomes auto-descritivos, e Julia suporta Unicode (favor não exagerar nos emojis!).

0.2 Disclaimer

Em princípio, esse trabalho usa técnicas numericamente estáveis, como vimos em sala. Implementar nossos próprios algoritmos é útil didaticamente, mas ignora anos de pesquisa; além disso, é fácil errar a implementação de alguma parte do algoritmo e perder estabilidade numérica. No futuro, se precisar achar a decomposição SVD (ou os autovalores de uma matriz) e puder usar o algoritmo da biblioteca padrão da sua linguagem, prefira!

Esse não é um conselho condescendente: Muito provavelmente a biblioteca padrão da sua linguagem seguiu o mesmo conselho e está chamando alguma

rotina amplamente testada, como as rotinas do pacote LAPACK (que existe desde 1992 e utiliza primitivas básicas especificadas pelo padrão BLAS, que existe desde 1979). Esse é o caso da biblioteca padrão do Julia e do SciPy no Python, entre outros.

1 Primitivas

1.1 Norma de vetor

A norma euclideana de um vetor $v = (v_1, \ldots, v_n) \in \mathbb{R}^n$ é dada por

$$\|v\|_2 = \sqrt{v_1^2 + \cdots + v_n^2} \tag{1}$$

Infelizmente, vimos na Lista 1 que a fórmula acima não é um bom jeito de calcular numericamente a norma de um vetor: Elevar as componentes ao quadrado pode zerá-las, mesmo no caso em que elas são significativas para a norma.

Seja $\beta = \max_{1 \leqslant i \leqslant n} |v_i|$ e $w = (1/\beta) \cdot v$, de modo que o vetor w tem pelo menos uma coordenada de norma 1. Usando esse fato, é possível mostrar que o erro relativo no cálculo de $\|w\|$ usando a equação (1) é pequeno. Computamos a norma do vetor original v computando a norma do vetor w e posteriormente multiplicando a resposta por β.

Questão 1. Implemente uma função de calcular norma de um vetor no $\mathbb{R}^n$ usando o truque acima. Ache um vetor cuja norma é mal-aproximada pelo uso ingênuo da fórmula (1), e teste sua rotina nesse caso.

1.2 Rotações de Givens – caso real

Dados $f, g \in \mathbb{R}$, é possível computar números $c, s, r \in \mathbb{R}$ tais que

$$R(c,s) := \begin{pmatrix} c & s \\ -s & c \end{pmatrix} \text{ é ortogonal} \qquad \text{e} \qquad R(c,s) \cdot \begin{pmatrix} f \\ g \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$$

para algum $r \in \mathbb{R}$. Caso a igualdade acima valha, a matriz $R(c,s)$ é chamada de uma *rotação de Givens*.

Como transformações ortogonais preservam norma, temos que $|r| = \|(f,g)\|$. Escolheremos o sinal de modo que r seja não-negativo.

Questão 2. Faça uma função que recebe dois números reais f e g e calcula

c, s e r com as propriedades acima. Use sua rotina de computar norma, mas não use funções trigonométricas na resposta final.

Observação: No caso complexo, que é fora do escopo dessa lista, existem muitas escolhas para c, s e r. Existe uma escolha padrão de rotação, motivada por considerações de estabilidade e continuidade, explicada em detalhes no artigo [3].

Essa primitiva será usada na Parte 3, onde você implementará o algoritmo de Golub–Reinsch para decomposição SVD.

1.3 Refletores de Householder – caso real

Vimos em sala que um refletor em $\mathbb{R}^n$ é uma transformação ortogonal da forma $Q = I - \gamma v v^T$, para $v \in \mathbb{R}^n$ e $\gamma = 2/\|v\|^2$. Dado um vetor $w \in \mathbb{R}^n$, podemos computar v e $gamma$ tais que

$$Qw = \begin{pmatrix} \pm\tau \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

onde $\tau = \|w\|$. Lembre-se que vimos como calcular tal transformação em sala, e que o sinal de τ é escolhido de modo a evitar cancelamento nas contas.

Questão 3. Implemente uma função que recebe um vetor w e computa γ e v com as propriedades acima.

Note que refletores satisfazem $Q = Q^T$, o que será útil no futuro.

1.4 Decomposição QR

A essa altura, você já tem tudo que é necessário para implementar sua decomposição QR. Além de ser uma rotina interessante por si só, iremos usar tal rotina para verificar a corretude do nosso algoritmo de autovetores.

Questão 4. Implemente uma função que calcule a decomposição QR de uma matriz A:

	Nome	Descrição
Entrada	A	Matriz $n \times m$
Saída	Q	Matriz $n \times n$ ortogonal
	R	Matriz $n \times m$ triangular superior
Propriedades		$A = QR$
Complexidade		$O(nm^2)$

Comece com $Q = I$. Para cada coluna, você irá:

- Computar um refletor Q_i tal que $Q_i A$ tenha a i-ésima coluna zerada abaixo da diagonal.
- Fazer $A \leftarrow Q_i A$ e $Q \leftarrow QQ_i$. **Você deve fazer isso sem montar a matriz Q_i do refletor para que o algoritmo tenha a complexidade correta**. Na Lista 2, você mostrou que, mesmo sem montar a matriz Q_i, algumas das computações que estamos fazendo no cálculo de Q são redundantes, mas vamos ignorar isso por simplicidade.

Dica: Estar confortável com aplicar refletores à esquerda e à direita "na parte certa da matriz" é importante para os algoritmos que seguem. Teste sua decomposição QR com cuidado antes de prosseguir.

2 Autovalores e autovetores

Nessa seção, iremos implementar um procedimento que recebe uma matriz A real $n \times n$ e calcula uma matriz D *diagonal em blocos* tal que A e D são aproximadamente similares. Os blocos da diagonal de D serão no máximo de tamanho 2×2; a existência de tais blocos em D corresponde a autovalores complexos conjugados da matriz real A.

No caso em que A é simétrica, D será diagonal porque os autovalores de A são todos reais. Nesse caso, também iremos computar explicitamente uma matriz unitária Q tal que $A \approx QDQ^*$; essa equação diz que a i-ésima coluna de Q é autovetor de A correspondente ao autovalor d_{ii}. É possível calcular a matriz Q mesmo no caso não-simétrico, mas não iremos fazer isso (ver seção 5.7 de [11]).

2.1 Redução à forma de Hessenberg

Questão 5. Implemente uma função que reduza uma matriz A à forma de Hessenberg:

	Nome	Descrição
Entrada	A	Matriz geral
Saída	H	Matriz em forma de Hessenberg
	Q	Matriz unitária (produto de refletores)
Propriedades		$A = QHQ^*$
Complexidade		$O(n^3)$

Dizemos que uma matriz está em forma de Hessenberg se $a_{i,j} = 0$ sempre que $i > j + 1$. Em outras palavras, a matriz é quase uma matriz triangular superior, exceto pela possível existência da primeira subdiagonal não-nula.

Como vimos em sala, iremos aplicar $n - 2$ refletores para transformar A numa matriz Hessenberg. O objetivo do i-ésimo refletor, Q_i, é limpar as entradas $a_{i,i+2}$ até $a_{i,n}$, mas lembre que tal refletor será aplicado à esquerda e à direita.

- Não monte as matrizes Q_i explicitamente, mas lembre-se de ir atualizando a matriz de saída Q, de modo similar ao feito na decomposição QR.
- Teste se sua saída satisfaz as propriedades desejadas! H tem que ser Hessenberg, Q tem que ser unitária e $A = QHQ^*$. Todas essas coisas podem ser verificadas com código.

Curiosidade extra: Estamos retornando uma matriz Q para representar o resultado de aplicar os n refletores. No entanto, cada Q_i opera num espaço de dimensão $n - i$; nesse espaço, $Q_i = I - \tau_i v_i v_i^T$, e portanto só precisamos guardar τ_i e outros $n - i - 1$ números para representar v_i (a primeira coordenada de v_i é τ_i). Numa matriz Hessenberg H, existem exatamente $n - i - 1$ zeros abaixo da subdiagonal na i-ésima coluna, e portanto, a menos dos τ_i, uma implementação compacta pode retornar tanto os v_i quanto a matriz de saída B sobrescrevendo a matriz de entrada A.

2.2 Iteração de Francis de grau 2

Nosso objetivo nas próximas seções é resolver a seguinte questão

Questão 6. Implemente a iteração de Francis:

	Nome	Descrição
Entrada	H	Matriz $n \times n$ Hessenberg própria.
	Q	Matriz $n \times n$ tal que $A = QHQ^*$.
Saída	$\hat{H}$	Matriz $n \times n$ Hessenberg com $\hat{H}e_1 = \alpha p(H)e_1$.
		Sobrescreve a matriz Q de modo que $A = Q\hat{H}Q^*$.
Complexidade		$O(n^2)$ no caso geral.
		$O(n)$ no caso simétrico se não quisermos alterar Q.

No nosso caso, iremos fazer uma iteração de Francis de grau 2. Isso significa que $p(H) = (H - \rho_1 I)(H - \rho_2 I)$, onde ρ_1 e ρ_2 são dois shifts a escolher.

A vantagem dos shifts de grau 2 é que, se ρ_1 e ρ_2 forem complexos conjugados, todas as contas feitas pelo algoritmo envolverão apenas números reais. Para implementar o algoritmo, siga as instruções das próximas subseções.

2.2.1 Autovalores de uma matriz 2×2

Nessa seção, iremos investigar como calcular os autovalores de uma matriz

$$M = \begin{pmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{pmatrix}.$$

A primeira vista, isso é fácil: Bastaria resolver a equação do segundo grau $x^2 - (m_{11} + m_{22})x + m_{11}m_{22} - m_{12}m_{21}$. Veja o Apêndice A para entender porque o jeito tradicional de fazer isso pode não ser uma boa ideia.

O jeito utilizado pela biblioteca LAPACK é o seguinte:

- Primeiro, divida tudo por $s = |m_{11}| + |m_{12}| + |m_{21}| + |m_{22}|$, por motivo similar ao do truque da Seção 1.1. Lembre-se de multiplicar os autovalores por isso ao final!
- Defina $t = (m_{11} + m_{22})/2$. Note que a matriz $M - tI$ tem traço zero (autovalores da forma $\pm x$) e seu determinante é $d = -x^2 = (m_{11} - t)(m_{22} - t) - m_{12}m_{21}$.
- Se $d > 0$, então $x \notin \mathbb{R}$. Assim, os dois autovalores de $M - tI$ são $\pm i\sqrt{d}$, e portanto os autovalores de M são $t \pm i\sqrt{d}$. Se $d \leqslant 0$, os autovalores reais de M são $t \pm \sqrt{|d|}$.

É um exercício interessante entender porque esse algoritmo é estável. O Exercício 5.6.31 de Watkins dá outro modo (mais complicado de implementar mas mais evidentemente correto) de calcular autovalores 2×2, usando rotações.

2.2.2 Shifts de Rayleigh e de Wilkinson de grau 1

Como vimos em sala, o quociente de Rayleigh correspondente ao vetor e_n é a entrada $h_{n,n}$ da matriz H, o que dá o shift de Rayleigh de dimensão 1. O shift de Wilkinson é o autovalor da matriz

$$H_{\text{canto}} := \begin{pmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{n,n} \end{pmatrix}$$

mais próximo de $h_{n,n}$. Wilkinson [12] mostrou que o algoritmo de Francis com shift de Rayleigh tem convergência local cúbica, e que os chamados shifts de Wilkinson garantem convergência global quadrática (no pior caso) no caso simétrico. O shift de Rayleigh nem sempre converge globalmente, como mostra o exemplo

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

que vimos em sala, mas no caso simétrico ele tem uma propriedade interessante, provada por Kahan [8]: Dado um vetor inicial v_0, ou o método converge para um autovalor, ou converge para um ponto equidistante de dois autovalores. Além disso, quando o segundo caso ocorre, então quase qualquer perturbação de v_0 faz o algoritmo convergir.

Na prática, isso significa que pequenos erros de precisão ajudam a convergência do algoritmo! Se tal perturbação não for introduzida por erros de precisão e o algoritmo não estiver convergindo, shifts "excepcionais" (que podem ser aleatórios ou calculados de outro modo) são usados para causar a perturbação.

2.2.3 Shift de Rayleigh generalizado (o que usaremos!)

Se quisermos escolher dois shifts, como é o nosso caso, o análogo do shift de Rayleigh em dimensão 2 são *os dois* autovalores da matriz H_{canto}. Isso pode parecer com o shift de Wilkinson, mas lembre que lá estávamos escolhendo um shift, e aqui estamos escolhendo dois shifts complexos conjugados. Esse análogo faz com que autovalores complexos conjugados sejam extraídos ao mesmo tempo. Valem as mesmas propriedades de convergência local.

Note que nossa motivação de escolher os shifts desse modo é evitar usar números complexos no computador. No entanto, se os dois autovalores da matriz H_{canto} forem números reais, podemos usar o shift de Wilkinson duas vezes (isto é, pegar o autovalor mais próximo de $h_{n,n}$ e usá-lo duas vezes, ignorando o outro autovalor). Isso corresponde a fazer duas iterações do shift de Wilkinson de grau 1.

Questão 7. Implemente uma função que calcula os shifts que usaremos: O shift de Rayleigh generalizado (ρ_1 e ρ_2 serão os dois autovalores da matriz H_{canto}) se eles não forem reais, ou duas cópias do shift de Wilkinson (ou seja, $\rho_1 = \rho_2$ = autovalor da matriz H_{canto} mais próximo de $h_{n,n}$) se os autovalores forem reais.

Curiosidade extra: Existe shift de Wilkinson de dimensão 2, definido como o par de autovalores da submatriz 4×4 inferior direita de H mais próximo do par de Rayleigh. Tal estratégia não é utilizada na biblioteca LAPACK.

2.2.4 Aplicando Q_0 tal que $Q_0^*(p(H)e_1) = \beta e_1$

O primeiro passo da iteração de Francis é computar uma transformação tal que $Q_0^*(p(H)e_1) = \beta e_1$. Isso, juntamente com retornar à forma de Hessenberg, corresponderá ao procedimento de iterar subespaços.

Questão 8. Encontre manualmente uma fórmula explícita para o vetor $(H - \rho_1 I)(H - \rho_2 I)e_1$ usando que H é Hessenberg. Use sua função da seção 1.3 para encontrar um refletor Q_0 pequeno (que opera em poucas linhas/colunas) com a propriedade desejada.

Multiplicando por Q_0 à esquerda, combinamos as linhas 1 a 3 da matriz. Fazendo o mesmo à direita, combinamos as colunas 1 a 3, o que quebra a forma de Hessenberg.

2.2.5 Voltando à forma de Hessenberg

Agora, as entradas 3 e 4 da primeira coluna da matriz podem ser não-nulas. Iremos consertar isso: Combinando as linhas 2 a 4 com um refletor pequeno Q_1 conseguimos limpar a primeira coluna. No entanto, ao multiplicar pelo mesmo refletor à direita, combinamos as colunas, e o resultado final é que a segunda coluna agora pode ter as entradas 4 e 5 não nulas (verifique você também!). Repetir o procedimento $n-2$ vezes, a matriz volta a ser Hessenberg.

Note que é necessário cuidado para aplicar o refletor eficientemente: Sua implementação deve aplicar o refletor numa coluna/linha de uma matriz em tempo constante.

Questão Extra. Se A é simétrica, H é tridiagonal, pois uma matriz similar a uma simétrica é também simétrica. Se não quisermos calcular $\hat{Q}$, implemente uma otimização que faça uma iteração de Francis (no caso simétrico) em tempo $O(n)$ e verifique que sua função se comporta igual

ao caso geral.

Dica: Aplicar os refletores do algoritmo numa matriz tridiagonal demora tempo $O(1)$, pois, para um dado refletor, quase todas as linhas/colunas conterão zeros nas posições relevantes.

Com isso, computamos refletores $Q_0, \ldots, Q_{n-2}$, e computamos a matriz $\hat{H} = Q_{n-2} \cdots Q_0 H Q_0 \cdots Q_{n-2}$. Para terminar com a propriedade de que $A = Q\hat{H}Q^*$, precisamos aplicar (do lado certo!) as mesmas transformações em Q.

2.3 Juntando tudo no caso simétrico

Nessa seção, descreveremos como realizar várias iterações de modo eficiente. Se o algoritmo convergir, o resultado final desse procedimento será obter, a partir da matriz A simétrica original, matrizes H diagonal em blocos (1×1 ou 2×2) e Q unitária tais que $A \approx QHQ^*$.

Seja n a dimensão da matriz A. A ideia do procedimento é mantermos um índice r tal que submatriz dada pelas linhas/colunas de $r+1$ a n já estejam diagonalizadas; inicialmente, definimos $r = n$ para indicar que nenhum pedaço da matriz foi diagonalizado. A medida que o algoritmo progredir, iremos descobrir blocos 1×1 e 2×2 e com isso diminuir r em 1 ou 2.

Há outra preocupação a se ter em mente. Como vimos, é essencial trabalhar com matrizes *propriamente* Hessenberg para que a iteração de Francis tenha as propriedades desejadas de iteração de subespaços. Para isso, manteremos um índice ℓ, que indicará a "última entrada desprezível da subdiagonal inferior" do pedaço não-diagonalizado da matriz. Formalmente, ℓ é o maior inteiro entre 1 e r tal que

$$|a_{\ell,\ell-1}| \leqslant u \cdot \left(|a_{\ell,\ell}| + |a_{\ell-1,\ell-1}| \right), \tag{2}$$

onde entradas fora da matriz valem 0 por convenção e u é o erro da representação em ponto flutuante. Na prática, podemos verificar se

$$\mathrm{fl}\left(\left(|a_{\ell,\ell}| + |a_{\ell-1,\ell-1}| \right) + |a_{\ell,\ell-1}| \right) = \mathrm{fl}\left(|a_{\ell,\ell}| + |a_{\ell-1,\ell-1}| \right),$$

ou seja, se somar $|a_{\ell,\ell-1}|$ a $|a_{\ell,\ell}| + |a_{\ell-1,\ell-1}|$ em ponto flutuante é o mesmo que não somar nada.

Calculando ℓ desse modo, garantimos a submatriz dada pelas linhas/colunas entre ℓ e r é propriamente Hessenberg mesmo se desprezarmos elementos insignificantes da subdiagonal[1]. Nesse caso, zeramos $a_{\ell,\ell-1}$ e temos uma decisão a tomar:

[1] Tal critério foi popularizado por Wilkinson, mas pode ser melhorado. Veja [1] para o critério usado na biblioteca LAPACK.

- Caso 1: Se ℓ for $r-1$ ou $r-2$, achamos um bloco 1×1 ou 2×2. Nesse caso, atualizamos r para refletir o novo bloco encontrado, e redefinimos ℓ para o menor valor possível, de modo que o próximo passo foque em toda a matriz restante.
- Caso 2: Se ℓ não for nem $r-1$ nem $r-2$, podemos executar a iteração de Francis na submatriz que começa na linha/coluna ℓ e termina na linha/coluna r (isso também se aplica às operações que faremos na matriz Q) e ter certeza de que estamos trabalhando com uma matriz Hessenberg própria. Depois da iteração, atualizamos ℓ (que pode aumentar ou permanecer o mesmo, mas não diminuir, visto que não alteramos $a_{\ell,\ell-1}$).

Repetimos o procedimento até processarmos toda a matriz (ou seja, até que r seja 0). Se isso ocorrer, dizemos que o algoritmo *convergiu*. Vimos que, com shifts de Rayleigh, tal procedimento convergirá quase sempre. Para evitar loops infinitos, precisamos de um critério para "declarar derrota" em caso de não-convergência. Usaremos o critério da biblioteca LAPACK: Para uma matriz $n\times n$, consideraremos que o algoritmo não convergiu se não terminarmos o procedimento em até $30n$ iterações. Em usos práticos do e algoritmo, [9] reporta que, em média, 1,7 iterações são necessárias para extrair um autovalor.

Questão 9. Implemente o procedimento acima para fazer várias iterações de Francis. Faça seu código imprimir uma mensagem dizendo qual dos dois casos (Caso 1 ou Caso 2) reduziu o problema. Qual caso acontece mais frequentemente?

Curiosidade: Às vezes, um elemento pode não ser insignificante por si só, mas mesmo assim possibilitar a quebra em subproblemas. Isso ocorre quando dois elementos consecutivos da subdiagonal inferior têm produto muito pequeno (ver [13] para mais detalhes de como isso é feito na prática). Não nos preocuparemos com essa otimização aqui.

2.4 Por que funciona?

Lembre-se do que está acontecendo teoricamente: As operações unitárias computadas numa iteração de Francis, quando combinadas, dão uma matriz unitária Q tal que $\hat{H} = Q^*HQ$ Denotaremos por $\{q_1,\dots,q_n\}$ as colunas dessa matriz.

Vimos que $q_1 = Qe_1 = \alpha p(H)e_1$, por construção de Q_0 e porque as demais transformações não alteram tal propriedade. Se $\hat{H}$ não for Hessenberg própria, caímos em dois problemas menores, como vimos. Do contrário, vimos que,

para $k = 1, \dots, n$,

$$\begin{aligned}
\operatorname{span}\{q_1, \dots, q_k\} &= \operatorname{span}\{q_1, Hq_1, \dots, H^{k-1}q_1\} \\
&= \operatorname{span}\{p(H)e_1, Hp(H)e_1, \dots, H^{k-1}p(H)e_1\} \\
&= p(H)\operatorname{span}\{e_1, He_1, \dots, H^{k-1}e_1\} \\
&= p(H)\operatorname{span}\{e_1, e_2, \dots, e_k\},
\end{aligned}$$

onde a primeira igualdade usa que $\hat{H}$ é Hessenberg própria (ver Teorema 6.3.1 de Watkins) e a última usa que H é Hessenberg própria.

Moral da história: Fazer a mudança de H para $\hat{H}$ corresponde a aplicar $p(H)$ nos subespaço gerado pelos k primeiros vetores da base canônica (para qualquer k), e a matriz Q é a mudança de base correspondente. Fazendo isso várias vezes, estamos iterando subespaços. Vimos em sala que, se houver um *gap* entre os k maiores autovalores e os demais, uma generalização do método de potência diz que os k primeiros vetores irão convergir para o espaço gerado pelos k "maiores" autovetores. Isso separará um bloco $k \times k$ do resto da matriz. Isso é o que ocorre no Caso 2 da seção 2.3.

Além disso, vimos em sala que, por iteração de subespaços se comportar bem com relação a complemento ortogonal, vale que

$$\left(p(H)^*\right)^{-1} \operatorname{span}\{e_n\} = \operatorname{span}\{q_n\}.$$

De graça, portanto, estamos fazendo várias iterações de *shift-and-invert* na última coluna. Escolhemos o shift de Rayleigh porque ele é (localmente) a escolha ótima para shift-and-invert, como visto em sala; isso é o que ocorre no Caso 1 da seção 2.3.

Questão Extra. Para todo $k = 1, \dots, n$, verifique numericamente que $p(H)e_k$ pode ser expresso como combinação linear das colunas $\{q_1, \dots, q_k\}$, e portanto que estamos iterando subespaços. Lembre-se de se restringir ao pedaço da matriz que é Hessenberg próprio.

Como essa parte serve apenas para fins de verificação, não é necessário se preocupar com a complexidade do algoritmo. Existem vários jeitos de fazê-la:

- Em Julia ou em MATLAB, o comando `M\b` resolve o sistema $Mx = b$ quando há solução, ou acha a solução de mínimos quadrados no caso geral. Dado vetores $\{v_1, \dots, v_k\}$, você pode montar uma matriz M que tem os v_i como colunas, e usar isso para saber se um vetor w está no subespaço gerado pelos v.
- Você também pode ver se dois subespaços $\mathcal{S}_1$ e $\mathcal{S}_2$ são iguais gerando uma matriz com as colunas dos vetores que geram $\mathcal{S}_1$, outra com as colunas dos vetores que geram $\mathcal{S}_1 + \mathcal{S}_2$ (adicionando colunas) e comparando o posto numérico obtido com o comando `rank`.

2.5 Seção extra: E os autovetores no caso não simétrico?

Lembre-se que no caso simétrico, a matriz H inicial é tridiagonal (visto que é simétrica e Hessenberg). Como isso não ocorre no caso simétrico, a existência de coeficientes acima da diagonal faz com que as colunas de Q não sejam autovetores, com exceção da primeira.

No final do processo, teremos uma matriz triangular em blocos (com blocos 1×1 e 2×2 na diagonal). Achar os autovalores dessa matriz é fácil, mas ainda precisaríamos saber achar os autovetores de uma matriz como essa. Na seção 5.7 de [11], é detalhado o procedimento no caso em que não há blocos 2×2 (ou seja, a matriz é triangular); lá, Watkins explica que é possível generalizar o procedimento para o caso com blocos 2×2 na diagonal.

Nesse TP, não nos preocuparemos com achar autovetores de matrizes não-simétricas, apenas autovalores.

2.5.1 Subseção extra: E como achar os autovetores dos blocos 2×2?

A solução do Exercício 5.6.31 de Watkins, que explica como calcular os autovalores de uma matriz 2×2 através de transformações de similaridade (vide Seção 2.2.1), pode ser usada nesse caso.

3 SVD de matrizes reais $n \times m$ com $n \geqslant m$

Nosso objetivo é, dada uma matriz $A \in \mathbb{R}^{n \times m}$ qualquer, calcular matriz $\Sigma \in \mathbb{R}^{n \times m}$ diagonal e matrizes ortogonais $U \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{m \times m}$ tais que $A = U\Sigma V^T$.

O algoritmo também será iterativo. Para agilizar as iterações, primeiro reduziremos A a uma matriz bidiagonal.

3.1 Bidiagonalização de Golub–Kahan

Dizemos que uma matriz B é bidiagonal se $b_{i,j} = 0$ sempre que $j \neq i$ e $j \neq i+1$. Além disso, B é bidiagonal própria se todas as demais entradas são não nulas.

Questão 10. Implemente o algoritmo de bidiagonalização de Golub–Kahan.

	Nome	Descrição
Entrada	A	Matriz $n \times m$
Saída	U_0	Matriz $n \times n$ ortogonal
	B	Matriz $n \times m$ bidiagonal
	V_0	Matriz $m \times m$ ortogonal
Propriedades		$A = U_0 B V_0^T$
Complexidade		$O(nm^2)$

O algoritmo de Golub–Kahan [5] se aproveita do fato de que, ao contrário do problema de autovetores, não precisamos multiplicar pela mesma matriz dos dois lados. Assim, podemos limpar um pedaço da primeira coluna com uma multiplicação à esquerda, depois um pedaço da primeira linha com uma multiplicação à direita depois um pedaço da segunda linha, e assim por diante.

Como podemos ver, a matriz U_0 é o produto de no máximo m refletores (correspondentes às colunas) e V_0 é o produto de no máximo n refletores (correspondentes às linhas).

Na prática, se $n \gg m$, é vantajoso calcular uma decomposição QR de A e bidiagonalizar R depois. Não iremos implementar essa otimização.

Curiosidade extra: Como na redução à forma de Hessenberg, os espaços em branco da matriz B são suficientes para guardar os vetores que definem os refletores, faltando espaço apenas para os τ.

3.2 Golub–Reinsch: SVD de B bidiagonal própria

Veremos agora um algoritmo que, sem calcular BB^T ou B^TB, reduz o problema de calcular os valores singulares de B ao problema de autovalores e os calcula

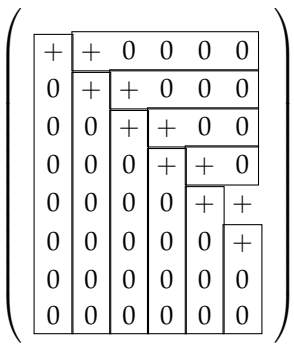

Figura 1: O que é usado para gerar os refletores

eficientemente. No que segue, iremos supor que B é uma matriz quadrada; assim, passar do resultado da bidiagonalização para o algoritmo de Golub–Reinsch requer um pouco de cuidado.

3.2.1 Motivação

Vimos em sala que, dada uma matriz B real, a matriz C definida por

$$C = \begin{pmatrix} 0 & B^T \\ B & 0 \end{pmatrix}.$$

reduz o problema de decomposição SVD ao problema de autovalores e autovetores. Se (v, u) denota o vetor-coluna obtido concatenando os vetores $v \in \mathbb{R}^m$ com $u \in \mathbb{R}^n$, a matriz acima é tal que:

- $C(v, u) = \sigma \cdot (v, u)$ se e somente se $B^T u = \sigma v$ e $Bv = \sigma u$.
- Vale que $C(v, u) = \sigma \cdot (v, u)$ se e só se $C(v, -u) = (-\sigma) \cdot (v, -u)$.
- Com o item anterior, vemos que se $\sigma > 0$ é tal que $C(v, u) = \sigma \cdot (v, u)$, então (v, u) é ortogonal a $(v, -u)$, por serem autovetores de uma matriz simétrica correspondentes a autovalores diferentes. Assim, $\|v\| = \|u\|$, e portanto podemos tomar u e v ambos com norma 1.

Veremos nas seções subsequentes que é possível executar o algoritmo de Francis para achar os autovalores da matriz C sem nem montá-la! É possível fazer as operações correspondentes diretamente na matriz B. Esse é o algoritmo de Golub–Reinsch [6].

3.2.2 A matriz $\tilde{C}$

Sabemos então que basta calcular os autovalores positivos da matriz C. Infelizmente, a matriz C não é tridiagonal (Hessenberg simétrica), que seria a forma

ideal para executarmos o algoritmo de Francis. No caso em que B é 5×5, a matriz C é exibida na Figura 2. Elementos não exibidos são zero.

$$C = \begin{pmatrix} 0 & B^T \\ B & 0 \end{pmatrix} = \begin{pmatrix}
0 & & & & & \beta_1 & 0 & & & \\
& 0 & & & & \gamma_1 & \beta_2 & 0 & & \\
& & 0 & & & 0 & \gamma_2 & \beta_3 & 0 & \\
& & & 0 & & & 0 & \gamma_3 & \beta_4 & 0 \\
& & & & 0 & & & 0 & \gamma_4 & \beta_5 \\
\beta_1 & \gamma_1 & 0 & & & 0 & & & & \\
0 & \beta_2 & \gamma_2 & 0 & & & 0 & & & \\
& 0 & \beta_3 & \gamma_3 & 0 & & & 0 & & \\
& & 0 & \beta_4 & \gamma_4 & & & & 0 & \\
& & & 0 & \beta_5 & & & & & 0
\end{pmatrix}$$

Figura 2: A matriz simétrica C da redução.

Para contornar esse problema, fazemos uma permutação das linhas e colunas de C para obter a matriz $\tilde{C}$, que iremos descrever pictorialmente na Figura 3. As colunas da matriz abaixo correspondem, respectivamente, às colunas 1, 6, 2, 7, 3, 8, 4, 9, 5, 10 (resultado de intercalar $1, 2, 3, 4, 5$ com $6, 7, 8, 9, 10$) da matriz C e similarmente para as linhas. Atente-se à posição dos zeros coloridos na matriz abaixo!

$$\tilde{C} = \begin{pmatrix}
0 & \beta_1 & & 0 & & & & & & \\
\beta_1 & 0 & \gamma_1 & & 0 & & & & & \\
& \gamma_1 & 0 & \beta_2 & & 0 & & & & \\
0 & & \beta_2 & 0 & \gamma_2 & & 0 & & & \\
& 0 & & \gamma_2 & 0 & \beta_3 & & 0 & & \\
& & 0 & & \beta_3 & 0 & \gamma_3 & & 0 & \\
& & & 0 & & \gamma_3 & 0 & \beta_4 & & 0 \\
& & & & 0 & & \beta_4 & 0 & \gamma_4 & \\
& & & & & 0 & & \gamma_4 & 0 & \beta_5 \\
& & & & & & 0 & & \beta_5 & 0
\end{pmatrix}$$

Figura 3: A matriz $\tilde{C}$, obtida por permutação das linhas e colunas de C.

3.2.3 Iteração de Golub–Reinsch

Agora, iremos aplicar uma iteração de Francis com shifts da forma $\pm\rho$, de modo que o polinômio relevante é $p(H) = (H - \rho I)(H + \rho I) = H^2 - \rho^2 I$. Vejamos passo a passo o que acontece ao fazer a iteração de Francis com esses shifts na matriz $\tilde{C}$ acima, relacionando com os elementos correspondentes da matriz B (os verdes).

- No primeiro passo (vide seção 2.2.4), multiplicamos a matriz $\tilde{C}$ por uma transformação Q_0. Calculando $p(H)e_1$, vemos que as únicas coordenadas não-nulas são a primeira e a terceira (a segunda coordenada é zero por causa dos shifts da forma $\pm\rho$). Assim, o refletor Q_0 vai combinar apenas as linhas/colunas 1 e 3.

  Ao calcular $Q_0^* \tilde{C} Q_0$, multiplicamos por Q_0 à esquerda e à direita. A multiplicação à esquerda combina as linhas 1 e 3 (alterando o elemento vermelho $(1,4)$, mas não alterando os elementos verdes). A multiplicação à direita combina as colunas 1 e 3, alterando o elemento verde $(4,1)$. Na matriz B, isso corresponde a multiplicar B à direita por uma transformação explícita V_0.

- Agora, queremos retornar à forma de Hessenberg. No segundo passo, precisamos limpar a primeira coluna da $\tilde{C}$ modificada. Para isso, iremos escolher um refletor (ou uma rotação) Q_1^* que combine as linhas 2 e 4. Ao fazê-lo, a multiplicação à esquerda por Q_1^* zera o elemento verde $(3,1)$ (o que corresponde a multiplicar B por uma transformação U_0 à esquerda) mas cria um elemento não-nulo verde na posição $(2,5)$. A multiplicação à direita por Q_1 combina as colunas 2 e 4, alterando o elemento vermelho simétrico $(5,2)$.

- No terceiro passo, queremos zerar o elemento vermelho $(5,2)$. Para isso, achamos um refletor Q_2^* que combine as linhas 3 e 5. A multiplicação à esquerda por Q_2^* zera o elemento desejado e altera o elemento vermelho $(3,6)$ (não alterando nenhum elemento verde), mas a multiplicação à direita por Q_2 combina as colunas 3 e 5, alterando o elemento simétrico verde $(6,3)$ (o que corresponde a multiplicar B por uma transformação V_1 à direita).

O procedimento continua descendo pelas diagonais de zeros coloridos: Os zeros verdes alterados aparecem alternadamente abaixo e acima da diagonal, e alterá-los corresponde a multiplicar B alternadamente por transformações V_i à direita e U_i à esquerda (no exemplo, os primeiros elementos verdes afetados são os da posição $(4,1)$, $(2,5)$, $(6,3)$). Olhando apenas para as posições onde os elementos afetados de B aparecem, obtemos a Figura 4.

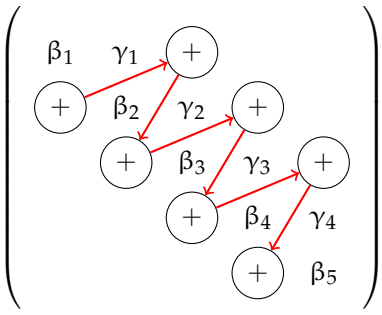

Figura 4: Aparição dos elementos não-nulos na matriz B verde.

Devido ao modo como as colunas foram intercaladas para gerar a matriz $\tilde{C}$, os refletores usados para retornar a matriz $\tilde{C}$ à forma de Hessenberg alteram, em B, elementos correspondentes aos demais sinais de $+$ na figura acima.

Na prática, fora a transformação que gera a entrada não-nula inicial, que precisa ser feita com cuidado para que a iteração de subespaços funcione, as demais podem ser intuídas: Para eliminar o primeiro $+$, por exemplo, combinamos as linhas 1 e 2, o que gera um elemento não-nulo na posição $(1, 3)$ de B. Agora, combinamos as colunas 2 e 3 para eliminar tal elemento não-nulo, e isso gera um novo elemento não-nulo na posição $(3, 2)$. O procedimento continua até que a matriz retorne para a forma bidiagonal.

Note que, como apenas um elemento não-nulo é gerado de cada vez, as transformações que retornam B à forma bidiagonal podem ser rotações de Givens ao invés de reflexões de Householder (como foi feito na seção 2.2.5).

Questão 11. Implemente uma iteração do algoritmo de Golub–Reinsch.

	Nome	Descrição
Entrada	B	Matriz $n \times n$ bidiagonal própria.
	U	Matriz ortogonal $n \times n$ com $A = UBV^T$
	V	Matriz ortogonal $m \times m$ com $A = UBV^T$
Saída	$\hat{B}$	Matriz bidiagonal.
		Sobrescreve as matrizes U, V de modo que $A = U\hat{B}V^T$.
Complexidade		$O(n^2)$ no caso geral.
		$O(n)$ se não quisermos alterar Q.

Dica: Lembre-se que você já implementou uma iteração de Francis! Se achar útil, pode montar a matriz $\tilde{C}$ e rodar a iteração nela com os shifts apropriados para ver o que acontece. Sua versão final não deve usar a matriz $\tilde{C}$, no entanto.

3.2.4 Como escolher o shift?

Note que, pelo formato da matriz $\tilde{C}$, é fácil calcular o shift de Rayleigh generalizado (os dois autovalores da matriz 2×2 $\tilde{C}_{\text{canto}}$): Eles são $\pm\beta_n$, ou, na notação original, $\pm b_{n,n}$.

3.2.5 O que fazer se B não for bidiagonal própria?

Para fins práticos, B não é diagonal própria se

$$|b_{i,i+1}| \leqslant u \cdot \left(|b_{i,i}| + |b_{i+1,i+1}|\right) \tag{3}$$

ou

$$|b_{i,i}| \leqslant u \cdot \left(|b_{i-1,i}| + |b_{i,i+1}|\right). \tag{4}$$

Tais condições correspondem à condição (2) na iteração de Francis em $\tilde{C}$.

Se (3) vale, ou seja, se algum $b_{i,i+1}$ for próximo de nulo, podemos zerar $b_{i,i+1}$ e ver a matriz como uma matriz no formato

$$B = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}.$$

Isso significa que temos dois subproblemas, um para B_1 e outro para B_2. Podemos fazer procedimento análogo ao da seção 2.3, usando variáveis ℓ e r para focar em submatrizes, para resolver ambos os subproblemas um após o outro.

Por outro lado, se (4) vale, ou seja, se algum $b_{i,i}$ for próximo de nulo, o problema não reduz imediatamente a dois problemas menores, e a matriz $\tilde{C}$ correspondente não será Hessenberg própria. Esse caso só ocorre se a matriz B tiver valores singulares nulos. O exercício 5.8.47 de Watkins explica como zerar, através de rotações à direita e à esquerda, as posições $b_{i-1,i}$ e $b_{i,i+1}$. Assim, recaímos no caso anterior.

Referências

[1] M. Ahues and F. Tisseur. A new deflation criterion for the qr algorithm. *LAPACK Working Note*, 122, 1997.

[2] Z. Bai and J. Demmel. On a block implementation of hessenberg multishift qr iteration. *International Journal of High Speed Computing*, 1(01):97–112, 1989.

[3] D. Bindel, J. Demmel, W. Kahan, and O. Marques. On computing givens rotations reliably and efficiently. *ACM Transactions on Mathematical Software (TOMS)*, 28(2):206–238, 2002.

[4] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):873–912, 1990.

[5] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224, 1965.

[6] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer, 1971.

[7] W. Kahan. On the cost of floating-point computation without extra-precise arithmetic. 2004. `http://www.cs.berkeley.edu/~wkahan/Qdrtcs.pdf`.

[8] B. N. Parlett. The rayleigh quotient iteration and some generalizations for nonnormal matrices. *Mathematics of Computation*, 28(127):679–693, 1974.

[9] B. N. Parlett. *The symmetric eigenvalue problem*, volume 20. siam, 1998.

[10] R. Schreiber and C. Van Loan. A storage-efficient wy representation for products of householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 10(1):53–57, 1989.

[11] D. S. Watkins. *Fundamentals of matrix computations*, volume 64. John Wiley & Sons, 2004.

[12] J. H. Wilkinson. Global convergene of tridiagonal qr algorithm with origin shifts. *Linear Algebra and its Applications*, 1(3):409–420, 1968.

[13] J. H. Wilkinson and C. Reinsch. *Handbook for Automatic Computation: Volume II: Linear Algebra*, volume 186. Springer Science & Business Media, 2012.

A Apêndice: O que há de errado com Bhaskara?

Vimos na escola que, se $a \neq 0$, as raízes da equação $ax^2 + bx + c$ podem ser calculadas por

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}, \tag{5}$$

conhecida no Brasil (e apenas no Brasil) como "fórmula de Bhaskara", e como "quadratic formula" em inglês. Uma variação, obtida multiplicando-se numerador e denominador pela quantia $-b \mp \sqrt{b^2 - 4ac}$, é

$$x_{1,2} = \frac{2c}{-b \mp \sqrt{b^2 - 4ac}}, \tag{6}$$

conhecida como "citardauq formula" (porque algumas pessoas, eu inclusive, não resistem a uma piada ruim).

Uma das coisas mais importantes que aprendemos sobre ponto flutuante nesse curso é o chamado *cancelamento catastrófico*: Mesmo que saibamos a e b com muitas casas de precisão, sabemos $a - b$ com poucas casas de precisão caso a e b sejam números muito próximos. Tendo isso em mente, há pelo menos dois problemas com a fórmula (5), pois há duas operações de soma/subtração (uma dentro da raiz e uma fora).

Resolver um dos problemas é fácil: Escolhemos o sinal do $\pm$ de modo a não ter cancelamento, e calculamos a outra raiz usando que o produto das raízes dá c/a:

$$x_1 = \frac{-b - \text{sinal}(b) \cdot \sqrt{b^2 - 4ac}}{2a} \qquad x_2 = \frac{c}{ax_1},$$

onde $\text{sinal}(b) = b/|b|$ (o caso $b = 0$ não é problema, pois não é necessário usar fórmulas nesse caso). Outro jeito é notar que as equações (5) e (6) se complementam: uma calcula uma raiz sem cancelamento catastrófico, e a outra calcula a outra.

Mas e o $b^2 - 4ac$? Pode haver problemas de cancelamento se as raízes forem quase iguais (além de possíveis overflows no cálculo de b^2). Kahan [7] tem um algoritmo que evita tais problemas se tivermos uma caixa-preta $\text{FMA}(a, b, c)$ que calcula $ab + c$ fazendo um único arredondamento, como é o caso em processadores modernos, mas iremos usar uma abordagem diferente.

B Apêndice: Onde ver implementações robustas?

A referência clássica para implementações robustas é o volume 2 do Handbook of Automatic Computation [13]. Muitas das propriedades teóricas desses algoritmos estão bem-explicadas em [9].

Desde então, foram feitas várias melhorias incrementais (e algumas radicais) nos algoritmos para os problemas estudados. Uma coleção de artigos, denominada *LAWNs* (*LAPACK Working Notes*), disponível em `http://www.netlib.org/lapack/lawns/downloads/index.html`, documenta descobertas importantes que são usadas em implementações robustas como a LAPACK.

Para referência, algumas rotinas relevantes da biblioteca LAPACK são as seguintes. Todas as funções abaixo usam representações compactas das entradas e saídas sempre que possível.

- `DGEHRD`: Redução de Hessenberg com refletores.
- `DLANV2`: Autovalores e autovetores de matriz 2×2 real, usando rotações.
- `DHSEQR`: Autovalores de matriz Hessenberg, usando multishift para aumentar paralelismo.
- `DLAHQR`: Autovalores de matriz Hessenberg, com iteração de Francis (dois shifts).
- `DGEBRD`: Bidiagonalização de Golub–Kahan.
- `DBDSCD`: SVD de matriz bidiagonal com divisão-e-conquista. Eventualmente chama `DLASDQ`.
- `DLASDQ`: Redução de SVD retangular a SVD quadrada. Usa decomposição QR para agilizar o processo se uma dimensão for muito maior que a outra.
- `DBDSQR`: SVD de matrizes quadradas. Usa o algoritmo de Demmel–Kahan se quisermos os vetores singulares, ou o algoritmo *dqds* se quisermos apenas os valores singulares.

Os nomes das rotinas na biblioteca LAPACK seguem a convenção `XYYZZZ`, onde:

- `X` é a precisão da entrada (`S` para single real, `D` para double real, `C` para single complexo e `Z` para double complexo),
- `YY` é o tipo da matriz de entrada (por exemplo, rotinas com `GE` aceitam matrizes gerais e as com `HS` aceitam só matrizes Hessenberg),
- `ZZZ` descreve o procedimento a ser realizado (por exemplo, `BRD` faz uma *bidiagonal reduction*).

C Apêndice: Detalhes de implementações reais

C.1 O algoritmo de Demmel–Kahan

O algoritmo de Demmel–Kahan [4] é um algoritmo para calcular a SVD. É uma modificação do algoritmo de Golub–Reinsch que vimos. Ele visa garantir que todos os valores singulares sejam calculados com erro relativo pequeno, mesmo que alguns dos valores singulares sejam muito maiores que outros. A ideia principal do algoritmo é similar, mas ele tem várias espertezas para manter a precisão e possibilitar a análise do algoritmo. Uma ideia chave é usar shifts 0 para extrair valores singulares pequenos.

Uma consequência dessa ideia é que os autovalores nulos são extraídos diretamente, o que significa que o caso em que B não é própria não precisa ser tratado em separado.

C.2 Iteração de Francis de grau 1 no caso complexo

O principal problema de fazer uma iteração de Francis de grau 1 no caso real é nos forçar a lidar com números complexos. No caso complexo, isso evidentemente não é um problema, e a iteração de Francis de grau 1 torna-se interessante. Nesse caso, pode-se usar o shift de Wilkinson para garantir convergência global sem maiores problemas; esse é o outro caso em que a biblioteca LAPACK usa shifts de Wilkinson.

C.3 Multishift

Ver [2].

C.4 Algoritmos divisão-e-conquista para autovalores

Ver algoritmo de Cuppen (página 502 de [11]).

C.5 Representação WY para refletores

Na prática, multiplicar uma matriz $m \times n$ por uma $n \times p$ é muito mais eficiente do que fazer p multiplicações matriz-vetor. Isso ocorre porque é possível multiplicar matrizes em blocos, isto é, dividir uma matriz grande em um número constante de submatrizes, multiplicá-las e depois combinar os resultados. Multiplicar matrizes em blocos explora os recursos computacionais (múltiplos processadores, cache do processador) de maneira muito mais eficiente.

Sabendo disso, podemos nos perguntar o seguinte: Será que existe um jeito mais eficiente de aplicar k refletores de Householder? A representação WY [10] é uma maneira compacta de representar k refletores por uma matriz pequena.