Metodi di Approssimazione

Course notes

federico.poloni@unipi.it

April 17, 2025

This document contains work-in-progress notes on the topics treated in the course. They were originally converted semi-automatically from the Beamer slides that I used in the past years, so the wording is still “slides-like” in many parts, but I am trying to fix it and expand on the proofs and text.

They are meant mainly for me to use as notes while I am teaching, but they could be useful for students and independent learners, too.

Chapter 1

Sylvester equations and invariant subspaces

Before dealing with matrix functions, we start from some related topics.

Vectorization *Vectorization* is a way to construct an explicit basis for the vector space of $m \times n$ matrices.

We define the map $\operatorname{vec}\colon \mathbb{C}^{m\times n} \to \mathbb{C}^{mn}$ as

$$X = \left[\begin{array}{c|c|c|c} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{array}\right] \in \mathbb{C}^{m\times n}, \quad \operatorname{vec} X := \left[\begin{array}{c} x_{11} \\ x_{21} \\ \vdots \\ x_{m1} \\ \hline x_{12} \\ x_{22} \\ \vdots \\ x_{m2} \\ \hline \vdots \\ \hline x_{1n} \\ x_{2n} \\ \vdots \\ x_{mn} \end{array}\right].$$

In plain words, $\operatorname{vec} X$ is the vector obtained by stacking the columns of the matrix X one on top of the other.

With vectorization, the elements of X are listed in *column-major* order: the leftmost index is the one that 'changes more often'. This choice matches the memory layout in which matrices are stored in memory in most languages, like

Matlab, Fortran, Python/Numpy (C/C++ prefer row-major instead), and it is often the most convenient one to work with.

```
>> A = [1 3 5; 2 4 6]
A =
       1 3 5
       2 4 6
>> reshape(A, [6,1]) % or A(:)
ans =
       1
       2
       3
       4
       5
       6
>> reshape(A, [3,2])
ans =
       1 4
       2 5
       3 6
```

There are explicit formulas to convert indices in the matrix into indices in the vector:

$$\begin{aligned} (X)_{ij} &= (\operatorname{vec} X)_{i+mj} && \text{0-based,} \\ (X)_{ij} &= (\operatorname{vec} X)_{i+m(j-1)} && \text{1-based.} \end{aligned}$$

In many cases, it is more convenient to use the pair (i,j) as a vector index, without performing this conversion.

Let now A, X, B be matrices of compatible dimensions for the product AXB; more precisely, $A \in \mathbb{C}^{p\times m}, X \in \mathbb{C}^{m\times n}, B \in \mathbb{C}^{n\times q}$. We are interested in finding out an expression for the matrix associated to the linear map $X \mapsto AXB$ in the basis given by vectorization; i.e., the matrix K (as a function of A and B) such that

$$\operatorname{vec}(AXB) = K \operatorname{vec} X \quad \forall X.$$

Let us take a generic element $(AXB)_{hl}$; this ends up in position $(\operatorname{vec}(AXB))_{h+p(l-1)}$ then,

$$(AXB)_{hl} = \sum_j (AX)_{hj}(B)_{jl} = \sum_j \sum_i A_{hi}X_{ij}B_{jl}$$
$$= \left[\begin{array}{cccc|cccc|c} A_{h1}B_{1l} & A_{h2}B_{1l} & \dots & A_{hm}B_{1l} & A_{h1}B_{2l} & A_{h2}B_{2l} & \dots & A_{hm}B_{2l} & \dots \end{array}\right.$$
$$\left.\begin{array}{|ccc} A_{h1}B_{nl} & A_{h2}B_{nl} & A_{hm}B_{nl} \end{array}\right] \operatorname{vec} X.$$

Hence, row $h + p(l-1)$ of K contains multiple copies of row h of matrix A, multiplied each time by a different entry in column l of B. Putting everything together, we obtain the expression

$$\operatorname{vec}(AXB) = \begin{bmatrix} b_{11}A & b_{21}A & \dots & b_{n1}A \\ b_{12}A & b_{22}A & \dots & b_{n2}A \\ \vdots & \vdots & \ddots & \vdots \\ b_{1q}A & b_{2q}A & \dots & b_{nq}A \end{bmatrix} \operatorname{vec} X. \tag{1.1}$$

Each block is a multiple of A, with coefficient given by the corresponding entry of $B^\top$. We can give a definition that models this structure.

Kronecker product Given two matrices $F \in \mathbb{C}^{m\times n}$ and $G \in \mathbb{C}^{p\times q}$, their Kronecker product, $F \otimes G$, is given by

$$F \otimes G := \begin{bmatrix} f_{11}G & f_{12}G & \dots & f_{1n}G \\ f_{21}G & f_{22}G & \dots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ f_{m1}G & f_{m2}G & \dots & f_{mn}G \end{bmatrix}.$$

With this definition, the matrix appearing in (1.1) can be expressed as $B^\top \otimes A$, where $B^\top$ denotes the transpose of B.

Properties of Kronecker products

- $\operatorname{vec} AXB = (B^\top \otimes A)\operatorname{vec} X$. *Warning*: this is not B^*, even when the matrices are complex. This is one of the few places where one has to deal with transposition-without-conjugation.
- $(A \otimes B)(C \otimes D) = (AC \otimes BD)$, when dimensions are compatible. *Proof*: $B(DXC^\top)A^\top = (BD)X(AC)^\top$.
- $(A \otimes B)^\top = A^\top \otimes B^\top$, and analogously for the conjugate-transpose $(A \otimes B)^*$.

Exercise 1.1. One can see that, for each m, n, there exists a permutation matrix $\Pi \in \mathbb{C}^{mn\times mn}$ such that $\Pi \operatorname{vec} X = \operatorname{vec}(X^\top)$ for each $X \in \mathbb{C}^{m\times n}$. This matrix is known as the commutation matrix, or perfect shuffle matrix.

1. What are the entries of the perfect shuffle matrix for $m = n = 2$? What about $m = 3, n = 2$?
2. Show that $A \otimes B = \Pi(B \otimes A)\Pi^\top$ for each A, B of suitable sizes.

A matrix equation is an equation whose unknown is a matrix. One of the simplest ones is the following, known as Sylvester equation: given A, B, C, find X that satisfies

$$AX - XB = C \tag{1.2}$$

with $A \in \mathbb{C}^{m\times m}$, $C, X \in \mathbb{C}^{m\times n}$, $B \in \mathbb{C}^{n\times n}$.

This is a $mn \times mn$ linear system, in fact, since the LHS is a linear function of X.

Solvability criterion for Sylvester equation

Theorem 1.2. *The Sylvester equation* (1.2) *has a unique solution iff* $\Lambda(A) \cap \Lambda(B) = \emptyset$.

Proof.

$$AX - XB = C \iff (I_n \otimes A - B^\top \otimes I_m)\,\mathrm{vec}(X) = \mathrm{vec}(C).$$

Let $A = Q_A U_A Q_A^*$, $B^\top = Q_B U_B Q_B^*$ be Schur decompositions. Then,

$$K = I_n \otimes A - B^\top \otimes I_m = (Q_B \otimes Q_A)(I_n \otimes U_A - U_B \otimes I_m)(Q_B \otimes Q_A)^*. \quad (1.3)$$

is a Schur decomposition: indeed, $(Q_B \otimes Q_A)(Q_B \otimes Q_A)^* = Q_B Q_B^* \otimes Q_A Q_A^* = I_n \otimes I_m = I_{mn}$, and $I_n \otimes U_A - U_B \otimes I_m$ is a triangular matrix.

We can read off the eigenvalues of K from this decomposition. What entries appear on the diagonal of $I_n \otimes U_A - U_B \otimes I_m$? If $\Lambda(A) = \{\lambda_1, \dots, \lambda_m\}$, $\Lambda(B) = \{\mu_1, \dots, \mu_n\}$, then on the diagonal we have elements $\Lambda(I_n \otimes A - B^\top \otimes I_m) = \{\lambda_i - \mu_j : i, j\}$. □

Exercise 1.3. Starting from singular value decompositions $A = U_A S_A V_A^*$ and $B = U_B S_B V_B^*$, use an idea similar to the one in (1.3) to construct a singular value decomposition of $A \otimes B$. (Note that one needs to reorder the singular values, since they do not appear in decresing order, and that further reorderings are needed in the case in which A, B are rectangular.)

Use this result to conclude that $\|A \otimes B\|_2 = \sigma_{\max}(A \otimes B) = \sigma_{\max}(A) \otimes \sigma_{\max}(B) = \|A\|_2 \|B\|_2$.

Solution algorithms We have seen that a Sylvester equation is equivalent to the linar system $K \,\mathrm{vec}\, X = \mathrm{vec}\, C$, with $K = I_n \otimes A - B^\top \otimes I_m$. Solving this linear system would cost $O((mn)^3)$, if one uses standard algorithms such as Gaussian elimination / LU factorization.

A much better algorithm is the *Bartels–Stewart algorithm* (1972), which solves the problem in $O(m^3 + n^3)$ exploiting the structure of the matrix.

Idea: invert factor by factor the decomposition

$$(Q_B \otimes Q_A)(I_n \otimes U_A - U_B \otimes I_m)(Q_B \otimes Q_A)^*.$$

- Solving orthogonal systems $\iff$ multiplying by their transpose, $O(m^3 + n^3)$ using the $\otimes$ structure.
- Solving upper triangular system $\iff$ back-substitution; costs $O(\mathrm{nnz}) = O(m^3 + n^3)$.

Bartels–Stewart algorithm A more operational description is the following.

Step 1: reduce to a triangular equation. Take Schur forms

$$Q_A U_A Q_A^* X - X \overline{Q_B} U_B^\top Q_B^\top = C$$

(take care not to mix up * and $^\top$); i.e., setting for ease of notation $L_B := U_B^\top$ (mnemonic: U / L for upper / lower triangular)

$$U_A Y - Y U_B^\top = E, \quad Y = Q_A^* X \overline{Q_B}, \quad E = Q_A^* C \overline{Q_B}.$$

We can compute E immediately. *Step 2*: Solve the equation $U_A Y - Y L_B = E$ by back-substitution to get Y. We can compute each entry Y_{ij}, by using the (i,j)th equation, as long as we have computed all the entries *below* and *to the right* of Y_{ij}. For instance, take the following 4×3 example: we wish to compute the $(2,2)$ entry of the equation $U_A Y - Y L_B = E$. To compute the product in the LHS, we need only to access the entries in red and blue.

$$\begin{bmatrix} * & * & * & * \\ 0 & \color{red}{*} & \color{red}{*} & \color{red}{*} \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{bmatrix} \begin{bmatrix} * & * & * \\ * & \color{blue}{*} & * \\ * & \color{red}{*} & * \\ * & \color{red}{*} & * \end{bmatrix} - \begin{bmatrix} * & * & * \\ * & \color{blue}{*} & \color{red}{*} \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} * & 0 & 0 \\ * & \color{red}{*} & 0 \\ * & \color{red}{*} & * \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & \color{red}{*} & * \\ * & * & * \\ * & * & * \end{bmatrix}$$

In particular, we can solve for the entry $Y_{2,2}$ displayed in blue, once we have computed all the entries below and to the right.

In general, we have the formula

$$\sum_{k \geq i} (U_A)_{ik} Y_{kj} - \sum_{k \geq j} Y_{ik} (L_B)_{kj} = E_{ij}$$

from which we can solve for Y_{ij} as

$$Y_{ij} = \frac{E_{ij} - \sum_{k>i} (U_A)_{ik} Y_{kj} + \sum_{k>j} Y_{ik} (L_B)_{kj}}{(U_A)_{ii} - (L_B)_{jj}}.$$

Step 3: $X = Q_A Y Q_B^\top$.

```
function Y = sylv_triangular(UA, LB, E)
% solve UA*Y - Y*LB = E with UA upper triangular
% and LB lower triangular

[m, n] = size(E);
Y = zeros(m,n);

for j = n:-1:1
    for i = m:-1:1
        Y(j+1:end,j)
        Y(i,j+1:end)
        num = E(i,j) - UA(i,i+1:end)*Y(i+1:end,j) ...
                    + Y(i,j+1:end)*LB(j+1:end,j);
        den = UA(i,i) - LB(j,j);
        Y(i,j) = num / den;
    end
end
```

```
>> rng(0);
>> m = 6; n = 4;
>> UA = triu(randn(m));
>> LB = tril(randn(n));
>> E = randn(m, n);
>> Y = sylv_triangular(UA, LB, E);
>> Y
Y =
   6.8220e+04 8.5491e+01 7.0973e+01 3.0548e+01
   3.5020e+04 1.0850e+02 9.6090e+01 4.8133e+01
   1.1360e+03 2.3123e+01 2.3093e+01 1.3657e+01
   1.5027e+02 2.4580e+00 3.0929e+00 4.1007e+00
  -7.5619e+02 -1.7189e+01 -1.9498e+01 -1.7543e+01
   2.7747e+00 6.6523e-01 1.0627e+00 -1.0303e+00
>> norm(UA*Y - Y*LB - E)
ans =
   3.4933e-12
>> norm(UA*Y - Y*LB - E) / (norm(UA)*norm(Y)+norm(Y)*norm(LB)+norm(E))
ans =
   7.2368e-18
```

The absolute residual is small, but not up to $\approx 10^{-15}$. The reason for this is that Y has large norm. This is not uncommon with Sylvester equation; we shall devote the next section to investigating their stability.

Exercise 1.4. Show that one can solve by substitution in a similar fashion also equations of the form $U_A Y - Y U_B = E$, with U_A, U_B upper triangular, or $L_A Y - Y L_B = E$, with L_A, L_B lower triangular. In which order do we need to compute the entries of Y in each case?

Exercise 1.5. Study in a similar way the matrix equation $X - AXB = C$ known as Stein's equation: show that the equation is uniquely solvable if and only if there exists no $\lambda \in \Lambda(A)$, $\mu \in \Lambda(B)$ such that $\lambda\mu = 1$, and formulate a Bartels–Stewart-like algorithm to solve the equation in time $O(m^3 + n^3)$.

Comments

- The idea (with some complications) works also with real Schur forms, i.e., block triangular forms with blocks of size 1 and 2: back-substitution gives block equations which are tiny 1×1, 1×2, 2×1 or 2×2 Sylvesters.
- A similar idea works for the more general case $AXB + CXD = E$, with some complications; one needs to start with the QZ decomposition of the pairs (A, C) and $(B^\top, D^\top)$.

- The idea does *not* work for three-term equations, $AXB+CXD+EXF=G$. For those, there is no (known) way to beat $O(m^3n^3)$.

Conditioning of Sylvester equations The most natural norm to study conditioning is $\|X\|_F = \|\operatorname{vec} X\|_2$.

The condition number $\kappa(K)$ is the ratio between

$$\sigma_{\max}(I \otimes A - B^\top \otimes I) = \|I \otimes A - B^\top \otimes I\| \leq \|I \otimes A\| + \|B^\top \otimes I\| \leq \|A\| + \|B\|$$

and

$$\sigma_{\min}(I \otimes A - B^\top \otimes I) = \min_Z \frac{\|AZ - ZB\|_F}{\|Z\|_F}.$$

There is no simple expression for this quantity, but it will appear in the following as well; we give it a name: the separation between A and B

$$\operatorname{sep}(A,B) := \sigma_{\min}(I \otimes A - B^\top \otimes I) = \min_Z \frac{\|AZ - ZB\|_F}{\|Z\|_F}.$$

An interesting case is the one in which A and B are normal matrices; this is equivalent to saying that in their Schur forms U_A, U_B are diagonal. When this happens, it is easy to see that K is normal, too. In this case, the Schur form is also (up to taking absolute values) a singular value decomposition, and hence

$$\sigma_{\min}(I \otimes A - B^\top \otimes I) = \min_{\lambda \in \Lambda(A), \mu \in \Lambda(B)} |\lambda - \mu|$$

But, in general, for a non-normal matrix M the minimum singular value $\sigma_{\min}(M)$ of a matrix can be arbitrary smaller than the absolute value of the smallest eigenvalue. As in many other settings, it is hard to bound singular values away from zero.

Exercise 1.6. Show that $\operatorname{sep}(A,B) = \operatorname{sep}(B,A)$, using the commutation / perfect shuffle matrix seen in Exercise 1.1.

Backward stability It is a classic result that back-substitution to solve a triangular linear system is backward stable, i.e., the computed solution $\tilde{x}$ computed in floating-point arithmetic satisfies exactly a nearby triangular linear system

$$(U + \Delta_U)\tilde{x} = b + \delta_b, \quad \|\Delta_U\|/\|U\| = O(\mathsf{u}), \quad \|\delta_b\|/\|b\| = O(\mathsf{u}).$$

Combining this with the fact that products with orthogonal matrices preserve norms, one can mimic the proof of backward stability for other algorithms based on orthogonal transformations (e.g., solving least squares with QR) and show that with the solution $\tilde{X}$ computed in floating-point arithmetic by the Bartels–Stewart method is the exact solution of a nearby linear system

$$(K + \Delta_K)\operatorname{vec}(\tilde{X}) = \operatorname{vec} C + \Delta_C, \quad \|\Delta_C\|_F/\|C\|_F, \quad \|\Delta_K\|/\|K\| = O(\mathsf{u}).$$

In particular, this implies a bound on the residual,

$$\|r\| = \|K \operatorname{vec} \tilde{X} - \operatorname{vec} C\| \leq O(\mathsf{u})(\|K\|\|\tilde{X}\| + \|C\|_F) \leq O(\mathsf{u})((\|A\| + \|B\|)\|\tilde{X}\| + \|C\|_F).$$

Note that in general we might have $\|X\| \approx \|\tilde{X}\| \gg \|A\|, \|B\|, \|C\|_F$, as in our Matlab example above, since $\|K^{-1}\| = \operatorname{sep}(A, B)^{-1}$ might be arbitrarily large.

However, an important observation is that Δ_K does not always have the same Kronecker-product structure of K; hence we <u>cannot</u> conclude that $\widetilde{X}$ solves a nearby matrix equation

$$(A + \Delta_A)\widetilde{X} - \widetilde{X}(B + \Delta_B) = C + \Delta_C, \tag{1.4}$$

i.e., we can show that the computed solution $\widetilde{X}$ is backward stable "as am $mn \times mn$ linear system" but not "as a Sylvester equation". The surprising result is that the backward error of the computed equation "as a Sylvester equation" is not guaranteed to be small, instead. We show why in the following.

Structural backward stability We are interested in studying the latter, stronger version of backward stability "as a Sylvester equation". This is called <u>structural</u> backward stability, since we are interested in finding a backward error matrix $\Delta_K = I \otimes \Delta_A + \Delta_B^\top \otimes I$ with the same structure as K.

To study structural backward stability, note that (1.4) is equivalent to the underdetermined linear system

$$\underbrace{\begin{bmatrix} -\widetilde{X}^\top \otimes I_n & I_m \otimes \widetilde{X} & I_{mn} \end{bmatrix}}_{=:S} \begin{bmatrix} \operatorname{vec} \Delta_A \\ \operatorname{vec} \Delta_B \\ \operatorname{vec} \Delta_C \end{bmatrix} = \underbrace{\operatorname{vec}(A\widetilde{X} - \widetilde{X}B - C)}_{=:r}.$$

Any solution of this linear system provides a perturbed matrix equation (1.4) whose (exact) solution is $\widetilde{X}$.

From standard results on underdetermined linear systems and least squares problems, the smallest-norm solution of the system is given by $S^+ r$, where S^+ is the Moore-Penrose pseudoinverse $S^+ = S^\top (SS^\top)^{-1}$. We have then

$$\left\| \begin{bmatrix} \operatorname{vec} \Delta_A \\ \operatorname{vec} \Delta_B \\ \operatorname{vec} \Delta_C \end{bmatrix} \right\| \leq \|S^+\|\|r\| = \frac{\|r\|}{\sigma_{\min}(S)} \leq \frac{O(\mathsf{u})((\|A\| + \|B\|)\|\tilde{X}\| + \|C\|)}{\sigma_{\min}(S)}.$$

To prove structured backward stability, we would like this quantity to be $O(\mathsf{u})$, when $\|A\|, \|B\|, \|C\| = O(1)$.

We now study the singular values of S. For simplicity, we shall restrict to the square case $m = n$. Up to an orthogonal change of basis, we can assume that $\tilde{X} = \operatorname{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$ is diagonal.

Then, we can compute explicitly $SS^\top = \tilde{X}^\top \tilde{X} \otimes I_n + I_n \otimes \tilde{X}\tilde{X}^\top + I_{n^2}$, which is the diagonal matrix with entries $\sigma_i^2 + \sigma_j^2 + 1$ for $i, j = 1, 2, \dots, n$. Hence, $\sigma_{\min}(S) = \sqrt{2\sigma_n^2 + 1}$. Thus we have

$$\left\| \begin{bmatrix} \operatorname{vec} \Delta_A \\ \operatorname{vec} \Delta_B \\ \operatorname{vec} \Delta_C \end{bmatrix} \right\| \leq \frac{O(\mathsf{u})((\|A\| + \|B\|)\sigma_1 + \|C\|)}{\sqrt{2\sigma_n^2 + 1}},$$

which might be large when $\sigma_1 \gg 1$ and $\sigma_n = O(1)$.

This analysis comes from [Higham '93], which contains also an explicit computed 2×2 example in which the backward error is large.

Block decoupling We can give an interesting viewpoint on the problem of solving Sylvester equations. We know from the theory of the Jordan form that

$$\begin{bmatrix} \lambda & 1 \\ 0 & \mu \end{bmatrix} \text{ is similar to } \begin{bmatrix} \lambda & 0 \\ 0 & \mu \end{bmatrix},$$

if $\lambda \neq \mu$, but

$$\begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix} \text{ is \textit{not} similar to } \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}.$$

In the first case, we can also obtain explicitly a change of basis matrix that realizes the transformation. Compute

$$\begin{bmatrix} 1 & x \\ 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \lambda & 1 \\ 0 & \mu \end{bmatrix} \begin{bmatrix} 1 & x \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \lambda & (\lambda-\mu)x + 1 \\ 0 & \mu \end{bmatrix}, \tag{1.5}$$

so if we take $x = \frac{-1}{\lambda-\mu}$ we have obtained an explicit change of basis. It is not surprising to see $\lambda - \mu$ in the denominator, since we know that the method must break down if $\lambda = \mu$.

With Sylvester equations, we can make a block version of this argument. Let us start from the block triangular matrix

$$\begin{bmatrix} A & C \\ 0 & B \end{bmatrix},$$

with $\Lambda(A) \cap \Lambda(B) = \emptyset$, and compute

$$\begin{bmatrix} I & X \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} = \begin{bmatrix} A & AX - XB + C \\ 0 & B \end{bmatrix}, \tag{1.6}$$

hence, if we take X that solves the Sylvester equation $AX - XB = -C$, then this argument shows that

$$\begin{bmatrix} A & -C \\ 0 & B \end{bmatrix} \text{ and } \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$$

are similar. If A and B have close-by eigenvalues, then $\operatorname{sep}(A, B)$ is small, and we expect to obtain a large X; hence, an ill-conditioned change of basis matrix is needed to obtain the similarity.

1.1 Invariant subspaces

Let $M \in \mathbb{C}^{n \times n}$ be a diagonalizable matrix, which we can write as $M = V \Lambda V^{-1}$, $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$. Recall that a column v_i of V is an eigenvector of M associated to λ_i, i.e.,

$$M v_i = v_i \lambda_i, \quad i = 1, \dots, n.$$

Then, we define the invariant subspace of M associated to a certain subset of eigenvalues, let's say without loss of generality $\{\lambda_1, \dots, \lambda_s\}$ with $s \leq n$, as the span of the corresponding eigenvectors, $\operatorname{span}(v_1, v_2, \dots, v_s)$. More generally, given a subset $\mathcal{S} \subseteq \mathbb{C}$, for instance the unit disc, we say that the invariant subspace of M associated to $\mathcal{S}$ is the one obtained by taking all the eigenvalues in $\Lambda(M) \cap \mathcal{S}$.

Here it is important to note that if there are multiple eigenvalues we want to take all of them, so that the subspace is well-defined as a function of M: if M has a multiple eigenvalue $\lambda_1 = \lambda_2$, then the eigenvectors v_1, v_2 are not unique, but the eigenspace $\operatorname{span}(v_1, v_2)$ is unique and well-defined independently of the choice of V.

We can generalize this definition to non-diagonalizable matrices. Let us take a Jordan form $M = VJV^{-1}$. Recall from your linear algebra courses the columns v_i of V contain so-called Jordan chains, i.e., vectors that satisfy certain recurrence relations. In particular, the columns whose indices belong to blocks with eigenvalue $\lambda \in \Lambda(M)$ form a basis for the so-called generalized eigenspace

$$\mathcal{V}_\lambda = \{v \in \mathbb{C}^n : (M - \lambda I)^k v = 0 \text{ for some } k > 0\}.$$

The invariant subspace $\mathcal{V}$ associated to a subset $\mathcal{S} = \{\lambda_1, \dots, \lambda_s\} \subseteq \Lambda(M)$ is the sum of subspaces $\mathcal{V}_{\lambda_1} + \dots + \mathcal{V}_{\lambda_s}$ of all generalized eigenspaces associated to eigenvalues in $\mathcal{S}$, i.e., the span of the v_i with indices i that correspond to blocks with eigenvalues in $\mathcal{S}$.

Example: the stable invariant subspace An interesting example of invariant subspace is the so-called stable invariant subspace

$$\mathcal{V} = \{v : \lim_{k \to \infty} M^k v = 0\}, \tag{1.7}$$

This space appears, for instance, in the study of iterative methods for linear systems like the Jacobi and Gauss-Seidel method: the error at step k follows the recurrence relation $e_{k+1} = Me_k$, where M is a matrix called the iteration matrix of the method; so the method converges iff $e_0 \in \mathcal{V}$. We have the following result.

Theorem 1.7. *The stable invariant subspace $\mathcal{V}$ defined in* (1.7) *is the invariant subspace associated to the interior of the unit disc $\mathcal{D} = \{z \in \mathbb{C} : |z| < 1\}$.*

Proof. Write M in a Jordan decomposition ordered such that

$$M = VJV^{-1}, \quad V = \begin{bmatrix} V_1 & V_2 \end{bmatrix}, \quad J = \begin{bmatrix} J_1 & \\ & J_2 \end{bmatrix}, \tag{1.8}$$

where J_1 contains only the Jordan blocks with eigenvalues inside the unit disc $\mathcal{D}$, and J_2 only those on the circle or outside. Then, the invariant subspace associated to $\mathcal{D}$ is $\operatorname{Im} V_1$.

If $w \in \operatorname{Im} V_1$, then we can write

$$w = V_1 v_1 = \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} v_1 \\ 0 \end{bmatrix},$$

and hence

$$M^k w = VJV^{-1}w = V \begin{bmatrix} J_1^k & \\ & J_2^k \end{bmatrix} \begin{bmatrix} v_1 \\ 0 \end{bmatrix} = V \begin{bmatrix} J_1^k v_1 \\ 0 \end{bmatrix} \to 0 \quad \text{when } k \to \infty.$$

To prove the reverse inclusion, we need to take $w \notin \mathcal{V}$, and prove that $M^k w$ does not converge to 0. We write

$$w = V_1 v_1 + V_2 v_2 = V \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

with $v_2 \neq 0$, and we obtain

$$M^k w = V \begin{bmatrix} J_1^k v_1 \\ J_2^k v_2 \end{bmatrix}.$$

Suppose the <u>last</u> nonzero entry of v_2 is in position m. Then, since J_2 is upper triangular, one sees that the last nonzero entry of $J_2^k v_2$ is

$$(J_2^k v_2)_m = (J_2^k)_{mm}(v_2)_m = (J_2)_{mm}^k (v_2)_m.$$

This entry does not converge to zero, as $(J_2)_{mm}$ is an eigenvalue of M outside $\mathcal{D}$, and $(v_2)_m \neq 0$. □

Block triangular decomposition To compute an invariant subspace, we do not need a Jordan decomposition; it is enough to have a factorization with a block triangular matrix

$$M = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} \begin{bmatrix} U_1 & U_2 \end{bmatrix}^{-1}, \quad A \in \mathbb{C}^{n_1 \times n_1}, \quad B \in \mathbb{C}^{n_2 \times n_2}. \tag{1.9}$$

Theorem 1.8. *Suppose that we have a triangular decomposition* (1.9) *with* $\Lambda(A)$ *and* $\Lambda(B)$ *disjoint. Then,* $\operatorname{Im} U_1$ *is the invariant subspace associated to* $\Lambda(A) \subseteq \Lambda(M)$.

Proof. First note that $\Lambda(M) = \Lambda(A) \cup \Lambda(B)$, because (1.9) is a similarity. To prove the result, we shall transform (1.9) into a Jordan form. We have already seen that, if X solves a certain Sylvester equation,

$$\begin{bmatrix} A & C \\ 0 & B \end{bmatrix} = \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} I & -X \\ 0 & I \end{bmatrix}.$$

Now take Jordan forms $A = V_A J_A V_A^{-1}$, $B = V_B J_B V_B^{-1}$ to obtain

$$\begin{aligned} M &= \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} \begin{bmatrix} U_1 & U_2 \end{bmatrix}^{-1} \\ &= \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} I & -X \\ 0 & I \end{bmatrix} \begin{bmatrix} U_1 & U_2 \end{bmatrix}^{-1} \\ &= \underbrace{\begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} \begin{bmatrix} V_A & \\ & V_B \end{bmatrix}}_{V} \begin{bmatrix} J_A & 0 \\ 0 & J_B \end{bmatrix} \underbrace{\begin{bmatrix} V_A^{-1} & \\ & V_B^{-1} \end{bmatrix} \begin{bmatrix} I & -X \\ 0 & I \end{bmatrix} \begin{bmatrix} U_1 & U_2 \end{bmatrix}^{-1}}_{V^{-1}} \end{aligned}$$

Expanding the product, one sees that

$$V = \begin{bmatrix} U_1 V_A & U_1 X V_A + U_2 V_B \end{bmatrix}, \tag{1.10}$$

so U_1 spans the same subspace as the first n_1 columns of V. □

Invariant subspaces are invariant Let us take a triangular decomposition (1.9). If $w \in \operatorname{Im} U_1$, we can write $w = U_1 v_1$ for a certain $v_1 \in \mathbb{C}^{n_1}$. Then, direct computation shows that

$$Mw = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} \begin{bmatrix} v_1 \\ 0 \end{bmatrix} = U_1(Av_1). \tag{1.11}$$

This tells us that $Mw \in \operatorname{Im} U_1$, i.e., the subspace $\operatorname{Im} U_1$ is invariant under multiplication by M. This property motivates the name "invariant subspaces". Another interesting observation that we can draw from this computation is that A is the matrix that represents the linear operator $f : w \mapsto Mw$, $f : \operatorname{Im} U_1 \to \operatorname{Im} U_1$, in the basis given by the columns of U_1.

Some remarks:

- For a block triangular decomposition, $\operatorname{Im} U_2$ is in general <u>not</u> invariant; this can be seen for instance from (1.10). It is invariant only if $C = 0$, i.e., the decomposition is actually block diagonal.
- The inclusion $M \operatorname{Im} U_1 \subseteq \operatorname{Im} U_1$ can be strict, if M has zero eigenvalues.
- Actually, these "invariant subspaces" are <u>not</u> the only subspaces that are invariant under multiplication by M. If M has multiple eigenvalues, there are more. For instance, take the 3×3 Jordan block

  $$J = \begin{bmatrix} \lambda & 1 & \\ & \lambda & 1 \\ & & \lambda \end{bmatrix}.$$

  This matrix has only one generalized eigenspace $\mathcal{V}_\lambda = \mathbb{C}^3$, but the subspaces $\mathcal{V} = \operatorname{span}(e_1)$ and $\mathcal{V} = \operatorname{span}(e_1, e_2)$ also satisfy $J\mathcal{V} \subseteq \mathcal{V}$. Indeed, since J is block triangular, we can repeat the computation in (1.11) with the first block of size $n_1 = 1$ or $n_1 = 2$. However, we cannot say that these two subspaces $\mathcal{V}$ are "associated to a subset of the spectrum" in any meaningful way. We have given our definition based on associated eigenvalues, since that is the concept that we are interested in studying.

- More generally, one can prove (we do not do it here) that all subspaces that are invariant under multiplication by M can be obtained by taking a Jordan form of M, and selecting a certain number of initial vector from each Jordan chain.

Computing invariant subspaces using the Schur form A notable decomposition in the form (1.9) is the Schur decomposition $M = UTU^*$, where U is unitary and A, B blocks of a triangular T. Hence, for any n_1, the first n_1 columns of U in a Schur form span the invariant subspace associated to the eigenvalues $T_{11}, \dots, T_{n_1 n_1}$. However, in applications typically we are not happy with $\underline{\text{any}}$ invariant subspace; we want to be able construct the invariant subspace associated to a certain subset of the eigenvalues. The Schur form is typically computed with the QR/Francis algorithm, and in that algorithm we are not free to choose the order of the eigenvalues in $\operatorname{diag}(T)$. Hence, we need a method to reorder these eigenvalues.

Reordering Schur forms Given a Schur form $M = UTU^*$, we wish to compute another Schur form $M = \hat{U}\hat{T}\hat{U}^*$ that has the eigenvalues in a different order. We can solve this problem with the help of Sylvester equations.

It is enough to have a method to swap the eigenvalues of two diagonal blocks: given an upper triangular matrix

$$\begin{bmatrix} A & C \\ 0 & B \end{bmatrix}$$

with $\Lambda(A) \cap \Lambda(B) = \emptyset$, we show how to construct a unitary Q such that

$$Q^* \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} Q = \begin{bmatrix} \hat{B} & \hat{C} \\ 0 & \hat{A} \end{bmatrix},$$

where $\hat{A}, \hat{B}$ are triangular with $\Lambda(\hat{A}) = \Lambda(A)$, $\Lambda(\hat{B}) = \Lambda(B)$.

Then, we can apply this method repeatedly on blocks of eigenvalues that are in the wrong order: if we wish to swap the eigenvalues in the blocks T_{22} and T_{33} of

$$\begin{bmatrix} T_{11} & * & * & * \\ & T_{22} & * & * \\ & & T_{33} & * \\ & & & T_{44} \end{bmatrix},$$

then we construct Q as above with $A = T_{22}$, $B = T_{33}$ and take $\hat{U} = \operatorname{diag}(I, Q, I)U$.

Hence we can focus on the block-2×2 problem. Let X solve the Sylvester equation $AX - XB = -C$. By permuting rows and columns in (1.6), we get

$$\underbrace{\begin{bmatrix} 0 & I \\ I & -X \end{bmatrix}}_{=\left[\begin{smallmatrix} X & I \\ I & 0 \end{smallmatrix}\right]^{-1}} \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} \begin{bmatrix} X & I \\ I & 0 \end{bmatrix} = \begin{bmatrix} B & 0 \\ 0 & A \end{bmatrix}.$$

So the matrix $\left[\begin{smallmatrix} X & I \\ I & 0 \end{smallmatrix}\right]$ does something similar to what we want, but it is *not* unitary. However, we have already done a big part of the job: we found a basis for the invariant subspace $\operatorname{Im} \begin{bmatrix} X \\ I \end{bmatrix}$ associated to $\Lambda(B)$.

To conclude, we replace $\left[\begin{smallmatrix} X & I \\ I & 0 \end{smallmatrix}\right]$ with its QR factor: let $QR = \left[\begin{smallmatrix} X & I \\ I & 0 \end{smallmatrix}\right]$. Then we have

$$R^{-1}Q^* \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} QR = \begin{bmatrix} B & 0 \\ 0 & A \end{bmatrix},$$

i.e.,

$$Q^* \begin{bmatrix} A & C \\ 0 & B \end{bmatrix} Q = R \begin{bmatrix} B & 0 \\ 0 & A \end{bmatrix} R^{-1}.$$

The matrix in the RHS is a product of upper triangular matrices; we can write

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

and hence obtain

$$R \begin{bmatrix} B & 0 \\ 0 & A \end{bmatrix} R^{-1} = \begin{bmatrix} R_{11}BR_{11}^{-1} & * \\ 0 & R_{22}AR_{22}^{-1} \end{bmatrix}.$$

Hence we have constructed a new matrix that has a leading triangular block $\hat{B}$ with $\Lambda(\hat{B}) = \Lambda(B)$.

Remark Since A, B are already triangular, to solve the Sylvester equation $AX - XB = -C$ we only need the back-substitution part of the Bartels–Stewart algorithm; the Schur forms and orthogonal transformations are not needed here.

Matlab example Computing the stable invariant subspace of a matrix M with `ordschur`.

```
>> rng(0); M = 1/2 * randn(5, 5)
M =
   0.2688 -0.6538 -0.6749 -0.1025 0.3357
   0.9169 -0.2168 1.5175 -0.0621 -0.6037
   -1.1294 0.1713 0.3627 0.7448 0.3586
   0.4311 1.7892 -0.0315 0.7045 0.8151
   0.1594 1.3847 0.3574 0.7086 0.2444
>> [Q, U] = schur(M, 'complex');
>> norm(Q*U*Q' - M)
```

```
ans =
   3.0833e-15
>> diag(U)
ans =
   -0.5816 + 1.3637i
   -0.5816 - 1.3637i
   -0.2013 + 0.0000i
   0.9045 + 0.0000i
   1.8236 + 0.0000i
>> [Q2, U2] = ordschur(Q, U, [0 0 1 1 0]);
>> diag(U2)
ans =
   -0.2013 + 0.0000i
   0.9045 + 0.0000i
   -0.5816 + 1.3637i
   -0.5816 - 1.3637i
   1.8236 + 0.0000i
>> norm(Q2*U2*Q2' - M)
ans =
   3.5872e-15
```

The first two columns of `Q2` span the stable invariant subspace of M. If we do not want to hardcode the vector `[0 0 1 1 0]` that selects the eigenvalues inside the unit disc, we can use the instruction `abs(diag(U))<1`, which creates a vector of true/false values. The function `ordschur` also accepts as second argument some special strings that solve for common cases, for instance we can write `ordschur(Q, U, 'udi')` to select the interior of the unit disc, or `'lhp'` to select the left half plane.

Sensitivity of invariant subspaces One of the reasons why we care about invariant subspaces is that they show that certain problems have better sensitivity than a full eigendecomposition. Consider for instance the problem of finding the stable invariant subspace of

$$M = \begin{bmatrix} 1/2 & 1 & 1 & 1 \\ & 1/2+\varepsilon & 1 & 1 \\ & & 2 & 1 \\ & & & 2+\varepsilon \end{bmatrix}.$$

Naively, one may think that we need to compute all the eigenvectors of M for this task. This is an ill-conditioned problem: the leading 2×2 matrix has two very close eigenvalues $1/2$ and $1/2+\varepsilon$, and we have seen in (1.5) the technique to diagonalize it; the resulting eigenvectors are

$$v_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad v_2 = \begin{bmatrix} \varepsilon^{-1} \\ 1 \\ 0 \\ 0 \end{bmatrix},$$

with the second eigenvector depending on the inverse of the eigenvalue difference $(1/2+\varepsilon)-\varepsilon$. Similarly for the trailing 2×2 block: "separating" the eigenvectors associated to the two eigenvalues 2 and $2+\varepsilon$ is an ill-conditioned operation, and the eigenvector matrix V is severely ill-conditioned.

However, here is that we do not need to "separate" those two pairs of eigenvectors: $\operatorname{span}\{v_1, v_2\}$ does not depend on x, and we do not need any dangerous division to compute it.

Even if we want to get the anti-stable invariant subspace, i.e., the one associated to the outside of the unit disc, we only have to reorder the Schur decomposition to swap the first two and last two entries. That is a much easier task than a full diagonalization, since

$$\operatorname{sep}\left(\begin{bmatrix}1/2 & 1\\ 0 & 1/2+\varepsilon\end{bmatrix}, \begin{bmatrix}2 & 1\\ 0 & 2+\varepsilon\end{bmatrix}\right) \approx 0.7$$

for small ε, so solving the associated Sylvester equation is a well-conditioned problem.

So, similarly, while the two eigenvectors v_3, v_4 are themselves ill-conditioned, the invariant subspace $\operatorname{span}\{v_3, v_4\}$ is much less sensitive to perturbation.

We can capture this intuition in a theorem that relates the invariant subspaces of M and a pertu rbation $M+\delta_M$; we shall see that the sensitivity is related to the function $\operatorname{sep}(A,B)$.

For simplicity, we can assume that the matrix M already comes in a block triangular decomposition (1.9), and that $U = I$ (up to an orthogonal change of basis).

Theorem 1.9. *[Stewart Sun book, Section V.2.2] Let*

$$M = \begin{bmatrix}A & C\\ 0 & B\end{bmatrix}$$

be a matrix with $\operatorname{sep}(A,B) > 0$*, and consider a perturbation* $M+\delta_M$*, with*

$$\delta_M = \begin{bmatrix}\delta_A & \delta_C\\ \delta_D & \delta_B,\end{bmatrix};$$

set $a = \|\delta_A\|_F$ *and analogously for* b, c, d*. If*

$$\begin{aligned}&\operatorname{sep}(A,B) - a - b \geq 0,\\ &(\operatorname{sep}(A,B)-a-b)^2 - 4d(\|C\|_F + c) \geq 0 \qquad (1.12)\end{aligned}$$

(which both hold when the perturbation δ_M *is sufficiently small), then there is a (unique)* X *with*

$$\|X\|_F \leq \frac{2d}{\operatorname{sep}(A,B)-a-b} \qquad (1.13)$$

such that $\begin{bmatrix}I\\ X\end{bmatrix}$ *spans an invariant subspace of* $M+\delta_M$*.*

In particular, $\frac{2}{\operatorname{sep}(A,B)}$ *can be considered as an absolute condition number of the invariant subspace* $\operatorname{Im}\left[\begin{smallmatrix}I\\0\end{smallmatrix}\right]$ *with respect to perturbation of the matrix* M*.*

Proof. Note that $M + \delta M = \left[\begin{smallmatrix} A+\delta_A & C+\delta_C \\ \delta_D & B+\delta_B \end{smallmatrix}\right]$, with a small $(2,1)$ block. We wish to construct a similarity transformation which zeroes out this block and constructs a block triangular decomposition of $M+\delta_M$. To do this, we use ideas analogous to the ones we have seen in the rest of this section; however, the form of our equations will be slightly different, since this time we want to zero out a block below the diagonal, instead of above. Hence, we try constructing a block triangular decomposition of the form

$$\begin{bmatrix} I & 0 \\ X & I \end{bmatrix}^{-1} (M+\delta_M) \begin{bmatrix} I & 0 \\ X & I \end{bmatrix} = \begin{bmatrix} \hat{A} & \hat{C} \\ 0 & \hat{B} \end{bmatrix};$$

if this equation holds, then $\operatorname{Im} \begin{bmatrix} I \\ X \end{bmatrix}$ is the invariant subspace of $(M+\delta_M)$, associated to the eigenvalues in $\Lambda(\hat{A})$. If $\|X\|$ is sufficiently small, then $\hat{A}$ and $\hat{B}$ are close to A and B, respectively, and their spectra are disjoint.

Expanding out the product in the LHS, we see that we need to solve the equation

$$X(A+\delta_A) - (B+\delta_B)X - \delta_D + X(C+\delta C)X = 0. \tag{1.14}$$

Note that this equation is more complicated than a Sylvester equation, because it has a degree-2 term. It is called algebraic Riccati equation, and we will encounter it again in the course.

We shall prove that this algebraic Riccati equation has a sufficiently small solution X. To do this, we first rewrite it as a fixed-point problem.

The linear part of this equation can be vectorized to get the operator $T = A^\top \otimes I - I \otimes B$, with $\sigma_{\min}(T) = \operatorname{sep}(B,A) = \operatorname{sep}(A,B)$, and its perturbed version $\hat{T} = (A+\delta_A)^\top \otimes I - I \otimes (B+\delta_B)$, with

$$\sigma_{\min}(\hat{T}) \geq \sigma_{\min}(T) - \|\delta_A^\top \otimes I - I \otimes \delta_B\| \geq \operatorname{sep}(A,B) - a - b;$$

the first inequality follows from SVD perturbation results: $|\sigma_{\min}(T+E) - \sigma_{\min}(T)| \leq \|E\|$ for any perturbation E.

We can rewrite (1.14) as

$$\operatorname{vec} X = \underbrace{\hat{T}^{-1} \operatorname{vec}(\delta_D - X(C+\delta C)X)}_{\Phi(\operatorname{vec} X)}.$$

This relation has the form of a fixed-point equation $x = \Phi(x)$, where $x = \operatorname{vec}(X)$, for a certain continuous map $\Phi : \mathbb{C}^{n^2} \to \mathbb{C}^{n^2}$. We wish to show that there exists a $r > 0$ such that Φ sends the ball $\mathcal{B}_r = \{x : \|x\| \leq r\}$ into itself. If this holds, then Φ must have a fixed point by Brouwer's fixed-point theorem, i.e., there exists a X with $\|X\|_F \leq r$ that satisfies (1.14).

Let X be a matrix such that $\|X\|_F \leq r$; then, taking norms,

$$\begin{aligned} \|\Phi(x)\| &= \left\| \hat{T}^{-1} \operatorname{vec}(\delta_D - X(C+\delta C)X) \right\| \\ &\leq \left\| \hat{T}^{-1} \right\| (d + \|X\|_F^2(\|C\|_F + c)) = \frac{1}{\beta}(\alpha + \gamma r^2), \end{aligned} \tag{1.15}$$

where we have set $\alpha = d, \beta = \operatorname{sep}(A,B) - a - b = \frac{1}{\|\hat{T}^{-1}\|}, \gamma = \|C\|_F + c$.

Suppose that there is a $r > 0$ such that $\frac{1}{\beta}(\alpha + \gamma r^2) = r$; then, (1.15) shows that $\|\Phi(x)\| \leq r$, as we want. We need to show that such a r exists. To do this, we study the (scalar) quadratic equation $\alpha - \beta r + \gamma r^2 = 0$ to show that it has a positive root. In particular, we focus on its smaller root r_-, since we want r to be as small as possible.

We switch to the reverse equation $\alpha s^2 - \beta s + \gamma = 0$, whose roots are the reciprocals $s_\pm = \frac{1}{r_\mp}$. This gives

$$\frac{1}{r_\mp} = s_\pm = \frac{\beta \pm \sqrt{\beta^2 - 4\alpha\gamma}}{2\alpha}.$$

The smaller root r_- corresponds to the larger $\frac{1}{r_-} = s_+$. If $\beta^2 - 4\alpha\gamma \geq 0$, then there are two positive solutions, and the largest one satisfies $\frac{1}{r_-} = s_+ \geq \frac{\beta}{2\alpha}$. After substitution, one sees that the equations $\beta^2 - 4\alpha\gamma \geq 0$ and $r_- \leq \frac{2\alpha}{\beta}$ are precisely the two inequalities (1.12) and (1.13) which appear in the text of our theorem.

Hence, if (1.12) holds, then the map Φ sends the ball $\mathcal{B}_r$ into itself, and hence there exists a vector $x \in \mathcal{B}_r$, or equivalently a matrix X that satisfies (1.13), such that (1.14) holds, and thus $\begin{bmatrix} I \\ X \end{bmatrix}$ is an invariant subspace of $M + \delta_M$. $\square$

Remark Here we are using $\|X\|_F$ as a measure of the perturbation between subspaces. There are ways to define more formally the distance between two subspaces of $\mathbb{C}^n$, but they do not add much to this discussion.

Chapter 2

Matrix functions

2.1 Definition(s) of matrix functions

Polynomials of matrices

Definition 2.1. Given a scalar polynomial $p(x) = c_0 + c_1 x + \cdots + c_d x^d$, and a square matrix $A \in \mathbb{C}^{n\times n}$, we set

$$p(A) := c_0 I + c_1 A + \cdots + c_d A^d.$$

We want to give an explicit formula for $p(A)$ in terms of a Jordan decomposition $A = V \operatorname{blkdiag}(J_1, J_2, \dots, J_s) V^{-1}$.

A formula for $p(J_0)$ Let

$$J_0 = \begin{bmatrix} 0 & 1 & & \\ & 0 & \ddots & \\ & & \ddots & 1 \\ & & & 0 \end{bmatrix} \in \mathbb{C}^{k\times k},$$

the Jordan block with eigenvalue 0. Then,

$$p(J_0) = \begin{bmatrix} c_0 & c_1 & \dots & c_{k-1} \\ & c_0 & \ddots & \vdots \\ & & \ddots & c_1 \\ & & & c_0 \end{bmatrix}.$$

This follows from evaluating the powers of J_0. If $d < k-1$, we take $c_{d+1} = c_{d+2} = \cdots = 0$.

A formula for $p(J_\lambda)$ If

$$J_\lambda = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{bmatrix} \in \mathbb{C}^{k\times k},$$

then

$$p(J_\lambda) = \begin{bmatrix} p(\lambda) & p'(\lambda) & \dots & \frac{p^{(k-1)}(\lambda)}{(k-1)!} \\ & p(\lambda) & \ddots & \vdots \\ & & \ddots & p'(\lambda) \\ & & & p(\lambda) \end{bmatrix}.$$

This follows from writing the polynomial in its Taylor expansion

$$p(x) = p(\lambda) + p'(\lambda)(x-\lambda) + \frac{p''(\lambda)}{2!}(x-\lambda^2) + \cdots + \frac{p^{(d)}(\lambda)}{d!}(x-\lambda)^d, \quad (2.1)$$

which reduces us to the previous case. Note that the formula (2.1) continues to hold if we evaluate in a matrix argument $p(A)$, since it follows from algebraic manipulations with powers of A (which commute with each other).

Proposition 2.2. *If $A = VJV^{-1}$ is a Jordan form, and $J = \operatorname{blkdiag}(J_1, J_2, \dots, J_s)$ with each block $J_i = J_{\lambda_i}$ of size $k_i \times k_i$, then*

$$p(A) = V \operatorname{blkdiag}(p(J_1), p(J_2), \dots, p(J_s))V^{-1}, \quad p(J_i) = \begin{bmatrix} p(\lambda_i) & p'(\lambda_i) & \dots & \frac{p^{(k_i-1)}(\lambda_i)}{(k_i-1)!} \\ & p(\lambda_i) & \ddots & \vdots \\ & & \ddots & p'(\lambda_i) \\ & & & p(\lambda_i) \end{bmatrix}. \quad (2.2)$$

Indeed, we have

$$p(A) = \sum c_i (VJV)^{-1} = V\left(\sum c_i J^i\right)V^{-1} = V \operatorname{blkdiag}(p(J_1), p(J_2), \dots, p(J_s))V^{-1},$$

and we can conclude using the previous results.

Functions of matrices [Higham book, '08, Ch. 1] We can use the same formula (2.2) even for scalar functions that are not polynomials, leading to a definition of matrix functions

Definition 2.3 (attempted)**.** Given a function $f : U \subseteq \mathbb{C} \to \mathbb{C}$, and a matrix A with Jordan decomposition as above, we say that f is *defined on A* if f is defined and differentiable at least $k_i - 1$ times on each eigenvalue λ_i of A, and its value is

$$f(A) := V \operatorname{blkdiag}(f(J_1), f(J_2), \dots, f(J_s)) V^{-1},$$
$$f(J_i) := \begin{bmatrix} f(\lambda_i) & f'(\lambda_i) & \dots & \frac{f^{(k_i-1)}(\lambda_i)}{(k_i-1)!} \\ & f(\lambda_i) & \ddots & \vdots \\ & & \ddots & f'(\lambda_i) \\ & & & f(\lambda_i) \end{bmatrix}.$$

However, there is a problem with this definition: in the Jordan decomposition J is unique, but V is not. For this definition to be well-posed, $f(A)$ should be independent of the choice of V. From this formula, it is unclear if that is the case. To prove this independence, we use another equivalent definition.

Alternate definition: via Hermite interpolation

Definition 2.4. Given $f : U \subseteq \mathbb{C} \to \mathbb{C}$ and $A \in \mathbb{C}^{n \times n}$ with Jordan decomposition as above, we define $f(A) := p(A)$, where $p(x)$ is any polynomial such that

$$f(\lambda_i) = p(\lambda_i), f'(\lambda_i) = p'(\lambda_i), \dots, f^{(k_i-1)}(\lambda_i) = p^{(k_i-1)}(\lambda_i) \text{ for each } i. \quad (2.3)$$

Remarks:

- We shall prove soon that there always exists a polynomial $p(x)$ that satisfies these conditions.
- Note that this definition does not depend on the choice of $p(x)$ among all the polynomials that satisfy the given conditions, since (2.2) shows that the value of $p(A)$ depends only on the $f^{(j)}(\lambda_i)$: if p and $\hat{p}$ are two polynomials that satisfy the interpolation conditions (2.3), then $p(A) = \hat{p}(A)$.
- This definition coincides with the previous Definition 2.3, again in view of (2.2), but now it is clear that it is independent of V, as V does not appear at all in Definition 2.4.
- If A has more than one Jordan block with the same eigenvalue, some of these conditions are repeated. This is not a problem, because the conditions are always consistent.

Example: square root

$$A = \begin{bmatrix} 4 & 1 & & \\ & 4 & 1 & \\ & & 4 & \\ & & & 0 \end{bmatrix}, \quad f(x) = \sqrt{x}.$$

We look for an interpolating polynomial with

$$p(0) = 0,\, p(4) = 2,\, p'(4) = f'(4) = \frac{1}{4},\, p''(4) = f''(4) = -\frac{1}{32}.$$

These are four conditions, so it makes sense to look for a degree-3 polynomial that satisfies them. This produces the linear system

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ 4^3 & 4^2 & 4 & 1 \\ 3\cdot 4^2 & 2\cdot 4 & 1 & 0 \\ 6\cdot 4 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_3 \\ p_2 \\ p_1 \\ p_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ \frac{1}{4} \\ -\frac{1}{32} \end{bmatrix}, \tag{2.4}$$

whose solution is

$$p(x) = \frac{3}{256}x^3 - \frac{5}{32}x^2 + \frac{15}{16}x.$$

Hence we have

$$p(A) = \frac{3}{256}A^3 - \frac{5}{32}A^2 + \frac{15}{16}A = \begin{bmatrix} 2 & \frac{1}{4} & -\frac{1}{64} & \\ & 2 & \frac{1}{4} & \\ & & 2 & \\ & & & 0 \end{bmatrix}.$$

One can check that $f(A)^2 = A$; this fits our intuition of a "matrix square root". We will prove in the following that with this definition $X = f(A)$ is always a solution of the matrix equation $X^2 = A$.

Hermite interpolation We are ready to prove that a suitable polynomial always exists:

Theorem 2.5. *Given distinct points $x_1, x_2, \dots, x_n \in \mathbb{C}$ and multiplicities $m_1, m_2, \dots, m_n \in \mathbb{N}$, there exists a unique polynomial of degree $d < m_1 + m_2 + \cdots + m_n$ such that*

$$p(x_i) = y_{i,0},\, p'(x_i) = y_{i,1},\, \dots,\, p^{(m_i-1)}(x_i) = y_{i,m_i-1}, \quad \text{for all } i = 1, \dots, n.$$

for each choice of the $y_{i,j}$.

Proof (sketch)

- These interpolation conditions always produce a "Vandermonde-like" square linear system (like in (2.4)) in the coefficients of the polynomial $p(x)$.
- We need to prove that its matrix V has trivial kernel. If $Vz = 0$ for a vector z, then the conditions in this linear system tells us that the associated polynomial $z(x)$ satisfies

  $$z(x_i) = 0,\, z'(x_i) = 0,\, \dots,\, z^{(m_i-1)}(x_i) = 0, \quad \text{for all } i = 1, \dots, n.$$

  I.e., $z(x)$ has a root at each x_i of multiplicity at least m_i. By degree reasons, $z(x)$ must be the zero polynomial.

Note that in our construction we only need a polynomial satisfying the interpolation conditions; not necessarily the one with minimum degree.

Example: matrix sign

$$A = V \begin{bmatrix} -3 & & & \\ & -2 & & \\ & & 1 & 1 \\ & & & 1 \end{bmatrix} V^{-1}, \quad f(x) = \operatorname{sign}(x) = \begin{cases} 1 & \operatorname{Re} x > 0, \\ -1 & \operatorname{Re} x < 0. \end{cases}$$

$$f(A) = V \begin{bmatrix} -1 & & & \\ & -1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix} V^{-1}.$$

It is interesting to note that this matrix function is not locally constant, unlike its scalar counterpart: a small change to the matrix V can produce a small, nonzero change to the matrix $f(A)$.

Example: matrix exponential

$$A = V \begin{bmatrix} -1 & & & \\ & 1 & & \\ & & 1 & 1 \\ & & & 1 \end{bmatrix} V^{-1}, \quad f(x) = \exp(x).$$

$$\exp(A) = V \begin{bmatrix} e^{-1} & & & \\ & e & & \\ & & e & e \\ & & & e \end{bmatrix} V^{-1}$$

Since there are 2 blocks with the same eigenvalue 1, there are only 3 interpolation conditions rather than 4:

$$p(-1) = e^{-1}, \quad p(1) = e, \quad p'(1) = e.$$

This is not a problem, though; we can still find a polynomial $p(x)$ that satisfies these conditions. The conditions appear multiple times, but this process never produces incompatible conditions like $p(1) = e$, $p(1) = e^2$. Since there are fewer conditions, we can even find a polynomial of degree ≤ 2, rather than ≤ 3; but optimizing this degree has never been our goal.

Note that the matrix $B = \exp(A)$ obtained in this way coincides with the limit of the matrix-valued power series $I + A + \frac{1}{2}A^2 + \frac{1}{3!}A^3 + \dots$. It is simple to see this for the diagonal terms, since the diagonal entries of this power series are just the scalar series for e^{-1} and e, but more complicated to see for $B_{3,4}$. We will prove later in more generality that a matrix function can also be computed as the limit of a Taylor series.

Exercise 2.6. Using the Jordan form of A, show that if each eigenvalue of A satisfies $\operatorname{Re}(\lambda) < 0$ then

$$\lim_{t \to \infty} \exp(tA) = 0.$$

(This is an if-and-only-if, and matrices that satisfy this property are called Hurwitz stable.)

Non-Example: square root The matrix function

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \quad f(x) = \sqrt{x}$$

is not defined, using our definition, because $f'(0)$ does not exist.

Indeed, one can prove that the equation $X^2 = A$ has no solution with this choice of A: since $\Lambda(A) = \{0\}$, any solution X must have $\Lambda(X) = \{0\}$ as well. But then the Jordan form of X is either

$$X = V \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} V^{-1} \text{ or } X = V \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} V^{-1},$$

and we see that in both cases $X^2 = 0 \neq A$.

This provides additional evidence that our definition is a good one: it fails in a case when it should, since $X^2 = A$ has no solution.

Example: complex square root In the past examples, we have glossed over the detail that the square root function is not uniquely defined. We now see an example in which A has complex eigenvalues. Let

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \quad f(x) = \sqrt{x}.$$

This matrix has $\Lambda(A) = \pm i$. To specify $f(x)$ fully, we need to choose which square root to take for $f(i)$ and $f(-i)$, since there are two possible values: we shall take $f(i) = \frac{1}{\sqrt{2}}(1+i)$, $f(-i) = \frac{1}{\sqrt{2}}(1-i)$, so that both $f(i)$ and $f(-i)$ are in the right half-plane.

We look for an interpolating polynomial $p(x)$ to $f(x)$, satisfying $f(\pm i) = \frac{1}{\sqrt{2}}(1 \pm i)$; in particular we can take $p(x) = \frac{1}{\sqrt{2}}(1+x)$. Hence

$$p(A) = \frac{1}{\sqrt{2}}(I + A) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}.$$

Branches of the square root Note that there can be many different ways to define the square root of scalars, due to the sign ambiguity. One cannot define a square root function that is continuous on the whole $\mathbb{C}$, or even on $\mathbb{C} \setminus \{0\}$, but it is possible to do so locally on smaller sets: for instance, we can define the so-called principal square root on $U = \mathbb{C} \setminus \{a + 0i \colon a \leq 0\}$ to be the one of the two square roots that lies in the right half-plane. This is a continuous function; geometers call it a branch of the square root, and they say that this function has a branch cut on the negative reals.

For a matrix A with s non-zero distinct eigenvalues, we can define 2^s different functions $f_1, \dots, f_{2^s}$ by choosing the sign in each (distinct) eigenvalue. These can be extended locally to continuous functions $f_k : U \to \mathbb{C}$, $k = 1, \dots, 2^s$. We can do so by taking, for instance, U to be the union of s disjoint balls centered in the eigenvalues. These "local versions" of the square root that are defined continuously on an open set $U \subseteq \mathbb{C}$ are called branches. If we apply the definition of matrix function starting from an appropriate branch of the square root, we can produce 2^s different matrices. They all satisfy the matrix equation $X^2 = A$.

Exercise 2.7. Let $A = \left[\begin{smallmatrix} 0 & 1 \\ -1 & 0 \end{smallmatrix}\right]$, as above. Compute $g(A)$, where $g(x)$ is the branch of the square root function such that $g(i) = \frac{1}{\sqrt{2}}(1+i)$, $g(-i) = \frac{1}{\sqrt{2}}(-1+i)$. Note that, unlike the previous one, $g(A)$ does not have real coefficients. (Indeed, it can't be real, because its eigenvalues do not come in complex conjugate pairs.)

Non-example: nonprimary square root All the matrix functions $X = f(A)$ that we have constructed starting from different branches of the square-root function satisfy $X^2 = A$. It is natural to ask if these are all solutions to this equation. The answer is no: there can be more solutions in cases where A has multiple Jordan blocks with the same eigenvalue. For instance, take

$$A = V \begin{bmatrix} 1 & & \\ & 1 & \\ & & 2 \end{bmatrix} V^{-1}$$

and

$$X = V \begin{bmatrix} 1 & & \\ & -1 & \\ & & \sqrt{2} \end{bmatrix} V^{-1}.$$

It is clear that $X^2 = A$, but in our definition of matrix function we have to choose either $f(1) = 1$, or $f(1) = -1$, so there is no branch choice that produces the matrix X.

In general, whenever a function f has multiple branches (like the square root or the complex logarithm), and a matrix A has multiple Jordan blocks with the same eigenvalue λ, we can define 'pseudo-matrix-functions', like X above, by taking different values of $f(\lambda)$ for different Jordan blocks. These are sometimes called *nonprimary matrix functions*, even if they are *not* matrix functions with our definition. When X is a non-primary function of A, one can see that there is no polynomial p for which $X = p(A)$. We will not deal with non-primary functions further in the course, but we just mention their existence.

2.2 Properties of matrix functions

We have proved that for each (sufficiently regular) function f and each matrix A, there is a polynomial p such that $f(A) = p(A)$. One may be led to think that "all matrix functions can be reduced to polynomials, so the definition is not interesting", but this is misleading: note that the polynomial $p(x)$ that is used in the definition depends on A! This would be like claiming that "all scalar functions are polynomials" because given any function f and $z \in \mathbb{C}$ we can always find a polynomial such that $f(z) = p(z)$ (for instance, a constant polynomial).

However, this representation as a polynomial is very handy when proving properties of matrix functions. We list here a few properties of matrix functions; most of them can be proved by replacing $f(A)$ with a polynomial $p(A)$.

- $f(A)$ commutes with A: $Af(A) = f(A)A$.
- $f(MAM^{-1}) = Mf(A)M^{-1}$, since this property holds for polynomials, and we can use the same p to express both $f(MAM^{-1})$ and $f(A)$, as they have the same spectrum.
- $f(\left[\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right]) = \begin{bmatrix} f(A) & 0 \\ 0 & f(B) \end{bmatrix}$, for the same reason: take a polynomial p that interpolates f on $\Lambda(\left[\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right])$. The same polynomial can be used to interpolate f on $\Lambda(A)$ and $\Lambda(B)$, since it satisfies stricter interpolation conditions.
- If $h(x) = f(x)g(x)$ for three scalar functions f, g, h, then $f(A)g(A) = h(A)$. Proof: replace $f(A), g(A)$ with polynomials $p_f(A), p_g(A)$ which interpolate f on $\Lambda(A)$. Then, set $p_h(x) := p_f(x)p_g(x)$; $p_f(A)p_g(A) = p_h(A)$ is true (by expanding), and $p_h(x)$ is an Hermite interpolant for $h(x)$. Analogous properties hold for sums and compositions of functions.
- In particular, from the previous point it follows that for the square root function $f(x) = \sqrt{x}$ (any branch of it) we have $f(A)^2 = A$ for each A: just use the product property on $g(x) = f(x) = \sqrt{x}$, $h(x) = x$ the identity function.
- More generally, if $f(x)$ satisfies a certain scalar identity built with sums, products, compositions, then $f(A)$ satisfies the matrix version of the same identity. For instance, the matrix functions $\cos(A)$ and $\sin(A)$ satisfy the identity $\cos(A)^2 + \sin(A)^2 = I$ for each A.

Exercise 2.8. Show that the matrix function $f(A)$ corresponding to the scalar function $f(x) = x^{-1}$ is the matrix inverse $f(A) = A^{-1}$.

The following properties instead are simpler to prove relying on the expression of $f(A)$ using functions of Jordan blocks.

- If the eigenvalues of A are $\lambda_1, \dots, \lambda_s$, the eigenvalues of $f(A)$ are $f(\lambda_1), \dots, f(\lambda_s)$. Their algebraic multiplicities stay the same, but geometric multiplicities may increase (when $f'(\lambda_i) = 0$).
- For a sequence of functions f_n such that
$$\lim_{n\to\infty} f_n^{(j)}(\lambda_i) \quad j < k_i,$$
(i.e., all the derivatives that appear in the definition of $f(A)$ converge), then $f_n(A) \to f(A)$.

- A less trivial property is continuity in the argument A: given a sequence of matrices $A_n \to A$, is it true that $f(A_n) \to f(A)$? We will see a proof soon; but note that this statement is more complicated because each A_n corresponds to a <u>different</u> polynomial p_{f,A_n}.

Exercise 2.9. Let f be a function such that $|f(x)| \leq \varepsilon$ for each x in the closed unit disc $\overline{D} = \{z \in \mathbb{C} \colon |z| \leq 1\}$. Let $A \in \mathbb{C}^{n\times n}$ be a matrix with $\Lambda(A) \subseteq \overline{D}$. Is $\|f(A)\|$ bounded, or can it be arbitrarily large?

Taylor series Another natural way to define matrix functions is via Taylor series. For instance, you have first defined the matrix exponential as

$$\exp(A) = I + A + \frac{1}{2!}A^2 + \frac{1}{3!}A^3 + \dots.$$

We can show that this coincides with the definition that we have given above; and, more generally, that we can compute matrix functions as limit of Taylor series.

Theorem 2.10 (Higham book Thm. 4.7)**.** *Suppose $f(x) = \sum_{k=0}^{\infty} c_k(x-\alpha)^k$, with $c_k = \frac{f^{(k)}(\alpha)}{k!}$, is a Taylor series with convergence radius r. Then,*

$$\lim_{d\to\infty} \sum_{k=0}^{d} c_k(A-\alpha I)^k = f(A)$$

for each A whose eigenvalues satisfy $|\lambda - \alpha| < r$.

Proof. Let $p_d(x) = \sum_{k=0}^{d} c_k(x-\alpha)^k$ be the truncated Taylor polynomial of degree d, i.e., the polynomial obtained by truncating the Taylor series at degree d.

Let $A = V \operatorname{blkdiag}(J_1, \dots, J_s)V^{-1}$ be a Jordan form. We have seen that

$$f(A) = V \operatorname{blkdiag}(f(J_1), \dots, f(J_s))V^{-1}$$

and

$$p_d(A) = V \operatorname{blkdiag}(p_d(J_1), \dots, p_d(J_s))V^{-1},$$

so it is sufficient to prove our statement for a single Jordan block J with eigenvalue λ.

The matrix $p_d(J)$ has $p_d(\lambda)$ on its diagonal, and clearly this converges to $f(\lambda)$. On the i-th superdiagonal of $p_d(J)$, we have

$$\frac{1}{i!}p_d^{(i)}(\lambda).$$

Note that $p_d^{(i)}(x)$ the Taylor polynomial of degree $d-i$ for the derivative $f^{(i)}(x)$, since power series can be differentiated term-by-term. We recall from our analysis courses that the power series for the derivative of a function has the same radius of convergence as the original power series. (This follows, from instance, from the so-called "root test" $1/r = \limsup_{n\to\infty} |c_n|^{1/n}$.) Hence, the terms on the i-th superdiagonal of $p_d(J)$ converge to $\frac{1}{i!}f^{(i)}(\lambda)$. $\square$

Matrix functions as Cauchy integrals The following result generalizes Cauchy's integral formula from complex analysis.

Proposition 2.11. *If f is holomorphic on and inside a contour Γ that encloses the eigenvalues of A,*

$$f(A) = \frac{1}{2\pi i}\int_\Gamma f(z)(zI - A)^{-1}dz. \tag{2.5}$$

Proof. Arguing as in the theorem above on the convergence of Taylor series, we can reduce to the case of a single Jordan block. Then,

$$\frac{1}{2\pi i}\int_\Gamma f(z)(zI-J)^{-1}dz = \frac{1}{2\pi i}\int_\Gamma f(z)\begin{bmatrix} z-\lambda & -1 & & \\ & z-\lambda & -1 & \\ & & \ddots & \ddots \\ & & & z-\lambda \end{bmatrix}^{-1} dz$$

$$= \frac{1}{2\pi i}\int_\Gamma f(z)\begin{bmatrix} \frac{1}{z-\lambda} & \frac{1}{(z-\lambda)^2} & \cdots & \frac{1}{(z-\lambda)^{k-1}} \\ & \ddots & \ddots & \vdots \\ & & \ddots & \frac{1}{(z-\lambda)^2} \\ & & & z-\lambda \end{bmatrix} dz$$

$$= \begin{bmatrix} \frac{1}{2\pi i}\int_\Gamma \frac{f(z)}{z-\lambda}dz & \frac{1}{2\pi i}\int_\Gamma \frac{f(z)}{(z-\lambda)^2}dz & \cdots & \frac{1}{2\pi i}\int_\Gamma \frac{f(z)}{(z-\lambda)^{k-1}}dz \\ & \ddots & \ddots & \vdots \\ & & \ddots & \frac{1}{2\pi i}\int_\Gamma \frac{f(z)}{(z-\lambda)^2}dz \\ & & & \frac{1}{2\pi i}\int_\Gamma \frac{f(z)}{z-\lambda}dz \end{bmatrix}$$

$$= \begin{bmatrix} f(\lambda) & f'(\lambda) & \cdots & \frac{f^{(k-1)}}{(k-1)!} \\ & \ddots & \ddots & \vdots \\ & & \ddots & f'(\lambda) \\ & & & f(\lambda) \end{bmatrix}$$

The last equality follows from the differentiation version of Cauchy's integral formula,

$$\frac{1}{k!}f^{(k)}(\lambda) = \frac{1}{2\pi i}\int_\Gamma \frac{f(z)}{(z-\lambda)^k}dz, \quad k = 0, 1, 2, \ldots. \qquad \square$$

Cauchy's integral formula (2.5) can be considered an alternative definition of matrix functions. It is surprisingly general, as it works also for infinite-dimensional operators that do not have a spectrum in the classical sense, and indeed it is popular in operator theory.

Corollary 2.12. *If f is holomorphic, then $f(A)$ is continuous in A, i.e., $A_n \to A$ implies $f(A_n) \to f(A)$.*

Proof. This follows from classical results that you have seen in your analysis courses, which we can also apply to matrix-valued integrals element-by-element. The integrand $f(z)(zI-A)^{-1}$ is a continuous function on $\mathbb{C}\setminus\Lambda(A)$. We can find a compact set K such that $\Gamma \subset K \subset \mathbb{C}\setminus\Lambda(A)$. Then, $\max_{z\in K}\|(A-zI)^{-1}\| = M < \infty$. By continuity and compactness, also $\max_{z\in K}\|(A_n-zI)^{-1}\| < M+\varepsilon$ from a certain n onwards. This uniform boundedness property is sufficient to conclude that

$$\int_K f(z)(zI-A_n)^{-1}dz \to \int_K f(z)(zI-A)^{-1}dz.$$

□

One can prove that $f(A)$ is continuous in A also in more general settings, e.g., for a matrix A with real eigenvalues and f that has enough continuous real derivatives. The proof is more complicated. Here is a sketch:

1. Given a sequence $A_n \to A$, take for each n an interpolating polynomial $p_n(x)$ of f in the spectrum of A_n.

2. The coefficients of these interpolating polynomials $p_n(x)$ converge to those of an interpolating polynomial $p(x)$ for A. This result is not obvious from our proof with a Vandermonde-like matrix, but it follows from other techniques to construct interpolating polynomials, such as divided differences.

3. Now we can write $\|f(A)-f(A_n)\| = \|p_n(A_n)-p(A)\| \leq \|p_n(A_n)-p_n(A)\| + \|p_n(A)-p(A)\|$, and both terms are bounded.

Chapter 3

Sensitivity of matrix functions

Fréchet derivative The Fréchet derivative is an "operator version" of the Jacobian. It is essentially what you would call "the differential" in geometry.

Definition 3.1. The *Fréchet derivative* of a matrix function f in $A \in \mathbb{C}^{n\times n}$ is the *linear* operator $L_{f,A} : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$ (when it exists) such that

$$f(A+E) = f(A) + L_{f,A}(E) + o(\|E\|).$$

Example $f(x) = x^2$, $f(A) = A^2$.

$$(A+E)^2 = A^2 + AE + EA + E^2 = A^2 + \underbrace{AE + EA}_{L_{f,A}(E)} + \underbrace{E^2}_{o(\|E\|)}.$$

$L_{f,A}$ is a linear operator that maps matrices to matrices; we can write explicitly the matrix associated to it, using vectorization. We look for the $n^2 \times n^2$ matrix such that

$$K \operatorname{vec} E = \operatorname{vec} L_{f,A}(E) = AE + EA.$$

By the properties of Kronecker products, it is easy to see that

$$K = A^\top \otimes I + I \otimes A.$$

(This is the result that we have seen when studying Sylvester equations, just with $B = -A$.)

The $n^2 \times n^2$ matrix K is the Jacobian matrix of the map $\operatorname{vec} A \mapsto \operatorname{vec} f(A)$. So this concept of derivative is nothing new with respect to the one you have studied in multivariate analysis, just a vectorized version of it.

A more complex example Let us compute directly the derivative of the function $f(x) = x^{-1}$, $f(A) = A^{-1}$.

$$\begin{aligned}(A+E)^{-1} &= \left((I+EA^{-1})X\right)^{-1} \\ &= A^{-1} \underbrace{-A^{-1}EA^{-1}}_{L_{f,A}(E)} + \underbrace{A^{-1}EA^{-1}EA^{-1} - A^{-1}EA^{-1}EA^{-1}EA^{-1} + \dots}_{o(\|E\|)}\end{aligned}$$

$$K = -A^{-T} \otimes A^{-1}.$$

Properties Follow from those of Jacobians:

- $L_{f+g,A} = L_{f,A} + L_{g,A}$.
- $L_{f\circ g,A} = L_{f,g(A)} \circ L_{g,A}$.
- $L_{f^{-1},A} = L^{-1}_{f,f^{-1}(A)}$.

Example We wish to compute the Fréchet derivative of the matrix square root, using the fact that it is the inverse function of $f(x) = x^2$. Let $f^{-1} : U \to \mathbb{C}$ be a branch of the square root, defined on a neighborhood of $\Lambda(A)$, and $A^{1/2} = f^{-1}(A)$ be the matrix square root of A.

Then, $X = L_{f^{-1},A}(E)$ is the matrix such that $L_{f,A^{1/2}}(X) = E$, i.e.,

$$A^{1/2}X + XA^{1/2} = E.$$

Hence, for any $B \in \mathbb{C}^{n\times n}$, the derivative of the principal square root $L_{f^{-1},A}(E)$ is the solution of the Sylvester equation

$$A^{1/2}X + XA^{1/2} = E$$

Note that this Sylvester equation is always solvable: indeed, the solvability condition is that $\Lambda(A^{1/2}) \cap \Lambda(-A^{1/2}) = \emptyset$, i.e., that there is no $\mu \in \mathbb{C}$ such that μ and $-\mu$ are both eigenvalues of $A^{1/2}$. These must come from an eigenvalue $\lambda = \mu^2 \in \Lambda(A)$. However, when we define f^{-1}, we have to define $f^{-1}(\mu^2)$ to be either μ or $-\mu$; these values cannot both appear in the image of f^{-1}.

Trick to compute $L_{f,A}(E)$ The following formula is a block version of $f(\left[\begin{smallmatrix}\lambda & 1\\ 0 & \lambda\end{smallmatrix}\right]) = \left[\begin{smallmatrix}f(\lambda) & f'(\lambda)\\ 0 & f(\lambda)\end{smallmatrix}\right]$, and lets us evaluate $L_{f,A}(E)$ "as fast as" $f(A)$ for $A \in \mathbb{C}^{2n\times 2n}$.

Theorem 3.2. *Let matrices $A, E \in \mathbb{C}^{n\times n}$ and a function f Fréchet differentiable in A be given. Then, (when these functions are well-defined)*

$$f\left(\begin{bmatrix}A & E\\ 0 & A\end{bmatrix}\right) = \begin{bmatrix}f(A) & L_{f,A}(E)\\ 0 & f(A)\end{bmatrix}. \tag{3.1}$$

Proof. Set $\varepsilon > 0$. We wish to evaluate $f\left(\begin{bmatrix} A+\varepsilon E & E \\ 0 & A \end{bmatrix}\right)$ by block-diagonalizing.

Take a solution of the Sylvester equation $(A+\varepsilon E)Z - ZA = -E$. Then,

$$\begin{bmatrix} I & -Z \\ 0 & I \end{bmatrix} \begin{bmatrix} A+\varepsilon E & E \\ 0 & A \end{bmatrix} \begin{bmatrix} I & Z \\ 0 & I \end{bmatrix} = \begin{bmatrix} A+\varepsilon E & 0 \\ 0 & A \end{bmatrix}.$$

By direct verification, one sees that $Z = -\frac{1}{\varepsilon} I$ is a solution. Note that we do not need to prove that the Sylvester equation has a unique solution; even if it were not unique, taking that particular solution Z always works for block-diagonalization.

Hence

$$\begin{aligned} f\left(\begin{bmatrix} A+\varepsilon E & E \\ 0 & A \end{bmatrix}\right) &= f\left(\begin{bmatrix} I & Z \\ 0 & I \end{bmatrix} \begin{bmatrix} A+\varepsilon E & 0 \\ 0 & A \end{bmatrix} \begin{bmatrix} I & -Z \\ 0 & I \end{bmatrix}\right) \\ &= \begin{bmatrix} I & Z \\ 0 & I \end{bmatrix} \begin{bmatrix} f(A+\varepsilon E) & 0 \\ 0 & f(A) \end{bmatrix} \begin{bmatrix} I & -Z \\ 0 & I \end{bmatrix} \qquad (3.2) \\ &= \begin{bmatrix} f(A+\varepsilon E) & \frac{f(A+\varepsilon E)-f(A)}{\varepsilon} \\ 0 & f(A) \end{bmatrix}. \end{aligned}$$

By the definition of Fréchet derivative,

$$f(A+\varepsilon E) = f(A) + \varepsilon L_{f,A}(E) + o(\varepsilon\|E\|)$$

when $\varepsilon \to 0$, hence the $(1,2)$ block converges to $L_{f,A}(E)$ when $\varepsilon \to 0$. $\square$

Exercise 3.3. Can you find an example of A, E such that the Sylvester equation $(A+\varepsilon E)Z - ZA = -E$ is singular for each $\varepsilon > 0$?

We can use the computation in this proof to give a sufficient condition for the Fréchet derivative to exist.

Theorem 3.4. *Let $A \in \mathbb{C}^{n\times n}$, with eigenvalues λ_i and algebraic multiplicities m_i. If f is of class $\mathcal{C}^{2m_i-1}$ in (a neighborhood of) each of the λ_i, then $f(A)$ is Fréchet differentiable in A.*

Note that each matrix $\tilde{A}$ inside a sufficiently small neighborhood $\mathcal{B}(A,\varepsilon)$ of A has Jordan block sizes in each λ_i of size at most m_i, and $\widehat{A} = \left[\begin{smallmatrix} \tilde{A} & E \\ 0 & \tilde{A} \end{smallmatrix}\right]$ has Jordan block sizes at most $2m_i$. So $f(\widehat{A})$ exists and is continuous in $\tilde{A} = A$. Hence, by the computation (3.2) in the above proof, we see that all directional derivatives exist and are continuous in A. By a standard result in multivariate analysis (in Italy it is called "the total differential theorem"), then f is differentiable.

Fréchet derivative and condition number Recall: the absolute condition number of a differentiable multivariable function $f : \mathbb{R}^m \to \mathbb{R}^n$ is the norm of its Jacobian. Indeed, $f(\tilde{x}) = f(x+h) = f(x) + \nabla_x f \cdot h + o(h)$ implies

$$\kappa_{abs}(f,x) = \lim_{\varepsilon\to 0} \sup_{\|\tilde{x}-x\|\leq\varepsilon} \frac{\|f(\tilde{x})-f(x)\|}{\|\tilde{x}-x\|} = \|\nabla f\|$$

$$\kappa_{rel}(f,x) = \lim_{\varepsilon\to 0} \sup_{\frac{\|\tilde{x}-x\|}{\|x\|}\leq\varepsilon} \frac{\frac{\|f(\tilde{x})-f(x)\|}{\|f(x)\|}}{\frac{\|\tilde{x}-x\|}{\|x\|}} = \kappa_{abs}(f,x)\frac{\|x\|}{\|f(x)\|}.$$

The same result applies to matrix functions: $\kappa_{abs}(f,A) = \|L_{f,A}\|$; however, we need some attention to which norms can be used here.

If the norm used for $\|\widetilde{A}-A\|$ is any matrix norm on $n\times n$ matrices, we must measure $\|L_{f,A}\|$ with the operator norm (on $n^2\times n^2$ matrices) induced by it.

Easy case If we take $\|\widetilde{A}-A\|_F$, it corresponds to $\|\operatorname{vec} A\|_2$, so $\kappa_{abs}(f,A) = \|K_{f,A}\|_2$.

Harder cases For all other norms ($\|\widetilde{A}-A\|_2$ in particular), there is no simple expression for the induced operator norm.

Even with the "easy norm", computing $\|K_{f,A}\|_2$ isn't immediate, because it is a huge $n^2\times n^2$ matrix. In [Higham book, Ch. 3], there are methods based on applying the power method, relying on the fact that we can compute the action $K_{f,A}(E)$ without forming a $n^2\times n^2$ matrix. This is not trivial, because in the power method for the norm we need also to compute the action of its transpose conjugate $K^*_{f,A}$.

The *eigenvalues* $\Lambda(K_{f,A})$ are simpler to compute, and can give us at least some partial insight on when these derivatives are large.

Eigenvalues of Fréchet derivatives [Higham book '08, Ch. 3]

Theorem 3.5. *Let $A\in\mathbb{C}^{n\times n}$ have eigenvalues $\lambda_1,\dots,\lambda_n$ (with their algebraic multiplicity). Then, the n^2 eigenvalues of $L_{f,A}$ (with multiplicity) are*

$$f[\lambda_i,\lambda_j] := \begin{cases} \frac{f(\lambda_i)-f(\lambda_j)}{\lambda_i-\lambda_j} & \lambda_i\neq\lambda_j, \\ f'(\lambda_i) & \lambda_i=\lambda_j. \end{cases}$$

for all $i,j=1,2,\dots,n$.

Proof. First of all, replace $f(x)$ with a polynomial that interpolates A with sufficiently high multiplicities, so that for each E

$$f\left(\begin{bmatrix} A & E\\ 0 & A\end{bmatrix}\right) = p\left(\begin{bmatrix} A & E\\ 0 & A\end{bmatrix}\right)$$

and hence $L_{f,A}(E) = L_{p,A}(E)$.

$$\begin{aligned} p(A+E) &= c_0 + (A+E) + c_1(A+E)^2 + c_2(A+E)^3 + \dots \\ &= c_0 + c_1(A+E) + c_2(A^2+EA+AE+E^2) + c_3(A^3+\dots) \\ &= p(A) + c_1E + c_2(EA+AE) + c_3(A^2E+AEA+A^2E) \\ &\quad + \dots + O(\|E\|^2) \end{aligned}$$

Vectorizing,

$$K_{f,A} = c_1 I + c_2(I \otimes A + A^\top \otimes I) + c_3(I \otimes A^2 + A^\top \otimes A + (A^2)^\top \otimes I) + \dots$$

i.e.,

$$K_{f,A} = \sum_{k=1}^{d} c_k \sum_{h=1}^{k} (A^{k-h})^\top \otimes A^{h-1}$$

Now take Schur forms $X = U_1 T_1 U_1^*$, $X^T = U_2 T_2 U_2^*$.

$$K_{f,X} = (U_2 \otimes U_1) \underbrace{\left(\sum_{k=0}^{d} p_k \sum_{h=1}^{k} T_2^{k-h} \otimes T_1^{h-1} \right)}_{:=T} (U_2 \otimes U_1)^*.$$

This is a Schur decomposition (unitary-triangular-unitary), so we can read off the eigenvalues on the diagonal: if $i \neq j$ we have

$$\begin{aligned} T_{i+n(j-1),i+n(j-1)} &= \sum_{k=0}^{d} p_k \left(\sum_{h=1}^{k} \lambda_i^{k-h} \lambda_j^{h-1} \right) = \sum_{k=0}^{d} p_k \frac{\lambda_i^k - \lambda_j^k}{\lambda_i - \lambda_j} \\ &= \frac{p(\lambda_i) - p(\lambda_j)}{\lambda_i - \lambda_j} = \frac{f(\lambda_i) - f(\lambda_j)}{\lambda_i - \lambda_j}. \end{aligned}$$

Similarly, if $i = j$ we get $f'(\lambda_i)$. □

Unfortunately, the same trick don't work to compute the SVD, because A and its powers do not have the same singular vectors and values. There is no simple formula for the singular values of K.

Condition number bound If A is diagonalizable, we can replace the Schur form with an eigendecomposition, and obtain a bound.

Proposition 3.6. *Let $A = V \Lambda V^{-1}$ be diagonalizable. Then, for the Frobenius norm,*

$$\kappa_{abs}(f, A) = \|K_{f,A}\| \leq \kappa_2(V)^2 \max_{i,j} |f[\lambda_i, \lambda_j]|.$$

Then, as usual, $\kappa_{rel}(f, A) = \kappa_{abs}(f, A) \frac{\|A\|}{\|f(A)\|}$.

This bound displays two possible causes of ill-conditioning:

- $|f[\lambda_i, \lambda_j]|$ is large, or
- $\kappa_2(V)$ is large, i.e., A is very non-normal.

Example Let $f(x) = \sqrt{x}$ (principal square root): for which choices of $\Lambda(A)$ do we encounter a large $|f[\lambda_i, \lambda_j]|$?

- λ_i's close to 0, and
- Pairs of close-by eigenvalues on opposite sides of the branch cut (negative real axis).

More generally, a large $f[\lambda_i, \lambda_j]$ may come from

- Large f',
- pairs of eigenvalues close to a discontinuity in f.

Chapter 4

Computational methods for general matrix functions

Matrix functions arise in several areas: $\exp(A)$ when solving ODEs, $A^{1/2}$ in matrix analysis, many "special" functions in physics, etc.

Our next topic: *how to compute them*, in practice on a computer? We start from methods for matrix functions in general, not restricting to specific choices of f. [Higham book, Ch. 4]

4.1 Diagonalization vs. Taylor series: between Scylla and Charybdis

Diagonalization If A diagonalizable, then we can simply write

$$f(A) = f(V\Lambda V^{-1}) = Vf(\Lambda)V^{-1} = V \begin{bmatrix} f(\lambda_1) & & \\ & \ddots & \\ & & f(\lambda_n) \end{bmatrix} V^{-1}.$$

If A is symmetric, Hermitian, or in general normal, then V can be taken unitary, and this method is perfectly stable: one can show that this computation is backward stable, provided that the scalar values $f(\lambda_i)$ are computed using a backward stable method. This is a fine strategy, and if we only cared about normal matrices, this chapter could stop here.

However, for non-normal matrices, trouble may arise due to the ill-conditioning of the matrix V. Even if we ignore the error in the eigendecomposition, the approximate computation of the diagonal values $|f(\lambda_i) - \tilde{f}(\lambda_i)| < \varepsilon$, causes an error

$$\|f(A) - \tilde{f}(A)\| = \|V(f(\Lambda) - \tilde{f}(\Lambda))V^{-1}\| \leq \kappa(V)\varepsilon,$$

so we may expect numerical issues, especially if A is non-diagonalizable (again!) or close to it.

Matlab example We take a matrix A that is very close to the non-diagonalizable matrix: $\left[\begin{smallmatrix} 3 & -1 \\ 1 & 1 \end{smallmatrix}\right]$, which has a Jordan block of size 2 with eigenvalue $\lambda = 2$.

```
>> A = [3 -1; 1 1+1e-15]
A =
    3.0000 -1.0000
    1.0000 1.0000
>> [V, D] = eig(A);
>> cond(V)
ans =
   6.7109e+07
>> X = V * sqrt(D) / V;
>> norm(X^2 - A)
ans =
   1.5500e-08
```

The fact that X^2 is not close to A shows that we have incurred in an error of order $\mathcal{O}(\sqrt{\mathsf{u}})$.

For this matrix, we can do much better with a Taylor series, instead. We already know that the eigenvalues of this matrix are close to 2, hence $\alpha = 2$ is a good choice for the centre of the Taylor expansion.

$$f(x) = f(2)+f'(2)(x-2)+O((x-2)^2) \implies \sqrt{x} = \sqrt{2}+\frac{1}{2\sqrt{2}}(x-2)+O((x-2)^2).$$

```
>> Y = 1/sqrt(8) * A + 1/sqrt(2) * eye(2);
>> norm(Y^2 - A)
ans =
   4.4409e-16
```

In this case, we obtain

```
>> alpha = trace(A) / 2;
>> I = eye(2);
>> Z = alpha^(1/2)*I + 1/2*alpha^(-1/2)*(A-alpha*I) ...
   - 1/4*alpha^(-3/2)*(A-alpha*I)^2
>> norm(Z^2 - A)
ans =
   4.4409e-16
```

In the case of our matrix, the series converges spectacularly fast because A is approximately a Jordan block and hence $(A - 2I)^2 \approx 0$.

Taylor series (and variants, such as rational approximants $f(x) = \frac{p(x)}{q(x)} + \mathcal{O}(x^{\deg p+\deg q+1})$, which we will see in more detail) work best when the eigenvalues of A are in a small region. But one can come up also with examples where diagonalization fares much better than Taylor expansion.

Drawbacks of Taylor series *Example*: Let us use Taylor series to compute the following matrix exponential, whose result is simple to determine exactly

$$A = \begin{bmatrix} 0 & 30 \\ -30 & 0 \end{bmatrix}, \quad \exp(A) = \begin{bmatrix} \cos 30 & \sin 30 \\ -\sin 30 & \cos 30 \end{bmatrix}$$

(note that these are 30 radiants!). The matrix A is a multiple of an orthogonal matrix, hence it is normal: its eigenvector matrix V is orthogonal. Thus computing $\exp(A)$ by diagonalization works very well.

```
>> A = [0 30; -30 0];
>> [V, D] = eig(A);
>> X = V * [exp(D(1,1)) 0; 0 exp(D(2,2))] / V
X =
   0.1543 -0.9880
   0.9880 0.1543
>> norm(X - expm(A))
ans =
       0
```

Comparing with Matlab's built-in `expm`, we get the exact same result. (Attention: in Matlab `exp(A)` computes the elementwise exponential; do not mix up those two Matlab functions!)

The eigenvalues of A are $\pm 30i$, hence a natural choice for the centre of the Taylor expansion is their average 0. (In general, the average of the eigenvalues is always available without diagonalizing the matrix: $\frac{1}{n}\operatorname{Trace}(A)$.)

We can write a simple function to compute the matrix exponential with its Taylor series.

```
function X = exptaylor(A, d)
% matrix exponential via Taylor series truncated to degree d

X = eye(2); % partial sums
T = eye(2); % 1/k! A^k, updated at each step
for k = 1:d
   T = 1/k * A * T;
   X = X + T;
end
```

We can check that our implementaion is correct by comparing it to Matlab's `expm` for some simple matrices. However, the results on the matrix A above are very poor.

```
A = [0 30; -30 0];
exact_exp = [cos(30) sin(30); -sin(30) cos(30)];
X = eye(2);
T = eye(2);
results = table();
for k = 1:120
```

```
    T = 1/k * A * T;
    X = X + T;
    results.term_norm(k) = norm(T);
    results.error(k) = norm(X - exact_exp);
end
semilogy(results.error)
```

We see that the method needs about 90 terms of the series to converge; this is already bad news, because we need a large number of terms for a fairly simple problem. More importantly, though, when the method converges, it computes a matrix X such that $\|X - \exp(A)\| \approx 10^{-5}$, an abysmally large error for the standards of numerical linear algebra.

The reason for this bad accuracy is the intermediate growth in the summands: if we plot

```
semilogy(results.term_norm)
```

we see that the terms have a huge intermediate growth, reaching $\|T\| \approx 10^{11}$ for $k \approx 30$. A clear "hump" is visible on the plot.

This plot explains perfectly the source of the large error that we observe: we need to compute the sum of intermediate values of magnitude $\approx 10^{11}$, which cancel out to compute a final result of magnitude ≈ 1. The largest terms are computed with an error $\approx 10^{11}\mathsf{u} \approx 10^{-5}$, and this is precisely the error that we observe in the final result.

You may have seen in your undergraduate numerical analysis courses a similar example when computing the scalar exponential $\exp(-30)$ with its Taylor series. In that case, we can avoid the issue by switching to the alternative formula $\exp(-30) = 1/\exp(30)$; but in this matrix case there is not an equally simple fix.

For a normal matrix A, we can at least estimate the growth in coefficients: thanks to $A = UDU^*$, with U unitary, one sees that $\|A\|_2^k = \|A\|_2^k = \rho(A)^k$ for each k. So computing $\exp(A)$ with its Taylor series is as accurate as computing $\exp(\|A\|)$ with its scalar version.

For non-normal matrices, however, the intermediate growth in coefficients can be worse. Even on a nilpotent matrix, the entries may become arbitrarily large:

$$A = \begin{bmatrix} 0 & 10 & & \\ & 0 & 10 & \\ & & 0 & 10 \\ & & & 0 \end{bmatrix}, \; A^2 = \begin{bmatrix} 0 & 0 & 100 & \\ & 0 & 0 & 100 \\ & & 0 & 0 \\ & & & 0 \end{bmatrix}, \; A^3 = \begin{bmatrix} 0 & 0 & 0 & 1000 \\ & 0 & 0 & 0 \\ & & 0 & 0 \\ & & & 0 \end{bmatrix}.$$

To sum up, the main issues with Taylor series are:

- Convergence is poor when the eigenvalues of A are not clustered around a certain α, and the intermediate growth in coefficients may cause loss of accuracy.

- Slow convergence comes with another issue: the cost to evaluate a polynomial of degree d is $O(n^3 d)$, with the Horner rule. There are more efficient methods that reduce the cost to $O(n^3\sqrt{d})$, but still the cost of this method may be more-than-cubic if many terms are required.

4.2 Extras: Polynomial evaluation

How to evaluate polynomials in a matrix argument? Two obvious strategies exist:

- *Direct evaluation*: compute powers of X by successive products, take a linear combination of them).
- *Horner method*: $(\dots((c_d X + c_{d-1})X + c_{d-2})X + \dots)X + c_0 I$

Unlike the scalar case, the two methods are essentially equivalent in terms of cost: $d-1$ matrix products, in both cases.

However, we can obtain a cheaper algorithm if we divide the terms into 'chunks' of size approximately $\sqrt{d}$, e.g., for $d = 8$, we have

$$(c_8A^2 + c_7A + c_6)(A^3)^2 + (c_5A^2 + c_4A + c_3)A^3 + (c_2A^2 + c_1A_1 + c_0).$$

We can compute this quantity with the algorithm

$$B = A^2 \tag{4.1}$$
$$C = AB = A^3 \tag{4.2}$$
$$D = C^2 = A^6 \tag{4.3}$$
$$E = c_2B + c_1A_1 + c_0 \tag{4.4}$$
$$F = c_5B + c_4A + c_3 \tag{4.5}$$
$$G = c_8A^2 + c_7A + c_6 \tag{4.6}$$
$$H = GD + FC + D, \tag{4.7}$$

which requires 5 products rather than 7. More in general, this strategy (the Paterson-Stockmayer method) has complexity $\mathcal{O}(n^3\sqrt{d})$. It requires fewer multiplications, but more storage for the intermediate values. Its stability properties are similar to those of the other two methods. We do not see much in this course, but you can read more on Higham's book.

4.3 Parlett recurrence

What Jordan can do, Schur can do better. Can one compute matrix functions using the Schur form of A?

If $A = UTU^{-1}$, $f(A) = Uf(T)U^{-1}$, so we reduce to the triangular case.

Example

$$T = \begin{bmatrix} t_{11} & t_{12} \\ 0 & t_{22} \end{bmatrix}, \quad f(T) = S = \begin{bmatrix} s_{11} & s_{12} \\ 0 & s_{22} \end{bmatrix}.$$

Clearly, $s_{11} = f(t_{11})$, $s_{22} = f(t_{22})$. But how to compute the remaining term s_{12}?

Trick: expanding $Af(A) = f(A)A$ gives an equation for s_{12}:

$$t_{11}s_{12} + t_{12}s_{22} = s_{11}t_{12} + s_{12}t_{22} \implies s_{12} = t_{12}\frac{s_{11} - s_{22}}{t_{11} - t_{22}}.$$

If $t_{11} = t_{22}$, the equation is not solvable and we already know (at least when $t_{12} = 1$) that the finite difference should be replaced by a derivative.

The same idea works for larger blocks (provided we compute things in the correct order):

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ & t_{22} & t_{23} \\ & & t_{33} \end{bmatrix}, \quad f(T) = S = \begin{bmatrix} s_{11} & s_{12} & s_{13} \\ & s_{22} & s_{23} \\ & & s_{33} \end{bmatrix},$$

$$t_{11}s_{13} + t_{12}s_{23} + t_{13}s_{33} = s_{11}t_{13} + s_{12}t_{23} + s_{13}t_{33}.$$

This formula requires elements from the first subdiagonal s_{12} and s_{23}, which we have already computed. More generally,

$$t_{ii}s_{ij} - s_{ij}t_{jj} = \sum_{i \le k < j} s_{ik}t_{kj} - \sum_{i < k \le j} t_{ik}s_{kj},$$

and the RHS includes only elements from lower subdiagonals which have already been computed.

Hence we can set up a back-substitution very similar to the Bartels–Stewart algorithm.

(In essence, we are solving the (singular) Sylvester equation $TX - XT = 0$ with the back-substitution technique that we have already seen, after computing the diagonal elements by hand to obtain the unique solution with $X_{ii} = f(A_{ii})$.)

To turn this into Matlab code, we need a few more observations. Each term s_{ij} depends only on terms on *below it* on the same column and terms *to its left* on the same row. Hence we can solve *column-by-column* starting from the first rather than superdiagonal-by-superdiagonal; this is easier to write and more efficient.

```
function S = funm_parlett(f, T)
% computes f(T) for upper triangular T
% with the Parlett recurrence
n = size(T, 1);
S = zeros(n, n);
for j = 1:n
    S(j,j) = f(T(j,j));
    for i = j-1:-1:1
        S(i,j) = S(i, i:j-1) * T(i:j-1, j) ...
                - T(i, i+1:j) * S(i+1:j, j);
        S(i,j) = S(i,j) / (T(i,i) - T(j,j));
    end
end
```

Problem: due to the $T(i,i) - T(j,j)$ denominator, this formula becomes very unstable when there are equal, or close-by, eigenvalues.

```
>> T = triu(ones(8) + 1e-5*randn(8));
>> S = funm_parlett(@sqrt, T);
>> norm(S^2 - T)
ans =
   1.7648e+21
```

Solution: the previous formulas also work *blockwise*, and they become Sylvester equations. If we can partition the eigenvalues into *well-separated clusters*, then we can use Taylor expansion on each cluster.

Algorithm (Schur–Parlett method)

1. Compute Schur form $A = QTQ^*$;
2. Reorder T so that it can be partitioned into blocks with 'well-separated eigenvalues' (with a configurable threshold);
3. Compute $f(T_{ii})$ for each block (e.g., with a Taylor series centered in the average of the cluster);
4. Use recurrences to compute off-diagonal blocks of $f(T)$;
5. Return $f(A) = Qf(T)Q^*$.

This algorithm tries to sail between Scylla and Charybdis: we use Taylor expansion when the eigenvalues are close, and recurrences when they are distant. This algorithm is implemented in Matlab's `funm` (at least for some matrix functions).

```
>> T = triu(ones(8) + 1e-5*randn(8));
>> [S, ~, details] = funm(T, @sin);
>> details
  struct with fields:

    terms: 10
      ind: {[1 2 3 4 5 6 7 8]}
      ord: [1 1 1 1 1 1 1 1]
        T: [8x8 double]
>> T = triu(randn(10));
>> [S, ~, details] = funm(T, @sin);
>> details
  struct with fields:
    terms: [1 1 8 1 1 1 1 1]
      ind: {8x1 cell}
      ord: [6 8 6 7 5 4 3 2 1 6]
        T: [10x10 double]
```

Problems with Schur–Parlett This method is more or less the state-of-the-art method for generic functions, and performs well in most cases, but it is not free of problems.

- What happens if the eigenvalues are not naturally divided into clusters? E.g., a single big cluster, or a long line of very close eigenvalues. Applying a degree-d Taylor polynomial to a $k \times k$ block (with k up to n) costs $O(n^3 d)$ (or $O(n^3\sqrt{d})$ with better algorithms), so if $d \sim n$ the cost is more than cubic.
- The correct metric to use to predict accuracy is not the difference between eigenvalues, but $\operatorname{sep}(T_{ii}, T_{jj})$, which is more complicated to handle.
- Derivatives must be known (or computable). Note that the Matlab implementation of `funm` cheats: it has hard-coded derivatives for a small number of standard functions like `@exp` or `@sin`, and for all other functions one must provide explicit derivatives. (Indeed, computing derivatives automatically is more complicated.)

Unfortunately, not much can be said in general on the stability of this method. For sure, the method is not backwards stable, because of the same issues that make Sylvester equations not (structurally) backward stable.

Extras: Parlett recurrence and block diagonalization The Parlett recurrence is very similar to computation via *block diagonalization*.

Consider the case of 2 blocks for simplicity. T can be block-diagonalized via

$$W^{-1}TW = \begin{bmatrix} I & -X \\ 0 & I \end{bmatrix} \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} I & X \\ 0 & I \end{bmatrix} = \begin{bmatrix} T_{11} & \\ & T_{22} \end{bmatrix}$$

where X solves $T_{11}X - XT_{22} + T_{12} = 0$ (Sylvester equation). Then

$$f(T) = W \begin{bmatrix} f(T_{11}) & \\ & f(T_{22}) \end{bmatrix} W^{-1} = \begin{bmatrix} f(T_{11}) & Xf(T_{22}) - f(T_{11})X \\ & f(T_{22}) \end{bmatrix}.$$

(Note indeed that $S = Xf(T_{22}) - f(T_{11})X$ solves the Sylvester equation appearing in the Parlett recurrence.)

So both methods solve a Sylvester equation with operator $Z \mapsto T_{11}Z - ZT_{22}$ and separation $\operatorname{sep}(T_{11}, T_{22})$.

Exercise 4.1 (American Mathematical Monthly, Problem 12451)**.** Show that

$$\exp\left(\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix}\right) = \begin{bmatrix} \exp(A) & \int_0^1 \exp(tA)\, dt \cdot B \\ 0 & I \end{bmatrix}.$$

Chapter 5

Intermezzo: Automatic differentiation

In this chapter, we say some words on *automatic differentiation*: a method that allows one to compute accurately derivatives of arbitrary functions on a computer. This is a topic that has become very popular in recent years, because of interest in machine learning and because of better support from many programming languages (Julia in the first place).

Problem
Given the code for `function y = f(x)` to compute a function $f : \mathbb{R} \to \mathbb{R}$, how does one compute (or approximate) $f'(x)$ in a given point?

```
function y = f(x)
z = x * x;
w = x + 5;
y = z * w;
```

We can assume that only basic arithmetic operations and standard functions such as $\exp, \log, \sin, \sqrt{\cdot}, \dots$ are used in the code.

5.1 Numerical differentiation

First attempt: *numerical differentiation*: compute $g = \frac{f(x+h)-f(x)}{h}$, with a fixed $h > 0$.

Problem: Two sources of error:

- analytical error: Using a Taylor expansion, we can see that $g - f'(x) = \frac{1}{2}f''(\xi)h$ for a point ξ between x and $x+h$.
- numerical error: because of machine arithmetic, even with perfect code we can compute only $f(x)(1+\delta_1)$ and $f(x+h)(1+\delta_2)$ with $|\delta_i| < \mathbf{u}$. So

the computed value $\tilde{g}$ of $g = \frac{f(x+h)-f(x)}{h}$ is affected by an error that we can bound with $\mathbf{u}\frac{|f(x)|+|f(x+h)|}{h}$

So we can bound the total error as

$$|\tilde{g} - f'(x)| \leq |\tfrac{1}{2}f''(\xi)|h + \mathbf{u}\frac{|f(x)|+|f(x+h)|}{h}.$$

Assuming $|\frac{1}{2}f''(\xi)|, |f(x)|, |f(x+h)| = \mathcal{O}(1)$, this error bound is minimized when $h \approx \mathbf{u}^{1/2}$ and is $\mathcal{O}(\mathbf{u}^{1/2})$.

Hence when computing derivatives numerically with the *forward difference*

$$f'(x) \approx \frac{f(x+h)-f(x)}{h},$$

the best accuracy is attained when $h \approx \mathbf{u}^{1/2}$, and the error is $\mathcal{O}(\mathbf{u}^{1/2})$. Importantly, this means that this method is not able to compute derivatives to full precision.

We can verify this also in Matlab.

```
>> x = 5
x =
    5
>> h = 1e-4; g = (f(x+h) - f(x)) / h
g =
    1.250020000097152e+02
% error ≈ 10^-4
>> h = 1e-8; g = (f(x+h) - f(x)) / h
g =
    1.250000025265763e+02
% error ≈ 10^-8
>> h = 1e-12; g = (f(x+h) - f(x)) / h
g =
    1.249986780749168e+02
% error ≈ 10^-4
```

Exercise 5.1. Prove that an analogous estimate holds for the *centered difference*

$$f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}.$$

In this case, the minimum error is $\mathcal{O}(u^{2/3})$, and it is attained for $h = \mathcal{O}(u^{1/3})$.

Extras: complex step differentiation A similar trick: if f is the restriction to $\mathbb{R}$ of a holomorphic function, and our code to compute it works also for complex inputs, then for $x \in \mathbb{R}$ one can write

$$f(x+ih) = f(x) + f'(x)ih - \frac{f''(x)}{2}h^2 + O(h^3),$$

so $g = \frac{\operatorname{Im} f(x+ih)}{h}$ is an approximation of the derivative $f'(x)$ with error $g - f'(x) = O(h^2)$.

Typically, the numerical error on $\operatorname{Im} f(x+ih)$ is $\sim |\operatorname{Im} f(x+ih)|\mathbf{u} = \mathcal{O}(h)\mathbf{u}$ (but if real/imaginary parts are 'mixed' in computation, it may get as large as $\sim |f(x)|\mathbf{u} = \mathcal{O}(\mathbf{u})$). Hence

$$|\tilde{g} - f'(x)| \leq \mathcal{O}(h^2) + \frac{\mathcal{O}(h)}{h}\mathbf{u}.$$

The total error is $\mathcal{O}(\mathbf{u})$ as long as $h \leq \mathcal{O}(\mathbf{u}^{1/2})$.

```
>> x = 5
x =
     5
>> h = 1e-4; g = imag(f(x+1i*h)) / h
g =
     1.249999999900000e+02
>> h = 1e-8; g = imag(f(x+1i*h)) / h
g =
     1.250000000000000e+02
>> h = 1e-12; g = imag(f(x+1i*h)) / h
g =
     1.250000000000000e+02
```

Key idea
We obtained a better bound by exploiting the fact that our code runs also for a *more general type* (complex numbers).

Still, this method does not solve every problem: it only works if f is the restriction of a holomorphic function, and it is real-valued. For instance, a limitation is that you cannot use this method recursively to compute second derivatives, because the outer instance would like to call the inner instance for $z \neq \mathbb{R}$, while it only works for $z \in \mathbb{R}$.

5.2 Automatic differentiation

Automatic differentiation via matrix functions Suppose our code works also for *matrix* arguments x, which we can achieve in Matlab with minor changes:

```
function y = f(x)
n = size(x, 1);
z = x * x;
w = x + 5*eye(n);
y = z * w;
```

Then,

$$f\left(\begin{bmatrix} \lambda & 1 & \\ & \lambda & 1 \\ & & \lambda \end{bmatrix}\right) = \begin{bmatrix} f(\lambda) & f'(\lambda) & \frac{f''(\lambda)}{2} \\ & f(\lambda) & f'(\lambda) \\ & & f(\lambda) \end{bmatrix}.$$

No "small h" and subtractions are needed this time $\implies$ the derivative can be computed with error $\mathcal{O}(\mathbf{u})$.

Automatic differentiation What we have seen is a form of *automatic differentiation*. It is something fundamentally different from numerical differentiation; it is more similar to *symbolic differentiation* with a computer algebra system, but easier to do algorithmically.

- This strategy can (typically) compute derivatives up to *machine precision* error $\mathcal{O}(\mathbf{u})$, by running the code with *more general types*.
- It can be used also to compute also higher derivatives.
- It works also if your code to compute f includes loops, conditionals, and more complicated functions.

```
function y = somefunction(x)
a = x*x + 1;
z = 2 / a;
while z < 5
  z = z^2;
end
y = exp(z);
```

This function is not continuous at certain decision points: when $z = 5$ at some iteration of the `while`. However, in all other points it is differentiable, and we can compute its derivative with the same method.

```
function y = somefunction(x)
n = size(x, 1);
a = x*x + eye(n);
z = 2 * inv(a);
while z(1,1) < 5
  z = z^2;
end
y = expm(z);
```

A better implementation Actually, we do not need matrices here: all operations are on triangular Toeplitz matrices, i.e., polynomials in

$$E := \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}.$$

So

$$\begin{bmatrix} a & b & c \\ 0 & a & b \\ 0 & 0 & a \end{bmatrix} = aI + bE + cE^2.$$

To compute operations between matrices with this form, we can just treat them as polynomials in an indeterminate E, with the relation $E^3 = 0$; i.e., we can just work in the polynomial algebra $\mathbb{R}[E]/(E^3)$; exactly like the complex numbers are isomorphic to $\mathbb{R}[x]/(x^2+1)$.

This observation suggests another enlightening interpretation: what we are really doing is propagating expansions involving an "infinitesimal" quantity $\varepsilon = E$: instead of the input x, for instance $x = 5$, we start from $5+\varepsilon$, and whenever we compute a quantity we compute alongside it the first n coefficients of its power series expansion in ε ; for instance given code

```
function y = f(x) % input: x=5
z = x * x; % z is 25
w = x + 5; % w is 10
y = z * w; % y is 250
```

we can use it to compute two derivatives ($n = 3$) alongside it:

```
function y = f(x) % input: $\hat{x} = 5+\varepsilon = 5 + 1\varepsilon + 0\varepsilon^2 + \mathcal{O}(\varepsilon^3)$
z = x * x; % $\hat{z} = \hat{x}^2 = (5 + 1\varepsilon + 0\varepsilon^2 + \mathcal{O}(\varepsilon^3))(5 + 1\varepsilon + 0\varepsilon^2 + \mathcal{O}(\varepsilon^3))$
% $= 25 + 10\varepsilon + 1\varepsilon^2 + \mathcal{O}(\varepsilon^3)$
w = x + 5; % $\hat{w} = \hat{x} + 5 = (5 + 1\varepsilon + 0\varepsilon^2 + \mathcal{O}(\varepsilon^3)) + 5$
% $= 10 + 1\varepsilon + 0\varepsilon^2 + \mathcal{O}(\varepsilon^3)$
y = z * w; % $\hat{y} = \hat{z}\hat{w} = (25 + 10\varepsilon + 1\varepsilon^2 + \mathcal{O}(\varepsilon^3))(10 + 1\varepsilon + 0\varepsilon^2 + \mathcal{O}(\varepsilon^3))$
% $y = 250 + 125\varepsilon + 20\varepsilon^2 + \mathcal{O}(\varepsilon^3)$
```

From this Taylor expansion we can read off the first two derivatives of $y = f(x)$ in $x = 5$.

Implementation via OOP We can get the computer to do all this automatically without changing our code, with a technique called *object oriented programming*. Matlab is not the best language in the world for this kind of programming, but it will suffice for our example.

The idea behind object-oriented programming is defining new types, or classes, that represent structured data, and together with them implement new functions / operations to operate on these types (in OOP jargon, these functions are called class methods).

In our case, we shall write a *class* `Taylor` that represents a Taylor expansion truncated to order 3. The data we need to save can be represented by length-3 vector, and we will define how Matlab operates on these vectors when one writes `a + b` or `a * b` with objects of type `Taylor` (*operator overloading*). This is done in Matlab by defining inside the class the methods `plus` and `mtimes` (the `m` stands for matrix, exactly like in `expm`). Indeed, when one writes `a + b`, Matlab converts it internally to the function call `plus(a, b)`.

```
function y = f(x) % input: x = Taylor[5 1 0]
z = x * x; % z = Taylor[5 1 0] * Taylor[5 1 0]
% z = Taylor[25 10 1]
w = x + 5; % w = Taylor[5 1 0] + Taylor[5 0 0]
% w = Taylor[10 1 0]
y = z * w; % y = Taylor[25 10 1] * Taylor[10 1 0]
% y = Taylor[250 125 20]
```

The rules that we need to implement to make our example works are the following:

- `Taylor[a0, a1, a2] + Taylor[b0, b1, b2]` = `Taylor[a0+b0, a1+b1, a2+b2]`
- `Taylor[a0, a1, a2] * Taylor[b0, b1, b2]` = `Taylor[a0*b0, a1*b0+a0*b1, a2*b0+a1*b1+a0*b2]`
- For a constant k, `Taylor[a0, a1, a2] + k` = `Taylor[a0, a1, a2] + Taylor[k, 0, 0]` (we need this rule for the operation `x + 5`).

```
classdef Taylor
   properties
      coeffs %length-3 vector
   end
   methods
      function obj = Taylor(v) %constructor
         obj.coeffs = v;
      end
      function c = plus(a, b)
         if isa(b, 'double'), b = Taylor([b 0 0]); end
         c = Taylor(a.coeffs + b.coeffs);
      end
      function c = mtimes(a, b)
         c = Taylor([a.coeffs(1)*b.coeffs(1), ...
           a.coeffs(1)*b.coeffs(2) + a.coeffs(2)*b.coeffs(1), ...
           a.coeffs(1)*b.coeffs(3) + a.coeffs(2)*b.coeffs(2) + a.coeffs(3)*b.coeffs(1)]);
      end
   end
end
```

Automatic differentiation, generically What if our code contains more complicated operations, such as `a / b`, or `exp(a)`, …?

For any elementary operation $z = f(a, b, \dots)$ that appears in our code, we can write the corresponding rules to compute the derivatives of z from those of $a, b, \dots$:

$$\begin{aligned} z' &= \frac{\partial f}{\partial a} a' + \frac{\partial f}{\partial b} b' + \dots \\ z'' &= \frac{\partial^2 f}{\partial a^2} (a')^2 + \frac{\partial f}{\partial a} a'' + \frac{\partial^2 f}{\partial b^2} (b')^2 + \frac{\partial f}{\partial b} b'' + \dots \\ \vdots \quad & \quad \vdots \end{aligned} \tag{5.1}$$

These formulas get lengthy for higher derivatives.

Global derivatives are computed together with each variable, and they are updated according to local rules for each line of code. Each operation that we use needs to be extended to specify how it acts on derivatives. If we define for our type `Taylor` each operation appearing in our code $(ab, a/b, \exp(a), \dots)$, we can effectively compute derivatives algorithmically.

For instance, if we have a framework to compute only the first derivative, and we wish to extend it to work also with the division between two scalar variables `z = a / b`, we need to implement the rule to compute z' (knowing a, a', b, b') as

$$z' = \frac{1}{b} a' - \frac{a}{b^2} b'.$$

If we wish to include scalar exponentiation `z = exp(a)`, we have to implement

$$z' = \exp(a) a'$$

which follows from the chain rule.

Special case: dual numbers The most common case is when one only needs the first derivative. The algebraic structure that reflects the formalism for this case is the ring of *dual numbers*, $\mathbb{R}[\varepsilon]/(\varepsilon^2)$.

- Each member of the ring can be written as $a + \varepsilon b$, for $a, b \in \mathbb{R}$ (or any other base ring).
- Operations are performed with usual algebraic rules plus $\varepsilon^2 = 0$; for instance, `a * b` when its argument are dual numbers becomes $(a + \varepsilon a')(b + \varepsilon b') = ab + (a'b + ab')\varepsilon$.
- $f'(x)$ is equal to the "epsilon part" of $f(x + \varepsilon)$.

Julia and automatic differentiation the language Julia has a great ecosystem for automatic differentiation. Differentiation rules like (5.1) are defined in the standard libraries for many common library functions (such as `eigvals` to compute eigenvalues), and when writing native Julia code one is encouraged to use idioms that work on multiple types automatically: in the example above, instead of writing `x + 5` above, one would typically use `x + 5*one(x)`, so that the correct ring identity is used depending on the type of `x`. The result is that in most cases it is sufficient to load one of several packages for automatic differentiation to get it to work out of the box:

```
julia> using LinearAlgebra # for eigvals
julia> using Zygote # a popular autodiff package
julia> spectralradius(A) = maximum(abs.(eigvals(A)));
julia> gradient(spectralradius, [1 2; 3 4])
([0.23888351606645317 0.5222329678670934;
  0.3481553119113956 0.7611164839335467],)
```

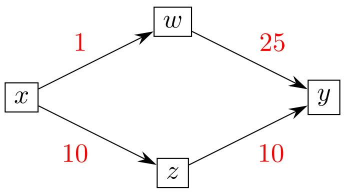
5.3 Reverse mode

The one we have seen in the previous section is called the *forward mode* of automatic differentiation. There is also a *reverse mode* which is more popular in some contexts; most notably machine learning, where it is known as *back-propagation*.

We give an idea of how it works, to compute only the first derivative.

General idea: After having computed $y = f(x)$, we revisit our code line-by-line $\underline{\text{backwards}}$, and for each intermediate variable a we compute $\frac{dy}{da}$, starting from $\frac{dy}{dy} = 1$.

This manipulation requires more complicated transformations to the source code than forward-mode; introducing new types is not sufficient. Typically it is performed in some way via a *computational graph representation* of the operations to perform. We see it at work in our running example. A computational graph for it is



```
function y = f(x) % input: x=5
z = x * x; % $\frac{\partial z}{\partial x} = 2x = 10$
w = x + 5; % $\frac{\partial w}{\partial x} = 1$
y = z * w; % $\frac{\partial y}{\partial w} = z = 25$, $\frac{\partial y}{\partial z} = w = 10$
```

Using these *edge derivatives*, we work our way right-to-left and compute starting from $\frac{dy}{dy} = 1$:

$$\frac{dy}{dw} = \frac{dy}{dy}\frac{\partial y}{\partial w} = 25, \quad \frac{dy}{dz} = \frac{dy}{dy}\frac{\partial y}{\partial z} = 10,$$

$$\frac{dy}{dx} = \frac{dy}{dw}\frac{\partial w}{\partial x} + \frac{dy}{dz}\frac{\partial z}{\partial x} = 25 \cdot 1 + 10 \cdot 10 = 125.$$

The multivariate case The same ideas (both for forward and reverse mode) work also in the multivariate case, when the input x and the output y are vectors. *Jacobian matrices* take the place of scalar derivatives. All products above like $\frac{\partial f}{\partial a} a'$ or $\frac{dy}{dw}\frac{\partial w}{\partial x}$ become matrix multiplications, and we start from $\frac{dx}{dx} = I$ or $\frac{dy}{dy} = I$.

For a function $f : \mathbb{R}^n \to \mathbb{R}^m$:

Forward mode: for each intermediate variable $w \in \mathbb{R}^p$, store its Jacobian $\frac{dw}{dx} \in \mathbb{R}^{p \times n}$. Whenever an instruction computes z from w, multiply on the *left*: $\frac{dz}{dx} = \frac{\partial z}{\partial w}\frac{dw}{dx}$.

Reverse mode: for each intermediate variable $w \in \mathbb{R}^p$, store its Jacobian $\frac{dy}{dw} \in \mathbb{R}^{m \times p}$. During the reverse part, for each z on which w depends, multiply on the *right*: $\frac{dy}{dz} = \frac{dy}{dw}\frac{\partial w}{\partial z}$.

Forward mode can also compute only a directional derivative $\frac{dy}{dv}$ in the direction of a given vector $v \in \mathbb{R}^n$, i.e., the matrix-vector product $\frac{dy}{dx}v$: we just need to change the initial value. Instead of starting from $\frac{dx}{dx} = I$, we start $\frac{dx}{dv} = v$ and then compute for each intermediate variable z the quantity $\frac{dz}{dv}$, using the exact same recursions.

Similarly, reverse mode can be used to compute only $v^T \frac{dy}{dx}$.

Computational complexity Which is cheaper, forward mode or reverse mode? This depends on the dimension of your data.

- For a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n \ll m$ that is the composition of many steps, the *forward mode* is faster because all intermediate Jacobians are tall-thin, with the second dimension equal to n.
- For a function $f : \mathbb{R}^n \to \mathbb{R}^m$ with $n \gg m$ that is the composition of many steps, the *reverse mode* is faster, because all intermediate Jacobians are short-fat, with the first dimension equal to m.

Machine learning is, essentially, fitting functions with a large number of parameters (weights) $\beta \in \mathbb{R}^n$ to given training data, by minimizing a certain scalar error function $E(\beta) \in \mathbb{R}$. Since n is very large (possibly of the order of billions!) and $m = 1$, *reverse mode* is much faster. In modern applications, neural network training is performed via gradient descent, and reverse-mode automatic differentiation (known in the field as back-propagation) is one of the main ingredients.

More applications Another field where automatic differentiation is becoming common is optimization, where it is used to compute Jacobians and sometimes Hessians automatically. Implicit methods for ODEs are another area where one can benefit from knowing the Jacobian of the function $f(t, y)$.

We will not see more in this direction, but these applications are too interesting not to mention automatic differentiation.

Remark: complex autodiff Extending this concept of automatic differentiation to complex functions is not immediate. We could restrict ourselves to holomorphic functions, but this would leave out common operations such as $z \mapsto |z|$. In several applications, we can identify $\mathbb{C}^n$ with $\mathbb{R}^{2n}$ and work with real Jacobians. A starting point to find out something more is the documentation inside the Julia automatic differentiation framework, e.g., `https://fluxml.ai/Zygote.jl/dev/complex/`, `https://juliadiff.org/ChainRulesCore.jl/stable/maths/complex.html`.

Chapter 6

The matrix exponential

We now discuss specialized algorithms for some specific important matrix functions. The first one:

$$\exp(A) = I + A + \frac{1}{2}A^2 + \frac{1}{3!}A^3 + \dots.$$

An important remark: in Matlab, `exp(A)` does entrywise exponentiation; one must use `expm` to compute the matrix function. It is very easy to mix up the two.

Differential equation solution A common application of the matrix exponential is the solution of the ODE initial value problem

$$\frac{d}{dt}v(t) = Av(t), \quad v(0) = v_0, \tag{6.1}$$

which is $v(t) = \exp(At)v_0$. *Proof*: we can differentiate term-by-term

$$v(t) = v_0 + tAv_0 + \frac{t^2}{2}A^2v_0 + \frac{t^3}{3}A^3v_0 + \dots.$$

For this application, often we are concerned with computing $\exp(At)$ for several values of $t \in \mathbb{R}$, or $\exp(A)v_0$ for a certain vector v_0.

Shifting Note that $\exp(A + B) \neq \exp(A)\exp(B)$, in general; this equality holds only if A and B commute. In particular, this holds for $B = \beta I$ is a multiple of the identity:

$$\exp(A + \beta I) = e^{\beta}\exp(A) \tag{6.2}$$

This fact has an interesting consequence.

Theorem 6.1. *Let A be a matrix with non-negative off-diagonal entries, i.e., $A_{ij} \geq 0$ for all $i \neq j$. Then, all entries of $\exp(A)$ are nonnegative, i.e., $\big(\exp(A)\big)_{ij} \geq 0$ for all i, j.*

Proof. For a sufficiently large $\beta > 0$, all the entries of $A + \beta I$ are non-negative. Hence,

$$\exp(A) = e^{-\beta} \exp(A + \beta I) = e^{-\beta} \sum_{k=0}^{\infty} \frac{1}{k!}(A + \beta I)^k \tag{6.3}$$

is a sum of non-negative terms. □

This subtraction-free formulation is useful also numerically: there is no numerical cancellation due to subtraction in the computation of (6.3). Hence algorithms based on this formula are extremely stable. [Shao, Gao, Xue '14]

Matrices that satisfy the hypothesis of this theorem are known as essentially non-negative matrices, or Metzler matrices, or $-Z$-matrices. They appear sometimes in Markov chain applications, together with their exponentials.

Another computational application of (6.2) is so-called argument reduction: sometimes, we can find β such that $\|A + \beta I\|$ is much smaller than $\|A\|$; in this case, it is convenient to "factor out" β and compute $\exp(A)$ via $\exp(A) = e^{-\beta} \exp(A + \beta I)$.

Some infinite series expressions relating $\exp(A), \exp(B), \exp(A+B)$ are used in the physics literature; see [Higham book, Section 10.1] if you are interested.

How to compute $\exp(A)$? It is easy to come up with ways that turn out to be unstable. There is a famous paper that studies this problem, titled Nineteen dubious ways to compute the exponential of a matrix [Moler, Van Loan, '78, with a '03 update]. We have already seen several pitfalls of Taylor series. Some other computational methods for $\exp(A)$ include solving the ODE (6.1), possibly with $v_0 = I$, so that we get the whole matrix $\exp(At)$. In particular, if we use the explicit Euler method we obtain the approximation $\exp(A)v_0 \approx (I + \frac{1}{n}A)^n v_0$.

The main source of the problem is the growth in intermediate results in all these methods. We have already seen one instance of this phenomenon with the Taylor series. We see another.

"Hump" phenomenons Even for a matrix with $\Lambda(A) \subset LHP$, $\exp(tA)$ may grow for small values of t before settling down and converging to 0.

Example [Higham book, Ch. 10]

```
>> A = [-0.97 25; 0 -0.3];
>> t = linspace(0,20,100);
>> for i = 1:length(t); y(i) = norm(expm(t(i)*A)); end
>> plot(t, y)
```

Note that this A is highly non-normal, because of the large off-diagonal entry A_{12}.

This example shows that we can expect intermediate growth and cancellation if we use methods that "go through" other values $\exp(At)$ with $t < 1$, for instance by solving the ODE problem

$$X'(t) = AX(t), \quad X(0) = I.$$

Since the intermediate results are much larger than the final result, there must occur some cancellation at some point, and this reduces accuracy.

Exception: again, everything works well for normal matrices. Since

$$\|\exp(A)\|_2 = \max_{\lambda \in \Lambda(A)} |e^\lambda| = e^{\max \operatorname{Re}(\lambda)},$$

we have $\|\exp(tA)\| = \|\exp(A)\|^t$. So if we plot the norm we see an exponential decrease (or increase) without "humps".

Fréchet derivative of the matrix exponential Using the same argument that we have used to compute the eigenvalues of Fréchet derivatives in Theorem 3.5, we can obtain

$$L_{\exp,A}[E] = E + \frac{1}{2!}(EA + AE) + \frac{1}{3!}(EA^2 + AEA + A^2E) + \dots$$

and the associated $n^2 \times n^2$ matrix

$$K = I_{n^2} + \frac{1}{2}(A^\top \otimes I + I \otimes A) + \frac{1}{3!}((A^2)^\top \otimes I + A^\top \otimes A + I \otimes A^2) + \dots.$$

This time we have a converging series rather than a finite sum as in Theorem 3.5. We can use this series to obtain a bound on $\|K\|$:

$$\begin{aligned}\|K\| &\le \|I_{n^2}\| + \frac{1}{2}\|A^\top \otimes I + I \otimes A\| + \frac{1}{3!}\|(A^2)^\top \otimes I + A^\top \otimes A + I \otimes A^2\| + \dots \\ &\le 1 + \frac{1}{2}2\|A\| + \frac{1}{3!}3\|A\|^2 + \dots \\ &= \sum_{k=0}^{\infty} \frac{1}{(k+1)!}(k+1)\|A\|^k = e^{\|A\|}.\end{aligned}$$

This shows that

$$\kappa_{abs}(\exp, A) = \|L_{\exp,A}\| \le e^{\|A\|}.$$

This is not good news, because we have seen that $e^{\|A\|}$ may be much larger than $\|\exp(A)\|$; indeed, in the worst case the matrix exponential can be quite ill-conditioned.

The picture gets much better if A is normal. In that case, Proposition 3.6 shows that

$$\|L_{\exp,A}\| = \max_{\lambda_i, \lambda_j \in \Lambda(A)} |\exp[\lambda_i, \lambda_j]|.$$

We can use Lagrange's theorem to estimate the magnitude of $|\exp[\lambda_i, \lambda_j]|$:

$$|\exp[\lambda_i, \lambda_j]| = \left| \frac{e^{\lambda_j} - e^{\lambda_i}}{\lambda_j - \lambda_i} \right| = |e^\xi| \le \max_{\lambda \in \Lambda(A)} |e^\lambda| = e^{\max \operatorname{Re}(\lambda)} = \|\exp(A)\|,$$

since for each i, j ξ is in the convex hull of the eigenvalues of A. So we get

$$\kappa_{abs}(\exp, A) \le \|\exp(A)\|,$$

which can be a large improvement over $e^{\|A\|}$. Note that this implies that for the relative condition number $\kappa_{rel}(\exp, A) \leq \|A\|$ (again, only in the normal case!).

A similar discussion based on different arguments is in [Higham book, Section 10.2].

6.1 Scaling and squaring

We now describe the method used in Matlab's `expm`, which is the state-of-the-art one.

Padé approximants *Padé approximants* A Padé approximant of degree (p, q) to a function f (in $x = 0$) is a rational function $\frac{N(x)}{D(x)}$ of degree (p, q) such that

$$f(x) = \frac{N(x)}{D(x)} + \mathcal{O}(x^{p+q+1})$$

when $x \to 0$. This idea generalizes Taylor series ($q = 0$).

We can typically find polynomals $N(x)$ and $D(x)$ that satisfy this equality (we will see how), since $N(x)$ and $D(x)$ have $p + q + 1$ coefficients, once we normalize setting $D(0) = 1$ (to account for common scaling: N and D can be multiplied by the same factor). So we can impose $p + q + 1$ conditions.

We see an example of the computation of a Padé approximant to the matrix exponential of degrees $p = q = 2$, using Matlab's symbolic toolbox.

```
>> syms x a b c d e
>> T = taylor(exp(x), x, 0, 'Order', 5)
T =
x^4/24 + x^3/6 + x^2/2 + x + 1
>> D = a*x^2+b*x+1;
>> N = c*x^2 + d*x + e;
>> collect(expand(T*D-N))
ans =
(a*x^6)/24 + (a/6 + b/24)*x^5 + (a/2 + b/6 + 1/24)*x^4 + (a + b/2 + 1/6)*x^3 + ...
(a + b - c + 1/2)*x^2 + (b - d + 1)*x - e + 1
>> C = coeffs(collect(expand(T*D-N)),x)
C =
[1 - e, b - d + 1, a + b - c + 1/2, a + b/2 + 1/6, a/2 + b/6 + 1/24, a/6 + b/24, a/24]
>> S = solve(C(1:5)==0, [a,b,c,d,e]);
>> [S.a, S.b, S.c, S.d, S.e]
ans =
[1/12, -1/2, 1/12, 1/2, 1]
```

Here we used the fact that

$$f(x) - \frac{N(x)}{D(x)} = \mathcal{O}(x^{p+q+1}) \iff f(x)D(x) - N(x) = \mathcal{O}(x^{p+q+1}),$$

since $D(0) \neq 1$. With this trick we could reduce the computation to the solution of a system of $p+q+1$ linear equations. For most choices of f, p, q, this system of equations has a unique solution, but this is by no means a guarantee: some functions f do not admit a Padé approximant for certain (p, q).

Padé approximants for the exponential exist and are known in closed form:

$$N_{pq}(x) = \sum_{j=0}^{p} \frac{(p+q-j)!p!}{(p+q)!j!(p-j)!} x^j,$$
$$D_{pq}(x) = N_{pq}(-x).$$

When approximating functions, Padé approximants often achieve a smaller error than a Taylor series with the same number of degrees of freedom. We can check numerically that the polynomial function $N(x)/D(x)$ that we have computed approximates e^x with lower error than the Taylor series $T(x)$ with the same order:

```
fplot([T-exp(x), subs(N/D,S)-exp(x)], [-1,1])
```

We can frame this approximation property as a sort of "backward error": for each x, there is a δ such that

$$\frac{N(x)}{D(x)} = e^{x+\delta},$$

i.e., the Padé approximation $N(x)/D(x)$ computes the exact exponential of a perturbed input $x + \delta$.

Some easy manipulations show that

$$\delta(x) = \log\left(e^{-x}\frac{N(x)}{D(x)}\right).$$

We have

$$e^x - \frac{N(x)}{D(x)} = \mathcal{O}(x^{p+q+1}) \implies 1 - e^{-x}\frac{N(x)}{D(x)} = \mathcal{O}(x^{p+q+1}) \implies \log\left(e^{-x}\frac{N(x)}{D(x)}\right) = \mathcal{O}(x^{p+q+1}),$$

so the function $\delta(x)$ is very flat around zero: it grows as x^{p+q+1}.

```
>> delta = log(exp(-x) * subs(N/D,S));
>> taylor(delta, x, 0, 'Order', 20)
ans =
- x^19/98035826688 - x^17/7309688832 + x^13/38817792 + x^11/2737152 - x^7/12096 - x^5/720
>> fplot(delta, [-1,1])
```

When x gets larger, the quality of this approximation degrades.

Padé approximations of the matrix exponential When A has small norm, we can use a Padé approximation to compute an approximation to $\exp(A)$:

$$\exp(A) \approx (D_{pq}(A))^{-1} N_{pq}(A).$$

Since $D(0) = 1$, if A has a small norm then $D(0) \approx I$ is well-conditioned, so we do not expect the inversion to be troublesome.

Note that the order of the two factors does not matter: since these are polynomials in A, $(D_{pq}(A))^{-1}N_{pq}(A) = N_{pq}(A)(D_{pq}(A))^{-1}$.

Can we give an error bound for this matrix approximation, relying on the scalar one? We define the matrix version of the "backward error" function $\delta(x)$ that we have defined above:

$$\Delta = \delta(A) = \log\left(\exp(-A)D(A)^{-1}N(A)\right).$$

Thanks to the properties of matrix functions, Δ and A commute, and then direct computation shows that

$$D(A)^{-1}N(A) = \exp(A)\exp(\Delta) = \exp(A + \Delta).$$

If we determine that $\|\Delta\|/\|A\|$ is of the order of machine precision, then $D(A)^{-1}N(A)$ is the exact exponential of a perturbed matrix, and the error we make in replacing $\exp(A)$ with $D(A)^{-1}N(A)$ is comparable to the one we make when we represent A with floating-point numbers (which is of the order of $\kappa(\exp, A)\mathtt{u}$). We cannot do better, numerically: this approximation is backward stable.

We have seen that the scalar function $\delta(x)$ is very small, in practice; for instance, our plot showed that for $p = q = 2$ we have $|\delta(x)| < 1.5 \cdot 10^{-8}$ whenever $|x| < 0.1$. But does this imply that the corresponding matrix function $\Delta = \delta(A)$ is small? This is a very interesting question that is related to an open problem we will see later in our course (Crouzeix's conjecture). For now, we only give a worse bound based on the Taylor expansion of $\delta(x)$:

$$\delta(x) = \sum_{k=p+q+1}^{\infty} c_k x^k \implies \|\Delta\| = \left\| \sum_{k=p+q+1}^{\infty} c_k A^k \right\| \leq \sum_{k=p+q+1}^{\infty} |c_k| \|A\|^k.$$

Unfortunately, the coefficients c_k have mixed signs, so the right-hand side is not $\delta(\|A\|)$, but something larger. We can see this also in our running example $p = q = 2$:

```
>> TD = taylor(delta/x, x, 0, 'Order', 20)
TD =
- x^18/98035826688 - x^16/7309688832 + x^12/38817792 ...
+ x^10/2737152 - x^6/12096 - x^4/720
>> c = coeffs(TD, 'all')
c =
[-1/98035826688, 0, -1/7309688832, 0, 0, 0, 1/38817792, 0, ...
1/2737152, 0, 0, 0, -1/12096, 0, -1/720, 0, 0, 0, 0]
>> double(abs(c) * (0.1 .^ (18:-1:0))')
ans =
   1.3897e-07
```

In the last line, we compute

$$\sum_{k=p+q+1}^{\infty} |c_k| 0.1^k < 1.39 \cdot 10^{-7}.$$

Hence this computation shows that $\|\Delta\| < 1.39 \cdot 10^{-7}$ when $\|A\| < 0.1$. This computation is not fully rigorous, because we truncated the series after x^{19}, but both the coefficients c_k and the powers 0.1^k decrease quite sharply, so we should not be too far from the truth. A fully rigorous approach that bounds the tail of this series is in [[Moler, Van Loan '03]].

Researchers obtained similar bounds for higher degrees p and q, for instance the following:

Proposition 6.2 ([Higham book '08, Table 10.2])**.** *If $p = q = 13$ and $\|A\| \leq 5.4$, then $\frac{\|H\|}{\|A\|} \leq \mathbf{u} \approx 2.2 \cdot 10^{-16}$ (machine precision). Moreover, $\|D(A)^{-1}\| \leq 17$.*

The bound on $\|D(A)^{-1}\|$ can be obtained with similar techniques, and ensures that the inversion is well-conditioned.

The degree 13, in particular, was chosen because it achieves a good ratio between accuracy and number of required operations. Thanks to the fact that $D(x) = N(-x)$, one can evaluate both $N_{13,13}$ and $D_{13,13}$ at the same time with only 6 matmuls. (For more details on this evaluation, see the Paterson–Stockmeyer method in Section 4.2.)

Remark 6.3. Padé approximants are the classical approach used in Matlab's `expm`; however, in more recent years, faster techniques to evaluate matrix polynomials have been found, making Taylor expansions more competitive with respect to Padé expansions [Sastre et al, 2009].

Remark 6.4. The same techniques to construct Padé approximants and to evaluate their backward stability can be applied to other functions as well; the exponential is just a nice example. In general, though, rational approximation methods work well only when the eigenvalues are in a sufficiently small region.

6.2 Scaling and squaring

What if $\|A\| > 5.4$? Idea: let us use the identity $\exp(A) = (\exp(\frac{1}{s}A))^s$.

Algorithm (scaling and squaring)

1. Find the smallest $s = 2^k$ such that $\|\frac{1}{s}A\| \leq 5.4$.
2. Compute $F = D_{13,13}(B)^{-1} N_{13,13}(B)$, where $D_{13,13}$ and $N_{13,13}$ are given polynomials and $B = \frac{1}{s}A$.
3. Compute F^{2^k} by repeated squaring.

This algorithm is used in Matlab's `expm`, currently (plus one minor optimization: an approximant of degree smaller than 13 may be used when $\|A\| \leq 5.4$ already holds).

Is scaling and squaring provably stable? No! 'Humps' may still give problems: $\exp(B)$ may be much larger than $\exp(A) = \exp(B)^{2^k}$, leading to cancellation when one computes the squares.

Scaling and squaring does not avoid entirely the intermediate growth problem entirely, but it is still the best algorithm available. And, in the end, our earlier theoretical results show that for large A computing the matrix exponential is an ill-conditioned problem: computing the 'tail' end of $\exp(tA)$ to high precision is impossible.

Chapter 7

The matrix sign function

In this chapter, we show how to compute the matrix function $\operatorname{sign}(M)$ associated to the scalar function

$$\operatorname{sign}(x) = \begin{cases} 1 & \operatorname{Re} x > 0, \\ -1 & \operatorname{Re} x < 0, \\ \text{undefined} & \operatorname{Re} x = 0. \end{cases}$$

We shall see that this matrix function is much more interesting than its scalar counterpart.

We assume in this chapter that the matrix M has no purely imaginary eigenvalues, so that $\operatorname{sign}(M)$ is well-defined.

Suppose the Jordan form of M is reblocked as

$$M = \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} J_1 & \\ & J_2 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^{-1},$$

where J_1 contains the Jordan blocks with eigenvalues in the LHP (left half-plane) and J_2 those in the RHP. Then,

$$\operatorname{sign}(M) = \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} -I & \\ & I \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^{-1}.$$

In particular, $\operatorname{sign}(M)$ is always diagonalizable with eigenvalues $\Lambda(M) \subseteq \{-1, 1\}$.

If M has all its eigenvalues in the LHP, then $\operatorname{sign}(M) = -I$, and analogously if $\Lambda(M) \subset RHP$ then $\operatorname{sign}(M) = I$.

Application: projector on the leftmost part of the spectrum In some physics problem, one must compute a few of the leftmost (in the complex plane) eigenvalues of a given matrix M. These correspond to certain electronic states with lowest energy, and in particular one is interested in the projector on the invariant subspace spanned by them.

A possible way to compute them is the following. Take $\mu \in \mathbb{R}$, and note that the eigenvalues in the LHP of $M - \mu I$ correspond to the eigenvalues of M with real part smaller than μ; in particular, the associated eigenvectors and Jordan blocks are the same.

If one computes $S = \operatorname{sign}(M - \mu I)$, then

$$\operatorname{Im}(S - I) = \operatorname{Im} \begin{bmatrix} V_1 & V_2 \end{bmatrix} \begin{bmatrix} -2I & \\ & 0 \end{bmatrix} \begin{bmatrix} V_1 & V_2 \end{bmatrix}^{-1} = \operatorname{Im} V_1,$$

and $\operatorname{Im} V_1$ is precisely the invariant subspace corresponding to the eigenvalues of M that have real part smaller than μ. Similarly, $\ker(S - I) = \operatorname{Im} V_2$.

Application: computing eigenvalues by bisection We can expand the same idea to compute eigenvalues by bisection.

Given $M \in \mathbb{C}^{n \times n}$, set $C := \alpha M + \beta I$ for suitable $\alpha, \beta \in \mathbb{C}$, and compute $S = \operatorname{sign}(C)$. Then, with the same notation as above, $\operatorname{Im} V_1$ is the invariant subspace corresponding to the half-plane $\mathcal{H} = \{\lambda \colon \alpha\lambda + \beta \in LHP\}$. If we let Q be the orthogonal factor in $\texttt{qr}(S - I)$, then

$$Q^* M Q = \begin{bmatrix} A & * \\ 0 & B \end{bmatrix},$$

where A and B contain the eigenvalues of M inside and outside $\mathcal{H}$ respectively.

This can be seen as the first step of a *bisection procedure* to compute the eigenvalues and eigenvectors of M: we have split the spectrum into two subsets; now we can repeat the procedure on A and B with new values of α, β.

7.1 The Schur-Parlett method

Schur–Parlett method A first algorithm to compute $\operatorname{sign}(M)$ comes from the Schur–Parlett strategy. Compute a Schur decomposition $M = UTU^*$, reordered so that eigenvalues in the LHP come first, to obtain

$$U^* M U = \begin{bmatrix} A & C \\ 0 & B \end{bmatrix}, \quad \Lambda(A) \subset LHP,\ \Lambda(B) \subset RHP.$$

hence

$$f(M) = U f\left(\begin{bmatrix} A & C \\ 0 & B \end{bmatrix}\right) U^* = U \begin{bmatrix} -I & Z \\ 0 & I \end{bmatrix} U^*$$

for a certain Z. We can compute Z by solving the Sylvester equation

$$AZ - ZB = f(A)C - Cf(B) = -2C. \tag{7.1}$$

We can then summarize the Schur–Parlett algorithm for the matrix sign as follows.

1. Compute a Schur decomposition $M = UTU^*$.

2. Reorder it so that eigenvalues in the LHP come first.

3. Compute Z by solving the Sylvester equation (7.1) (which has triangular coefficients).

4. $\operatorname{sign}(M) = U\left[\begin{smallmatrix} -I & Z \\ 0 & I \end{smallmatrix}\right]U^*$.

```
function S = sign_schurparlett(M)
% Computes the matrix sign function with the Schur-Parlett method

n = size(M, 1);
[Q, U] = schur(M, 'complex');
% overwrite Q, U with their reordered version
[Q, U] = ordschur(Q, U, "lhp");

% count the number of eigenvalues in the LHP
p = sum(real(diag(U)) < 0);

A = U(1:p, 1:p);
B = U(p+1:n, p+1:n);
C = U(1:p, p+1:n);

% Matlab function to solve a Sylvester equation
% Note that Matlab's syntax has different signs
Z = lyap(A, -B, 2*C);

S = zeros(n, n);
S(1:p, 1:p) = -eye(p);
S(1:p, p+1:end) = Z;
S(p+1:end, p+1:end) = eye(n-p);

S = Q*S*Q';

% If M is real, S=sign(M) is supposed to be real,
% but the computed one will have a tiny nonzero imaginary part,
% due to complex arithmetic in the Schur form.
% We remove it if that is the case.

if isreal(M)
    S = real(S);
end
```

Note that this algorithm is not useful for our application of computing eigenvalues via bisection, because it requires the Schur form, which reveals the eigenvalues already. Before seeing a different algorithm, we expand on perturbation theory.

7.2 Perturbation theory

The general results that we have obtained for perturbation of functions of matrices show that for a diagonalizable matrix $M = VDV^{-1}$ the condition number of the matrix sign satisfies

$$\kappa_{abs}(\text{sign}, M) \leq \kappa(V)^2 \max_{\substack{\lambda\in\Lambda(M)\cap LHP \\ \mu\in\Lambda(M)\cap RHP}} \frac{2}{|\lambda-\mu|}.$$

This result is slightly misleading, though, because of two reasons:

- The factor $\kappa(V)^2$ may be an overestimate, because if we think in terms of invariant subspaces we see that the difficulty is separating the two invariant subspaces relative to the LHP and RHP, which in general is a more well-conditioned task than a full diagonalization.
- For matrices with small separation, another source of error amplification comes from the fact that $\|\text{sign}(M)\|$ is itself large. Indeed, taking norms in the vectorized form of (7.1) gives

$$\|Z\| \leq \|(I\otimes A - B^T \otimes I)^{-1}\| \|2C\| = \frac{2\|C\|}{\text{sep}(A,B)}.$$

A more accurate result is the following.

Theorem 7.1 ([Byers He Mehrmann '97]). *Let $M = Q\left[\begin{smallmatrix} A & C \\ 0 & B\end{smallmatrix}\right]Q^*$ as above, with $\text{sep}(A,B) = \delta$, and let $\|E\|_F = \varepsilon$ be sufficiently small. Then,*

1. $$\frac{\|\text{sign}(M+E) - \text{sign}(M)\|_F}{\|\text{sign}(M)\|_F} = \mathcal{O}\left(\frac{\epsilon}{\delta^2}\right).$$
2. *However, the Hurwitz stable invariant subspace of $M+E$ is $Q\left[\begin{smallmatrix} I \\ X\end{smallmatrix}\right]$, where $\|X\|_F = \mathcal{O}(\frac{\epsilon}{\delta})$.*

So the sign is highly ill-conditioned in presence of a small separation δ, but the Hurwitz stable invariant subspace that we can compute with it is better conditioned. This is good news for our application.

Proof. Assume $Q = I$, up to a change of basis. Part 2 follows from the perturbation result for invariant subspaces: if $M + E = \left[\begin{smallmatrix} \tilde{A} & \tilde{C} \\ \tilde{D} & \tilde{B}\end{smallmatrix}\right]$, there exists X with $\|X\|_F = \mathcal{O}(\frac{\varepsilon}{\delta})$ such that

$$\begin{bmatrix} I & 0 \\ -X & I \end{bmatrix} (M+E) \begin{bmatrix} I & 0 \\ X & I\end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{A} + \tilde{C}X & \tilde{C} \\ 0 & \tilde{B} - X\tilde{C}\end{bmatrix}}_{:=\tilde{T}}.$$

We can continue from here to compute the sign.

If ε is sufficiently small, $\Lambda(\tilde{A} + \tilde{C}X) \subset LHP$, $\Lambda(\tilde{B} - X\tilde{C}) \subset RHP$, hence $\operatorname{sign}(\tilde{M}) = \begin{bmatrix} I & \tilde{Z} \\ 0 & I \end{bmatrix}$ for a certain matrix $\tilde{Z}$. Arguing as in the Schur-Parlett method, we must have $\operatorname{sign}(\tilde{M})\tilde{M} = \tilde{M}\operatorname{sign}(\tilde{M})$, hence $\tilde{Z}$ solves

$$(\tilde{A} + \tilde{C}X)\tilde{Z} - \tilde{Z}(\tilde{B} - X\tilde{C}) = 2\tilde{C}.$$

Then,

$$\operatorname{sign}(M + E) = \begin{bmatrix} I & 0 \\ X & I \end{bmatrix} \begin{bmatrix} I & \tilde{Z} \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ -X & I \end{bmatrix}. \tag{7.2}$$

The coefficients of the Sylvester equation for $\tilde{Z}$ are a perturbation of magnitude $\mathcal{O}(\frac{\varepsilon}{\delta})$ of those of $AZ - ZB = 2C$. We can apply the following classical result for perturbation of linear systems.

Lemma 7.2. *Let x be the solution to $Tx = c$, and $\tilde{x}$ be the solution to $(T + \delta_T)\tilde{x} = c$. Then,*

$$\frac{\|\tilde{x} - x\|}{\|x\|} \mathrel{\dot{\le}} \kappa(T)\frac{\|\delta_T\|}{\|T\|}.$$

Note that, unlike its counterpart for a perturbation of the RHS vector c, this result for a perturbation of the matrix T holds only up to terms of order $\left(\frac{\|\delta_T\|}{\|T\|}\right)^2$: indeed, the symbol $\dot{\le}$ stands for a first-order inequality.

Applying this result to the vectorization of $AZ - ZB = C$, we get

$$\|\tilde{Z} - Z\|_F \mathrel{\dot{\le}} \underbrace{\frac{1}{\operatorname{sep}(A, B)}}_{=\frac{1}{\delta}} \mathcal{O}(\frac{\varepsilon}{\delta})\|Z\|_F = \mathcal{O}(\frac{\varepsilon}{\delta^2})\|Z\|_F.$$

Plugging this into (7.2) shows that $\|\operatorname{sign}(M + E) - \operatorname{sign}(M)\| = \mathcal{O}(\frac{\varepsilon}{\delta^2})\|Z\|_F = \mathcal{O}(\frac{\varepsilon}{\delta^2})\|\operatorname{sign}(M)\|_F$. □

Even without going through the full proof, we can get an idea of the reason behind this result by thinking about the Schur–Parlett method: the invariant subspace of $\operatorname{sign}(A)$ is $\operatorname{Im} U_1$, that is, the same invariant subspace that is produced by the (backward stable) Schur factorization, and we have proved earlier that this invariant subspace has a condition number of $\frac{1}{\operatorname{sep}(A,B)}$. The computation of Z may introduce some more ill-conditioning of its own, since it is produced by solving a Sylvester equation with small separation $\operatorname{sep}(A, B)$. However, inaccuracies in the computation of Z have no effect on the invariant subspace $\operatorname{Im}(\operatorname{sign}(A) - I)$.

7.3 Newton for the matrix sign

We wish to see that the iteration

$$X_{k+1} = \frac{1}{2}(X_k + X_k^{-1}), \quad X_0 = M \tag{7.3}$$

satisfies $\lim_{k\to\infty} X_k = \operatorname{sign}(M)$.

To study the behavior of this iteration, let us start from the case when $M = V \operatorname{diag}(\lambda_1, \dots, \lambda_n) V^{-1}$ is diagonalizable. Then it is easy to see that

$$X_1 = V \operatorname{diag}(f(\lambda_1), \dots, f(\lambda_n)) V^{-1},$$

where $f(x) = \frac{1}{2}\left(x + \frac{1}{x}\right)$, the map that corresponds to the scalar version of the iteration (7.3). Similarly, by induction,

$$X_k = V \operatorname{diag}(f^{\circ k}(\lambda_1), \dots, f^{\circ k}(\lambda_n)) V^{-1}$$

(we use the symbol $f^{\circ k}$ to denote the composition of f with itself k times).

The map $f(x)$ is the one obtained by applying Newton's method to solve the equation $x^2 - 1 = 0$; this explains the name of the method. The map $f(x)$ has two fixed points ± 1, with (locally) quadratic convergence.

Convergence analysis of the scalar iteration

Theorem 7.3. *Let $x_0 \in \mathbb{C}$, with $\operatorname{Re}(x_0) \neq 0$. Then, the limit of the sequence $x_{k+1} = \frac{1}{2}\left(x_k + \frac{1}{x_k}\right)$ is $\operatorname{sign}(x_0)$.*

Proof. We make a change of variables (*Cayley transform*)

$$y = \frac{x-1}{x+1}, \text{ with inverse } x = \frac{1+y}{1-y}.$$

If $x \in \text{RHP}$, then $|x+1| > |x-1| \implies y$ inside the unit disk.

If $x \in \text{LHP}$, then $|x-1| > |x+1| \implies y$ outside the unit disk.

If $y_k = \frac{x_k - 1}{x_k + 1}$ for each k, then we can check with an explicit computation that $y_{k+1} = y_k^2$.

Hence

$$x_0 \in \text{RHP} \implies |y_0| < 1 \implies \lim_{k\to\infty} y_k = 0 \implies \lim_{k\to\infty} x_k = 1;$$

$$x_0 \in \text{LHP} \implies |y_0| > 1 \implies \lim_{k\to\infty} y_k = \infty \implies \lim_{k\to\infty} x_k = -1.$$

$\square$

Rational approximations of the step function Let $g(x) = \frac{1}{2}(x + 1/x)$; then its iterates $g^{\circ k}$ are rational approximations of the step function $\operatorname{sign}(x)$ around -1 and 1.

```
>> syms x
>> g = 1/2*(x + 1/x);
>> g2 = simplify(subs(g, x, g))
>> g3 = subs(g2, x, g)
>> fplot(g, [-2,2])
>> axis([-2 2 -2 2]);
>> hold on
>> fplot(g2, [-2,2])
>> fplot(g3, [-2,2])
```

(They diverge badly around 0, though.)

Exercise 7.4. Can you guess a general formula for $g^{\circ k}$ from the first iterates `g, g2, g3`? (Hint: it involves binomial coefficients.) Prove this formula by induction.

This rational approximant can also be obtained by imposing approximation properties in $\underline{\text{two}}$ different points, unlike one point for Padé approximant: $g^{\circ k}(x)$ is the only degree-$(k, k-1)$ rational function that satisfies $g^{\circ k}(x) - \operatorname{sign}(x) = \mathcal{O}(x^{2^k})$ for both $x \to \pm 1$.

Convergence analysis of the matrix iteration A modification of the convergence proof for the scalar case works in the matrix case.

Theorem 7.5. *Let $X_0 = M$ have no purely imaginary eigenvalues. Then, the sequence $X_{k+1} = \frac{1}{2}\left(X_k + X_k^{-1}\right)$ converges to* $\operatorname{sign}(M)$.

Proof. Set $S = \operatorname{sign}(M)$. Note that all the iterates X_k are rational functions of M, so they commute with M, with S, and with each other. We can assume (up to a change of basis) that M is upper triangular, in Schur form. Then S and X_k are upper triangular, too.

Set
$$Y_k = (X_k - S)(X_k + S)^{-1}. \tag{7.4}$$
Analyzing the diagonal entries of this expression involving upper triangular matrices, we see that the inverse $(X_k + S)^{-1}$ always exists, and that we have $\rho(Y_k) < 1$.

We can now compute

$$Y_{k+1} = (X_k^{-1}(X_k^2 + I - 2SX_k))X_k(X_k^2 + I + 2SX_k)^{-1} = Y_k^2.$$

Hence $Y_k = Y_0^{2^k} \to 0$.

We can now express X_k as a function of Y_k: from (7.4), since everything commutes,

$$Y_kX_k + Y_kS = X_k - S \implies X_k = S(I + Y_k)(I - Y_k)^{-1}.$$

hence $X_k \to S$. □

The algorithm

1. $X_0 = M$.
2. Repeat $X_{k+1} = \frac{1}{2}(X_k + X_k^{-1})$, until convergence.

We really need to compute a full matrix inverse here; this is unusual in numerical linear algebra.

Scaling Unfortunately, this iterative method requires a large number of iterations if the starting matrix M has a norm that is particularly large or small. Indeed, if $x_k \gg 1$, then

$$x_{k+1} = \frac{1}{2}\left(x_k + \frac{1}{x_k}\right) \approx \frac{1}{2}x_k,$$

and "the iteration is an expensive way to divide by 2" [Higham]. A similar behavior happens if $x_0 \ll 1$: $x_1 \gg 1$, and then convergence is very slow.

Solution: we can replace M with μM for any scalar $\mu > 0$, since $\operatorname{sign}(M) = \operatorname{sign}(\mu M)$. A good choice of μ will avoid this initial phase in which the method spends iterations just to get the eigenvalues close to 1.

Choices of scaling Ideally, we want to choose μ so that the eigenvalues of M are "as close to 1 as possible".

Possibility 1: (determinantal scaling): choose $\mu = (\det M)^{-1/n}$, so that $\det M = 1$. This choice reduces the "mean distance" from 1. This determinant is cheap to compute, since we already need to invert M, and methods to do it (e.g., PLU factorization) typically produce the determinant as a byproduct.

Possibility 2: (spectral scaling): choose μ so that $|\lambda_{\min}(\mu M)\lambda_{\max}(\mu M)| = 1$. We can use a few steps of the power method on M and M^{-1} to obtain cheaply estimate of these two extremal eigenvalues.

Possibility 3: (norm scaling): choose μ so that $\sigma_{\min}(\mu A)\sigma_{\max}(\mu A) = 1$. Again we can use the power method to get cheap estimates.

All these methods work reasonably well in practice. It is important to use one, at least at the first iteration, but which one does not matter much.

```
>> rng(0); X0 = randn(5, 5);
>> X = X0; for k = 1:8; X = 1/2 * (X + inv(X)); eig(X), max(abs(eig(X) - sign(eig(X)))), enc
ans =
   -1.4434e+00 + 0.0000e+00i
   -6.4772e-01 + 1.2086e+00i
   -6.4772e-01 - 1.2086e+00i
   1.1809e+00 + 0.0000e+00i
   1.9607e+00 + 0.0000e+00i
ans =
   9.6068e-01
ans =
   -1.0681e+00 + 0.0000e+00i
   -4.9611e-01 + 2.8288e-01i
   -4.9611e-01 - 2.8288e-01i
   1.0139e+00 + 0.0000e+00i
   1.2354e+00 + 0.0000e+00i
ans =
   4.2891e-01
ans =
   -1.0086e+00 + 2.9222e-01i
   -1.0086e+00 - 2.9222e-01i
   -1.0022e+00 + 0.0000e+00i
   1.0001e+00 + 0.0000e+00i
   1.0224e+00 + 0.0000e+00i
ans =
   5.0101e-02
ans =
   -9.6165e-01 + 1.3610e-02i
   -9.6165e-01 - 1.3610e-02i
   -1.0000e+00 + 0.0000e+00i
   1.0000e+00 + 0.0000e+00i
   1.0002e+00 + 0.0000e+00i
ans =
   3.8256e-02
ans =
   -1.0007e+00 + 5.5212e-04i
   -1.0007e+00 - 5.5212e-04i
   -1.0000e+00 + 0.0000e+00i
   1.0000e+00 + 0.0000e+00i
   1.0000e+00 + 0.0000e+00i
ans =
   6.6082e-04
ans =
   -1.0000e+00 + 3.6449e-07i
   -1.0000e+00 - 3.6449e-07i
   -1.0000e+00 + 0.0000e+00i
   1.0000e+00 + 0.0000e+00i
   1.0000e+00 + 0.0000e+00i
ans =
   6.5977e-08
ans =
   1.0000e+00 + 0.0000e+00i
   1.0000e+00 + 0.0000e+00i
   -1.0000e+00 + 2.4195e-14i
   -1.0000e+00 - 2.4195e-14i
   -1.0000e+00 + 0.0000e+00i
ans =
   6.4171e-14
ans =
   1.0000e+00 + 1.2465e-16i
   1.0000e+00 - 1.2465e-16i
   -1.0000e+00 + 0.0000e+00i
   -1.0000e+00 + 0.0000e+00i
   -1.0000e+00 + 0.0000e+00i
ans =
   4.4409e-16
```

Thanks to quadratic convergence, the iteration is fast even on large matrices.

```
>> rng(0); X0 = randn(500, 500);
>> X = X0; for k = 1:20; X = 1/2 * (X + inv(X)); max(abs(eig(X) - sign(eig(X)))), end
ans =
   1.0385e+01
   4.6593e+00
   1.9126e+00
   1.0053e+00
   1.6457e+00
   7.1975e+00
   3.0520e+00
   9.3412e-01
   6.1630e-01
   4.4063e-01
   7.1573e-01
   7.2528e-01
   1.2337e-01
   1.1769e-01
   5.1907e-03
   2.8347e-05
   7.1731e-10
   2.7978e-14
   3.0198e-14
   2.7089e-14
```

If the initial matrix X has large entries, convergence is slow:

```
>> X = X0; for k = 1:10; X = 1/2 * (X + inv(X)); max(abs(eig(X) - sign(eig(X)))), end
ans =
    1.8236e+06
    9.1179e+05
    4.5590e+05
    2.2795e+05
    1.1397e+05
    5.6986e+04
    2.8493e+04
    1.4246e+04
    7.1224e+03
    3.5607e+03
```

But we can speed it up again with determinantal scaling:

```
>> X = X0; for k = 1:10; X = X * det(X)^(-1/size(X,1)); X = 1/2 * (X + inv(X)); max(abs(eig(
ans =
   1.3678e+00
   3.5660e-01
```

```
  9.9525e-02
  4.1457e-03
  5.3582e-06
  4.4126e-12
  8.8818e-16
  4.4409e-16
  6.6613e-16
  4.4409e-16
```

Remarks:

- In practice, we can use the same LU decomposition to compute `det(X)` and `inv(X)`. Matlab does not offer a convenient library function to do this. Julia has a convenient `factorize` function.

  ```
  julia> using LinearAlgebra
  julia> X = randn(5, 5);
  julia> F = factorize(X)
  LU{Float64, Matrix{Float64}, Vector{Int64}}
  L factor:
  5x5 Matrix{Float64}:
     1.0 0.0 0.0 0.0 0.0
     -0.0375111 1.0 0.0 0.0 0.0
     0.328631 -0.624587 1.0 0.0 0.0
     -0.137578 -0.266761 -0.373021 1.0 0.0
     -0.573464 -0.948448 0.555985 -0.575514 1.0
  U factor:
  5x5 Matrix{Float64}:
     1.61843 0.496628 1.90895 0.92813 0.611563
     0.0 0.556946 -1.41037 1.46116 -2.0252
     0.0 0.0 -1.42944 -0.571146 0.276043
     0.0 0.0 0.0 -1.57252 -0.266344
     0.0 0.0 0.0 0.0 -0.901631
  julia> det(F)
  1.8268274679235024
  julia> inv(F)
  5x5 Matrix{Float64}:
     2.26607 0.354967 1.99599 1.96638 0.659751
     -5.25826 0.88547 -2.8292 -3.51083 -0.714405
     -0.289239 -0.506074 -0.56704 -0.00877184 0.087627
     0.187853 -0.301328 -0.150836 0.128479 -0.527811
     -1.1091 0.378543 -0.985766 -0.885224 -0.638303
  ```

- When the matrices are larger, some care must be taken to ensure that the determinant does not overflow.

Stability of the Newton iteration The analysis of (floating point) stability of the Newton iteration is complicated. [Byers He Mehrmann '97 and Bai Demmel '98] obtain some partial results.

Even though the algorithm contains only sums and inversions, it is difficult to propagate the impact of numerical errors in the first steps, which are the most ill-conditioned ones.

The stability analysis reflects the results of our conditioning analysis above: while the sign in itself is unstable because of the Z block, it produces invariant subspaces as good (numerically) as those computed via a reordered Schur decomposition.

7.4 Extra: Inversion-free sign

Suppose that we are given M, N such that $A = M^{-1}N$. Can we compute $\operatorname{sign}(A)$ without inverting M? Yes: the following algorithm shows how to do it.

Idea: suppose that we can find $\hat{M}, \hat{N}$ such that $MN^{-1} = \hat{M}^{-1}\hat{N}$. Then we can write

$$\begin{aligned} X_1 &= \frac{1}{2}(A + A^{-1}) = \frac{1}{2}(M^{-1}N + N^{-1}M) \\ &= \frac{1}{2}M^{-1}(N + MN^{-1}M) \\ &= \frac{1}{2}M^{-1}(N + \hat{M}^{-1}\hat{N}M) \\ &= \frac{1}{2}M^{-1}\hat{M}^{-1}(\hat{M}N + \hat{N}M) \\ &= (\hat{M}M)\,\frac{1}{2}(\hat{M}N + \hat{N}M) =: M_1^{-1}\,N_1. \end{aligned}$$

Similarly one can produce $M_2, N_2, M_3, N_3, \ldots$

How do we actually find $\hat{M}, \hat{N}$ such that $MN^{-1} = \hat{M}^{-1}\hat{N}$? We can rewrite this relation as $\hat{M}M = \hat{N}N$, or $\begin{bmatrix}\hat{M} & \hat{N}\end{bmatrix}\begin{bmatrix}M \\ -N\end{bmatrix} = 0$. The rightmost matrix has full column rank, if M is invertible; hence we can obtain $\hat{M}, \hat{N}$ from any basis of $\ker \begin{bmatrix}M \\ -N\end{bmatrix}$.

Hence we have replaced the inversion with a kernel computation. In some cases, this can be a much more well-conditioned task than inverting M and/or N, e.g., when

$$\begin{bmatrix}M \\ -N\end{bmatrix} = \begin{bmatrix}1 & 0 \\ 0 & \varepsilon \\ \varepsilon & 0 \\ 0 & 1\end{bmatrix}.$$

In the worst case, this problem is as ill-conditioned as the inversion of M, since one can always take the special solution

$$\begin{bmatrix}\hat{M} & \hat{N}\end{bmatrix} = \begin{bmatrix}NM^{-1} & I\end{bmatrix}.$$

In an article by Benner and Byers, this trick is extended to a ‘linear algebra on matrix pencils’: we can compute operations that involve a matrix A by representing it as a pair (N, M) such that $A = M^{-1}N$. With suitable tricks, these operations can be computed without ever needing to invert M. There is a final project on this topic.

Chapter 8

The matrix square root

Next (and last, for us) matrix function: the square root.

When A has no negative or zero eigenvalues, it is always possible to choose (uniquely) a branch so that $\Lambda(f(A))$ lies in the right half-plane. The matrix function produced by this choice is called the *principal square root* of A:

$$f : \rho e^{i\theta} \mapsto \rho^{1/2} e^{i\frac{\theta}{2}}, \quad \theta \in (-\pi, \pi), \rho \geq 0.$$

We set $f(0) = 0$, but we leave $f(x)$ undefined when x is a negative real number. We denote this function with $x^{1/2}$, and the corresponding matrix function with $A^{1/2}$.

Hence $A^{1/2}$ is well-defined unless A has:

- Real eigenvalues $\lambda_i < 0$, or
- Non-trivial Jordan blocks at $\lambda_i = 0$ (because $f(x) = x^{1/2}$ is not differentiable at 0).

Condition number / sensitivity We have already computed the Fréchet derivative of this function in an earlier example. Let us recall the argument: the Fréchet derivative of $g(Y) = Y^2$ is

$$L_{g,Y}(E) = YE + EY, \quad K = I \otimes Y + Y^T \otimes I.$$

The Fréchet derivative of $f(X) = X^{1/2}$ is its inverse,

$$K_{f,X} = (I \otimes X^{1/2} + (X^{1/2})^T \otimes I)^{-1}.$$

In particular,

$$\|K_{f,X}\| = \operatorname{sep}(X^{1/2}, -X^{1/2}) = \operatorname{sep}(f(X), -f(X)).$$

As we noted earlier, $L_{f,X}$ has eigenvalues with $\frac{1}{\lambda_i^{1/2}+\lambda_j^{1/2}}$, $i, j = 1, \dots, n$. This shows that f is necessarily ill-conditioned for matrices that either:

- have a small eigenvalue (taking $i = j$), or
- have two complex conjugate eigenvalues close to the negative real axis (because then $\lambda_i^{1/2} \approx ai$, $\lambda_j^{1/2} \approx -ai$).

8.1 The modified Schur method

Let us recall the Schur-Parlett method to compute matrix functions:

1. Reduce to a triangular $U = Q^* A Q$ using a Schur form;
2. Compute the diagonal of $S = f(U)$;
3. Compute the off-diagonal entries of S from $SU = US$. The resulting formula involves a denominator $u_{ii} - u_{jj}$; if this quantity is small or 0, trouble ensues, and to avoid it we must work blockwise.
4. Return $f(A) = QSQ^*$.

In the case of $A^{1/2}$, we have another option: rather than $SU = US$, we can use $S^2 = U$ to get the off-diagonal entries of S:

$$s_{ii}s_{ij} + s_{i,i+1}s_{i+1,j} + \cdots + s_{ij}s_{jj} = u_{ij}. \tag{8.1}$$

The advantage is that now the denominator $s_{ii} + s_{jj}$, which is guaranteed to be nonzero because $s_{ii} + s_{jj} \in RHP$.

The method:

1. Reduce to a triangular $U = Q^* A Q$ using a Schur form;
2. Compute the diagonal of $S = f(U)$;
3. For each $j = 1, 2, \ldots, n$ and $i = j-1, j-2, \ldots, 1$, compute the off-diagonal entry s_{ij} of S by solving the equation (8.1).
4. Return $f(A) = QSQ^*$.

```
function B = sqrtm_schurparlett(A)
[Q, U] = schur(A, 'complex');
n = size(A, 1);
S = zeros(n, n);
for j = 1:n
    S(j,j) = sqrt(U(j,j));
    for i = j-1:-1:1
        num = U(i,j) - S(i,i+1:j-1)*S(i+1:j-1,j);
        S(i,j) = num / (S(i,i)+S(j,j));
    end
end
B = Q*S*Q';
```

```
if isreal(A)
    % removes the tiny imaginary part
    B = real(B);
end
```

Matlab does something similar in `sqrtm`, but in a divide-and-conquer fashion: it splits U into four blocks of the same size, computes $S_{11} = f(U_{11})$ and $S_{12} = f(U_{22})$ recursively, and then solves the Sylvester equation $S_{11}S_{12}+S_{12}S_{22} = U_{12}$ to obtain the missing block S_{12}.

Stability of the modified Schur method In general, not much can be said about the stability of the Schur-Parlett method; for a generic function one cannot easily obtain backward stability results. However, in the case of the square root we can prove a stability result.

Theorem 8.1. *Let $U \in \mathbb{C}^{n\times n}$ be an upper triangular matrix. Then, the matrix $\tilde{S}$ computed with machine precision u using the Schur-Parlett variant described above satisfies*

$$\tilde{S}^2 = U + \delta_U, \quad |\delta_U| \leq |S|^2\mathcal{O}(n\mathsf{u}).$$

Here, $|M|$ is componentwise absolute value.

Combining this bound with a Schur form and converting it into a normwise bound, we get for a generic matrix A

$$\|\tilde{X}^2 - A\|_F \leq \|X\|_F^2\mathcal{O}(n^3\mathsf{u}).$$

Note that this bound is weaker than backward stability, because in the RHS we have $\|X\|_F^2$ instead of $\|A\|_F = \|X^2\|_F$: and, due to cancellation, the former may be significantly larger.

Proof. We use standard forward error analysis, in which we replace the result of an operation $a \circledast b$ performed on the machine with its floating-point approximation $(a * b)(1+\varepsilon)$, with $|\varepsilon| \leq \mathsf{u}$; this holds for the four operations $* \in \{+, -, \cdot, /\}$.

Let p denote the scalar product between $\tilde{s}_{i,i+1:j-1}$ and $\tilde{s}_{i+1:j-1,j}$; it is a standard result that a scalar product can be computed with error

$$|\tilde{p} - p| \leq n\mathsf{u}\left(|\tilde{s}_{i,i+1}||\tilde{s}_{i+1,j}| + \cdots + |\tilde{s}_{i,j-1}||\tilde{s}_{j-1,j}|\right) + \mathcal{O}(\mathsf{u}^2).$$

In machine arithmetic, we have

$$\tilde{s}_{ij} = (u_{ij} \ominus \tilde{p}) \oslash (\tilde{s}_{ii} \oplus \tilde{s}_{jj}) = \frac{u_{ij} - \tilde{p}}{\tilde{s}_{ii} + \tilde{s}_{jj}}(1 + \varepsilon_1 + \varepsilon_2 + \varepsilon_3 + \mathcal{O}(\mathsf{u}^2)),$$

where $\varepsilon_1, \varepsilon_2, \varepsilon_3$ account for these three additional operations. Rearranging,

$$u_{ij} - \tilde{p} - (\tilde{s}_{ii}\tilde{s}_{ij} + \tilde{s}_{ij}\tilde{s}_{jj})(1 - \varepsilon_1 - \varepsilon_2 - \varepsilon_3 + \mathcal{O}(\mathsf{u}^2)).$$

Combining with the analysis of the scalar product, we arrive to

$$\begin{aligned}&|u_{ij} - \tilde{s}_{ii}\tilde{s}_{ij} - \tilde{s}_{i,i+1}\tilde{s}_{i+1,j} - \cdots - \tilde{s}_{ij}\tilde{s}_{jj}| \\ &\qquad \leq n\mathsf{u}\left(|u_{ij}| + |\tilde{s}_{ii}||\tilde{s}_{ij}| + |\tilde{s}_{i,i+1}||\tilde{s}_{i+1,j}| + \cdots + |\tilde{s}_{ij}||\tilde{s}_{jj}|\right) + \mathcal{O}(\mathsf{u}^2).\end{aligned}$$

We can remove all tildes, since this introduces an error of the same order as the term $\mathcal{O}(\mathsf{u}^2)$ that we already have, and use

$$|u_{ij}| = |s_{ii}s_{ij} + \cdots + s_{ij}s_{jj}| \leq |s_{ii}||s_{ij}| + \cdots + |s_{ij}||s_{jj}|$$

to get rid of the term $|u_{ij}|$. This gives a bound on the (i,j) entry of $U - S^2$. □

8.2 Relation to the sign function and matrix iterations

The following result relates the sign function and the matrix square root, showing that we can use one to compute the other and viceversa.

Proposition 8.2. *1. For $A \in \mathbb{C}^{n\times n}$ with no real negative eigenvalues,* $\operatorname{sign}(A) = A(A^2)^{-1/2}$*, where $A^{-1/2}$ denotes the inverse of the principal square root $A^{1/2}$.*

2. For $A, B \in \mathbb{C}^{n\times n}$ such that AB has no real negative eigenvalues, (and hence neither does BA),

$$\operatorname{sign}\begin{bmatrix}0 & A\\ B & 0\end{bmatrix} = \begin{bmatrix}0 & C\\ C^{-1} & 0\end{bmatrix}, \quad C = A(BA)^{-1/2}.$$

Proof. 1. It is enough to prove the corresponding scalar identity, $\operatorname{sign}(x) = \frac{x}{(x^2)^{1/2}}$, since algebraic identities between scalar functions extend to the corresponding matrix functions. The quantity x^2 has two square roots, $+x$ and $-x$; the principal square root is x if $x \in RHP$ and $-x \in LHP$, and from here we conclude easily.

2. Use $\operatorname{sign}(A) = A(A^2)^{-1/2}$, and then use the relation $\operatorname{sign}(A)^2 = I$ to show that the $(2,1)$ block does indeed contain C^{-1}.

□

In particular,

$$\operatorname{sign}\begin{bmatrix}0 & A\\ I & 0\end{bmatrix} = \begin{bmatrix}0 & A^{1/2}\\ A^{-1/2} & 0\end{bmatrix}.$$

This relationship suggests using similar iterations to those used for the matrix sign.

In fact, the first method we start from is another classical method, which (we will see) can be transformed into a variant of the matrix sign iteration.

Newton method on $X^2 - A$

We can run the Newton method on the map $G(X) = X^2 - A$, $G : \mathbb{C}^{n\times n} \to \mathbb{C}^{n\times n}$.

Its Jacobian is the Fréchet derivative $L_{G,X}[E] = EX + XE$, hence we have

$$X_{k+1} = X_k - L^{-1}_{G,X_k}[G(X_k)],$$

i.e.,

$$X_{k+1} = X_k - E, \quad \text{where } E \text{ solves } EX_k + X_kE = X_k^2 - A. \tag{8.2}$$

On paper, using this iteration is much more expensive than the Schur method: we must solve a Sylvster equation at each step, and in turn this requires a Schur factorization. Clearly a method that requires computing one Schur factorization per step cannot be better than a method that requires only one Schur factorization plus a (cheap) back-substitution step. However, we can find a cheap closed-form solution to those Sylvester equations.

Lemma 8.3. *Suppose the method* (8.2) *is run with an initial matrix X_0 that commutes with A, for instance $X_0 = \alpha I$ or $X_0 = \alpha A$, for $\alpha > 0$. Then,*

1. *A and X_k commute;*
2. *we can take $E = (2X_k)^{-1}(X_k^2 - A)$ at each step.*

Proof. We prove both points at the same time by induction. Point 1 is obvious for $k = 0$. Once point 1 is established, $(2X_k)^{-1}(X_k^2 - A)$ commutes with A, and we can plug it into the Sylvester equation to check that it satisfies it. This not only proves 2, but shows that E commutes with A and hence also $X_{k+1} = X_k - E$ does. □

After plugging in this formula for E, we obtain the following cheaper algorithm, the Modified Newton iteration (MN).

$$X_{k+1} = \frac{1}{2}(X_k + X_k^{-1}A), \quad X_0 = \alpha I \text{ or } X_0 = \alpha A.$$

We still have to prove that the Newton method converges to the <u>principal</u> square root rather than to another solution of $X^2 = A$.

Theorem 8.4. *Assume A has no eigenvalues in $\mathbb{R}_-$. Then, the MN and TN iterations converge to the principal square root $A^{1/2}$ for each starting point of the form $X_0 = \alpha I$ or $X_0 = \alpha A$, with $\alpha > 0$.*

Proof. We start from MN. Pre-multiply by $A^{-1/2}$, and use commutativity:

$$A^{-1/2}X_{k+1} = \frac{1}{2}\left(A^{-1/2}X_k + (A^{-1/2}X_k)^{-1}\right).$$

This is the Newton iteration for the matrix sign! Hence $A^{-1/2}X_k \to \operatorname{sign}(A^{-1/2}X_0) = I$.

As the two formulas produce the same sequence of matrices X_n, the same property holds for TN. □

Theory and practice *Problem* All of this holds in *exact arithmetic*, but MN often doesn't work in practice in machine arithmetic!

```
format short e; % for better error display
rng(0); M = randn(10); M = M*M';
X = eye(size(M));
Y = eye(size(M));
T = table();
for k = 1:15
    X = X - lyap(X, X, M-X^2); % TN
    Y = 1/2*(Y + Y\M); % MN
    T.TNres(k) = norm(X^2-M)/norm(M);
    T.MNres(k) = norm(Y^2-M)/norm(M);
    T.difference(k) = norm(X-Y)/norm(Y);
    T.TNcommute(k) = norm(X*M-M*X)/norm(M)/norm(X);
    T.MNcommute(k) = norm(Y*M-M*Y)/norm(M)/norm(Y);
end
T
```

```
TNres MNres difference TNcommute MNcommute
---------- ---------- ---------- ---------- ----------

1.2716e+01 1.2716e+01 0.0000e+00 4.5914e-17 4.5914e-17
2.9472e+00 2.9472e+00 9.4733e-16 2.8536e-16 9.2812e-17
5.5013e-01 5.5013e-01 1.9541e-15 6.8206e-16 4.4933e-16
4.8810e-02 4.8810e-02 3.3700e-15 3.5178e-16 3.1431e-15
5.6788e-04 5.6788e-04 2.4031e-14 9.0480e-17 2.3496e-14
8.0577e-08 8.0577e-08 1.8106e-13 1.0374e-16 1.7888e-13
1.5991e-15 1.4569e-12 1.3761e-12 5.7152e-17 1.3656e-12
8.2069e-17 1.1128e-11 1.0497e-11 8.2657e-17 1.0438e-11
9.9611e-17 8.5061e-11 8.0189e-11 1.1024e-16 7.9815e-11
9.1823e-17 6.5044e-10 6.1301e-10 6.9810e-17 6.1043e-10
8.7288e-17 4.9746e-09 4.6877e-09 8.7521e-17 4.6690e-09
1.0434e-16 3.8049e-08 3.5853e-08 6.9442e-17 3.5713e-08
5.8770e-17 2.9104e-07 2.7423e-07 1.0662e-16 2.7318e-07
8.5101e-17 2.2262e-06 2.0976e-06 6.9301e-17 2.0896e-06
8.5801e-17 1.7029e-05 1.6045e-05 6.1899e-17 1.5984e-05
```

There is nothing apparently wrong with our matrix M, apart with moderate ill-conditioning $\kappa(M) \approx 266$, but we see that already in this simple example MN reaches residual 10^{-12} at best, but then starts to diverge. On the other hand, TN gives good results.

The final two columns `TNcommute` and `MNcommute` hint to a possible reason for this notable discrepancy with the results in exact arithmetic: the matrices computed by MN no longer commute with A, as they were supposed to.

The geometric picture TN and MN coincide on the manifold of matrices that commute with A, $\{X \in \mathbb{C}^{n\times n} \colon AX = XA\}$, but not on the rest of $\mathbb{C}^{n\times n}$.

Numerical perturbations cause the iterates to fall outside of the manifold, where the two methods do not coincide anymore.

From the general theory if Newton methods, it follows that TN is quadratically convergent. However, MN does not have a *stable fixed point* in $A^{1/2}$: there are starting points arbitrarily close to $A^{1/2}$ for which the sequence diverges. To prove this formally, we need to recall a few facts from the theory of (discrete-time) dynamical systems.

Discrete-time dynamical systems Consider the discrete-time dynamical system induced by a sufficiently regular map $F : \mathbb{C}^n \to \mathbb{C}^n$, i.e.,

$$x_{k+1} = F(x_k).$$

Starting from $x_0 = x_* + e$ close to a *fixed point* $x_* = F(x_*)$,

$$\begin{aligned} x_1 &= F(x_* + e) = x_* + F'_{x_*} e + \mathcal{O}(\|e\|^2), \\ x_k &= x_* + (F'_{x_*})^k e + \mathcal{O}(\|e\|^2). \end{aligned}$$

If $\rho(F'_{x_*}) < 1$, then x_k converges to x_*; we call x_* a *stable fixed point.* If $\rho(F'_{x_*}) > 1$, the iterates (for almost all starting points) diverge away from x_*; we call it an *unstable fixed point.*

This kind of convergence is known as *linear convergence* in numerical linear algebra:

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x_*\|}{\|x_k - x_*\|} \to \rho(F'_{x_*}),$$

but other fields use the term *exponential convergence* (yes, this is confusing).

We can study our maps MN and TN in this framework, after vectorization. The Jacobian F'_{x_*} then becomes the Fréchet derivative of the map $X_k \mapsto X_{k+1}$.

Local stability We can study the local stability of the MN map $h(X) = \frac{1}{2}(X + X^{-1}A)$. Its Fréchet derivative is

$$L_{h,X}(E) = \frac{1}{2}(E - X^{-1}EX^{-1}A).$$

Hence, $L_{h,A^{1/2}} = \frac{1}{2}(E - A^{-1/2}EA^{1/2})$, or $K_{h,A^{1/2}} = \frac{1}{2}\left(I - (A^{1/2})^T \otimes A^{-1/2}\right)$. The n^2 eigenvalues of $K_{h,A^{1/2}}$ are given by $\frac{1}{2} - \frac{1}{2}\lambda_i^{1/2}\lambda_j^{-1/2}$, where $\lambda_1, \dots, \lambda_n$ are the eigenvalues of A.

It is easy to construct an example in which one of these eigenvalues is larger than 1, by taking a matrix A with two eigenvalues λ_i, λ_j that are sufficiently far apart; hence $A^{1/2}$ is an *unstable fixed point* of $h(X)$.

On the other hand, TN has quadratic (or in other fields "doubly exponential") convergence. This follows from the fact that it is a multivariate Newton method, and the multivariate Newton method is defined so that the Jacobian of the iteration map is 0.

Together, these results explain the surprising difference between these two apparently equivalent methods.

Extras: Denman–Beavers iteration However, the stability properties are significantly different for slight variations of the modified Newton's method. If we set $Y_k = A^{-1}X_k$, we can get the coupled iteration

$$X_{k+1} = \frac{1}{2}(X_k + Y_k^{-1}),$$
$$Y_{k+1} = \frac{1}{2}(Y_k + X_k^{-1}),$$

known as Denman–Beavers iteration [Denman–Beavers, '76]. The same iteration can be obtained by expanding blocks in the Newton iteration for $\operatorname{sign}(\left[\begin{smallmatrix} 0 & A \\ I & 0 \end{smallmatrix}\right])$, which we have seen above to produce $A^{1/2}$.

Local stability of the DB iteration Theorem
The DB iteration satisfies $\lim(X_k, Y_k) = (A^{1/2}, A^{-1/2})$, and it is locally stable.

We have
$$L_{DB,(X,Y)}\left(\begin{bmatrix} E \\ F \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} E - Y^{-1}FY^{-1} \\ F - X^{-1}EX^{-1} \end{bmatrix}$$

Using the fact that $X_*Y_* = I$, one can verify that the Jacobian is idempotent, i.e., $(K_{DB,(B,B^{-1})})^2 = K_{DB,(B,B^{-1})}$. Hence it has bounded powers, and this implies a weaker form of stability: the perturbations coming from machine arithmetic produce an error that stays bounded after each step (at least in the first-order).

Exercise 8.5. Perform a local stability analysis of the Newton method for the matrix sign. You should be able to conclude that it has the same stability properties as the DB iteration.

Other variants of the MN method are studied on [Higham book, Ch. 6], if you are interested.

Chapter 9

Functions of large-scale matrices

Functions of large-scale matrices How do we compute $f(A)$ if A is large and sparse? This is a topic of recent research. We can consider it as an extension of methods to solve large-scale linear systems, which is the case $f(x) = x^{-1}$.

Most of the time, one wants $f(A)b$ rather than $f(A)$, because $f(A)$ is full. Some of the main techniques are the following:

1. Replace f with an *approximating polynomial/rational function* on a region U that includes the spectrum of A. This might be challenging if the spectrum of A is spread across a large region, or if it is unknwon to begin with.

2. *Contour integration* paired with a numerical quadrature formula:

$$f(A)b = \frac{1}{2\pi i}\int_\Gamma f(z)(zI - A)^{-1}b\mathsf{d}z \approx \sum_{k=1}^{N} w_k f(x_k)(x_k I - A)^{-1}b.$$

   This is an interesting approach that uses our Cauchy integral formula. Unfortunately, the convergence is typically linear with the number of quadrature nodes N. In the end, this formula also produces a rational approximant for f in the RHS, so the problem of choosing the quadrature formula to use is equivalent to the one of choosing a rational approximant.

3. Ad-hoc methods for special problems, e.g., discretization of *differential equations*: $\exp(A)b = v(1)$ where $\dot{v}(t) = Av(t)$, $v(0) = b$. We have already outlined the drawbacks of using differential equations to compute the matrix exponential; moreover, the classical discretization methods for ODEs produce once again polynomial or rational functions of A, so we are led back to the same problem.

9.1 Arnoldi for matrix functions

A different possibility, which contains a way to construct a well-suited approximation function, is using the "Swiss-army knife" algorithm for large matrices: Arnoldi.

Krylov spaces and their effectiveness Let us recall the Arnoldi algorithm, with matrix functions in mind. Let $A \in \mathbb{C}^{m\times m}$, and $n \leq m$. We define the Krylov subspace

$$\begin{aligned} K_n(A,b) &= \operatorname{span}(b, Ab, A^2b, \dots, A^{n-1}b) \\ &= \{p(A)b \colon p \text{ polynomial of degree } < n\}. \end{aligned}$$

The interesting feature of Krylov spaces is that in a problem with a matrix A and a vector b, such as a linear system, most of the "action" often happens inside the subspace $K_n(A,b)$, so we can replace the problem with its projection on the space $K_n(A,b)$, and in many cases the solution of the projected problem converges quickly (as n grows) to that of the original problem. Let us describe this idea more formally. Take an orthonormal basis V_n of $K_n(A,b)$, and define the orthogonal projection matrix $P = V_nV_n^*$. The projection of b is $Pb = b$, while the projected version of A is

$$PAP = V_n \underbrace{(V_n^* A V_n)}_{A_n \in \mathbb{C}^{n\times n}} V_n^*,$$

So, for instance, to compute an approximation of the solution of the linear system $Ax = b$ we can search for $\hat{x} = V_n y$ that solves the projected problem $V_n^*(Ax - b) = 0$.

The effectiveness of Krylov spaces is related to their eigenvalue approximation property: for most choices of A and b, one sees that $\Lambda(A_n)$ approximates well the outer eigenvalues of A, i.e., those with larger absolute value. A very nice visual example is on `https://en.wikipedia.org/wiki/Arnoldi_iteration#/media/File:Arnoldi_Iteration.gif`. We will not elaborate on why this happens (also because it does not hold for all matrices), but the intuition is that if (λ_i, v_i) are the eigenpairs of a diagonalizable A, and b is written in the eigenvector basis as

$$b = v_1\alpha_1 + \cdots + v_n\alpha_n,$$

then

$$A^kb = v_1\alpha_1\lambda_1^k + \cdots + v_n\alpha_n\lambda_n^k,$$

and the largest components here are those with large $|\lambda_i|$: so A^kb lies approximately in the span of the outer eigenvectors. Hence the space of the vectors A^kb and that of the leading eigenvectors are "close". In the context of Krylov methods, the eigenvalues of A_n are called Ritz values of A.

The Arnoldi algorithm is an algorithm to compute (in a more stable way) an orthonormal basis V_n of $K_n(A,b)$, and together with it an expression for the projected operator $A_n = V_n^* A V_n$ and one for the vector $V_n^* b$ (Actually, in Arnoldi, $V_n^* b = e_1 \beta$, where e_1 is the first vector of the canonical basis, and $\beta = \|b\|$.)

Formula for $p(A)b$

Lemma 9.1. *For all polynomials of degree $d < n$,*

$$p(A)b = V_n p(A_n) V_n^* b = V_n p(A_n) V_n^* b.$$

Proof. By linearity, it is sufficient to show that $A^j b = V_n H_n^j V_n^* b$ for $j < n$.

$$V_n H_n^j V_n^* = V_n V_n^* A V_n V_n^* A \cdots V_n V_n^* A V_n V_n^* A V_n V_n^* b$$

Let us start from the right. $V_n V_n^*$ is the orthogonal projection matrix onto the Krylov space. Since $b \in K_n(A, b)$, $V_n V_n^* b = b$.

Now the rightmost part of this expression reads $V_n V_n^* A b$; but this equals Ab because $Ab \in K_n(A, b)$, and so on. $\square$

Arnoldi, matrix functions, and polynomial approximations We already know that $f(A) = p(A)$ for a certain polynomial p of degree $\deg p < m$, the interpolating polynomial; however, since its degree is so high, there is no advantage in using this p. But the previous lemma suggests an idea: we can take $c = V_n f(A_n) V_n^* b$ as an approximation of $f(A)$, for an arbitrary f.

Note that the vector c is actually a polynomial approximation of $f(A)b$:

$$c = V_n f(A_n) V_n^* b = V_n \tilde{p}(A_n) V_n^* b = \tilde{p}(A) b,$$

where $p_n(x)$ is the interpolating polynomial to f on $\Lambda(A_n)$, of degree n; this is different from the approximating polynomial $p(x)$ $\Lambda(A)$, which has much higher degree m. But recall that often the eigenvalues of A_n (Ritz values) approximate the outer eigenvalues of A.

We can sketch an informal argument to explain why (and when) this approximation should work well.

For a diagonalizable $A = W \Lambda W^{-1}$, we are computing $c = W p_n(\lambda) W^{-1} b$ instead of $f(A)b = W f(\lambda) W^{-1} b$;

- for "outer" eigenvalues, we can expect that $f(\lambda) \approx p_n(\lambda)$ because $f(\mu) = p_n(\mu)$ for a nearby Ritz value $\mu \in \Lambda(A_n)$, $\mu \approx \lambda$.
- for "inner" eigenvalues, we have no guarantees. However, we can expect that the contributions of these inner eigenvalues is smaller, if our function is chosen so that $|f(\lambda)|$ is larger for the outer eigenvalues. This happens, for instance, for $\exp(A)$, or for $A^{1/2}$.

The assumption that $|f(\lambda)|$ is larger for "outer" eigenvalues is not too far-fetched: it reminds us of the maximum modulus principle in complex analysis: if f is a holomorphic function defined on a connected closed domain $D \subseteq \mathbb{C}$, then $|f(z)|$ takes its maximum on the boundary ∂D.

A more precise error bound, for Hermitian A We can prove an actual bound here, and it is a strong one: the polynomial p_n produced by Arnoldi is at most a factor 2 away from the best possible one.

Theorem 9.2. *Let A be Hermitian, and consider the interval $\mathcal{I} = [\lambda_{\min}, \lambda_{\max}]$. Let $a(x)$ be the best-approximation polynomial to f on $\mathcal{I}$, i.e., the one that attains the minimum of $\delta = \max_{x\in\mathcal{I}}|f(x) - a(x)|$ among all polynomials of degree $d < n$. Then,*

$$\|f(A)b - c\| \leq 2\delta\|b\|.$$

(And, magically, Arnoldi can achieve this error without even computing $a(x)$!)

Proof. Since the Arnoldi approximation is exact on the polynomial $a(x)$, we can sum and subtract $a(A)b$ to obtain

$$\begin{aligned}\|f(A)b - c\| &= \|f(A)b - V_n f(A_n)V_b^* b\| \\ &= \|(f-a)(A)b - V_n(f-a)(A_n)V_n^* b\| \\ &\leq \|(f-a)(A)\|\|b\| + \|V_n\|\|(f-a)(A_n)\|\|V_n^*\|\|b\|.\end{aligned}$$

We would now like to show that both summands in the last line are bounded by $\delta\|b\|$. For the first one, the result follows by diagonalization: applying the function $f-a$ to $A = Q\Lambda Q^*$, with Q orthogonal, we get $(f-a)(A) = Qf(\Lambda)Q^*$, where $f(\Lambda)$ is the diagonal matrix that has $(f-a)(\lambda_i)$ on its diagonal; as $\lambda_i \in \mathcal{I}$, we have

$$|(f-a)(\lambda_i)| \leq \delta \quad i = 1, \dots, m,$$

and hence $\|(f-a)(A)\| = \|(f-a)(\Lambda)\| \leq \delta$.

To conclude, we use the same argument to bound $\|(f-a)(A_n)\| \leq \delta$; however, to do this, we have to show that the eigenvalues of $A_n = A_n^*$ are in $\mathcal{I}$, too. Let μ be a Ritz value; then $A_n x = x\mu \implies \mu = \frac{x^* A_n x}{x^* x} = \frac{x^* V_n^* A V_n x}{x^* V_n^* V_n x} = \frac{y^* A y}{y^* y}$, where $y = V_n x$; hence μ is a Rayleigh quotient for A. We use again the diagonalization $A = Q\Lambda Q^*$ and set $z = Q^* y$, to get

$$\mu = \frac{y^* A y}{y^* y} = \frac{z^* \Lambda z}{z^* z} = \frac{\sum |z_i|^2 \lambda_i}{\sum |z_i|^2},$$

i.e., μ is a convex combination of the eigenvalues of A. Since $\mathcal{I}$ is convex and $\lambda_i \in \mathcal{I}$ for each i, we also have $\mu \in \mathcal{I}$. □

It is easy to extend this result to any <u>normal</u> matrix A: the only change is that we need to replace the interval $\mathcal{I}$ with $\operatorname{hull}(\Lambda(A))$, the convex hull of the eigenvalues of A.

Is there any hope to generalize this bound to a non-normal matrix A? We see it in the next section, since there are several interesting points.

Towards an error bound for non-normal matrices To prove a similar bound also for non-normal A, there are two hurdles. First of all, it is no longer true that the eigenvalues of A_n lie in the convex hull of the eigenvalues of A. However, there is a way to generalize this inclusion.

Define the *field of values* or *numerical range*

$$\mathbb{W}(A) = \left\{ \frac{x^*Ax}{x^*x} : x \in \mathbb{C}^n \setminus \{0\} \right\} = \{\text{set of Rayleigh quotients of } A\}.$$

Clearly $\Lambda(A) \subseteq \mathbb{W}(A)$, but this region is not simple to describe in general.

Exercise 9.3. 1. If A is a normal matrix, show that $\mathbb{W}(A) = \operatorname{hull}(\Lambda(A))$.

2. If $A = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$, show that $\mathbb{W}(A) = B(0, 1/2)$, the disc in the complex plane with center 0 and radius 1/2.

It is simple to see that for each A

$$\Lambda(A) \subset \mathbb{W}(A) \subset B(0, \|A\|)$$

(where in the right-hand side we have the ball centered in 0 of radius $\|A\|$); moreover, a theorem due to Hausdorff and Toeplitz tells us that $\mathbb{W}(A)$ is a closed convex set.

Note that $\mathbb{W}(A_n) \subseteq \mathbb{W}(A)$, since each Rayleigh quotient for A_n is also a Rayleigh quotient for A, as we have seen above. Hence we can try to use $\mathbb{W}(A)$ to replace the interval $\mathcal{I}$ in the normal case.

The other hurdle is that A and A_n are not normal anymore, so the expression $\|f(A)\| = \max_{\lambda \in \Lambda(A)} |f(\lambda)|$ is not true anymore.

The following technical result, whose proof is far from easy, provides a generalization, which relies again on the numerical range.

Theorem 9.4 (Crouzeix-Palencia theorem [Crouzeix, Palencia 2017]). *Let $\gamma = 1 + \sqrt{2}$. Then, for any matrix A and any function f that is holomorphic on $\mathbb{W}(A)$ we have the inequality*

$$\|f(A)\| \leq \gamma \max_{z \in \mathbb{W}(A)} |f(z)|.$$

For a short (but not easy) proof, see also [Crouzeix, Palencia 2017, Ransford, Schwenninger 2018].

Crouzeix's conjecture, which is still an open problem, states that we can replace $\gamma = 1 + \sqrt{2}$ with the smaller constant $\gamma = 2$.

An error bound for non-normal matrices We now have all the ingredients to give the following bound.

Theorem 9.5. *Let $A \in \mathbb{C}^{n \times n}$, and let $a(z)$ be the best-approximation polynomial to f in $\mathbb{W}(A)$, i.e., the one that attains the minimum of*

$$\delta = \max_{z \in \mathbb{W}(A)} |f(z) - a(z)|$$

over all polynomials of degree $d < n$. Then,

$$\|f(A)b - c\| \leq 2\gamma\delta\|b\|,$$

where γ is the constant in the Crouzeix-Palencia theorem.

Proof. As we noted above, $\mathbb{W}(A_n) \subseteq \mathbb{W}(A)$, so $\|(f-a)(A_n)\| \leq \gamma\delta$. Now we can follow the same strategy as in the normal case:

$$\begin{aligned}\|f(A)b - c\| &= \|f(A)b - V_n f(A_n)V_n^* b\| \\ &= \|(f-a)(A)b - V_n(f-a)(A_n)V_n^* b\| \\ &\leq \|(f-a)(A)\|\|b\| + \|V_n\|\|(f-a)(A_n)\|\|V_n^*\|\|b\| \\ &\leq \gamma\delta\|b\| + \gamma\delta\|b\|.\end{aligned}$$ □

9.2 Arnoldi for matrix function: experiments

We use the following implementation of Arnoldi.

```
function [V, H] = arnoldi(A, b, n)
% Arnoldi algorithm
%
% Compute a nested orthogonal basis V for K_{n+1}(A,b)
% as well as the n+1 x n matrix of scalar products H
% note that V(:,1:n)' * A * V(:,1:n) = H(1:end-1, :)

H = zeros(n+1,n);
V = zeros(length(b), n+1);

V(:, 1) = b / norm(b);
for j = 1 : n
    w = A * V(:, j);
    % Modified Gram-Schmidt: the following loop
    % is equivalent to the orthogonal projection
    % w = w - V(:,1:i)*V(:,1:i)'*w
    % but the order of operations in the loop
    % is more stable
    for i = 1:j
        H(i,j) = V(:, i)' * w;
        w = w - V(:, i) * H(i,j);
    end
    % the following loop is the so-called
    % "reorthogonalization", which improves
    % numerical stability.
    % In exact arithmetic, it would do nothing,
    % since w is already orthogonal to all
    % columns of V(:,1:j)
    for i = 1:j
        betai = V(:, i)' * w;
        w = w - betai * V(:, i);
    end
    H(j+1,j) = norm(w);
    V(:, j+1) = w / H(j+1,j);
end
```

Using this implementation, we can test matrix function computation with different values of n.

```
rng(0);
m = 2000;
A = randn(m, m); b = randn(m, 1);
nmax = 100;
[V, H] = arnoldi(A, b, nmax);
v = expm(A)*b; % the exact result
errors = [];
for n = 1:nmax-1
    % this produces the same matrices as re-running Arnoldi
    % with n <= nmax steps, due to the "nested" property
    Vn = V(:, 1:n);
    An = H(1:n, 1:n);
    c = Vn * expm(An) * eye(n,1)*norm(b);
    errors(n) = norm(c-v) / norm(v);
end
semilogy(errors);
```

After 100 iterations, the relative error is down to 10^{-12}; hence we have have computed (the action of) the exponential of a 2000×2000 matrix by only running `expm` on a 100×100 one.

The result is slightly better if we replace A with the Hermitian $M = A + A^T$; in this case the error has a plateau around $n = 90$ at the order of 10^{-14}. This order of magnitude is the best we can expect, since the condition number of the exponential on a Hermitian M is $\|M\| \approx 125.8$.

Improving on Arnoldi Our intuition from the paragraphs above is that Arnoldi approximates the "outer" eigenvalues; and the approximation is effective to compute $f(A) = \exp(A)$ because the eigenvalues that have the largest $|f(\lambda_i)| = |e_i^{\lambda}| = e^{\mathrm{Re}(\lambda_i)}$ are the rightmost ones. However, one might think that it was wasteful of Arnoldi to approximate the eigenvalues on the <u>left</u> of the spectrum accurately, since their exponential is very small and unlikely to have a noticeable effect in the approximation. If we recall the functioning of the power algorithm and orthogonal iteration, we can try to fix the issue in the same way, with a "shift-and-invert Arnoldi": we compute V_n not as a basis of $K_n(A, b)$, but as a basis of $K_n((A - \alpha I)^{-1}, b)$, where α is a suitable shift value. If α is real and lies to the right of the spectrum of A, the largest eigenvalues of $(A-\alpha I)^{-1}$ are obtained from the rightmost eigenvalues of A. We see an example here.

```
rng(0);
m = 2000;
A = randn(m, m); b = randn(m, 1);
M = inv(200*eye(m) - A);
nmax = 100;
[V, H] = arnoldi(M, b, nmax);
v = expm(A) * b; % the exact result
errors = [];
for n = 1:nmax-1
    % this produces the same matrices as re-running Arnoldi
    % with n <= nmax steps, due to the "nested" property
    Vn = V(:, 1:n);
    % there is no convenient formula for An anymore
    An = Vn' * A * Vn;
    c = Vn * expm(An) * eye(n,1)*norm(b);
    errors(n) = norm(c-v) / norm(v);
end
clf;
figure(1);
semilogy(errors);
figure(2);
lambda = eig(A);
mu = eig(An);
plot(real(lambda), imag(lambda), 'x', real(mu), imag(mu), 'o');
```

This method displays faster convergence, taking only 60 iterations to reach a plateau of about 10^{-14}. Plotting the eigenvalues of A and the Ritz values supports our intuition that this method approximates the eigenvalues in the right part of $\Lambda(A)$. With a more careful implementation than our one, the only additional cost with respect to standard Arnoldi would be the one to compute a sparse LU factorization of $A-\alpha I$.

The idea can be generalized to more than one pole, obtaining so-called rational Arnoldi.

9.3 Extras: the Arnoldi algorithm (in a generalizable way)

Computing Krylov spaces To work with Krylov subspaces efficiently, one must compute an orthogonal basis V_n.

The first idea is computing $V_n = [v_1, v_2, \dots, v_n]$ as the (thin) Q factor of

$$\mathrm{qr}([b, Ab, \dots, A^{j-1}b]).$$

Unfortunately this idea is doomed to fail, because the condition number of this matrix is typically very large: indeed, by the power method, for large k we expect the vectors $A^k b$ to be very close to multiples of the leading eigenvector of A.

As Krylov spaces are nested one into the other

$$K_1(A,b) \subset K_2(A,b) \subset K_3(A,b) \subset \dots$$

$$\operatorname{span}(b) \subset \operatorname{span}(b, Ab) \subset \operatorname{span}(b, Ab, A^2 b), \dots,$$

it makes sense to compute a set of nested orthonormal bases, i.e., a sequence $v_1, v_2, \dots$ such that $(v_1, \dots, v_j)$ is a basis of $K_j(A,b)$ for every j.

The problem we wish to solve is the following then: given an orthonormal basis $(v_1, v_2, \dots, v_j)$ of $K_j(A,b)$, can we find one additional vector v_{j+1} so that $(v_1, v_2, \dots, v_j, v_{j+1})$ is an orthonormal basis of $K_{j+1}(A,b)$?

Arnoldi iteration Idea: to compute v_{j+1}, it is sufficient to take any vector $w \in K_{j+1}(A,b) \setminus K_j(A,b)$ and orthogonalize it against all the previous vectors, using the Gram-Schmidt process.

```
w = A*V(:,j); % the continuation vector
for i = 1:j
    alpha(i,j) = V(:,i)' * w;
    w = w - V(:,i) * alpha(i,j);
end
alpha(j+1,j) = norm(w);
V(:,j+1) = w / alpha(j+1,j);
```

As a starting point for the iteration, we take $v_1 = \frac{b}{\beta}$, with $\beta = \|b\|$, so that $\|v_1\| = 1$.

Remark: the above form for the loop is called *modified Gram–Schmidt* (MGS): we compute coordinates α_{ij} one by one, and we subtract the component $v_i \alpha_{ij}$ from w *immediately*. This is more stable than the alternative form in which replace the whole `for` loop with

```
    alpha(1:j,j) = V(:,1:j)' * w;
    w = w - V(:,1:j) * alpha(1:j, j);
```

(traditional Gram–Schmidt).

Remarks on Arnoldi This algorithm computes nested bases for the Krylov subspaces:

$$K_j(A,b) = \operatorname{Im} \begin{bmatrix} v_1 & v_2 & \dots & v_j \end{bmatrix}, \quad j = 1, 2, \dots, n.$$

Why did we choose $w = Av_j$ here? Because with this choice we can prove that $\alpha_{j+1,j} \neq 0$.

Lemma 9.6. *Suppose $K_{j+1}(A,b)$ has maximal dimension $j+1$. Then,* *1.* $v_j \in K_j(A,b) \setminus K_{j-1}(A,b)$

2. $\alpha_{j+1,j} \neq 0$.

Proof. We prove the two statements together by induction on j. We can start from $j = 1$, as long as we set $K_0(A,b) = \{0\}$, and then the first statement is obvious. Note that 1. means that the relation $v_j = p(A)b$ holds with a polynomial p of degree *exactly* $j-1$. This polynomial is uniquely determined because $K_{j+1}(A,b)$ has maximal dimension.

Hence, before the `for` cycle, we have $w = Av_j = q(A)b$ with $q(x) = xp(x)$ of degree exactly j, i.e., $w \in K_{j+1}(A,b) \setminus K_j(A,b)$. Again, this polynomial is unique. The same property holds for the value of the variable w after the `for` cycle, since at each step we subtract from it an element of $K_j(A,b)$, i.e., a vector of the form $q(A)b$, where $\deg(q) < j$. Thus, after the loop $w \in K_{j+1}(A,b) \setminus K_j(A,b)$; in particular $\alpha_{j+1,j} = \|w\| \neq 0$. □

Arnoldi: the associated matrix Gathering all the relations involving the Av_j in a matrix, we get

$$A \underbrace{\begin{bmatrix} v_1 & \dots & v_n \end{bmatrix}}_{V_n} = \underbrace{\begin{bmatrix} v_1 & \dots & v_{n+1} \end{bmatrix}}_{V_{n+1}} \underbrace{\begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \dots & \alpha_{1,n} \\ \alpha_{1,2} & \alpha_{2,2} & \alpha_{2,3} & \dots & \alpha_{2,n} \\ 0 & \alpha_{3,2} & \alpha_{3,3} & \dots & \alpha_{3,n} \\ 0 & 0 & \ddots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \alpha_{n,n} \\ 0 & \dots & \dots & 0 & \alpha_{n+1,n} \end{bmatrix}}_{\underline{H}_n}.$$

When $\alpha_{n+1,n} = 0$ (breakdown), $AV_n = V_n H_n$, where H_n is $\underline{H}_n$ without the last row. This is an invariant subspace relation.

Note that $H_n = V_n^* A V_n = A_n$. The matrix H_n plays a double role here: it gives the action of $A : K_j \to K_{j+1}$ on the Krylov subspace, and it is a projected version of A.

9.4 Extras: rational Arnoldi [Güttel '13]

Idea: we can use a variant of Arnoldi to compute bases for other spaces related to rational functions rather than polynomials in A. In particular, we wish to replace the polynomials $p(z)$ that appear in Krylov spaces with rational functions with a fixed denominator $q(z)$ of degree $n-1$: this leads to the definition of the rational Krylov subspaces

$$\begin{aligned} K_{q,n}(A,b) &= q(A)^{-1}K_n(A,b) = K_n(A, q(A)^{-1}b) \\ &= \{r(A)b \colon r(z) = p(z)/q(z),\ p \text{ any polynomial of degree } d < n\} \end{aligned}$$

It is important that the denominator $q(z)$ here is the same one for all vectors, because otherwise we would not get a vector space: the sum of two rational functions $p_1(z)/q_1(z) + p_2(z)/q_2(z)$ is a rational function of degree larger than that of the two summands, in general.

In the following, we shall write q as a product of linear factors

$$q(z) = (z - \xi_1)(z - \xi_2)\dots(z - \xi_{n-1}).$$

The ξ_j are called poles, since they are the poles of the rational function $p(z)/q(z)$. When the poles ξ_j are distinct, for any $p(z)$ of degree $d \leq n$ one can find coefficients γ_j such that

$$\frac{p(z)}{q(z)} = \gamma_0 + \frac{\gamma_1}{z - \xi_1} + \frac{\gamma_2}{z - \xi_2} + \dots + \frac{\gamma_{n-1}}{z - \xi_{n-1}};$$

this is often a convenient representation to work with.

Rational Arnoldi A basis for $K_{q,n}(A, b)$ can be computed with a loop similar to the classical Arnoldi one: we construct a sequence of nested bases for

$$\operatorname{span}(b) = K_{1,1}(A, b) \subseteq K_{(z-\xi_1),2}(A, b) \subseteq K_{(z-\xi_1)(z-\xi_2),3}(A, b) \subseteq \dots.$$

In each iteration, we add one more pole ξ_i, and at the same time we increase by 1 the degree of the numerator.

The only detail we need to change with respect to classical Arnoldi is the choice of the "continuation vector" $w = Av_j$.

To add a pole ξ_j, we choose w of the form

$$w = (A - \xi_j I)^{-1} v \in K_{q(z)(z-\xi_j),j+1}(A, b) \setminus K_{q(z),j}(A, b),$$

where v is a suitable vector in $K_{q(z),j}(A, b)$. We do not go into detail on how one can find such a v; we just note that, if ξ_j is different from all previous poles, we can take $v = b$: this follows from the fact that $\frac{1}{z-\xi_j}$ does not belong to the space of rational functions with denominator $q(z)$.

One can also combine steps of rational Arnoldi and classical Arnoldi, i.e., take $w = Av$ at certain iterations and $w = (A - \xi_j I)v$ in certain others. A step of classical Arnoldi has the effect of raising by 1 the allowed degree of the numerator j, while not raising the degree of the denominator $q(z)$; it can be interpreted as adding a "pole at infinity", projectively.

Putting together all orthogonalization relations yields an equality of the form

$$AV_{n+1}\underline{K}_n = V_{n+1}\underline{H}_n.$$

Again, it is not immediate to obtain an expression for $A_n = V_n^* A V_n$ from $\underline{K}_n$ and $\underline{H}_n$; we will not go into details on how to do it.

Many of the results that we have proved for classical Arnoldi continue to hold, replacing polynomials with rational functions. In particular, we can consider the approximation

$$f(A)b \approx V_n f(A_n) V_n^* b = V_n f(A_n) V_n^* b, \quad A_n = V_n^* A V_n. \tag{9.1}$$

Lemma 9.7. *Let V_n be the basis matrix produced by rational Arnoldi, which we suppose to have full rank. If f is a rational function with denominator $q(z)$, then*

$$f(A)b = V_n f(A_n) V_n^* b.$$

Proof. Use $K_{q,n}(A,b) = K_n(A, q(A)^{-1}b)$ to reduce to the previous case. $\square$

Costs and benefits With Rational Arnoldi, one needs to solve several linear systems with $(A - \xi_i I)^{-1}$, so a sparse LU is needed for each (distinct) pole. This is significantly more expensive than Arnoldi, which only relies on matrix products, but it can be compensated by having additional degrees of freedom in the choice of the poles.

The key issue is how much more effective is *rational interpolation* (for a given f and A) than *polynomial interpolation*, so that this trade-off is convenient. How to choose good poles ξ_j?

There are many classical and current research results on these aspects. No details here; I am not an expert myself. For a good overview, check the review paper [Güttel '13].

Matlab examples Using Rktoolbox by S. Güttel `http://guettel.com/rktoolbox/`.

```
>> rng(0); A = randn(100) + 10*eye(100);
>> v = eig(A); plot(real(v), imag(v), 'x');
>> b = randn(size(A,1), 1);
>> poles = [-20:-1, inf]; % inf as last pole
>> [V, K, H] = rat_krylov(A, b, poles);
>> An = H(1:end-1,:) / K(1:end-1,:);
>> v = eig(A); w = eig(An);
>> plot(real(v), imag(v), 'x', real(w), imag(w), 'o');
>> c = V(:, 1:end-1)*expm(An) * V(:, 1:end-1)'*b;
>> norm(expm(A)*b - c) / norm(c)
```

Try again with `poles = [21:40, inf]`, or `inf*ones(1,21)` (classical Arnoldi), `[0*ones(1,10), inf*ones(1,10)]` (extended Arnoldi), ...

The choice of poles directs which eigenvalues are best approximated and influences performance greatly.

Chapter 10

Lyapunov equations

Before turning to control systems, we go back to matrix equations, and study in more detail a special case of the Sylvester equation.

Given $A, Q \in \mathbb{C}^{n\times n}$ with $Q = Q^* \succeq 0$, the Lyapunov equation is the matrix equation

$$A^*W + WA + Q = 0 \tag{10.1}$$

in the unknown $W \in \mathbb{C}^{n\times n}$. From the theory of Sylvester equations, we already know that there is a unique solution if and only if $\Lambda(A^*) \cup \Lambda(-A) = \emptyset$. An important case when this holds if when $\Lambda(A) \subset LHP$ (the open left half-plane). Matrices that have all their eigenvalues in the left half-plane are called Hurwitz stable.

We start by showing a few properties of the solution.

Lemma 10.1. *Suppose* (16.2) *has a unique solution W; then W is Hermitian.*

Proof. Transpose everything; W^* is another solution. □

Lemma 10.2. *Suppose $\Lambda(A) \subset LHP$ (open). Then, the (unique) solution of* (16.2) *can be written as*

$$W = \int_0^\infty e^{A^*t} Q e^{At}\,\mathsf{d}t. \tag{10.2}$$

Proof. First of all, note that the integral converges: since $\Lambda(A) \subset LHP$, $\exp(tA) \to 0$ (as shown in Exercise 2.6), and we can show that the decrease is exponential: $\exp(tA) = O(e^{t\alpha})$, where $\alpha = \max_{\lambda\in\Lambda(A)} \operatorname{Re}(\lambda)$ (the so-called spectral abscissa of A).

To prove the formula, compute $\frac{\mathsf{d}}{\mathsf{d}t} e^{A^*t}Qe^{At} = A^*e^{A^*t}Qe^{At} + e^{A^*t}Qe^{At}A$, then integrate both sides. □

Lyapunov equation: positivity

Lemma 10.3. *Suppose $\Lambda(A) \subset LHP$ (open). Then, $Q \succeq 0$ implies $W \succeq 0$, and $Q \succ 0$ implies $W \succ 0$.*

Proof. This follows from the integral formula (10.2). □

Lemma
Suppose $Q \succ 0$ and $W \succ 0$. Then, $\Lambda(A) \subset LHP$.

Proof. Let $Av = \lambda v$; then

$$0 < v^*Qv = -v^*(A^*W + WA)v = -(\overline{\lambda} + \lambda)v^*Wv;$$

hence, $2\operatorname{Re}(\lambda) = \overline{\lambda} + \lambda = -\frac{v^*Qv}{v^*Wv} < 0$. □

Remark 10.4. In this lemma, we cannot replace the $\succ$ symbols with $\succeq$. This is easy to see by considering a special case: obviously from $0 \cdot A + A \cdot 0 = 0$ we cannot deduce anything on A!

Relation to linear dynamical systems Consider the continuous-time linear dynamical system

$$\begin{cases} \dot{x}(t) = Ax(t), & x : [0, \infty] \to \mathbb{C}^n \\ x(0) = x_0. \end{cases}$$

We know that the solution to this ODE is $x(t) = \exp(At)x_0$. This system is called *asymptotically stable* if

$$\lim_{t\to\infty} x(t) = 0 \quad \text{for all choices of } x_0 \in \mathbb{C}^n,$$

and this happens if and only if $\Lambda(A) \subset LHP$.

In view of the lemmas above, it is sufficient to exhibit $W \succ 0$ such that $A^*W + WA \prec 0$ to prove that A has all its eigenvalues in the LHP and the system is asymptotically stable.

At the time of Lyapunov (1857–1918), doing this (together with factorizations to show that $W, Q \succ 0$) was easier than computing the full spectrum $\Lambda(A)$ (without a computer!).

Example 10.5. Consider the matrix

$$A = \begin{bmatrix} -1 & -2 & 0 \\ 0 & -2 & 1 \\ 1 & 0 & -3 \end{bmatrix}.$$

With Matlab, we can compute the eigenvalues and show that

```
>> eig(A)
ans =
   -3.5214 + 0.0000i
   -1.2393 + 0.8579i
   -1.2393 - 0.8579i
```

hence A is Hurwitz stable. Without a computer, however, it might be simpler to choose

$$W = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and compute the LDL factorization

$$Q = -(A^*W+WA) = \begin{bmatrix} 4 & 4 & -1 \\ 4 & 4 & -1 \\ -1 & -1 & 6 \end{bmatrix} = LDL^*, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -\frac{1}{4} & 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{23}{4} \end{bmatrix}.$$

Since we have $Q \succ 0, W \succ 0$, this shows that A is Hurwitz stable.

Remark One can also show directly that $x(t) \to 0$ without speaking about eigenvalues: note that the matrix W that solves (16.2) is an *energy function* for the system: if $V(x) = x^*Wx$, then $\frac{d}{dt}V(x(t)) < 0$, by direct verification. Hence $V(x(t))$ decreases. With a little more care with the bounds, we can show that it decreases exponentially, and in particular $V(x(t)) \to 0$, which implies $\|x(t)\| \to 0$.

Discrete-time version These results, as well as many of the following ones, also come in a discrete-time variant.

Discrete-time linear dynamical system

$$\begin{cases} x_0 \in \mathbb{C}^n \\ x_{k+1} = Ax_k, \quad k = 0, 1, 2, \dots \end{cases}$$

The system is *asymptotically stable*, i.e., $\lim_{k\to\infty} x_k = 0$ for each x_0, if and only if A has its eigenvalues in the *open unit disc* $\mathbb{D}$.

The analogue of the Lyapunov equation is the Stein equation

$$W - A^*WA = Q, \quad Q \succ 0 \tag{10.3}$$

If $W \succ 0$ solves (16.3), then $V(x) = x^*Wx$ is an *energy function*, i.e., $V(x_{k+1}) < V(x_k)$.

Lemma
$\Lambda(A) \subset \mathbb{D}$ *iff* (16.3) holds with $W, Q \succ 0$.

Proof Analogous to the continuous-time one. Closed formula:

$$W = \sum_{k=0}^{\infty} (A^*)^k Q A^k.$$

Proof. Vectorizing, (16.3) becomes $(I - A^T \otimes A^*)\operatorname{vec}(W) = \operatorname{vec}(Q)$. Then use the Neumann series $(I - M)^{-1} = I + M + M^2 + \dots$. □

Remark (16.3) can be solved with a Bartels-Stewart-like method. More generally, Bartels-Stewart-type methods can be obtained for all equations of the form $AXB + CXD = E$, using QZ factorizations of (A, C) and (D^T, B^T).

Chapter 11

Introduction to control theory

11.1 Examples of control systems

Control theory [Datta, Ch. 5] is the study of dynamical systems with controllers; it is an important topic in engineering.

Example can we keep an 'inverted pendulum' of length 1 in the unstable upright position (12 o' clock) by applying a steering force?

We suppose that the pendulum is a massless stiff bar of length 1 with with weight at the end, so that there is only one degree of freedom, the angle θ that the bar makes with the vertical (12 o' clock $\leftrightarrow \theta = 0$).

The equation of motion is $\ddot{\theta} = g \sin\theta \approx g\theta$. We can rewrite it in terms of the *state* $x(t) = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$, to obtain the matrix version

$$\dot{x} = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} x_2 \\ g x_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ g & 0 \end{bmatrix} x.$$

The system is not stable: $A = \begin{bmatrix} 0 & 1 \\ g & 0 \end{bmatrix}$ has one positive and one negative eigenvalue.

Example: controlling an inverted pendulum Now we apply an additional steering force u (*control*): we have $\ddot{\theta} = g\theta + u$, or in matrix form

$$\dot{x} = Ax + Bu, \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Can we choose $u(t)$ so that the system is stable? Yes: we can even choose one of the form $u(t) = Fx(t)$, $F \in \mathbb{R}^{1\times 2}$

We can literally build a contraption (engine + camera) that sets the appropriate force according to the current state only (*feedback control*). $u = \begin{bmatrix} f_1 & f_2 \end{bmatrix} x$ gives the *closed-loop system*

$$\dot{x} = (A + BF)x = \begin{bmatrix} 0 & 1 \\ f_1 + g & f_2 \end{bmatrix} x.$$

Choosing f_1, f_2, we can move the eigenvalues of $A + BF$ arbitrarily.

Remark: A (linear) 'controller' that observes only the position and not the velocity corresponds to $f_2 = 0$. It is easy to see that this is not enough to stabilize the system: if $f_2 = 0$, there is no choice of f_1 for which $\Lambda(A + BF) \subset LHP$.

Example: heating a long corridor with a window *Heat equation:* in a bar of uniform material (the segment $[0, 1]$), one endpoint 1 is kept at constant temperature 0°C, and we apply a variable temperature (amount of 'heat') $u(t)$ at the other endpoint 0.

The temperature $x(y, t)$ at position y and time t follows

$$\frac{\partial}{\partial t} x(y, t) = \alpha \frac{\partial^2}{\partial y^2} x(y, t), \quad x(0, t) = u(t),\, x(1, t) = 0.$$

We discretize in space: $x(t)$ is a vector of temperatures at equi-spaced points $h, 2h, \ldots, (n-1)h$ (those at 0 and $(n+1)h = 1$ are prescribed).

$$\frac{d}{dt} x(t) = Ax(t) + Bu(t),$$

$A = \alpha h^2 \operatorname{tridiag}(1, -2, 1)$, $B = \alpha h^2 e_1$.

Other examples in [Datta, Ch. 5], e.g. electrical circuits.

Another impressive example of a control system is the triple pendulum on a cart; see e.g. the video `youtu.be/cyN-CRNrb3E`. This is a system with 3 degrees of freedom.

11.2 Controllability

We now consider control systems in their standard form

$$\dot{x} = Ax + Bu, \quad A \in \mathbb{C}^{n \times n},\, B \in \mathbb{C}^{n \times m}.$$

There are two basic questions that we can ask ourselves:

Q1 Can we *stabilize* the system around 0, i.e., choose $u(t) = Fx(t)$ so that the system is asymptotically stable?

Q2 Can we *control* the system, i.e., choose $u(t)$ to reach a given value of $x(t_F)$ at a target time t_F?

Not always: counterexample:

$$\begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ 0 \end{bmatrix}. \tag{11.1}$$

No matter what $u(t)$ we choose, we cannot change the dynamics of the second block $x_2(t)$. If A_{22} has eigenvalues outside the LHP, the system will always be unstable.

We shall see that this is essentially the only case when a system is not controllable, but this structure may be hidden behind a change of basis for the state $A \leftarrow MAM^{-1}, B \leftarrow MB$, so it might be more difficult to identify. The key to analyze it is invariant subspaces: in the situation (11.1), all columns of B_1 belong to a nontrivial invariant subspace $\operatorname{Im}[\begin{smallmatrix} I \\ 0 \end{smallmatrix}]$ of A. The concept of invariant subspaces does not depend on the basis.

Hence, to identify a structure like (11.1) we can construct the smallest invariant subspace for A that contains the columns of B. We can give a formula for this subspace.

Lemma 11.1. *Let $A \in \mathbb{C}^{n\times n}, B \in \mathbb{C}^{n\times m}$. The smallest A-invariant subspace that contains the columns of B is*

$$K(A,B) := \operatorname{Im}[B, AB, A^2B, \dots].$$

Proof. It is simple to check that $K(A,B)$ is in fact invariant, and that every invariant subspace must contains the columns of $B, AB, A^2B, \dots$. $\square$

Note the connection with Krylov subspaces: if B is a single vector, $K(A,B)$ is the union of all Krylov subspaces $K_n(A,B)$.

(In fact, this K does not stand for Krylov but for Kalman, another key figure in control theory.)

Definition 11.2. The space $K(A,B)$ is called the controllability space of (A,B). A matrix pair $(A,B) \in \mathbb{C}^{n\times n} \times \mathbb{C}^{n\times m}$ is called controllable when $K(A,B) = \mathbb{C}^n$.

Properties

- Controllability depends only on $\operatorname{Im} B$, hence (A,B) controllable $\iff$ (A,BK) controllable, for any invertible K.
- Similarly, (A,B) controllable $\iff$ $(A,BR^{-1}B^*)$ controllable for any positive definite $R \in \mathbb{C}^{m\times m}$; we will use this property in future.
- (A,B) controllable $\iff$ $(A-\alpha I, B)$ controllable, since the powers of $A-\alpha I$ are linear combinations of the powers of A.

Controllability **[Datta, Ch. 6, with more streamlined proofs]** We shall show that indeed the controllability space reveals the structure we were interested in.

Lemma 11.3 (Kalman decomposition)**.** *For each pair (A, B), there exists a nonsingular $M \in \mathbb{C}^{n\times n}$ such that the following block decomposition holds, and (A_{11}, B_1) is controllable.*

$$M^{-1}AM = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \quad M^{-1}B = \begin{bmatrix} B_1 \\ 0 \end{bmatrix}.$$

The blocks must be of the same size, i.e., $B_1 \in \mathbb{C}^{n_1\times m}$ if $A_{11} \in \mathbb{C}^{n_1\times n_1}$.

Moreover, $n_1 = \dim K(A, B)$, and in particular we have $n_1 = n$ if and only if (A, B) is controllable.

Proof. It is sufficient to take $M = [\, M_1 \; M_2 \,]$ such that M_1 is a basis of $K(A, B)$; then, the blocks B_2 and A_{21} must be zero, because $K(A, B)$ contains the columns of B and is A-invariant.

If (A_{11}, B_1) were not controllable, then $K(A_{11}, B_{11})$ (extended with zeros) would be a smaller invariant subspace of $M^{-1}AM$ that contains the columns of $M^{-1}B$, contradicting minimality. □

We now need to prove that this concept that we dubbed <u>controllability</u> of a matrix pair is indeed related to the controllability of the dynamical system $\dot{x} = Ax + Bu$.

Theorem 11.4. *The following are equivalent.*

1. *The system $\dot{x} = Ax + Bu$, $x(0) = x_0$ is controllable, i.e., given any target state x_F and time t_F we can choose a control function $u(t)$ such that $x(t_F) = x_F$.*
2. *The pair (A, B) is controllable.*
3. *The matrix*
$$W_t = \int_0^t \exp(A\tau)BB^* \exp(A^*\tau)d\tau$$
is invertible (for a specific $t > 0$, or, equivalently, for all of them).

Before starting the proof, we remark that the expression of W_t resembles the integral formula for the solution of the Lyapunov equation that we have proved earlier. Indeed, we see that $W = \lim_{t\to\infty} W_t$. In particular, this equivalence shows that $W \succ 0$ if and only if (A, B) is controllable.

Proof. $1 \implies 2$ Suppose, by contradiction, that $K(A, B)$ is not the whole space.

Recall the ugly closed formula for the solution of a linear differential equation $\dot{x}(t) = Ax(t) + f(t)$, where in our case $f(t) = Bu(t)$. We have

$$x(t) = \exp(At)x_0 + \int_0^t \exp(A(t-\tau))Bu(t)dt. \tag{11.2}$$

Since $\exp(A(t-\tau))$ is a matrix function and hence a polynomial in A, one sees that the integral always takes values in $K(A,B)$. Hence, independently of $u(t)$, we cannot obtain all possible values of $x(t)$; it is sufficient to take x_F such that

$$x_F - \exp(At_F)x_0 \notin K(A,B)$$

to get a vector that cannot be reached.

2 $\implies$ 3 Suppose $W_t v = 0$ for a certain $t > 0$. Then, $0 = v^* W_t v = \int_0^t \|v^* \exp(A\tau)B\|^2 dt$. This must mean that the (continuous) function $\phi(\tau) = v^* \exp(A\tau)B = 0$. Hence, in particular.

$$\begin{aligned} 0 &= \phi(0) = v^*B, \\ 0 &= \phi'(0) = v^*AB, \\ 0 &= \phi''(0) = v^*A^2B, \\ &\vdots \quad \vdots \end{aligned}$$

and this shows that $v^*[B, AB, A^2B, \dots] = 0$, so the controllable space is not the whole $\mathbb{C}^n$. 3 $\implies$ 1 Take a control $u(t)$ of the form

$$u(t) = B^* \exp(A^*(t_F - t))y,$$

with $y \in \mathbb{C}^n$. Plugging it into (11.2), and recognizing the matrix W_t inside the expression (up to a change of variables $\tau = t_F - t$), we obtain

$$x(t_F) = \exp(At_F)x_0 + W_{t_F}y.$$

Since W_{t_F} is invertible, with a suitable choice of y we can obtain any value of $x(t_F) \in \mathbb{C}^n$. □

Other controllability criteria

Lemma 11.5 (Popov (or Hautus) criterion)**.** (A,B) *controllable* $\iff$ $\operatorname{rank}[A - \lambda I,\, B] = n$ *for all* $\lambda \in \Lambda(A)$ $\iff$ $\operatorname{rank}[A-\lambda I, B] = n$ *for all* $\lambda \in \mathbb{C}$*.*

It is sufficient to test the condition on $\lambda \in \Lambda(A)$, because for all other λs we already have $\operatorname{rank}(A - \lambda I) = n$.

Proof. $\Leftarrow$ We can assume (up to a change of basis) that (A,B) is in a Kalman decomposition, with a non-trivial block A_{22}. Take a left eigenpair $v^*A_{22} = \lambda v^*$: then, $[0,\, v^*][A-\lambda I,\, B] = 0$.

$\Rightarrow$ If $v^*[A - \lambda I, B] = 0$ for some $\lambda \in \Lambda(A)$, then we get $0 = v^*B = v^*AB = v^*A^2B = \dots$, and hence $K(A,B) \neq \mathbb{C}^n$ as in the proof of the previous theorem. □

How to test controllability numerically? Numerically, almost any pair (A, B) is controllable, exactly like almost any $n \times n$ matrix is singular: a certain quantity needs to be exactly zero for controllability to fail.

Anyway, various options:

- Compute $\operatorname{rank}[B, AB, A^2B, \dots, A^{n-1}B]$. We can stop at $n-1$, because A^n is a linear combination of $I, A, A^2, \dots, A^{n-1}$ by the Cayley-Hamilton theorem.
- If B is a single vector, you can also run a Krylov algorithm until the final iteration, and check for breakdown.
- Compute $\Lambda(A)$ and check that $\operatorname{rank}[A - zI, B] = n$ for each $z \in \Lambda(A)$.
- Assume (up to replacing it with $A - \alpha I$) that $\Lambda(A) \subset LHP$. Solve the Lyapunov equation $AW + WA^* + BB^* = 0$ and check if $W \succ 0$.

Note that all these methods rely on a *rank decision*: are certain singular values, or certain computed values, zero or not?

Remark There are methods to compute the distance of a certain matrix pair (A, B) to the nearest uncontrollable pair, exactly like the condition number of a matrix M is a measure of the distance of M to the nearest singular matrix. They are somewhat more complex, and research is still active on the best one.

Remark The criterion with the Lyapunov equation actually corresponds to a physical quantity: $x_0^* W^{-1} x_0$ is the minimal amount of *energy* $\int_0^{t_F} u(\tau)^* u(\tau) d\tau$ that we need to reach $x(t_F) = 0$ starting from $x(0) = x_0$. We won't prove it here. Hence, the closer to uncontrollable a system is, the more energy you need to put in to actually control it.

(Matlab examples: construct a numerically non-controllable (A, B) from a Kalman decomposition, and apply the various methods.)

11.3 Stabilizability

If a system is controllable, then it is also stabilizable: we can find F such that $\Lambda(A + BF) \subset LHP$. The following result gives us a practical way to compute one such choice of F.

Theorem (Bass algorithm)
Let (A, B) be controllable, $\alpha > \rho(A)$, and W the solution of

$$(-A - \alpha I)W + W(-A - \alpha I)^* + 2BB^* = 0. \tag{11.3}$$

Then, $W \succ 0$ and $F = -B^* W^{-1}$ is a stabilizing feedback.

Note that $(-A - \alpha I, B)$ is controllable because (A, B) is so, that $\Lambda(-A - \alpha I) \subset LHP$, and that $Q := 2BB^* \succeq 0$. By Lyapunov eq. results, this implies $W \succ 0$. Rearranging (11.3) gives

$$(A + BF)W + W(A + BF)^* + 2\alpha W = 0.$$

By Lyapunov eq. results, $W \succ 0$, $Q := 2\alpha W \succ 0$ implies $\Lambda(A+BF) \subset LHP$.

Remark We can actually find F such that $A+BF$ has any chosen spectrum. (We won't prove it here.) [Datta, Ch. 11]

Stabilizability Sometimes, even if a system is not controllable, we can still ensure that the solution converges to 0. Example: take a system already in Kalman decomposition

$$A = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \quad B = \begin{bmatrix} B_1 \\ 0 \end{bmatrix}, \quad x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

with $\Lambda(A_{22}) \subset LHP$. Then, the control does not act on $x_2(t)$, but $x_2(t) \to 0$ already by itself!

To stabilize this system with a feedback control, just take $F = [F_1 \, 0]$, where F_1 is chosen so that $\Lambda(A_{11} + B_1 F_1) \subset LHP$ (it exists because (A_{11}, B_1) is controllable by definition of Kalman decomposition).

Stabilizability conditions **Theorem**
The following conditions are equivalent; if they hold, (A, B) is called *stabilizable*.

1. $\Lambda(A_{22}) \subset LHP$ in the Kalman decomposition;
2. $\operatorname{rk}[A - \alpha I, B] = n$ for all $\alpha \notin LHP$;
3. We can find $u(t)$ such that $\lim_{t\to\infty} x(t) = 0$;
4. We can find F such that $\Lambda(A+BF) \subset LHP$ (hence we can take $u(t) = Fx(t)$ to satisfy the previous point).

We won't see a full proof, but it mostly follows from things we have already stated.

Some Matlab examples

```
% the inverted pendulum
% (dubious example because the true sys is nonlinear)
A = [0 1; 1 0]; B = [0;1]; x0 = [0.1; -0.05];

% open-loop system
[t, x] = ode45(@(t,x) A*x, [0,5], x0);
plot(t,x);

% feedback that observes the position and tries to push it back
F = [-1.5 0];
[t, x] = ode45(@(t,x) (A+B*F)*x, [0,5], x0);
plot(t,x);
```

```
% random feedback (try several)
F = randn(1,2);
[t, x] = ode45(@(t,x) (A+B*F)*x, [0,5], x0);
plot(t,x);
```

```
% heat equation on a steel bar
n = 10; y = linspace(h,n*h,n);
h = 1/(n+1); x0 = rand(n, 1);

A = h^2*(-2*eye(n) + diag(ones(n-1,1),1) ...
     + diag(ones(n-1,1),-1));
B = h^2*eye(n,1);

% open-loop system
[t, x] = ode45(@(t,x) A*x, [0,1000], x0);
surf(y, t, x);

% constant stream of heating
[t, x] = ode45(@(t,x) A*x + B*1, [0,1000], x0);
surf(y, t, x);
```

```
% feedback
F = rand(1,n);
[t, x] = ode45(@(t,x) A*x + B*F*x, [0,1000], x0);
surf(y, t, x);

% thermal sensor midway along the bar
F = zeros(1,n); F(end/2) = -1000*h^2;
[t, x] = ode45(@(t,x) A*x + B*F*x, [0,1000], x0);
surf(y, t, x);

% thermal sensor with wrong sign
F = zeros(1,n); F(end/2) = 1000*h^2;
[t, x] = ode45(@(t,x) A*x + B*F*x, [0,1000], x0);
surf(y, t, x);
```

```
% controlling to a specified position
tf = 1000; xf = rand(n,1);

Winf = lyap(A, B*B');
W = integral(@(t) expm(A*(tf-t))*B*B'*expm(A'*(tf-t)), ...
    0, tf, 'ArrayValued', true);
eig(Winf), eig(W) % system barely controllable

y = W \ (xf - expm(A*tf)*x0);
```

```
[t, x] = ode45(@(t,x) A*x + B*B'*expm(A'*(tf-t))*y, ...
    [0,1000], x0);
surf(y, t, x); % hard-to-control system

[x(end,:); xf'] % not too accurate!

[t, x] = ode45(@(t,x) A*x + B*B'*expm(A'*(tf-t))*y, ...
   [0,1000], x0, odeset('RelTol', 1e-8, 'AbsTol', 1e-10));
[x(end,:); xf'] % but it was just an ode45 accuracy issue
```

```
% Bass's algorithm
alpha = 1.1*max(abs(eig(A)))
W = lyap(-A-alpha*eye(n), 2*B*B')
F = -B'/W; eig(A+B*F) %all in LHP!

[t, x] = ode45(@(t,x) A*x + B*F*x, [0,1000], x0);
surf(y, t, x);
```

Chapter 12

Optimal control

Optimal control Several choices available for stabilizing feedback F: for instance, you can choose different α's in Bass algorithm.

Is there an 'optimal' one? One possible way to formalize this: the control that uses the minimum *energy*, defined by a quadratic form ($R \succeq 0$, $Q \succeq 0$).

Linear-quadratic optimal control
Find $u : [0, \infty) \to \mathbb{C}^m$ (piecewise C^0, let's say) that minimizes

$$V(u) = \int_0^\infty x^* Q x + u^* R u \, \mathsf{d}t$$
$$\text{s.t. } \dot{x} = Ax + Bu, \, x(0) = x_0, \, \lim_{t\to\infty} x(t) = 0.$$

We assume here that $R \succ 0$: control is never free. Optimal control becomes a trickier problem otherwise.

Linear-quadratic regulator theorem [Datta, Thm 10.5.1] A solution follows from calculus of variations principles; here is a self-contained version.

Theorem
Let $Q \succeq 0$, $R \succ 0$, (A, B) controllable. Set $G = BR^{-1}B^* \succeq 0$.
There exists a unique $X = X^* \in \mathbb{C}^{n\times n}$ such that

1. $A^*X + XA + Q - XGX = 0$,
2. $\Lambda(A - GX) \subset LHP$.

The optimal value of the minimum problem

$$\min \int_0^\infty x(t)^* Q x(t) + u(t)^* R u(t) \, \mathsf{d}t,$$
$$\text{s.t. } \dot{x}(t) = Ax(t) + Bu(t), \quad \lim_{t\to\infty} x(t) = 0$$

is $x_0^* X x_0$, attained with the feedback control $u(t) = Fx(t)$ obtained with $F = -R^{-1}B^* X$.

Note that indeed $A + BF = A - GX$ is stable by the conditions we imposed on X. The equation

$$A^*X + XA + Q - XGX = 0$$

is called <u>continuous-time algebraic Riccati equation</u>, and X that satisfies 1-2 is called its <u>stabilizing solution</u>.

Proof. Proving the existence of X with those properties will be long, and it is the topic of the rest of this chapter. We shall now conclude assuming it exists.

Note that $\Lambda(A - GX) \subset LHP$ implies $\lim_{t\to\infty} x(t) = 0$, so this u is admissible. Take a generic stabilizing control u, and compute

$$\begin{aligned}\frac{\mathsf{d}}{\mathsf{d}t} x^*Xx &= \dot{x}^*Xx + x^*X\dot{x}\\ &= (Ax + Bu)^*Xx + x^*X(Ax + Bu)\\ &= x^*(A^*X + XA)x + u^*B^*Xx + x^*XBu\\ &= x^*(XBR^{-1}B^*X - Q)x + u^*B^*Xx + x^*XBu\\ &= \underbrace{(u + R^{-1}B^*Xx)^*R(u + R^{-1}B^*Xx)}_{\geq 0} - x^*Qx - u^*Ru.\end{aligned}$$

Integrating from 0 to ∞,

$$\int_0^\infty x^*Qx + u^*Ru \, \mathsf{d}t \geq x_0^*Xx_0 - \underbrace{x(\infty)^*Xx(\infty)}_{=0},$$

with equality if $u + R^{-1}B^*Xx \equiv 0$. □

Riccati equation and subspaces The equation

$$A^*X + XA + Q - XGX = 0, \quad Q \succeq 0, \, G \succeq 0$$

is called *algebraic Riccati equation* (ARE). It is an *invariant subspace problem* in disguise, because if X satisfies the equation then

$$\begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix} \begin{bmatrix} I \\ X \end{bmatrix} = \begin{bmatrix} I \\ X \end{bmatrix} (A - GX).$$

Hence the problem of finding X can be recast as finding a suitable invariant subspace of $\mathcal{H}$. The road to proving the existence of our solution X passes through studying the properties of the matrix $\mathcal{H}$.

Hamiltonian matrices A matrix of the form

$$\mathcal{H} = \begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix}, \quad Q = Q^*, G = G^*$$

is called <u>Hamiltonian matrix</u>.

Lemma 12.1. *Let $\mathcal{H}$ be Hamiltonian, and $\lambda \in \Lambda(\mathcal{H})$. Then, $-\overline{\lambda} \in \Lambda(\mathcal{H})$, too, and the two have the same multiplicity.*

In other words, the spectrum of $\mathcal{H}$ is symmetric with respect to the imaginary axis.

Proof. Let $J = \begin{bmatrix} & I \\ -I & \end{bmatrix}$. One can verify directly the equality $J^{-1}\mathcal{H}J = -\mathcal{H}^*$. Hence, $\mathcal{H}$ and $-\mathcal{H}^*$ are similar, and they have the same spectrum (also counted with multiplicities). □

We can say more.

Theorem 12.2. *Assume $Q \succeq 0$, $G = BR^{-1}B^* \succeq 0$, (A, B) (or, equivalently, (A, G)) stabilizable, and (A^*, Q) stabilizable. Then, $\mathcal{H}$ has no eigenvalues with* $\operatorname{Re}\lambda = 0$.

Proof. Suppose instead $\mathcal{H}\left[\begin{smallmatrix} z_1 \\ z_2 \end{smallmatrix}\right] = \imath\omega\left[\begin{smallmatrix} z_1 \\ z_2 \end{smallmatrix}\right]$. Writing the blocks out explicitly, we get

$$\begin{aligned} Az_1 - Gz_2 &= \imath\omega z_1, \\ -Qz_1 - A^*z_2 &= \imath\omega z_2. \end{aligned}$$

We can eliminate $A - \imath\omega$ from these equations: multiply the first equation by z_2^*, transpose the second, and multiply it by z_1. Then we are left with

$$z_1^*Qz_1 + z_2^*Gz_2 = 0.$$

Since Q and G are positive semidefinite, it must be the case that $Qz_1 = Gz_2 = 0$. Substituting it above, we get $(A - \imath\omega I)z_1 = 0$, $(A - \imath\omega)^*z_2 = 0$. We then obtain

$$z_1^*[A^* + \imath\omega \quad Q] = 0, \quad z_2^*[A - \imath\omega \quad G] = 0.$$

Since at least one of z_1 and z_2 is nonzero, we contradict one of the two stabilizability conditions (Popov test). □

Hence, $\mathcal{H}$ has n eigenvalues in the LHP and n in the RHP, counted with multiplicity. In particular, it has a (unique) n-dimensional invariant subspace associated to its eigenvalues in the LHP, i.e., there is $U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \in \mathbb{C}^{2n \times n}$ such that

$$\begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \mathcal{S}, \quad \Lambda(\mathcal{S}) \subset LHP. \tag{12.1}$$

We call this invariant subspace the <u>stable invariant subspace</u>. We can prove a particular property.

Lemma 12.3. *Let $\mathcal{H}$ be Hamiltonian, and $\begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$ be a basis matrix for its stable invariant subspace. Then,*

$$\begin{bmatrix} U_1 \\ U_2 \end{bmatrix}^* J \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} = U_2^* U_1 - U_1^* U_2 = 0.$$

Proof. Consider a Jordan basis for $\mathcal{H}$, partitioned into a set of Jordan chains in the LHP and one in the RHP.

$$\mathcal{H} = V \begin{bmatrix} J_{LHP} & 0 \\ 0 & J_{RHP} \end{bmatrix} V^{-1}. \tag{12.2}$$

Note that the first n columns of V are a basis for the stable invariant subspace of $\mathcal{H}$. By transposing and negating (12.2), one sees that the $\underline{\text{last}}$ n rows of V^{-1} are a basis for the stable invariant subspace of $-\mathcal{H}^*$. In particular, these two subspaces must be orthogonal, because $V^{-1}V = I$.

We know that U is a basis for the stable invariant subspace of $\mathcal{H}$, and, thanks to the relation $J^{-1}\mathcal{H}J = -\mathcal{H}^*$, we see that JU is a basis for the stable invariant subspace of $\mathcal{H}^*$. Hence, in particular, U and JU are orthogonal. □

A subspace such that U and JU are orthogonal is called $\underline{\text{Lagrangian subspace}}$.

Existence of X We are now very close to proving the existence of X. If we can prove that U_1 is invertible, then we can take a different basis

$$\begin{bmatrix} U_1 \\ U_2 \end{bmatrix} U_1^{-1} = \begin{bmatrix} I \\ U_2 U_1^{-1} \end{bmatrix}$$

for that invariant subspace, and get (with $X = U_2 U_1^{-1}$)

$$\begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix} \begin{bmatrix} I \\ X \end{bmatrix} = \begin{bmatrix} I \\ X \end{bmatrix} \widehat{\mathcal{S}}, \quad \widehat{\mathcal{S}} = U_1 \mathcal{S} U_1^{-1}. \tag{12.3}$$

Expanding out the blocks we get

$$-Q - A^*X = X\widehat{\mathcal{S}} = X(A - GX),$$

which is the Riccati equation, and $\Lambda(A - GX) = \Lambda(\widehat{\mathcal{S}}) \subset LHP$.

Theorem 12.4. *Suppose (A, B) and (A^*, Q) stabilizable, $Q \succeq 0$, $G \succeq 0$. Then, U_1 is invertible.*

Proof. The key is proving that $\ker U_1$ is an invariant subspace for $\mathcal{S}$. Let $v \in \ker U_1$,

$$-v^* U_2^* G U_2 v = \begin{bmatrix} v^* U_2^* & 0 \end{bmatrix} \mathcal{H} \begin{bmatrix} 0 \\ U_2 v \end{bmatrix} = v^* \underbrace{\begin{bmatrix} U_2^* & -U_1^* \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}}_{=0} \mathcal{S} v = 0$$

implies $GU_2v = 0$. Then looking at the first block row of

$$\begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix} \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} v = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix} \mathcal{S}v$$

we get $U_1\mathcal{S}v = 0$ as needed.

If $\ker U_1$ is nontrivial, we can find v, $\lambda \in LHP$ such that $U_1v = 0$, $\mathcal{R}v = \lambda v$. Now the second block row gives $-A^*U_2v = \lambda U_2v$. This (together with $GU_2v = 0$ from above) contradicts stabilizability. □

Symmetry of the solution

$$X^* - X = U_1^{-*}U_2^* - U_2U_1^{-1} = U_1^{-*}(U_2^*U_1 - U_1^*U_2)U_1^{-1} = 0.$$

Positive definiteness of the solution Note that

$$ARE \iff (A - GX)^*X + X(A - GX) + Q + XGX = 0.$$

So X solves the Lyapunov equation

$$\hat{A}^*X + X\hat{A} + \hat{Q} = 0, \quad \hat{A} = A - GX,\, \hat{Q} = Q + XGX.$$

And we know that $\Lambda(\hat{A}) \subset LHP, \hat{Q} \succeq 0 \implies X \succeq 0$.

Under slightly stronger assumptions one can also show that $(\hat{A}^*, \hat{Q})$ controllable $\implies X \succ 0$.

Factorization Once we know X exists, we can write the factorization

$$\begin{bmatrix} I & 0 \\ -X & I \end{bmatrix} \mathcal{H} \begin{bmatrix} I & 0 \\ X & I \end{bmatrix} = \begin{bmatrix} A - GX & -G \\ 0 & -(A - GX)^* \end{bmatrix},$$

which displays clearly the eigenvalue pairing $\lambda, -\overline{\lambda}$.

How to solve Riccati equations

- Newton's method (historically the first option).
- Invariant subspace computation: via unstructured methods (QR), 'semi-structured' methods (Laub trick), or fully structured methods (URV).
- Sign iteration (and variants).

Chapter 13

Newton's method for AREs

Historically, the first method used to solve algebraic Riccati equations is the Newton method.

Each iterate of Newton's method is a Lyapunov equation:

$$F(X) = A^*X + XA + Q - XGX$$

$$L_{F,X}(E) = A^*E + EA - EGX - XGE = E(A - GX) + (A - GX)^*E.$$

$$K_{F,X} = (A - GX)^T \otimes I + I \otimes (A - GX)^*.$$

If X_* is the stabilizing solution then $\Lambda(A - GX_*) \subset LHP \implies L_{F,X_*}$ is nonsingular.

Newton's method
For $k = 0, 1, 2, \ldots$

1. Solve the Lyapunov equation $E(A - GX_k) + (A - GX_k)^*E = F(X_k)$ for E;
2. Set $X_{k+1} = X_k - E$.

Newton's method Note that $H(A - GX_k) + (A - GX_k)^*H = F(X_k)$ is equivalent to

$$X_{k+1}(A - GX_k) + (A - GX_k)^*X_{k+1} = -Q - X_kGX_k \preceq 0.$$

If $\Lambda(A - GX_k) \subset LHP$, then $X_{k+1} \succeq 0$, by the results on Lyapunov equations.

Actually, something stronger holds.

Theorem
Suppose X_0 is chosen such that $\Lambda(A - GX_0) \subset LHP$. Then, $X_1 \succeq X_2 \succeq X_3 \succeq \cdots \succeq X_* \succeq 0$. Moreover, $X_k \to X_*$ quadratically.

Remark 13.1. The thesis does not include $X_0 \succeq X_1$: anything could happen in the first iteration!

The proof is tedious, using many times the lemmas relating Lyapunov equations, stability, and positive definiteness. We give only a sketch.

Proof. (sketch) Coupled induction. Set $A_k := A - GX_k$. Some algebra gives

$$(X_k - X_{k+1})A_k + A_k^*(X_k - X_{k+1}) = -(X_k - X_{k-1})G(X_k - X_{k-1})$$
$$(X_* - X_{k+1})A_k + A_k^*(X_* - X_{k+1}) = -(X_* - X_k)G(X_* - X_k)$$

hence A_k stable $\implies X_k \succeq X_{k+1} \succeq X_*$.

$$\begin{aligned}(X_{k+1} - X_*)A_{k+1} + A_{k+1}^*(X_{k+1} - X_*)\\ = -(X_{k+1} - X_k)G(X_{k+1} - X_k) - (X_{k+1} - X_*)G(X_{k+1} - X_*)\end{aligned}$$

This does not prove immediately that A_{k+1} is stable, because the RHS is not $\prec 0$; but plugging in $A_{k+1}v = \lambda v$ with $\operatorname{Re}\lambda \geq 0$ we get $B(X_{k+1} - X_k)v = 0$, hence also $A_k v = \lambda v$. $\square$

```
function X = care_newton(A, G, Q, k, X0)
% k steps of Newton's method to s, starting from X0
X = X0;
for it = 1:k
    F = A'*X+X*A+Q-X*G*X;
    E = lyap((A-G*X)', -F);
    X = X - E;
end
```

Newton: wrap-up To solve a Riccati equation with this method, we can proceed as follows.

Algorithm

- Use Bass's algorithm to find X_0 such that $A - GX_0$ is stable
- Run Newton iterations until convergence.

This method is expensive: each iteration requires a Schur form.

Standard results on the quadratic convergence of the multivariate Newton method hold: if the solution is simple (which is the case whenever the Hamiltonian has no imaginary eigenvalues $\iff$ $L_{F,X}$ is invertible), then $\|X_* - X_{k+1}\| \sim \|X_* - X_k\|^2$.

Defect correction / iterative refinement Newton's method is expensive on its own, but one way we can use it is to improve the quality of an approximate solution computed with another method. This typically requires only 1-2 iterations, thanks to its quadratic convergence.

Chapter 14

Invariant subspace methods for CAREs

Invariant subspace methods for CAREs Recall that$X$ solves the CARE $A^*X + XA + Q = XGX$ if and only if

$$\begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix} \begin{bmatrix} I \\ X \end{bmatrix} = \begin{bmatrix} I \\ X \end{bmatrix} \mathcal{R}, \quad \mathcal{R} = A - GX.$$

One can find X through an invariant subspace of the Hamiltonian.

```
function X = care_schur(A, G, Q)
H = [A -G; -Q -A'];
[U, T] = schur(H, 'complex');
[U2, T2] = ordschur(U, T, 'lhp');
n = size(A,1);
X = U2(n+1:2*n, 1:n) / U2(1:n, 1:n);
```

We show an example, which relies on the `carex` test suite, a collection of benchmark examples for algebraic Riccati equations.

```
>> [A,G,Q] = carex(4);
>> X = care_schur(A, G, Q)
X =
    0.8919 0.7366 0.6023 0.5212 0.5929 0.3488 0.2199 0.1415
    0.7366 1.3795 1.0765 0.8039 0.7005 0.5191 0.3348 0.1744
    0.6023 1.0765 1.4920 1.0138 0.8014 0.7435 0.4192 0.2031
    0.5212 0.8039 1.0138 1.1488 0.7327 0.5313 0.3410 0.1732
    0.5929 0.7005 0.8014 0.7327 0.5921 0.4293 0.2847 0.1476
    0.3488 0.5191 0.7435 0.5313 0.4293 0.3553 0.2377 0.1241
    0.2199 0.3348 0.4192 0.3410 0.2847 0.2377 0.1965 0.1024
    0.1415 0.1744 0.2031 0.1732 0.1476 0.1241 0.1024 0.0795
>> norm(A'*X + X*A + Q - X*G*X) / (norm(A'*X) + norm(X*A) + norm(Q) + norm(X*G*X))
ans =
    3.4242e-15
>> max(real(eig(A - G*X)))
ans =
    -0.1006
```

Note that the method produces a symmetric stabilizing solution X; we already know that this must be the case.

Recall: backward stability QR-like algorithms are *backward stable*: thanks to the fact that all transformations are orthogonal, one can prove that the computed $\tilde{U}, \tilde{T}$ are the exact Schur decomposition of a perturbed input

$$\mathcal{H} + \Delta_{\mathcal{H}} = \tilde{U}\tilde{T}\tilde{U}^*.$$

In particular, the Schur method computes a true invariant subspace of $\mathcal{H} + \Delta_{\mathcal{H}}$, with $\|\Delta\mathcal{H}\|$ small.

However, an important drawback is that this method is not *structureally* backward stable: the error $\Delta_{\mathcal{H}}$ is not Hamiltonian.

Among the consequences, eigenvalues close to the imaginary axis can be 'mixed up'. An example is `carex(14)`, an example in which the Hamiltonian matrix has eigenvalues very close to the imaginary axis.

```
>> [A, G, Q] = carex(14);
>> format short e
>> eig([A -G; -Q -A'])
ans =
  -3.7321e+00 + 0.0000e+00i
   3.7321e+00 + 0.0000e+00i
  -2.6795e-01 + 0.0000e+00i
   2.6795e-01 + 0.0000e+00i
   4.9991e-13 + 1.0000e+00i
   4.9991e-13 - 1.0000e+00i
  -5.0007e-13 + 1.0000e+00i
  -5.0007e-13 - 1.0000e+00i
>> X = care_schur(A, G, Q)
X =
    1.0003e+00 - 2.9186e-17i 4.6455e-04 - 2.6292e-17i -2.6185e-04 + 2.9186e-17i
-4.6355e-04 + 2.6292e-17i
    -4.6455e-04 - 7.7034e-17i 1.0003e+00 - 3.0087e-17i 4.6355e-04 + 7.7034e-17i
-2.6185e-04 + 3.0087e-17i
    -2.6185e-04 - 4.4409e-22i -4.6555e-04 + 4.4409e-16i 1.0003e+00 + 0.0000e+00i
4.6455e-04 - 4.4409e-16i
    4.6555e-04 - 9.9493e-17i -2.6185e-04 + 2.5750e-17i -4.6455e-04 + 9.9494e-17i
1.0003e+00 - 2.5750e-17i
>> norm(A'*X + X*A + Q - X*G*X) / (norm(A'*X) + norm(X*A) + norm(Q) + norm(X*G*X))
ans =
    6.5270e-16
```

```
>> max(real(eig(A - G*X)))
ans =
   -5.0028e-13
>> norm(X - X') / norm(X)
ans =
    1.8572e-03
```

On this problem, the Schur method produces an invariant subspace $\mathcal{U}$ that does *not* give a symmetric X. The explanation is that this is the wrong invariant subspace: this solution is not the stabilizing one, and is not even close to it; it is close to a non-stabilizing solution of the CARE.

To improve accuracy on ill-conditioned problems, it would be ideal to have a *structurally backward stable* method.

Indefinite scalar products and symplectic transformations Ultimately, the reason why the previous algorithm fails to preserve the eigenvalue pairing is that orthogonal transformations do not preserve the matrix structure of the Hamiltonian.

This structure is intimately related with indefinite scalar products. Let us consider the indefinite scalar product (bilinear form) defined by the matrix J, i.e.,

$$\langle u, v\rangle = u^* J v = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}^* \begin{bmatrix} 0 & I \\ -I & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = u_1^* v_2 - u_2^* v_1.$$

The matrix $\mathcal{H}$ is skew-self-adjoint with respect to this scalar product, i.e., $\langle u, \mathcal{H}v\rangle = \langle -\mathcal{H}^* u, v\rangle$; indeed, this is equivalent to Lemma TODO. Indeed, any matrix H with $H_{11} = -H_{22}^*$, $H_{21} = H_{21}^*$, $H_{12} = H_{12}^*$ is so, even without the positive semidefiniteness constraint. These are called Hamiltonian matrices, and they all satisfy the eigenvalue pairing lemma.

So we must look for orthogonal transformations with respect to this scalar product.

Definition 14.1. A matrix $S \in \mathbb{C}^{2n\times 2n}$ is called *symplectic* if it is orthogonal w.r.t the scalar product J, that is, if $S^* J S = J$.

Lemma
If $\mathcal{H}$ is Hamiltonian and S is symplectic, then $S^{-1}\mathcal{H}S$ is Hamiltonian.

Proof: $(S^{-1}\mathcal{H}S)^* J = J(S^{-1}\mathcal{H}S) \iff (S^{-1}\mathcal{H}S)^* S^* J S = S^* J S(S^{-1}\mathcal{H}S) \iff S^*\mathcal{H}^* J S = S^* J \mathcal{H} S$.

Remark: unlike orthogonal transformations, symplectic ones do not automatically ensure stability: $\|v\|$ small does not imply $\|Sv\|$ small: for instance, any matrix of the form $S = \left[\begin{smallmatrix} A & 0 \\ 0 & A^{-T} \end{smallmatrix}\right]$ is symplectic.

Orthosymplectic transformations *Ideal setting*: construct successive changes of bases $\mathcal{H} \mapsto S^{-1}\mathcal{H}S$ where S is *both* orthogonal (for stability reasons) and symplectic (for structure preservation reasons). These are called orthosymplectic matrices.

Examples of orthosymplectic matrices:

- If $Q \in \mathbb{C}^{n\times n}$ is any orthogonal matrix, then $\operatorname{blkdiag}(Q, Q)$ is orthosymplectic.
- A Givens matrix that acts on entries k and $n+k$ (i.e., that generated with the Matlab commands

  ```
  G = eye(2*n); G([k,n+k], [k,n+k]) = [c s; -s c];
  ```

  is orthosymplectic.

The Laub trick There is a certain *orthogonal and symplectic* matrix that reduces $\mathcal{H}$ to a special form.

Theorem
Let $U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix}$ be unitary s.t. $\begin{bmatrix} U_{11} \\ U_{21} \end{bmatrix}$ spans the stable invariant subspace. Then,

1. $V = \begin{bmatrix} U_{11} & -U_{21} \\ U_{21} & U_{11} \end{bmatrix}$ is orthosymplectic;
2. $V^*\mathcal{H}V = \begin{bmatrix} T_{11} & T_{12} \\ 0 & -T_{11}^* \end{bmatrix}$, with T_{11} upper triangular and T_{12} symmetric (*Hamiltonian Schur form*).

Proof (1) follows from the fact that $\left[\begin{smallmatrix} U_{11} \\ U_{21} \end{smallmatrix}\right]$ has orthonormal columns, and we showed earlier that $U_{21}^* U_{11} - U_{11}^* U_{21} = 0$.

(2) follows from the facts that $\left[\begin{smallmatrix} U_{11} \\ U_{21} \end{smallmatrix}\right]$ spans an invariant subspace and that $V^*\mathcal{H}V$ is Hamiltonian.

An orthogonal symplectic algorithm Numerically, the Laub trick is no more effective than the Schur method, because they compute the same invariant subspace.

But the existence of this structured factorization suggests that there may be a structure-preserving method to compute it.

Problem ("curse of Van Loan")
Is there a *structure-preserving QR method* that produces the Hamiltonian Schur form via a sequence of orthosymplectic transformations applied to $\mathcal{H}$?

Roadblock: we have proved that a stable invariant subspace exists if (A, B) controllable and $G, Q \succeq 0$, but there are Hamiltonian matrices that do not satisfy these assumptions; e.g., $\mathcal{H} = \left[\begin{smallmatrix} 1 & 2 \\ -1 & -1 \end{smallmatrix}\right]$ with eigenvalues $\pm i$.

$\implies$ algorithms to compute a HSF must become unstable when G, Q are ill-conditioned.

Extras: Chu–Liu–Mehrmann algorithm [Chu-Liu-Mehrmann '98] A solution comes by going through another, different decomposition: $\mathcal{H} = URV^*$, with U, V orthosymplectic and

$$R = \begin{bmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{bmatrix}$$

with $R_{11}R_{22}^*$ upper triangular.

(Reminds of the SVD.)

It can be computed in $O(n^3)$ via backward stable orthosymplectic transformations.

Note that R is *not* Hamiltonian and $\Lambda(\mathcal{H}) \neq \Lambda(R)$, in general.

URV decomposition — sketch

- Left-multiply by blkdiag(Q, Q) to get $\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}$
- Left-multiply by a Givens on $(1, n+1)$ to get $\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}$
- Right-multiply by blkdiag(Q, Q) to get $\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & 0 & * & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}$
- Right-multiply by a Givens on $(2, n+2)$ to get $\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & * & * & * & * & * \\ 0 & * & * & * & * & * \end{bmatrix}$
- Repeat on smaller blocks to zero out the $(2,1)$ block: $\begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & * & * & * & * \end{bmatrix}, \begin{bmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * & * \end{bmatrix}.$

URV — the final step Finally, left multiply by blkdiag(Q, Q) to replace R_1, R_2 with QR_1, QR_2^* so that $QR_1R_2^*Q^*$ is upper triangular.

Note that $\mathcal{H} = URV$ together with symplecticity implies

$$\mathcal{H} = V \begin{bmatrix} -R_{22}^* & R_{12}^* \\ 0 & -R_{11}^* \end{bmatrix} U^*.$$

Then

$$\mathcal{H}^2 = V \begin{bmatrix} -R_{11}R_{22}^* & * \\ 0 & -R_{22}R_{11}^* \end{bmatrix} V^*.$$

This is a Schur-like decomposition that reveals the eigenvalues and eigenvectors of $\mathcal{H}^2$ (not of $\mathcal{H}$). However, if v_1 is an eigenvector of $\mathcal{H}^2$ then span$(v_1, \mathcal{H}v_1)$ is an invariant subspace of $\mathcal{H}$, and it can be used to compute an eigenvector of $\mathcal{H}$ and deflate it (many details omitted).

Chapter 15

The sign function method for CAREs

Sign-like methods for CAREs Let us consider the matrix sign iteration

$$H_{k+1} = \frac{1}{2}(H_k + H_k^{-1}), \quad H_0 = \mathcal{H}.$$

One can see that H_k is Hamiltonian at each step (i.e., $JH_k = -H_k^* J$). Indeed, the following properties hold.

Lemma 15.1. *The following propertiess hold.*

- *Let H be Hamiltonian. Then H^{-1} is Hamiltonian, too.*
- *Let H_1, H_2 be Hamiltonian. Then $H_1 + H_2$ is Hamiltonian, too.*

It is easy to prove this lemma by direct verification of the property $JH_k = -H_k^* J$. The guiding idea is that Hamiltonian matrices are like antisymmetric ones: properties that one expects for antisymmetric matrices often hold for Hamiltonian, too.

Structure-preserving sign iteration In machine arithmetic, the H_k won't be exactly Hamiltonian — unless we modify our algorithm to ensure that they are.

Observe that the defining property of Hamiltonian matrices $JH_k = -H_k^* J$ can be rewritten as: $\mathcal{H}$ is Hamiltonian iff $J\mathcal{H}$ is symmetric.

So we can rewrite the sign iteration in terms of $Z_k := JH_k$:

$$Z_{k+1} = \frac{1}{2}(Z_k + JZ_k^{-1}J), \quad Z_0 = J\mathcal{H}.$$

This version preserves symmetry exactly, assuming that the numerical method we use for inversion does; on Matlab, `inv` on a symmetric matrix uses an LDL decomposition, and ensures that the result is exactly symmetric: if Z is symmetric, `IZ = inv(Z); IZ - IZ'` is guaranteed to return a matrix of exact zeros.

We can incorporate scaling, exactly as discussed in the section on the Newton method for the matrix sign.

```
function [X, k] = care_sign(A, G, Q)

n = size(A);
J = [zeros(n) eye(n); -eye(n) zeros(n)];
Z = [-Q -A'; -A G];
err = inf;
k = 0;
while err >= 1e-15
    Zold = Z;
    Z = 1/2*(Z + J*inv(Z)*J);
    % these products with J could be replaced
    % with direct block reordering, for
    % better performance
    err = norm(Zold - Z) / norm(Z);
    k = k + 1;
end
U = null(Z + J);
X = U(n+1:2*n, 1:n) / U(1:n, 1:n);
```

Whenever we compute the nullspace of a matrix, we should be careful since the matrix could have decaying singular values, making it difficult to guess correctly the dimension of its nullspace. However, since symmetry is preserved exactly, it is guaranteed that the limit $\mathcal{H}_*$ has n eigenvalues equal to 1 and n equal to -1; so we have nothing to guess. It is still possible that $\mathcal{H}_*$ is severely ill-conditioned, though, if $\mathcal{H}$ is far from normal and its stable and anti-stable invariant subspaces are badly separated.

```
>> [A, G, Q] = carex(4);
>> X = care_sign(A, G, Q);
>> norm(A'*X + X*A + Q - X*G*X) / (norm(A'*X) + norm(X*A) + norm(Q) + norm(X*G*X))
ans =
    1.4435e-15
>> max(real(eig(A - G*X)))
ans =
    -1.0057e-01
```

What happens in a more ill-conditioned example?

```
>> [A, G, Q] = carex(14);
>> X = care_sign(A, G, Q);
>> norm(A'*X + X*A + Q - X*G*X) / (norm(A'*X) + norm(X*A) + norm(Q) + norm(X*G*X))
ans =
   2.4419e-05
>> max(real(eig(A - G*X)))
ans =
   -5.0006e-13
>> norm(X - X') / norm(X)
ans =
   8.7455e-16
```

The method takes many iterations, 48, to reach our stopping criterion; and the computed X is only a very rough approximation of the solution, since its relative residual if of the order of 10^{-5}. However, this matrix is symmetric; this is a sign that, unlike the Schur method, the sign method computes an approximation of the correct solution: by preserving structure, we ensure that we are computing the invarfiant subspace of a Hamiltonian matrix.

As noted above, we can use the Newton method to improve the quality of this approximate solution: we apply to steps of the Newton method, each time using the previous X as a starting value.

```
>> X = care_newton(A, G, Q, 1, X);
>> norm(A'*X + X*A + Q - X*G*X) / (norm(A'*X) + norm(X*A) + norm(Q) + norm(X*G*X))
ans =
   3.2018e-10
>> X = care_newton(A, G, Q, 1, X);
>> norm(A'*X + X*A + Q - X*G*X) / (norm(A'*X) + norm(X*A) + norm(Q) + norm(X*G*X))
ans =
   1.0205e-16
```

The residual drops sharply, as expected for a quadratically converging method.

15.1 The doubling algorithm

In this section, we describe an interesting algorithm that implements the sign method in an implicit fashion.

Recall that, in the sign iteration, if we set $Y_k = (I - X_k)^{-1}(I + X_k)$, then $Y_{k+1} = -Y_k^2$.

In an ideal world without rounding errors, we could compute $Y_0, Y_1, Y_2, \dots,$ and then get the stable invariant subspace as $\ker Y_\infty$ (or, rather, the invariant subspace associated to the n smallest singular values of Y_∞, since in an ideal world without rounding errors this matrix is nonsingular).

We can do these steps also in machine arithmetic, if we work in a suitable format.

Standard Symplectic Form We would like to find E_0, F_0, G_0, H_0 such that the matrix $Y_0 = (I - \mathcal{H})^{-1}(I + \mathcal{H})$ can be factored as

$$Y_0 = \begin{bmatrix} I & G_0 \\ 0 & F_0 \end{bmatrix}^{-1} \begin{bmatrix} E_0 & 0 \\ H_0 & I \end{bmatrix}.$$

Trick: this is equivalent to finding M such that

$$M \begin{bmatrix} (I - \mathcal{H}) & (I + \mathcal{H}) \end{bmatrix} = \begin{bmatrix} I & G_0 & E_0 & 0 \\ 0 & F_0 & H_0 & I \end{bmatrix}.$$

Clearly this matrix M must be the inverse of block columns 1 and 4.

Structural properties:

- if $\mathcal{H}$ is Hamiltonian, Y_0 is symplectic.

  Proof: via $(I - \mathcal{H})^* J (I - \mathcal{H}) = (I + \mathcal{H})^* J (I + \mathcal{H})$.
- If Y_0 is symplectic, $E_0 = F_0^*, G_0 = G_0^*, H_0 = H_0^*$.
- Moreover, if $G \succeq 0$, $H \succeq 0$, then $G_0 \succeq 0$, $H_0 \preceq 0$ (tedious).

Doubling algorithm *Plan* Given $Y_k = \begin{bmatrix} I & G_k \\ 0 & E_k^* \end{bmatrix}^{-1} \begin{bmatrix} E_k & 0 \\ H_k & I \end{bmatrix}$, compute $Y_{k+1} = -Y_k^2 = \begin{bmatrix} I & G_{k+1} \\ 0 & E_{k+1}^* \end{bmatrix}^{-1} \begin{bmatrix} E_{k+1} & 0 \\ H_{k+1} & I \end{bmatrix}$.

Similar to the 'inverse-free sign method' described earlier.

The swap: If $Y_k = \mathcal{M}_k^{-1}\mathcal{N}_k$, then $-Y_k^2 = -\mathcal{M}_k^{-1}\mathcal{N}_k\mathcal{M}_k^{-1}\mathcal{N}_k = \mathcal{M}_k^{-1}\widehat{\mathcal{M}}_k^{-1}\widehat{\mathcal{N}}_k\mathcal{N}_k = (\widehat{\mathcal{M}}_k\mathcal{M}_k)^{-1}(\widehat{\mathcal{N}}_k\mathcal{N}_k)$, where $\widehat{\mathcal{M}}_k, \widehat{\mathcal{N}}_k$ satisfy $\widehat{\mathcal{M}}_k^{-1}\widehat{\mathcal{N}}_k = -\mathcal{N}_k\mathcal{M}_k^{-1}$, i.e.,

$$\begin{bmatrix} \widehat{\mathcal{M}}_k & \widehat{\mathcal{N}}_k \end{bmatrix} \begin{bmatrix} \mathcal{N}_k \\ \mathcal{M}_k \end{bmatrix} = 0.$$

Doubling: the swap

$$\begin{bmatrix} I & \widehat{G}_k & \widehat{E}_k & 0 \\ 0 & \widehat{F}_k & \widehat{H}_k & I \end{bmatrix} \begin{bmatrix} E_k & 0 \\ H_k & I \\ I & G_k \\ 0 & E_k^* \end{bmatrix} = 0$$

holds if

$$\begin{aligned} \begin{bmatrix} \widehat{G}_k & \widehat{E}_k \\ \widehat{F}_k & \widehat{H}_k \end{bmatrix} &= -\begin{bmatrix} E_k & 0 \\ 0 & E_k^* \end{bmatrix} \begin{bmatrix} H_k & I \\ I & G_k \end{bmatrix}^{-1} \\ &= \begin{bmatrix} E_k & 0 \\ 0 & E_k^* \end{bmatrix} \begin{bmatrix} G_k(I - H_kG_k)^{-1} & -(I - G_kH_k)^{-1} \\ -(I - H_kG_k)^{-1} & H_k(I - G_kH_k)^{-1} \end{bmatrix}. \end{aligned}$$

Doubling: the formulas Putting everything together,

$$\begin{bmatrix} E_{k+1} & 0 \\ H_{k+1} & I \end{bmatrix} = \begin{bmatrix} -E_k(I-G_kH_k)^{-1} & 0 \\ E_k^*H_k(I-G_kH_k)^{-1} & I \end{bmatrix} \begin{bmatrix} E_k & 0 \\ H_k & I \end{bmatrix}$$
$$= \begin{bmatrix} -E_k(I-G_kH_k)^{-1}E_k & 0 \\ H_k + E_k^*H_k(I-G_kH_k)^{-1}E_k & I \end{bmatrix}$$

and an analogous computation gives E_{k+1}^*, G_{k+1}:

Structured doubling algorithm

$$E_{k+1} = -E_k(I-G_kH_k)^{-1}E_k,$$
$$G_{k+1} = G_k + E_kG_k(I-H_kG_k)^{-1}E_k^*,$$
$$H_{k+1} = H_k + E_k^*H_k(I-G_kH_k)^{-1}E_k.$$

SDA: details Note that (even when the series does not converge)

$$G_k(I-H_kG_k)^{-1} = G_k + G_kH_kG_k + G_kH_kG_kH_kG_k + \cdots = (I-G_kH_k)^{-1}G_k,$$

and this matrix is symmetric. If $G_k = B_kB_k^*$, then it can also be rewritten as $B_k(I - B_k^*H_kB_k)^{-1}B_k^*$ (inverting a symmetric matrix).

Monotonicity If $H_k \preceq 0$ then $G_k(I-H_kG_k)^{-1} \succeq 0$. Hence, $0 \preceq G_0 \preceq G_1 \preceq \dots$, and $0 \succeq H_0 \succeq H_1 \succeq H_2 \succeq \dots$

Cost As much as a $2n \times 2n$ inversion $M^{-1}N$, if you put everything together. Unlike the sign algorithm, we have a bound $\sigma_{\min}(I - H_kG_k) \geq 1$ (because $G_k \succeq 0$, $H_k \preceq 0$).

SDA: the dual equation To analyze convergence, we need to introduce another matrix. Let Y be the matrix such that

$$\mathcal{H}\begin{bmatrix} -Y \\ I \end{bmatrix} = \begin{bmatrix} A & -G \\ -Q & -A^* \end{bmatrix}\begin{bmatrix} -Y \\ I \end{bmatrix} = \begin{bmatrix} -Y \\ I \end{bmatrix}\widehat{\mathcal{R}}$$

is the *anti-stable* invariant subspace of $\mathcal{H}$, i.e., $\Lambda(\widehat{\mathcal{R}}) \subset RHP$.

$\begin{bmatrix} I \\ Y \end{bmatrix}$ spans the stable subspace of $\mathcal{H}^* = -J\mathcal{H}J$; we can prove that the subspace has this form if (A^T, C^T) controllable (typically satisfied).

SDA: convergence (intuitively) **Theorem**
In SDA, $E_k \to 0, G_k \to Y, H_k \to -X$. Convergence is quadratic, i.e., $\|H_k + X\| = \mathcal{O}(\rho^{2^k})$ for some $\rho \in [0,1)$, as $k \to \infty$.

Intuitive view $E_k \to 0$, approximately squared at each time. Hence

$$\mathcal{H}_k = \begin{bmatrix} I & G_k \\ 0 & E_k^* \end{bmatrix}^{-1} \begin{bmatrix} E_k & 0 \\ H_k & I \end{bmatrix}$$

has n eigenvalues $\to 0$ and n that $\to \infty$. $\ker \mathcal{H}_k \approx \begin{bmatrix} I \\ -H_k \end{bmatrix}$, so $-H_k \to X$.

Dually, "$\ker \mathcal{H}_k^{-1}$" (a thing that shouldn't exist…) $\approx \begin{bmatrix} -G_k \\ I \end{bmatrix}$, so $G_k \to Y$.

SDA convergence (formally) *Proof* some manipulations give

$$\mathcal{H}_0 \begin{bmatrix} I \\ X \end{bmatrix} = (I - \mathcal{H})^{-1}(I + \mathcal{H}) \begin{bmatrix} I \\ X \end{bmatrix} = \begin{bmatrix} I \\ X \end{bmatrix} (I - \mathcal{R})^{-1}(I + \mathcal{R}).$$

where $\mathcal{S} = (I - \mathcal{R})^{-1}(I + \mathcal{R})$ has eigenvalues in the unit circle. Thus

$$\begin{bmatrix} I & G_k \\ 0 & E_k^* \end{bmatrix} \begin{bmatrix} I \\ X \end{bmatrix} = \begin{bmatrix} I \\ X \end{bmatrix} \begin{bmatrix} E_k & 0 \\ H_k & I \end{bmatrix} \begin{bmatrix} I \\ X \end{bmatrix} \mathcal{S}^{2^k}.$$

which implies

$$\begin{aligned} E_k &= (I + G_k X)\mathcal{S}^{2^k}, \\ H_k + X &= E_k^* X \mathcal{S}^{2^k} = (\mathcal{S}^{2^k})^*(I + X G_k)\mathcal{S}^{2^k} \succeq 0. \end{aligned}$$

The same computation on the dual equation gives $G_k \preceq Y$, so G_k is bounded and $E_k \to 0, H_k + X \to 0$ (quadratically as $\mathcal{S}^{2^k}$).

Chapter 16

Methods for large-scale control systems

Methods for large-scale control systems We give a hint of the methods used for large-scale control systems.

What does a large-scale control system look like?

Example: the heat equation: finite-difference discretization of a 2D or 3D structure, possibly on a non-square domain.

- Large, sparse $A \in \mathbb{R}^{n\times n}$, often produced by a discretization.
- $B \in \mathbb{R}^{n\times m}$ with $m \ll n$: the control usually acts only on a few points.

We aim to solve these problems with a cost that is approximately linear in n (or in the number of nonzeros of A, which is the true 'dimension' of the problem).

Large-scale Lyapunov equations We focus on a very simple problem: solving Lyapunov equations with $m = 1$

$$AX + XA^* + bb^* = 0, \quad \Lambda(A) \subset LHP, \quad b \in \mathbb{C}^n \tag{16.1}$$

This is the bare minimum that we need to solve all the problems that we have encountered. Indeed,

- If we have a Lyapunov equation with $B = [b_1, b_2, \dots, b_m]$, we can compute its solution X as $X = X_1 + X_2 + \cdots + X_m$, where X_i solves $AX_i + X_iA^* = b_ib_i^*$ for each $i = 1, 2, \dots, m$. This is because Lyapunov equations are linear.
- Once we know how to solve Lyapunov equations, we can run Newton's method to solve an algebraic Riccati equation.

There are other methods to address these problems directly, but to get a hang of the techniques we will focus on the simple case (16.1) only.

We shall also assume that we can solve linear systems of the form $(A-\alpha I)v = w$ efficiently. In practice, for a sparse A, this is achieved using a sparse LU factorization. but in principle the methods are applicable also to more general structures (e.g., a Toeplitz matrix A).

A first difficulty is that the solution X is typically dense and full-rank: indeed, we know that $X = X^* \succ 0$, since usually (A, b) is controllable, so X is nonsingular. If it is dense, it has n^2 nonzero entries, and we cannot afford to compute nor return all of them.

However, it is often the case that X has eigenvalues that decay very quickly, so we can approximate it with $X \approx ZZ^*$, for a tall thin Z. We shall see in the following why. Our algorithms will return Z rather than X.

Matlab example

```
>> n = 1000;
>> rng(0); A = sprandn(n, n, 0.01); % sparse random A with 1% nonzeros
>> A = A - 10*speye(n); % to ensure Lambda(A) in LHP
>> max(real(eig(full(A)))) %and indeed it is so
ans =
   -6.9058
>> b = randn(n, 1);
>> X = lyap(A, b*b'); % this uses a dense O(n^3) method
>> format short e
>> eig(X)
ans =
[...]
5.0019e-15
5.0876e-15
7.2166e-15
9.6627e-15
1.4224e-14
2.5617e-13
9.9637e-12
3.7134e-10
1.6017e-08
6.0124e-07
2.3502e-05
9.2131e-04
3.4250e-02
1.4232e+00
5.1405e+01
>>
```

All the other 985 eigenvalues are essentially zero!

16.1 ADI (alternating-direction implicit iteration)

Idea Let's convert our continuous-time problem (Lyapunov equation)

$$AX + XA^* + bb^* = 0 \tag{16.2}$$

to a discrete-time one (Stein equation)

$$X - \hat{A}X\hat{A}^* = \hat{b}\hat{b}^*, \tag{16.3}$$

since those can be solved with a simpler fixed-point iteration.

Theorem 16.1. *Let $\tau > 0$, so that $\Lambda(A-\tau I) \subset LHP$. Then, X solves* (16.2) *if and only if it solves* (16.3) *with $\hat{A} := (A-\tau I)^{-1}(A+\tau I)$, $\hat{b} := \sqrt{2\tau}(A-\tau I)^{-1}b$.*

Proof. Expand in two ways

$$(A-\tau I)X(A-\tau I)^* - (A+\tau I)X(A+\tau I)^* - 2\tau bb^* = 0.$$

□

Solving Stein equations We can solve (16.3) with the fixed-point iteration

$$X_k = \hat{A}X_{k-1}\hat{A}^* + \hat{b}\hat{b}^*. \tag{16.4}$$

Lemma 16.2. *If $\Lambda(A) \subset LHP$ and $\tau > 0$, then $\Lambda(\hat{A}) \subset \mathbb{D}$ (unit disk).*

Proof. If $\lambda \in LHP$, then $\operatorname{dist}(\lambda, -\tau) < \operatorname{dist}(\lambda, \tau)$, thus $\frac{|\lambda+\tau|}{|\lambda-\tau|} < 1$. □

Extras: Time discretization It is interesting to note that $\hat{A}$ and a (scaled) version of $\hat{b}$ be obtained by discretizing the control system with the *midpoint method*:

$$\dot{x} = Ax + Bu$$

is discretized to

$$\frac{x_{k+1} - x_k}{h} = \frac{1}{2}\left(Ax_k + Bu_k + Ax_{k+1} + Bu_{k+1}\right),$$

i.e.,

$$x_{k+1} = (I - \tfrac{h}{2}A)^{-1}(I + \tfrac{h}{2}A)x_k + (I - \tfrac{h}{2}A)^{-1}B(u_k + u_{k+1})\tfrac{h}{2}.$$

This specific method is particularly nice, because it preserves stability: the open-loop system $\dot{x} = Ax$ is stable iff $x_{k+1} = (I - \frac{h}{2}A)^{-1}(I + \frac{h}{2}A)x_k$ is so.

Low-rank formulation Starting from $X_0 = 0$, we have

$$X_k = \hat{b}\hat{b}^* + \hat{A}\hat{b}\hat{b}^*\hat{A} * + \hat{A}^2\hat{b}\hat{b}^*\hat{A}^{2*} + \cdots + \hat{A}^{k-1}\hat{b}\hat{b}^*\hat{A}^{(k-1)*}.$$

Or, in terms of its *low-rank factor*

$$Z_k = \begin{bmatrix}\hat{b} & \hat{A}\hat{b} & \hat{A}^2\hat{b} & \dots & \hat{A}^{k-1}\hat{b}\end{bmatrix}, \quad X_k = Z_k Z_k^*.$$

We can compute the columns of the Z_k's iteratively

$$\begin{cases} v_1 = \hat{b}, \\ v_{k+1} = \hat{A}v_k = (A - \tau I)^{-1}(A + \tau I)v_k = v_k + 2\tau(A - \tau I)^{-1}v_k. \end{cases}$$

Cost: One shifted solve with A per iteration.

```
function Z = adi_single_shift(A, b, tau, k)
n = size(A,1);
v = (A - tau*speye(n)) \ b * sqrt(2*tau);
Z = [v];

% this could be improved by computing only once
% a factorization of A - tau*I

for i = 1:k-1
    v = v + (A - tau*speye(n)) \ v * 2*tau;
    Z = [Z v];
end
```

Take care with parentheses here: if we had written `v = sqrt(2*tau) * (A - tau*speye(n)) \ b`, Matlab would have associated starting from the left and computed `v = (sqrt(2*tau) * (A - tau*speye(n))`
i.e., we would have erroneously divided, rather than multiplied, by $\sqrt{2\tau}$

On our example, convergence is pretty fast. With 20 iterations and $\tau = 5.0$:

```
    >> Z = adi_single_shift(A, b, 5.0, 20);
    >> norm(Z*Z' - X) / norm(X)
    ans =
       1.1886e-14
```

We can estimate the convergence speed rigorously.

Lemma 16.3. *For the iteration* (16.4),

$$X_k - X = \hat{A}^k(X_0 - X)\hat{A}^{k*}.$$

Proof. Induction. □

Hence convergence is linear:

$$\|X_k - X\| \leq \|\hat{A}^k\|\|X_0 - X\|\|\hat{A}^{k*}\| = \|\hat{A}^k\|^2\|X\| \sim \rho(\hat{A})^{2k}.$$

On our example, the spectral radius $\rho(\hat{A})$ is rather small.

```
>> Ahat = (A-5.0*speye(n)) \ (A+5.0*speye(n));
>> max(abs(eig(full(Ahat))))
ans =
   4.4949e-01
```

On other matrices, we might be less lucky.

One can see that if A has an eigenvalue with $|\mathrm{Re}(\lambda)| \ll \tau$ then $\hat{A}$ has an eigenvalue $\frac{\lambda+\tau}{\lambda-\tau} \approx -1$, and similarly if $|\lambda| \gg \tau$ then $\frac{\lambda+\tau}{\lambda-\tau} \approx 1$.

This suggests that the optimal τ is close to the eigenvalues of $-A$. But if A has eigenvalues with both very small and very large negative real parts, we cannot find a value τ that works well with all eigenvalues, and inevitably $\rho(\hat{A})$ will be very close to 1.

16.2 ADI with multiple shifts

The key to get faster convergence is changing the value of τ at each step: given an arbitrary sequence of positive *shifts* (or *poles*) $\tau_1, \tau_2, \dots$, we set

$$\hat{A}_k := (A - \tau_k I)^{-1}(A + \tau_k I), \quad \hat{b}_k := \sqrt{2\tau_k}(A - \tau_k I)^{-1} b.$$

Then one can set up the iteration

$$X_0 = 0, \quad X_k = \hat{A}_k X_{k-1} \hat{A}_k^* + \hat{b}_k \hat{b}_k^*.$$

At each iteration we change the fixed-point equation that we use, but still the iteration behaves similarly.

Proceeding as above one gets

$$Z_k = \begin{bmatrix} \hat{b}_k & \hat{A}_k \hat{b}_{k-1} & \hat{A}_k \hat{A}_{k-1} \hat{b}_{k-2} & \dots & \hat{A}_k \hat{A}_{k-1} \cdots \hat{A}_2 \hat{b}_1 \end{bmatrix}.$$

Rearranging the computation It's less clear from this formulation how to compute the columns of Z_k iteratively. However one can rearrange things using commutativity, to reach a form that makes the iterative structure more clear:

$$\begin{aligned}
&\frac{1}{\sqrt{2\tau_{k-2}}} \hat{A}_k \hat{A}_{k-1} \hat{b}_{k-2} \\
&= (A - \tau_k I)^{-1}(A + \tau_k I)(A - \tau_{k-1} I)^{-1}(A + \tau_{k-1} I)(A - \tau_{k-2} I)^{-1} b \\
&= \underbrace{(A - \tau_{k-2} I)^{-1}(A + \tau_{k-1} I)\,\underbrace{(A - \tau_{k-1} I)^{-1}(A + \tau_k I)\,\underbrace{(A - \tau_k I)^{-1} b}_{=:w_1}}_{=:w_2}}_{=:w_3}
\end{aligned}$$

Thus we get $w_1 = (A - \tau_k I)^{-1} \hat{b}$ and

$$\begin{aligned}
w_{j+1} &= (A - \tau_{k-j} I)^{-1}(A + \tau_{k-j+1} I) w_j \\
&= w_j + (\tau_{k-j} + \tau_{k-j+1})(A - \tau_{k-j} I)^{-1} w_j.
\end{aligned}$$

Low-rank ADI: the formulation Reversing the order of the τ_j for simplicity, we get

Low-rank ADI with multiple shifts

$$v_1 = \sqrt{2\tau_1}(A-\tau_1)^{-1}b, \quad v_j = \frac{\sqrt{2\tau_j}}{\sqrt{2\tau_{j-1}}}\left(v_j + (\tau_{j-1}+\tau_j)\left(A-\tau_j I\right)^{-1}v_j\right).$$

$$Z_k = \begin{bmatrix} v_1 & v_2 & \dots & v_k \end{bmatrix}.$$

One can also use complex shifts (details omitted; complex conjugates $\overline{\tau}_j$ appear).

ADI: convergence Proceeding analogously to the one-shift case, one gets

$$X_k - X_* = \hat{A}_k\hat{A}_{k-1}\cdots\hat{A}_1(X_0-X_*)\hat{A}_1^*\cdots\hat{A}_{k-1}^*\hat{A}_k^* = g(A)(X_0-X_*)g(A)^*,$$

where $g(x) = \prod_{j=1}^k \frac{x-\tau_j}{x+\tau_j}$.

Hence the key to get a fast convergence is choosing the τ_j's so that $\|g(A)\|$ is small.

If $A = V\Lambda V^{-1}$, then

$$\|g(A)\| = \|Vg(\Lambda)V^{-1}\| \leq \kappa(V)\max_{\lambda\in\Lambda(A)}\prod_{j=1}^k \frac{|\lambda-\tau_j|}{|\lambda+\tau_j|}.$$

If A has at most k distinct eigenvalues, we can choose $\tau_j = -\lambda_j$ to get $g(A)=0$ and *exact convergence* in k steps.

If A has k *clusters*, we get a small $\|g(A)\|$ after k steps by choosing shifts close to the centers of these clusters (cfr. Arnoldi convergence theory).

The ADI shift choice problem We can express the optimal shifts $\tau_1,\dots,\tau_k$ as the solution of an approximation problem, analogously to the theory in Arnoldi:

$$\eta_k = \min_{\tau_1,\dots,\tau_k}\max_{\lambda\in\Lambda(A)}\prod_{j=0}^{k-1}\frac{|\lambda-\tau_j|}{|\lambda+\tau_j|}.$$

This computation, in practice, is unfeasible: to compute an optimal solution, we need the full spectrum $\Lambda(A)$, which is unfeasible. And the choice would change at each step k, requiring recomputation of previous iterates.

One usually computes a small number k' of shifts initially, before starting the iteration, and reuses them cyclically with $\tau_k = \tau_{\mathrm{mod}(k,k')}$.

Often the max is taken by the largest or smallest eigenvalues of A. Hence we can run a few steps of Arnoldi on A and A^{-1} to get $\{\mu_1,\dots,\mu_d\}$ that approximate the extremal eigenvalues of A, and get a simpler, smaller-scale problem

$$\min_{\tau_1,\dots,\tau_k}\max_{\lambda\in\{\mu_1,\dots,\mu_d\}}\prod_{j=0}^{k-1}\frac{|\lambda-\tau_j|}{|\lambda+\tau_j|}.$$

ADI optimal shifts Alternatively, we can replace $\Lambda(A)$ with a region $C \subset LHP$ enclosing the eigenvalues of A: for instance, if $A = A^*$, all eigenvalues are in an interval $C = [a, b]$. Then, look for

$$\hat{\eta}_k = \min_{\tau_0,\dots,\tau_k} \max_{\lambda\in C} \prod_{j=0}^{k-1} \frac{|\lambda - \tau_j|}{|\lambda + \tau_j|}.$$

This is a classical problem from approximation theory: look for polynomials that are *small* on C and *large* on $-C$. Explicit solutions can be constructed from elliptic functions for many choices of C. It is known that $\hat{\eta}_k \sim r^k$ for a certain $r < 1$. This value r, known as <u>logarithmic capacity</u> of C, depends on $\frac{b}{a}$ in the symmetric case: so the convergence bounds re worse for ill-conditioned A).

Consequence Since $\|X_* - X_k\| \sim r^k$, and $\operatorname{rk} X_k = k$, it follows that $\sigma_{k+1}(X) \lesssim r^k$, so X has *low numerical rank*, whenever the eigenvalues decay fast enough. This argument validates our earlier claim that X often has rapidly decaying eigenvalues.

Extra: Residual computation *Detail* As a stopping criterion for ADI, we would like to use the residual $\|AZ_kZ_k^* + Z_kZ_k^*A^* + bb^*\|$, but how can we compute it without assembling all these large matrices? We show a method to do it.

For $X_k = Z_kZ_k^*$, with $Z_k \in \mathbb{R}^{n\times k}$, we have

$$AZ_kZ_k^* + Z_kZ_k^*A^* + BB^* = \begin{bmatrix} Z_k & AZ_k & B \end{bmatrix} \begin{bmatrix} 0 & I & 0 \\ I & 0 & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} Z_k & AZ_k & B \end{bmatrix}^*.$$

Let us compute a thin QR factorization $Q_0R_0 = \begin{bmatrix} Z_k & AZ_k & B \end{bmatrix}$; this is a tall-thin matrix so the cost is low. Since the Q_0 factors have orthonormal columns, we have

$$\|AZ_kZ_k^* + Z_kZ_k^*A^* + BB^*\| = \left\| R \begin{bmatrix} 0 & I & 0 \\ I & 0 & 0 \\ 0 & 0 & I \end{bmatrix} R^* \right\|.$$

The total cost is $O(nk^2)$, linear in the matrix size n and quadratic in the number of iterations k.

16.3 Rational Arnoldi

An alternative algorithm for large-scale Lyapunov equations comes from Krylov subspace ideas. Note that the approximation Z_k computed by ADI has columns of the form $r(A)b$, where $r(x) = p(x)/q(x)$, with fixed denominator $q(x) = (x-\tau_1)(x-\tau_2)\dots(x-\tau_k)$. In other words, the columns of Z_k (and hence also those of X_k) belong to the *rational Arnoldi subspace*

$$K_q(A, b) = \{q(A)^{-1}p(A)b \colon \deg p < k\} = q(A)^{-1}K_k(A, b).$$

Idea: first compute this subspace, then look for an approximated solution with $ImZ_k \subset K_q(A, b)$ by 'projecting the problem'.

Galerkin Projection Given an orthonormal basis U_k of $K_q(A, b)$:

1. Set $X_k = U_k Y_k U_k^*$;
2. Assume 'orthogonal residual': $U_k^*(AX_k + X_k A^* + bb^*)U_k = 0$.

This strategy produces a projected $k \times k$ Lyapunov equation

$$(U_k^* A U_k)Y + Y(U_k^* A U_k)^* + U_k^* bb^* U_k = 0.$$

Since its size is smaller, we can solve it using direct methods.

Difficulty 1 Even if $\Lambda(A) \subset LHP$, the same property does not always hold for $A_k = U_k^* A U_k$. Recall: the eigenvalues of $A_k = U_k^* A U_k$ are in the *field of values* of A, which is $\operatorname{hull} \Lambda(A)$ for normal A, but larger (possibly by much) for non-normal A. If A is far from normal, it is a common occurrence that this method produces projected equations for which $\Lambda(A_k) \not\subset LHP$. *Difficulty 2* (the main one, shared with ADI): good *pole selection* is crucial for convergence.

Solving large and sparse Lyapunov and Riccati equation is another very active research area. Many more sophisticated methods have been introduced in recent years.