Computational Linear Algebra

Lecture Notes

Daniel Kressner
EPFL / MATH / ANCHP

Spring 2024

May 24, 2024

Remarks

- Although not necessary, you may find it helpful to complement these lecture notes with additional literature. A selection of recommended books:

  [**Dem97**] J. W. Demmel. Applied numerical linear algebra. SIAM, Philadelphia, PA, 1997.

  [**ElmSW05**] H. Elman, D. J. Silvester, and A. J. Wathen. Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. Oxford University Press, New York, 2005.

  [**GolV13**] G. H. Golub and C. F. Van Loan. Matrix computations. Fourth edition. Johns Hopkins University Press, Baltimore, MD, 2013.

  [**HorJ13**] R. A. Horn and C. R. Johnson. Matrix analysis. Second edition. Cambridge University Press, 2013.

  [**Saa03**] Y. Saad. Iterative methods for sparse linear systems. Second edition. SIAM, Philadelphia, PA, 2003.

  [**TreB97**] L. N. Trefethen and D. Bau. Numerical linear algebra. SIAM, Philadelphia, PA, 1997.

  Please note: None of the books in this list is needed to prepare for the exam.

- Not all of the material contained in these lecture notes will be covered in class. Only the material covered in class is relevant to the exam. You may skip reading these parts, but it may be worthwhile not to do so.

- Chapter 1 is partly based on Peter Arbenz: *Lecture Notes on Solving Large-Scale Eigenvalue Problems*. Section 1.6 is based on [von Luxburg, U. A tutorial on spectral clustering. Stat. Comput. 17 (2007), no. 4, 395–416]. Section 4.2.4 is based on [Saa03]. Section 4.2.5 is based on [ElmSW05].

- If you have any suggestions or spot mistakes, please send a message to

  `daniel.kressner@epfl.ch`

Chapter 1

Eigenvalue Problems: Basics and Applications

1.1 Notation and statement of the problem

Definition 1.1 *Given a square matrix $A \in \mathbb{C}^{n\times n}$, a scalar $\lambda \in \mathbb{C}$ is called an* **eigenvalue** *of A if there is a nonzero vector $\mathbf{x} \in \mathbb{C}^n$ such that*

$$A\mathbf{x} = \lambda\mathbf{x}. \tag{1.1}$$

The vector $\mathbf{x}$ is called an **eigenvector** *(belonging to λ) and $(\lambda, \mathbf{x})$ is called an* **eigenpair**.

Equivalently to Definition 1.1, we have

$$\begin{aligned} (\lambda, \mathbf{x}) \text{ is an eigenpair} &\iff (A - \lambda I)\mathbf{x} = 0, \quad \mathbf{x} \neq 0 \\ &\iff \text{the matrix } A - \lambda I \text{ is not invertible} \\ &\iff \det(A - \lambda I) = 0. \end{aligned}$$

Since the determinant of a matrix is a sum of products of n matrix entries, the expression $\det(A - \lambda I)$ is a polynomial of degree n in λ. In fact, it can be shown that

$$\det(A - \lambda I) = (-1)^n\lambda^n + (-1)^{n-1}\text{trace}(A)\lambda^{n-1} + \cdots [\text{...complicated...}] \cdots + \det(A),$$

where $\text{trace}(A) = a_{11} + a_{22} + \cdots + a_{nn}$. The polynomial $\det(A - \lambda I)$ is called the **characteristic polynomial** of A. By the fundamental theorem of algebra, there are n roots of $\det(A - \lambda I)$ (counting multiplicities) and hence there are n eigenvalues of A (counting multiplicities). The set of all eigenvalues of A is called the **spectrum** of A and denoted by $\sigma(A)$.

In most applications, A will actually be a real matrix. However, as the following example demonstrates, this does not imply that the eigenvalues and eigenvectors are real.

Example 1.2 *The eigenvalues of the matrix* $A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ *are given by*

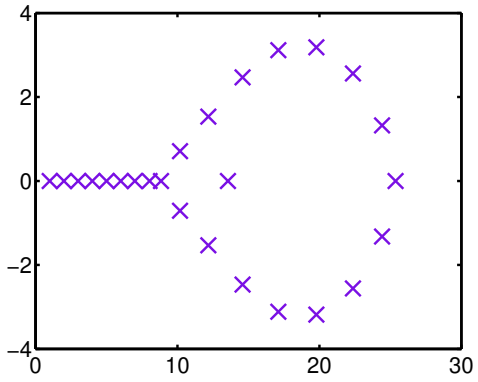
$$0 = \det \begin{pmatrix} -\lambda & 1 \\ -1 & -\lambda \end{pmatrix} = \lambda^2 + 1 \quad \Leftrightarrow \quad \lambda = \pm \mathrm{i},$$

where i *denotes the imaginary unit.*

In principle, one could compute the eigenvalues numerically by forming the characteristic polynomial and applying, e.g., Newton's method to compute its roots. The following MATLAB script realizes this idea for a diagonal matrix with entries $1, 2, 3, \ldots, 25$.

MATLAB

```
A = diag(1:25);
p = poly(A);
plot(roots(p), 'kx');
```



The figure on the right shows that some of the numerically computed roots are far away from the true eigenvalues. The reason is that the characteristic polynomial has coefficients of wildly varying magnitudes, which leads to severe numerical cancellation already when evaluating the polynomial, leave alone computing its roots.

The characteristic polynomial therefore remains primarily a theoretical tool.

If x is an eigenvector belonging to an eigenvalue λ then

$$Ax = \lambda x \quad \Longrightarrow \quad A(\alpha x) = \lambda(\alpha x)$$

for any $\alpha \in \mathbb{C}$. This shows that not only x but *any* nonzero scalar multiple of x is an eigenvector. In fact, the set of all eigenvectors belonging to λ is the null space of $A - \lambda I$. This null space $\mathcal{N}(A - \lambda I)$ is called the **eigenspace** of A belonging to λ.

1.2 Application 1: Vibrating strings

Let us consider a string as displayed in Figure 1.1. The string is clamped at both ends, at $x = 0$

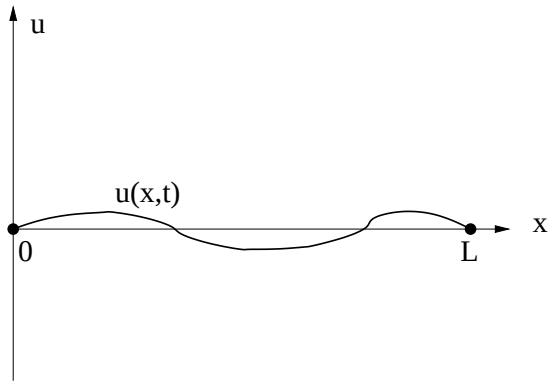


Figure 1.1. *A vibrating string clamped at both ends.*

and $x = L$. The function $u(x, t)$ is the vertical displacement of the string at time t and position $x \in [0, L]$. We assume that the string has a certain initial displacement $u_0(x)$ and velocity $u_1(x)$ at time 0:

$$u(x, 0) = u_0(x), \qquad \frac{\partial}{\partial t} u(x, 0) = u_1(x), \qquad x \in [0, L]. \tag{1.2}$$

As explained in detail below, properties of the string are closely related to solutions of the following equation:

$$-\frac{T}{\mu}w''(x) = \lambda w(x), \quad 0 < x < L, \quad w(0) = w(L) = 0. \tag{1.3}$$

The constant $T > 0$ is the tension and the constant μ is the mass density of the string. A scalar λ is called a **eigenfrequency** (or eigenvalue) of the string if there is a nonzero function w (the so called **eigenfunction**) such that the pair (λ, w) satisfies (1.3).

The general form for the solution of (1.3) without boundary conditions is given by

$$w(x) = \tilde{a}\cos(\sqrt{\lambda\mu/T}x) + \tilde{b}\sin(\sqrt{\lambda\mu/T}x), \quad \tilde{a}, \tilde{b} \in \mathbb{R}.$$

Since $0 = w(0) = \tilde{a}$, the boundary condition $w(L) = 0$ can only be satisfied for a nonzero w if $\sqrt{\lambda\mu/T}L$ is an integer multiple of π. Hence, (1.3) has a nonzero solution if and only if λ takes one of the following values:

$$\lambda_k = \frac{k^2\pi^2 T}{L^2\mu}, \qquad k = 0, 1, 2, 3, \ldots. \tag{1.4}$$

These values $\lambda_0, \lambda_1, \lambda_2, \lambda_3, \ldots$ constitute all eigenfrequencies/eigenvalues. The scalar λ_1 plays a particular role and is called the **fundamental frequency**.

Supplementary material: Derivation and motivation for (1.3)

If the displacement remains relatively small, it follows from Newton's second law that the motion of the string can be well described by

$$\frac{\partial^2}{\partial t^2}u(x,t) = \frac{T}{\mu}\frac{\partial^2}{\partial x^2}u(x,t). \tag{1.5}$$

Moreover, we have the boundary conditions

$$u(0,t) = u(L,t) = 0. \tag{1.6}$$

To solve (1.5)–(1.6), we forget the initial conditions (1.2) for a moment and assume **separation of variables**, that is, u can be written as $u(x,t) = v(t)w(x)$ for some functions v and w with $w(0) = w(L) = 0$. We have

$$\frac{\partial^2}{\partial t^2}u(t,x) = v''(t)w(x), \qquad \frac{\partial^2}{\partial x^2}u(t,x) = v(t)w''(x),$$

and inserting these relations into (1.5) gives

$$\frac{v''(t)}{v(t)} = \frac{T}{\mu}\frac{w''(x)}{w(x)} =: -\lambda. \tag{1.7}$$

It follows that λ is a constant. From the first equality in (1.7), we obtain the equation $v''(t) = -\lambda v(t)$, which has the general solution

$$v(t) = a\cos(\sqrt{\lambda}t) + b\sin(\sqrt{\lambda}t), \quad a, b \in \mathbb{R}.$$

From the second equality in (1.7), we obtain the equation (1.3). By superposition of all separable solutions corresponding to the eigenvalues (1.4), the general solution of (1.5)–(1.6) is given by

$$u(x,t) = \sum_{k=1}^{\infty} \sin(k\pi L^{-1}x)(a_k\cos(\sqrt{\lambda_k}t) + b_k\sin(\sqrt{\lambda_k}t)). \tag{1.8}$$

The coefficients $a_k, b_k \in \mathbb{R}$ are determined from the initial conditions:

$$\begin{aligned} u_0(x) &= u(x,0) = \sum_{k=1}^{\infty} a_k \sin(\pi k L^{-1}x) \\ u_1(x) &= \frac{\partial}{\partial t}u(x,0) = \sum_{k=1}^{\infty} b_k\sqrt{\lambda_k}\sin(\pi k L^{-1}x) \end{aligned}$$

Hence a_k, b_k are the (scaled) Fourier coefficients of u_0, u_1 with respect to the orthogonal basis $\{\sin(\pi kx)\colon k = 1, 2, \ldots\}$ of some subspace in $L_2(0, L)$. In most practical situations, these Fourier coefficients will decay rapidly as $k \to \infty$. Consequently, the first few terms in the sum (1.8) will be dominant. In fact, when using strings in music (e.g., guitar strings), it is usually desirable that only the frequency corresponding to the first term(s) for $k = 1$ is audible. This motivates the name fundamental frequency for λ_1.

1.2.1 Sturm-Liouville eigenvalue problem

In the example above, we were fortunate that the solution to the problem (1.3) could be written down explicitly. This is quite exceptional and will not be possible for more elaborate models.

The so called **Sturm-Liouville eigenvalue problem** is a generalization of (1.3):

$$-(p(x)w'(x))' + q(x)w(x) = \lambda w(x), \qquad w(0) = w(L) = 0. \tag{1.9}$$

Again, the task is to find values of λ for which (1.9) has a nontrivial solution w. It is known that (1.9) has infinitely many real positive eigenvalues $0 < \lambda_1 \le \lambda_2 \le \cdots$, $(\lambda_k \underset{k\to\infty}{\longrightarrow} \infty)$. The equation (1.9) has a non-zero solution $w_k(x)$ *only* for these particular values λ_k. In contrast to (1.3), it is in general not possible to write down the eigenvalues and eigenfunctions of (1.9) explicitly. In the following, we discuss two different numerical approaches for this purpose.

1.2.2 Solution of Sturm-Liouville problem by finite differences

We approximate the solution $w(x)$ of (1.9) by its values at the discrete points $x_i = ih$, $h = L/(n+1)$, $i = 1, \ldots, n$.

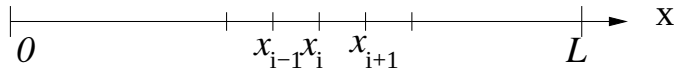


Figure 1.2. *Grid points in the interval* $(0, L)$.

At point x_i we approximate the derivative of a function by a central finite difference quotient. We proceed as follows. First we write

$$\frac{d}{dx}g(x_i) \approx \frac{g(x_{i+\frac{1}{2}}) - g(x_{i-\frac{1}{2}})}{h}.$$

For $g = p\frac{dw}{dx}$ we get

$$g(x_{i+\frac{1}{2}}) = p(x_{i+\frac{1}{2}})\frac{w(x_{i+1}) - w(x_i)}{h}$$

and finally, for $i = 1, \ldots, n$,

$$\begin{aligned} -\frac{d}{dx}\left(p\frac{dw}{dx}(x_i)\right) &\approx -\frac{1}{h}\left[p(x_{i+\frac{1}{2}})\frac{w(x_{i+1}) - w(x_i)}{h} - p(x_{i-\frac{1}{2}})\frac{w(x_i) - w(x_{i-1})}{h}\right] \\ &= \frac{1}{h^2}\left[-p(x_{i-\frac{1}{2}})w_{i-1} + (p(x_{i-\frac{1}{2}}) + p(x_{i+\frac{1}{2}}))w_i - p(x_{i+\frac{1}{2}})w_{i+1}\right]. \end{aligned}$$

Note that at the interval endpoints $w_0 = w_{n+1} = 0$.

We can collect all equations into a matrix equation,

$$\begin{bmatrix} \frac{p(x_{\frac{1}{2}})+p(x_{\frac{3}{2}})}{h^2} + q(x_1) & \frac{-p(x_{\frac{3}{2}})}{h^2} & & \\ \frac{-p(x_{\frac{3}{2}})}{h^2} & \frac{p(x_{\frac{3}{2}})+p(x_{\frac{5}{2}})}{h^2} + q(x_2) & \frac{-p(x_{\frac{5}{2}})}{h^2} & \\ & \frac{-p(x_{\frac{5}{2}})}{h^2} & \ddots & \ddots \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \lambda \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix}$$

or, briefly,

$$A\mathbf{w} = \lambda\mathbf{w}.$$

This happens to be the eigenvalue problem for a matrix A we defined in Section 1.1! By construction, A is symmetric and tridiagonal. One can show that it is positive definite as well.

1.2.3 Solution of Sturm-Liouville problem by finite elements

By multiplying (1.9) with a test function ϕ and integrating over the domain, we can rewrite the Sturm-Liouville problem in the following form.

Find a function w with $w(0) = w(L) = 0$ such that

$$\int_0^L \left[-(p(x)w'(x))' + q(x)w(x) - \lambda w(x)\right]\phi(x)dx = 0$$

for all sufficiently smooth functions ϕ that satisfy $\phi(0) = \phi(L) = 0$.

We integrate by parts and obtain the so-called weak form of (1.9).

Find a function w with $w(0) = w(L) = 0$ such that

$$\int_0^L [p(x)w'(x)\phi'(x) + q(x)w(x)\phi(x) - \lambda w(x)\phi(x)]\, dx = 0 \qquad (1.10)$$

for all sufficiently smooth functions ϕ that satisfy $\phi(0) = \phi(L) = 0$.

Remark 1.3 *In the formulation* (1.10)*, it is crucial to impose the correct smoothness conditions on w and ϕ. It turns out that the notion of classical (once or twice) differentiability is too strong and inappropriate. On the one hand, this would not allow for a piecewise constant p and on the other hand, we will make use of non-differentiable basis functions. Instead, we require w and ϕ to be* weakly *differentiable. In terms of Sobolev spaces: $w, \phi \in H_0^1([0, L])$.*

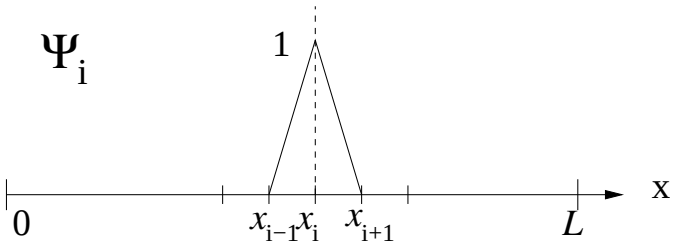


Figure 1.3. *A basis function of the finite element space: a hat function.*

To discretize (1.10), we write w as a linear combination

$$w(x) = \sum_{i=1}^{n} \xi_i\, \Psi_i(x), \qquad (1.11)$$

where

$$\Psi_i(x) = \left(1 - \frac{|x - x_i|}{h}\right)_+ = \max\{0,\ 1 - \frac{|x - x_i|}{h}\},$$

is the function that is linear in each interval (x_i, x_{i+1}) and satisfies

$$\Psi_i(x_k) = \delta_{ik} := \begin{cases} 1, & i = k, \\ 0, & i \neq k. \end{cases}$$

An example of such a *hat function* is given in Figure 1.3.

We now replace w in (1.10) by the linear combination (1.11), and replace testing 'against all ϕ' by testing against all Ψ_j. In this way (1.10) becomes

$$\int_0^L \left(-p(x)(\sum_{i=1}^{n} \xi_i\, \Psi_i'(x))\Psi_j'(x) + (q(x) - \lambda) \sum_{i=1}^{n} \xi_i\, \Psi_i(x)\Psi_j(x) \right) dx = 0 \quad \text{for all } j.$$

This leads to the so called **Rayleigh–Ritz–Galerkin** equations

$$\sum_{i=1}^{n} \xi_i \int_0^L \big(p(x)\Psi_i'(x)\Psi_j'(x) + (q(x)-\lambda)\Psi_i(x)\Psi_j(x)\big)\, dx = 0 \quad \text{for all } j. \tag{1.12}$$

The unknowns in (1.12) are the n values ξ_i and the eigenvalue λ. In matrix notation (1.12) becomes

$$A\xi = \lambda M\xi \tag{1.13}$$

with

$$a_{ij} = \int_0^L \big(p(x)\Psi_i'\Psi_j' + q(x)\Psi_i\Psi_j\big)\, dx \quad \text{and} \quad m_{ij} = \int_0^L \Psi_i\Psi_j\, dx$$

For the specific case $p(x) = 1 + x$ and $q(x) = 1$ we get

$$\begin{aligned} a_{kk} &= \int_{(k-1)h}^{kh} \left[(1+x)\frac{1}{h^2} + \left(\frac{x-(k-1)h}{h}\right)^2\right] dx \\ &\quad + \int_{kh}^{(k+1)h} \left[(1+x)\frac{1}{h^2} + \left(\frac{(k+1)h-x}{h}\right)^2\right] dx = 2(n+1+k) + \frac{2}{3}\frac{1}{n+1} \\ a_{k,k+1} &= \int_{kh}^{(k+1)h} \left[(1+x)\frac{1}{h^2} + \frac{(k+1)h-x}{h}\cdot\frac{x-kh}{h}\right] dx = -n - \frac{3}{2} - k + \frac{1}{6}\frac{1}{n+1} \end{aligned}$$

In the same way we get

$$M = \frac{1}{6(n+1)} \begin{bmatrix} 4 & 1 & & \\ 1 & 4 & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{bmatrix}$$

Note that both matrices A and M are symmetric tridiagonal and positive definite.

Because of the presence of the matrix M, the setting (1.13) is slightly different from our definition of matrix eigenvalue problems. A problem of the form $A\mathbf{x} = \lambda M\mathbf{x}$ is called **generalized eigenvalue problem**. As M is invertible, this can be reduced to the standard eigenvalue problem $M^{-1}A\mathbf{x} = \lambda\mathbf{x}$. This has the disadvantage that $M^{-1}A$ is not symmetric. For a symmetric positive definite matrix M, it is preferable to compute its Cholesky factorization $M = R^{\mathsf{T}}R$. After setting $\mathbf{y} := R\mathbf{x}$, this leads to the standard eigenvalue problem $\tilde{A}\mathbf{y} = \lambda\mathbf{y}$ with the symmetric matrix $\tilde{A} := R^{-\mathsf{T}}AR^{-1}$.

1.2.4 A numerical experiment

We consider the above 1-dimensional eigenvalue problem

$$-((1+x)w'(x))' + w(x) = \lambda w(x), \qquad w(0) = w(1) = 0, \tag{1.14}$$

and solve it with the finite difference and finite element methods presented in Sections 1.2.2 and 1.2.3, respectively. The results are given in Table 1.1. For both, the finite difference and finite element methods, the eigenvalues exhibit quadratic convergence rates. If the mesh width h is reduced by a factor of 2, the error in the eigenvalues is reduced by the factor $2^2 = 4$.

Finite difference method				
k	$\lambda_k(n=10)$	$\lambda_k(n=20)$	$\lambda_k(n=40)$	$\lambda_k(n=80)$
1	15.245	15.312	15.331	15.336
2	56.918	58.048	58.367	58.451
3	122.489	128.181	129.804	130.236
4	206.419	224.091	229.211	230.580
5	301.499	343.555	355.986	359.327
6	399.367	483.791	509.358	516.276
7	492.026	641.501	688.398	701.185
8	578.707	812.933	892.016	913.767
9	672.960	993.925	1118.969	1153.691
10	794.370	1179.947	1367.869	1420.585

Finite element method				
k	$\lambda_k(n=10)$	$\lambda_k(n=20)$	$\lambda_k(n=40)$	$\lambda_k(n=80)$
1	15.447	15.367	15.345	15.340
2	60.140	58.932	58.599	58.511
3	138.788	132.657	130.979	130.537
4	257.814	238.236	232.923	231.531
5	426.223	378.080	365.047	361.648
6	654.377	555.340	528.148	521.091
7	949.544	773.918	723.207	710.105
8	1305.720	1038.433	951.392	928.983
9	1702.024	1354.106	1214.066	1178.064
10	2180.159	1726.473	1512.784	1457.733

Table 1.1. *Numerical solutions of problem* (1.14)

1.3 Application 2: Laplace eigenvalue problem

In this section, we consider the **Laplace eigenvalue problem**

$$-\Delta u(\mathbf{x}) = \lambda u(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{1.15}$$

on a domain $\Omega \subset \mathbb{R}^d$ with $d = 2$ or $d = 3$. Recall that $\Delta = \frac{\partial^2}{\partial x_1^2} + \cdots + \frac{\partial^2}{\partial x_d^2}$ is the Laplace operator. Additionally, we consider mixed boundary conditions

$$u(\mathbf{x}) = 0, \qquad \mathbf{x} \in C_1 \subset \partial\Omega, \tag{1.16}$$

$$\frac{\partial u}{\partial n}(\mathbf{x}) + \alpha(\mathbf{x})u(\mathbf{x}) = 0, \qquad \mathbf{x} \in C_2 \subset \partial\Omega, \tag{1.17}$$

where C_1 and C_2 are *disjoint* subsets of the boundary $\partial\Omega$ with $C_1 \cup C_2 = \partial\Omega$. $\frac{\partial u}{\partial n}$ denotes the derivative of u in direction of the outer normal vector $\mathbf{n}$.

Supplementary material: Example and motivation for (1.15)
The instationary temperature distribution $u(\mathbf{x},t)$ in an insulated container can be modeled with the equations

$$\begin{aligned} \frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x},t) &= 0, && \mathbf{x}\in\Omega,\ t>0,\\ \frac{\partial u(\mathbf{x},t)}{\partial n} &= 0, && \mathbf{x}\in\partial\Omega,\ t>0,\\ u(\mathbf{x},0) &= u_0(\mathbf{x}), && \mathbf{x}\in\Omega. \end{aligned} \tag{1.18}$$

Here, $\Omega\subset\mathbb{R}^3$ is a 3-dimensional domain with boundary $\partial\Omega$ and $u_0(\mathbf{x}), \mathbf{x}=(x_1,x_2,x_3)^\mathsf{T}\in\mathbb{R}^3$, is a given bounded, sufficiently smooth function. Once again, separation of variables is employed to solve the heat equation. We write u in the form $u(\mathbf{x},t)=v(t)w(\mathbf{x})$. Suppose that a constant λ can be found such that

$$\begin{aligned} \Delta w(\mathbf{x}) + \lambda w(\mathbf{x}) = 0, \quad w(\mathbf{x})\neq 0, &\quad \mathbf{x} \text{ in } \Omega,\\ \frac{\partial w(\mathbf{x},t)}{\partial n} = 0, &\quad \mathbf{x} \text{ on } \partial\Omega. \end{aligned} \tag{1.19}$$

Then the product $u=vw$ is a solution of (1.18) if and only if $\frac{dv(t)}{dt}+\lambda v(t)=0$. This a simple ordinary differential equation with all solutions of the form $a\cdot\exp(-\lambda t)$ for some scalar a. A value λ, for which (1.19) has a *nontrivial* (i.e., a nonzero) solution w is called an *eigenvalue*; w is called a *corresponding eigenfunction*.
If λ_n is an eigenvalue of problem (1.19) with corresponding eigenfunction w_n, then

$$e^{-\lambda_n t} w_n(\mathbf{x})$$

is a solution of the first two equations in (1.18). It is known that equation (1.19) has *infinitely many* real eigenvalues $0\le\lambda_1\le\lambda_2\le\cdots, (\lambda_n \underset{t\to\infty}{\longrightarrow}\infty)$. Multiple eigenvalues are counted according to their multiplicity. Under relatively mild assumption on the shape of Ω, an arbitrary bounded piecewise continuous function can be represented as a linear combination of the eigenfunctions $w_1, w_2, \ldots$. Therefore, the solution of (1.18) can be written in the form

$$u(\mathbf{x},t)=\sum_{n=1}^{\infty} c_n e^{-\lambda_n t} w_n(\mathbf{x}), \tag{1.20}$$

where the coefficients c_n are determined such that

$$u_0(\mathbf{x})=\sum_{n=1}^{\infty} c_n w_n(\mathbf{x}).$$

The smallest eigenvalue of (1.19) is $\lambda_1=0$ with $w_1=1$ and $\lambda_2>0$. Therefore we see from (1.20) that $u(\mathbf{x},t)\underset{t\to\infty}{\longrightarrow} c_1$. Thus, in the limit (i.e., as t goes to infinity), the temperature will be constant in the whole container. The convergence rate towards this equilibrium is determined by the smallest *positive* eigenvalue λ_2 of (1.19):

$$\begin{aligned} |u(\mathbf{x},t)-c_1| = \left|\sum_{n=2}^{\infty} c_n e^{-\lambda_n t} w_n(\mathbf{x})\right| &\le \sum_{n=2}^{\infty} |e^{-\lambda_n t}||c_n w_n(\mathbf{x})|\\ &\le e^{-\lambda_2 t}\sum_{n=2}^{\infty}|c_n w_n(\mathbf{x})| \le e^{-\lambda_2 t}|u_0(\mathbf{x})|. \end{aligned}$$

1.3.1 Solution by the finite difference method

In the following, we restrict ourselves on *two-dimensional* domains and write (x,y) instead of (x_1,x_2). In general it is not possible to solve a problem of the form (1.15)–(1.17) exactly (analytically). In this section, we *illustrate* how the finite difference method can be used for approximating the solution of (1.15)–(1.17) numerically.

We assume for simplicity that the domain Ω is a square with sides of length 1: $\Omega=(0,1)\times$

$(0,1)$. We consider the eigenvalue problem

$$\begin{aligned} -\Delta u(x,y) &= \lambda u(x,y), && 0<x,y<1 \\ u(0,y) = u(1,y) &= u(x,0) = 0, && 0<x,y<1, \\ \frac{\partial u}{\partial n}(x,1) &= 0, && 0<x<1. \end{aligned}$$

This eigenvalue problem occurs in the computation of eigenfrequencies and eigenmodes of a homogeneous quadratic membrane with three fixed and one free side. It can be solved analytically by separation of the two spatial variables x and y. The eigenvalues are

$$\lambda_{k,l} = \left(k^2 + \frac{(2l-1)^2}{4}\right)\pi^2, \quad k,l \in \mathbb{N},$$

and the corresponding eigenfunctions are

$$u_{k,l}(x,y) = \sin k\pi x \sin \frac{2l-1}{2}\pi y.$$

In the finite difference method one proceeds by defining a rectangular grid with grid points $(x_i, y_j), 0 \le i,j \le N$. The coordinates of the grid points are

$$(x_i, y_j) = (ih, jh), \qquad h = 1/N.$$

By a Taylor expansion one can show for sufficiently smooth functions u that

$$\begin{aligned} -\Delta u(x,y) = \frac{1}{h^2}&(4u(x,y) - u(x-h,y) - u(x+h,y) - u(x,y-h) - u(x,y+h)) \\ &+ O(h^2). \end{aligned}$$

It is therefore straightforward to replace the differential equation $\Delta u(x,y) + \lambda u(x,y) = 0$ by a difference equation at the interior grid points

$$4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = \lambda h^2 u_{i,j}, \quad 0 < i,j < N. \tag{1.21}$$

We consider the unknown variables $u_{i,j}$ as approximations of the eigenfunctions at the grid points (i,j): $u_{i,j} \approx u(x_i, x_j)$. The Dirichlet boundary conditions are replaced by the equations

$$u_{i,0} = u_{i,N} = u_{0,i}, \qquad 0 < i < N. \tag{1.22}$$

At the points at the upper boundary of Ω we first take the difference equation (1.21)

$$4u_{i,N} - u_{i-1,N} - u_{i+1,N} - u_{i,N-1} - u_{i,N+1} = \lambda h^2 u_{i,N}, \quad 0 \le i \le N. \tag{1.23}$$

The value $u_{i,N+1}$ corresponds to a grid point *outside* of the domain! However, the Neumann boundary conditions suggest to reflect the domain at the upper boundary and to extend the eigenfunction symmetrically beyond the boundary. This procedure leads to the equation $u_{i,N+1} = u_{i,N-1}$. Plugging this into (1.23) and multiplying the new equation by the factor $1/2$ gives

$$2u_{i,N} - \frac{1}{2}u_{i-1,N} - \frac{1}{2}u_{i+1,N} - u_{i,N-1} = \frac{1}{2}\lambda h^2 u_{i,N}, \quad 0 < i < N. \tag{1.24}$$

In summary, from (1.21) and (1.24), taking into account that (1.22), we get the matrix equation

$$\begin{pmatrix} 4 & -1 & 0 & -1 & & & & & & & & \\ -1 & 4 & -1 & 0 & -1 & & & & & & & \\ 0 & -1 & 4 & 0 & 0 & -1 & & & & & & \\ -1 & 0 & 0 & 4 & -1 & 0 & -1 & & & & & \\ & -1 & 0 & -1 & 4 & -1 & 0 & -1 & & & & \\ & & -1 & 0 & -1 & 4 & 0 & 0 & -1 & & & \\ & & & -1 & 0 & 0 & 4 & -1 & 0 & -1 & & \\ & & & & -1 & 0 & -1 & 4 & -1 & 0 & -1 & \\ & & & & & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\ & & & & & & -1 & 0 & 0 & 2 & -\frac{1}{2} & 0 \\ & & & & & & & -1 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} \\ & & & & & & & & -1 & 0 & -\frac{1}{2} & 2 \end{pmatrix} \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3} \end{pmatrix}$$

$$= \lambda h^2 \begin{pmatrix} 1 & & & & & & & & & & & \\ & 1 & & & & & & & & & & \\ & & 1 & & & & & & & & & \\ & & & 1 & & & & & & & & \\ & & & & 1 & & & & & & & \\ & & & & & 1 & & & & & & \\ & & & & & & 1 & & & & & \\ & & & & & & & 1 & & & & \\ & & & & & & & & 1 & & & \\ & & & & & & & & & \frac{1}{2} & & \\ & & & & & & & & & & \frac{1}{2} & \\ & & & & & & & & & & & \frac{1}{2} \end{pmatrix} \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3} \end{pmatrix}.$$

For arbitrary $N > 1$ we define

$$\mathbf{u}_i := \begin{pmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,N-1} \end{pmatrix} \in \mathbb{R}^{N-1}, \quad T := \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \in \mathbb{R}^{(N-1)\times(N-1)}.$$

In this way we obtain from (1.21), (1.22), (1.24) the generalized matrix eigenvalue problem

$$\begin{pmatrix} T & -I & & \\ -I & T & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & \frac{1}{2}T \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N-1} \\ \mathbf{u}_N \end{pmatrix} = \lambda h^2 \begin{pmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & & \frac{1}{2}I \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N-1} \\ \mathbf{u}_N \end{pmatrix}$$

of size $N(N-1)$. This can be easily turned into a standard eigenvalue problem with the trick described in Section 1.2.3.

In real-world applications, the domain can often not be easily covered with a rectangular grid. In this situation, and if boundary conditions are complicated, the method of finite differences can be difficult to implement and the finite element method is often the method of choice.

1.3.2 Solution by the finite element method (FEM)

Let $(\lambda, u) \in \mathbb{R} \times V$ be an eigenpair of problem (1.15)–(1.17). Then

$$\int_\Omega (\Delta u + \lambda u) v \, dx \, dy = 0, \quad \forall v \in V,$$

for a suitable vector space V of functions satisfying the Dirichlet boundary conditions (1.16). By partial integration (Green's formula) this becomes

$$\int_\Omega \nabla u \nabla v \, dx \, dy + \int_{\partial\Omega} \alpha \, u \, v \, ds = \lambda \int_\Omega u \, v \, dx \, dy, \quad \forall v \in V,$$

or

$$a(u, v) = \lambda(u, v), \qquad \forall v \in V$$

where

$$a(u, v) = \int_\Omega \nabla u \, \nabla v \, dx \, dy + \int_{\partial\Omega} \alpha \, u \, v \, ds, \quad \text{and} \quad (u, v) = \int_\Omega u \, v \, dx \, dy.$$

We use the Sobolev space $V = H^1_{C_1}$ of weakly differentiable functions with zero trace on C_1. Together with the usual H^1 scalar product, V becomes a Hilbert space H.

To summarize, we arrive at the weak formulation of the Laplace eigenvalue problem:

$$\begin{aligned} &\text{Find } (\lambda, u) \in \mathbb{R} \times V \text{ such that} \\ &a(u, v) = \lambda(u, v) \quad \forall v \in V. \end{aligned} \tag{1.25}$$

The Rayleigh–Ritz–Galerkin method

In the Rayleigh–Ritz–Galerkin method one proceeds as follows: A set of linearly independent functions

$$\phi_1(x, y), \cdots, \phi_n(x, y) \in V, \tag{1.26}$$

are chosen. These functions span a *subspace* V_n of V. Then, problem (1.25) is approximated by replacing V with V_n:

$$\begin{aligned} &\text{Find } (\lambda, u) \in \mathbb{R} \times V_n \text{ such that} \\ &a(u, v) = \lambda(u, v) \quad \forall v \in V_n. \end{aligned} \tag{1.27}$$

With the Ritz ansatz $u = \sum_{i=1}^n x_i \phi_i$, the equation (1.27) becomes

$$\begin{aligned} &\text{Find } (\lambda, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^n \text{ such that} \\ &\sum_{i=1}^n x_i a(\phi_i, v) = \lambda \sum_{i=1}^n x_i (\phi_i, v), \quad \forall v \in V_n. \end{aligned} \tag{1.28}$$

This equation must hold *for all* $v \in V_n$, in particular for $v = \phi_1, \cdots, \phi_n$. But since the functions $\phi_i, 1 \le i \le n$, form a basis of V_n, (1.28) is equivalent to

$$\sum_{i=1}^n x_i a(\phi_i, \phi_j) = \lambda \sum_{i=1}^n x_i (\phi_i, \phi_j), \quad 1 \le j \le n.$$

This is a matrix eigenvalue problem of the form

$$A\mathbf{x} = \lambda M \mathbf{x} \tag{1.29}$$

where

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}, \quad M = \begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{pmatrix}$$

with

$$a_{ij} = a(\phi_i, \phi_j) = \int_\Omega \nabla\phi_i \, \nabla\phi_j \, dx \, dy + \int_{\partial\Omega} \alpha \, \phi_i \, \phi_j \, ds$$

and

$$m_{ij} = (\phi_i, \phi_j) = \int_\Omega \phi_i \, \phi_j \, dx \, dy.$$

The **finite element method (FEM)** is a *special case* of the Rayleigh–Ritz method. In the FEM the subspace V_n and in particular the basis $\{\phi_i\}$ are chosen in a particularly clever way. For simplicity, we assume that the domain Ω is a simply connected domain with a polygonal boundary, c.f. Figure 1.4. (This means that the boundary is composed of straight line segments entirely.) This domain is now partitioned into triangular subdomains $T_1, \cdots, T_N$, such that

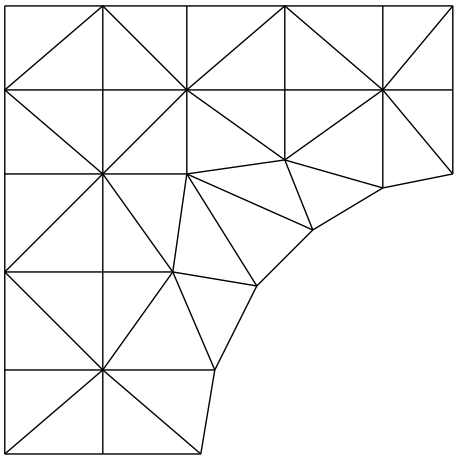


Figure 1.4. *Triangulation of a domain* Ω

$$\begin{aligned} T_i \cap T_j &= \emptyset, \qquad i \neq j, \\ \bigcup_e \overline{T_e} &= \overline{\Omega}. \end{aligned}$$

Finite element spaces for solving (1.15)–(1.17) are typically composed of functions that are *continuous* in Ω and are *polynomials* on the individual subdomains T_e. Such functions are called *piecewise polynomials*.

An essential issue is the selection of the *basis* of the finite element space. If $S_1 \subset V$ is the space of continuous, piecewise linear functions (the restriction to T_e is a polynomial of degree 1) then a function in S_1 is uniquely determined by its values at the vertices of the triangles. Let these *nodes*, except those on the boundary portion C_1, be numbered from 1 to n, see Fig. 1.5. Let the coordinates of the i-th node be (x_i, y_i). Then $\phi_i(x, y) \in S_1$ is defined by

$$\phi_i((x_j, y_j)) := \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{1.30}$$

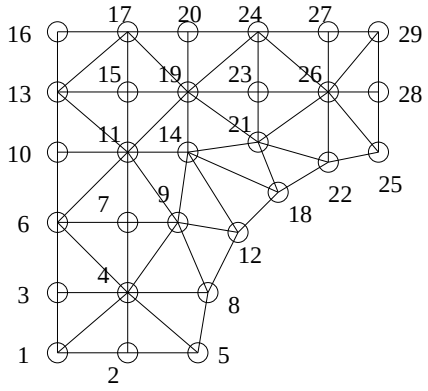


Figure 1.5. *Numbering of nodes on Ω (piecewise linear polynomials)*

A typical basis function ϕ_i is sketched in Figure 1.6.

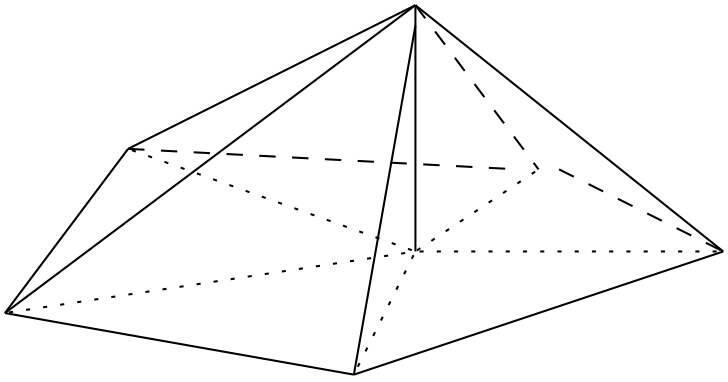


Figure 1.6. *A piecewise linear basis function (or hat function)*

Another often used finite element element space is $S_2 \subset V$, the space of continuous, piecewise quadratic polynomials. These functions are (or can be) uniquely determined by their values at the vertices and edge midpoints of the triangle. The basis functions are defined according to (1.30). There are two kinds of basis functions ϕ_i now, first those that are 1 at a vertex and second those that are 1 in an edge midpoint, cf. Figure 1.7. One immediately sees that for most $i \neq j$

$$a(\phi_i, \phi_j) = 0, \quad (\phi_i, \phi_j) = 0.$$

Therefore the matrices A and M in (1.29) will be **sparse**. The matrix M is positive definite as

$$\mathbf{x}^\mathsf{T} M \mathbf{x} = \sum_{i,j=1}^{N} x_i x_j m_{ij} = \sum_{i,j=1}^{N} x_i x_j (\phi_i, \phi_j) = (u, u) > 0, \quad u = \sum_{i=1}^{N} x_i \phi_i \neq 0,$$

because $\phi_1, \ldots, \phi_N$ are linearly independent. Similarly it is shown that

$$\mathbf{x}^\mathsf{T} A \mathbf{x} \geq 0.$$

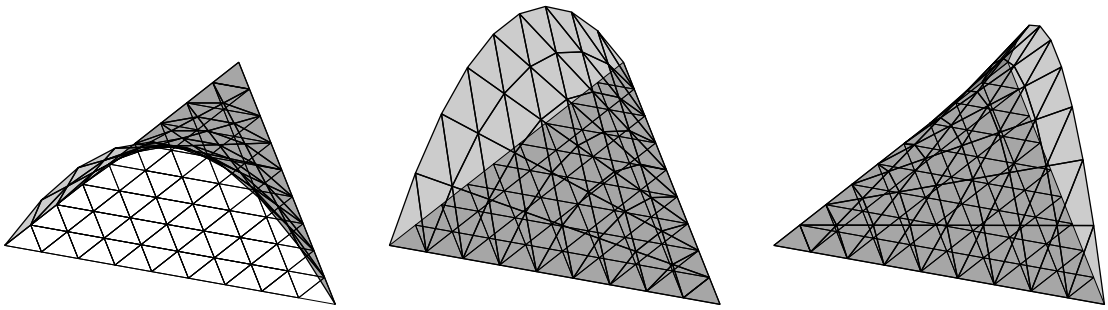


Figure 1.7. *A piecewise quadratic basis function corresponding to a edge midpoint*

It is possible to have $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} = 0$ for a nonzero vector $\mathbf{x}$. This is the case if the constant function $u = 1$ is contained in the finite element space, for example if Neumann boundary conditions $\frac{\partial u}{\partial n} = 0$ are posed on the whole boundary $\partial\Omega$. Then,

$$u(x, y) = 1 = \sum_i \phi_i(x, y),$$

i.e., we have $\mathbf{x}^\mathsf{T}\mathbf{A}\mathbf{x} = 0$ for $\mathbf{x} = [1, 1, \ldots, 1]$.

1.3.3 A numerical example

We want to determine the acoustic eigenfrequencies and corresponding modes in the interior of a car. This is of interest in the manufacturing of cars, since an appropriate shape of the form of the interior can suppress the often unpleasant droning of the motor. The problem is three-dimensional, but by separation of variables the problem can be reduced to two dimensions. If rigid, acoustically hard walls are assumed, the mathematical model of the problem is again the Laplace eigenvalue problem (1.19) together with Neumann boundary conditions. The domain is given in Fig. 1.8 where three finite element triangulations are shown with 87 (grid_1), 298 (grid_2), and 1095 (grid_3) vertices (nodes), respectively. The results obtained with piecewise linear

Finite element method			
k	$\lambda_k(\text{grid}_1)$	$\lambda_k(\text{grid}_2)$	$\lambda_k(\text{grid}_3)$
1	0.0000	-0.0000	0.0000
2	0.0133	0.0129	0.0127
3	0.0471	0.0451	0.0444
4	0.0603	0.0576	0.0566
5	0.1229	0.1182	0.1166
6	0.1482	0.1402	0.1376
7	0.1569	0.1462	0.1427
8	0.2162	0.2044	0.2010
9	0.2984	0.2787	0.2726
10	0.3255	0.2998	0.2927

Table 1.2. *Numerical solutions of acoustic vibration problem*

polynomials are listed in Table 1.2. From the results we notice again the quadratic convergence

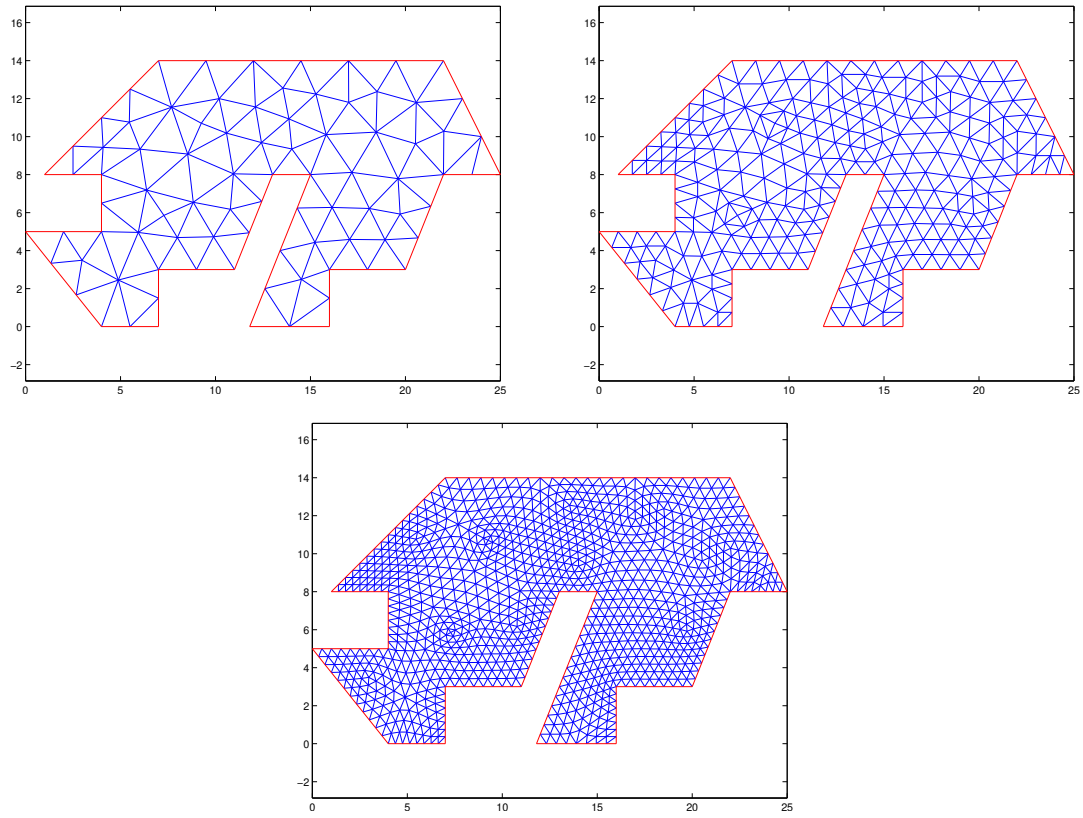


Figure 1.8. *Three meshes for the car length cut*

rate. The smallest eigenvalue is always zero. The corresponding eigenfunction is the constant function. This function can be represented exactly by the finite element spaces, whence its value is correct (up to rounding error).

The fourth eigenfunction of the acoustic vibration problem is displayed in Fig. 1.9. The physical meaning of the function value is the difference of the pressure at a given location to the normal pressure. Large amplitudes thus means that the corresponding noise is very much noticeable.

1.4 Similarity transformations

All methods for computing the eigenvalues and eigenvectors of a *small* matrix A are based on transforming the matrix A to some simpler form, from which the eigenvalues can be read off. The following definition and theorem characterizes the allowable transformations.

Definition 1.4 *Two matrices $A, B \in \mathbb{C}^{n\times n}$ are called* **similar** *to each other if there is an invertible matrix $S \in \mathbb{C}^{n\times n}$ such that*

$$S^{-1}AS = B.$$

The mapping $A \mapsto S^{-1}AS$ is called **similarity transformation**.

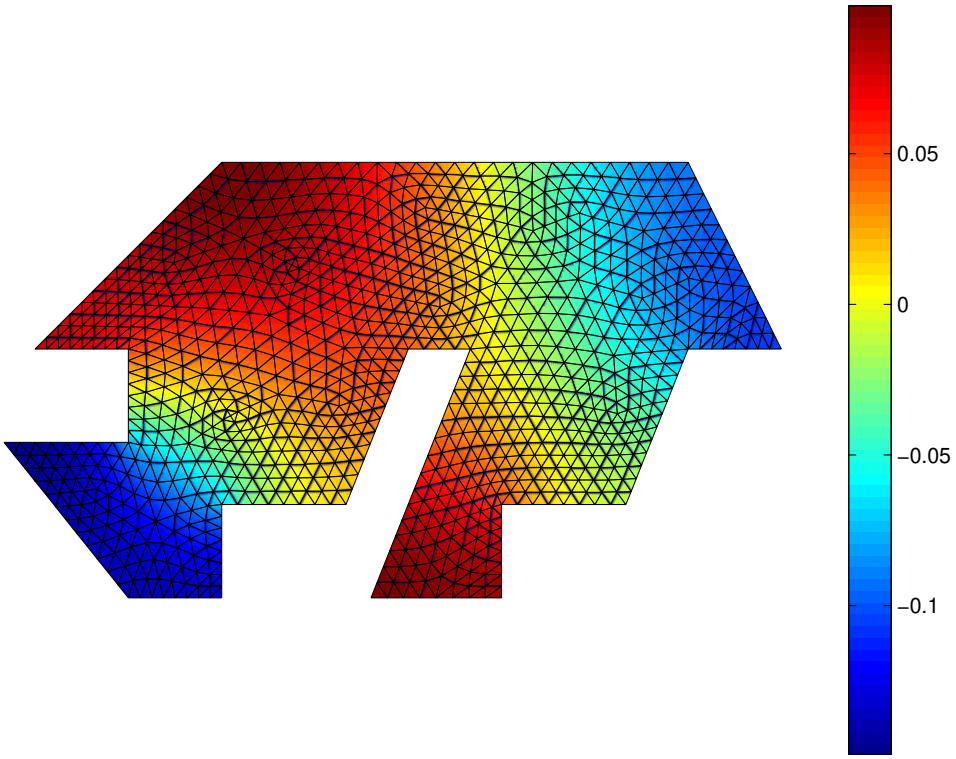


Figure 1.9. *Fourth eigenmode of the acoustic vibration problem*

Theorem 1.5 *The eigenvalues of a matrix do not change under similarity transformations.*

Proof. Let $(\lambda, \mathbf{x})$ be an eigenpair of A and $B = S^{-1}AS$. Then

$$BS^{-1}\mathbf{x} = S^{-1}A\underbrace{SS^{-1}}_{=I}\mathbf{x} = S^{-1}A\mathbf{x} = \lambda S^{-1}\mathbf{x}$$

shows that $(\lambda, S^{-1}\mathbf{x})$ is an eigenpair of B. This shows that every eigenvalue of A is an eigenvalue of B. In the other direction, it can be shown analogously that if $(\lambda, \mathbf{x})$ is an eigenpair of B then $(\lambda, S\mathbf{x})$ is an eigenpair of A. ∎

Let $\lambda_1, \ldots, \lambda_n$ be the eigenvalues of A with eigenvectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Then the relations $A\mathbf{x}_i = \lambda_i \mathbf{x}_i$ for $i = 1, \ldots, n$ can be written in matrix form as

$$\begin{aligned} A \cdot \big(\mathbf{x}_1, \ldots, \mathbf{x}_n\big) &= \big(A\mathbf{x}_1, \ldots, A\mathbf{x}_n\big) \\ &= \big(\lambda_1\mathbf{x}_1, \ldots, \lambda_n\mathbf{x}_n\big) \\ &= \big(\mathbf{x}_1, \ldots, \mathbf{x}_n\big) \cdot \mathrm{diag}(\lambda_1, \ldots, \lambda_n), \end{aligned}$$

where $\operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ denotes an $n \times n$ diagonal matrix with diagonal entries $\lambda_1, \ldots, \lambda_n$. *Under the assumption* that the eigenvectors are $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are linearly independent, the matrix $S := \big(\mathbf{x}_1, \ldots, \mathbf{x}_n\big)$ transforms A to diagonal form:

$$S^{-1}AS = \operatorname{diag}(\lambda_1, \ldots, \lambda_n).$$

We then say that A is diagonalizable. Such a diagonalizing transformation is not always possible as the following example shows.

Example 1.6 *The matrix* $A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ *has the eigenvalue* 0. *Hence if there was a matrix* S *diagonalizing* A, *then* $S^{-1}AS = \operatorname{diag}(0,0) = 0$, *which implies* $A = S0S^{-1} = 0$, *a contradiction.*

A basic result from linear algebra shows that eigenvectors belonging to mutually different eigenvalues are linearly independent. Hence, a sufficient (but not necessary) condition for diagonalizability is that A has n mutually different eigenvalues. For completeness, we give the classical linear algebra result that fully characterizes diagonalizability.

Definition 1.7 *Let* λ *be an eigenvalue of* A. *The dimension of* $\mathcal{N}(A - \lambda I)$ *is called the* **geometric multiplicity** *of* λ, *denoted by* $g(\lambda)$. *The multiplicity of* λ *as a root of* $\det(A - \lambda I)$ *is called the* **algebraic multiplicity** *of* λ, *denoted by* $a(\lambda)$.

Theorem 1.8 *There is an invertible matrix* S *such that* $S^{-1}AS$ *is diagonal if and only if* $a(\lambda) = g(\lambda)$ *for every eigenvalue* λ *of* A.

In Example 1.6, the condition of Theorem 1.8 is violated as $a(0) = 2$ but $g(0) = 1$.

In MATLAB, all eigenvalues and eigenvectors of a matrix A are computed by typing `[V,D] = eig(A)`. The result is a matrix `V` containing the eigenvectors in its columns and a diagonal matrix `D` containing the eigenvalues in its diagonal entries. When only the eigenvalues are of interest, unnecessary computational effort is avoided when just typing `e = eig(A)`.

The command `eig` utilizes the QR algorithm, which will be briefly described in Section 2.2. The QR algorithm subsequently reduces A to a simpler form by similarity transformations. Actually, its aim is *not* to reduce A to diagonal form, even when A is diagonalizable! The reason for this becomes clear when considering the matrix

$$A = \begin{pmatrix} \varepsilon & 1 \\ 0 & -\varepsilon \end{pmatrix}, \qquad \varepsilon \approx 0,$$

which has the eigenvalues ε and $-\varepsilon$ with corresponding eigenvectors $\mathbf{x}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\mathbf{x}_2 = \begin{pmatrix} 1 \\ -2\varepsilon \end{pmatrix}$, respectively. Hence, a matrix that diagonalizes A is given by

$$S = \begin{pmatrix} 1 & 1 \\ 0 & -2\varepsilon \end{pmatrix}.$$

For very small ε, this matrix is extremely ill-conditioned; its condition number[1] is given by $\kappa(A) \approx 1/\varepsilon$. Ill-conditioning comes with all sorts of problems in finite-precision arithmetic and should be avoided whenever possible. For example, when the original matrix A is subject to a perturbation $\triangle \in \mathbb{C}^{n \times n}$ with $\|\triangle\| \approx 0$ then

$$S^{-1}(A + \triangle)S = S^{-1}AS + S^{-1}\triangle S.$$

[1] The condition number of an invertible matrix S defined as $\kappa(S) = \|S\|_2 \, \|S^{-1}\|_2$, where $\|\cdot\|_2$ denotes the matrix 2-norm. In MATLAB, $\kappa(S)$ is computed by `cond(S)`.

The perturbation in the transformed matrix can be bounded by

$$\|S^{-1}\triangle S\|_2 \leq \|S^{-1}\|_2\,\|\triangle\|_2\,\|S\|_2 = \kappa(S)\|\triangle\|_2. \tag{1.31}$$

Consequently, when $\kappa(S)$ is large, an innocently looking perturbation $\triangle$ might get magnified by a large factor. For example, if $\triangle$ refers to the roundoff error made when storing the matrix A, this could mean that an algorithm employing such a transformation returns completely inaccurate eigenvalues.

In numerical computations, general similarity transformations should be avoided whenever possible.

The magnification problem (1.31) is completely avoided when using similarity transformations with unitary matrices.

Definition 1.9 *A matrix $U \in \mathbb{C}^{n\times n}$ is called unitary if $U^*U = UU^* = I$. A matrix $Q \in \mathbb{R}^{n\times n}$ is called orthogonal if $Q^{\mathsf{T}}Q = QQ^{\mathsf{T}} = I$.*

Note that U^* denotes the Hermitian transpose of a matrix: $\left[U^*\right]_{ij} = \overline{u_{ji}}$ for $i, j = 1, \ldots, n$.

Because of

$$\|Ux\|_2^2 = (Ux)^*Ux = x^*U^*Ux = x^*x = \|x\|_2^2,$$

unitary matrices preserve the Euclidean norm of a vector. This implies

$$\|U\|_2 := \max_{\|x\|_2=1} \|Ux\|_2 = \max_{\|x\|_2=1} \|x\|_2 = 1,$$

as well as $\|U^{-1}\|_2 = \|U^*\|_2 = 1$. Hence, a unitary (or orthogonal) matrix U is always very well-conditioned: $\kappa(U) = 1$.

When restricting the allowable similarity transformations to unitary matrices, we cannot expect to attain a diagonal matrix. Instead, as we will see below, we can only attain an upper triangular matrix. However, an upper triangular matrix is equally good for the purpose of computing eigenvalues, as shown by the following result.

Lemma 1.10 *Consider an upper block triangular matrix $A = \begin{pmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{pmatrix}$ with $A_{11} \in \mathbb{C}^{k\times k}$ and $A_{22} \in \mathbb{C}^{(n-k)\times(n-k)}$. Then $\sigma(A) = \sigma(A_{11}) \cup \sigma(A_{22})$. In particular, the eigenvalues of an upper triangular matrix A are simply the diagonal entries $a_{11}, a_{22}, \ldots, a_{nn}$.*

Proof. This follows from

$$\det(A - \lambda I) = \det\begin{pmatrix} A_{11} - \lambda I & A_{12} \\ 0 & A_{22} - \lambda \end{pmatrix} = \det(A_{11} - \lambda I)\det(A_{22} - \lambda I).$$

□

Theorem 1.11 (Schur decomposition) *For every $A \in \mathbb{C}^{n\times n}$ there is a unitary matrix $U \in \mathbb{C}^{n\times n}$ such that*

$$U^*AU = T = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ 0 & t_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & t_{n-1,n} \\ 0 & \cdots & 0 & t_{nn} \end{pmatrix}, \tag{1.32}$$

that is, T is upper triangular.

Proof. The proof is by induction with respect to n. It is clear that the result of the theorem holds for $n = 1$, by simply setting $U = 1$ and $T = A$ in this case. Suppose that the result holds for all matrices of size $(n-1) \times (n-1)$.

Let $(\lambda_1, \mathbf{x}_1)$ be an eigenpair of A. We assume that $\mathbf{x}_1$ is normalized: $\|\mathbf{x}_1\|_2 = 1$. Then there is a matrix $X_\perp \in \mathbb{C}^{n\times(n-1)}$ such that $U_1 := \big(\mathbf{x}_1, X_\perp\big) \in \mathbb{C}^{n\times n}$ is unitary.[2] Then

$$\begin{aligned} U_1^* A U_1 &= \begin{pmatrix} \mathbf{x}_1^* A\mathbf{x}_1 & \mathbf{x}_1^* A X_\perp \\ X_\perp^* A\mathbf{x}_1 & X_\perp^* A X_\perp \end{pmatrix} = \begin{pmatrix} \lambda_1 \mathbf{x}_1^*\mathbf{x}_1 & \mathbf{x}_1^* A X_\perp \\ \lambda_1 X_\perp^*\mathbf{x}_1 & X_\perp^* A X_\perp \end{pmatrix} \\ &= \begin{pmatrix} \lambda_1 & \mathbf{x}_1^* A X_\perp \\ \mathbf{0} & X_\perp^* A X_\perp \end{pmatrix} =: \begin{pmatrix} t_{11} & \mathbf{x}_1^* A X_\perp \\ \mathbf{0} & X_\perp^* A X_\perp \end{pmatrix}. \end{aligned}$$

By applying the induction hypothesis to $A_2 := X_\perp^* A X_\perp \in \mathbb{C}^{(n-1)\times(n-1)}$, there is a unitary matrix U_2 such that $T_2 := U_2^* A_2 U_2$ is upper triangular. After setting $U := U_1 \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & U_2 \end{pmatrix}$, this completes the proof of the theorem as

$$\begin{aligned} U^* A U &= \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & U_2^* \end{pmatrix} U_1^* A U_1 \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & U_2 \end{pmatrix} \\ &= \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & U_2^* \end{pmatrix} \begin{pmatrix} t_{11} & \mathbf{x}_1^* A X_\perp \\ \mathbf{0} & X_\perp^* A X_\perp \end{pmatrix} \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & U_2 \end{pmatrix} = \begin{pmatrix} t_{11} & \star \\ \mathbf{0} & T_2 \end{pmatrix} \end{aligned}$$

is upper triangular.
Note: This proof is *not* constructive, as it assumes the availability of eigenvalues and eigenvectors right from the start. □

The Schur decomposition (1.32) is of fundamental theoretical and practical importance. Many eigenvalue solvers, including the QR algorithm, actually aim at computing the Schur decomposition or at least parts of it. In MATLAB, the Schur decomposition of a complex matrix A is computed by typing `[U,T] = schur(A)`.

Note that the order of the eigenvalues on the diagonal of T is arbitrary. In fact, the proof of Theorem 1.11 reveals that any order of the eigenvalues is possible. Unfortunately, this order cannot be directly influenced when using `schur`. However, the MATLAB command `ordschur` allows to achieve any desirable order in a simple post-processing step.

While the first column of the matrix U in a Schur decomposition $U^*AU = T$ is clearly an eigenvector belonging to the eigenvalue $\lambda_1 = t_{11}$, there is no analogous statement for the other columns of U. Let $k \le n$ and let $\mathbf{u}_1, \ldots, \mathbf{u}_k$ denote the first k columns of U. Then the first k columns of the relation $AU = UT$ read as

$$A\big(\mathbf{u}_1, \ldots, \mathbf{u}_k\big) = \big(\mathbf{u}_1, \ldots, \mathbf{u}_k\big) \begin{pmatrix} t_{11} & \cdots & t_{1k} \\ & \ddots & \vdots \\ & & t_{kk} \end{pmatrix}. \tag{1.33}$$

The vectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ are sometimes called **Schur vectors** and (1.33) a partial Schur decomposition. The relation (1.33) implies that each $A\mathbf{u}_j$ is a linear combination of $\mathbf{u}_1, \ldots, \mathbf{u}_k$. This makes $\mathcal{U}_k := \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ an invariant subspace for every $k = 1, \ldots, n$.

[2] One way to construct such a matrix is to complement x_1 to a basis $\{x_1, y_2, \ldots, y_n\}$ of $\mathbb{C}^n$ and apply Gram-Schmidt orthonormalization. In MATLAB, one can simply type `[U,R] = qr(x1)`.

Definition 1.12 *A subspace $\mathcal{X} \subset \mathbb{C}^n$ is called* **invariant** *under a matrix $A \in \mathbb{C}^{n\times n}$ if $A\mathcal{X} := \{A\mathbf{x} \colon \mathbf{x} \in \mathcal{X}\}$ satisfies $A\mathcal{X} \subseteq \mathcal{X}$.*

Let the columns of $X \in \mathbb{C}^{n\times k}$ form a basis for an invariant subspace $\mathcal{X}$. Then, by Definition 1.12, there is a matrix $B \in \mathbb{C}^{k\times k}$ such that

$$AX = XB. \tag{1.34}$$

For diagonalizable matrices, invariant subspaces are always spanned by eigenvectors. In fact, suppose that the matrix B in (1.34) can be diagonalized with a similarity transformation: $D = S^{-1}BS = \operatorname{diag}(\lambda_1, \ldots, \lambda_k)$ for an invertible matrix $S \in \mathbb{C}^{k\times k}$. Then

$$A(XS) = (XS)S^{-1}BS = (XS) \cdot \operatorname{diag}(\lambda_1, \ldots, \lambda_k)$$

shows that the columns of the transformed basis XS are eigenvectors belonging to the eigenvalues $\lambda_1, \ldots, \lambda_k$ of A.

Finally, if A is a real matrix then Theorem 1.11 does not yield real matrices U and T. This would be impossible in general. For example, the real 2×2 matrix

$$\begin{pmatrix} \alpha & \beta \\ -\gamma & \alpha \end{pmatrix}, \quad \alpha, \beta, \gamma \in \mathbb{R}, \quad \beta > 0, \quad \gamma > 0, \tag{1.35}$$

has complex conjugate pair of eigenvalues $\alpha \pm \mathrm{i}\sqrt{\beta\gamma}$. If there was a real orthogonal matrix U transforming A to upper triangular form, then $U^{\mathsf{T}}AU$ would be a real matrix with the (complex) eigenvalues on the diagonal. This is clearly a contradiction. However, for computational efficiency it is desirable to be able to compute Schur-like form in real arithmetic. For this purpose, the notion of triangularity needs to be slightly relaxed.

Theorem 1.13 (Real Schur decomposition) *For every $A \in \mathbb{R}^{n\times n}$ there is an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ such that*

$$Q^{\mathsf{T}}AQ = T = \begin{pmatrix} T_{11} & T_{12} & \cdots & T_{1n_b} \\ 0 & T_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & T_{n_{b-1},n_b} \\ 0 & \cdots & 0 & T_{n_b n_b} \end{pmatrix} \in \mathbb{R}^{n\times n}, \tag{1.36}$$

is **upper quasi-triangular**. *This means that the n_b diagonal blocks of T are either 1×1 (scalar) or 2×2. The scalar blocks correspond to the real eigenvalues of A. The 2×2 blocks take the form* (1.35) *and correspond to complex conjugate pairs of eigenvalues of A.*

Proof. The proof is similar to the proof of Theorem 1.11 and omitted. See [GolV96] for details. □

For a real matrix A, the MATLAB command `[V,D] = schur(A)` actually computes the real Schur decomposition (1.36). To compute the complex Schur decomposition also for a real matrix, one needs to type `[V,D] = schur(A,'complex')`.

1.5 Hermitian matrices

A complex matrix $A \in \mathbb{C}^{n\times n}$ is called **Hermitian** if

$$A = A^*,$$

which – in particular – implies that the diagonal elements of A are real. A real matrix $A \in \mathbb{R}^{n\times n}$ is called **symmetric** if $A = A^{\mathsf{T}}$. The eigenvalues and eigenvectors of Hermitian and symmetric matrices have rather distinctive properties.

Let us consider the Schur decomposition $A = UTU^*$ of a Hermitian matrix A. Then

$$T = U^*AU = U^*A^*U = (U^*AU)^* = T^*,$$

which implies that T is Hermitian. At the same time, T is upper triangular and hence T must be diagonal. Moreover, the diagonal entries of T (which correspond to the eigenvalues of A) are real. To summarize, we obtain the following important result.

Theorem 1.14 (Spectral decomposition) *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian. Then there exist a unitary matrix $U \in \mathbb{C}^{n\times n}$ and a diagonal matrix $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^{n\times n}$ such that*

$$A = U\Lambda U^*. \tag{1.37}$$

For a symmetric matrix $A \in \mathbb{R}^{n\times n}$, the real Schur decomposition implies that there exist an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ and a diagonal matrix $\Lambda \in \mathbb{R}^{n\times n}$ such that $A = Q\Lambda Q^{\mathsf{T}}$.

Theorem 1.14 shows not only that a Hermitian matrix is diagonalizable but also that the corresponding similarity transformation can be performed with a unitary matrix U. As $\kappa(U) = 1$, it is numerically safe to diagonalize a Hermitian matrix, in contrast to a general matrix, see the discussion after Theorem 1.8.

The relation (1.37) implies that the columns $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ of U are eigenvectors belonging to the eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ of A. We can rewrite (1.37) as

$$A = \lambda_1\mathbf{u}_1\mathbf{u}_1^* + \lambda_2\mathbf{u}_2\mathbf{u}_2^* + \cdots + \lambda_n\mathbf{u}_n\mathbf{u}_n^*.$$

This is the more common formulation of the **spectral decomposition**.

The eigenvectors $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n$ are orthogonal to each other in the standard inner product, simply because the columns of a unitary matrix are orthogonal to each other. However, one has to be a bit careful concluding now that *any* eigenvectors of a Hermitian matrix are orthogonal to each other. For example, if $\lambda_i = \lambda_j$ for $i \neq j$ then any linear combination $\mathbf{v} = \alpha\mathbf{u}_i + \beta\mathbf{u}_j \neq 0$ is also an eigenvector belonging to λ_j. Note that if $\alpha \neq 0 \neq \beta$, $\mathbf{v}$ can never be orthogonal to $\mathbf{u}_i$ and $\mathbf{u}_j$. On the other hand, the eigenvectors belonging to *different* eigenvalues are always orthogonal to each other. Let $A\mathbf{u} = \lambda\mathbf{u}$ and $A\mathbf{v} = \tilde{\lambda}\mathbf{v}$ with $\lambda \neq \tilde{\lambda}$. Then

$$\begin{aligned} & \mathbf{v}^*A\mathbf{u} = \mathbf{v}^*(A\mathbf{u}) = \lambda\mathbf{v}^*\mathbf{u}, \quad \mathbf{v}^*A\mathbf{u} = (A\mathbf{v})^*\mathbf{u} = \tilde{\lambda}\mathbf{v}^*\mathbf{u} \\ \Longrightarrow \qquad & \lambda\mathbf{v}^*\mathbf{u} = \tilde{\lambda}\mathbf{v}^*\mathbf{u} \quad \overset{\lambda\neq\tilde{\lambda}}{\Longrightarrow} \quad \mathbf{v}^*\mathbf{u} = 0, \end{aligned}$$

showing that $\mathbf{u} \perp \mathbf{v}$.

In applications and numerical methods for solving Hermitian eigenvalue problems, the properties of the so called Rayleigh quotient are important.

Definition 1.15 *Let $A \in \mathbb{C}^{n\times n}$ and $\mathbf{x} \in \mathbb{C}^n$ with $\mathbf{x} \neq 0$. Then*

$$\rho_A(\mathbf{x}) = \frac{\mathbf{x}^*A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}$$

is the **Rayleigh quotient** *of A at $\mathbf{x}$.*

Note that $\rho_A(\mathbf{x}) = \rho_A(\alpha\mathbf{x})$ for any nonzero scalar α. Hence it is sufficient to consider vectors $\mathbf{x}$ of unit norm.

In the following we will relate the Rayleigh quotient to the minimal and maximal eigenvalues of A. For this purpose, we will assume that the eigenvalues in the spectral decomposition of A are ordered:

$$A = \sum_{j=1}^{n} \lambda_j \mathbf{u}_j \mathbf{u}_j^*, \qquad \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n, \tag{1.38}$$

which can of course always be achieved by simply renumbering the eigenvalues and eigenvectors.

Theorem 1.16 *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian with eigenvalues $\lambda_1 \leq \cdots \leq \lambda_n$. Then*

$$\lambda_1 = \min_{\mathbf{x}\neq 0} \rho_A(\mathbf{x}), \qquad \lambda_n = \max_{\mathbf{x}\neq 0} \rho_A(\mathbf{x}).$$

Proof. From the spectral decomposition (1.38), we have

$$\mathbf{x}^* A\mathbf{x} = \sum_{j=1}^{n} \lambda_j \mathbf{x}^* \mathbf{u}_j \mathbf{u}_j^* \mathbf{x} = \sum_{j=1}^{n} \lambda_j |\mathbf{u}_j^* \mathbf{x}|^2 \geq \lambda_1 \sum_{j=1}^{n} |\mathbf{u}_j^* \mathbf{x}|^2.$$

On the other hand, using $I = \sum_{j=1}^{n} u_j u_j^*$ shows

$$\mathbf{x}^*\mathbf{x} = \sum_{j=1}^{n} \mathbf{x}^* \mathbf{u}_j \mathbf{u}_j^* \mathbf{x} = \sum_{j=1}^{n} |\mathbf{u}_j^* \mathbf{x}|^2.$$

Therefore, $\rho_A(x) = \mathbf{x}^* A\mathbf{x} \,/\, \mathbf{x}^*\mathbf{x} \geq \lambda_1$. As $\rho_A(\mathbf{u}_1) = \lambda_1$, this proves the first part of the theorem. The second part follows from the first part when replacing A by $-A$. ∎

For a real symmetric matrix A, it is sufficient to minimize (or maximize) over real vectors in Theorem 1.16.

Theorem 1.16 reflects a deep relation between Hermitian/symmetric eigenvalue problems and the calculus of variations. We will not discuss this interesting topic further and refer to `http://en.wikipedia.org/wiki/Calculus_of_variations` for more information.

A variant of Theorem 1.16 is obtained when constraining $\mathbf{x}$ to be orthogonal to the first $p-1$ eigenvectors.

Lemma 1.17 *Consider the spectral decomposition* (1.38) *of a Hermitian matrix A and set $\mathcal{U}_{p-1} := \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_{p-1}\}$ for $2 \leq p \leq n$. Then*

$$\lambda_p = \min_{\substack{\mathbf{x}\in\mathcal{U}_{p-1}^{\perp} \\ \mathbf{x}\neq 0}} \rho_A(\mathbf{x}).$$

Proof. The proof is along the lines of the proof of Theorem 1.16, using that $|\mathbf{u}_1^*\mathbf{x}| = \cdots = |\mathbf{u}_{p-1}^*\mathbf{x}| = 0$. ∎

A disadvantage of Lemma 1.17 is that one needs exact knowledge of the first $p-1$ eigenvectors before gaining access to the pth eigenvalue. The next result extends Theorem 1.16 more directly to interior eigenvalues.

Theorem 1.18 (Courant-Fischer minimax theorem) *Let $A \in \mathbb{C}^{n\times n}$ be Hermitian with eigenvalues $\lambda_1 \le \cdots \le \lambda_n$. Then*

$$\lambda_p = \min_{\substack{\mathcal{X} \text{ subspace of } \mathbb{C}^n \\ \dim(\mathcal{X})=p}} \max_{\substack{\mathbf{x}\in\mathcal{X} \\ \mathbf{x}\neq 0}} \rho_A(\mathbf{x}) = \min_{\substack{X\in\mathbb{C}^{n\times p} \\ \operatorname{rank}(X)=p}} \max_{\mathbf{y}\neq\mathbf{0}} \rho_A(X\mathbf{y}) \tag{1.39}$$

holds for every $p = 1, \ldots, n$.

Proof. As the second equality is obvious, it is sufficient to prove the first equality of (1.39). Consider the spectral decomposition (1.38) and set

$$\mathcal{U}_{p-1} := \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_{p-1}\}.$$

By dimension counting, for every p-dimensional subspace $\mathcal{X}$ there is $\mathbf{x} \in \mathcal{X} \cap \mathcal{U}_{p-1}^{\perp}$ with $\mathbf{x}^*\mathbf{x} = 1$. From Lemma 1.17, it follows that $\rho_A(x) \ge \lambda_p$. Equality is obtained by setting $\mathcal{X} = \mathcal{U}_p$ and $\mathbf{x} = \mathbf{u}_p$. □

For computational purposes, the most important consequence of the minimax theorem is the following result.

Corollary 1.19 (Monotonicity principle) *Let $V \in \mathbb{C}^{n\times p}$ satisfy $V^*V = I_p$, that is, the columns of V form an orthonormal basis. Then the eigenvalues $\lambda_1 \le \cdots \le \lambda_n$ of a Hermitian matrix A and the eigenvalues $\mu_1 \le \cdots \le \mu_p$ of $M := V^*AV \in \mathbb{C}^{p\times p}$ satisfy*

$$\lambda_k \le \mu_k, \qquad 1 \le k \le p.$$

Proof. Let $\mathcal{V} := \operatorname{span}(V)$. By Theorem 1.18, we have

$$\begin{aligned}
\lambda_k &= \min_{\substack{\mathcal{X} \text{ subspace of } \mathbb{C}^n \\ \dim(\mathcal{X})=k}} \max_{\substack{\mathbf{x}\in\mathcal{X} \\ \mathbf{x}\neq 0}} \rho_A(\mathbf{x}) \\
&\le \min_{\substack{\mathcal{X} \text{ subspace of } \mathcal{V} \\ \dim(\mathcal{X})=k}} \max_{\substack{\mathbf{x}\in\mathcal{X} \\ \mathbf{x}\neq 0}} \rho_A(\mathbf{x}) \\
&= \min_{\substack{X\in\mathbb{C}^{p\times k} \\ \operatorname{rank}(X)=k}} \max_{\substack{\mathbf{y}\in\mathbb{C}^k \\ \mathbf{y}\neq 0}} \rho_A(VX\mathbf{y}) = \mu_k,
\end{aligned}$$

where the last equality follows from

$$\rho_A(VX\mathbf{y}) = \frac{(VX\mathbf{y})^*A(VX\mathbf{y})}{(VX\mathbf{y})^*(VX\mathbf{y})} = \frac{(X\mathbf{y})^*M(X\mathbf{y})}{(X\mathbf{y})^*(X\mathbf{y})} = \rho_M(X\mathbf{y}).$$

□

Recall that the *trace* of a matrix $A \in \mathbb{C}^{n\times n}$ is defined to be the sum of the diagonal elements of a matrix. Matrices that are similar have equal trace. By the Schur decomposition,

$$\operatorname{trace}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i \tag{1.40}$$

holds for a general matrix A. Another way to see (1.40) is to consider the coefficient of λ^{n-1} in the characteristic polynomial, see Section 1.1.

Theorem 1.20 (Ky-Fan theorem) *Let A be Hermitian with eigenvalues $\lambda_1 \le \cdots \le \lambda_n$. Then*

$$\lambda_1 + \lambda_2 + \cdots + \lambda_p = \min_{\substack{V\in\mathbb{C}^{n\times p} \\ V^*V=I_p}} \operatorname{trace}(V^*AV).$$

Proof. Let $\mu_1 \leq \ldots \leq \mu_p$ be the eigenvalues of V^*AV. By the monotonicity principle,

$$\lambda_1 + \cdots + \lambda_p \leq \mu_1 + \cdots + \mu_p.$$

Equality is obtained when considering the spectral decomposition (1.38) and setting $V = \big(\mathbf{u}_1, \ldots, \mathbf{u}_p\big)$. □

1.6 Application 3: Spectral clustering

The goal of *clustering* is to group a given set of data points $x_1, \ldots, x_n$ into k clusters such that members from the same cluster are (in some sense) close to each other and members from different clusters are (in some sense) well separated from each other.

A popular approach to clustering is based on *similarity graphs*. For this purpose, we need to assume some notion of similarity $s(x_i, x_j) \geq 0$ between pairs of data points x_i and x_j. An undirected graph $G = (V, E)$ is constructed such that its vertices correspond to the data points: $V = \{x_1, \ldots, x_n\}$. Two vertices x_i, x_j are connected by an edge if the similarity s_{ij} between x_i and x_j is sufficiently large. Moreover, a weight $w_{ij} > 0$ is assigned to the edge, depending on s_{ij}. If two vertices are not connected we set $w_{ij} = 0$. The weights are collected into a *weighted adjacency matrix*

$$W = \big(w_{ij}\big)_{i,j=1}^{n}.$$

There are several possibilities to define the weights of the similarity graph associated with a set of data points and a similarity function:

fully connected graph All points with positive similarity are connected with each other and we simply set $w_{ij} = s(x_i, x_j)$. Usually, this will only result in reasonable clusters if the similarity function models locality very well. One example of such a similarity function is the Gaussian $s(x_i, x_j) = \exp\big(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\big)$, where $\|x_i - x_j\|$ is some distance measure (e.g., Euclidean distance) and σ is some parameter controlling how strongly locality is enforced.

k-nearest neighbors Two vertices x_i, x_j are connected if x_i is among the k-nearest neighbors of x_j or if x_j is among the k-nearest neighbors of x_i (in the sense of some distance measure). The weight of the edge between connected vertices x_i, x_j is set to the similarity function $s(x_i, x_j)$.

ϵ-neighbors Two vertices x_i, x_j are connected if their pairwise distance is smaller than ϵ for some parameter $\epsilon > 0$. In this case, the weights are usually chosen uniformly, e.g., $w_{ij} = 1$ if x_i, x_j are connected and $w_{ij} = 0$ otherwise.

Assuming that the similarity function is symmetric ($s(x_i, x_j) = s(x_j, x_i)$ for all x_i, x_j) all definitions above give rise to a symmetric weight matrix W. In practice, the choice of the most appropriate definition depends – as usual – on the application.

1.6.1 The graph Laplacian

In the following we construct the so called *graph Laplacian*, whose spectral decomposition will later be used to determine clusters. For simplicity, we assume the weight matrix W to be symmetric. The degree of a vertex x_i is defined as

$$d_i = \sum_{j=1}^{n} w_{ij}. \tag{1.41}$$

In the case of an unweighted graph, the degree d_i amounts to the number of vertices adjacent to v_i (counting also v_i if $w_{ii} = 1$). The degree matrix is defined as

$$D = \operatorname{diag}(d_1, d_2, \ldots, d_n).$$

The *graph Laplacian* is then defined as

$$L = D - W.$$

By (1.41), the row sums of L are zero. In other words, $L\mathbf{e} = 0$ with $\mathbf{e}$ the vector of all ones. This implies that 0 is an eigenvalue of L with the associated eigenvector $\mathbf{e}$. Since L is symmetric all its eigenvalues are real and one can show that 0 is the smallest eigenvalue; hence L is positive semidefinite. It may easily happen that more than one eigenvalue is zero. For example, if the set of vertices can be divided into two subsets $\{x_1, \ldots, x_k\}$, $\{x_{k+1}, \ldots, x_n\}$, and vertices from one subset are not connected with vertices from the other subset, then

$$L = \begin{pmatrix} L_1 & 0 \\ 0 & L_2 \end{pmatrix},$$

where L_1, L_2 are the Laplacians of the two disconnected components. Thus L has two eigenvectors $\binom{\mathbf{e}}{\mathbf{0}}$ and $\binom{\mathbf{0}}{\mathbf{e}}$ with eigenvalue 0. Of course, any linear combination of these two linearly independent eigenvectors is also an eigenvector of L. In particular, when keeping the second eigenvector $\mathbf{u}_2$ orthogonal to the known eigenvector of all ones, then $\mathbf{u}_2 = \pm\binom{\mathbf{e}}{-\mathbf{e}}$ and the two subsets of vertices can be recovered from the sign pattern of $\mathbf{u}_2$. This is an intuitive but also quite heuristic explanation for the use of the so called **Fiedler vector** $\mathbf{u}_2$ in clustering. A more sophisticated explanation follows below.

1.6.2 Graph cuts

For two disjoint subsets of vertices $A, B \subset V$ denote by

$$W(A, B) := \sum_{i \in A, j \in B} w_{ij}$$

the total weight of the edges connecting A and B. If $V_1 \subset V$ and $\overline{V}_1 = V \setminus V_1$ denotes its complement in V so that V_1 and $\overline{V}_1$ partition the graph into two pieces, then $W(V_1, \overline{V}_1)$ sums up the weights of the edges that need to be cut. For an unweighted graph, this is simply the number of edges cut. This notion can be easily extended to a partitioning of V into k pairwise disjoint subsets $V_1, \ldots, V_k$:

$$\mathsf{cut}(V_1, \ldots, V_k) := \frac{1}{2} \sum_{i=1}^{k} W(V_i, \overline{V}_i). \tag{1.42}$$

(The factor one half only serves for normalization that edges are not counted twice.) A first approach to a good clustering would be to choose the partitioning such that cut is minimal. However, this usually does not lead to a satisfactory clustering; typically most of the subsets consist of a single vertex. To attain a more balanced partitioning, we introduce the cardinalities $|V_i|$ as a weighting factor:

$$\mathsf{ratiocut}(V_1, \ldots, V_k) := \sum_{i=1}^{k} \frac{W(V_i, \overline{V}_i)}{|V_i|} = \sum_{i=1}^{k} \frac{\mathsf{cut}(V_i, \overline{V}_i)}{|V_i|}.$$

Unfortunately, the introduction of these weights makes the minimization of $\mathsf{ratiocut}$ an NP-hard problem. We will need to relax to make this problem tractable.

1.6.3 Relaxation of ratiocut for $k = 2$

Let us first consider the case $k = 2$: partitioning V into a subset V_1 and its complement $\overline{V_1}$. This yields the optimization problem

$$\min_{V_1 \subset V} \mathsf{ratiocut}(V_1, \overline{V}_1). \tag{1.43}$$

To rewrite the objective as a Rayleigh quotient, we associate the following vector $\mathbf{u} = \big(u_1, \ldots, u_n\big)^\mathsf{T}$ with V_1:

$$u_i = \begin{cases} \sqrt{|\overline{V}_1|/|V_1|} & \text{if } v_i \in V_1, \\ -\sqrt{|V_1|/|\overline{V}_1|} & \text{otherwise.} \end{cases} \tag{1.44}$$

Then

$$\begin{aligned} \mathbf{u}^\mathsf{T} L \mathbf{u} &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(u_i - u_j)^2 \\ &= \frac{1}{2} \sum_{i \in V_1, j \in \overline{V}_1} w_{ij} \Big(\sqrt{|\overline{V}_1|/|V_1|} + \sqrt{|V_1|/|\overline{V}_1|} \Big)^2 \\ &\quad + \frac{1}{2} \sum_{i \in \overline{V}_1, j \in V_1} w_{ij} \Big(\sqrt{|\overline{V}_1|/|V_1|} + \sqrt{|V_1|/|\overline{V}_1|} \Big)^2 \\ &= W(V_1, \overline{V}_1) \Big(|\overline{V}_1|/|V_1| + |V_1|/|\overline{V}_1| + 2 \Big) \\ &= |V| \cdot \mathsf{ratiocut}(V_1, \overline{V}_1). \end{aligned}$$

Moreover,

$$\mathbf{e}^\mathsf{T} \mathbf{u} = \sum_{i=1}^{n} u_i = \sum_{i \in V_1} \sqrt{|\overline{V}_1|/|V_1|} - \sum_{i \in \overline{V}_1} \sqrt{|V_1|/|\overline{V}_1|} = 0$$

and

$$\|\mathbf{u}\|_2^2 = \sum_{i=1}^{n} u_i^2 = |\overline{V}_1| + |V_1| = n.$$

To summarize, the minimization problem (1.43) is equivalent to

$$\min_{\substack{V_1 \subset V \\ \mathbf{u} \text{ as in } (1.44)}} \mathbf{u}^\mathsf{T} L \mathbf{u} \quad \text{subject to } \mathbf{u} \perp \mathbf{e},\ \|\mathbf{u}\|_2 = \sqrt{n}.$$

This is still an NP-hard discrete optimization problem, because of the discrete nature of the definition (1.44) of $\mathbf{u}$. We discard discreteness and obtain the **relaxed optimization problem**

$$\min_{\mathbf{u} \in \mathbb{R}^n} \mathbf{u}^\mathsf{T} L \mathbf{u} \quad \text{subject to } \mathbf{u} \perp \mathbf{e},\ \|\mathbf{u}\|_2 = \sqrt{n}. \tag{1.45}$$

Since the smallest eigenvalue of L is $\lambda_1 = 0$ with eigenvector $\mathbf{u}_1 = \mathbf{e}$, Lemma 1.17 can be applied to show that a solution of (1.45) is given by an eigenvector $\mathbf{u}_2$ belonging to the second smallest eigenvalue λ_2 of L. The minimum is given by $n\lambda_2$. The simplest way to obtain a clustering from the Fiedler vector $\mathbf{u} := \mathbf{u}_2$ is to choose

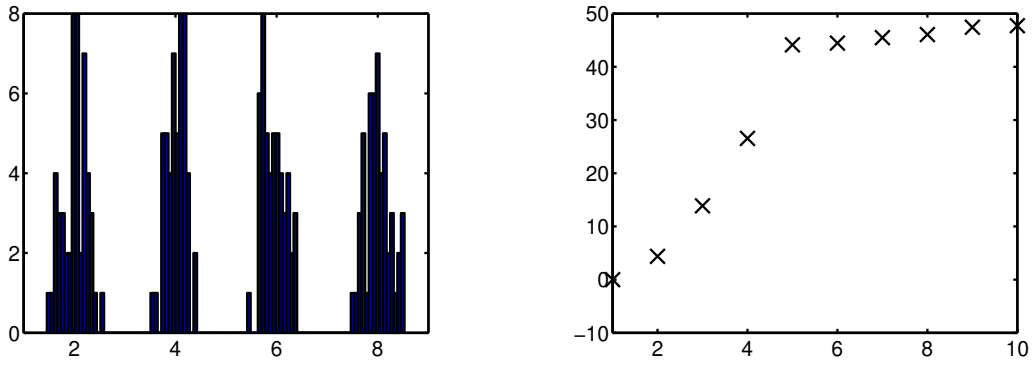
$$\begin{cases} v_i \in V_1 & \text{if } u_i \geq 0, \\ v_i \in \overline{V}_1 & \text{otherwise.} \end{cases}$$

A more sophisticated and robust way is to apply a simple clustering algorithm (such as k-means) to the entries of the Fiedler vector.

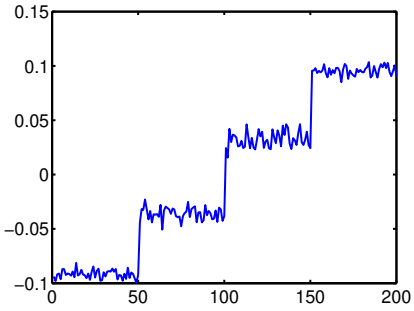
Example 1.21 200 *real numbers are generated by superimposing samples from* 4 *Gaussian distributions with* 4 *different means:*

```
m = 50; randn('state',0);
x = [2+randn(m,1)/4;4+randn(m,1)/4;6+randn(m,1)/4;8+randn(m,1)/4];
```

The following figure show the histogram of the distribution of the entries of x *and the* 10 *smallest eigenvalues of the graph Laplacian for the fully connected similarity graph with similarity function* $s(x_i, x_j) = \exp\big(-\frac{|x_i - x_j|^2}{2}\big)$*:*



As expected, one eigenvalue is exactly zero. The following figure shows the entries of the Fiedler vector, the eigenvector belonging to the second smallest eigenvalue of L*:*



1.6.4 Relaxation of ratiocut for $k > 2$

For a partitioning of V into $k > 2$ subsets $V_1, \ldots, V_k$, we define a matrix $Q \in \mathbb{R}^{n \times k}$ of indicator vectors as

$$q_{ij} = \begin{cases} 1/\sqrt{|V_j|} & \text{if } v_i \in V_j, \\ 0 & \text{otherwise.} \end{cases} \tag{1.46}$$

Similar to the calculations in the previous section, it can be shown that

$$\mathsf{ratiocut}(V_1, \ldots, V_k) = \operatorname{trace}(Q^\mathsf{T} L Q).$$

Hence, the problem of minimizing ratiocut can be equivalently written as

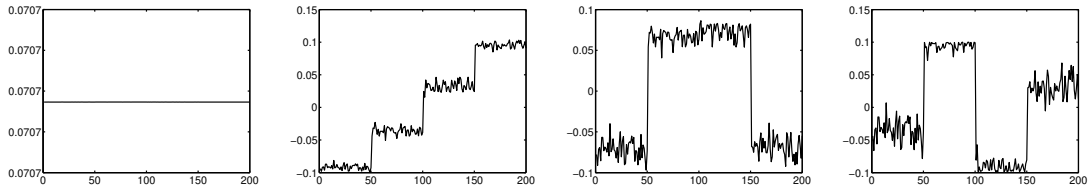
$$\min_{V_1,\ldots,V_k} \operatorname{trace}(Q^\mathsf{T} L Q) \quad \text{subject to } Q \text{ as in (1.46)}, \; Q^\mathsf{T} Q = I.$$

The relaxed problem becomes

$$\min_{Q\in\mathbb{R}^{n\times k}, Q^\mathsf{T}Q=I} \operatorname{trace}(Q^\mathsf{T}LQ).$$

By Theorem 1.20, a solution of this problem is given by $Q = \big(\mathbf{u}_1, \dots, \mathbf{u}_k\big)$, where $\mathbf{u}_1, \dots, \mathbf{u}_k$ are eigenvectors belonging to the k smallest eigenvalues of L.

Example 1.21 ctd. *The following four figures show the entries of the* 4 *eigenvectors belonging to the* 4 *smallest eigenvalues of* L*:*



On the one hand, it is clearly visible that the eigenvectors are well approximated by linear combinations of indicator vectors. On the other hand, none of the eigenvectors is close to an indicator vector itself and hence no immediate conclusion on the clusters is possible.

To solve the issue observed in the preceding example, we "transpose" the basis and consider the row vectors of Q:

$$Q^\mathsf{T} = \big(\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_n\big), \quad q_i \in \mathbb{R}^k.$$

If Q contained indicator vectors then each of the small vectors $\mathbf{q}_i$ would be a unit vector $\mathbf{e}_j$ for some $1 \le j \le k$ (possibly divided by $|V_j|$). In particular, the $\mathbf{q}_i$ would separate very well into k different clusters. Hence, applying a clustering algorithm to $\mathbf{q}_1, \dots, \mathbf{q}_n$ allows us to detect the membership of $\mathbf{q}_i$. The key point is that the small vectors $\mathbf{q}_1, \dots, \mathbf{q}_n$ are much better separated than the original data $x_1, \dots, x_n$. Hence, a much simpler algorithm can be used for clustering. One of the most basic algorithms is *k-means clustering*. Initially, this algorithm assigns each $\mathbf{q}_i$ randomly[3] to a cluster ℓ with $1 \le \ell \le k$ and then iteratively proceeds as follows:

1. Compute cluster centers $\mathbf{c}_\ell$ as cluster means:

$$\mathbf{c}_\ell = \sum_{i \text{ in cluster } \ell} \mathbf{u}_i \Big/ \sum_{i \text{ in cluster } \ell} 1.$$

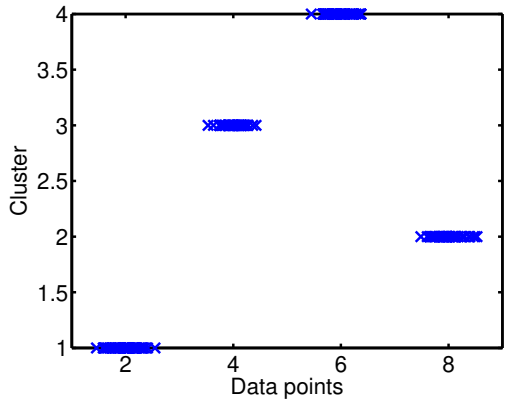
2. Assign each $\mathbf{u}_i$ to the cluster with the nearest cluster center.

3. Goto Step 1.

The algorithm is stopped when the assigned clusters do not change in an iteration.

Example 1.21 ctd. *The k-means algorithm applied to the matrix* Q *converges after* 2 *iterations and results in the following clustering:*

[3]For unlucky choices of random assignments the k-means algorithm may end up with less than k clusters. A simple albeit dissatisfying solution is to restart k-means with a different random assignment.



1.7 The Singular Value Decomposition (SVD)

The singular value decomposition is among the most important decompositions in (numerical) linear algebra.

Theorem 1.22 (SVD) *Let $A \in \mathbb{C}^{n\times p}$. Then there are unitary matrices $U \in \mathbb{C}^{n\times n}$ and $V \in \mathbb{C}^{p\times p}$ such that*

$$A = U\Sigma V^*, \quad \textit{with} \quad \Sigma = \begin{pmatrix} \sigma_1 & & & \\ & \ddots & & 0 \\ & & \sigma_r & \\ & 0 & & 0 \end{pmatrix} \tag{1.47}$$

and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r \geq 0$ for $r = \min\{n, p\}$.

Proof. Without loss of generality, we may assume $p \leq n$; otherwise we can simply transpose A and prove the result for A^*. The proof is by induction over p, similar to the proof of the Schur decomposition in Theorem 1.11. For $p = 1$, A is a column vector and the result of the theorem is trivially obtained by setting $U = A/\|A\|_2$, $\Sigma = \|A\|_2$, $V = 1$.

For general p, let $\mathbf{v}_1$ be such that $\|\mathbf{v}_1\|_2 = 1$ and be such that

$$\|A\mathbf{v}_1\|_2 = \max_{\|\mathbf{v}\|_2=1} \|A\mathbf{v}\|_2 =: \|A\|_2, \tag{1.48}$$

where $\|A\|_2$ denotes the **matrix 2-norm** (sometimes also called spectral norm). Set $\sigma_1 := \|A\mathbf{v}_1\|_2 = \|A\|_2$ and $\mathbf{u}_1 := A\mathbf{v}_1/\sigma_1$. (If $\sigma_1 = 0$, the vector $\mathbf{u}_1$ can be chosen as an arbitrary vector satisfying $\|\mathbf{u}_1\|_2 = 1$.) Then, by definition,

$$A\mathbf{v}_1 = \sigma_1\mathbf{u}_1.$$

If we choose unitary matrices $U_1 = \big(\mathbf{u}_1, U_\perp\big) \in \mathbb{C}^{n\times n}$ and $V_1 = \big(\mathbf{v}_1, V_\perp\big) \in \mathbb{C}^{p\times p}$, this implies

$$U_1^* A V_1 = \left(\begin{array}{c|c} \mathbf{u}_1^* A\mathbf{v}_1 & \mathbf{u}_1^* A V_\perp \\ \hline U_\perp^* A\mathbf{v}_1 & U_\perp^* A V_\perp \end{array}\right) = \left(\begin{array}{c|c} \sigma_1 & \mathbf{w}^* \\ \hline \mathbf{0} & A_1 \end{array}\right), \tag{1.49}$$

with $\mathbf{w} := V_\perp^* A^* \mathbf{u}_1$ and $A_1 = U_\perp^* A V_\perp$. To show $\mathbf{w} = 0$, we need one basic fact about $\|\cdot\|_2$: the matrix 2-norm does not change under multiplication with unitary matrices. Combined

with (1.49), this yields

$$\sigma_1 = \|A\|_2 = \|U_1^* A V_1\|_2 = \left\| \left(\begin{array}{c|c} \sigma_1 & \mathbf{w}^* \\ \hline \mathbf{0} & A_1 \end{array} \right) \right\|_2 .$$

On the other hand, setting $\mathbf{v} = \frac{1}{\sqrt{\sigma_1^2 + \|\mathbf{w}\|_2^2}} \binom{\sigma_1}{\mathbf{w}}$ (which has norm 1) gives

$$\begin{aligned} \left\| \left(\begin{array}{c|c} \sigma_1 & \mathbf{w}^* \\ \hline \mathbf{0} & A_1 \end{array} \right) \right\|_2 &\geq \left\| \left(\begin{array}{c|c} \sigma_1 & \mathbf{w}^* \\ \hline \mathbf{0} & A_1 \end{array} \right) \mathbf{v} \right\|_2 = \frac{1}{\sqrt{\sigma_1^2 + \|\mathbf{w}\|_2^2}} \left\| \begin{pmatrix} \sigma_1^2 + \|\mathbf{w}\|_2^2 \\ A_1 \mathbf{w} \end{pmatrix} \right\|_2 \\ &\geq \sqrt{\sigma_1^2 + \|\mathbf{w}\|_2^2}, \end{aligned}$$

where we have used the maximization property (1.48). Hence, $\sigma_1 \geq \sqrt{\sigma_1^2 + \|\mathbf{w}\|_2^2}$, which is only possible if $\mathbf{w} = 0$. In summary we have

$$U_1^* A V_1 = \left(\begin{array}{c|c} \sigma_1 & \mathbf{0} \\ \hline \mathbf{0} & U_\perp^* A V_\perp \end{array} \right),$$

with $\sigma_2 := \|U_\perp^* A V_\perp\|_2 \leq \sigma_1$. The rest of the induction argument is along the lines of the proof of Theorem 1.11. ∎

Given an SVD $A = U\Sigma V^*$, the nonnegative scalars $\sigma_1, \ldots, \sigma_r$ are called the **singular values**, the columns of V **right singular vectors**, and the columns of U **left singular vectors** of A. Note that (1.47) implies the spectral decompositions

$$AA^* = U\Sigma\Sigma^* U^*, \qquad A^*A = V\Sigma^*\Sigma V^*$$

with

$$\Sigma\Sigma^* = \operatorname{diag}\big(\sigma_1^2, \ldots, \sigma_r^2, \underbrace{0, \ldots, 0}_{n-r \text{ times}}\big), \quad \Sigma^*\Sigma = \operatorname{diag}\big(\sigma_1^2, \ldots, \sigma_r^2, \underbrace{0, \ldots, 0}_{p-r \text{ times}}\big).$$

This shows that the singular values of A can be computed by computing the eigenvalues AA^* or A^*A and taking square roots.

In MATLAB, the SVD of a matrix A is computed by typing `[U,D,V] = svd(A)`. If A is not square, say $n > p$, a reduced or "economic" SVD is computed when typing `[U,D,V] = svd(A,0)`. This yields a matrix $U \in \mathbb{C}^{n\times p}$ with orthonormal columns and a unitary matrix $V \in \mathbb{C}^{p\times p}$ such that

$$A = U \begin{pmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_p \end{pmatrix} V^*.$$

From the proof of Theorem 1.22, it can be seen that the the first singular value σ_1 equals the matrix 2-norm of A. If A is square and invertible then the SVD takes the form

$$A = U\Sigma V^*, \quad \text{with} \quad \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$$

and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n > 0$. Inverting both sides of this equality yields

$$A^{-1} = V\Sigma^{-1}U^*, \quad \text{with} \quad \Sigma^{-1} = \operatorname{diag}(\sigma_1^{-1}, \ldots, \sigma_n^{-1}).$$

This shows that σ_n^{-1} is the largest singular value of A^{-1} and hence

$$\|A^{-1}\|_2 = \frac{1}{\sigma_n} \qquad \Rightarrow \qquad \kappa(A) = \|A\|_2 \, \|A^{-1}\|_2 = \frac{\sigma_1}{\sigma_n}.$$

Much more can (and will) be said about the properties of A revealed by its SVD. For example, the number of *nonzero* singular values is identical with the rank of A. This is actually how the rank is computed in MATLAB: the (numerical) rank is determined as the number of singular values above $\max\{n,p\} \times \varepsilon$, where $\varepsilon \approx 2 \cdot 10^{-16} \times \|A\|_2$.

The following result is central to many applications of the SVD. Given an SVD $A = U\Sigma V^*$, for any $k \leq \min\{n,p\}$ let $U_k \in \mathbb{C}^{n\times k}$ and $V_k \in \mathbb{C}^{p\times k}$ contain the first k columns of U and V, respectively. Then the matrix

$$A_k := U_k \Sigma_k V_k^*, \quad \text{with} \quad \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_k). \tag{1.50}$$

clearly has rank at most k. Moreover,

$$\|A - A_k\|_2 = \big\|\operatorname{diag}(\sigma_{k+1}, \ldots, \sigma_r)\big\|_2 = \sigma_{k+1},$$

which turns out to be minimal.

Theorem 1.23 (Eckart–Young theorem) *For $A \in \mathbb{C}^{n\times p}$, let A_k be defined as in* (1.50). *Then*

$$\|A - A_k\|_2 = \min\big\{\|A - B\|_2 : \ B \in \mathbb{C}^{n\times p} \text{ has rank at most } k\big\}.$$

This theorem also holds when the matrix 2-norm is replaced by the Frobenius matrix norm (or, more generally, by any unitarily invariant norm).

1.8 Application 4: Principal Component Analysis

Principal component analysis (PCA) is a procedure that originates from statistics, where it is used to transform observations of possibly correlated variables into values of uncorrelated variables, the so called **principal components**. Nowadays, PCA is used in a much broader context, primarily to reduce the dimensionality in all kinds of situations.

Suppose we have N independently drawn observations for K random variables $X_1, \ldots, X_K$. Each of the observations is arranged in a vector $\mathbf{x}_j \in \mathbb{R}^K$ with $j = 1, \ldots, N$. The sample mean is defined as $\overline{\mathbf{x}} := \frac{1}{N}(\mathbf{x}_1 + \cdots + \mathbf{x}_N)$ and the **sample covariance matrix** is defined as

$$C := \frac{1}{N-1} \sum_{j=1}^{N} (\mathbf{x}_j - \overline{x})(\mathbf{x}_j - \overline{x})^\mathsf{T}.$$

A diagonal entry c_{ii} estimates the variance of X_i, while an off-diagonal entry c_{ik} estimates the covariance between X_i and X_k. Defining $A := \big[\mathbf{x}_1 - \overline{x},\ \ldots,\ \mathbf{x}_N - \overline{x}\big] \in \mathbb{R}^{K\times N}$, we can equivalently write

$$C = \frac{1}{N-1} A A^\mathsf{T}.$$

We now aim to find a linear combination $Y_1 = w_1 X_1 + \cdots + w_N X_N$ with $w_1, \ldots, w_N \in \mathbb{R}$ and $w_1^2 + \cdots + w_N^2 = 1$ that captures most of the observed variation, that is, the variance of the new variable Y_1 is maximal. Since the corresponding observations of Y_1 are given by $\mathbf{w}^\mathsf{T}\mathbf{x}_1, \ldots, \mathbf{w}^\mathsf{T}\mathbf{x}_N$ with sample mean $\mathbf{w}^\mathsf{T}\overline{\mathbf{x}}$, this problem corresponds to

$$\arg \max_{\substack{\mathbf{w}\in\mathbb{R}^N \\ \|w\|_2 = 1}} \sum_{j=1}^{N} (\mathbf{w}^\mathsf{T}\mathbf{x}_j - \mathbf{w}^\mathsf{T}\overline{\mathbf{x}})^2 = \arg \max_{\substack{\mathbf{w}\in\mathbb{R}^N \\ \|w\|_2 = 1}} \|\mathbf{w}^\mathsf{T} A\|_2^2 = \arg \max_{\substack{\mathbf{w}\in\mathbb{R}^N \\ \|w\|_2 = 1}} \mathbf{w}^\mathsf{T} A A^\mathsf{T} \mathbf{w}.$$

By Theorem 1.16, an optimal vector $\mathbf{w}_1$ is given by an eigenvector corresponding to the largest eigenvalue of AA^T or, equivalently, a left singular vector corresponding to the largest singular value of A. This vector yields the **first principal vector** $\mathbf{w}_1$.

One obtains the second principal vector $\mathbf{w}_2$ by subtracting $\mathbf{w}_1$ from the data, that is, one considers $AA^\mathsf{T} - \mathbf{w}_1\mathbf{w}_1^\mathsf{T}$ and repeats the procedure. Thus, $\mathbf{w}_2$ is given by an eigenvector corresponding to the second largest eigenvalue of AA^T or, equivalently, a left singular vector corresponding to the second largest singular value of A. Repeating this procedure over and over again gives all K principal vectors

$$\mathbf{w}_1, \ldots, \mathbf{w}_K$$

as the left singular vectors of A. One is often only interested in the first few (dominant) principal vectors as they already represent most of the variation in the data. Using only the first two principal vectors allows for the 2D visualization of the data.[4]

1.9 Other Sources of Eigenvalue Problems

The selection of applications above may lead to the impression that eigenvalue problems in practice virtually always require the computation of a few smallest or largest eigenvalues of a symmetric matrix. This is *not* the case. For example, a linear stability analysis requires the computation of all eigenvalues on or close to the imaginary axis of a nonsymmetric matrix. Computational methods for decoupling the stable/unstable parts of a dynamical system require the computation of all eigenvalues in the left and/or right half of the complex plane. As we will see in the following chapters, the region of eigenvalues we are interested in determines the difficulty of the eigenvalue problem to a large extent (along with the matrix order and structure). This region also guides the choice of algorithm for solving an eigenvalue problem.

[4] See `http://rdcu.be/b9Y7` for an application to the study of early civilization.

Chapter 2

Eigenvalue Problems: Krylov subspace methods and friends

This chapter is concerned with Krylov subspace methods, the most popular class of methods for solving general large-scale eigenvalue problems. As the power method and the subspace iteration provide the foundation of Krylov subspace methods, we will discuss these two more basic methods first.

2.1 The power method

2.1.1 Application 4: Stationary distribution of Markov chains

To illustrate the idea behind the power method, let us consider a finite-dimensional **Markov chain** of the form

$$\mathbf{x}^{(k)} = P\mathbf{x}^{(k-1)}, \tag{2.1}$$

where $\mathbf{x}^{(k)} \in \mathbb{R}^n$ is a vector with entries between 0 and 1. There are n states $\{1, \ldots, n\}$ and the jth entry of $\mathbf{x}^{(k)}$ denotes the probability of being in state j at time k. The matrix P is called the **transition matrix** of the Markov chain; its entries are nonnegative and each column of P sums up to 1:

$$p_{ij} \geq 0 \quad \forall i, j = 1, \ldots, n, \qquad \sum_{i=1}^{n} p_{ij} = 1 \quad \forall j = 1, \ldots, n. \tag{2.2}$$

Any matrix satisfying (2.2) is called column stochastic. The second part of (2.2) can also be written as $\mathbf{e}^\mathsf{T} P = \mathbf{e}^\mathsf{T}$, showing that 1 is an eigenvalue of P with left[5] eigenvector $\mathbf{e}$.

Not seriously meant, Figure 2.1 gives an example for (2.1). At time $k = 0$, a mouse is located in cage I and all doors (denoted by red bars) between the cages are closed. If each entry of the vector $\mathbf{x}^{(k)} \in \mathbb{R}^4$ contains the probability that the mouse is in the corresponding cage, this means $\mathbf{x}^{(0)} = \big(1,\, 0,\, 0,\, 0\big)^\mathsf{T}$. At time $k = 1$ all doors are opened and the floor is electrified. The mouse runs at random (that is, with probability 1/3 each) in one of the other cages. As soon as the mouse is located in a different cage, the doors are closed (and the electrification is turned off for a moment). Hence, $\mathbf{x}^{(1)} = \big(0,\, 1/3,\, 1/3,\, 1/3\big)^\mathsf{T}$. The whole procedure is repeated and – according to the usual rules for calculating conditional probabilities – we have $\mathbf{x}^{(2)} = P\mathbf{x}^{(1)} = \frac{1}{9}\big(4,\, 1,\, 3,\, 1\big)$

[5] For a matrix A, a nonzero vector $\mathbf{v} \in \mathbb{C}^n$ is called left eigenvector if $\mathbf{v}^* A = \lambda \mathbf{v}^*$ for an eigenvalue λ of A.

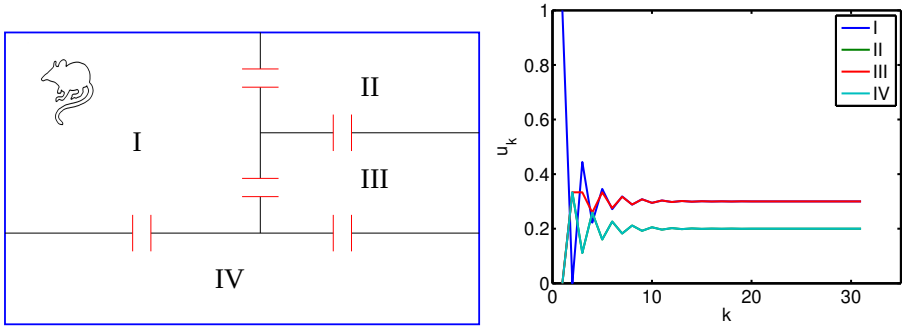


Figure 2.1. Left: *Mouse in a system of* 4 *cages.* Right: *Distribution of probabilities that mouse is in cage* $\{I,II,III,IV\}$ *vs.* k.

with

$$P = \begin{pmatrix} 0 & 1/2 & 1/3 & 1/2 \\ 1/3 & 0 & 1/3 & 0 \\ 1/3 & 1/2 & 0 & 1/2 \\ 1/3 & 0 & 1/3 & 0 \end{pmatrix}. \tag{2.3}$$

In general, we have $\mathbf{x}^{(k)} = P\mathbf{x}^{(k-1)} = P^k\mathbf{x}^{(0)}$. The evolution of the entries of $\mathbf{x}^{(k)}$ for increasing k is shown in Figure 2.1. It can be seen that the distribution becomes stationary as $k \to \infty$.

Note that the eigenvalues of P are given by $1, -2/3, -1/3, 0$. Let $\mathbf{u}_1$ be the eigenvector belonging to the eigenvalue 1, normalized such that the entries of $\mathbf{u}_1$ sum up to 1. Then $\mathbf{u}_1 = \frac{1}{10}(3,\, 2,\, 3,\, 2)^{\mathsf{T}}$, which coincides with the stationary probability distribution observed in Figure 2.1.

2.1.2 Basic form of the power method

Given a matrix $A \in \mathbb{C}^{n\times n}$ and an initial vector $\mathbf{x}^{(0)} \in \mathbb{C}^n$, the power method generates a sequence of vectors by repeated matrix-vector multiplication with A. However, unless A has some special properties, this will quickly lead to vectors of extremely small or extremely large norm. To avoid this effect, the vectors need to be normalized. Combining these two ideas leads to Algorithm 2.1.

Algorithm 2.1 (Basic form of power method)
Input: Matrix $A \in \mathbb{C}^{n\times n}$.
1: Choose starting vector $\mathbf{x}^{(0)} \in \mathbb{C}^n$.
2: $k = 0$.
3: **repeat**
4: Set $k := k + 1$.
5: Compute $\mathbf{y}^{(k)} := A\mathbf{x}^{(k-1)}$.
6: Normalize $\mathbf{x}^{(k)} := \mathbf{y}^{(k)}/\|\mathbf{y}^{(k)}\|_2$.
7: **until** convergence is detected

As we will show below, the vectors generated by Algorithm 2.1 converge to an eigenvector of A,

under rather mild conditions. Except when $\mathbf{x}^{(0)} \in \mathbb{C}^n$ is very particularly chosen, this eigenvector belongs to the so called **dominant eigenvalue** of A, that is, the eigenvalue of largest magnitude.

Given an approximation $\mathbf{x}^{(k)}$ to an eigenvector satisfying $\|\mathbf{x}^{(k)}\|_2 = 1$, how can an approximation μ to the corresponding eigenvalue be computed? For this purpose, consider the **residual**

$$\mathbf{r}^{(k)} := A\mathbf{x}^{(k)} - \mu\mathbf{x}^{(k)},$$

which would be zero if $(\mu, \mathbf{x}^{(k)})$ was an exact eigenpair of A. For simplicity, let us only consider the case that A and $\mathbf{x}^{(k)}$ are real. We choose μ such that the norm of $\mathbf{r}^{(k)}$ is minimized:

$$\mu_k := \arg\min_{\mu\in\mathbb{R}} \|A\mathbf{x}^{(k)} - \mu\mathbf{x}^{(k)}\|_2.$$

Since

$$\frac{\partial}{\partial\mu}\|A\mathbf{x}^{(k)} - \mu\mathbf{x}^{(k)}\|_2^2 = -2\big(\mathbf{x}^{(k)}\big)^T A\mathbf{x}^{(k)} + 2\mu\big(\mathbf{x}^{(k)}\big)^T\mathbf{x}^{(k)} = -2\big(\mathbf{x}^{(k)}\big)^T A\mathbf{x}^{(k)} + 2\mu,$$

this implies that

$$\mu_k = \big(\mathbf{x}^{(k)}\big)^T A\mathbf{x}^{(k)} = \rho_A\big(\mathbf{x}^{(k)}\big). \tag{2.4}$$

Remark 2.2 *In the case of a complex vector $\mathbf{x}^{(k)}$ and a complex matrix A one obtains, in a similar fashion, $\mu_k = \big(\mathbf{x}^{(k)}\big)^* A\mathbf{x}^{(k)} = \rho_A\big(\mathbf{x}^{(k)}\big)$. Note that the complex derivative of $\|A\mathbf{x}^{(k)} - \mu\mathbf{x}^{(k)}\|_2^2$ with respect μ does not exist and one needs to work with the real and imaginary parts of μ instead.*

When do we declare Algorithm 2.1 to be converged? A common criterion is to stop Algorithm 2.1 when the residual is sufficiently small:

$$\|A\mathbf{x}^{(k)} - \mu_k\mathbf{x}^{(k)}\|_2 \le \mathsf{tol} \tag{2.5}$$

for some user-specified tolerance tol. One has to be careful with interpreting the meaning of (2.5). Let $\mathbf{r}^{(k)} = A\mathbf{x}^{(k)} - \mu_k\mathbf{x}^{(k)}$ be as above. Then

$$(A + \triangle)\mathbf{x}^{(k)} = \mu_k\mathbf{x}^{(k)} \quad \text{with} \quad \triangle := -\mathbf{r}^{(k)}\big(\mathbf{x}^{(k)}\big)^*. \tag{2.6}$$

Since $\|\triangle\|_2 = \|\mathbf{r}^{(k)}\|_2 \le \mathsf{tol}$, this means that $(\mu_k, \mathbf{x}^{(k)})$ is an exact eigenpair of a slightly perturbed matrix.

2.1.3 An excursion into eigenvalue perturbation theory

There can be many reasons that we compute the eigenvalues of a *perturbed* matrix $A+\triangle$ instead of the (desired) eigenvalues of A. We have seen one reason above in (2.6); other manifestations of $\triangle$ include discretization, modelling, and roundoff errors. *If* A is Hermitian the impact of such a perturbation on eigenvalues is bounded.

Theorem 2.3 (Weyl perturbation theorem) *Let $\lambda_k(\cdot)$ denote the kth smallest eigenvalue of a Hermitian matrix. Then $|\lambda_k(A + \triangle) - \lambda_k(A)| \le \|\triangle\|_2$ holds for Hermitian matrices $A, \triangle$.*

Proof. For any nonzero vector $\mathbf{x}$, the Rayleigh quotient of $\triangle$ satisfies

$$|\rho_\triangle(x)| = \frac{|\mathbf{x}^*\triangle\mathbf{x}|}{|\mathbf{x}^*\mathbf{x}|} \le \frac{\|\triangle\|_2\|\mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} = \|\triangle\|_2.$$

By the Courant-Fischer minimax principle (see Theorem 1.18),

$$\begin{aligned}\lambda_k(A+\triangle) &= \min_{\substack{\mathcal{X} \text{ subspace of } \mathbb{C}^n \\ \dim(\mathcal{X})=k}} \max_{\substack{\mathbf{x}\in\mathcal{X} \\ \mathbf{x}\neq 0}} \rho_{A+\triangle}(\mathbf{x}) = \min_{\substack{\mathcal{X} \text{ subspace of } \mathbb{C}^n \\ \dim(\mathcal{X})=k}} \max_{\substack{\mathbf{x}\in\mathcal{X} \\ \mathbf{x}\neq 0}} \big(\rho_A(\mathbf{x}) + \rho_\triangle(\mathbf{x})\big) \\ &\leq \min_{\substack{\mathcal{X} \text{ subspace of } \mathbb{C}^n \\ \dim(\mathcal{X})=k}} \max_{\substack{\mathbf{x}\in\mathcal{X} \\ \mathbf{x}\neq 0}} \rho_A(\mathbf{x}) + \|\triangle\|_2 = \lambda_k(A) + \|\triangle\|_2.\end{aligned}$$

Analogously, one shows $\lambda_k(A) = \lambda_k((A+\triangle)-\triangle) \leq \lambda_k(A+\triangle)+\|\triangle\|_2$. The inequalities together imply the result of the theorem. □

Another way to view the result of Theorem 2.3 is that eigenvalues of Hermitian matrices are Lipschitz continuous with Lipschitz constant 1 (when choosing the spectral norm on the matrix space). Note that the jth eigenvalue is not necessarily differentiable (EFY: Give an example). Many variations and refinements of Theorem 2.3 exist; see Section 4.3 in [HorJ13].

Lipschitz continuity usually fails to hold for *multiple* eigenvalues of *non-Hermitian* matrices. For example, the perturbed matrix

$$A+\triangle = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 1\\ \varepsilon & 0\end{bmatrix}, \quad \varepsilon > 0, \quad \varepsilon \approx 0,$$

has eigenvalues $\pm\sqrt{\varepsilon}$ and, hence, the eigenvalue 0 of A is not Lipschitz continuous with respect to arbitrarily small perturbations of A.[6]

For a *simple* eigenvalue λ, let $\mathbf{x}, \mathbf{y} \neq 0$ denote the corresponding, normalized right/left eigenvectors of A, that is,

$$A\mathbf{x} = \lambda\mathbf{x}, \quad \mathbf{y}^*A = \lambda\mathbf{y}^*, \quad \|\mathbf{x}\|_2 = \|\mathbf{y}\|_2 = 1.$$

Then $\mathbf{y}^*\mathbf{x} \neq 0$ (EFY). Using the implicit function theorem, it can be shown that there exist differentiable (in fact, analytic) functions $\mathbf{x}(\varepsilon)$ and $\lambda(\varepsilon)$ such that

$$(A+\varepsilon\triangle)\mathbf{x}(\varepsilon) = \lambda(\varepsilon)x(\varepsilon), \quad \lambda(0) = \lambda, \quad \mathbf{x}(\varepsilon) = \mathbf{x}(0).$$

The derivative of this relation with respect to ε at $\varepsilon = 0$ gives

$$A\dot{\mathbf{x}}(0) + \triangle\mathbf{x} = \dot{\lambda}(0)\mathbf{x} + \lambda\dot{\mathbf{x}}(0).$$

Applying $\mathbf{y}^*$ to both sides, dividing by $\mathbf{y}^*\mathbf{x}$ and taking absolute values gives

$$|\dot{\lambda}(0)| = \frac{|\mathbf{y}^*\triangle\mathbf{x}|}{|\mathbf{y}^*\mathbf{x}|} \leq \frac{\|\triangle\|_2}{|\mathbf{y}^*\mathbf{x}|},$$

where the inequality becomes an equality for the perturbation direction $\triangle = \mathbf{y}\mathbf{x}^*$. For this reason $\mathsf{cond}(\lambda) = 1/|\mathbf{y}^*\mathbf{x}|$ is called the **eigenvalue condition number** of λ.

Applied to the power method, this shows (by the Taylor expansion) that (2.6) implies the existence of an eigenvalue of A such that

$$|\lambda - \mu_k| \leq \mathsf{cond}(\lambda) \times \|\triangle\|_2 + O(\|\triangle\|_2^2), \tag{2.7}$$

provided that the eigenvalues of A are simple. Eigenvalue condition numbers can be computed in MATLAB with the command `condeig`.

[6] One can show that any eigenvalue of A is $1/L$-Hölder continuous, where L denotes the size of the largest Jordan block containing this eigenvalue.

2.1.4 Convergence analysis of the power method

In the following, we will study the convergence of the iterates $\mathbf{x}^{(k)}$ produced by the power method towards an eigenvector $\mathbf{u}$. Recall that eigenvectors are not uniquely defined, any scalar multiple $\alpha\mathbf{u}$ is also an eigenvector. It is therefore not appropriate to consider the difference $\|\mathbf{x}^{(k)} - \mathbf{u}\|_2$ when studying the convergence. Instead, we will consider the angle.

One way to define the angle $\theta \in [0, \pi]$ between two general vectors $\mathbf{x}, \mathbf{y} \in \mathbb{C}^n$ is to consider the orthogonal projection of $\mathbf{y}$ onto the orthogonal complement of $\operatorname{span}(\mathbf{x})$:

$$\Pi_{\mathbf{x}}^{\perp}(\mathbf{y}) := \left(I - \frac{\mathbf{x}\mathbf{x}^*}{\|\mathbf{x}\|_2^2}\right)\mathbf{y}$$

and set

$$\sin\theta(\mathbf{x}, \mathbf{y}) := \frac{\|\Pi_{\mathbf{x}}^{\perp}(\mathbf{y})\|_2}{\|\mathbf{y}\|_2} = \min_{\hat{\mathbf{x}} \in \operatorname{span}(\mathbf{x})} \frac{\|\mathbf{y} - \hat{\mathbf{x}}\|_2}{\|\mathbf{y}\|_2}, \tag{2.8}$$

where the latter expression follows from the fact that the orthogonal projection minimizes the distance between $\mathbf{y}$ and $\operatorname{span}(\mathbf{x})$. Using $\sin^2\theta + \cos^2\theta = 1$, the more common formula

$$\cos\theta(\mathbf{x}, \mathbf{y}) = \frac{\|\Pi_{\mathbf{x}}(\mathbf{y})\|_2}{\|\mathbf{y}\|_2} = \frac{|\mathbf{x}^*\mathbf{y}|}{\|\mathbf{x}\|_2\,\|\mathbf{y}\|_2}$$

follows. Moreover,

$$\tan\theta(\mathbf{x}, \mathbf{y}) = \frac{\|\Pi_{\mathbf{x}}^{\perp}(\mathbf{y})\|_2}{\|\Pi_{\mathbf{x}}(\mathbf{y})\|_2}. \tag{2.9}$$

As the angle between two vectors is invariant under rescaling, we may drop the normalization in Algorithm 2.1 in the convergence analysis and study the convergence of the sequence

$$\mathbf{x}^{(k)} := A^k\mathbf{x}^{(0)} \tag{2.10}$$

instead. In the following, we assume that A is diagonalizable, i.e., there is an invertible matrix S of eigenvectors such that

$$S^{-1}AS = \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n), \qquad |\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n|. \tag{2.11}$$

Moreover, we assume that the largest eigenvalue λ_1 is strictly larger than all other eigenvalues:

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|. \tag{2.12}$$

Since $S^{-1}A^kS = \big(S^{-1}AS\big)^k = \Lambda^k$, we can rewrite (2.10) as

$$\mathbf{y}^{(k)} := \Lambda^k\mathbf{y}^{(0)}, \qquad \mathbf{y}^{(0)} := S^{-1}\mathbf{x}^{(0)}, \quad \mathbf{y}^{(k)} := S^{-1}\mathbf{x}^{(k)}. \tag{2.13}$$

Note that $\mathbf{e}_1 = \big(1, 0, \ldots, 0\big)^{\mathsf{T}}$ is an eigenvector of Λ belonging to the eigenvalue λ_1.

Let us partition

$$\mathbf{y}^{(k)} = \begin{pmatrix} y_1^{(k)} \\ \mathbf{y}_2^{(k)} \end{pmatrix}, \quad \Lambda = \begin{pmatrix} \lambda_1 & \mathbf{0} \\ \mathbf{0} & \Lambda_2 \end{pmatrix}.$$

Then (2.13) can be written as

$$\mathbf{y}^{(k)} = \begin{pmatrix} y_1^{(k)} \\ \mathbf{y}_2^{(k)} \end{pmatrix} = \begin{pmatrix} \lambda_1^k y_1^{(0)} \\ \Lambda_2^k \mathbf{y}_2^{(0)} \end{pmatrix} = \lambda_1^k \begin{pmatrix} y_1^{(0)} \\ \big(\frac{\Lambda_2}{\lambda_1}\big)^k \mathbf{y}_2^{(0)} \end{pmatrix}.$$

Since Λ_2 is diagonal, we have

$$\left\| \left(\frac{\Lambda_2}{\lambda_1} \right)^k \right\|_2 = \left\| \frac{\Lambda_2}{\lambda_1} \right\|_2^k = \left| \frac{\lambda_2}{\lambda_1} \right|^k,$$

where $|\lambda_2|/|\lambda_1| < 1$, see (2.12). Since $\Pi_{\mathbf{e}_1}^{\perp}(\mathbf{y}^{(k)}) = \begin{pmatrix} 0 \\ \mathbf{y}_2^{(k)} \end{pmatrix}$ and $\Pi_{\mathbf{e}_1}(\mathbf{y}^{(k)}) = \begin{pmatrix} y_1^{(k)} \\ 0 \end{pmatrix}$, it follows using (2.9) that

$$\tan\theta(\mathbf{e}_1, \mathbf{y}^{(k)}) = \frac{\|\mathbf{y}_2^{(k)}\|_2}{|y_1^{(k)}|_2} \leq \left| \frac{\lambda_2}{\lambda_1} \right|^k \frac{\|\mathbf{y}_2^{(0)}\|_2}{|y_1^{(0)}|} = \left| \frac{\lambda_2}{\lambda_1} \right|^k \tan\theta(\mathbf{e}_1, \mathbf{y}^{(0)}), \tag{2.14}$$

where we assumed that $y_1^{(0)} \neq 0$. The following theorem extends the derivation above to the diagonalizable case, in which the condition number of the diagonalizer S enters the bound as an additional constant. Note that this condition number is *not* invariant under the scaling of the eigenvectors; it is common practice to scale the eigenvectors to have norm one when forming S.

Theorem 2.4 *Let $A \in \mathbb{C}^{n\times n}$ be diagonalizable, $S^{-1}AS = \Lambda$, with the eigenvalues ordered as in (2.11). Assume that $|\lambda_1| > |\lambda_2|$ and that the eigenvector $\mathbf{u}_1 = S\mathbf{e}_1$ belonging to λ_1 satisfies $\|\mathbf{u}_1\|_2 = 1$. Then the iterates $\mathbf{x}^{(k)}$ of the power method satisfy*

$$\sin\theta(\mathbf{u}_1, \mathbf{x}^{(k)}) \leq \frac{\kappa(S)}{|\mathbf{v}_1^* \mathbf{x}^{(0)}|} \left| \frac{\lambda_2}{\lambda_1} \right|^k \sin\theta(\mathbf{u}_1, \mathbf{x}^{(0)}),$$

provided that $\mathbf{v}_1^* \mathbf{x}^{(0)} \neq 0$, where $\mathbf{v}_1 = S^{-*}\mathbf{e}_1$ is a left eigenvector belonging to λ_1.

Proof. The assumption $\mathbf{v}_1^* \mathbf{x}^{(0)} \neq 0$ implies $y_1^{(0)} = \mathbf{e}_1^* S^{-1} S \mathbf{y}^{(0)} = \mathbf{v}_1^* \mathbf{x}^{(0)} \neq 0$. Therefore, all assumptions of the derivations above are satisfied and (2.14) holds.

It remains to transform (2.14) back to the original coordinate system. For this purpose, we first rewrite (2.14) in terms of sines:

$$\sin\theta(\mathbf{e}_1, \mathbf{y}^{(k)}) \leq \tan\theta(\mathbf{e}_1, \mathbf{y}^{(k)}) \leq \left| \frac{\lambda_2}{\lambda_1} \right|^k \tan\theta(\mathbf{e}_1, \mathbf{y}^{(0)}) = \frac{\|\mathbf{y}^{(0)}\|_2}{|y_1^{(0)}|} \left| \frac{\lambda_2}{\lambda_1} \right|^k \sin\theta(\mathbf{e}_1, \mathbf{y}^{(0)}).$$

Using (2.8), it follows that

$$\begin{aligned}
\sin\theta(\mathbf{u}_1, \mathbf{x}^{(k)}) &= \sin\theta(S\mathbf{e}_1, S\mathbf{y}^{(k)}) = \min_{\mathbf{z}\in \mathrm{span}(\mathbf{y}^{(k)})} \|S\mathbf{e}_1 - S\mathbf{z}\|_2 \\
&\leq \|S\|_2 \min_{\mathbf{z}\in \mathrm{span}(\mathbf{y}^{(k)})} \|\mathbf{e}_1 - \mathbf{z}\|_2 = \|S\|_2 \sin\theta(\mathbf{e}_1, \mathbf{y}^{(k)}) \\
&\leq \frac{\|S\|_2 \, \|\mathbf{y}^{(0)}\|_2}{|y_1^{(0)}|} \left| \frac{\lambda_2}{\lambda_1} \right|^k \sin\theta(\mathbf{e}_1, \mathbf{y}^{(0)}) \\
&= \frac{\|S\|_2 \, \|\mathbf{y}^{(0)}\|_2}{|y_1^{(0)}|} \left| \frac{\lambda_2}{\lambda_1} \right|^k \min_{\mathbf{z}\in \mathrm{span}(\mathbf{e}_1)} \frac{\|\mathbf{z} - \mathbf{y}^{(0)}\|_2}{\|\mathbf{y}^{(0)}\|_2} \\
&\leq \frac{\|S\|_2 \|S^{-1}\|_2 \|\mathbf{y}^{(0)}\|_2}{|y_1^{(0)}|} \left| \frac{\lambda_2}{\lambda_1} \right|^k \min_{\mathbf{z}\in \mathrm{span}(\mathbf{e}_1)} \frac{\|S\mathbf{z} - S\mathbf{y}^{(0)}\|_2}{\|\mathbf{y}^{(0)}\|_2} \\
&= \frac{\kappa(S)}{|y_1^{(0)}|} \left| \frac{\lambda_2}{\lambda_1} \right|^k \sin\theta(\mathbf{u}_1, \mathbf{x}^{(0)}),
\end{aligned}$$

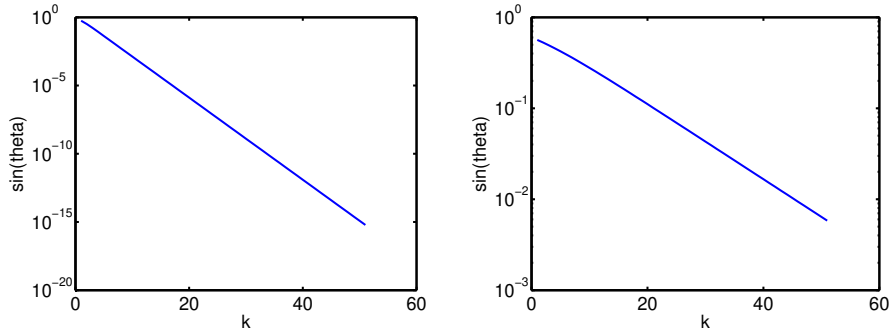


Figure 2.2. *Convergence of* $\sin\theta(\mathbf{u}_1, \mathbf{x}^{(k)})$ *for* $A = \mathrm{diag}(2, 1, 1)$ *(left figure) and for* $A = \mathrm{diag}(1.1, 1, 1)$ *(right figure), with a randomly chosen starting vector.*

which shows the desired result. □

Theorem 2.4 depends on a number of assumptions, which merit some discussion:

1. The assumption that A is diagonalizable was made to simplify the derivation. This can be replaced by the weaker assumption that λ_1 is a simple eigenvalue.
2. In contrast, the assumption $|\lambda_1| > |\lambda_2|$ can *not* be relaxed. For example, consider the matrix $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ with eigenvalues 1 and -1. It is easy to see that the power method does not converge for this matrix unless a particular choice of starting vector is made. Also, $\lambda_1 \approx \lambda_2$ is not favorable, as the ratio $|\lambda_2/\lambda_1|$ determines the convergence rate, see Figure 2.2.
3. The assumption $\mathbf{v}_1^* \mathbf{x}^{(0)} \neq 0$ is necessary, but in practice this is virtually always true. Even in the unlikely situation that $\mathbf{v}_1^* \mathbf{x}^{(0)} = 0$ holds in exact arithmetic, this relation is almost certainly destroyed by roundoff error; see also Section 2.1.6 below.

2.1.5 Merits of the Hermitian case

Figure 2.3 displays not only the eigenvector error $\sin\theta(\mathbf{u}_1, \mathbf{x}^{(k)})$ but also the eigenvalue error $|\mu_k - \lambda_1|$ for $\mu_k = \left(\mathbf{x}^{(k)}\right)^* A \mathbf{x}^{(k)} / \left(\mathbf{x}^{(k)}\right)^* \mathbf{x}^{(k)}$. It turns out that the eigenvalue converges much faster than the eigenvector in the case when A is symmetric. The following theorem explains this observation.

Theorem 2.5 *Let A be a Hermitian matrix with the ordered eigenvalues $\lambda_1, \ldots, \lambda_n$ and assume that $\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. Let $\mathbf{u}_1$ be an eigenvector belonging to λ_1. Then the iterates $\mathbf{x}^{(k)}, \mu_k$ of the power method satisfy*

$$\begin{aligned} \tan\theta(\mathbf{u}_1, \mathbf{x}^{(k)}) &\leq \left|\frac{\lambda_2}{\lambda_1}\right|^k \tan\theta(\mathbf{u}_1, \mathbf{x}^{(0)}), \\ |\lambda_1 - \mu_k| &\leq (\lambda_1 - \lambda_n) \left|\frac{\lambda_2}{\lambda_1}\right|^{2k} \tan^2\theta(\mathbf{u}_1, \mathbf{x}^{(0)}), \end{aligned}$$

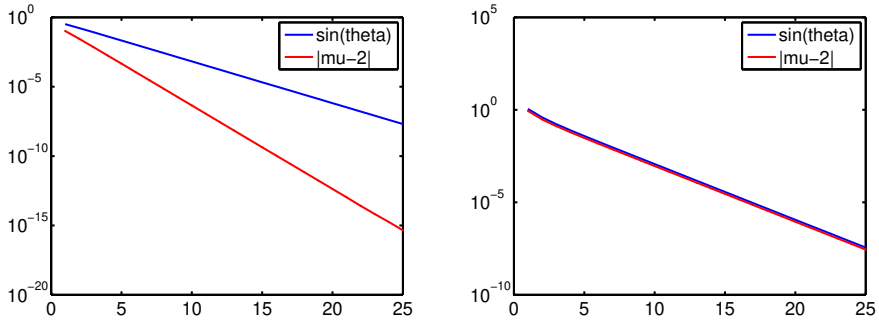


Figure 2.3. *Convergence of* $\sin\theta(\mathbf{u}_1, \mathbf{x}^{(k)})$ *and* $|\mu_k - 2|$ *for* $A = \mathrm{diag}(2,1,1)$ *(left figure) and for* $A = P\cdot\mathrm{diag}(2,1,1)\cdot P^{-1}$ *(right figure) for a random invertible matrix* P*, with a randomly chosen starting vector.*

provided that $\mathbf{u}_1^*\mathbf{x}^{(0)} \neq 0$.

Proof. The first inequality follows directly from (2.14), noting that S can be chosen unitary for Hermitian matrices and angles do not change under unitary transformations. For the second inequality, we assume for simplicity that $\mathbf{x}^{(k)}$ is scaled such that the first component of $\mathbf{y}^{(k)} = S^*\mathbf{x}^{(k)}$ is 1. Then

$$\begin{aligned}\mu_k &= \frac{\left(\mathbf{x}^{(k)}\right)^* A\mathbf{x}^{(k)}}{\left(\mathbf{x}^{(k)}\right)^*\mathbf{x}^{(k)}} = \frac{\left(\mathbf{y}^{(k)}\right)^*\Lambda\mathbf{y}^{(k)}}{\left(\mathbf{y}^{(k)}\right)^*\mathbf{y}^{(k)}} \\ &= \frac{\lambda_1 + \left(\mathbf{y}_2^{(k)}\right)^*\Lambda_2\mathbf{y}_2^{(k)}}{1 + \|\mathbf{y}_2^{(k)}\|_2^2} = \left(\lambda_1 + \left(\mathbf{y}_2^{(k)}\right)^*\Lambda_2\mathbf{y}_2^{(k)}\right)\left(1 - \sin^2\theta(\mathbf{e}_1, \mathbf{y}^{(k)})\right),\end{aligned}$$

where we used the easily verifiable relation $1-\sin^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}) = \frac{1}{1+\|\mathbf{y}_2^{(k)}\|_2^2}$ in the last step. Hence,

$$\begin{aligned}\lambda_1 - \mu_k &= \lambda_1\sin^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}) - \left(\mathbf{y}_2^{(k)}\right)^*\Lambda_2\mathbf{y}_2^{(k)}\cdot\cos^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}) \\ &\leq \lambda_1\sin^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}) - \lambda_n\|\mathbf{y}_2^{(k)}\|_2^2\cdot\cos^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}) \qquad \text{(Theorem 1.16)} \\ &= (\lambda_1-\lambda_n)\sin^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}) \leq (\lambda_1-\lambda_n)\tan^2\theta(\mathbf{e}_1,\mathbf{y}^{(k)}),\end{aligned}$$

which completes the proof. ∎

2.1.6 Probabilistic analysis

Controlling the constants

As already discussed above, the conditions $\mathbf{v}_1^*\mathbf{x}^{(0)} \neq 0$ in Theorem 2.4 and $\mathbf{u}_1^*\mathbf{x}^{(0)} \neq 0$ in Theorem 2.5 are almost always satisfied if $\mathbf{x}^{(0)}$ is randomly chosen. In the following, we will perform a more detailed analysis explaining why this inner products are not expected become very small (which would otherwise lead to large constants in the theorems).

To simplify the discussion we consider real vectors (the complex case is discussed in the exercises) and use $\mathbf{u}^T\mathbf{x}$ to denote the inner product under consideration. It is very common to type `x = randn(n); x = x / norm(x);` in order to generate the random starting vector $\mathbf{x}$. This corresponds to choosing $\mathbf{x}$ according to the uniform distribution on the $(n-1)$-sphere. Equivalently, x is obtained by normalizing a standard normal random vector. The following lemma due to Dixon[7] shows that $|\mathbf{u}^T\mathbf{x}|$ is unlikely to become very small.

Lemma 2.6 *Let $n \geq 2$ and $\mathbf{u} \in \mathbb{R}^n$ with $\|\mathbf{u}\|_2 = 1$. Let $\mathbf{x}$ be a real random vector of length n distributed according to the uniform distribution on the $(n-1)$-sphere. Then for any $0 < \sigma < 1$ it holds that*

$$P(|\mathbf{u}^T\mathbf{x}| < \sigma) \leq \sqrt{\frac{2n}{\pi}}\sigma. \tag{2.15}$$

Proof. The following proof is "dense" and does not provide provide detailed arguments for basic facts from probability theory. All these arguments can be found in textbooks / wikipedia. Let Q be an orthogonal matrix such that $\mathbf{u} = Q\mathbf{e}_1$. Setting $\tilde{\mathbf{x}} = Q^T\mathbf{x}$, we have

$$\mathbf{u}^T\mathbf{x} = \mathbf{e}_1^T Q^T\mathbf{x} = \mathbf{e}_1^T\tilde{\mathbf{x}} = \tilde{x}_1.$$

As the distribution of standard normal random vectors is invariant under orthogonal transformations, the same holds for the uniform distribution on the $(n-1)$-sphere. In turn the distribution of $\tilde{x}_1^2$ is identical with the distribution of an entry of a normalized standard normal random vector of length n:

$$\tilde{x}_1^2 = \frac{y_1^2}{y_1^2 + \cdots + y_n^2}, \quad y_i \sim N(0,1), \quad i = 1, \ldots, n.$$

It follows from basic properties of beta random variables that $\tilde{x}_1^2$ is beta-distributed with parameters $1/2$, $(n-1)/2$. Therefore, the probability density function of $\tilde{x}_1^2$ is given by

$$f(t) = \alpha_n t^{-1/2}(1-t)^{(n-3)/2}, \quad \alpha_n = \frac{1}{B(1/2,(n-1)/2)} = \frac{\Gamma(n/2)}{\Gamma(1/2)\Gamma((n-1)/2)}.$$

By Stirling's formula, $\alpha_n \to \sqrt{n/2\pi}$ for $n \to \infty$ and, with some effort, it can be shown that $\alpha_n < \sqrt{n/2\pi}$ for $n \geq 2$. In turn, we have

$$P(|\mathbf{u}^T\mathbf{x}| < \sigma) = P((\mathbf{u}^T\mathbf{x})^2 < \sigma^2) = \alpha_n \int_0^{\sigma^2} t^{-1/2}(1-t)^{(n-3)/2}\,\mathrm{d}t = 2\alpha_n \int_0^{\sigma} (1-s^2)^{(n-3)/2}\,\mathrm{d}s.$$

For $n = 2$, the claimed result by direct evaluation of the integral. For $n \geq 3$, the claimed result follows from the inequality $1 - s^2 \leq 1$. ∎

[7] Dixon, John D. Estimating extremal eigenvalues and condition numbers of matrices. SIAM J. Numer. Anal. 20 (1983), no. 4, 812–814

Convergence without gaps

The convergence analysis of the power method assumes that there is a gap between the largest and second largest eigenvalues. In practice, the gap indeed plays a crucial role for the convergence of the *eigenvector*. On the other hand, the convergence of the *eigenvalue* can remain robust in the presence of small gaps and or even in the absence of a gap (see exercises). The following result[8] provides some insights into this observation.

Lemma 2.7 *Let $A \in \mathbb{R}^{n\times n}$ by a symmetric positive semi-definite matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$ and an associate orthonormal basis of eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_n$. Let $\mu_k = \rho_A(\mathbf{x}^{(k)})$ denote the eigenvalue approximation obtained after k steps of the power method with starting vector $\mathbf{x}^{(0)} \in \mathbb{R}^n$, $\|\mathbf{x}^{(0)}\|_2 = 1$. Then*

$$\mu_k \geq \lambda_1 \langle \mathbf{x}^{(0)}, \mathbf{u}_1 \rangle^{1/k}.$$

Proof. Because the Rayleigh quotient is invariant under multiplying the vector with a scalar, we have $\mu_k = \rho_A(\mathbf{x}^{(k)}) = \rho_A(A^k\mathbf{x}^{(0)})$. To simplify notation, we set $\mathbf{x} := \mathbf{x}^{(0)}$ and obtain

$$\mu_k = \frac{\mathbf{x}^T A^{2k+1}\mathbf{x}}{\mathbf{x}^T A^{2k}\mathbf{x}}.$$

Applying Jensen's inequality to the convex function $z \mapsto z^{\frac{2k+1}{2k}}$ it follows that

$$\mathbf{x}^T A^{2k+1}\mathbf{x} = \sum_{i=1}^{n} \lambda_i^{2k+1} \langle \mathbf{x}, \mathbf{u}_i \rangle^2 \geq \Big(\sum_{i=1}^{n} \lambda_i^{2k} \langle \mathbf{x}, \mathbf{u}_i \rangle^2 \Big)^{\frac{2k+1}{2k}}$$

and hence

$$\mu_k \geq \Big(\sum_{i=1}^{n} \lambda_i^{2k} \langle \mathbf{x}, \mathbf{u}_i \rangle^2 \Big)^{\frac{1}{2k}} \geq \lambda_1 \langle \mathbf{x}, \mathbf{u}_1 \rangle^{1/k}.$$

□

Lemmas 2.6 and 2.7 can be combined to show that μ_k is within a constant factor of λ_1 with high probability after only a few iterations. Setting $\sigma = 2^{-k}$ in Lemma 2.6 it follows that

$$\frac{1}{2}\lambda_1 \leq \mu_k \leq \lambda_1$$

holds with probability at least $1 - \sqrt{\frac{2n}{\pi}} 2^{-k}$. For example, when $n = 10\,000$, the success probability is at least 99.995% for $k = 20$. A more detailed (and more technical) analysis and tighter bounds have been obtained by, e.g., Kuczyński and Woźniakowski.[9] Section 3 of the recent preprint by Kireeva and Tropp[10] has an elegant and accesible derivation of bounds on the expected value of the eigenvalue error.

2.1.7 Inverse iteration

For the applications in Chapter 1, not the largest eigenvalue but the smallest eigenvalue is of interest. More generally, we will now discuss the question how to compute the eigenvalue closest to a given **shift** $\sigma \in \mathbb{C}$. The following simple lemma provides the idea.

[8] This result is due to Henrik Eisenmann, MPI Leipzig.

[9] Kuczyński, J.; Woźniakowski, H. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. SIAM J. Matrix Anal. Appl. 13 (1992), no. 4, 1094–1122.

[10] A. Kireeva, J. A. Tropp. Randomized matrix computations: Themes and variations. arXiv:2402.17873, 2024

Lemma 2.8 *Let $A \in \mathbb{C}^{n\times n}$ with a shift $\sigma \in \mathbb{C}$ such that $A - \sigma I$ is invertible. Then $(\lambda, \mathbf{x})$ is an eigenpair of A if and only if $(\frac{1}{\lambda-\sigma}, \mathbf{x})$ is an eigenpair of $(A - \sigma I)^{-1}$.*

Proof. The statement follows from the equivalences

$$A\mathbf{x} = \lambda\mathbf{x} \quad \Leftrightarrow \quad (A - \sigma I)\mathbf{x} = (\lambda - \sigma)\mathbf{x} \quad \Leftrightarrow \quad \frac{1}{\lambda - \sigma}\mathbf{x} = (A - \sigma I)^{-1}\mathbf{x}.$$

□

It is now straightforward to modify Algorithm 2.1 such that it converges to the eigenvector belonging to the eigenvalue closest to σ; simply replace A by $(A - \sigma I)^{-1}$. This means that in every iteration a linear system needs to be solved. If this is done via an LU factorization with column pivoting,

$$P(A - \sigma I) = LU,$$

it is advisable to compute this factorization once in the beginning and reuse the factors for the rest of the algorithm. This reduces the cost of each iteration to $O(n^2)$ for dense matrices.

Algorithm 2.9 (Inverse iteration)
Input: Matrix $A \in \mathbb{C}^{n\times n}$, shift $\sigma \in \mathbb{C}$.
1: Choose starting vector $\mathbf{x}^{(0)} \in \mathbb{C}^n$ with $\|\mathbf{x}^{(0)}\|_2 = 1$.
2: Compute LU factorization $P(A - \sigma I) = LU$.
3: $k = 0$.
4: **repeat**
5: Set $k := k + 1$.
6: Compute $\mathbf{y}^{(k)} := U^{-1}L^{-1}P\mathbf{x}^{(k-1)}$.
7: Compute $\mu_k := \sigma + 1/\big(\mathbf{x}^{(k-1)}\big)^* \mathbf{y}^{(k)}$.
8: Normalize $\mathbf{x}^{(k)} := \mathbf{y}^{(k)}/\|\mathbf{y}^{(k)}\|_2$.
9: **until** convergence is detected

Theorems 2.4 and 2.5 can be easily modified to describe the convergence of Algorithm 2.9, with the notable difference that the convergence rate $|\lambda_2|/|\lambda_1|$ is replaced by

$$\frac{|\tilde{\lambda}_1 - \sigma|}{|\tilde{\lambda}_2 - \sigma|}, \tag{2.16}$$

where $\tilde{\lambda}_1$ is the eigenvalue of A closest to σ and $\tilde{\lambda}_2$ is second closest to σ.

Example 2.10 Let $A \in \mathbb{R}^{n\times n}$ be the finite-difference discretization of $-\triangle$ on the interval $[0, 1]$ with zero Dirichlet boundary conditions. For $n = 100$, the smallest eigenvalue of A is given by 9.86880867886... The following table shows the convergence of Algorithm 2.9 with shift $\sigma = 0$ and shift $\sigma = 9$, respectively.

	$\sigma = 0$		$\sigma = 9$	
k	$\sin^2\theta(\mathbf{u}_1, \mathbf{x}^{(k)})$	$\lvert\lambda_1 - \mu_k\rvert$	$\sin^2\theta(\mathbf{u}_1, \mathbf{x}^{(k)})$	$\lvert\lambda_1 - \mu_k\rvert$
1	2.91e-01	7.69e-01	3.13e-02	8.38e-04
2	5.05e-02	1.98e-02	6.16e-04	3.21e-07
3	1.12e-02	9.42e-04	1.63e-05	2.16e-10
4	2.73e-03	5.53e-05	4.59e-07	7.86e-12

Obviously, the choice of a shift closer to the eigenvalue accelerates convergence.

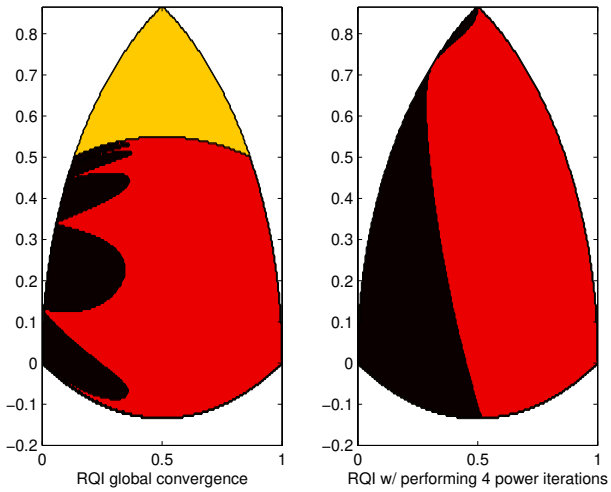


Figure 2.4. *Global convergence of the Rayleigh quotient iteration for* $A = \mathrm{diag}(2, 1.99, 1)$. *The* MATLAB *file used for generating this figure is available from the lecture's homepage.*

The observation from the previous example motivates the following adaptive choice of the shift: set σ to the Rayleigh quotient of the previous iterate. This idea leads to the so called Rayleigh quotient iteration, see Algorithm 2.11.

Algorithm 2.11 (Rayleigh quotient iteration)
Input: Matrix $A \in \mathbb{C}^{n\times n}$.
1: Choose starting vector $\mathbf{x}^{(0)} \in \mathbb{C}^n$ with $\|\mathbf{x}^{(0)}\|_2 = 1$.
2: $k = 0$.
3: **repeat**
4: Set $k := k + 1$.
5: Set $\sigma_k := \mathbf{x}^{(k-1)^*} A \mathbf{x}^{(k-1)}$.
6: Solve $(A - \sigma_k I)\mathbf{y}^{(k)} = \mathbf{x}^{(k-1)}$ for $\mathbf{y}^{(k)}$.
7: $\mathbf{x}^{(k)} := \mathbf{y}^{(k)}/\|\mathbf{y}^{(k)}\|_2$.
8: **until** convergence is detected

The Rayleigh quotient iteration converges locally quadratically for general matrices and locally cubically for Hermitian matrices. However, this convergence acceleration comes with two drawbacks.

- For almost every starting vector, the Rayleigh quotient iteration converges. However, it is in general not clear to which eigenpair it will converge. This question of global convergence can be quite subtle, as shown in Figure 2.4 for the matrix $A = \mathrm{diag}(2, 1.99, 1)$. In the figure, a triangle on the sphere is drawn. Each corner of the triangle represents an eigenvector of A (eigenvector for $\lambda = 2$: bottom left corner; eigenvector for $\lambda = 1.99$: bottom right corner; eigenvector for $\lambda = 1$: top middle corner). For each point of the triangle, Algorithm 2.11 is applied with a starting vector, whose distance to one of the eigenvectors is determined by the distance of the point to the corresponding corner. The point of the triangle is colored

according to which eigenvector Algorithm 2.11 converges. For example, brown corresponds to the eigenvector for lambda = 2. On the left plot, it can be seen that even a starting vector very close to the eigenvector for $\lambda = 2$ may converge to a different eigenvector. This phenomenon can be partly avoided by performing first a few, say 4, steps of the (inverse) power iteration without adapting the shift. This is demonstrated by the right plot, which reveals a much larger domain of attraction for $\lambda = 2$.

- In contrast to Algorithm 2.9, it is not possible to reuse a single LU factorization in Algorithm 2.11, because the system matrix of the linear system to be solved changes in every iteration. For a dense matrix A, this implies $O(n^3)$ instead of $O(n^2)$ operations for each iteration.

For these reasons, Algorithm 2.11 itself is rarely used in practice. However, some of its variants, such as the Jacobi-Davidson algorithm, are practically important.

2.1.8 Computing more than one eigenvalue

Suppose we have computed the largest eigenvalue λ_1 of a Hermitian matrix A with a normalized eigenvector $\mathbf{u}_1$, and we are interested in computing also the second largest eigenvalue λ_2. In theory, this can be achieved simply by initializing the power method with a starting vector $\mathbf{x}^{(0)}$ orthogonal to $\mathbf{u}_1$. From the spectral decomposition $A = \sum_{j=1}^{n} \lambda_j \mathbf{u}_j \mathbf{u}_j^*$, it follows that all subsequent iterates stay orthogonal to $\mathbf{u}_1$. In effect, the first eigenvector is not "seen"; the power method behaves as it was applied to the deflated matrix $\tilde{A} = \sum_{j=2}^{n} \lambda_j \mathbf{u}_j \mathbf{u}_j^*$, with largest eigenvalue λ_2. In practice, roundoff error will quickly destroy this orthogonality relationship. To retain orthogonality numerically, it is necessary to perform one step of Gram-Schmidt orthogonalization in every iteration:

$$\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k)} - (\mathbf{u}_1^* \mathbf{x}^{(k)}) \mathbf{u}_1.$$

Although this idea appears nice and can be easily extended to more than two eigenvalues, it is rarely used in practice. A notable exception is when one or more eigenvectors are readily available and require no computation (for example, in Application 3). In all other cases, the algorithms discussed in the subsequent sections are significantly more effective at computing several eigenvalues.

2.2 Subspace iteration

To extend the power method to the computation of p eigenvectors, a tempting idea is to replace the starting vector $\mathbf{x}^{(0)} \in \mathbb{C}^n$ by a block of p vectors $X^{(0)} \in \mathbb{C}^{n \times p}$. However, without a suitable normalization this idea will not work, simply because all columns of $A^k X^{(0)}$ converge to the same vector, the eigenvector belonging to the largest eigenvalue of A. This effect can be prevented by keeping the columns of the iterates orthogonal with the help of the QR factorization.

Theorem 2.12 (QR factorization) *Let $X \in \mathbb{C}^{n \times p}$ with $n \geq p$. Then there is a unitary matrix $Q \in \mathbb{C}^{n \times n}$ such that*

$$X = QR, \quad \textit{with} \quad R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = \begin{pmatrix} \diagdown \\ 0 \end{pmatrix}, \tag{2.17}$$

that is, $R_1 \in \mathbb{C}^{p \times p}$ is an upper triangular matrix.

In MATLAB, a QR factorization is computed by typing `[Q,R] = qr(X)`. For a rectangular matrix $X \in \mathbb{C}^{n \times p}$ with $n > p$, the full QR factorization is actually rarely needed and the

following "economic" variant is more useful. By letting $Q_1 \in \mathbb{C}^{n\times p}$ contain the first p columns of Q, the factorization (2.17) implies

$$X = Q_1 R_1 = Q_1 \cdot \boxed{}. \tag{2.18}$$

In the following, we will not need (2.17) and simply refer to (2.18) as the QR factorization of X. This variant can be more directly computed in MATLAB by typing `[Q,R] = qr(X,0)`.

If $\mathbf{x}_1, \ldots, \mathbf{x}_p$ and $\mathbf{q}_1, \ldots, \mathbf{q}_p$ denote the columns of X and Q_1, respectively, then (2.18) implies

$$\operatorname{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_j\} = \operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_j\} \quad \forall j = 1, \ldots, p. \tag{2.19}$$

In particular, $\operatorname{span}(X) = \operatorname{span}(Q_1)$ implies that X can be replaced by the (numerically better behaved) matrix Q_1 if only the space spanned by the columns of X is of interest. This idea leads to the subspace iteration in Algorithm 2.13.

Algorithm 2.13 (Basic form of subspace iteration)
Input: Matrix $A \in \mathbb{C}^{n\times n}$.
1: Choose starting matrix $X^{(0)} \in \mathbb{C}^{n\times p}$ with $\left(X^{(0)}\right)^* X^{(0)} = I_p$.
2: $k = 0$.
3: **repeat**
4: Set $k := k + 1$.
5: Compute $Y^{(k)} := AX^{(k-1)}$.
6: Compute QR factorization (2.18): $Y^{(k)} = QR$.
7: Set $X^{(k)} := Q$.
8: **until** convergence is detected

2.2.1 Angles between subspaces

To study the convergence of Algorithm 2.13, we need to be able to compare two p-dimensional subspaces $\mathcal{X}, \mathcal{Y} \subset \mathbb{C}^n$ with each other. Let the columns of $X, Y \in \mathbb{C}^{n\times p}$ contain orthonormal bases of $\mathcal{X}, \mathcal{Y}$, respectively. Then the singular values $0 \le \sigma_1 \le \cdots \le \sigma_p \le 1$ of the matrix X^*Y play a crucial role in understanding the relation between $\mathcal{X}$ and $\mathcal{Y}$. We call

$$\theta_i(\mathcal{X}, \mathcal{Y}) := \arccos \sigma_i, \quad i = 1, \ldots, p, \tag{2.20}$$

the **canonical angles between $\mathcal{X}$ and $\mathcal{Y}$**. Clearly, θ_1 is the largest canonical angle, which coincides for $p = 1$ with the notion of angles between vectors discussed in Section 2.1.4. More generally, we have the geometric characterization

$$\theta_1(\mathcal{X}, \mathcal{Y}) = \max_{\substack{\mathbf{x}\in\mathcal{X}\\ \mathbf{x}\neq 0}} \min_{\substack{\mathbf{y}\in\mathcal{Y}\\ \mathbf{y}\neq 0}} \theta(x, y).$$

It follows that $\theta_1(\mathcal{X}, \mathcal{Y}) = 0$ if and only if $\mathcal{X} \cap \mathcal{Y}^\perp \neq \{0\}$.

There are a number of equivalent characterizations for θ_1.

Lemma 2.14 *With the notation defined above,*

$$\sin \theta_1(\mathcal{X}, \mathcal{Y}) = \|XX^* - YY^*\|_2.$$

Note that XX^* and YY^* are orthogonal projectors on $\mathcal{X}$ and $\mathcal{Y}$, respectively. For our analysis, the following result will be more useful.

Lemma 2.15 *Let $Q \in \mathbb{C}^{(n-p)\times p}$, and*

$$\mathcal{X} = \operatorname{span}\begin{pmatrix} I_p \\ 0 \end{pmatrix}, \qquad \mathcal{Y} = \operatorname{span}\begin{pmatrix} I_p \\ Q \end{pmatrix}.$$

Then $\theta_1(\mathcal{X}, \mathcal{Y}) = \arctan \|Q\|_2$.

Proof. The columns of $\binom{I_p}{Q}(I + Q^*Q)^{-1/2}$ form an orthonormal basis of $\mathcal{Y}$. By definition (2.20), this implies that $\cos\theta_1(\mathcal{X}, \mathcal{Y})$ is the smallest singular value of $(I + Q^*Q)^{-1/2}$. By the SVD of Q, it follows that

$$\cos\theta_1(\mathcal{X}, \mathcal{Y}) = \frac{1}{\sqrt{1 + \|Q\|_2^2}}.$$

This shows the result. ∎

2.2.2 Convergence of subspace iteration

We are now ready to extend the convergence analysis of the power method from Section 2.1.4 to the subspace iteration. For the moment, we assume that A is diagonal and that the p eigenvalues of largest magnitude are separated from the rest of the spectrum:

$$A = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \qquad |\lambda_1| \geq \cdots \geq |\lambda_p| > |\lambda_{p+1}| \geq \cdots \geq |\lambda_n|.$$

As in the analysis of the power method, we can drop the "normalization" performed by the QR factorization in the course of the subspace iteration. Then the subspace $\mathcal{X}^{(k)}$ generated in the kth iteration of Algorithm 2.13 is spanned by the columns of $X^{(k)} := A^k X^{(0)}$. Let us partition

$$A = \begin{pmatrix} \Lambda_1 & 0 \\ 0 & \Lambda_2 \end{pmatrix}, \quad X^{(k)} = \begin{pmatrix} X_1^{(k)} \\ X_2^{(k)} \end{pmatrix}, \quad \Lambda_1, X_1^{(k)} \in \mathbb{C}^{p\times p}.$$

Then

$$X^{(k)} = A^k X^{(0)} = \begin{pmatrix} \Lambda_1^k X_1^{(0)} \\ \Lambda_2^k X_2^{(0)} \end{pmatrix} = \begin{pmatrix} I \\ Q^{(k)} \end{pmatrix} \Lambda_1^k X_1^{(0)}$$

with

$$Q^{(k)} = \Lambda_2^k X_2^{(0)} \big(\Lambda_1^k X_1^{(0)}\big)^{-1},$$

where we have made the (weak) assumption that $X_1^{(0)}$ is invertible.

Let $\mathcal{E} = \operatorname{span}\binom{I_p}{0}$, the invariant subspace associated with $\lambda_1, \ldots, \lambda_p$. Then Lemma 2.15 implies

$$\tan\theta_1(\mathcal{E}, \mathcal{X}^{(k)}) = \|Q^{(k)}\|_2 \leq \big\|\Lambda_2^k\big\|_2 \big\|\Lambda_1^{-k}\big\|_2 \big\|X_2^{(0)}\big(X_1^{(0)}\big)^{-1}\big\|_2 = \left|\frac{\lambda_{p+1}}{\lambda_p}\right|^k \tan\theta_1(\mathcal{E}, \mathcal{X}^{(0)}).$$

In summary, we have that $\mathcal{X}^{(k)}$ converges linearly at a rate $|\lambda_{p+1}|/|\lambda_p|$ to the invariant subspace associated with the p dominant eigenvalues. Similarly as for the power method, this statement remains valid for general A, provided of course that the eigenvalue gap $|\lambda_p| > |\lambda_{p+1}|$ holds.

Remark 2.16 *Algorithm 2.13 has a nested structure. For* any *$i = 1, \ldots, p$, the first i columns of $X^{(k)}$ correspond to a subspace iteration of A applied to the first i columns of $X^{(0)}$. If $|\lambda_i| > |\lambda_{i+1}|$, the analysis above implies that these columns converge to the invariant subspace associated with the i dominant eigenvalues at a rate of $|\lambda_{i+1}|/|\lambda_i|$.*

2.2.3 Ritz value/vector extraction

Although we know that the iterates produced by Algorithm 2.13 provide increasingly good approximations to an invariant subspace, our eventual goal is to compute eigenvalues and eigenvectors. More generally, the question is: Given an orthonormal basis $U \in \mathbb{C}^{n\times p}$ for a subspace $\mathcal{U}$, how can we obtain good eigenvalue/eigenvector approximations $(\mu, \mathbf{u})$ for a matrix A from $\mathcal{U}$? The key to this question is to impose the **Galerkin condition**

$$A\mathbf{u} - \mu\mathbf{u} \perp \mathcal{U}. \tag{2.21}$$

Since $\mathbf{u} \in \mathcal{U}$, we can write $\mathbf{u} = U\mathbf{w}$ for some $\mathbf{w} \in \mathbb{C}^p$. Then (2.21) becomes equivalent to

$$U^*AU\mathbf{w} - \mu\mathbf{w} = 0.$$

In other words, $(\mu, \mathbf{w})$ is an eigenpair for the $p \times p$ matrix U^*AU.

Definition 2.17 *Consider a matrix $A \in \mathbb{C}^{n\times n}$ and an orthonormal basis $U \in \mathbb{C}^{n\times p}$ for a subspace $\mathcal{U}$. Then an eigenvalue μ of U^*AU is called* **Ritz value** *and $\mathbf{u} = U\mathbf{w}$ is called* **Ritz vector** *for an eigenvector $\mathbf{w}$ of U^*AU. The pair $(\mu, \mathbf{u})$ is called* **Ritz pair**.

Let $U = X^{(k)}$ be the orthonormal basis produced by Algorithm 2.13 during the kth iteration. Then we compute the p Ritz pairs

$$(\mu_1, \mathbf{u}_1), \quad (\mu_2, \mathbf{u}_2), \quad \ldots, \quad (\mu_p, \mathbf{u}_p)$$

belonging to the p eigenvalues $\mu_1, \ldots, \mu_p$ of U^*AU. Each of these Ritz pairs provides an approximation for an eigenpair of A. The norm of the residual,

$$\|A\mathbf{u}_i - \mu_i\mathbf{u}_i\|_2,$$

provides a verification of the approximation quality. It can be shown that the convergence of the ith largest Ritz vector towards the ith largest eigenvector is linear at a rate of $|\lambda_{p+1}/\lambda_i|$. Note that the space $\mathcal{X}^{(k)}$ as a whole converges only at a rate $|\lambda_{p+1}/\lambda_p|$. This can be seen by running the MATLAB script `ssiter.m` from the homepage.

Similarly as for the power method, the Ritz values converge to the ith eigenvalue at an improved rate $|\lambda_{p+1}/\lambda_i|^2$ if A is Hermitian.

2.2.4 Relation to the QR algorithm

The QR algorithm behind the MATLAB commands `schur` and `eig` is the workhorse for solving small- to medium-sized eigenvalue problems. Its most basic form is obtained from applying the subspace iteration in Algorithm 2.13 for $p = n$, with the identity matrix as the starting "vectors". In the first iteration, this corresponds to computing a QR factorization

$$A = Q^{(1)}R^{(1)}.$$

The second iteration of subspace iteration would require a QR factorization of $AQ^{(1)}$. Premultiplication of this matrix with $\left(Q^{(1)}\right)^*$ yields

$$\left(Q^{(1)}\right)^* AQ^{(1)} = R^{(1)}Q^{(1)} =: A^{(1)}.$$

For this matrix, we perform a QR factorization $A^{(1)} = Q^{(2)}R^{(2)}$, and so forth, see Algorithm 2.18. After k iterations of this algorithm, the accumulated matrix

$$Q_k = Q^{(1)}Q^{(2)}\cdots Q^{(k)}$$

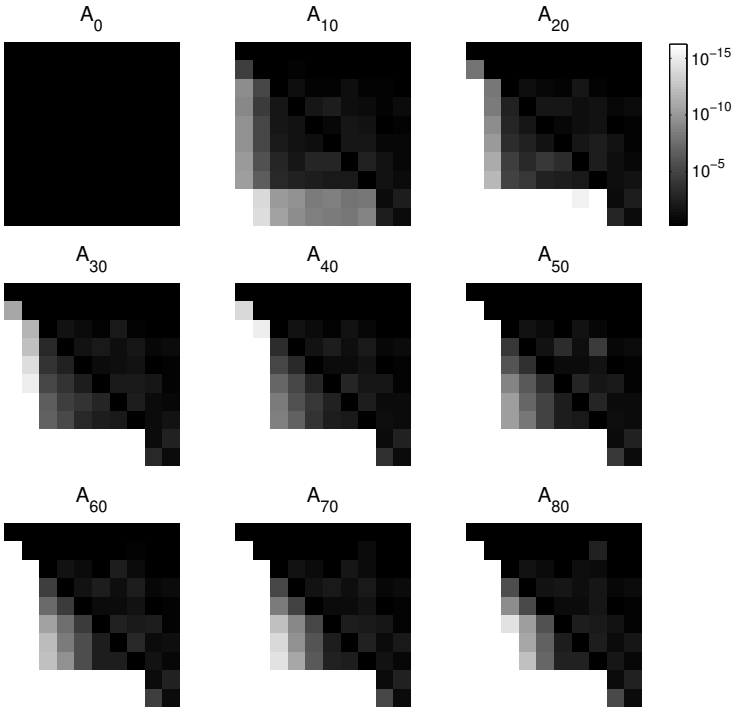


Figure 2.5. *Convergence pattern of Algorithm 2.18.*

is identical with the matrix $X^{(k)}$ obtained from k subspace iterations. According the Remark 2.16, we expect the first i columns of Q_k to converge to an invariant subspace for every $i = 1, \ldots, n$, at least when all eigenvalues of A are different in magnitude. In turn, the matrix

$$A^{(k)} = Q_k^* A Q_k$$

is expected to converge linearly to upper triangular form.

Algorithm 2.18 (Basic form of QR algorithm)
Input: Matrix $A \in \mathbb{C}^{n\times n}$.
1: $k = 0$. Set $A^{(0)} := A$.
2: **repeat**
3: Set $k := k + 1$.
4: Compute QR factorization (2.18): $A^{(k-1)} =: Q^{(k)} R^{(k)}$.
5: Compute $A^{(k)} = R^{(k)} Q^{(k)}$.
6: **until** convergence is detected

The following example demonstrates that the convergence of this iteration can be rather slow, especially if the eigenvalues of A are not well separated.

Example 2.19 *Consider the* 10×10 *matrix* $A = X\Lambda X^{-1}$*, where*

$$\Lambda = \mathrm{diag}(4, 2, 0.9, 0.8, 0.7, 0.6, 0.59, 0.58, 0.1, 0.09)$$

and X is a random matrix with condition number $\kappa_2(X) = 10^3$. We applied 80 iterations of Algorithm 2.18 to A; the absolute values of the entries in $A^{(k)}$ for $k = 0, 10, \ldots, 80$ are displayed in Figure 2.5. First, the eigenvalue cluster $\{0.1, 0.09\}$ converges in the bottom right corner, followed by the individual eigenvalues 4 and 2 in the top left corner. The other eigenvalues converge much slower. Only after $k = 1909$ iterations is the norm of the strictly lower triangular part of $A^{(k)}$ less than $10^{-16}\,\|A\|_2$. The diagonal entries of $A^{(k)}$ approximate the diagonal entries of Λ in descending order.

State-of-the-art implementations of the QR algorithm employ shifting techniques to avoid the observed slow linear convergence. Moreover, a preliminary reduction to so called Hessenberg form reduces the cost in each iteration of the QR algorithm from $O(n^3)$ to $O(n^2)$.

2.3 Krylov subspace methods

The fundamental idea of **Krylov subspace methods** is to extract eigenvector approximations from the space spanned by the iterates of the power method.

Definition 2.20 *Let $A \in \mathbb{C}^{n\times n}$ and $\mathbf{x} \in \mathbb{C}^n$. Then*

$$\mathcal{K}_k(A, \mathbf{x}) = \operatorname{span}\{\mathbf{x}, A\mathbf{x}, \ldots, A^{k-1}\mathbf{x}\}$$

is called the kth **Krylov subspace** *for A and $\mathbf{x}$.*

Figure 2.6 illustrates that the Krylov subspace $\mathcal{K}_k(A, \mathbf{x})$ may contain a much better approximation to the eigenvector belonging to the largest eigenvalue compared to the kth iterate of the power method. Moreover, it also contains increasingly good approximations to the eigenvectors belonging to the second largest eigenvalue, to the third largest eigenvalue, and so on.

Unless $k > n$, it can usually be expected that the dimension of $\mathcal{K}_k(A, \mathbf{x})$ is k. Exceptions are if $\mathbf{x} = 0$, if $\mathbf{x}$ is an eigenvector of A, or more generally if $\mathbf{x}$ is contained in an invariant subspace of dimension smaller than k.

The following basic result links Krylov subspaces to polynomials. Note that for a polynomial $p(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2 + \cdots + \alpha_{k-1} t^{k-1}$, the corresponding matrix polynomial of $A \in \mathbb{C}^{n\times n}$ is defined as

$$p(A) = \alpha_0 I_n + \alpha_1 A + \alpha_2 A^2 + \cdots + \alpha_{k-1} A^{k-1}. \tag{2.22}$$

Lemma 2.21 *Let $A \in \mathbb{C}^{n\times n}$ and $\mathbf{x} \in \mathbb{C}^n$. Then*

$$\mathcal{K}_k(A, \mathbf{x}) = \big\{p(A)\mathbf{x} : \ p \in \Pi_{k-1}\big\},$$

where Π_{k-1} denotes the space of polynomials of degree at most $k-1$.

Proof. This result follows immediately from the observation that every element of $\mathcal{K}_k(A, \mathbf{x})$ can be written as a linear combination

$$\alpha_0\mathbf{x} + \alpha_1 A\mathbf{x} + \cdots + \alpha_{k-1} A^{k-1}\mathbf{x} = p(A)\mathbf{x},$$

where $p(A)$ is as in (2.22) ∎

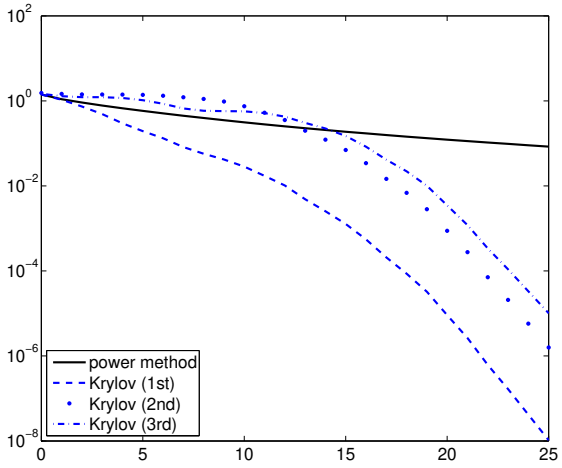


Figure 2.6. *Angles* $\theta(\mathbf{e}_1, A^k\mathbf{x})$ *(power method),* $\theta(\mathbf{e}_1, \mathcal{K}_k(A,\mathbf{x}))$ *(Krylov 1st),* $\theta(\mathbf{e}_2, \mathcal{K}_k(A,\mathbf{x}))$ *(Krylov 2nd),* $\theta(\mathbf{e}_3, \mathcal{K}_k(A,\mathbf{x}))$ *(Krylov 3rd) for* $A = \operatorname{diag}(0.95, 0.95^2, \ldots, 0.95^{100})$ *and a random vector* $\mathbf{x}$.

2.3.1 The Arnoldi method

We need to have a basis to work with $\mathcal{K}_k(A,\mathbf{x})$. The most natural choice

$$\{\mathbf{x}, A\mathbf{x}, \ldots, A^{k-1}\mathbf{x}\} \tag{2.23}$$

is numerically extremely badly behaved. Because of the convergence of the power method, the vectors of this basis tend to lie in the same direction, rendering the basis useless in finite-precision arithmetic. For a similar reason, applying Gram-Schmidt directly to (2.23) will, in finite-precision arithmetic, not yield a useful approach because the damage is already done when forming $A^j\mathbf{x}$. The following result allows to avoid explicit reference to $A^j\mathbf{x}$.

Lemma 2.22 *Suppose that* $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}$ *and* $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}, \mathbf{u}_j$ *are bases for* $\mathcal{K}_{j-1}(A,\mathbf{x})$ *and* $\mathcal{K}_j(A,\mathbf{x})$*, respectively. Then*

$$\mathcal{K}_{j+1}(A,\mathbf{x}) = \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_j, A\mathbf{u}_j\}.$$

Proof. Since $A^{j-1}\mathbf{x} \in \mathcal{K}_j(A,\mathbf{x}) = \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_j\}$, there are $\beta_1, \ldots, \beta_j \in \mathbb{C}$ such that

$$A^{j-1}\mathbf{x} = \beta_1\mathbf{u}_1 + \cdots + \beta_{j-1}\mathbf{u}_{j-1} + \beta_j\mathbf{u}_j.$$

Moreover we have $\beta_j \neq 0$, because $\beta_j = 0$ would imply that $A^{j-1}\mathbf{x} \in \mathcal{K}_{j-1}(A,\mathbf{x})$ and therefore $\dim \mathcal{K}_j(A,\mathbf{x}) \leq j-1$, contradicting the assumption that $\mathcal{K}_j(A,\mathbf{x})$ has j basis elements. From

$$A^j\mathbf{x} - \beta_j A\mathbf{u}_j = A\sum_{i=1}^{j-1}\beta_i\mathbf{u}_i \in \mathcal{K}_j(A,\mathbf{x}),$$

it follows that the vectors $A^j\mathbf{x}$ and a nonzero multiple of $A\mathbf{u}_j$ only differ by an element from $\mathcal{K}_j(A,\mathbf{x})$. Hence, it does not matter whether we add $A^j\mathbf{x}$ or $A\mathbf{u}_j$ to $\mathcal{K}_j(A,\mathbf{x})$; in both cases $\mathcal{K}_{j+1}(A,\mathbf{x})$ is obtained. ∎

Given an orthonormal basis $U_j = \big(\mathbf{u}_1, \ldots, \mathbf{u}_j\big) \in \mathbb{C}^{n\times j}$ of $\mathcal{K}_j(A, \mathbf{x})$, Lemma 2.22 shows that we can obtain an orthonormal basis of $\mathcal{K}_{j+1}(A, \mathbf{x})$ if we add $\mathbf{w} = A\mathbf{u}_j$ to the basis and apply one step of the Gram-Schmidt procedure:

$$\mathbf{h}_j := U_j^*\mathbf{w}, \quad \widetilde{\mathbf{u}}_{j+1} := \mathbf{w} - U_j\mathbf{h}_j, \quad \mathbf{u}_{j+1} = \widetilde{\mathbf{u}}_{j+1}/\|\widetilde{\mathbf{u}}_{j+1}\|_2. \tag{2.24}$$

This leads to Algorithm 2.23, the Arnoldi method.

Algorithm 2.23 (Arnoldi method)
Input: Matrix $A \in \mathbb{C}^{n\times n}$, starting vector $\mathbf{x} \neq 0$, $k \in \mathbb{N}$.
Output: Orthonormal basis $U_{k+1} = \big(\mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{u}_{k+1}\big)$ of $\mathcal{K}_{k+1}(A, \mathbf{x})$.

1: Set $\mathbf{u}_1 = \mathbf{x}/\|\mathbf{x}\|_2, \quad U_1 = \mathbf{u}_1$.
2: **for** $j = 1, 2, \ldots, k$ **do**
3: Compute $\mathbf{w} = A\mathbf{u}_j$.
4: Compute $\mathbf{h}_j = U_j^*\mathbf{w}$.
5: Compute $\widetilde{\mathbf{u}}_{j+1} = \mathbf{w} - U_j\mathbf{h}_j$.
6: Set $h_{j+1,j} = \|\widetilde{\mathbf{u}}_{j+1}\|_2$.
7: Set $\mathbf{u}_{j+1} = \widetilde{\mathbf{u}}_{j+1}/h_{j+1,j}$.
8: Set $U_{j+1} = \big(U_j, \mathbf{u}_{j+1}\big)$.
9: **end for**

It will pay off to save the coefficients $\mathbf{h}_j$ and $h_{j+1,j}$ generated in the course of the Arnoldi method. Let $\mathbf{h}_j =: \big(h_{1j}, h_{2j}, \ldots, h_{jj}\big)^\mathsf{T}$, then all coefficients can be collected in the $(k+1)\times k$ matrix

$$\widetilde{H}_k = \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1k} \\ h_{21} & h_{22} & \ddots & \vdots \\ & \ddots & \ddots & h_{k-1,k} \\ & & h_{k,k-1} & h_{kk} \\ & & & h_{k+1,k} \end{pmatrix}. \tag{2.25}$$

If we drop the last row, we obtain the $k \times k$ matrix

$$H_k = \begin{pmatrix} h_{11} & h_{12} & \cdots & h_{1k} \\ h_{21} & h_{22} & \ddots & \vdots \\ & \ddots & \ddots & h_{k-1,k} \\ & & h_{k,k-1} & h_{kk} \end{pmatrix}.$$

A matrix of this form, which has zero entries below the first subdiagonal, is called (upper) **Hessenberg matrix**.

The relation (2.24), which led to the Arnoldi method, can be rewritten as

$$\begin{aligned} A\mathbf{u}_j &= U_j\mathbf{h}_j + \widetilde{\mathbf{u}}_{j+1} = U_j\mathbf{h}_j + h_{j+1,j}\mathbf{u}_{j+1} \\ &= \big(U_j, \mathbf{u}_{j+1}\big)\begin{pmatrix} \mathbf{h}_j \\ h_{j+1,j} \end{pmatrix} = U_{k+1}\begin{pmatrix} \mathbf{h}_j \\ h_{j+1,j} \\ \mathbf{0} \end{pmatrix}. \end{aligned}$$

Collecting these relations for $j = 1, \ldots, k$ yields the so called **Arnoldi decomposition**:

$$AU_k = U_{k+1}\widetilde{H}_k,$$

or – equivalently –

$$AU_k = U_k H_k + h_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^{\mathsf{T}}, \tag{2.26}$$

where $\mathbf{e}_k = (0, \ldots, 0, 1)^{\mathsf{T}} \in \mathbb{C}^k$. In particular, multiplying both sides with U_k^* gives

$$H_k = U_k^* A U_k. \tag{2.27}$$

Remark 2.24 *Even though the extreme numerical instabilities discussed above are avoided by Algorithm 2.23, this algorithm still suffers from a (milder) numerical instability inherited from the Gram-Schmidt process. This instability happens if the vector* $\mathbf{w} = \mathbf{A}\mathbf{u}_j$ *is nearly contained in span*(U_j)*, resulting in nearly identical vectors* $\mathbf{w}$ *and* $U_j\mathbf{h}_j = U_j U_j^*\mathbf{w}$*, which leads to numerical cancellation in the subtraction* $\widetilde{\mathbf{u}}_{j+1} = \mathbf{w} - U_j\mathbf{h}_j$*. As a consequence, we have* **loss of orthogonality***: the vector* $\mathbf{u}_{j+1}$ *will not be nearly orthogonal to the columns of* U_j*. This is illustrated by the script* `loss.m` *from the homepage.*

There is a remarkably simple cure to this instability. In critical cases, when numerical cancellation has taken place, we just apply Gram-Schmidt once again. Such critical cases are signaled by a small norm of $\widetilde{\mathbf{u}}_{j+1}$*. In practice, it is common to consider* $\widetilde{\mathbf{u}}_{j+1}$ *small when* $\|\widetilde{\mathbf{u}}_{j+1}\|_2 < 0.7\|\mathbf{w}\|_2$*. To summarize, we insert the following lines after Line 5 of Algorithm 2.23:*

if $\|\widetilde{\mathbf{u}}_{j+1}\|_2 < 0.7\|\mathbf{w}\|_2$ ***then***
$\widehat{\mathbf{h}}_j = U_j^*\widetilde{\mathbf{u}}_{j+1}, \quad \mathbf{h}_j = \mathbf{h}_j + \widehat{\mathbf{h}}_j, \quad \widetilde{\mathbf{u}}_{j+1} = \widetilde{\mathbf{u}}_{j+1} - U_j\widehat{\mathbf{h}}_j$
end if

It can be shown that if such a **reorthogonalization** *is performed at most once in each iteration then the resulting basis is in nearly all cases numerically orthogonal, see also the script* `noloss.m` *from the homepage. The additional computational effort is – relative to the other parts of the Arnoldi method – often negligible.*

Ritz value/vector extraction

As discussed in Section 2.2.3 in the context of subspace iteration, we can extract Ritz pairs $(\mu, \mathbf{v})$ with $\mathbf{v} \in \mathcal{K}_k(A, \mathbf{x})$ by imposing the Galerkin condition

$$A\mathbf{v} - \mu\mathbf{v} \;\perp\; \mathcal{K}_k(A, \mathbf{x}). \tag{2.28}$$

Having an orthonormal basis U_k of $\mathcal{K}_k(A, \mathbf{x})$, there is $\mathbf{z} \in \mathbb{C}^k$ such that $\mathbf{v} = U_k\mathbf{z}$ and (2.28) becomes equivalent to

$$0 = U_k^*(A\mathbf{v} - \mu\mathbf{v}) = U_k^*(AU_k\mathbf{z} - \mu U_k\mathbf{z}) = H_k\mathbf{z} - \mu\mathbf{z},$$

where we used (2.27). Hence, a Ritz pair $(\mu, \mathbf{v})$ is obtained from any eigenpair $(\mu, \mathbf{z})$ of H_k by setting $\mathbf{v} = U_k\mathbf{z}$.

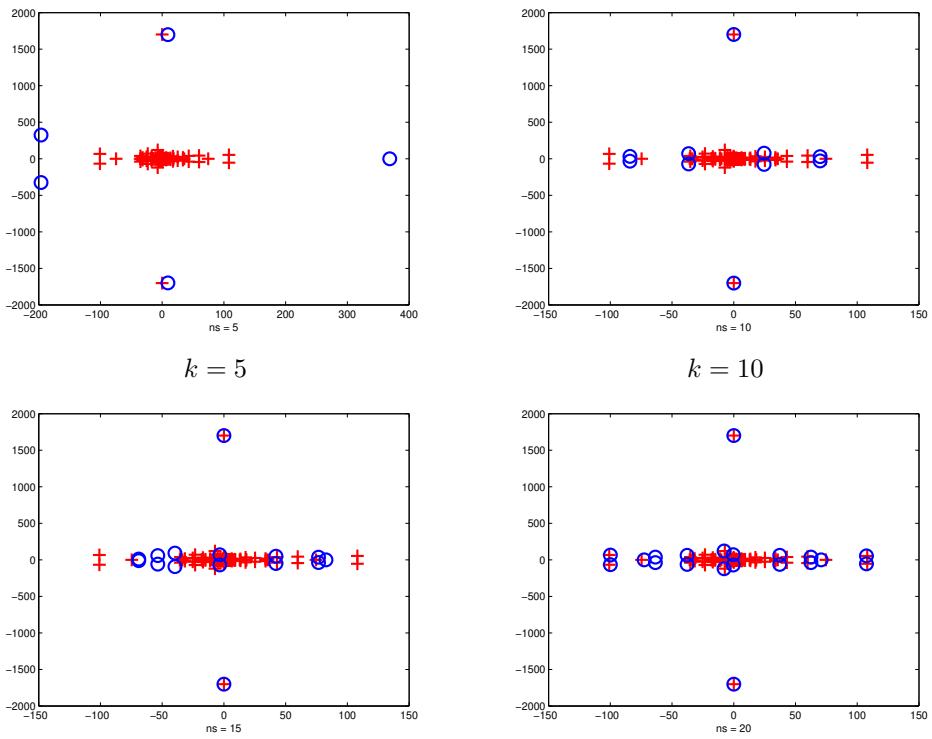
To check whether such a Ritz pair $(\mu, \mathbf{v})$ is a good approximation, we test – as usual – the

norm of the residual:

$$\begin{aligned}\|A\mathbf{v} - \mu\mathbf{v}\|_2 &= \|AU_k\mathbf{z} - \mu U_k\mathbf{z}\|_2 \\ &= \|U_k H_k\mathbf{z} + h_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^\mathsf{T}\mathbf{z} - \mu U_k\mathbf{z}\|_2 \\ &= \|U_k(\underbrace{H_k\mathbf{z} - \mu\mathbf{z}}_{=0}) + h_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^\mathsf{T}\mathbf{z}\|_2 \\ &= |h_{k+1,k}||\mathbf{e}_k^\mathsf{T}\mathbf{z}|\,\underbrace{\|\mathbf{u}_{k+1}\|_2}_{=1} = |h_{k+1,k}||\mathbf{e}_k^\mathsf{T}\mathbf{z}|\end{aligned}$$

where we used the Arnoldi decomposition (2.26) in the second step. Consequently, the norm of the residual can be very cheaply evaluated by multiplying $h_{k+1,k}$ with the kth entry of the eigenvector $\mathbf{z}$ of H_k.

Convergence considerations



$k = 5$ $k = 10$

$k = 15$ $k = 20$

Figure 2.7. *Ritz values (marked by ∘) computed by the Arnoldi method applied to a nonsymmetric* 479×479 *matrix (*`load west0479`*) with eigenvalues marked by +.*

Figure 2.7, which can be generated with the script `arngo.m` from the homepage, is representative for the typical convergence behavior of the Arnoldi method.

The Ritz values produced by the Arnoldi method tend to converge first to exterior eigenvalues.

As a rule of thumb, convergence is faster to an eigenvalue well separated from the rest of the spectrum.

A formal convergence analysis of the Arnoldi method turns out to be surprisingly difficult. For the moment, let us assume that $A = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ and that λ_1 is the eigenvalue of interest, with the corresponding eigenvector $\mathbf{e}_1$. In contrast to the analyses of the power method and the subspace iteration, no ordering of the eigenvalues needs to be assumed. By Lemma 2.21, any element in the Krylov subspace $\mathcal{K}_k(A, \mathbf{x})$ can be written as $p(A)\mathbf{x}$ for some $p \in \Pi_{k-1}$. After a suitable partitioning of $A = \begin{pmatrix} \lambda_1 & 0 \\ 0 & A_2 \end{pmatrix}$ and $x = \begin{pmatrix} x_1 \\ \mathbf{x}_2 \end{pmatrix}$:

$$p(A)\mathbf{x} = \begin{pmatrix} p(\lambda_1)x_1 \\ p(A_2)\mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{p(A_2)\mathbf{x}_2}{p(\lambda_1)x_1} \end{pmatrix} p(\lambda_1)x_1,$$

provided that $p(\lambda_1)x_1 \neq 0$. From Lemma 2.15, it follows that

$$\tan\theta\big(\mathbf{e}_1, p(A)\mathbf{x}\big) = \left\| \frac{p(A_2)\mathbf{x}_2}{p(\lambda_1)x_1} \right\|_2 \leq \frac{\|p(A_2)\|_2 \|\mathbf{x}_2\|_2}{|p(\lambda_1)|\,|x_1|} = \max_{\lambda \in \{\lambda_2, \ldots, \lambda_n\}} \frac{|p(\lambda)|}{|p(\lambda_1)|} \tan\theta(\mathbf{e}_1, \mathbf{x}).$$

This yields

$$\begin{aligned} \tan\theta\big(\mathbf{e}_1, \mathcal{K}_k(A, \mathbf{x})\big) &:= \min_{\mathbf{y} \in \mathcal{K}_k(A, \mathbf{x})} \tan\theta(\mathbf{e}_1, \mathbf{y}) \\ &= \min_{p \in \Pi_{k-1}} \max_{\lambda \in \{\lambda_2, \ldots, \lambda_n\}} \frac{|p(\lambda)|}{|p(\lambda_1)|} \tan\theta(\mathbf{e}_1, \mathbf{x}) \end{aligned} \tag{2.29}$$

For general diagonalizable A, we obtain the following result.

Theorem 2.25 *Let $A \in \mathbb{C}^{n \times n}$ be diagonalizable: $X^{-1}AX = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$. Then*

$$\tan\theta\big(\mathbf{v}_1, \mathcal{K}_k(A, \mathbf{x})\big) \leq C \min_{p \in \Pi_{k-1}} \max_{\lambda \in \{\lambda_2, \ldots, \lambda_n\}} \frac{|p(\lambda)|}{|p(\lambda_1)|} \tan\theta(\mathbf{v}_1, \mathbf{x}), \tag{2.30}$$

where $\mathbf{v_1}$ is an eigenvector belonging to λ_1 and $C \approx \kappa(X)$. For unitary X, $C = 1$.

How can we proceed from Theorem 2.25? To get meaningful bounds, we have to find a polynomial p that is large at λ_1 and not too small at all other eigenvalues. As a rule of thumb, this is possible if λ_1 is an exterior eigenvalue well separated from $\lambda_2, \ldots, \lambda_n$. Unfortunately, coming up with a rigorous convergence bound in the general case is a difficult complex approximation problem. In the following, we restrict ourselves to the case that all eigenvalues are real and

$$\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n.$$

We will make use of Chebyshev polynomials[11] to construct the polynomial p. The **Chebyshev polynomial** of the first kind of degree j is defined by

$$T_j(\mu) = \cos(j \arccos(\mu)), \qquad -1 \leq \mu \leq 1. \tag{2.31}$$

[11] A complete introduction into the beautiful theory of Chebyshev polynomials is beyond the scope of this lecture. See `http://en.wikipedia.org/wiki/Chebyshev_polynomials` for more information.

In particular, $T_0(\mu) = 1$ and $T_1(\mu) = \mu$. The trigonometric relation

$$\cos((j+1)\theta) + \cos((j-1)\theta) = 2\cos\theta\cos(j\theta)$$

implies the three-term recurrence relation

$$T_j(\mu) = 2\mu T_{j-1}(\mu) - T_{j-2}(\mu), \quad \text{for} \quad j = 2, 3, \ldots, \tag{2.32}$$

which shows that T_j is indeed a polynomial: $T_j \in \Pi_j$. For our purpose, it will be important to extend (2.31) outside the interval $[-1, 1]$, that is, $|\mu| > 1$. Using the properties of the hyperbolic cosine $\cosh\theta = \frac{1}{2}(e^\theta + e^{-\theta})$, we obtain that

$$T_j(\mu) = \cosh(j \operatorname{arccosh}(\mu)), \qquad \mu \geq 1$$

satisfies the recurrence (2.32) as well. Because T_j is an even/odd function for even/odd j, we also have

$$T_j(\mu) = (-1)^n \cosh(j \operatorname{arccosh}(-\mu)), \qquad \mu \leq -1.$$

Inserting $\operatorname{arccosh}(|\mu|) = \ln(|\mu| + \sqrt{\mu^2 - 1})$ into $\cosh(j\theta) = \frac{1}{2}((e^\theta)^j + (e^{-\theta})^j)$ gives the useful relation

$$|T_j(\mu)| = \frac{1}{2}\left(\left(|\mu| + \sqrt{\mu^2 - 1}\right)^j + \left(|\mu| + \sqrt{\mu^2 - 1}\right)^{-j}\right), \qquad |\mu| \geq 1.$$

In particular, the inequality

$$T_j(\mu) \geq \frac{1}{2}\left(|\mu| + \sqrt{\mu^2 - 1}\right)^j, \qquad |\mu| \geq 1, \tag{2.33}$$

holds and becomes increasingly tight for larger j. This shows that T_j grows very quickly outside the interval $[-1, 1]$, see also Figure 2.8.

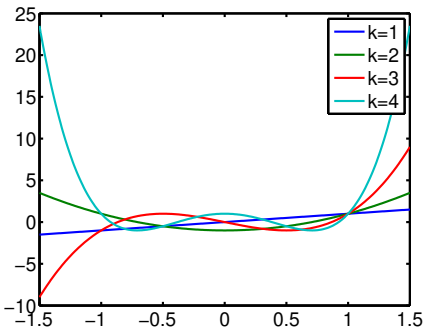


Figure 2.8. *Chebyshev polynomials $T_j(\lambda)$ for $j = 1, \ldots, 4$.*

We use a linear transformation to map the interval $[\lambda_2, \lambda_n]$ to $[-1, 1]$:

$$p(\lambda) := T_{k-1}\big((2\lambda - \lambda_2 - \lambda_n)/(\lambda_n - \lambda_2)\big) \in \Pi_{k-1}.$$

Because of (2.31), the magnitude of this polynomial is bounded by one on the interval $[\lambda_2, \lambda_n]$ and, because of (2.33), it grows quickly outside the interval. Inserting this polynomial into the result of Theorem 2.25 yields the following result.

Corollary 2.26 *Consider the setting of Theorem 2.25 and assume that all eigenvalues are real and ordered as follows:* $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$. *Then*

$$\tan\theta\big(\mathbf{v}_1, \mathcal{K}_k(A,\mathbf{x})\big) \leq \frac{C}{|T_{k-1}(\gamma_1)|}\,\tan\theta(\mathbf{v}_1,\mathbf{x}), \quad \gamma_1 = -1 - 2\frac{\lambda_2-\lambda_1}{\lambda_n-\lambda_2}.$$

Analogously, if $\lambda_1 \leq \cdots \leq \lambda_{n-1} < \lambda_n$ *then*

$$\tan\theta\big(\mathbf{v}_n, \mathcal{K}_k(A,\mathbf{x})\big) \leq \frac{C}{|T_{k-1}(\gamma_n)|}\,\tan\theta(\mathbf{v}_n,\mathbf{x}), \quad \gamma_n = 1 + 2\frac{\lambda_n-\lambda_{n-1}}{\lambda_{n-1}-\lambda_1}.$$

Similar bounds can also be derived for the second smallest eigenvalue, for the second largest eigenvalue, and so on. More details can be found in the book [Y. Saad. Numerical Methods for Large Eigenvalue Problems. Halstead Press, 1992], which is available from `http://www-users.cs.umn.edu/~saad/books.html`.

We now try to gain more insight into the bound of Corollary 2.26. Define

$$\mathsf{relgap} := \frac{\lambda_2-\lambda_1}{\lambda_n-\lambda_1}.$$

Then $|\gamma_1| = (1+\mathsf{relgap})/(1-\mathsf{relgap})$ and it follows that, using (2.33),

$$|T_{k-1}(\gamma_1)| \geq \frac{1}{2}\left(|\gamma_1| + \sqrt{\gamma_1^2-1}\right)^{k-1} = \frac{1}{2}\left(\frac{1+\sqrt{\mathsf{relgap}}}{1-\sqrt{\mathsf{relgap}}}\right)^{k-1} \gtrsim \frac{1}{2}\left(1+2\sqrt{\mathsf{relgap}}\right)^{k-1},$$

where the second inequality holds asymptotically for $\mathsf{relgap} \approx 0$. Hence,

$$\tan\theta\big(\mathbf{v}_1, \mathcal{K}_k(A,\mathbf{x})\big) \lesssim 1/(1+2\sqrt{\mathsf{relgap}})^{k-1}\tan\theta(\mathbf{v}_1,\mathbf{x}).$$

This nicely shows the influence of the **relative gap** $\frac{\lambda_2-\lambda_1}{\lambda_n-\lambda_1}$ on the convergence of the Arnoldi method.

Let us compare the convergence rate $|\lambda_{n-1}/\lambda_n|^k$ of the power method with the second bound of Corollary 2.26 for the matrix $A = \mathrm{diag}(1,2,\ldots,100)$.

k	$\frac{1}{\lvert T_{k-1}(\gamma_n)\rvert}$	$\lvert\lambda_{n-1}/\lambda_n\rvert^k$
2	0.9800	0.9801
3	0.9238	0.9703
4	0.8413	0.9606
5	0.7443	0.9510
10	0.3172	0.9044
15	0.1184	0.8601
20	0.0433	0.8179
30	0.0058	0.7397
40	0.0008	0.6690

Hence, the bound of Corollary 2.26 clearly reflects the much better convergence behavior of the Arnoldi method compared to the power method.

Further convergence considerations

Theorem 2.25 shows that the subspace $\mathcal{K}_k(A,\mathbf{x})$ contains increasingly good approximations to the eigenvector $\mathbf{v}_1$ but these bounds do not directly apply to the actual approximations returned by the Arnoldi method (Ritz vector and value). For the case that A is Hermitian, the following discussion shows that similar bounds hold for these approximations. As before, we assume that the eigenvalues of A are ordered increasingly: $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$.

Theorem 2.27 *Let $\mu_1^{(k)}$ be the smallest Ritz value extracted (via the Arnoldi method) from $\mathcal{K}_k(A,\mathbf{x})$. Then*

$$0 \le \mu_1^{(k)} - \lambda_1 \le (\lambda_n - \lambda_1)\left(\frac{\tan\theta(\mathbf{v}_1,\mathbf{x})}{|T_{k-1}(\gamma_1)|}\right)^2$$

Proof. The first inequality follows from eigenvalue interlacing. For the second inequality we note that

$$\mu_1^{(k)} = \min_{\mathbf{u}\in\mathcal{K}_k(A,\mathbf{x}),\mathbf{u}\neq\mathbf{0}} \rho_A(\mathbf{u}),$$

which implies

$$\begin{aligned}
\mu_1^{(k)} - \lambda_1 &= \min_{\mathbf{u}\in\mathcal{K}_k(A,\mathbf{x}),\mathbf{u}\neq\mathbf{0}} \rho_{A-\lambda_1 I}(\mathbf{u}) = \min_{q\in\Pi_{k-1},q\neq 0} \rho_{A-\lambda_1 I}(q(A)\mathbf{x}) \\
&= \min_{q\in\Pi_{k-1},q\neq 0} \frac{\langle (A-\lambda_1 I)q(A)\mathbf{x}, q(A)\mathbf{x}\rangle}{\langle q(A)\mathbf{x}, q(A)\mathbf{x}\rangle} = \min_{q\in\Pi_{k-1},q\neq 0} \frac{\sum_{j=2}^n (\lambda_j-\lambda_1)|\alpha_j q(\lambda_j)|^2}{\sum_{j=1}^n |\alpha_j q(\lambda_j)|^2} \\
&\le (\lambda_n-\lambda_1) \min_{q\in\Pi_{k-1},q\neq 0} \frac{\sum_{j=2}^n |\alpha_j q(\lambda_j)|^2}{\sum_{j=1}^n |\alpha_j q(\lambda_j)|^2} \le (\lambda_n-\lambda_1) \min_{q\in\Pi_{k-1},q\neq 0} \frac{\sum_{j=2}^n |\alpha_j q(\lambda_j)|^2}{|\alpha_1 q(\lambda_1)|^2} \\
&\le (\lambda_n-\lambda_1) \min_{q\in\Pi_{k-1},q\neq 0} \max_{\lambda\in\{\lambda_2,\ldots,\lambda_n} \frac{|q(\lambda)|^2}{|q(\lambda_1)|^2} \frac{\sum_{j=2}^n |\alpha_j|}{|\alpha_1|^2} \\
&\le (\lambda_n-\lambda_1) \frac{1}{|T_{k-1}(\lambda)|^2} \tan^2\theta(\mathbf{v}_1,\mathbf{x}),
\end{aligned}$$

where we expanded $\mathbf{x}$ into the orthonormal basis of eigenvectors $\mathbf{v}_1,\ldots,\mathbf{v}_n$: $\mathbf{x} = \sum_{j=1}^n \alpha_j \mathbf{v}_j$. □

For the Ritz vector belonging to $\mu_1^{(k)}$, a convergence result similar to Corollary 2.26 can be shown (see Section 6.3 in the book by Saad) with an additional constant $\sim 1/(\lambda_2-\lambda_1)$.

2.3.2 The implicitly restarted Arnoldi method

As j increases the cost of an iteration of the Arnoldi method increases: $O(nj)$ memory is needed for storing the vectors $\mathbf{u}_1,\ldots,\mathbf{u}_j$ and $O(nj)$ operations are needed to orthogonalize the new vector $\mathbf{w}$ against these vectors. For very large n, the increasing memory consumption is often a limiting factor that prevents from achieving the desired accuracy.

We will now discuss strategies to limit the growth of the Krylov subspace dimension. A straightforward strategy is to stop after, say, m iterations, compute a Ritz vector $\mathbf{v}$ belonging to the Ritz value of interest and apply the Krylov subspace method with the new starting vector $\mathbf{v}$ instead of $\mathbf{x}_1$. Theorem 2.25 provides a theoretical justification for this so called **explicit restarting** strategy: When $\mathbf{v}$ is a good approximation to $\mathbf{v}_1$, the last factor in (2.30) becomes small and fast(er) convergence can be expected. The main disadvantage of this strategy is that the dimension is always cut to 1; this leads to an unnecessary loss of information.

Instead, the following **implicit restarting** strategy is usually preferred. After m steps of the Arnoldi method, the Arnoldi decomposition

$$AU_m = U_m H_m + h_{m+1,m}\mathbf{u}_{m+1}\mathbf{e}_m^* \tag{2.34}$$

is produced. We then compute an *ordered* Schur decomposition (see Remarks after Theorem 1.11):

$$Q^* H_m Q = T = \begin{pmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{pmatrix}, \qquad \begin{aligned} \Lambda(T_{11}) &= \{\mu_1,\ldots,\mu_k\}, \\ \Lambda(T_{22}) &= \{\mu_{k+1},\ldots,\mu_m\}, \end{aligned}$$

where $\mu_1, \ldots, \mu_k$ are the *wanted* Ritz values and $\mu_{k+1}, \ldots, \mu_m$ are the *unwanted* Ritz values. For example, if the eventual goal is to compute the 10 smallest eigenvalues, one would choose $k = 10$ (or slightly larger) and let $\mu_1, \ldots, \mu_k$ correspond to the smallest Ritz values.

Applying the transformation matrix Q to (2.34) yields

$$A\widehat{U}_m = \widehat{U}_m T + \mathbf{u}_{m+1}\widehat{\mathbf{b}}_m^*, \quad \widehat{U}_m = U_m Q,\ \widehat{\mathbf{b}}_m = h_{m+1,m} Q^* \mathbf{e}_m.$$

Neglecting the parts associated with the unwanted Ritz values results in the truncated decomposition

$$A\widehat{U}_k = \widehat{U}_k T_{11} + \widehat{\mathbf{u}}_{k+1}\widehat{\mathbf{b}}_k^*, \tag{2.35}$$

where $\widehat{U}_k \in \mathbb{C}^{n\times k}$ contains the first k columns of $\widehat{U}_m$, and $\widehat{\mathbf{u}}_{k+1} = \mathbf{u}_{m+1}$.

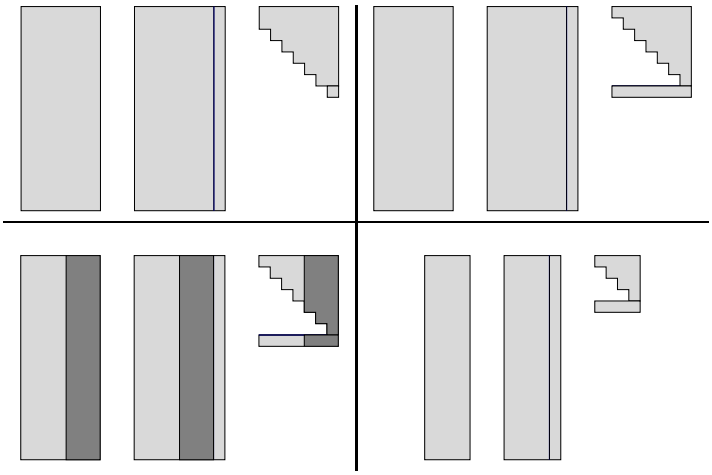


Figure 2.9. *Illustration of implicit restarting.*

After such an implicit restart, the truncated Arnoldi-like decomposition (also called Krylov-Schur decomposition) is expanded again to order m by applying the Arnoldi method. Essentially, this amounts to using Algorithm 2.23 with starting vector $\widehat{\mathbf{u}}_{k+1}$ with the small twist that we also orthogonalize against $\widehat{U}_k$. The whole procedure is repeated until all wanted eigenvalues have converged. To make quick progress, m should be chosen significantly larger than k; a typical choice is $m = 2k$.

Remark 2.28 *It turns out that implicit restarting is* mathematically *equivalent to explicit restarting with the updated starting vector* $\widehat{\mathbf{x}} = q(A)\mathbf{x}$ and *performing* k *steps of Arnoldi, where* $q(z) = (z - \mu_{k+1}) \cdots (z - \mu_m)$. *Implicit restarting has the advantage that the additional* k *steps of the Arnoldi method are not needed, saving* k *matrix-vector multiplications.*

The implicitly restarted Arnoldi method is implemented in the software package ARPACK, which is behind the MATLAB command `eigs`.

2.3.3 Lanczos method

For a Hermitian matrix A, the Arnoldi method simplifies significantly. In particular, this implies that the Hessenberg matrix $H_k = U_k^* A U_k$ is also Hermitian. A symmetric Hermitian matrix is

automatically tridiagonal:

$$H_k = \begin{pmatrix} \alpha_1 & \beta_1 & & \\ \beta_1 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ & & \beta_{k-1} & \alpha_k \end{pmatrix}, \quad \alpha_j \in \mathbb{R}, \quad \beta_j > 0. \tag{2.36}$$

If this structure is exploited in the Arnoldi method, the resulting method is called **Lanczos method**.

Algorithm 2.29 (Lanczos method)
Input: Hermitian matrix $A \in \mathbb{C}^{n\times n}$, starting vector $\mathbf{x} \neq 0$, $k \in \mathbb{N}$.
Output: Orthonormal basis $\{\mathbf{u}_1, \ldots, \mathbf{u}_k, \mathbf{u}_{k+1}\}$ of the Krylov subspace $\mathcal{K}_{k+1}(A, \mathbf{x})$.

$\mathbf{u}_1 \leftarrow \mathbf{x}/\|\mathbf{x}\|_2$
for $j = 1, 2, \ldots, k$ **do**
  $\mathbf{w} = A\mathbf{u}_j$
  $\alpha_j = \mathbf{u}_j^* \mathbf{w}$
  $\widetilde{\mathbf{u}}_{j+1} = \mathbf{w} - \mathbf{u}_j \alpha_j$
  if $j > 1$ **then**
    $\widetilde{\mathbf{u}}_{j+1} \leftarrow \widetilde{\mathbf{u}}_{j+1} - \mathbf{u}_{j-1}\beta_{j-1}$
  end if
  $\beta_j = \|\widetilde{\mathbf{u}}_{j+1}\|_2$
  $\mathbf{u}_{j+1} = \widetilde{\mathbf{u}}_{j+1}/\beta_j$
end for

The extraction of Ritz values and Ritz vectors is performed as for the Arnoldi method, see Section 2.3.1. The improved convergence of eigenvalues in the power method for the Hermitian case carries to the Lanczos method. For example, it can be shown for the smallest Ritz value $\mu^{(k)}$ in the kth iteration of Algorithm 2.29 that

$$0 \leq \mu^{(k)} - \lambda_1 \leq \frac{4(\lambda_n - \lambda_1)}{|T_{k-1}\big((2\lambda_1 - \lambda_2 - \lambda_n)/(\lambda_n - \lambda_2)\big)|^2} \tan^2 \theta(\mathbf{v}_1, \mathbf{x}).$$

Algorithm 2.29 only requires access to the last two vectors of the orthonormal basis in every loop. Hence, if only approximations to eigenvalues (and no eigenvectors) are required, we could run Algorithm 2.29 by storing only 2–3 vectors of length n, allowing to deal with extremely large n. Even if we need the full basis U_k for computing a Ritz vector $\mathbf{v} = U_k\mathbf{z}$, we could still store U_k externally (on hard disk/network storage) during Algorithm 2.29. Another alternative is to run Algorithm 2.29 twice: first for computing the eigenvalues and eigenvectors of H_k, and second for performing the matrix vector multiplications of the form $\mathbf{v} = U_k\mathbf{z}$ to construct the corresponding Ritz vectors.

Unfortunately, there is another serious obstacle to not storing U_k: The loss of orthogonality already discussed in Remark 2.24 for the Arnoldi method is even more severe for the Lanczos method. Numerical experiments and an analysis by Paige reveal that numerical orthogonality is completely destroyed as soon as a Ritz vector has converged to an eigenvector, see the MATLAB script `losslan.m` from the homepage. Again, this loss can be avoided by reorthogonalization (see

`nolosslan.m`), but this comes at a high price: All vectors in U_k need to be available, e.g., fetched from hard disk/network storage. A number of strategies have been developed to partly avoid this and reorthogonalize only selectively, see [J.K. Cullum, R.A. Willoughby. Lanczos algorithms for large symmetric eigenvalue computations: Theory. SIAM, 2002]. Although these strategies still play a role, modern implementations of the Lanczos method tend to prefer implicit restarting to avoid excessive memory requirements.

Chapter 3

Sparse direct solvers for solving large-scale linear systems

In the context of using the Lanczos method for computing the smallest eigenvalue of a PDE-eigenvalue problem, we have already encountered the need for applying A^{-1} or, equivalently, for solving a linear system of the form

$$A\mathbf{x} = \mathbf{b}. \tag{3.1}$$

Of course, there are many other applications for solving linear systems.

Applying Gaussian elimination or LU/Cholesky factorization to the solution of (3.1) requires $O(n^2)$ memory and $O(n^3)$ operations, which becomes quickly infeasible for an $n \times n$ matrix A arising from the discretization of a 2D or 3D PDE. There are mainly two classes of solvers for dealing with such large-scale linear systems. First, *sparse direct solvers* are based on suitable modifications of the LU/Cholesky factorization, trying to minimize the amount of required memory and operations. Such factorization-based methods are only possible for an explicitly given sparse matrix A (that is, most of its entries are zero) and their success depends on the pattern of sparsity in a complicated way. Nevertheless, these methods can be very successful and represent the methods of choice for dealing with finite element discretizations of 2D PDEs. On the other hand, *iterative solvers* can be applied to any matrix A that admits a fast matrix-vector product. In particular, the entries of A do not need to be given explicitly. However, especially for discretizations of PDEs, the convergence of these iterative solvers can become very slow. This effect can be avoided by the use of preconditioners, which is in fact absolutely essential in most applications. Preconditioning is an art and often requires additional insights into the problem at hand; we will learn more about this in the forthcoming chapters.

3.1 The Cholesky factorization

To simplify the discussion and avoid the need for pivoting to ensure numerical stability, we will assume that A is symmetric positive definite, see Section 3.4 for a discussion on the extension to the nonsymmetric case.

Definition 3.1 *A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is called positive definite if its (real) eigenvalues are all positive.*

An equivalent (but less direct) definition of positive definiteness is the requirement that $\mathbf{x}^\mathsf{T} A\mathbf{x} > 0$ for all nonzero vectors $\mathbf{x}$. For the special case $\mathbf{x} = \mathbf{e}_i$, we obtain $a_{ii} = \mathbf{e}_i^\mathsf{T} A\mathbf{e}_i > 0$.

Hence, all diagonal entries of A are positive. In particular, this allows to factor A as

$$A = \begin{pmatrix} a_{11} & \mathbf{a}_1^\mathsf{T} \\ \mathbf{a}_1 & A_{22} \end{pmatrix} = \underbrace{\begin{pmatrix} \sqrt{a_{11}} & \mathbf{0} \\ \frac{\mathbf{a}_1}{\sqrt{a_{11}}} & I_{n-1} \end{pmatrix}}_{=:L_1} \begin{pmatrix} 1 & \mathbf{0} \\ \mathbf{0} & A_{22} - \frac{\mathbf{a}_1\mathbf{a}_1^\mathsf{T}}{a_{11}} \end{pmatrix} \underbrace{\begin{pmatrix} \sqrt{a_{11}} & \frac{\mathbf{a}_1^\mathsf{T}}{\sqrt{a_{11}}} \\ \mathbf{0} & I_{n-1} \end{pmatrix}}_{=:L_1^\mathsf{T}}.$$

Repeating this factorization for the symmetric positive definite matrix $\widetilde{A}_{22} := A_{22} - \frac{\mathbf{a}_1\mathbf{a}_1^\mathsf{T}}{a_{11}}$ leads to Algorithm 3.2 and eventually results in a factorization of the form $A = LL^\mathsf{T}$.

Algorithm 3.2 (Cholesky factorization)
Input: Symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$.
Output: Factorization $A = LL^\mathsf{T}$ with lower triangular matrix L having positive diagonal entries.

1: **for** $k = 1, \ldots, n-1$ **do**
2: $\quad \ell_{kk} := \sqrt{a_{kk}}$
3: $\quad L\big(k+1:n, k\big) := \frac{1}{\ell_{kk}} A\big(k+1:n, k\big)$
4: $\quad A\big(k+1:n, k+1:n\big) \leftarrow A\big(k+1:n, k+1:n\big) - L\big(k+1:n, k\big) \cdot L\big(k+1:n, k\big)^\mathsf{T}$
5: **end for**

Theorem 3.3 *A matrix $A \in \mathbb{R}^{n\times n}$ is positive definite if and only if there is a lower triangular matrix $L \in \mathbb{R}^{n\times n}$ with positive diagonal entries such that*

$$A = LL^\mathsf{T}. \tag{3.2}$$

Proof. One direction of the statement follows from Algorithm 3.2. For the other direction, note that

$$\mathbf{x} \neq 0 \quad \Rightarrow \quad \mathbf{y} := L^\mathsf{T}\mathbf{x} \neq 0 \quad \Rightarrow \quad \mathbf{x}^\mathsf{T}LL^\mathsf{T}\mathbf{x} = \mathbf{y}^\mathsf{T}\mathbf{y} = \|\mathbf{y}\|_2^2 > 0,$$

where the first implication follows from the invertibility of L. This shows that LL^T is symmetric positive definite. ∎

The relation (3.2) is called Cholesky factorization of A. It can be shown that the Cholesky factor L is uniquely defined (if it is required to be lower triangular and have positive diagonal entries). The MATLAB function `chol` can be used to compute the Cholesky factorization.[12] Attempting to compute the Cholesky factorization also happens to be the best way to check whether a given symmetric matrix is positive definite.

3.2 Sparsity and the Cholesky factorization

3.2.1 Storage of sparse matrices

Before coming to algorithms for sparse Cholesky factorizations, we will briefly discuss how matrices are stored in memory. For matrices *without* any significant sparsity, there are essentially only two different formats.

[12] Note that `chol` computes the transpose of L, i.e., an upper triangular matrix R such that $A = R^\mathsf{T}R$. A lower triangular factor is computed directly when typing `L = chol(A,'lower')`.

Column major: Matrix *columns* are stored successively in memory (Fortran, MATLAB, OpenGL).

Row major: Matrix *rows* are stored successively in memory (C arrays, bitmaps, Python).

Example 3.4 (Row vs. column major)

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Column major:

A_arr	1	4	7	2	5	8	3	6	9

Row major:

A_arr	1	2	3	4	5	6	7	8	9

◇

The transfer of data between the memory hierarchy of modern computers (register, L1 cache, L2 cache, main memory, hard disk, network) typically consumes a significant portion of the execution time. It is therefore important to design an algorithm such that its memory access is aligned with the storage of the data.[13]

Example 3.5 The following two algorithms both execute $n(n-1)$ operations. The left algorithm proceeds in a column-wise fashion, while the right algorithm proceeds in a row-wise fashion.

MATLAB
```
A = randn(n);
for j = 1:n-1,
  A(:,j+1) = A(:,j+1) - A(:,j);
end
```

MATLAB
```
A = randn(n);
for i = 1:n-1,
  A(i+1,:) = A(i+1,:) - A(i,:);
end
```

It turns out that the row-wise variant requires roughly $3\times$ the execution time of the column-wise variant for sufficiently large n. The reason is that the inner loop accesses an entire row of the matrix, and the entries of a row are located far away from each other in memory when a column major format is used for storing A. ◇

A sparse matrix $A \in \mathbb{R}^{n\times n}$ contains many zeros, that is,

$$\texttt{nnz}(A) := \sharp\{(i,j) : a_{ij} \neq 0\} \ll n^2,$$

where `nnz` stands for *n*umber of *n*on*z*eros. The storage formats above are not suited for such a sparse matrix. There are a number of alternatives, which all have in common that they only store the nonzero entries of A and some information about the position of these nonzero entries. A popular format is the **Compressed Row Storage format** (short: CRS format). The information on the entries of $A \in \mathbb{R}^{m\times n}$ is stored in three arrays:

`real`	`val`	length $\texttt{nnz}(A)$	
`int`	`col_ind`	length $\texttt{nnz}(A)$	
`int`	`row_ptr`	length $n+1$	$\texttt{row\_ptr}[n+1] = \texttt{nnz}(A)+1$

The access to a matrix entry $a_{ij} \neq 0$, $1 \le i,j \le n$, proceeds as follows:

$$\texttt{val}[k] = a_{ij} \quad \Leftrightarrow \quad \begin{cases} \texttt{col\_ind}[k] = j, \\ \texttt{row\_ptr}[i] \le k < \texttt{row\_ptr}[i+1], \end{cases} \qquad 1 \le k \le \mathrm{nnz}(A).$$

Example 3.6 (CRS format)

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

[13] See also `http://en.wikipedia.org/wiki/Cache`.

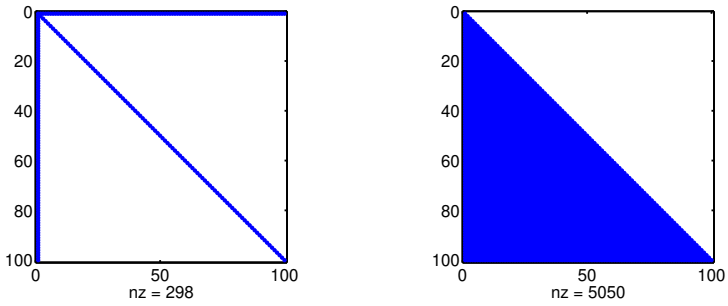


Figure 3.1. *An arrowhead matrix A with arrow pointing to top left corner (*left*) and its Cholesky factor (*right*).*

val	*10*	*-2*	*3*	*9*	*3*	*7*	*8*	*7*	*3 . . . 9*	*13*	*4*	*2*	*-1*
col_ind	*1*	*5*	*1*	*2*	*6*	*2*	*3*	*4*	*1 . . . 5*	*6*	*2*	*5*	*6*

row_ptr	*1*	*3*	*6*	*9*	*13*	*17*	*20*

◇

Basic commands for working with sparse matrices in MATLAB:

MATLAB
```
A = sparse(m,n); A = spalloc(m,n,nnz)
A = sparse(i,j,s,m,n);
A = spdiags(B,d,m,n); A = speye(n); A = spones(S);
```

When initializing a sparse matrix, it is important to take the particularities of the storage format into account. An unfortunate implementation can easily increase the execution time by a factor of 100. See exercise and `http://blogs.mathworks.com/loren/2007/03/01/creating-sparse-finite-element-matrices-in-matlab/`.

3.2.2 The envelope of a matrix

In general, the Cholesky factor will *not* inherit the sparsity of the matrix A.

Example 3.7 The following MATLAB code computes the Cholesky factorization of a so called arrowhead matrix.

MATLAB
```
n = 100;
A = (n+1)*speye(100); A(2:end,1) = 1; A(1,2:end) = 1;
L = chol(A,'lower'); spy(L)
```

While A is a sparse matrix, it can be seen from Figure 3.1 that its Cholesky factor is completely full. When we reorder A, this effect disappears.

MATLAB
```
A = A(end:-1:1,end:-1:1); L = chol(A,'lower'); spy(L)
```

Now, the Cholesky factor perfectly inherits the sparsity of A, see Figure 3.2. ◇

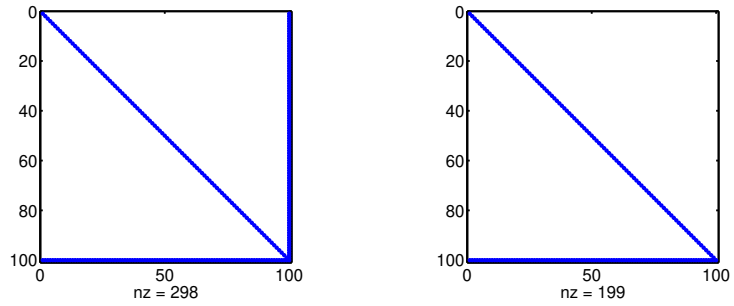


Figure 3.2. *An arrowhead matrix A with arrow pointing to bottom right corner (*left*) and its Cholesky factor (*right*).*

An important lesson from Example 3.7: The ordering of the matrix can have a tremendous impact on the sparsity of its Cholesky factor. The effect observed in Example 3.7 is well captured by the **envelope** of $A \in \mathbb{R}^{n\times n}$, which is defined as

$$\mathsf{env}(A) := \big\{(i,j) : \ J_i(A) \le j < i\big\}, \quad J_i(A) := \min\{j : a_{ij} \neq 0\}.$$

Note that the envelope of a matrix with the sparsity pattern displayed in Figure 3.1 contains *all* entries below the diagonal. In contrast, the envelope of a matrix with the sparsity pattern displayed in Figure 3.2 only contains the entries in the last row. The envelope of a matrix is inherited by its Cholesky factor.

Theorem 3.8 *Let $A \in \mathbb{R}^{n\times n}$ be symmetric positive definite and let $A = LL^\mathsf{T}$ be its Cholesky factorization. Then*

$$\ell_{ij} = 0 \quad \textit{for} \quad i > j \quad \textit{and} \quad (i,j) \notin \mathsf{env}(A).$$

Proof. This fact can be easily shown by induction. □

One can easily modify Algorithm 3.2 such that it only computes the entries of L within the envelope. This reduces the cost from $\frac{1}{3}n^3 + O(n^2)$ operations to

$$\frac{1}{2}\sum_{k=1}^{n} \omega_k(A)^2 + \text{low-order terms}$$

operations for the factorization, where $\omega_k(A) = \sharp\big\{j > k : (k,j) \in \mathsf{env}(A)\big\}$. The cost for solving a linear system given the factorization $A = LL^\mathsf{T}$ reduces to $2\sum_{k=1}^{n}(\omega_k(A)+1)$ operations.

3.2.3 Fill-in and basic concepts from graph theory

Given a Cholesky factorization $A = LL^\mathsf{T}$, indices (i,j) that satisfy $\ell_{ij} \neq 0$ but $a_{ij} = 0$ are called **fill-in**. From Theorem 3.8 we already know that fill-in can only take place within the envelope of A. In the following, we will discuss orderings of A that aim to reduce the envelope and the fill-in.

On the matrix level, a symmetry-preserving reordering of the columns and rows of A corresponds to $P^{\mathsf{T}}AP$ with a permutation matrix P. It is conceptually simpler to phrase such a reordering and its effect on the sparsity pattern in terms of the associated graph. Given a symmetric matrix $A \in \mathbb{R}^{n\times n}$, we define an *undirected graph* $G = (V, E)$ with vertices $V = \{v_1, \dots, v_n\}$ and

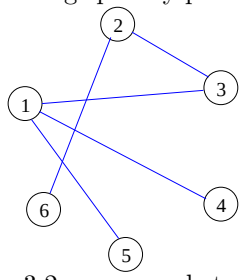
$$\{v_i, v_j\} \in E \quad \Leftrightarrow \quad a_{ij} \neq 0.$$

Then $P^T AP$ simply corresponds to a renumbering of the vertices.

We will now study the effect of Algorithm 3.2 on the graphs associated with the matrices generated during the factorization.

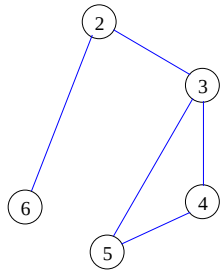
Example 3.9 Consider a matrix A with the following sparsity pattern:

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



In terms of the graph, the first step of Algorithm 3.2 corresponds to introducing edges between vertices that are indirectly connected by a path of length 2 via v_1. Afterwards, all edges attached to v_1 and the vertex v_1 itself are eliminated:

$$\left(\begin{array}{c|ccccc} \times & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & \times & \times & 0 & 0 & \times \\ 0 & \times & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & 0 & \times & \times & \times & 0 \\ 0 & \times & 0 & 0 & 0 & \times \end{array}\right)$$



The construction in the example above can be easily generalized. Let $G^{(1)}$ be the graph associated with A. More generally, let $G^{(k)}$ denote the graph associated with the submatrix $A(k:n, k:n)$ before the kth step of Algorithm 3.2 is executed. Then $G^{(k+1)}$ is obtained from $G^{(k)}$ by introducing edges between vertices that are indirectly connected by a path of length 2 via v_k, and then eliminating v_k along with its attached edges. The memory requirement for storing the Cholesky factor L is then given by

$$n + \sum_{k=1}^{n-1} \mathsf{deg}_{G^{(k)}}(v_k), \tag{3.3}$$

where the *degree* deg of a vertex is defined as the number of attached edges.

3.3 Ordering strategies

Algorithms for factorizing a sparse matrix typically consist of two phases. In the first phase, the sparsity pattern of A is analyzed and a suitable reordering aiming at a reduced fill-in is computed. (This is often referred to as the symbolic phase.) The second phase consists of a sophisticated implementation of Algorithm 3.2, restricting its storage and computations to the

predicted sparsity pattern of L. In the following, we discuss three quite different strategies for the first phase: **Reverse Cuthill-McKee**, **Approximate minimum degree**, and **nested dissection**.

3.3.1 Reverse Cuthill-McKee

The Cuthill-McKee (CM) and Reverse Cuthill-McKee (RCM) aim at minimizing the bandwidth of a sparse matrix in a heuristic way. To attain this goal, an ordering is produced in which neighboring vertices obtain numbers close together.

Algorithm 3.10 (Cuthill-McKee)
Input: Undirected graph (V, E) with $|V| = n$. Starting vertex $v \in V$.
Output: Numbering $\{1, \ldots, n\}$ of the vertices in V.
1: Initialize FIFO queue $F = (v)$.
2: $k = 1$.
3: **while** $F \neq \emptyset$ **do**
4: Remove first element v from F and assign number k to v.
5: Let W contain all unnumbered vertices in $\mathsf{Adj}(v)$.
6: Put all vertices from W into F in ascending order of degree.
7: $k = k + 1$
8: **end while**

In Algorithm 3.10, we use

$$\mathsf{Adj}(v) = \{w \in V : \ \{v, w\} \in E\}.$$

The choice of starting vertex v is important. Preferably, v should be a *peripheral vertex*, that is, there exists a path of maximal length starting from v. A (nearly) peripheral vertex can be found from repeated breadth first searches. If we let P_{CM} denote the permutation matrix associated with the numbering produced by Algorithm then the corresponding reordered matrix takes the form $P_{\mathsf{CM}}^{\mathsf{T}} A P_{\mathsf{CM}}$. It turns out that a better ordering is obtained when reversing the numbering obtained from Algorithm 3.10, leading to Algorithm 3.11. It can be shown [Liu'1975] that the corresponding reordered matrix $P_{\mathsf{RCM}}^{\mathsf{T}} A P_{\mathsf{RCM}}$ satisfies

$$\big|\mathsf{env}\big(P_{\mathsf{RCM}}^{\mathsf{T}} A P_{\mathsf{RCM}}\big)\big| \leq \big|\mathsf{env}\big(P_{\mathsf{CM}}^{\mathsf{T}} A P_{\mathsf{CM}}\big)\big|.$$

Algorithm 3.11 (Reverse Cuthill-McKee)
Input: Undirected graph (V, E) with $|V| = n$.
Output: Numbering $\{1, \ldots, n\}$ of the vertices in V.
1: Choose (nearly) peripheral starting vertex $v \in V$.
2: Determine the CM numbering by applying Algorithm 3.10.
3: Reverse the numbering.

Algorithm 3.11 is implemented in the MATLAB command `symrcm`.

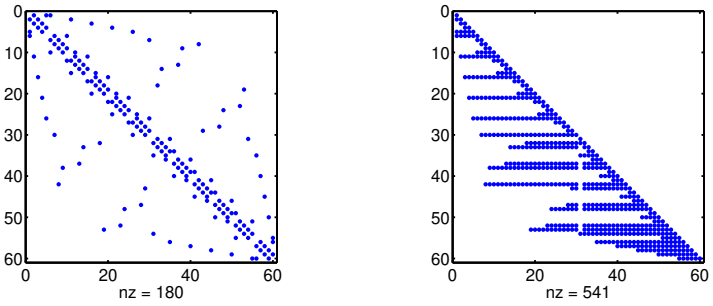


Figure 3.3. *Bucky ball matrix A (*left*) and its Cholesky factor (*right*).*

3.3.2 Approximate minimum degree ordering

While RCM is cheap and still quite popular, the obtained reordering is usually far from optimal. One problem with RCM is that it aims at minimizing the bandwidth, while the ultimate target is to minimize the fill-in and not the bandwidth. Both minimization problems are NP-hard. A greedy approach to minimizing the fill-in is presented in Algorithm 3.12. Note that $G^{(k)}$ refers to the elimination graphs introduced in Section 3.2.3, with the notable difference that the vertices to be eliminated are selected dynamically.

Algorithm 3.12 (Minimum Degree ordering)
Input: Undirected graph $G^{(1)} = \left(V^{(1)}, E^{(1)}\right)$ associated with symmetric matrix A.
Output: Numbering $\{1, \ldots, n\}$ of vertices.
1: **for** $k = 1, \ldots, n$ **do**
2: Select vertex v of $V^{(k)}$ with minimum degree and assign number k to v.
3: Obtain $G^{(k+1)}$ and $G^{(k)}$ by eliminating vertex v as described in Section 3.2.3.
4: **end for**

Algorithm 3.12 is quite costly, in particular due to the need to evaluate the degrees of all vertices in $G^{(k)}$. A cheaper, nearly equally effective alternative has been developed, called the Approximate Minimum Degree (AMD) ordering. A detailed description of AMD is beyond the scope of this lecture and we refer to the excellent book [Davis, T. A. Direct methods for sparse linear systems. SIAM, 2006].

Example 3.13 This example is taken from the documentation of `symamd`. The Bucky ball example corresponds to the connectivity graph of a soccer ball and can be obtained in MATLAB by calling `bucky`. The matrix and its Cholesky factor are shown in Figure 3.3.

MATLAB

```
B = bucky+4*speye(60);
r = symrcm(B); R = B(r,r);
p = symamd(B); S = B(p,p);
```

The results of the reordering with `symrcm` and `symamd` are shown in Figures 3.4 and 3.5, respectively.

Practically more important examples will be discussed in the exercises. ◇

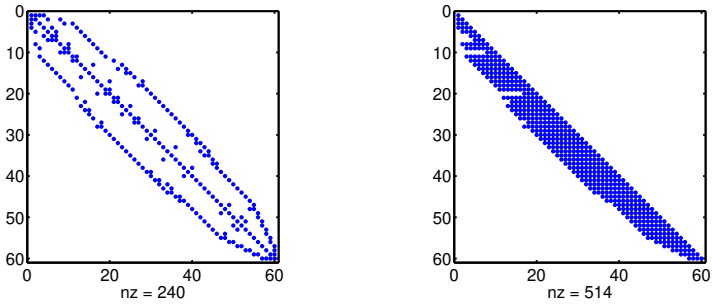


Figure 3.4. *Bucky ball matrix A (*left*) and its Cholesky factor (*right*) after reordering* `symrcm`.

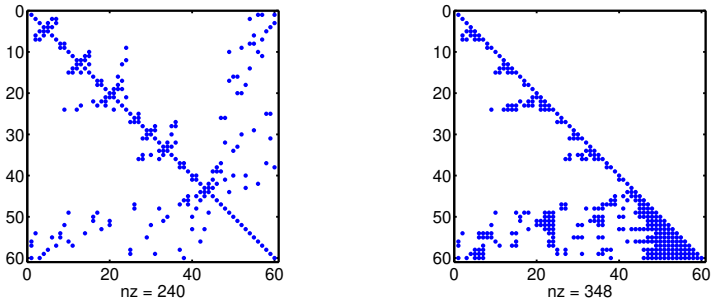


Figure 3.5. *Bucky ball matrix A (*left*) and its Cholesky factor (*right*) after reordering* `symamd`.

3.3.3 Nested dissection

A graph separator S for an undirected graph G partitions the set of vertices V into three disjoint sets

$$V = V_1 \cup V_2 \cup S,$$

such that no edges exist that connect vertices in V_1 with vertices in V_2. If we use a numbering such that the vertices in V_1 appear first, then the vertices in V_2, and then the vertices in S, this means that the matrix takes the form

$$A = \begin{pmatrix} A_{V_1,V_1} & 0 & A_{V_1,S} \\ 0 & A_{V_2,V_2} & A_{V_2,S} \\ A_{S,V_1} & A_{S,V_2} & A_{S,S} \end{pmatrix}. \tag{3.4}$$

By Theorem 3.8, the Cholesky factor will inherit the two zero off-diagonal blocks. The cardinality of the separator S should be small and preferably the cardinalities of V_1 and V_2 are balanced. This is an instance of a graph clustering problem; we might discuss this connection in more detail in the exercises.

Recursively applying the dissection (3.4) leads to Algorithm 3.14.

	Storage	Operations
2D	$O(N^2 \log N)$	$O(N^3)$
3D	$O(N^4)$	$O(N^6)$

Table 3.1. *Complexity of the Cholesky factorization of finite difference or finite element matrices, ordered by nested dissection, arising from regular $N \times N$ grid (2D) or a regular $N \times N \times N$ grid (3D).*

Algorithm 3.14 (Nested Dissection)
Input: Undirected graph $G = (V, E)$ associated with symmetric matrix A.
Output: Numbering $\{1, \ldots, n\}$ of vertices.
1: Select "good" separator S and corresponding partitioning $V = V_1 \cup V_2 \cup S$.
2: Determine a numbering of the vertices within V_1 by applying the algorithm recursively to the subgraph of G associated with V_1.
3: Determine a numbering of the vertices within V_2 by applying the algorithm recursively to the subgraph of G associated with V_2.
4: Put vertices in V_1 first (according to the numbering from Step 2), vertices in V_2 second (according to the numbering from Step 3), and vertices in S third.

The art in Algorithm 3.14 lies in selecting a "good" separator. This is relatively easy for problems where the underlying geometry is known, for example in FE discretizations of 2D or 3D PDEs. Then repeated geometric subdivision of the computational domain generally leads to a good choice of separators. For problems without an underlying geometry, graph clustering techniques (for example the ones from Section 1.6) can be used to determine good separators.

Table 3.1 summarizes the complexity of the Cholesky factorization when using nested dissection. The complete proofs for the 2D case can be found in Chapter 8 of [George/Liu. Computer solution of large sparse positive definite systems. Prentice-Hall, 1981], where it is also shown that the complexity cannot be improved by any other ordering. Here we only give a rough sketch for the 2D and 3D case.

2D case. Let $s(N)$ denote the memory required to store the Cholesky factor for a square grid of size $N \times N$, and assume that N is even. Subdividing the grid horizontally and vertically breaks it up into four square grids of size roughly $N/2 \times N/2$. In view of (3.4), we obtain

$$s(N) = 4s(N/2) + 2N^2 + O(N), \tag{3.5}$$

where, for simplification, $s(N/2)$ includes the memory needed for storing the Cholesky factor corresponding to A_{V_i,V_i} *as well as* $A_{V_i,S}$ / A_{S,V_i}. (In [George/Liu'1981] this is covered more rigourously by considering bordered matrices.) The memory needed for the $2N \times 2N$ matrix $A_{S,S}$ constitutes the term $2N^2 + O(N)$ in (3.5). Recurrence relations of the form (3.5) can be resolved using the so called Master theorem,[14] yielding

$$s(N) = O(N^2 \log N).$$

Similarly, the number of operations satisfies the relation $c(N) = 4c(N/2) + \frac{8}{3}N^3 + O(N^2)$. According to the Master theorem, this gives $c(N) = O(N^3)$.

[14] See `http://en.wikipedia.org/wiki/Master_theorem`.

3D case. Let $s(N)$ denote the memory required to store the Cholesky factor for a cubic grid of size $N \times N \times N$, and assume that N is even. Subdividing the grid along each coordinate direction breaks it up into eight cubes of size roughly $N/2 \times N/2$. Following the arguments above, we obtain

$$s(N) = 8s(N/2) + \frac{9}{2}N^4 + O(N^3),$$

which gives $s(N) = O(N^4)$. Note that the separators are now interfaces containing $3N^2$ nodes in total. The number of operations satisfies

$$c(N) = 8c(N/2) + 9N^6 + O(N^3),$$

which gives $c(N) = O(N^6)$.

3.4 Final remarks on sparse direct factorizations

In principle, the considerations above can be extended in a straightforward manner to the LU factorization of a nonsymmetric matrix A by considering the graph associated with $|A| + |A|^T$ instead of A. However, choosing the largest pivot element in the active column to ensure numerical stability during the LU factorization may dictate an ordering that leads to significant fill-in. In general, the two goals *optimal numerical stability* and *minimal fill-in* are incompatible and a compromise has to be made, e.g., by using non-optimal pivot elements.

By today, sparse direct solvers have matured to an extent that they represent the first choice for a wide range of sparse problems in scientific computing, in particular for finite difference/finite elements discretization of 2D PDEs and sometimes even 3D PDEs. Examples of software packages include MUMPS, PARDISO, SuperLU, and UMFPACK. Matlab's backslash and Python's `scipy.sparse.linalg` provide convenient access to sparse solvers based on SuperLU and UMFPACK. More flexibility is offered by PETSc, which can be accessed through `petsc4py` from within Python.

Chapter 4

Iterative solvers for solving large-scale linear systems

The sparse direct methods discussed in Chapter 3 can be quite effective, provided that the matrix under consideration is sparse and can be reordered to avoid excessive fill-in during the factorization. In certain applications, such as finite difference or finite element discretizations of 1D or 2D PDEs, sparse direct methods should always be the first choice and one should not look any further. However, there are many situations in which factorization-based methods are not applicable. We have already seen in the exercises that the fill-in for 3D PDEs is quite high and quickly reaches the limits of the available memory. An even more common situation is that the matrix is large but instead of sparsity there is some other structure that allows the efficient storage of the matrix. In these cases, the matrix will usually not be given explicitly but only represented implicitly in terms of a function that multiplies a vector with the matrix. One then has to resort to iterative methods for approximating the solution.

4.1 Krylov subspace methods

Given a linear system

$$A\mathbf{x} = \mathbf{b} \tag{4.1}$$

and starting vector $\mathbf{x}_0$ (e.g., $\mathbf{x}_0 = 0$), a number of popular iterative methods extract an approximation to the solution of (4.1) from the Krylov subspace $\mathcal{K}_k(A, \mathbf{r}_0)$. Here, $\mathbf{r}_0 := \mathbf{b} - A\mathbf{x}_0$ denotes the residual belonging to the starting vector, and we recall from Definition 2.20 that

$$\mathcal{K}_k(A, \mathbf{r}_0) = \operatorname{span}\big\{\mathbf{r}_0, A\mathbf{r}_0, \ldots, A^{k-1}\mathbf{r}_0\big\}.$$

There are many different ways of extracting such an approximation, each of which gives rise to a different method.

4.1.1 GMRES

The generalized minimum residual method (GMRES) selects the vector from $\mathcal{K}_k(A, \mathbf{r}_0)$ that minimizes the 2-norm of the residual. For this purpose, the Arnoldi method (see Algorithm 2.23) is used to generate an orthonormal basis $U_k = \big(\mathbf{u}_1, \ldots, \mathbf{u}_k\big)$ of $\mathcal{K}_k(A, \mathbf{r}_0)$, where $\mathbf{u}_1 = \mathbf{r}_0/\beta_0$ and $\beta_0 := \|\mathbf{r}_0\|_2$.

As explained in Section 2.3.1, the coefficients generated by the Arnoldi method can be arranged in an Arnoldi decomposition

$$AU_k = U_k H_k + h_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^{\mathsf{T}} = U_{k+1}\widetilde{H}_k \tag{4.2}$$

with an upper Hessenberg matrix H_k. For a vector $\widetilde{\mathbf{x}} = \mathbf{x}_0 + U_k\mathbf{y} \in \mathbf{x}_0 + \mathcal{K}_k(A, \mathbf{r}_0)$ with some $\mathbf{y} \in \mathbb{R}^k$, the 2-norm of the residual can be computed as follows:

$$\begin{aligned} \|\mathbf{b} - A\widetilde{\mathbf{x}}\|_2 &= \|\mathbf{r}_0 - AU_k\mathbf{y}\|_2 = \|\mathbf{r}_0 - U_{k+1}\widetilde{H}_k\mathbf{y}\|_2 \\ &= \|U_{k+1}\big(\beta_0\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\big)\|_2 = \|\beta_0\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\|_2. \end{aligned}$$

Hence, the residual norm is minimized when $\mathbf{y}$ solves the $(k+1) \times k$ linear least-squares problem

$$\min_{\mathbf{y}\in\mathbb{R}^k} \|\beta_0\mathbf{e}_1 - \widetilde{H}_k\mathbf{y}\|_2. \tag{4.3}$$

The upper Hessenberg structure of $\widetilde{H}_k$ can be exploited to solve this problem in $O(k^2)$ (instead of $O(k^3)$) operations, by computing its QR decomposition with a sequence of k Givens rotations. Moreover, updating techniques can be used to reuse the QR decomposition computed in the previous step for determining $\mathbf{y}_{k-1}$, see [GolV96] for more details. In effect, the vector $\mathbf{y}_k$ and the corresponding residual norm $\beta_k := \|\beta_0\mathbf{e}_1 + \widetilde{H}_k\mathbf{y}_k\|_2$ can be determined at essentially no cost.

If we let $\mathbf{y}_k$ denote the solution of (4.3) then the solution produced after k steps of GMRES is given by

$$\mathbf{x}_k := \mathbf{x}_0 + U_k\mathbf{y}_k.$$

In summary, we obtain Algorithm 4.1.

Algorithm 4.1 (GMRES)
Input: Matrix $A \in \mathbb{R}^{n\times n}$ and right-hand side $\mathbf{b} \in \mathbb{R}^n$, starting vector $\mathbf{x}_0$, $k \in \mathbb{N}$.
Output: Approximate solution $\mathbf{x}_k$ to $A\mathbf{x} = \mathbf{b}$.

1: Set $\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$.
2: Set $\beta_0 = \|\mathbf{r}_0\|_2$ and $\mathbf{u}_1 = \mathbf{r}_0/\beta_0$, $\quad U_1 = \mathbf{u}_1$.
3: **for** $j = 1, 2, \ldots, k$ **do**
4: Compute $\mathbf{w} = A\mathbf{u}_j$.
5: Compute $\mathbf{h}_j = U_j^*\mathbf{w}$.
6: Compute $\widetilde{\mathbf{u}}_{j+1} = \mathbf{w} - U_j\mathbf{h}_j$.
7: Set $h_{j+1,j} = \|\widetilde{\mathbf{u}}_{j+1}\|_2$.
8: Set $\mathbf{u}_{j+1} = \widetilde{\mathbf{u}}_{j+1}/h_{j+1,j}$.
9: Set $U_{j+1} = \big(U_j, \mathbf{u}_{j+1}\big)$.
10: Determine $\mathbf{y}_j$ by solving (4.3) and set $\beta_j = \|\beta_0\mathbf{e}_1 - \widetilde{H}_j\mathbf{y}_j\|_2$.
11: **end for**
12: Return $\mathbf{x}_k := \mathbf{x}_0 + U_k\mathbf{y}_k$.

A few remarks concerning practical aspects of Algorithm 2.23:

- Given a user-specified tolerance tol, the algorithm is usually stopped when $\beta_j < \mathsf{tol} \cdot \beta_0$.
- As discussed in Section 2.3.1, the approximate orthogonality of the basis U_k will get quickly lost due to the influence of roundoff error. This issue is less critical than for eigenvalue problems, as some of the nuisances implied by loss of orthogonality (such as appearance of ghost eigenvalues) are not relevant for linear systems. Nevertheless, loss of orthogonality may slow down convergence and it is advisable to perform the reorthogonalization strategy explained in Remark 2.24.

- If convergence is slow, a large value of k is needed and the $O(nk)$ memory required to store U_k will limit the applicability of Algorithm 4.1. This can be avoided by a simple restarting strategy. For this purpose, the value of k is kept relatively small ($k = 10$, $k = 20$ or $k = 50$ are typical values) and Algorithm 4.1 is run repeatedly, by supplying the solution $\mathbf{x}_k$ obtained from one run as the starting vector for the next run. This variant of GMRES is referred to as GMRES(k).

In the following, we analyze the convergence behavior of Algorithm 4.1. To obtain bounds for the convergence of the residual $\mathbf{r}_k := \mathbf{b} - A\mathbf{x}_k$ towards zero, let us first note that we have

$$\begin{aligned} \|\mathbf{r}_k\|_2 &= \min_{\mathbf{u}\in\mathcal{K}_k(A,\mathbf{r}_0)} \|\mathbf{b} - A(\mathbf{x}_0 + \mathbf{u})\|_2 = \min_{\mathbf{u}\in\mathcal{K}_k(A,\mathbf{r}_0)} \|\mathbf{r}_0 - A\mathbf{u}\|_2 \\ &= \min_{p\in\Pi_{k-1}} \|\mathbf{r}_0 - Ap(A)\mathbf{r}_0\|_2 = \min_{\substack{p\in\Pi_k \\ p(0)=1}} \|p(A)\mathbf{r}_0\|_2 \end{aligned} \tag{4.4}$$

by definition.

Theorem 4.2 *Let A be diagonalizable: $X^{-1}AX = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$. Let $\mathbf{r}_k$ denote the residual obtained after applying k steps of GMRES with starting vector $\mathbf{x}_0$. Then*

$$\frac{\|\mathbf{r}_k\|_2}{\|\mathbf{r}_0\|_2} \le \kappa(X) \min_{\substack{p\in\Pi_k \\ p(0)=1}} \max_{\lambda\in\{\lambda_1,\ldots,\lambda_n\}} |p(\lambda)|.$$

Proof. The result follows from relation (4.4) combined with

$$\|p(A)\mathbf{r}_0\|_2 = \|Xp(\Lambda)X^{-1}\mathbf{r}_0\|_2 \le \kappa(X)\|p(\Lambda)\|_2\,\|\mathbf{r}_0\|_2 = \kappa(X) \max_{\lambda\in\{\lambda_1,\ldots,\lambda_n\}} |p(\lambda)|\|\mathbf{r}_0\|_2.$$

□

To continue from Theorem 4.2, we face the same problem as in the convergence analysis of Krylov subspace methods for nonsymmetric eigenvalue problems: The polynomial optimization problem in the bound is difficult to solve in general and requires (usually unavailable) a priori information on the spectrum. Sensible statements can only be made for special cases. For example, if A is diagonalizable and has only $n_e \ll n$ different eigenvalues then GMRES converges in at most n_e iterations. A less trivial situation is given in the following theorem.

Theorem 4.3 *Let A be symmetric and assume that the eigenvalues are contained in $[-a, -b] \cup [c, d]$ with $a > b > 0$ and $d > c > 0$. Then*

$$\frac{\|\mathbf{r}_{2k}\|_2}{\|\mathbf{r}_0\|_2} \le 2\left(\frac{\sqrt{ad} - \sqrt{bc}}{\sqrt{ad} + \sqrt{bc}}\right)^k.$$

Proof. From Theorem 4.2, we obtain

$$\frac{\|\mathbf{r}_{2k}\|_2}{\|\mathbf{r}_0\|_2} \le \min_{\substack{p\in\Pi_{2k} \\ p(0)=1}} \max_{\lambda\in[-a,-b]\cup[c,d]} |p(\lambda)|.$$

The rest of the proof proceeds essentially by squaring the matrix A such that all its eigenvalues become positive, and construct the optimal polynomial on the positive interval containing the squared eigenvalues. We refer to Section 6.2.3 in [ElmSW05] for more details. □

If A is symmetric, GMRES simplifies considerably and can be implemented much more efficiently, requiring only access to the last two vectors in the basis (provided that no reorthogonalization is performed). Hence, the memory requirements reduce from $O(kn)$ down to $O(n)$, completely avoiding the need for restarting. This symmetric variant of GMRES is called **MINRES**. For completeness, Algorithm 4.4 contains the pseudo-code of MINRES. While we refer to Section 2.4 in [ElmSW05] for details, let us give some intuition how GMRES turns into Algorithm 4.4 when A is symmetric. Lines 4–9 implement the Lanczos method; see Algorithm 2.29; the reduction of Arnoldi to symmetric matrices. Lines 10–11 successively compute/update a QR factorization of the $(k+1) \times k$ matrix $\widetilde{H}_k$ needed to solve the least-squares problem (4.3). This QR factorization greatly simplifies because the top $k \times k$ matrix is tridiagonal (see (2.36)) and it is effected using Givens rotations, computed in line 11. The most important (and technical) part of MINRES is that it updates the sequence of the approximate solution in line 13, *without* needing access to the whole basis U_k. For this purpose, an auxiliary sequence of vectors $\mathbf{p}_j$ needs to be maintained; see line 12.

Algorithm 4.4 (MINRES method)
Input: Symmetric $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$. Starting vector $\mathbf{x}_0 \in \mathbb{R}^n$. $k \in \mathbb{N}$.
Output: Approximate solution $\mathbf{x}_k$ to $A\mathbf{x} = \mathbf{b}$.

1: $\widetilde{\mathbf{u}}_1 = \mathbf{b} - A\mathbf{x}_0$, $\beta_0 = \|\widetilde{\mathbf{u}}_1\|_2$, $\mathbf{u}_1 = \widetilde{\mathbf{u}}_1/\beta_0$
2: $\mathbf{p}_0 = \mathbf{p}_1 = \mathbf{0}$, $c_0 = c_1 = 1$, $s_0 = s_1 = 0$, $\eta = \beta_0$
3: **for** $j = 1, \ldots, k$ **do**
4: $\quad \mathbf{w} = A\mathbf{u}_j$, $\alpha_j = \mathbf{u}_j^* \mathbf{w}$
5: $\quad \widetilde{\mathbf{u}}_{j+1} \leftarrow \mathbf{w} - \mathbf{u}_j \alpha_j$
6: $\quad$ **if** $j > 1$ **then**
7: $\quad\quad \widetilde{\mathbf{u}}_{j+1} \leftarrow \widetilde{\mathbf{u}}_{j+1} - \mathbf{u}_{j-1}\beta_{j-1}$
8: $\quad$ **end if**
9: $\quad \beta_j = \|\widetilde{\mathbf{u}}_{j+1}\|_2$, $\mathbf{u}_{j+1} = \widetilde{\mathbf{u}}_{j+1}/\beta_j$
10: $\quad \delta_0 = c_j\alpha_j - c_{j-1}s_j\beta_{j-1}$, $\delta_1 = \sqrt{\delta_0^2 + \beta_j^2}$, $\delta_2 = s_j\alpha_j + c_{j-1}c_j\beta_{j-1}$, $\delta_3 = s_{j-1}\beta_{j-1}$
11: $\quad c_{j+1} = \delta_0/\delta_1$, $s_{j+1} = \beta_j/\delta_1$
12: $\quad \mathbf{p}_{j+1} = (\mathbf{u}_j - \mathbf{p}_{j-1}\delta_3 - \mathbf{p}_j\delta_2)/\delta_1$
13: $\quad \mathbf{x}_j = \mathbf{x}_{j-1} + \mathbf{p}_{j+1}c_{j+1}\eta$
14: $\quad \eta \leftarrow -s_{j+1}\eta$
15: **end for**

Assume that $a = \kappa(A), b = 1, c = 1, d = \kappa(A)$ in Theorem 4.3. Then the convergence rate of Theorem 4.3 amounts to

$$\frac{\kappa(A) - 1}{\kappa(A) + 1}, \tag{4.5}$$

clearly demonstrating that a large condition number can be expected to lead to slow convergence. This rate improves for positive definite matrices, but MINRES is *not* the method of choice in the positive definite case – the CG method discussed in the next is usually preferred.

4.1.2 CG

Instead of minimizing the residual, an alternative way to extract an approximate solution from $\mathbf{x}_k \in \mathbf{x}_0 + \mathcal{K}_k(A, \mathbf{r}_0)$ is to impose a Galerkin condition on the approximate solution:

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k \perp \mathcal{K}_k(A, \mathbf{r}_0). \tag{4.6}$$

Given an orthonormal basis U_k of $\mathcal{K}_k(A, \mathbf{r}_0)$ and setting $\mathbf{x}_k = \mathbf{x}_0 + U_k\mathbf{y}$, this condition is equivalent to

$$0 = U_k^\mathsf{T}\mathbf{r}_k = U_k^\mathsf{T}\big(\mathbf{b} - A(\mathbf{x}_0 + U_k\mathbf{y})\big) = \beta_0\mathbf{e}_1 - H_k\mathbf{y}.$$

Hence, we obtain $\mathbf{y} = \beta_0 H_k^{-1}\mathbf{e}_1$, provided that H_k is invertible.

For general nonsymmetric matrices, the method resulting from this approach is called FOM (full orthogonalization method). It has few advantages to offer. Compared to GMRES, it has the additional technical difficulty that H_k might not be invertible (or very ill-conditioned) for some k.

For a symmetric positive definite matrix A, however, the described approach leads to the CG (conjugate gradient) method. In this case, H_k is a $k \times k$ tridiagonal symmetric positive definite matrix. The CG method is given by Algorithm 4.5, which can be derived from (4.6) although this is by no means straightforward to see. We will skip the (beautiful) derivation of the method, and mention two properties of the vector sequences $\mathbf{p}_j$ and $\mathbf{r}_j$ generated in the course of the method:

- The vectors $\mathbf{r}_0, \dots, \mathbf{r}_{k-1}$ form an orthogonal basis of $\mathcal{K}_k(A, \mathbf{r}_0)$.
- The vectors $\mathbf{p}_1, \dots, \mathbf{p}_k$ form an A-orthogonal basis of $\mathcal{K}_k(A, \mathbf{r}_0)$, that is, $(\mathbf{p}_i, A\mathbf{p}_j) = 0$ for $i \neq j$. These are the "conjugate" search directions, from which the name "conjugate gradient" stems.

Algorithm 4.5 (CG method)
Input: Symmetric positive definite matrix $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$. Starting vector $\mathbf{x}_0 \in \mathbb{R}^n$. $k \in \mathbb{N}$.
Output: Approximate solution $\mathbf{x}_k$ to $A\mathbf{x} = \mathbf{b}$.

$\mathbf{p}_1 := \mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$
for $j = 1, \dots, k$ **do**
  $\alpha_j := (\mathbf{r}_{j-1}, \mathbf{r}_{j-1})/(\mathbf{p}_j, A\mathbf{p}_j)$
  $\mathbf{x}_j := \mathbf{x}_{j-1} + \alpha_j\mathbf{p}_j$
  $\mathbf{r}_j := \mathbf{r}_{j-1} - \alpha_j A\mathbf{p}_j$
  $\beta_{j+1} := (\mathbf{r}_j, \mathbf{r}_j)/(\mathbf{r}_{j-1}, \mathbf{r}_{j-1})$
  $\mathbf{p}_{j+1} := \mathbf{r}_j + \beta_{j+1}\mathbf{p}_j$
end for

To study the convergence of Algorithm 4.5, we go back to the original condition (4.6), which can also be written as

$$A(\mathbf{x} - \mathbf{x}_k) \perp \mathcal{K}_k(A, \mathbf{r}_0).$$

In other words, the error $\mathbf{x} - \mathbf{x}_k$ is A-orthogonal to $\mathcal{K}_k(A, \mathbf{r}_0)$. This implies

$$\|\mathbf{x} - \mathbf{x}_k\|_A = \min_{\mathbf{u}\in\mathcal{K}_k(A,\mathbf{r}_0)} \|\mathbf{x} - \mathbf{x}_0 - \mathbf{u}\|_A,$$

where $\|\mathbf{v}\|_A := \sqrt{\mathbf{v}^\mathsf{T}A\mathbf{v}}$. Hence,

$$\begin{aligned}\|\mathbf{x} - \mathbf{x}_k\|_A &= \min_{p\in\Pi_{k-1}} \|\mathbf{x} - \mathbf{x}_0 - p(A)\mathbf{r}_0\|_A = \min_{p\in\Pi_{k-1}} \|\mathbf{x} - \mathbf{x}_0 - Ap(A)(\mathbf{x} - \mathbf{x}_0)\|_A \\ &= \min_{\substack{p\in\Pi_k \\ p(0)=1}} \|p(A)(\mathbf{x} - \mathbf{x}_0)\|_A,\end{aligned}$$

which implies

$$\frac{\|\mathbf{x}-\mathbf{x}_k\|_A}{\|\mathbf{x}-\mathbf{x}_0\|_A} \leq \min_{\substack{p\in\Pi_k \\ p(0)=1}} \|p(A)\|_2 = \min_{\substack{p\in\Pi_k \\ p(0)=1}} \max_{\lambda\in\{\lambda_1,\ldots,\lambda_n\}} |p(\lambda)|. \tag{4.7}$$

There are several ways to proceed from (4.7). The most common is to replace the discrete collection $\{\lambda_1,\ldots,\lambda_n\}$ by the interval $[\lambda_{\min},\lambda_{\max}]$ containing all eigenvalues. This yields

$$\frac{\|\mathbf{x}-\mathbf{x}_k\|_A}{\|\mathbf{x}-\mathbf{x}_0\|_A} \leq \min_{\substack{p\in\Pi_k \\ p(0)=1}} \max_{\lambda\in[\lambda_{\min},\lambda_{\max}]} |p(\lambda)|, \tag{4.8}$$

where the optimization problem on the right can be solved explicitly: the optimal p takes the form of a scaled and shifted Chebyshev polynomial [ElmSW05].

Theorem 4.6 *Let $\mathbf{x}_k$ denote the approximate solution obtained after applying k steps of CG with starting vector $\mathbf{x}_0$. Then*

$$\frac{\|\mathbf{x}-\mathbf{x}_k\|_A}{\|\mathbf{x}-\mathbf{x}_0\|_A} \leq 2\left(\frac{\sqrt{\kappa(A)}-1}{\sqrt{\kappa(A)}+1}\right)^k.$$

What is the meaning of Theorem 4.6 in practice? For a matrix with a moderate condition number, say 10 or at most 100, the CG method will converge with satisfactory speed. Unfortunately, many applications give rise to ill-conditioned matrices. For the piecewise linear FE discretization of an elliptic PDE on a quasi-uniform mesh with mesh width h, we have $\kappa(A)\sim h^{-2}$. Hence, if we refine the mesh such that h gets divided by 2, the condition number grows with a factor of 4. In effect, it can be expected that the number of CG iterations needed to attain the same accuracy grows by a factor 2 according to Theorem 4.6.

4.2 Preconditioners

Slow convergence is by far the biggest obstacle when using the iterative methods discussed above. For a symmetric matrix A, Theorem 4.6 on the convergence of the CG method and the convergence rate (4.5) of the MINRES method indicate that a large condition number $\kappa(A)$ will usually cause slow convergence. For nonsymmetric matrices, the situation is more complicated but a large condition number is certainly not helpful.

The aim of preconditioning is to modify the linear system $A\mathbf{x}=\mathbf{b}$ such that convergence is accelerated. A **preconditioner** is any invertible matrix $M\in\mathbb{R}^{n\times n}$. To be useful in practice, it should be easy to apply M^{-1} (at least easier than applying A itself) and $M^{-1}A$ should be well-behaved in terms of convergence. There are three common ways of applying a preconditioner M to the linear system.

1. **Left preconditioning**:
$$M^{-1}A\mathbf{x}=M^{-1}\mathbf{b}.$$
2. **Right preconditioning**:
$$AM^{-1}\mathbf{u}=\mathbf{b},\quad \mathbf{x}=M^{-1}\mathbf{u}.$$
3. **Split preconditioning**: For a factorization $M=M_LM_R$,
$$M_L^{-1}AM_R^{-1}\mathbf{u}=M_L^{-1}\mathbf{b},\quad \mathbf{x}=M_R^{-1}\mathbf{u}$$

Using the fact that symmetric matrices can diagonalized simultaneously, it can be shown that the condition numbers are the same for right and left preconditioning:

$$\kappa(M^{-1}A) = \kappa(AM^{-1})$$

if M and A are symmetric. This property does not hold for nonsymmetric matrices and it does not hold for split preconditioning (even when M and A are symmetric). However, if $M_L^{-1}AM_R^{-1}$ is symmetric, then we have

$$\kappa(M_L^{-1}AM_R^{-1}) = \frac{\lambda_{\max}(M_L^{-1}AM_R^{-1})}{\lambda_{\min}(M_L^{-1}AM_R^{-1})} = \frac{\lambda_{\max}(M^{-1}A)}{\lambda_{\min}(M^{-1}A)}. \tag{4.9}$$

Also, $\kappa(M_L^{-1}AM_R^{-1}) = \|A\|_M\|A^{-1}\|_M$.

A good choice of M is an art. We will discuss simple, essentially black-box preconditioners as well as preconditioners that are tailored to discretizations of PDEs.

4.2.1 Preconditioned CG

Given a symmetric positive definite matrix A, how can the CG method profit from the availability of a symmetric positive definite preconditioner M? It is of course not possible to apply CG to $M^{-1}A$ or AM^{-1} as none of these matrices is symmetric. If we had a Cholesky decomposition $M = LL^{\mathsf{T}}$, we could apply CG to

$$L^{-1}AL^{-\mathsf{T}}\widehat{\mathbf{x}} = L^{-1}\mathbf{b}, \qquad \widehat{\mathbf{x}} = L^{\mathsf{T}}\mathbf{x}, \tag{4.10}$$

as $L^{-1}AL^{-\mathsf{T}}$ is symmetric positive definite. Let us denote the iterates of the CG applied to (4.10) as $\widehat{\mathbf{x}}_k$. The vectors $\widehat{\mathbf{x}}_k$ approximate the solution of the preconditioned systems (4.10), and hence the vectors $\mathbf{x}_k := L^{-\mathsf{T}}\widehat{\mathbf{x}}_k$ approximate the solution $\mathbf{x}$ of the original system. The three recursions determining the CG method are replaced by the equivalent recursions

$$\mathbf{x}_k = \mathbf{x}_{k-1} + \alpha_k\widehat{\mathbf{p}}_k, \quad \widehat{\mathbf{r}}_k = \widehat{\mathbf{r}}_{k-1} - \alpha_k A\widehat{\mathbf{p}}_k, \quad \widehat{\mathbf{p}}_{k+1} = M^{-1}\widehat{\mathbf{r}}_k + \beta_{k+1}\widehat{\mathbf{p}}_k \tag{4.11}$$

with

$$\widehat{\mathbf{p}}_k := L^{-\mathsf{T}}\mathbf{p}_k, \quad \widehat{\mathbf{r}}_k := L\mathbf{r}_k.$$

For the coefficients α_k and β_{k+1} of the CG method applied to (4.10), we obtain

$$\begin{aligned}
\alpha_k &= \frac{(\mathbf{r}_{k-1},\mathbf{r}_{k-1})}{(\mathbf{p}_k,\mathbf{p}_k)_{L^{-1}AL^{-\mathsf{T}}}} = \frac{(L^{-1}\widehat{\mathbf{r}}_{k-1},L^{-1}\widehat{\mathbf{r}}_{k-1})}{(\mathbf{p}_k,L^{-1}AL^{-\mathsf{T}}\mathbf{p}_k)} \\
&= \frac{(\widehat{\mathbf{r}}_{k-1},M^{-1}\widehat{\mathbf{r}}_{k-1})}{(L^{-\mathsf{T}}\mathbf{p}_k,AL^{-\mathsf{T}}\mathbf{p}_k)} = \frac{(\widehat{\mathbf{r}}_{k-1},M^{-1}\widehat{\mathbf{r}}_{k-1})}{(\widehat{\mathbf{p}}_k,\widehat{\mathbf{p}}_k)_A}, \\
\beta_{k+1} &= \frac{(\mathbf{r}_k,\mathbf{r}_k)}{(\mathbf{r}_{k-1},\mathbf{r}_{k-1})} = \frac{(\widehat{\mathbf{r}}_k,M^{-1}\widehat{\mathbf{r}}_k)}{(\widehat{\mathbf{r}}_{k-1},M^{-1}\widehat{\mathbf{r}}_{k-1})}.
\end{aligned}$$

This reformulation does not involve the Cholesky factor L anymore, but only refers to M^{-1}. In summary, we have obtained the following algorithm (where we removed the hats from $\mathbf{r}_k$ and $\mathbf{p}_k$ to increase readability).

Algorithm 4.7 Preconditioned CG
Input: SPD $A, M \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$. Starting vector $\mathbf{x}_0 \in \mathbb{R}^n$. $k \in \mathbb{N}$.
Output: Approximation $\mathbf{x}_k$ to the solution of $A\mathbf{x} = \mathbf{b}$.

$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0,\ \mathbf{p}_1 := M^{-1}\mathbf{r}_0$
for $j = 1, \ldots, k$ **do**
  $\alpha_k := (\mathbf{r}_{k-1}, M^{-1}\mathbf{r}_{k-1})/(\mathbf{p}_k, A\mathbf{p}_k)$
  $\mathbf{x}_k := \mathbf{x}_{k-1} + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_k := \mathbf{r}_{k-1} - \alpha_k A\mathbf{p}_k$
  $\beta_{k+1} := (\mathbf{r}_k, M^{-1}\mathbf{r}_k)/(\mathbf{r}_{k-1}, M^{-1}\mathbf{r}_{k-1})$
  $\mathbf{p}_{k+1} := M^{-1}\mathbf{r}_k + \beta_{k+1}\mathbf{p}_k$
end for

There are two extreme cases of Algorithm 4.7: $M = I$ leads to the standard CG method; $M = A$ leads to $\mathbf{r}_1 = \mathbf{0}$, that is, Algorithm 4.7 terminates successfully after one iteration. The use of Algorithm 4.7 is between these two extreme cases.

Corollary 4.8 *The approximation $\mathbf{x}_k$ obtained from Algorithm 4.7 satisfies*

$$\|\mathbf{x} - \mathbf{x}_k\|_A \leq 2c^k \, \|\mathbf{x} - \mathbf{x}_0\|_A$$

with

$$c = \frac{\sqrt{\tilde{\kappa}(M^{-1}A)} - 1}{\sqrt{\tilde{\kappa}(M^{-1}A)} + 1} < 1,$$

where $\tilde{\kappa}(M^{-1}A)$ denotes the ratio between the largest and smallest eigenvalues of $M^{-1}A$.

Proof. As discussed above, the preconditioned CG method can be seen as applying CG to

$$L^{-1}AL^{-\mathsf{T}}\widehat{\mathbf{x}} = L^{-1}\mathbf{b}.$$

Theorem 4.6 implies

$$\|\widehat{\mathbf{x}} - \widehat{\mathbf{x}}_k\|_{L^{-1}AL^{-\mathsf{T}}} \leq 2c^k \, \|\widehat{\mathbf{x}} - \widehat{\mathbf{x}}_0\|_{L^{-1}AL^{-\mathsf{T}}}$$

with

$$c = \frac{\sqrt{\kappa(L^{-1}AL^{-\mathsf{T}})} - 1}{\sqrt{\kappa(L^{-1}AL^{-\mathsf{T}})} + 1}.$$

Using $\kappa(L^{-1}AL^{-\mathsf{T}}) = \tilde{\kappa}(M^{-1}A)$, by (4.9), and

$$\|\widehat{\mathbf{x}} - \widehat{\mathbf{x}}_k\|_{L^{-1}AL^{-\mathsf{T}}} = (\widehat{\mathbf{x}} - \widehat{\mathbf{x}}_k, L^{-1}A\mathbf{x} - L^{-1}A\mathbf{x}_k) = (\mathbf{x} - \mathbf{x}_k, A\mathbf{x} - A\mathbf{x}_k) = \|\mathbf{x} - \mathbf{x}_k\|_A,$$

conclude the proof. □

The conclusion from Corollary 4.8 is that a preconditioner implying a moderate ratio between the largest and smallest eigenvalues of $M^{-1}A$ leads to fast convergence of CG.

4.2.2 Simple Preconditioners

There is a number of simple preconditioners one can always try first before attempting to use more advanced strategies. One example are preconditioners based on incomplete LU or Cholesky factorizations. ILU(0) and IC(0) are variants of the sparse LU and Cholesky factorization where

no fill-in outside the original sparsity pattern of A is allowed. In ILUT and ICT, elements smaller in magnitude than a local drop tolerance are dropped from the resulting factor (except for the diagonal element). Given such an incomplete factorizations $A = LU + E$, the preconditioner is set to $M = LU$. See the MATLAB commands `ilu` and `ichol` for more information.

Any matrix splitting of the form

$$A = M - N$$

leads to a fixed point equation $M\mathbf{x} = N\mathbf{x} + \mathbf{b}$ with the corresponding fixed point iteration

$$\mathbf{x}_{k+1} = M^{-1}(N\mathbf{x}_k + \mathbf{b}),$$

which converges to the solution for all $\mathbf{b}$ if and only $\rho(M^{-1}N) < 1$. We recall some examples of such stationary iterative methods:

Jacobi iteration $M = D$, where D is the diagonal part of A.

Gauss-Seidel iteration $M = L$, where L is the lower triangular part of A.

Richardson iteration $M = \omega^{-1}I$.

While the convergence of these methods is often too slow to be useful in practice, any of the matrices M is a potential candidate for a simple preconditioner. Especially the diagonal preconditioner is useful in applications that lead to a strictly diagonally dominant matrix A.

4.2.3 Multigrid

Usually none of the simple preconditioners mentioned above is particularly effective for linear systems arising from the discretization of PDEs. In particular, the convergence can be seen to deteriorate as the mesh width decreases. In contrast, multigrid methods (used either as iterative solvers or as preconditioners) often achieve convergence rates that are independent of the mesh width. As we will see below, this favorable behavior comes at the price of having to inject additional information, at least for geometric multigrid methods.

Richardson iteration for a model problem

Much insight can be gained from studying the model problem

$$\begin{aligned} -u''(x) &= f(x) \quad \text{for } x \in]0,1[, \\ u(0) = u(1) &= 0. \end{aligned}$$

Applying the second-order central finite difference on $n+2$ uniformly spaced grid points $x_0, x_1, \ldots, x_{n+1}$ leads to the linear system

$$A\mathbf{u} = \mathbf{f}, \tag{4.12}$$

with

$$A = \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad \mathbf{f} = h^2 \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{pmatrix}, \tag{4.13}$$

where $h = 1/(n+1)$.

The eigenvalues of A are given by

$$\lambda_k = 4\sin^2\frac{\theta_k}{2}, \quad \text{with} \quad \theta_k = \frac{k\pi}{n+1}, \tag{4.14}$$

and the corresponding eigenvectors are given by

$$\mathbf{w}_k = \begin{pmatrix} \sin\theta_k \\ \sin(2\theta_k) \\ \vdots \\ \sin(n\theta_k) \end{pmatrix}.$$

This can be verified using the trigonometric relation

$$\sin((j+1)\theta) + \sin((j-1)\theta) = 2\sin(j\theta)\cos\theta.$$

The most important property about the eigenvectors to note is that they become increasingly oscillatory as k increases, see Figure 4.1.

We now consider the Richardson iteration depending on a parameter ω:

$$\mathbf{u}_{j+1} = \mathbf{u}_j + \omega(\mathbf{f} - A\mathbf{u}_j) = \underbrace{(I - \omega A)}_{=:G_\omega}\mathbf{u}_j + \omega\mathbf{f}.$$

For symmetric positive definite A, this method converges if $0 < \omega < 2/\lambda_{\max}(A)$. Choosing ω optimally requires a good estimate $\lambda_{\min}(A)$, which is usually not at hand. Even a good estimate $\lambda_{\max}(A)$ may be considered too expensive to compute. Instead, one often uses an upper bound $\lambda_{\max}(A) \le \gamma$, which can be obtained – for example – by the Gershgorin theorem, and sets $\omega = 1/\gamma$.

For the matrix A in (4.13), the Gershgorin theorem yields $\gamma = 4$ and hence we set $\omega = 1/4$. We now apply the Richardson iteration to the linear system (4.12) and study the behavior of the error $\mathbf{d}_j := \mathbf{u} - \mathbf{u}_j$ which satisfies the relation

$$\mathbf{d}_j = G_\omega^j \mathbf{d}_0.$$

Expanding $\mathbf{d}_0$ in the eigenvectors of A,

$$\mathbf{d}_0 = \sum_{k=1}^{n} \xi_k \mathbf{w}_k$$

yields

$$\mathbf{d}_j = \sum_{k=1}^{n} \underbrace{\Big(1 - \frac{\lambda_k}{\gamma}\Big)}_{=:\eta_k}^{j} \xi_k \mathbf{w}_k. \tag{4.15}$$

Inserting the eigenvalues (4.14) for our model problem and $\gamma = 4$ gives

$$\eta_k = 1 - \sin^2\frac{\theta_k}{2}, \quad \theta_k = \frac{k\pi}{n+1}.$$

In particular, $\eta_k \approx 1$ for small k and large n. This is what causes the Richardson iteration to converge very slowly! On the other hand, η_k is significantly smaller than 1 for larger k. More specifically, for $k \ge (n+1)/2$ we have

$$\eta_k = \cos^2\frac{k\pi}{2(n+1)} \le \cos^2\frac{\pi}{4} = \frac{1}{2}.$$

This implies that these parts of $\mathbf{d}_j$ get rather quickly damped. Taking into account that the eigenvectors $\mathbf{w}_k$ are relatively smooth for small k and oscillatory for larger k, this means that the *oscillatory part gets damped*. This behavior of the Richardson iteration is often referred to as a **smoother**. It remains to deal with the smooth part. Multigrid methods do not attempt to eliminate the smooth part on the fine grid, but move down to a coarser grid, where the smooth eigenvectors become more oscillatory.

Remark 4.9 *A similar analysis can be performed for many other stationary iterations, including weighted Jacobi and Gauss-Seidel iterations. See [Saa03] for more details. In turn, these iterations can also be used as smoothers.*

Prolongation and restriction

Multigrid methods need to go back and forth between different grid levels. The transition between a fine mesh Ω_h and a coarse mesh Ω_H is performed by so called **prolongation** and **restriction** operators.

A **prolongation operator** maps vectors from the coarse to the fine mesh:

$$I_H^h : \Omega_H \to \Omega_h.$$

Piecewise linear interpolation is the simplest way of defining a sensible prolongation. To illustrate this for our model problem, let $x_0, x_1, \ldots, x_n, x_{n+1}$ denote the grid points of the fine mesh, where $h = 1/(n+1)$ and x_0, x_{n+1} are boundary points. Assuming that n is odd, the grid points of the coarse mesh with $H = 2h = 2/(n+1)$ are given by $x_0, x_2, x_4, \ldots, x_{n-1}, x_{n+1}$. Given data

$$v_0^{2h}, v_1^{2h}, \ldots, v_{(n+1)/2}^{2h}$$

on the coarse mesh, the entries of the prolongated data on the fine mesh is defined as:

$$\begin{aligned} v_0^h &= v_0^{2h}, \\ v_1^h &= \tfrac{1}{2}\big(v_0^{2h} + v_1^{2h}\big) \\ v_2^h &= v_1^{2h}, \\ v_3^h &= \tfrac{1}{2}\big(v_1^{2h} + v_2^{2h}\big) \\ &\vdots \\ v_{n+1}^h &= v_{(n+1)/2}^{2h}. \end{aligned}$$

More compactly, this can be written as

$$v_{2j}^h = v_j^{2h}, \qquad v_{2j+1}^h = \frac{1}{2}(v_j^{2h} + v_{j+1}^{2h})$$

for $j = 0, \ldots, (n+1)/2$. Note that the data at the boundary points is fixed to be zero ($v_0^{2h} = v_0^{2h} = 0$ and $v_{n+1}^h = v_{(n+1)/2}^{2h} = 0$) due to the zero Dirichlet boundary conditions of the model problem. Consequently, only the interior points need to be processed by the interpolation operator:

$$I_{2h}^h : \mathbf{v}^{2h} \mapsto \mathbf{v}^h, \quad \text{with} \quad \mathbf{v}^h = \begin{pmatrix} v_1^h \\ v_2^h \\ \vdots \\ v_n^h \end{pmatrix}, \quad \mathbf{v}^{2h} = \begin{pmatrix} v_1^{2h} \\ v_2^{2h} \\ \vdots \\ v_{(n+1)/2-1}^{2h} \end{pmatrix}.$$

The matrix representation of I_{2h}^h is given by

$$I_{2h}^h = \frac{1}{2}\begin{pmatrix} 1 & & & \\ 2 & & & \\ 1 & 1 & & \\ & 2 & & \\ & 1 & 1 & \\ & & \vdots & \\ & & 1 & 1 \\ & & & 2 \\ & & & 1 \end{pmatrix}$$

A **restriction operator** maps vectors from the fine to the coarse mesh:

$$I_h^H : \Omega_h \to \Omega_H.$$

If $\Omega_H \subset \Omega_h$, the simplest restriction $I_h^H \mathbf{v}^h$ would consist of returning the part of the data in $\mathbf{v}^h$ that corresponds to the coarse mesh. This is called injection operator. However, it is more common to perform averaging with the neighboring grid points. For our model problem, the restriction operator $I_h^{2h} : \mathbf{v}^h \mapsto \mathbf{v}^{2h}$ is then defined by

$$v_j^{2h} = \frac{1}{4}(v_{2j-1}^h + 2v_{2j}^h + v_{2j+1}^h).$$

This has the matrix representation

$$I_h^{2h} = \frac{1}{4}\begin{pmatrix} 1 & 2 & 1 & & & & & & \\ & & 1 & 2 & 1 & & & & \\ & & & & 1 & 2 & 1 & & \\ & & & & \cdots & \cdots & \cdots & & \\ & & & & & & 1 & 2 & 1 \end{pmatrix},$$

which gives the relation

$$I_{2h}^h = 2(I_h^{2h})^\mathsf{T}.$$

Two-grid cycle

We will now combine the ingredients above (smoother, prolongation, restriction) to construct a two-grid cycle that operates on two meshes, a fine mesh Ω_h and a coarse mesh Ω_h.

Let $\mathbf{u}_0^h$ denote the current approximation of the solution on Ω_h. Then we first apply the smoother (for example, the Richardson iteration) ν_1 times:

$$\mathbf{u}^h \leftarrow \text{smooth}^{\nu_1}(A_h, \mathbf{u}_0^h, \mathbf{f}^h).$$

If we solved the correction equation $A_h\Delta^h = r^h$ with the residual $\mathbf{r}^h := \mathbf{f}^h - A_h\mathbf{u}^h$ *exactly* then it can be easily seen that $\mathbf{u}^h + \Delta^h$ is the exact solution. The idea is now to solve this correction equation approximately, by Galerkin projection, on the coarse mesh. This corresponds to the solution of the linear system

$$A_H\Delta^H = \mathbf{r}^H, \tag{4.16}$$

where

$$A_H = I_h^H A_h I_H^h \tag{4.17}$$

and $\mathbf{r}^H = I_h^H \mathbf{r}^h$ is the coarsened residual. Once (4.16) is solved, we add the prolongated correction to $\mathbf{u}^h$:

$$\mathbf{u}^h \leftarrow \mathbf{u}^h + I_H^h \Delta^H.$$

Optionally, and to make things look more symmetric, ν_2 smoothing steps are performed at the end of the cycle. To summarize, we obtain Algorithm 4.10.

Algorithm 4.10 Two-grid cycle

$\mathbf{u}^h \leftarrow \text{smooth}^{\nu_1}(A_h, \mathbf{u}_0^h, \mathbf{f}^h)$	*% Presmoothing*
$\mathbf{r}^h \leftarrow \mathbf{f}^h - A_h \mathbf{u}^h$	
$\mathbf{r}^H = I_h^H \mathbf{r}^h$	*% Coarsening*
Solve $A_H \Delta^H = \mathbf{r}^H$	*% Coarse grid correction equation*
$\mathbf{u}^h \leftarrow \mathbf{u}^h + I_H^h \Delta^H$	*% Correction*
$\mathbf{u}^h \leftarrow \text{smooth}^{\nu_2}(A_h, \mathbf{u}^h, \mathbf{f}^h)$	*% Postsmoothing*

The whole two-grid cycle can be compactly written as

$$\mathbf{u}^h = S_h^{\nu_2} \left[S_h^{\nu_1} \mathbf{u}_0^h - I_H^h A_H^{-1} I_h^H A_h S_h^{\nu_1} \mathbf{u}_0^h \right] + \mathbf{g} = M_H^h \mathbf{u}_0^h + \mathbf{g},$$

where S_h denotes the smoothing operator, $\mathbf{g}$ is some vector, and

$$M_H^h = S_h^{\nu_2} \left[I - I_H^h A_H^{-1} I_h^H A_h \right] S_h^{\nu_1}.$$

The matrix inside the brackets,

$$T_h^H = I - I_H^h A_H^{-1} I_h^H A_h \tag{4.18}$$

is called the **coarse grid correction operator**. The following result is an essential property of the two-grid cycle.

Lemma 4.11 *Let A_h be symmetric positive definite and let A_H be defined as in* (4.17)*, such that $I_H^h A_H^{-1} I_h^H$ is symmetric. Then the coarse grid correction operator T_h^H in* (4.18) *is an orthogonal projector with respect to the inner product induced by A_h. Moreover, the range of T_h^H is A_h-orthogonal to the range of I_h^H.*

Proof. First, we verify that $I - T_h^H = I_H^h A_H^{-1} I_h^H A_h$ (and hence, also T_h^H) is a projector:

$$(I - T_h^H)^2 = I_H^h A_H^{-1} \underbrace{I_h^H A_h I_H^h}_{A_H} A_H^{-1} I_h^H A_h = I_H^h A_H^{-1} I_h^H A_h = I - T_h^H.$$

To prove the first part, it remains to show that $I_H^h A_H^{-1} I_h^H A_h$ is self-adjoint with respect to the A_h-inner product:

$$\begin{aligned} (I_H^h A_H^{-1} I_h^H A_h \mathbf{x}, \mathbf{y})_{A_h} &= (A_h I_H^h A_H^{-1} I_h^H A_h \mathbf{x}, \mathbf{y}) \\ &= (\mathbf{x}, A_h I_H^h A_H^{-1} I_h^H A_h \mathbf{y}) \\ &= (\mathbf{x}, I_H^h A_H^{-1} I_h^H A_h \mathbf{y})_{A_h}. \end{aligned}$$

The proof of the second part is a simple exercise. □

Based on Lemma 4.11, it can be shown that the repeated application of the two-grid cycle converges at a rate independent of h, under certain assumptions on the smoothing, prolongation and restriction operators. We refer to Section 13.5 in [Saa03] for details.

V-cycle and W-cycle

The practical usefulness of Algorithm 4.10 is limited, as the exact solution of the coarse grid correction equation (4.16) is still to costly. This can be avoided by recursion. We consider a mesh hierarchy with several levels. To approximate the solution of the coarse grid correction, we perform γ iterations of the multi-grid cycle. When we reach the coarsest mesh in the hierarchy, the coarse grid correction equation is solved exactly. The traversal through the mesh hierarchy for different values of γ is illustrated in Figure 4.2. Because of the shape of these diagrams, the case $\gamma = 1$ is called a V-cycle and the case $\gamma = 2$ is called a W-cycle.

4.2.4 Domain decomposition

To motivate domain decomposition techniques, let us consider the partial differential equation

$$\begin{array}{rcll} -\triangle u & = & f & \text{in } \Omega, \\ u & = & 0 & \text{on } \partial\Omega, \end{array} \tag{4.19}$$

for some domain $\Omega \subset \mathbb{R}^2$.

We choose two subdomains, that is, open sets $\Omega_1, \Omega_2 \subset \Omega$ such that $\overline{\Omega} = \overline{\Omega}_1 \cup \overline{\Omega}_2$. It is important to note that we allow for overlap; $\Omega_1 \cap \Omega_2$ is not necessarily empty. In the following, $\Gamma_1 = \partial\Omega_1 \backslash \partial\Omega$ and $\Gamma_2 = \partial\Omega_2 \backslash \partial\Omega$ denote the portions of the subdomain boundaries that are not included in the boundary of the whole domain. Often, Γ_1 and Γ_2 are called **artificial boundaries**.

The classical **alternating Schwarz method** for (4.19) iterates between restrictions of (4.19) to the two subdomains. More specifically, given the current iterate u_2^j for $u|_{\Omega_2}$, one first solves

$$\begin{array}{rcll} -\triangle u_1^{j+1} & = & f & \text{in } \Omega_1, \\ u_1^{j+1} & = & 0 & \text{on } \partial\Omega_1 \backslash \Gamma_1, \\ u_1^{j+1} & = & u_2^j & \text{on } \Gamma_1. \end{array} \tag{4.20}$$

The updated solution u_1^{j+1} is used to define the artificial boundary for the second subproblem:

$$\begin{array}{rcll} -\triangle u_2^{j+1} & = & f & \text{in } \Omega_2, \\ u_2^{j+1} & = & 0 & \text{on } \partial\Omega_2 \backslash \Gamma_2, \\ u_2^{j+1} & = & u_1^{j+1} & \text{on } \Gamma_2. \end{array} \tag{4.21}$$

In the next iteration, the solution u_2^{j+1} to this subproblem is then inserted into (4.20), and so on. Under suitable conditions on the regularity of the boundaries and right-hand sides, one can show that the resulting sequences u_1^j and u_2^j converge for $j \to \infty$ to the solution u of (4.19) restricted to the two subdomains Ω_1 and Ω_2, respectively.

Originally, this method proposed 1870 by Schwarz was intended for theoretical purposes. Much later it was discovered that this method gives rise to effective preconditioners for discretizations of (4.19).

One-level multiplicative Schwarz method for 2 subdomains

Let $A\mathbf{u} = \mathbf{f}$ be the linear system arising from a finite element (or finite difference) discretization of (4.19). We will now interpret the meaning of (4.20) and (4.21) for this linear system. The decomposition of the domain Ω induces an overlapping partitioning of the degrees of freedom in the finite element discretization. Let $\mathbf{u}_1$ and $\mathbf{u}_2$ denote the restriction of $\mathbf{u}$ to the degree of

freedoms associated with Ω_1 and Ω_2, respectively. We partition $\mathbf{u}_1$ and $\mathbf{u}_2$ further to distinguish the parts containing the artificial boundaries:

$$\mathbf{u}_1 = \begin{pmatrix} \mathbf{u}_{\Omega_1} \\ \mathbf{u}_{\Gamma_1} \end{pmatrix}, \quad \mathbf{u}_2 = \begin{pmatrix} \mathbf{u}_{\Omega_2} \\ \mathbf{u}_{\Gamma_2} \end{pmatrix}.$$

The discrete version of the iteration (4.20) then takes the form

$$A_{\Omega_1} \mathbf{u}_{\Omega_1}^{j+1} = \mathbf{f}_{\Omega_1} - A_{\Omega_1,\Gamma_1} \mathbf{u}_{\Gamma_1}^{j},$$

where A_{Ω_1} is the part of A with rows and columns restricted to Ω_1, and A_{Ω_1,Γ_1} is the part with rows restricted to Ω_1 and columns restricted to Γ_1. Since A_{Ω_1} is invertible, we can solve this linear system and obtain

$$\mathbf{u}_{\Omega_1}^{j+1} = \mathbf{u}_{\Omega_1}^{j} + A_{\Omega_1}^{-1}\big(\mathbf{f}_{\Omega_1} - A_{\Omega_1}\mathbf{u}_{\Omega_1}^{j} - A_{\Omega_1,\Gamma_1}\mathbf{u}_{\Gamma_1}^{j}\big).$$

To remove the explicit dependence on the artificial boundary we replace $A_{\Omega_1,\Gamma_1}\mathbf{u}_{\Gamma_1}^{j}$ by $A_{\Omega_1,\Omega\setminus\Omega_1}\mathbf{u}_{\Omega\setminus\Omega_1}^{j}$. These two terms are identical when there is no direct coupling between vertices on opposite sides of the artificial boundaries, which is usually the case for low-order finite element methods. After the replacement, we obtain

$$\begin{aligned} \mathbf{u}_{\Omega_1}^{j+1} &= \mathbf{u}_{\Omega_1}^{j} + A_{\Omega_1}^{-1}\big(\mathbf{f}_{\Omega_1} - A_{\Omega_1}\mathbf{u}_{\Omega_1}^{j} - A_{\Omega_1,\Omega\setminus\Omega_1}\mathbf{u}_{\Omega\setminus\Omega_1}^{j}\big) \\ &= \mathbf{u}_{\Omega_1}^{j} + A_{\Omega_1}^{-1}\big(\mathbf{f}_{\Omega_1} - A_{\Omega_1,\Omega}\mathbf{u}^{j}\big). \end{aligned}$$

Finally, this can be rewritten as

$$\mathbf{u}^{j+1/2} = \mathbf{u}^{j} + \begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix}\big(\mathbf{f} - A\mathbf{u}^{j}\big). \tag{4.22}$$

An analogous reasoning applied to (4.20) yields

$$\mathbf{u}^{j+1} = \mathbf{u}^{j+1/2} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix}\big(\mathbf{f} - A\mathbf{u}^{j+1/2}\big). \tag{4.23}$$

It is important to note that, unless $\Omega_1 \cap \Omega_2 = \emptyset$, the partitioning of these two matrices is not identical. The iteration defined by (4.22)–(4.23) is known as **multiplicative Schwarz method**. It can be seen as a generalized block Gauss-Seidel method.

We can write (4.22)–(4.23) in even more compact form in terms of the restriction operators R_1 and R_2:

$$\mathbf{u}_{\Omega_1} = R_1\mathbf{u} := \begin{pmatrix} I & 0 \end{pmatrix}\begin{pmatrix} \mathbf{u}_{\Omega_1} \\ \mathbf{u}_{\Omega\setminus\Omega_1} \end{pmatrix}, \quad \mathbf{u}_{\Omega_2} = R_2\mathbf{u} := \begin{pmatrix} I & 0 \end{pmatrix}\begin{pmatrix} \mathbf{u}_{\Omega_2} \\ \mathbf{u}_{\Omega\setminus\Omega_2} \end{pmatrix}$$

By setting

$$B_1 := R_1^{\mathsf{T}}(R_1 A R_1^{\mathsf{T}})^{-1} R_1, \quad B_2 := R_2^{\mathsf{T}}(R_2 A R_2^{\mathsf{T}})^{-1} R_2,$$

the iteration (4.22)–(4.23) becomes

$$\mathbf{u}^{j+1/2} = \mathbf{u}^{j} + B_1\big(\mathbf{f} - A\mathbf{u}^{j}\big), \quad \mathbf{u}^{j+1} = \mathbf{u}^{j+1/2} + B_2\big(\mathbf{f} - A\mathbf{u}^{j+1/2}\big). \tag{4.24}$$

This can be combined into a single step:

$$\mathbf{u}^{j+1} = \mathbf{u}^{j} + (B_1 + B_2 - B_2 A B_1)(\mathbf{f} - A\mathbf{u}^{j}).$$

While these compact forms are convenient for theoretical purposes, one would of course never form the matrices B_1 and B_2 (or even R_1 and R_2) explicitly!

The iteration (4.24) is applied as a preconditioner to a vector $\mathbf{r}$, by setting $\mathbf{f} = \mathbf{r}$ and $\mathbf{u}^{j} = 0$.

Multiplicative Schwarz method as preconditioner $\mathbf{p} \leftarrow M^{-1}\mathbf{r}$:

$$\mathbf{p} \leftarrow B_1\mathbf{r}, \qquad \mathbf{p} \leftarrow \mathbf{p} + B_2\big(\mathbf{r} - A\mathbf{p}\big).$$

Note that this preconditioner cannot be used in the preconditioned CG method; P is nonsymmetric even when A itself is symmetric. To avoid this, we can use a well-known trick for symmetrizing Gauss-Seidel methods and add a backward step to (4.24):

$$\begin{aligned} \mathbf{u}^{j+1/3} &= \mathbf{u}^{j} + B_1\big(\mathbf{f} - A\mathbf{u}^{j}\big), \\ \mathbf{u}^{j+2/3} &= \mathbf{u}^{j+1/3} + B_2\big(\mathbf{f} - A\mathbf{u}^{j+1/3}\big), \\ \mathbf{u}^{j+1} &= \mathbf{u}^{j+2/3} + B_1\big(\mathbf{f} - A\mathbf{u}^{j+2/3}\big), \end{aligned} \tag{4.25}$$

One-level multiplicative Schwarz method for $p > 2$ subdomains

We now consider the general case of p overlapping subdomains $\Omega_1, \ldots, \Omega_p$ of Ω with $\overline{\Omega} = \overline{\Omega}_1 \cup \cdots \cup \overline{\Omega}_p$. A suitable extension of (4.24) is given by

$$\begin{aligned} \mathbf{u}^{j+1/p} &= \mathbf{u}^{j} + B_1\big(\mathbf{f} - A\mathbf{u}^{j}\big), \\ \mathbf{u}^{j+2/p} &= \mathbf{u}^{j+1/p} + B_2\big(\mathbf{f} - A\mathbf{u}^{j+1/p}\big), \\ &\vdots \\ \mathbf{u}^{j+1} &= \mathbf{u}^{j+(p-1)/p} + B_p\big(\mathbf{f} - A\mathbf{u}^{j+(p-1)/p}\big), \end{aligned} \tag{4.26}$$

where $B_i := R_i^{\mathsf{T}}(R_i A R_i^{\mathsf{T}})^{-1} R_i$ with the restriction operator R_i for the subdomain Ω_i defined as above. The corresponding preconditioner takes the following form.

Multiplicative Schwarz method as preconditioner $\mathbf{p} \leftarrow M^{-1}\mathbf{r}$:
$\mathbf{p} \leftarrow B_1\mathbf{r}$.
For $i = 2, \ldots, p$ do $\mathbf{p} \leftarrow \mathbf{p} + B_i\big(\mathbf{r} - A\mathbf{p}\big)$.

It can be shown that the preconditioner satisfies

$$M^{-1} = \big(I - (I - B_p A)\cdots(I - B_1 A)\big)A^{-1}.$$

This explicit formula in terms of products is the reason for the name *multiplicative* Schwarz method.

As for $p = 2$, the above preconditioner is nonsymmetric and therefore not well suited for preconditioned CG. Again, this can be easily fixed by adding backward steps.

A major disadvantage of the current formulation is the sequential nature of the application of B_i for the different subdomains. To increase parallelism one can instead simultaneously process subdomains with no overlap. This procedure can be made systematic by coloring the subdomains; for more details see [Smith, B; Bjorstad, P.; Gropp, W. D. Domain decomposition. Parallel multilevel methods for elliptic partial differential equations. Cambridge University Press, 1996].

One-level additive Schwarz method

As mentioned above, the formulation (4.22)–(4.23) can be viewed as an extension of the block Gauss-Seidel method. The corresponding extension of the block Jacobi method takes the form

$$\mathbf{u}^{j+1} = \mathbf{u}^{j} + \left(\begin{pmatrix} A_{\Omega_1}^{-1} & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & A_{\Omega_2}^{-1} \end{pmatrix}\right)\big(\mathbf{f} - A\mathbf{u}^{j}\big).$$

With the notation introduced above, this so called **additive Schwarz method** can be compactly written as

$$\mathbf{u}^{j+1} = \mathbf{u}^j + (B_1 + B_2)\big(\mathbf{f} - A\mathbf{u}^j\big).$$

The generalization to arbitrarily many subdomains is given by

$$\mathbf{u}^{j+1} = \mathbf{u}^j + \sum_{i=1}^{p} B_i\big(\mathbf{f} - A\mathbf{u}^j\big). \tag{4.27}$$

Hence, the resulting preconditioner is simply given by

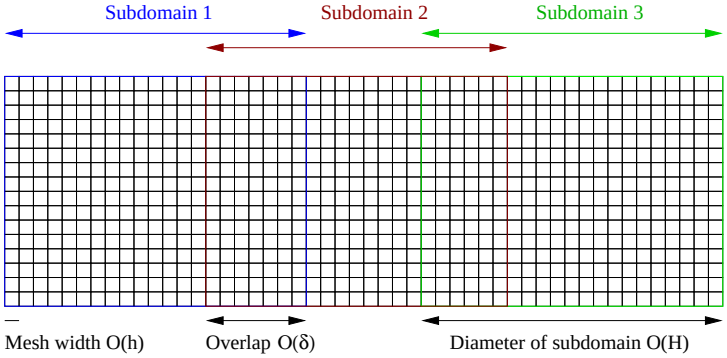
$$M^{-1} = \sum_{i=1}^{p} B_i.$$

The additive Schwarz method has a number of advantages compared to the multiplicative Schwarz method: It is easily parallelizable and the preconditioner is symmetric, provided that A is symmetric.

It is important to note that the iteration (4.27) itself will usually *not* converge. Still, when used as a preconditioner in a Krylov subspace method, it can be quite effective.

Summary of convergence behavior for one-level Schwarz methods

In the exercises, the convergence behavior of the Schwarz method, when used as a preconditioner in a Krylov subspace method, is studied experimentally. To summarize the findings, let us define the quantities indicated in the picture:



- Each cell of the mesh has diameter proportional to h.
- Each subdomain has diameter proportional to H.
- Subdomains overlap each other with a width proportional to δ.

We have the following typical observations:

- the number of iterations (to attain a constant accuracy) grows as $1/H$;
- if δ is kept proportional to H, the number of iterations is bounded independently of h and H/h;
- the number of iterations for the multiplicative Schwarz method (used as a preconditioner) is roughly half of the iterations for the additive Schwarz method;

- no overlap ($\delta = 0$) leads to poor convergence, but already a small value of δ can give acceptable convergence.

These observations can be made rigorous by a detailed analysis, which however requires to take properties of the PDE into account. Among these observations, probably the most important one is the deterioration of the convergence as the number of subdomains increases. An intuitive reason for this behavior is that the exchange of information from one boundary to the other boundary of the domain takes longer with increasingly many subdomains. To avoid this, some global exchange of information in the form of a correction step is needed. This will be addressed in the next section.

Two-level Schwarz methods

Any of the one-level Schwarz methods discussed above can replace the role of Jacobi or Gauss-Seidel as smoothers in a multi-grid method. Let $A_F \mathbf{u}_F = \mathbf{f}$ denote the discretization of the PDE on a fine mesh. As in multi-grid methods, we calculate the residual on the fine grid and perform a correction on the coarse grid. If R and R^T denote the restriction and prolongation operators between the two grids, and A_C denotes the discretization of the PDE on the coarse grid, then a simple **coarse grid correction** takes the forms

$$\mathbf{u}_F \leftarrow \mathbf{u}_F + R^\mathsf{T} A_C^{-1} R(\mathbf{f} - A_F \mathbf{u}_F).$$

This can be combined with a one-level additive or multiplicative Schwarz preconditioner leading to the following two-level preconditioners.

Two-level multiplicative Schwarz preconditioner $\mathbf{p} \leftarrow M^{-1}\mathbf{r}$:
$\mathbf{p} \leftarrow R^\mathsf{T} A_C^{-1} R\mathbf{r}$.
For $i = 1, \dots, p$ do $\mathbf{p} \leftarrow \mathbf{p} + B_i\big(\mathbf{r} - A_F\mathbf{p}\big)$.

Two-level additive Schwarz preconditioner $\mathbf{p} \leftarrow M^{-1}\mathbf{r}$:

$$\mathbf{p} \leftarrow \left(R^\mathsf{T} A_C^{-1} R + \sum_{i=1}^{p} B_i \right) \mathbf{r}.$$

Coarse grid correction significantly improves convergence and completely avoids convergence deterioration for a growing number of subdomains. We quote the following convergence result from the book by Smith, Bjorstad, and Gropp.

Theorem 4.12 *Consider a piecewise linear finite element discretization of* (4.19) *with mesh cell diameter proportional to h, subdomain diameter proportional to H, and overlap width proportional to $\delta > 0$. Then the condition number of the preconditioned system when using a two-level (additive or multiplicative) Schwarz preconditioner is bounded independently of h and H.*

In contrast to h and H, the condition number depends on the overlap width δ. Under suitable conditions, one can show that the condition number behaves like $O(1/\delta)$ as $\delta \to 0$.

4.2.5 Saddle Point Systems

All preconditioners discussed so far aim at reducing the condition number of the linear system. In view of Corollary 4.8 on the convergence of preconditioned CG and the convergence rate (4.5)

of MINRES, this is a natural goal. However, it is important to remember that these results only give *upper* bounds on the convergence. The actual convergence is governed by a polynomial approximation problem (4.4). Even for a symmetric matrix A, it is easy to arrive at a situation where the condition number tells very little about the actual convergence behavior. In the following, we discuss such a situation.

The **Stokes equations**

$$\begin{aligned} -\triangle u + \nabla p &= 0 \\ \nabla \cdot u &= 0 \end{aligned} \qquad \text{in } \Omega \tag{4.28}$$

are a fundamental model of viscous flow in a bounded two-dimensional or three-dimensional Ω. The variable u is a vector-valued function representing the velocity of the fluid, and p is a scalar function representing the pressure. We consider Dirchlet/Neumann boundary conditions of the form

$$u = w \text{ on } \partial\Omega_D, \qquad \frac{\partial u}{\partial n} - np = s \text{ on } \partial\Omega_N \tag{4.29}$$

for some partitioning $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N$. Note that n is the outward-pointing normal and w, s are prescribed vector-valued functions. The standard weak formulation of (4.28)–(4.29) uses $H^1_0(\Omega) = \{u \in H^1(\Omega)^d : \ u = 0 \text{ on } \partial\Omega_D\}$ as a test space for the velocity and $L_2(\Omega)$ as a test space for the pressure. The finite element discretization is then obtained from choosing finite element subspaces $X^h_0 \subset H^1_0(\Omega)$ $M^h \subset L_2(\Omega)$. Since these spaces are approximated independently, this is often called a **mixed finite element approximation**. Given a basis $\phi_1, \ldots, \phi_{n_u}$ of X^h_0 and a basis $\psi_1, \ldots, \psi_{n_p}$ of M^h, the resulting linear system takes the form

$$\begin{pmatrix} A & B^\mathsf{T} \\ B & 0 \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}. \tag{4.30}$$

The entries of the matrices $A \in \mathbb{R}^{n_u \times n_u}$ and $B \in \mathbb{R}^{n_p \times n_u}$ are defined by

$$a_{ij} = \int_\Omega \nabla\phi_i : \nabla\phi_j, \qquad b_{ij} = -\int_\Omega \phi_i \nabla \cdot \phi_j.$$

Systems of the form (4.30) are (informally) called **saddle point systems**, as linear systems of the same form occur in the characterization of saddle points in constrained optimization problems.

The most straightforward choices of finite element spaces lead to unstable formulations (in the sense of the inf-sup condition). This can be avoided by adding a regularization term, that is, a suitably chosen $n_p \times n_p$ matrix $-C$ in the $(2,2)$ block of (4.30). As such a nonzero $(2,2)$ block complicates the subsequent derivations, we will assume that a stable discretization is used. An example of such stable formulations includes the Taylor-Hood method, which uses $\mathbf{Q}_2$–$\mathbf{Q}_1$ on a quadrilateral mesh, that is, biquadratic finite elements for $\mathbf{u}$ and bilinear finite elements for $\mathbf{p}$. For a triangular mesh, a stable formulation is described in Section 5.3.3 of [ElmSW05].

Properties of the Galerkin matrix

The Galerkin matrix

$$K = \begin{pmatrix} A & B^\mathsf{T} \\ B & 0 \end{pmatrix}$$

from (4.30) is highly indefinite. This can be seen from the block Cholesky factorization

$$\begin{pmatrix} A & B^\mathsf{T} \\ B & 0 \end{pmatrix} = \begin{pmatrix} I & 0 \\ BA^{-1} & I \end{pmatrix} \begin{pmatrix} A & 0 \\ 0 & -BA^{-1}B^\mathsf{T} \end{pmatrix} \begin{pmatrix} I & A^{-1}B^\mathsf{T} \\ 0 & I \end{pmatrix}.$$

By Sylvester's law of inertia, the number of positive/negative/zero eigenvalues does not change under a congruence transformation. Moreover, the use of a stable formulation implies that $BA^{-1}B^\mathsf{T}$ is positive definite (or has $n_p - 1$ positive eigenvalues and one zero eigenvalue in the case of enclosed flow). Hence, K has n_u positive eigenvalues and n_p (or $n_p - 1$) negative eigenvalues.

To preserve the block structure of the saddle point system, a preconditoner should preferably be of the form

$$M = \begin{pmatrix} P & 0 \\ 0 & T \end{pmatrix}$$

for symmetric positive definite matrices $P \in \mathbb{R}^{n_u \times n_u}$ and $T \in \mathbb{R}^{n_p \times n_p}$. The preconditioned matrix has the eigenvalues of the matrix pencil

$$K - \lambda M = \begin{pmatrix} A & B^\mathsf{T} \\ B & 0 \end{pmatrix} - \lambda \begin{pmatrix} P & 0 \\ 0 & T \end{pmatrix}.$$

If $P = A$ then $\lambda = 1$ is an eigenvalue of multiplicity at least $n_u - n_p$. This can be seen from the fact that every vector $\mathbf{u}$ with $B\mathbf{u} = 0$ satisfies $(K - M)\left(\begin{smallmatrix} \mathbf{u} \\ \mathbf{0} \end{smallmatrix}\right) = 0$. If also $T = BA^{-1}B^\mathsf{T}$ then the remaining eigenvalues satisfy

$$(1 - \lambda)A\mathbf{u} = -B^\mathsf{T}\mathbf{p}, \qquad B\mathbf{u} = \lambda BA^{-1}B^\mathsf{T}\mathbf{p},$$

or after eliminating $\mathbf{u}$,

$$(\lambda^2 - \lambda - 1)BA^{-1}B^\mathsf{T}\mathbf{p} = 0.$$

If $BA^{-1}B^\mathsf{T}$ is positive definite, this implies that the remaining eigenvalues are $1/2 + \sqrt{5}/2$ and $1/2 - \sqrt{5}/2$, each with multiplicity n_p.

In summary, when using $P = A$ and $T = BA^{-1}B^\mathsf{T}$, the preconditioned matrix has only three distinct eigenvalues 1 and $1/2 \pm \sqrt{5}/2$. From 4.2, this implies that the MINRES applied to the preconditioned linear system **converges in at most 3 steps** to the exact solution!

Realistic preconditioners

The choice $P = A$ and $T = BA^{-1}B^\mathsf{T}$ for constructing the preconditioner is often unrealistic.

Most notably, the application of M would require the inversion of the dense matrix $BA^{-1}B^\mathsf{T}$ and it is not clear how this can be done efficiently. Fortunately, when using a stable finite element discretization, this matrix is spectrally equivalent to the (sparse) mass matrix $Q \in \mathbb{R}^{n_p \times n_p}$ for the pressure. This means there are constants $\gamma, \Gamma > 0$ (independent of h) such that

$$\gamma^2 \mathbf{p}^\mathsf{T} Q \mathbf{p} \le \mathbf{p}^\mathsf{T} BA^{-1}B^\mathsf{T}\mathbf{p} \le \Gamma^2 \mathbf{p}^\mathsf{T} Q \mathbf{p}$$

for all $\mathbf{p}$. In particular, this shows that every eigenvalue μ of $BA^{-1}B^\mathsf{T} - \lambda Q$ satisfies $\gamma^2 \le \mu \le \Gamma^2$. Hence, when using the preconditioner

$$M = \begin{pmatrix} A & 0 \\ 0 & Q \end{pmatrix},$$

the eigenvalues of the preconditioned matrix are in the intervals

$$\left[\frac{1 - \sqrt{4 + \Gamma^2}}{2}, \frac{1 - \sqrt{4 + \gamma^2}}{2}\right] \cup \{1\} \cup \left[\frac{1 + \sqrt{4 + \gamma^2}}{2}, \frac{1 + \sqrt{4 + \Gamma^2}}{2}\right].$$

This implies that the condition number of $M^{-1}K$ remains bounded as $h \to 0$ and the number of MINRES iterations will not grow as h decreases.

By a similar analysis, it can be shown that also A can be replaced by a spectrally equivalent matrix P. Examples include V-cycle or W-cycle multigrid preconditioners, which are well suited as A is the discretization of the Laplace operator.

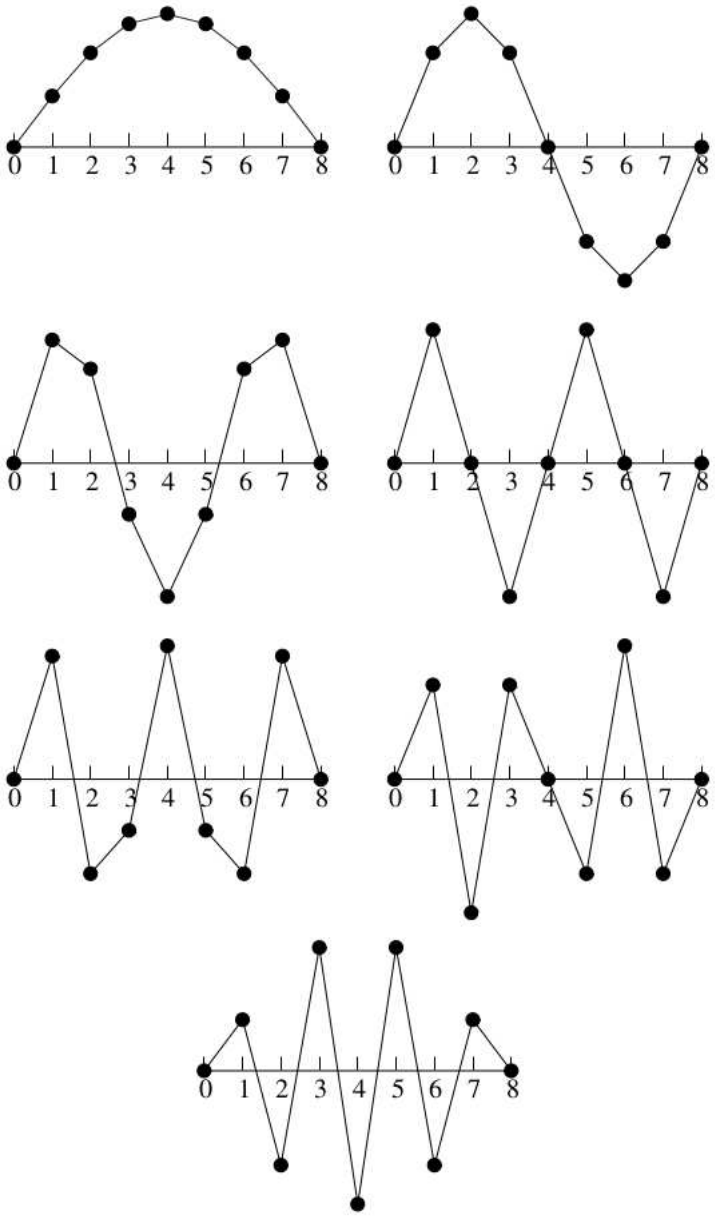


Figure 4.1. *The seven eigenfunctions of the discretized one-dimensional Laplace operator when $n = 7$. Figure taken from [Saa03].*

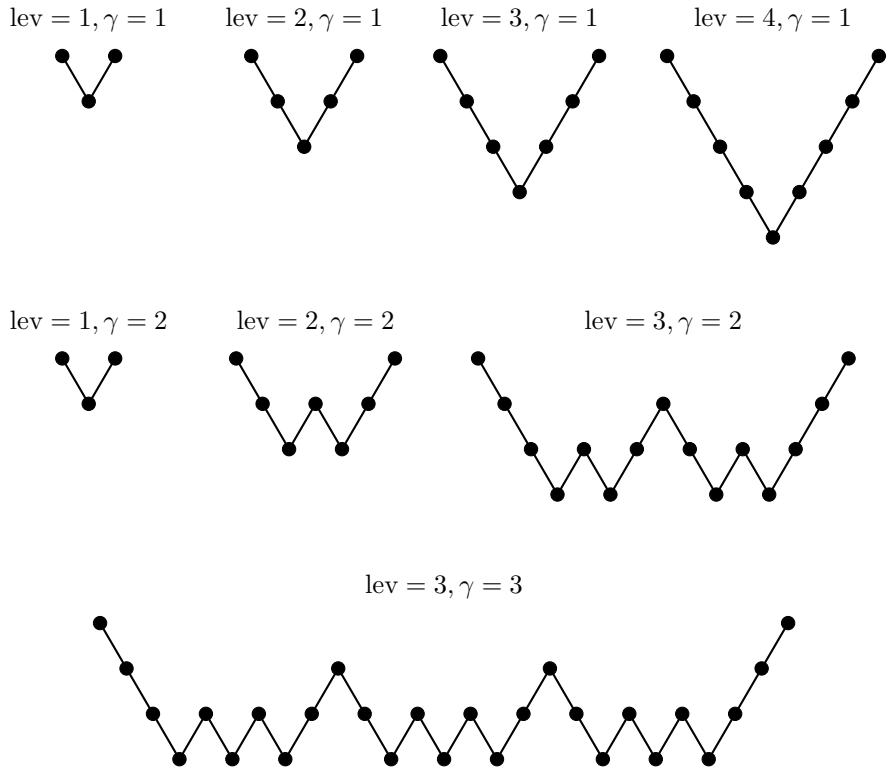


Figure 4.2. *Representations of various V-cycles and W-cycles.*

Chapter 5

Preconditioned solvers for eigenvalue problems

With our knowledge on iterative methods for linear systems and preconditioners, it is time to go back to eigenvalue problems. More specifically, we consider the efficient computation of small eigenvalues for a symmetric positive definite matrix A. When A arises from the discretization of an elliptic PDE eigenvalue problem, we have observed in the exercises of Chapter 2 that it is more advisable to apply Lanczos to A^{-1} instead of A. The reason for this advice is that the relative gap between the smallest and second smallest eigenvalue of A (compared to the width of the spectrum) converges to 0 as $h \to 0$. As this gap features prominently in the convergence bound (see Corollary 2.26), the number of Lanczos iterations to attain a certain accuracy increases rapidely as $h \to 0$. In contrast, the relative gap between the largest and second largest eigenvalue of A^{-1} stays bounded from below as $h \to 0$. Hence, the number of Lanczos iterations remains constant as $h \to 0$.

When a sparse Cholesky factorization of A can be (cheaply) computed, applying Lanczos to A^{-1} is the approach of choice for computing the smallest eigenvalue(s) of A. In the following, we consider a situation where only a preconditioner M (for example, a multigrid preconditioner) is available. Clearly, it does not make sense to apply the Lanczos method to M^{-1}, as the eigenvalues of M have little in common with the eigenvalues of A. Instead, we could replace the action of A^{-1} by running the CG method with the preconditioner M in every Lanczos iteration. However, such an approach would require a very small stopping tolerance for CG to faithfully reproduce the action A^{-1}. This chapter discusses two algorithms that use M in a more direct fashion and are thus much more favourable.

5.1 Preconditioned Inverse Iteration (PINVIT)

The basic idea of PINVIT is to inject the exact information on A via the residual

$$\mathbf{r}^{(k)} = A\mathbf{x}^{(k)} - \mu_k \mathbf{x}^{(k)},$$

where $\mathbf{x}^{(k)}$ with $\|\mathbf{x}^{(k)}\|_2 = 1$ is the previous iterate, and $\mu_k = \rho_A(\mathbf{x}^{(k)}) = \left(\mathbf{x}^{(k)}\right)^\mathsf{T} A\mathbf{x}^{(k)}$ denotes the Rayleight quotient. The next step of inverse iteration would be defined via $A^{-1}\mathbf{x}^{(k)}$ (or a scalar multiple thereof), which can be rewritten as

$$\mu_k A^{-1}\mathbf{x}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^{(k)} + \mu_k A^{-1}\mathbf{x}^{(k)} = \mathbf{x}^{(k)} - A^{-1}\mathbf{r}^{(k)}.$$

In PINVIT, the action of A^{-1} is replaced by M^{-1}:

$$\widetilde{\mathbf{x}}^{(k+1)} = \mathbf{x}^{(k)} - M^{-1}\mathbf{r}^{(k)}, \qquad \mathbf{x}^{(k+1)} = \widetilde{\mathbf{x}}^{(k+1)}/\|\widetilde{\mathbf{x}}^{(k+1)}\|_2. \tag{5.1}$$

To gain insights into the convergence of PINVIT, we note that (5.1) can be viewed as a perturbed inverse iteration:

$$\begin{aligned}\widetilde{\mathbf{x}}^{(k+1)} &= \mathbf{x}^{(k)} - M^{-1}A\mathbf{x}^{(k)} + \mu_k M^{-1}\mathbf{x}^{(k)} \\ &= \mu_k A^{-1}\mathbf{x}^{(k)} + (I - M^{-1}A)(\mathbf{x}^{(k)} - \mu_k A^{-1}\mathbf{x}^{(k)}).\end{aligned}$$

The smaller the term $I - M^{-1}A$, the closer PINVIT approaches inverse iteration. We will measure the size of $I - M^{-1}A$ with the matrix norm induced by $\|\mathbf{x}\|_A = \sqrt{\mathbf{x}^\mathsf{T} A\mathbf{x}}$:

$$\|B\|_A = \sup_{\substack{\mathbf{x}\in\mathbb{R}^n \\ \mathbf{x}\neq 0}} \frac{\|B\mathbf{x}\|_A}{\|\mathbf{x}\|_A} = \left\|A^{1/2}BA^{-1/2}\right\|_2.$$

Theorem 5.1 (Neymeyr) *Let A be symmetric positive definite with eigenvalues $\lambda_1 < \lambda_2 \leq \cdots \leq \lambda_n$. If $\|I - M^{-1}A\|_A \leq \gamma < 1$ then the sequence $\mu_k = \rho_A(\mathbf{x}^{(k)})$ defined by PINVIT (5.1) decreases monotonically and $(\mu_k, \mathbf{x}^{(k)})$ converges to an eigenpair of A (usually to the smallest). Provided that $\mu_k \to \lambda_1$, and $\mu_k < \lambda_2$, we have*

$$\frac{\mu_{k+1} - \lambda_1}{\mu_k - \lambda_1} \leq \frac{1 - (1-\gamma^2)\frac{\lambda_2 - \mu_k}{\lambda_2}}{1 + (1-\gamma)^2\frac{(\mu_k - \lambda_1)(\lambda_2 - \mu_k)}{\lambda_1\lambda_2}} < 1. \tag{5.2}$$

Inserting $\mu_k = \lambda_1$ into (5.2), the result of Theorem 5.1 shows that the convergence rate of PINVIT is asymptotically bounded by

$$\frac{(1-\gamma^2)\lambda_1 + \gamma^2\lambda_2}{\lambda_2}.$$

In particular, if there is a $\gamma < 1$ such that $\|I - M^{-1}A\|_A \leq \gamma$ independent of the discretization, this shows that PINVIT achieves a convergence rate independent of h.

An SPD preconditioner M is called **spectrally equivalent** if there are constants $c_L, c_U > 0$ (indpendent of h) such that

$$c_L\big(\mathbf{x}^\mathsf{T} M\mathbf{x}\big) \leq \mathbf{x}^\mathsf{T} A\mathbf{x} \leq c_U\big(\mathbf{x}^\mathsf{T} M\mathbf{x}\big) \tag{5.3}$$

for all $\mathbf{x} \in \mathbb{R}^n$. For example, the V-cycle and W-cycle multigrid preconditioners discussed in Section 4.2.3 are spectrally equivalent. By a suitable scaling of M, we may assume that (5.3) holds with $c_U = 1$ and some $0 \leq c_L \leq 1$. Note that

$$\|I - M^{-1}A\|_A = \|I - A^{1/2}M^{-1}A^{1/2}\|_2.$$

The smallest eigenvalue of $A^{1/2}M^{-1}A^{1/2}$ is the reciprocal of the largest eigenvalue of $A^{-1/2}MA^{-1/2}$, which can be bounded by

$$\lambda_{\max}\big(A^{-1/2}MA^{-1/2}\big) = \max_{\mathbf{x}\neq 0} \frac{\mathbf{x}^\mathsf{T} A^{-1/2}MA^{-1/2}\mathbf{x}}{\mathbf{x}^\mathsf{T}\mathbf{x}} = \max_{\mathbf{x}\neq 0} \frac{\mathbf{x}^\mathsf{T} M\mathbf{x}}{\mathbf{x}^\mathsf{T} A\mathbf{x}} \leq \frac{1}{c_L}.$$

Hence, the smallest eigenvalue of $A^{1/2}M^{-1}A^{1/2}$ is bounded from below by c_L. Analogously it can be shown that the largest eigenvalue of $A^{1/2}M^{-1}A^{1/2}$ is bounded from above by $c_U = 1$. In summary, we obtain $\|I - M^{-1}A\|_A \leq 1 - c_L$ and therefore the condition of Theorem 5.1 is satisfied with $\gamma = 1 - c_L$.

5.2 Locally Optimal Preconditioned CG (LOPCG)

In PINVIT, the next iterate is obtained as a particular linear combination of the current iterate $x^{(k)}$ and the preconditioned residual. It is tempting to ask whether there might be a better combination available. Moreover, in view of the advantages of Lanczos over inverse iteration, it makes sense to also take the previous last iterate $x^{(k-1)}$ into account. The idea of LOPCG is to choose the optimal linear combination of all three vectors.

Let us formalize the idea above for the generalized eigenvalue problem

$$A - \lambda B,$$

where both matrices $A, B \in \mathbb{R}^{n\times n}$ are symmetric positive definite. Given the subspace

$$\mathcal{U} = \operatorname{span}\{\mathbf{x}^{(k)}, \mathbf{r}^{(k)}, \mathbf{x}^{(k-1)}\} \tag{5.4}$$

with the preconditioned residual $\mathbf{r}^{(k)} = M^{-1}(A\mathbf{x}^{(k)} - \mu_k B\mathbf{x}^{(k)})$, we minimize the generalized Rayleigh quotient restricted to this subspace:

$$\mu_{k+1} = \min_{\substack{\mathbf{x}\in\mathcal{U}\\ \mathbf{x}\neq 0}} \frac{\mathbf{x}^\mathsf{T} A\mathbf{x}}{\mathbf{x}^\mathsf{T} B\mathbf{x}}. \tag{5.5}$$

The vector $\mathbf{x}$ attaining the minimum yields the next iterate $\mathbf{x}^{(k+1)}$.

Lemma 5.2 *Let $\mathcal{U} \subset \mathbb{R}^n$ be an m-dimensional subspace and let the columns of $U \in \mathbb{R}^{n\times m}$ form a basis $\mathcal{U}$. Then the optimum μ_{k+1} in (5.5) is given by the smallest eigenvalue of the matrix pencil $U^\mathsf{T}AU - \lambda(U^\mathsf{T}BU)$. Letting $\mathbf{y}$ denote an associated eigenvector, the optimum is attained by $\mathbf{x} = U\mathbf{y}$.*

Proof. We have

$$\min_{\substack{\mathbf{x}\in\mathcal{U}\\ \mathbf{x}\neq 0}} \frac{\mathbf{x}^\mathsf{T} A\mathbf{x}}{\mathbf{x}^\mathsf{T} B\mathbf{x}} = \min_{\substack{\mathbf{y}\in\mathbb{R}^m\\ \mathbf{y}\neq 0}} \frac{\mathbf{y}^\mathsf{T} U^\mathsf{T} AU\mathbf{y}}{\mathbf{y}^\mathsf{T} U^\mathsf{T} BU\mathbf{y}} = \min_{\substack{\mathbf{y}\in\mathbb{R}^m\\ \mathbf{y}\neq 0}} \frac{\mathbf{y}^\mathsf{T} U^\mathsf{T} AU\mathbf{y}}{\mathbf{y}^\mathsf{T} R^\mathsf{T} R\mathbf{y}} = \min_{\substack{\tilde{\mathbf{y}}\in\mathbb{R}^m\\ \tilde{\mathbf{y}}\neq 0}} \frac{\tilde{\mathbf{y}}^\mathsf{T} R^{-\mathsf{T}} U^\mathsf{T} AUR^{-1}\tilde{\mathbf{y}}}{\tilde{\mathbf{y}}^\mathsf{T}\tilde{\mathbf{y}}},$$

where R is the Cholesky factor of $U^\mathsf{T}BU$ and $\tilde{\mathbf{y}} := R\mathbf{y}$. From Theorem 1.16 we know that the last minimization problem yields the smallest eigenvalue of $R^{-\mathsf{T}}U^\mathsf{T}AUR^{-1}$, and it is attained by an associated eigenvector $\tilde{\mathbf{y}}$. By a similarity transformation, it can be seen that the eigenvalues of $R^{-\mathsf{T}}U^\mathsf{T}AUR^{-1} - \lambda I$ and $U^\mathsf{T}AU - \lambda R^\mathsf{T}R$ are identical. Moreover, the eigenvector $\tilde{\mathbf{y}}$ is transformed to an eigenvector $\mathbf{y} = R^{-1}\tilde{\mathbf{y}}$ of $U^\mathsf{T}AU - \lambda(U^\mathsf{T}BU)$. □

To implement LOPCG, we need a basis for $\mathcal{U}$. In view of (5.4), the most natural choice would be $\{\mathbf{x}^{(k)}, \mathbf{r}^{(k)}, \mathbf{x}^{(k-1)}\}$. However, since both $\mathbf{x}^{(k)}$ and $\mathbf{x}^{(k-1)}$ are expected to converge to the same vector, this basis would be very ill-conditoned, leading to numerical difficulties when attempting to solve the reduced eigenvalue problem $U^\mathsf{T}AU - \lambda(U^\mathsf{T}BU)$. To avoid this, an auxiliary vector sequence $\mathbf{p}^{(1)}, \mathbf{p}^{(2)}, \ldots$ is maintained with

$$\mathbf{p}^{(k+1)} = \left(\mathbf{0}, \mathbf{r}^{(k)}, \mathbf{p}^{(k)}\right)\mathbf{y}.$$

Initially, $\mathbf{p}^{(0)}$ is an empty vector. It can then be shown that

$$\operatorname{span}\{\mathbf{x}^{(k)}, \mathbf{r}^{(k)}, \mathbf{x}^{(k-1)}\} = \operatorname{span}\{\mathbf{x}^{(k)}, \mathbf{r}^{(k)}, \mathbf{p}^{(k)}\}.$$

The latter basis turns to be numerically more robust.[15]

[15] Even this basis can sometimes lead to numerical difficulties. An additional improvement based on orthogonalizing the basis in the B-inner product has been proposed in [Hetmaniuk, U.; Lehoucq, R. Basis selection in LOBPCG. J. Comput. Phys. 218 (2006), no. 1, 324–332].

The following algorithm summarizes our observations.

Algorithm 5.3 (LOPCG)
Input: Symmetric positive definite matrices $A, B \in \mathbb{R}^{n\times n}$, preconditioner M. Starting vector $\mathbf{x}^{(0)} \neq 0$.
Output: Approximate smallest eigenpair $(\mu, \mathbf{x})$
$\mu_0 = \langle \mathbf{x}^{(0)}, \mathbf{x}^{(0)} \rangle_A / \langle \mathbf{x}^{(0)}, \mathbf{x}^{(0)} \rangle_B$.
$\mathbf{p}^{(0)} = []$
for $k = 0, 1, \ldots$ (until converged) **do**
$\mathbf{r}^{(k)} = M^{-1}(A\mathbf{x}^{(k)} - \mu_k B\mathbf{x}^{(k)})$
$U = \left[\mathbf{x}^{(k)}, \mathbf{r}^{(k)}, \mathbf{p}^{(k)}\right]$
$\tilde{A} = U^\mathsf{T} A U, \ \tilde{B} = U^\mathsf{T} B U$
Find eigenpair $(\mu_{k+1}, \mathbf{y})$ for smallest eigenvalue of matrix pencil $\tilde{A} - \lambda\tilde{B}$.
$\mathbf{p}^{(k+1)} = y_2 \cdot \mathbf{r}^{(k)} + y_3 \cdot \mathbf{p}^{(k)}$
$\mathbf{x}^{(k+1)} = y_1 \cdot \mathbf{x}^{(k)} + \mathbf{p}^{(k+1)}$
$\mathbf{x}^{(k+1)} \leftarrow \mathbf{x}^{(k+1)} / \|\mathbf{x}^{(k+1)}\|_2$
end for
Return $(\mu, \mathbf{x}) = (\mu_{k+1}, \mathbf{x}^{(k+1)})$.

Remark 5.4 *The name of Algorithm 5.3 originates from the fact that it can be seen as an acceleration of the standard gradient method applied to the optimization problemn* $\min \mathbf{x}^\mathsf{T} A\mathbf{x}/\mathbf{x}^\mathsf{T} B\mathbf{x}$.

In Figure 5.1, we compare Algorithm 5.3 with PINVIT for the usual finite difference discretization of the unit square Laplace eigenvalue problem. The absolute error between the smallest eigenvalue and μ_k is shown. It can be seen that LOPCG is significantly faster than PINVIT, especially when the quality of the preconditioner is poor. In the exercises, we will experiment with the use of multigrid-based preconditioners.

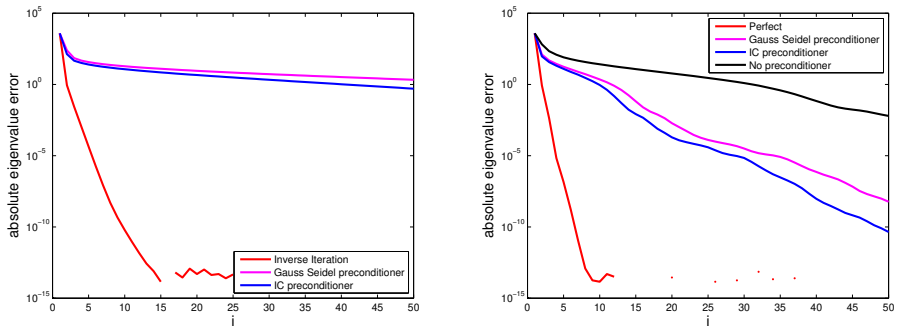


Figure 5.1. Left: *Convergence of inverse iteration and PINVIT with incomplete Cholesky IC(0) and Gauss-Seidel preconditioner.* Right: *Convergence of LOPCG with perfect ($M = A$), incomplete Cholesky IC(0) and Gauss-Seidel preconditioner.*

So far, we have only considered the computation of *the* smallest eigenpair. Although Algorithm 5.3 works with a three-dimensional subspace $\mathcal{U}$, it is not reasonable to expect that $\mathcal{U}$

contains information on larger eigenpairs. To compute the $p \geq 2$ smallest eigenpairs, a block variant called LOBPCG should be used, see Algorithm 5.5. In this algorithm, $[\Theta, Y] = \mathsf{RR}(U, A, B)$ returns a diagonal matrix $\Theta \in \mathbb{R}^{p\times p}$ containing the p smallest eigenvalues of $U^\mathsf{T}AU - \lambda U^\mathsf{T}BU$, and a matrix $Y \in \mathbb{R}^{n\times p}$ containing the associated eigenvectors. RR is usually implemented by a call to MATLAB's `eig`, which applies the transformation in the proof of Lemma 5.2 to reduce $U^\mathsf{T}AU - \lambda U^\mathsf{T}BU$ to a standard symmetric positive definite eigenvalue problem.

Algorithm 5.5 (LOBPCG)

Input: Symmetric positive definite matrices $A, B \in \mathbb{R}^{n\times n}$, preconditioner M. Starting matrix $X^{(0)} \in \mathbb{R}^{n\times p}$.

Output: Pair (Θ, X) containing approximations to p smallest eigenvalues of $A - \lambda B$.

$[\Theta_0, Y] \leftarrow \mathsf{RR}(X^{(0)}, A, B)$
$X^{(0)} \leftarrow X^{(0)}Y$
$P^{(0)} = []$
for $k = 0, 1, \ldots$ (until converged) **do**
  $R^{(k)} = M^{-1}(AX^{(k)} - BX^{(k)}\Theta_k)$
  $U = \left[X^{(k)},\, R^{(k)},\, P^{(k)}\right]$
  $[\Theta_{k+1}, Y] = \mathsf{RR}(U, A, B)$
  $X^{(k+1)} \leftarrow \left[X^{(k)},\, R^{(k)},\, P^{(k)}\right]Y$
  $P^{(k+1)} \leftarrow \left[0,\, R^{(k)},\, P^{(k)}\right]Y$
end for
Return $(\Theta, X) = (\Theta_{k+1}, X^{(k+1)})$.

Chapter 6

Matrix functions

Together with linear systems and eigenvalue problems, the evaluation of matrix functions is one of the central topics in numerical linear algebra. This chapter is meant to be a brief excursion; we refer to the excellent book [N. J. Higham. Functions of matrices. SIAM, 2008.] for the (nearly) complete picture.

6.1 Basic definition and properties

Given a square matrix $A \in \mathbb{C}^{n\times n}$ and a scalar function $f : \Omega \to \mathbb{C}$ with $\Omega \subset \mathbb{C}$, the matrix function $f(A)$ is again an $n \times n$ matrix. In the following, we will always assume that f is analytic on Ω and that $\Omega \subset \mathbb{C}$ contains the eigenvalues of A.

When $f \equiv p$ happens to be a polynomial

$$p(z) = \alpha_0 + \alpha_1 z + \alpha_2 z^2 + \cdots + \alpha_m z^m,$$

we can define the matrix function $p(A)$ by simply replacing z with A:

$$p(A) = \alpha_0 + \alpha_1 A + \alpha_2 A^2 + \cdots + \alpha_m A^m,$$

where the power A^j is understood as multiplying j times A with itself. We can extend this definition to general f by polynomial interpolation. For example, if

$$A = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}, \quad \lambda_1 \neq \lambda_2,$$

we can choose the linear interpolating polynomial

$$p(z) = \frac{z-\lambda_2}{\lambda_1-\lambda_2} f(\lambda_1) + \frac{z-\lambda_1}{\lambda_2-\lambda_1} f(\lambda_2),$$

which satisfies $p(\lambda_1) = f(\lambda_1)$ and $p(\lambda_2) = f(\lambda_2)$. This then yields

$$f(A) := p(A) = \frac{f(\lambda_1)}{\lambda_1-\lambda_2}(A-\lambda_2 I_2) + \frac{f(\lambda_2)}{\lambda_2-\lambda_1}(A-\lambda_1 I_2) = \begin{pmatrix} f(\lambda_1) & 0 \\ 0 & f(\lambda_2) \end{pmatrix}.$$

Moreover, any other polynomial p satisfying the interpolation conditions $p(\lambda_1) = f(\lambda_1)$ and $p(\lambda_2) = f(\lambda_2)$ will yield the same matrix $p(A)$.

Things become a little more complicated if A cannot be diagonalized by a similarity transformation. For example, consider the 2×2 Jordan block

$$A = \begin{pmatrix} \lambda & 1 \\ 0 & \lambda \end{pmatrix}.$$

Clearly, the constant polynomial $p_1(z) \equiv f(\lambda)$ interpolates f at the only eigenvalue λ of A. Another possible choice is $p_2(z) = f(\lambda) + (z - \lambda)f'(\lambda)$, which is the truncated Taylor expansion of f. With these two choices we obtain

$$p_1(A) = \begin{pmatrix} f(\lambda) & 0 \\ 0 & f(\lambda) \end{pmatrix} \neq \begin{pmatrix} f(\lambda) & f'(\lambda) \\ 0 & f(\lambda) \end{pmatrix} = p_2(A).$$

To avoid this ambiguity, we need to perform Hermite interpolation in order to also match the derivatives of f in the presence of Jordan blocks.

Definition 6.1 *Let $A \in \mathbb{C}^{m\times m}$ have the pairwise distinct eigenvalues $\lambda_1, \ldots, \lambda_s$ and let $\mathsf{ind}_{\lambda_i}(A)$ denote the index of λ_i, i.e., the size of the largest Jordan block associated with λ_i. Then the matrix function associated with a scalar function f is defined as $f(A) := p(A)$, where $p(z)$ is the unique Hermite interpolating polynomial of degree less than $\sum_{i=1}^{s} \mathsf{ind}_{\lambda_i} A$ satisfying*

$$\frac{\partial^g}{\partial z^g} p(\lambda_i) = \frac{\partial^g}{\partial z^g} f(\lambda_i), \quad g = 0, \ldots, \mathsf{ind}_{\lambda_i} A - 1, \quad i = 1, \ldots, s. \tag{6.1}$$

In the following paragraphs, we collect some basic properties of matrix functions.

Diagonalization and Jordan canonical form It is easy to see that for every invertible matrix P the relation

$$p(A) = P \cdot p(P^{-1}AP) \cdot P^{-1}.$$

holds for any polynomial p. Hence, the same statement,

$$f(A) = P \cdot f(P^{-1}AP) \cdot P^{-1}, \tag{6.2}$$

holds for a general function f. This becomes particularly interesting if A is diagonalizable:

$$P^{-1}AP = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) =: \Lambda.$$

Then (6.2) implies

$$f(A) = P \cdot f(\Lambda) \cdot P^{-1} = P \cdot \operatorname{diag}\big(f(\lambda_1), \ldots, f(\lambda_n)\big) \cdot P^{-1}. \tag{6.3}$$

For a matrix that is not diagonalizable, we have to consider its Jordan canonical form:

$$P^{-1}AP = \operatorname{diag}\big(J_1, J_2, \ldots, J_p\big),$$

with $n_i \times n_i$ Jordan blocks

$$J_i = \begin{pmatrix} \lambda_i & 1 & & \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{pmatrix}$$

Each Jordan block is processed individually. The truncated Taylor expansion

$$p_i(\lambda) := f(\lambda_i) + f'(\lambda_i)(\lambda - \lambda_i) + \frac{f''(\lambda_i)}{2}(\lambda - \lambda_i)^2 + \cdots + \frac{f^{(n_i-1)}(\lambda_i)}{(n_i-1)!}(\lambda - \lambda_i)^{n_i-1}$$

gives the Hermite interpolating polynomial satisfying the requirements of Definition 6.1 for $J_i(\lambda_i)$. Thus,

$$f\big(J_i\big) = \begin{pmatrix} f(\lambda_i) & f'(\lambda_i) & \cdots & \frac{f^{(n_i-1)}(\lambda_i)}{(n_i-1)!} \\ & \ddots & \ddots & \vdots \\ & & \ddots & f'(\lambda_i) \\ & & & f(\lambda_i) \end{pmatrix}.$$

In total, we obtain

$$f(A) = P \cdot \operatorname{diag}\big(f(J_1), f(J_2), \ldots, f(J_p)\big) \cdot P^{-1},$$

which could equivalently be used for defining a matrix function.

Power series Suppose that $f : \Omega \to \mathbb{C}$ admits a power series

$$f(z) = \sum_{k=0}^{\infty} \alpha_k (z - z_0)^k$$

that converges absolutely for every $z \in \Omega$. Then

$$f(A) = \sum_{k=0}^{\infty} \alpha_k (A - z_0 I)^k,$$

also converges absolutely (with respect to a matrix norm). This can be easily seen, at least for diagonalizable matrices.

Cauchy integral Since $f : \Omega \to \mathbb{C}$ is analytic, Cauchy's integral formula yields

$$f(\lambda) = \frac{1}{2\pi \mathrm{i}} \oint_\Gamma \frac{f(z)}{z - \lambda} \, \mathrm{d}z,$$

for any contour $\Gamma \subset \Omega$ encircling λ. Applied to each of the diagonal entries of $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$, we obtain

$$f(\Lambda) = \frac{1}{2\pi \mathrm{i}} \oint_\Gamma f(z)(zI - \Lambda)^{-1} \, \mathrm{d}z,$$

for any contour $\Gamma \subset \Omega$ encircling $\lambda_1, \ldots, \lambda_n$.

Using $P(zI - \Lambda)^{-1}P^{-1} = (zI - P\Lambda P^{-1})^{-1}$, we thus obtain together with (6.2)

$$f(A) = \frac{1}{2\pi \mathrm{i}} \oint_\Gamma f(z)(zI - A)^{-1} \, \mathrm{d}z. \tag{6.4}$$

This Cauchy integral formula for matrix functions not only holds for diagonalizable matrix A; it remains true for general A.

In the following, we will discuss the most popular examples of matrix functions.

Matrix exponential

$\exp(A)$ is the "mother" of all matrix functions. It can be defined either via Definition 6.1 or via the power series

$$\exp(A) = \sum_{k=0}^{\infty} \frac{1}{k!} A^k, \tag{6.5}$$

which converges for every matrix A. The following lemma collects the most important properties of the matrix exponential.

Lemma 6.2 *Let $A \in \mathbb{C}^{n\times n}$. Then the following statements hold.*

(i) $\exp(A^H) = \exp(A)^H$.

(ii) $\frac{\partial}{\partial t} e^{tA} = Ae^{tA}$

(iii) If $AB = BA$ for some matrix $B \in \mathbb{C}^{n\times n}$ then $\exp(A+B) = \exp(A)\cdot\exp(B)$.

(iv) $\exp(A)$ *is always invertible and* $\exp(A)^{-1} = \exp(-A)$.

(v) $\det(\exp(A)) = e^{\mathrm{trace}(A)}$.

Proof. (i) immediately follows from (6.5).

(ii):

$$\frac{\partial}{\partial t} e^{tA} = \frac{\partial}{\partial t}\left(\sum_{k=0}^{\infty} \frac{t^k}{k!} A^k\right) = \sum_{k=1}^{\infty} \frac{t^{k-1}}{(k-1)!} A^k = \sum_{k=0}^{\infty} \frac{t^k}{k!} A^{k+1} = Ae^{tA}.$$

(iii): If A and B commute, it immediately follows that all the four matrices A, B, e^{tA}, e^{tB} commute with each other. Let $X(t) = e^{tA}e^{tB}$. Using that the product rule also holds when differentiating matrix-valued functions (but it is important to maintain the order of multiplication), we obtain that

$$\begin{aligned} X'(t) &= \frac{\partial}{\partial t}\big(e^{tA}e^{tB}\big) = \Big(\frac{\partial}{\partial t}e^{tA}\Big)e^{tB} + e^{tA}\Big(\frac{\partial}{\partial t}e^{tB}\Big) \\ &= Ae^{tA}e^{tB} + e^{tA}Be^{tB} = (A+B)e^{tA}e^{tB} = (A+B)X(t). \end{aligned}$$

Hence, $X(t)$ satisfies the differential equation $X'(t) = (A+B)X(t)$ with initial condition $X(0) = I_n$. We know from part (ii) that one possible solution for this initial value problem is given by $e^{t(A+B)}$. By a fundamental result on ordinary differential equations, this solution is unique and hence, necessarily, $X(t) = e^{t(A+B)}$.

(iv) This follows directly from part (iii) by setting $B = -A$.

(v) is an exercise for you. □

It is important to remark that the statement of Lemma 6.2 (iii) does *not* hold for general A, B. For example, choose $A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$. Then

$$e^A = \begin{pmatrix} e^1 & 0 \\ 0 & 1 \end{pmatrix}, \quad e^B = I + B + 0 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad \Rightarrow \quad e^A e^B = \begin{pmatrix} e^1 & e^1 \\ 0 & 1 \end{pmatrix}.$$

On the other hand, $e^{A+B} = \begin{pmatrix} e^1 & e^1 - 1 \\ 0 & 1 \end{pmatrix}$.

For general A, B, the famous Baker-Campbell-Hausdorff formula gives $\exp(A)\exp(B) = \exp(C)$ for

$$C = A + B + \frac{1}{2}[A,B] + \frac{1}{12}[A,[A,B]] - \frac{1}{12}[B,[A,B]] - \frac{1}{24}[B,[A,[A,B]]] + \cdots,$$

with the commutator $[A,B] := AB - BA$.

Matrix logarithm

The matrix logarithm of $A \in \mathbb{C}^{n\times n}$ is any matrix X such that $\exp(X) = A$. Any invertible matrix A has a logarithm. The concept of matrix logarithm is highly non-unique; it depends which branch of the (scalar) logarithm is taken for each Jordan block of A. If A is invertible and has no negative real eigenvalues then the *principal* logarithm $\log(A)$ is obtained by choosing the principal branch for each eigenvalue. Thus the eigenvalues of $\log(A)$ are located in the strip $\{z : -\pi < \mathsf{Im}(z) < \pi\}$.

Matrix square root

The matrix square root of $A \in \mathbb{C}^{n\times n}$ is any matrix X such that $X^2 = A$. Any invertible matrix A has a square root; a singular matrix may or may not have a square root depending on the structure of the Jordan blocks for the eigenvalue zero (diagonalizable matrices are, of course, safe). Similar to the logarithm, the *principal* square root $A^{1/2}$ of a matrix is obtained by taking the principal square root for each eigenvalue of A, which is possible if A has no real negative eigenvalues.

Matrix sign function

For any complex number with nonzero real part, define

$$\operatorname{sign}(z) = \begin{cases} 1 & \mathsf{Re}(z) > 0 \\ -1 & \mathsf{Re}(z) < 0 \end{cases}$$

Then the matrix sign function $\operatorname{sign}(A)$ can be defined for any matrix A that has no eigenvalues with zero real part. If A can be transformed to

$$P^{-1}AP = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix},$$

for example by means of the Jordan canonical form, then

$$\operatorname{sign}(A) = P \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} P^{-1}.$$

The sign function gives the ability to apply a step function to the spectrum, which makes it useful in eigenvalue computation.

Matrix inverse

Finally, the matrix inverse A^{-1} is of course also a matrix function $f(A)$, with $f(z) = 1/z$.

6.2 Exponential integrators*

Matrix exponentials appear naturally in the context of systems of ordinary differential equations (ODEs). In particular, the solution of the linear ODE

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t), \quad \mathbf{x}(0) = \mathbf{x}_0 \tag{6.6}$$

is given by

$$\dot{\mathbf{x}}(t) = \exp(tA)\mathbf{x}_0.$$

Exponential integrators are useful for particular types of stiff nonlinear ODEs, for which it is possible to identify a simpler prototype equation with similar stiffness properties. For an autonomous ODE, this can sometimes be achieved by linearization, giving rise to an ODE of the form

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + g(\mathbf{x}(t)), \quad \mathbf{x}(0) = \mathbf{x}_0.$$

By variation of constants, its solution satisfies the integral equation

$$\mathbf{x}(h) = \exp(hA)\mathbf{x}_0 + \int_0^h \exp((h-s)A)g(\mathbf{x}(s))\ \mathrm{d}s,$$

on any interval $[0, h]$. The simplest way to proceed from here is to approximate the integral by

$$\begin{aligned}\int_0^h \exp((h-s)A)g(\mathbf{x}(s))\ \mathrm{d}s &\approx \int_0^h \exp((h-s)A)g(\mathbf{x}_0)\ \mathrm{d}s \\ &= A^{-1}[\exp(hA) - I]g(\mathbf{x}_0) = h\varphi_1(hA)g(\mathbf{x}_0),\end{aligned}$$

with the φ-function $\varphi_1(z) = (e^z - 1)/z$.

This approximation gives the first step of the **exponential Euler method**

$$\mathbf{x}_1 = \exp(hA)\mathbf{x}_0 + h\varphi_1(hA)g(\mathbf{x}_0).$$

Repeating the described procedure, we obtain the recursion

$$\mathbf{x}_{k+1} = \exp(hA)\mathbf{x}_k + h\varphi_1(hA)g(\mathbf{x}_k). \tag{6.7}$$

It turns out that the φ-function needed in (6.7) can also be computed via the matrix exponential.

Lemma 6.3

$$\exp\begin{pmatrix} A & \mathbf{c} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \exp(A) & h\varphi_1(hA)\mathbf{c} \\ 0 & 0 \end{pmatrix}$$

Proof. We have

$$\begin{pmatrix} A & \mathbf{c} \\ 0 & 0 \end{pmatrix}^2 = \begin{pmatrix} A^2 & A\mathbf{c} \\ 0 & 0 \end{pmatrix}, \quad \dots \quad \begin{pmatrix} A & \mathbf{c} \\ 0 & 0 \end{pmatrix}^2 = \begin{pmatrix} A^k & A^{k-1}\mathbf{c} \\ 0 & 0 \end{pmatrix}.$$

By the power series (6.5) it follows that

$$\exp\begin{pmatrix} A & \mathbf{c} \\ 0 & 0 \end{pmatrix} = \sum_{k=0}^{\infty} \frac{1}{k!}\begin{pmatrix} A & \mathbf{c} \\ 0 & 0 \end{pmatrix}^k = \begin{pmatrix} \sum_{k=0}^{\infty} \frac{1}{k!}A & \sum_{k=1}^{\infty} \frac{1}{k!}A^{k-1}\mathbf{c} \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} \exp(A) & \varphi_1(A)\mathbf{c} \\ 0 & 0 \end{pmatrix},$$

where we used

$$\varphi_1(z) = \frac{1}{z}(e^z - 1) = \frac{1}{z}\sum_{k=1}^{\infty} \frac{1}{k!}z^k = \sum_{k=1}^{\infty} \frac{1}{k!}z^{k-1}.$$

□

6.3 The Schur-Parlett algorithm for small matrices

On first sight, the most natural way to compute $f(A)$ seems to consist of diagonalizing A and using (6.3). However, as we have already seen in the exercises, this leads to loss of accuracy if the matrix P is not particularly well-conditioned. Unless A is symmetric (or, more generally, a normal matrix) approaches based on diagonalization should be avoided.

For a *general* matrix function $f(A)$, the MATLAB command `funm` is based on first computing the Schur form

$$Q^H A Q = T,$$

with an upper triangular matrix T. Since $f(A) = Q f(T) Q^H$, it remains to compute $F = f(T)$. From the definition of matrix functions, it is clear that F is also upper triangular and its diagonal entries are given by

$$f_{11} = f(t_{11}),\ f_{22} = f(t_{22}),\ \ldots,\ f_{nn} = f(t_{nn}).$$

The elements in the strictly upper triangular part are determined from the fact that F and T must commute, $FT = TF$. Resolving this relation gives Algorithm 6.4, which requires the diagonal entries (that is, the eigenvalues) of T to be mutually distinct. If this condition is not satisfied or if some diagonal elements are close to each other (in a certain sense), a block variant of the algorithm should be used. This block variant of Algorithm 6.4 is state-of-the-art for evaluating *general* functions of small to medium-sized matrices. Nevertheless, it is rarely used in practice. For nearly all functions of practical interest, specialized methods are the preferred choice. For example, the MATLAB commands `expm` and `logm` are based on a totally different class of methods, the so-called scaling and squaring algorithm discussed in the next section.

Algorithm 6.4 (Schur-Parlett algorithm)
Input: Upper triangular matrix $T \in \mathbb{R}^{n\times n}$ with mutually distinct diagonal entries.
Output: $f(T)$.
for $i = 1, \ldots, n$ **do**
$f_{ii} = f(t_{ii})$
end for
for $j = 2, \ldots, n$ **do**
for $i = j-1, j-2, \ldots, 1$ **do**

$$f_{ij} = t_{ij}\frac{f_{ii} - f_{jj}}{t_{ii} - t_{jj}} + \frac{1}{t_{ii} - t_{jj}} \sum_{k=i+1}^{j-1} (f_{ik} t_{kj} - t_{ik} f_{kj})$$

end for
end for

6.4 Scaling and squaring for matrix exponentials

The state-of-the-art approach for computing matrix exponentials of small- to medium-sized matrices is the scaling and squaring method.

The first ingredient of scaling and squaring is that the exponential function $\exp(x)$ admits good polynomial and rational approximations when $|x|$ remains modest. For example, consider the truncated Taylor expansion

$$\exp(x) = \sum_{k=0}^{\infty} \frac{1}{k!} x^k \approx \sum_{k=0}^{N} \frac{1}{k!} x^k$$

for some integer N. Replacing x by A gives the approximation $\sum_{k=0}^{N} \frac{1}{k!}A^k$ with the approximation error satisfying

$$\Big\| \exp(A) - \sum_{k=0}^{N} \frac{1}{k!}A^k \Big\|_2 = \Big\| \sum_{k=N+1}^{\infty} \frac{1}{k!}A^k \|_2 \leq \sum_{k=N+1}^{\infty} \frac{1}{k!}\|A\|_2^k = \frac{\|A\|_2^{N+1}}{(N+1)!}\exp(\|A\|_2).$$

For example, for $N = 4$ and $\|A\|_2 = 0.01$, this implies that the error is bounded by 10^{-10} while for $\|A\|_2 = 3$ this bound increases to 13.6. This illustrates that is much easier to approximate $\exp(A)$ when $\|A\|_2$ is small or, at least, not too large.

The second ingredient of scaling and squaring is the insight that the norm of A can be effectively reduced using basic properties of matrix exponentials. Repeatedly applying the relation $\exp(A/2)\exp(A/2) = \exp(A)$ gives

$$\exp(2^{-j}A)^{2^j} = \exp(A), \quad j \in \mathbb{N}.$$

Thus, after scaling by 2^{-j} for sufficiently large $j \in \mathbb{N}$, one can cheaply approximate $\exp(2^{-j}A)$, for example, by Taylor expansion and then squares the result j times with itself. This leads to the following template for **scaling and squaring**:

1. Choose $j \in \mathbb{N}$ such that $\|\tilde{A}\|_2$ is small for $\tilde{A} := 2^{-j}A$.
2. Obtain approximation $Y \exp(\tilde{A})$ by polynomial or rational approximation of the exponential function.
3. for $k = 1, \ldots, j$, $Y \leftarrow Y^2$, end
4. Return $Y \approx \exp(A)$.

Step 1 requires an estimate of $\|A\|_2$, which can be obtained by a few steps of the power method. The choice of which norm $\|\tilde{A}\|_2$ is considered small depends on the approximation performed in Step 2. Reducing the norm to 10^{-3} allows one to use a truncated Taylor polynomial of modest degree in Step 2 in order to attain double precision accuracy. If one uses rational approximations (Padé approximation) it suffices to reduce the norm to 1. These choices (threshold for norm reduction, polynomial/rational approximation) are hard coded in implementations of the scaling and squaring method, such as Matlab's `expm`. We refer to Section 10.3 of Higham's book for details.

6.5 The Arnoldi method for large matrices

The Arnoldi method can be extended in a straightforward fashion to approximate $f(A)\mathbf{b}$ for a vector $\mathbf{b} \in \mathbb{C}^n$. Suppose that we have run k steps of Algorithm 2.23 to generate an orthonormal basis U_k for $\mathcal{K}_k(A, \mathbf{b})$. This yields the Arnoldi decomposition

$$AU_k = U_k H_k + h_{k+1,k}\mathbf{u}_{k+1}\mathbf{e}_k^{\mathsf{T}}. \tag{6.8}$$

Then we obtain the approximation

$$f(A)\mathbf{b} \approx \|\mathbf{b}\|_2 U_k f(H_k)\mathbf{e}_1 =: \mathbf{f}_k. \tag{6.9}$$

Theorem 6.5 *Let $A \in \mathbb{C}^{n\times n}$ be a Hermitian matrix with eigenvalues contained in the interval $[\alpha, \beta]$ and let $f : \Omega \to \mathbb{C}$ be analytic on a domain $\Omega \subset \mathbb{C}$ satisfying $[\alpha, \beta] \subset \Omega$. Then*

$$\|f(A)\mathbf{b} - \mathbf{f}_k\|_2 \leq 2\|\mathbf{b}\|_2 \cdot \min_{p\in\Pi_{k-1}} \max_{z\in[\alpha,\beta]} |p(z) - f(z)| \tag{6.10}$$

holds with $\mathbf{f}_k$ defined in (6.9).

Proof. We start by noting that $U_k U_k^T$ is the orthogonal projector onto $\mathcal{K}_k(A, \mathbf{b})$. Because $\mathbf{b} \in \mathcal{K}_k(A, \mathbf{b})$, we have $U_k U_k^T \mathbf{b} = \mathbf{b}$. If $k \geq 2$, we also have that $A\mathbf{b} \in \mathcal{K}_k(A, \mathbf{b})$, and thus $A\mathbf{b} = U_k U_k^T A\mathbf{b} = U_k U_k^T A U_k U_k^T \mathbf{b} = \|\mathbf{b}\|_2 U_k H_k \mathbf{e}_1$. We can apply this argument repeatedly to show for all $j \leq k-1$ that

$$A^j \mathbf{b} = U_k U_k^T A^j \mathbf{b} = U_k U_k^T A U_k U_k^T A^{j-1} \mathbf{b} = \cdots = U_k (U_k^T A U_k)^j U_k^T \mathbf{b} = \|\mathbf{b}\|_2 U_k H_k^j \mathbf{e}_1$$

By linearity, it follows that

$$p(A)\mathbf{b} = \|\mathbf{b}\|_2 U_k p(H_k)\mathbf{e}_1, \quad \forall p \in \Pi_{k-1}.$$

In other words, the approximation (6.9) is exact for any polynomial of degree at most $k-1$. Thus,

$$\begin{aligned}
\|f(A)\mathbf{b} - \mathbf{f}_k\|_2 &= \big\|f(A)\mathbf{b} - p(A)\mathbf{b} + \|\mathbf{b}\|_2 U_k p(H_k)\mathbf{e}_1 - \|\mathbf{b}\|_2 U_k f(H_k)\mathbf{e}_1\big\|_2 \\
&\leq \big\|f(A)\mathbf{b} - p(A)\mathbf{b}\big\|_2 + \|\mathbf{b}\|_2 \big\|p(H_k)\mathbf{e}_1 - f(H_k)\mathbf{e}_1\big\|_2 \\
&\leq \|\mathbf{b}\|_2 \big(\|f(A) - p(A)\|_2 + \|f(H_k) - p(H_k)\|_2\big) \\
&= \|\mathbf{b}\|_2 \big(\max_{z \in \Lambda(A)} |f(z) - p(z)| + \max_{z \in \Lambda(H_k)} |f(z) - p(z)|\big) \\
&\leq 2\|\mathbf{b}\|_2 \cdot \max_{z \in [\alpha, \beta]} |f(z) - p(z)|.
\end{aligned}$$

holds for any $p \in \Pi_{k-1}$. In the last step, we have used that $\Lambda(H_k) \subset [\alpha, \beta]$ holds due to eigenvalue interlacing. □

Two remarks on Theorem 6.5:

- The right-hand side of (6.10) is a classical polynomial approximation problem. This can be addressed by different polynomial expansion techniques. For example, if $\alpha < \beta \leq 0$, a standard estimate for the remainder of the Taylor expansion of $f(z) = \exp(z)$ gives

  $$\|f(A)\mathbf{b} - \mathbf{f}_k\|_2 \leq 2\|\mathbf{b}\|_2 \frac{|\alpha|^k}{k!}.$$

  Much better estimates can be obtained, e.g., by Chebyshev expansion.[16]

- The result of Theorem 6.5 can be extended to general matrices by replacing $[\alpha, \beta]$ with the numerical range of A. For details, we refer to the seminal paper [M. Hochbruck and C. Lubich, On Krylov subspace approximations to the matrix exponential operator, SIAM J. Numer. Anal., 34 (1997), pp. 1911–1925].

[16] See [H. Tal-Ezer, Polynomial approximation of functions of matrices and applications, J. Sci. Comput. 4 (1989), pp. 25–60].