Lecture Notes on
Solving Large Scale Eigenvalue Problems

Prof. Dr. Peter Arbenz
Computer Science Department
ETH Zürich
Email: arbenz@inf.ethz.ch

Spring semester 2016

Contents

Chapter 1

Introduction

Before we start with the subject of this notes we want to show how one actually arrives at large eigenvalue problems in practice. In the following, we restrict ourselves to problems from physics [7, 18, 14] and computer science.

1.1 What makes eigenvalues interesting?

In physics, eigenvalues are usually related to vibrations. Objects like violin strings, drums, bridges, sky scrapers can swing. They do this at certain frequencies. And in some situations they swing so much that they are destroyed. On November 7, 1940, the Tacoma narrows bridge collapsed, less than half a year after its opening. Strong winds excited the bridge so much that the platform in reinforced concrete fell into pieces. A few years ago the London millennium footbridge started wobbling in a way that it had to be closed. The wobbling had been excited by the pedestrians passing the bridge. These are prominent examples of vibrating structures.

But eigenvalues appear in many other places. Electric fields in cyclotrones, a special form of particle accelerators, have to oscillate in a precise manner, in order to accelerate the charged particles that circle around its center. The solutions of the Schrödinger equation from quantum physics and quantum chemistry have solutions that correspond to vibrations of the, say, molecule it models. The eigenvalues correspond to energy levels that molecule can occupy.

Many characteristic quantities in science *are* eigenvalues:

- decay factors,
- frequencies,
- norms of operators (or matrices),
- singular values,
- condition numbers.

In the sequel we give a number of examples that show why computing eigenvalues is important. At the same time we introduce some notation.

1.2 Example 1: The vibrating string

1.2.1 Problem setting

Let us consider a string as displayed in Fig. 1.1. The string is fixed at both ends, at $x = 0$ and $x = L$. The x-axis coincides with the string's equilibrium position. The displacement of the rest position at x, $0 < x < L$, and time t is denoted by $u(x,t)$.

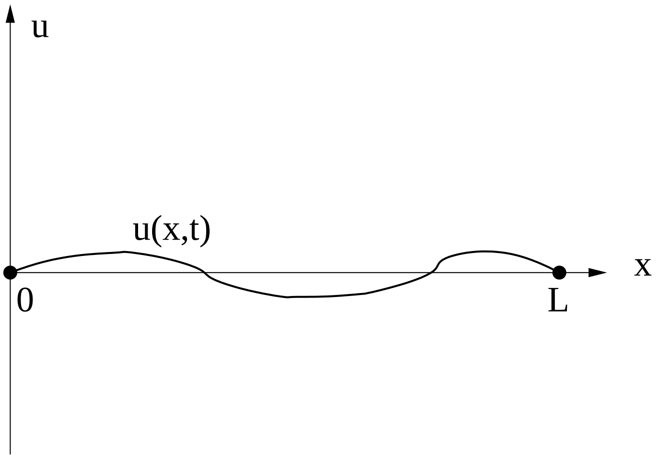


Figure 1.1: A vibrating string fixed at both ends.

We will assume that the spatial derivatives of u are not very large:

$$\left|\frac{\partial u}{\partial x}\right| \quad \text{is small.}$$

This assumption entails that we may neglect terms of higher order.

Let $v(x,t)$ be the velocity of the string at position x and at time t. Then the kinetic energy of a string section ds of mass $dm = \rho\, ds$ is given by

$$dT = \frac{1}{2}dm\, v^2 = \frac{1}{2}\rho\, ds\, \left(\frac{\partial u}{\partial t}\right)^2. \tag{1.1}$$

From Fig. 1.2 we see that $ds^2 = dx^2 + \left(\frac{\partial u}{\partial x}\right)^2 dx^2$ and thus

$$\frac{ds}{dx} = \sqrt{1 + \left(\frac{\partial u}{\partial x}\right)^2} = 1 + \frac{1}{2}\left(\frac{\partial u}{\partial x}\right)^2 + \text{ higher order terms.}$$

Plugging this into (1.1) and omitting also the second order term (leaving just the number 1) gives

$$dT = \frac{\rho\, dx}{2}\left(\frac{\partial u}{\partial t}\right)^2.$$

The kinetic energy of the whole string is obtained by integrating over its length,

$$T = \int_0^L dT(x) = \frac{1}{2}\int_0^L \rho(x)\left(\frac{\partial u}{\partial t}\right)^2 dx$$

The potential energy of the string has two components

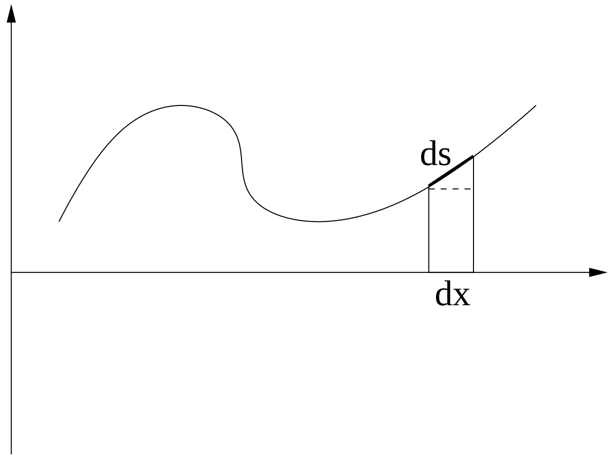


Figure 1.2: A vibrating string, local picture.

1. the stretching times the exerted strain τ,

$$\tau\int_0^L ds-\tau\int_0^L dx=\tau\int_0^L\left(\sqrt{1+\left(\frac{\partial u}{\partial x}\right)^2}-1\right)dx$$
$$=\tau\int_0^L\left(\frac{1}{2}\left(\frac{\partial u}{\partial x}\right)^2+\text{ higher order terms}\right)dx$$

2. exterior forces of density f,

$$-\int_0^L fudx.$$

Summing up, the potential energy of the string becomes

$$V=\int_0^L\left(\frac{\tau}{2}\left(\frac{\partial u}{\partial x}\right)^2-fu\right)dx. \tag{1.2}$$

To consider the motion (vibration) of the string in a certain time interval $t_1\le t\le t_2$ we form the integral

$$I(u)=\int_{t_1}^{t_2}(T-V)\,dt=\frac{1}{2}\int_{t_1}^{t_2}\int_0^L\left[\rho(x)\left(\frac{\partial u}{\partial t}\right)^2-\tau\left(\frac{\partial u}{\partial x}\right)^2+2fu\right]dx\,dt \tag{1.3}$$

Here functions $u(x,t)$ are admitted that are differentiable with respect to x and t and satisfy the **boundary conditions (BC)** that correspond to the fixing,

$$u(0,t)=u(L,t)=0,\qquad t_1\le t\le t_2, \tag{1.4}$$

as well as given **initial conditions** and **end conditions**,

$$\begin{matrix}u(x,t_1)=u_1(x),\\ u(x,t_2)=u_2(x),\end{matrix}\qquad 0<x<L. \tag{1.5}$$

According to the **principle of Hamilton** a mechanical system with kinetic energy T and potential energy V behaves in a time interval $t_1 \leq t \leq t_2$ for given initial and end positions such that

$$I = \int_{t_1}^{t_2} L\,dt, \qquad L = T - V,$$

is minimized.

Let $u(x,t)$ be such that $I(u) \leq I(w)$ *for all* w, that satisfy the initial, end, and boundary conditions. Let $w = u + \varepsilon\, v$ with

$$(*) \qquad v(0,t) = v(L,t) = 0, \qquad v(x,t_1) = v(x,t_2) = 0.$$

v is called a *variation*. We now consider $I(u + \varepsilon\, v)$ as a function of ε. Then we have the equivalence

$$I(u) \text{ minimal} \quad \Longleftrightarrow \quad \boxed{\frac{dI}{d\varepsilon}(u) = 0 \text{ for all admitted } v.}$$

Plugging $u + \varepsilon\, v$ into eq. (1.3) we obtain

$$(1.6)\quad \begin{aligned} I(u+\varepsilon\, v) &= \frac{1}{2}\int\limits_{t_1}^{t_2}\int\limits_{0}^{L}\left[\rho(x)\left(\frac{\partial(u+\varepsilon\, v)}{\partial t}\right)^2 - \tau\left(\frac{\partial(u+\varepsilon\, v)}{\partial x}\right)^2 + 2f(u+\varepsilon\, v)\right] dx\,dt \\ &= I(u) + \varepsilon \int\limits_{t_1}^{t_2}\int\limits_{0}^{L}\left[\rho(x)\frac{\partial u}{\partial t}\frac{\partial v}{\partial t} - \tau\frac{\partial u}{\partial x}\frac{\partial v}{\partial x} + 2fv\right] dx\,dt + \mathcal{O}(\varepsilon^2). \end{aligned}$$

Thus, after integration by parts, exploiting the conditions in $(*)$, the equation

$$\frac{\partial I}{\partial \varepsilon} = \int_{t_1}^{t_2}\int_0^L \left[\rho\frac{\partial^2 u}{\partial t^2} - \tau\frac{\partial^2 u}{\partial x^2} + 2\,f\right] v\,dx\,dt = 0$$

must hold *for all* admissible v. Therefore, the bracketed expression must vanish,

$$(1.7) \qquad -\rho\frac{\partial^2 u}{\partial t^2} + \tau\frac{\partial^2 u}{\partial x^2} = 2\,f.$$

This last differential equation is named **Euler-Lagrange equation**.

If the force is proportional to the displacement $u(x,t)$ (like, e.g., in Hooke's law) then we get a differential equation of the form

$$(1.8) \qquad \boxed{\begin{array}{c} -\rho(x)\dfrac{\partial^2 u}{\partial t^2} + \dfrac{\partial}{\partial x}\left(p(x)\dfrac{\partial u}{\partial x}\right) + q(x)u(x,t) = 0. \\ u(0,t) = u(1,t) = 0 \end{array}}$$

which is a special case of the Euler-Lagrange equation (1.7). Here, $\rho(x)$ plays the role of a mass density, $p(x)$ of a locally varying elasticity module. We do not specify initial and end conditions for the moment. Note that there are no *external* forces present in (1.8).

From physics we know that $\rho(x) > 0$ and $p(x) > 0$ for all x. These properties are of importance also from a mathematical view point! For simplicity, we assume that $\rho(x) = 1$.

1.2.2 The method of separation of variables

For the solution u in (1.8) we make the *ansatz*

$$u(x,t) = v(t)w(x). \tag{1.9}$$

Here, v is a function that depends only on the time t, while w depends only on the spatial variable x. With this ansatz (1.8) becomes

$$v''(t)w(x) - v(t)(p(x)w'(x))' - q(x)v(t)w(x) = 0. \tag{1.10}$$

Now we *separate* the variables depending on t from those depending on x,

$$\frac{v''(t)}{v(t)} = \frac{1}{w(x)}(p(x)w'(x))' + q(x).$$

This equation holds *for any* t and x. We can vary t and x independently of each other without changing the value on each side of the equation. Therefore, each side of the equation must be equal to a constant value. We denote this value by $-\lambda$. Thus, from the left side we obtain the equation

$$-v''(t) = \lambda v(t). \tag{1.11}$$

This equation has the well-known solution $v(t) = a \cdot \cos(\sqrt{\lambda}t) + b \cdot \sin(\sqrt{\lambda}t)$ where $\lambda > 0$ is assumed. The right side of (1.10) gives a so-called **Sturm-Liouville problem**

$$\boxed{-(p(x)w'(x))' + q(x)w(x) = \lambda w(x), \qquad w(0) = w(1) = 0.} \tag{1.12}$$

A value λ for which (1.12) has a *non-trivial* (i.e. nonzero) solution w is called an **eigenvalue**; w is a corresponding **eigenfunction**. It is known that all eigenvalues of (1.12) are positive. By means of our ansatz (1.9) we get

$$u(x,t) = w(x)\left[a \cdot \cos(\sqrt{\lambda}t) + b \cdot \sin(\sqrt{\lambda}t)\right]$$

as a solution of (1.8). It is known that (1.12) has infinitely many real positive eigenvalues $0 < \lambda_1 \le \lambda_2 \le \cdots$, $(\lambda_k \underset{k\to\infty}{\longrightarrow} \infty)$. (1.12) has a non-zero solution, say $w_k(x)$, *only* for these particular values λ_k. Therefore, the general solution of (1.8) has the form

$$u(x,t) = \sum_{k=0}^{\infty} w_k(x)\left[a_k \cdot \cos(\sqrt{\lambda_k}\,t) + b_k \cdot \sin(\sqrt{\lambda_k}\,t)\right]. \tag{1.13}$$

The coefficients a_k and b_k are determined by initial and end conditions. We could, e.g., require that

$$u(x,0) = \sum_{k=0}^{\infty} a_k w_k(x) = u_0(x),$$
$$\frac{\partial u}{\partial t}(x,0) = \sum_{k=0}^{\infty} \sqrt{\lambda_k}\, b_k w_k(x) = u_1(x),$$

where u_0 and u_1 are given functions. It is known that the w_k form an orthogonal basis in the space of square integrable functions $L_2(0,1)$,

$$\int_0^1 w_k(x)w_\ell(x)dx = \gamma_k\delta_{k\ell}.$$

Therefore, it is not difficult to compute the coefficients a_k and b_k,

$$a_k = \int_0^1 u_0(x) w_k(x) dx / \gamma_k, \qquad b_k = \int_0^1 u_1(x) w_k(x) dx / \gamma_k \sqrt{\lambda_k}.$$

In concluding, we see that the difficult problem to solve is the eigenvalue problem (1.12). Knowing the eigenvalues and eigenfunctions the general solution of the time-dependent problem (1.8) is easy to form.

Eq. (1.12) can be solved analytically only in very special situation, e.g., if all coefficients are constants. In general a *numerical method* is needed to solve the Sturm-Liouville problem (1.12).

1.3 Numerical methods for solving 1-dimensional problems

In this section we consider three methods to solve the Sturm-Liouville problem.

1.3.1 Finite differences

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

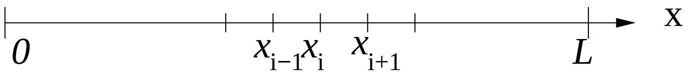


Figure 1.3: Grid points in the interval $(0, L)$.

At point x_i we approximate the derivatives by **finite differences**. 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 in 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,

$$A\mathbf{w} = \lambda \mathbf{w}. \tag{1.14}$$

By construction, A is symmetric and tridiagonal. One can show that it is positive definite as well. Note that this matrix has just a few nonzeros: out of the n^2 elements of A only $3n-2$ are nonzero. This is an example of a **sparse** matrix.

1.3.2 The finite element method

We write (1.12) in the form

> Find a twice differentiable function w with $w(0) = w(1) = 0$ such that
>
> $$\int_0^1 \left[-(p(x)w'(x))' + q(x)w(x) - \lambda w(x)\right] \phi(x)dx = 0$$
>
> *for all* smooth functions ϕ that satisfy $\phi(0) = \phi(1) = 0$.

To relax the requirements on w we integrate by parts and get the new so-called *weak* or *variational form* of the problem:

> Find a differentiable function w with $w(0) = w(1) = 0$ such that
>
> $$\int_0^1 \left[p(x)w(x)'\phi'(x) + q(x)w(x)\phi(x) - \lambda w(x)\phi(x)\right] dx = 0 \tag{1.15}$$
>
> *for all* differentiable functions ϕ that satisfy $\phi(0) = \phi(1) = 0$.

Remark: Requiring continuous differentiability is too strong and does not lead to a mathematically suitable formulation. In particular, the test functions that will be used below are not differentiable in the classical sense. It is more appropriate to require w and ϕ to be *weakly* differentiable. In terms of Sobolev spaces: $w, \phi \in H_0^1([0,1])$. An introduction to Sobolev spaces is, however, beyond the scope of these notes.

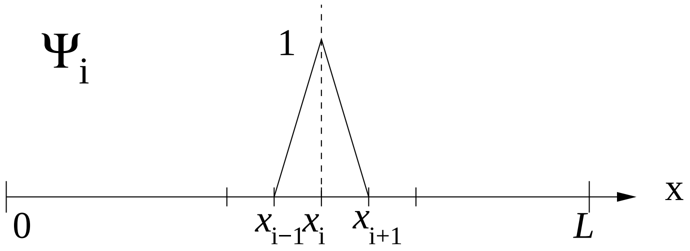


Figure 1.4: A basis function of the finite element space: a hat function.

We now write w as the linear combination

$$w(x) = \sum_{i=1}^{n} \xi_i \, \Psi_i(x), \tag{1.16}$$

where

$$\Psi_i(x) = \left(1 - \frac{|x - x_i|}{h}\right)_+ = \max\{0,\ 1 - \frac{|x - x_i|}{h}\}, \tag{1.17}$$

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 basis function, a so-called *hat function*, is displayed in Fig. 1.4.

We now replace w in (1.15) by the linear combination (1.16), and replace testing 'against all ϕ' by testing against all Ψ_j. In this way (1.15) becomes

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

or,

$$\boxed{\sum_{i=1}^n \xi_i \int_0^1 \left(p(x)\Psi_i'(x)\Psi_j'(x) + (q(x) - \lambda)\Psi_i(x)\Psi_j(x) \right) dx = 0, \quad \text{for all } j.} \tag{1.18}$$

These last equations are called the **Rayleigh–Ritz–Galerkin** equations. Unknown are the n values ξ_i and the eigenvalue λ. In matrix notation (1.18) becomes

$$A\mathbf{x} = \lambda M \mathbf{x} \tag{1.19}$$

with

$$a_{ij} = \int_0^1 \left(p(x)\Psi_i'\Psi_j' + q(x)\Psi_i\Psi_j \right) dx \quad \text{and} \quad m_{ij} = \int_0^1 \Psi_i \Psi_j \, dx$$

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

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

In the same way we get

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

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

1.3.3 Global functions

Formally we proceed as with the finite element method, i.e., we solve equation (1.18). But now we choose the $\Psi_k(x)$ to be functions with global support[1]. We could, e.g., set

$$\Psi_k(x) = \sin k\pi x,$$

[1] The support of a function f is the set of arguments x for which $f(x) \neq 0$.

functions that are differentiable and satisfy the homogeneous boundary conditions. The Ψ_k are eigenfunctions of the nearby problem $-u''(x) = \lambda u(x)$, $u(0) = u(1) = 0$ corresponding to the eigenvalue $k^2\pi^2$. The elements of matrix A are given by

$$\begin{aligned} a_{kk} &= \int_0^1 \left[(1+x)k^2\pi^2 \cos^2 k\pi x + \sin^2 k\pi x\right] dx = \frac{3}{4}k^2\pi^2 + \frac{1}{2}, \\ a_{kj} &= \int_0^1 \left[(1+x)kj\pi^2 \cos k\pi x \cos j\pi x + \sin k\pi x \sin j\pi x\right] dx \\ &= \frac{kj(k^2+j^2)((-1)^{k+j}-1)}{(k^2-j^2)^2}, \quad k \neq j. \end{aligned}$$

1.3.4 A numerical comparison

We consider the above 1-dimensional eigenvalue problem

$$-((1+x)w'(x))' + w(x) = \lambda w(x), \qquad w(0) = w(1) = 0, \tag{1.20}$$

and solve it with the finite difference and finite element methods as well as with the global functions method. The results are given in Table 1.1.

Clearly the global function method is the most powerful of them all. With 80 basis functions the eigenvalues all come right. The convergence rate is exponential.

With the finite difference and finite element methods the eigenvalues exhibit quadratic convergence rates. If the mesh width h is reduced by a factor of $q = 2$, the error in the eigenvalues is reduced by the factor $q^2 = 4$. There exist higher order finite elements and higher order finite difference stencils [11, 6].

1.4 Example 2: The heat equation

The instationary temperature distribution $u(\mathbf{x}, t)$ in an insulated container satisfies the equations

$$\begin{aligned} \frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x},t) &= 0, & \mathbf{x} \in \Omega,\ t > 0, \\ \frac{\partial u(\mathbf{x},t)}{\partial n} &= 0, & \mathbf{x} \in \partial\Omega,\ t > 0, \\ u(\mathbf{x},0) &= u_0(\mathbf{x}), & \mathbf{x} \in \Omega. \end{aligned} \tag{1.21}$$

Here Ω is a 3-dimensional domain[2] with boundary $\partial\Omega$. $u_0(\mathbf{x}), \mathbf{x} = (x_1, x_2, x_3)^T \in \mathbb{R}^3$, is a given bounded, sufficiently smooth function.

$$\Delta u = \sum \frac{\partial^2 u}{\partial {x_i}^2} \tag{1.22}$$

is called the *Laplace operator* and $\frac{\partial u}{\partial n}$ denotes the derivative of u in direction of the outer normal vector $\mathbf{n}$. To solve the heat equation the **method of separation of variables** is employed. We write u in the form

$$u(\mathbf{x}, t) = v(t)w(\mathbf{x}). \tag{1.23}$$

[2] In the sequel we understand a domain to be bounded and simply connected.

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

Global function method				
k	$\lambda_k(n = 10)$	$\lambda_k(n = 20)$	$\lambda_k(n = 40)$	$\lambda_k(n = 80)$
1	15.338	15.338	15.338	15.338
2	58.482	58.480	58.480	58.480
3	130.389	130.386	130.386	130.386
4	231.065	231.054	231.053	231.053
5	360.511	360.484	360.483	360.483
6	518.804	518.676	518.674	518.674
7	706.134	705.631	705.628	705.628
8	924.960	921.351	921.344	921.344
9	1186.674	1165.832	1165.823	1165.822
10	1577.340	1439.083	1439.063	1439.063

Table 1.1: Numerical solutions of problem (1.20)

If a constant λ can be found such that

$$\boxed{\begin{array}{lll} \Delta w(\mathbf{x}) + \lambda w(\mathbf{x}) = 0, & w(\mathbf{x}) \neq 0, & \mathbf{x} \text{ in } \Omega, \\ \dfrac{\partial w(\mathbf{x})}{\partial n} = 0, & & \mathbf{x} \text{ on } \partial\Omega, \end{array}} \tag{1.24}$$

then the product $u = vw$ is a solution of (1.21) if and only if

$$\frac{dv(t)}{dt} + \lambda v(t) = 0, \tag{1.25}$$

the solution of which has the form $a\cdot\exp(-\lambda t)$. By separating variables, the problem (1.21) is divided in two subproblems that are hopefully easier to solve. A value λ, for which (1.24) has a *nontrivial* (i.e. a nonzero) solution is called an *eigenvalue*; w then is called a *corresponding eigenfunction.*

If λ_n is an eigenvalue of problem (1.24) with corresponding eigenfunction w_n, then

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

is a solution of the first two equations in (1.21). It is known that equation (1.24) has *infinitely many* real eigenvalues $0 \leq \lambda_1 \leq \lambda_2 \leq \cdots$, that tend to infinity, $\lambda_n \longrightarrow \infty$ as $n \to \infty$. Multiple eigenvalues are counted according to their multiplicity. 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 (1.21) can be written in the form

$$u(\mathbf{x}, t) = \sum_{n=1}^{\infty} c_n e^{-\lambda_n t} w_n(\mathbf{x}), \tag{1.26}$$

where the coefficients c_n are determined such that

$$u_0(\mathbf{x}) = \sum_{n=1}^{\infty} c_n w_n(\mathbf{x}). \tag{1.27}$$

The smallest eigenvalue of (1.24) is $\lambda_1 = 0$ with $w_1 = 1$ and $\lambda_2 > 0$. Therefore we see from (1.26) that

$$u(\mathbf{x}, t) \underset{t\to\infty}{\longrightarrow} c_1. \tag{1.28}$$

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 (1.24):

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

Here we have assumed that the value of the constant function $w_1(\mathbf{x})$ is set to unity.

1.5 Example 3: The wave equation

The air pressure $u(\mathbf{x},t)$ in a volume with acoustically "hard" walls satisfies the equations

$$\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} - \Delta u(\mathbf{x},t) = 0, \qquad \mathbf{x} \in \Omega,\ t > 0, \tag{1.29}$$

$$\frac{\partial u(\mathbf{x},t)}{\partial n} = 0, \qquad \mathbf{x} \in \partial\Omega,\ t > 0, \tag{1.30}$$

$$u(\mathbf{x},0) = u_0(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{1.31}$$

$$\frac{\partial u(\mathbf{x},0)}{\partial t} = u_1(\mathbf{x}), \qquad \mathbf{x} \in \Omega. \tag{1.32}$$

Sound propagates with speed $-\nabla u$, along the (negative) gradient from high to low pressure.

To solve the wave equation we proceed as with the heat equation in section 1.4: separation of u according to (1.23) leads again to equation (1.24) but now together with

$$\frac{d^2 v(t)}{dt^2} + \lambda v(t) = 0. \tag{1.33}$$

We know this equation from the analysis of the vibrating string, see (1.11). From there we know that the general solution of the wave equation has the form

$$u(x,t) = \sum_{k=0}^{\infty} w_k(x) \left[A_k \cdot \cos(\sqrt{\lambda_k}\, t) + B_k \cdot \sin(\sqrt{\lambda_k}\, t) \right]. \tag{1.13}$$

where the w_k, $k = 1, 2, \ldots$, are the eigenfunctions of the eigenvalue problem (1.24). The coefficients a_k and b_k are determined by (1.31) and (1.32).

If a harmonic oscillation is forced on the system, an *inhomogeneous* problem

$$\frac{\partial^2 u(\mathbf{x},t)}{\partial t^2} - \Delta u(\mathbf{x},t) = f(\mathbf{x},t), \tag{1.34}$$

is obtained. The boundary and initial conditions are taken from (1.29)–(1.32). This problem can be solved by expanding u and f in the eigenfunctions $w_n(\mathbf{x})$,

$$\begin{aligned} u(\mathbf{x},t) &:= \sum_{n=1}^{\infty} \tilde{v}_n(t) w_n(\mathbf{x}), \\ f(\mathbf{x},t) &:= \sum_{n=1}^{\infty} \phi_n(t) w_n(\mathbf{x}). \end{aligned} \tag{1.35}$$

With this approach, $\tilde{v}_n$ has to satisfy equation

$$\frac{d^2 \tilde{v}_n}{dt^2} + \lambda_n \tilde{v}_n = \phi_n(t). \tag{1.36}$$

If $\phi_n(t) = a_n \sin \omega t$, then the solution becomes

$$\tilde{v}_n = A_n \cos \sqrt{\lambda_n} t + B_n \sin \sqrt{\lambda_n} t + \frac{1}{\lambda_n - \omega^2} a_n \sin \omega t. \tag{1.37}$$

A_n and B_n are real constants that are determined by the initial conditions. If ω gets close to $\sqrt{\lambda_n}$, then the last term can be very large. In the limit, if $\omega = \sqrt{\lambda_n}$, $\tilde{v}_n$ gets the form

$$\tilde{v}_n = A_n \cos\sqrt{\lambda_n}t + B_n \sin\sqrt{\lambda_n}t + a_n t \sin\omega t, \tag{1.38}$$

in which case, $\tilde{v}_n$ is not bounded in time anymore. This phenomenon is called *resonance*. Often resonance is not desirable; it may, e.g., mean the blow up of some structure. In order to prevent resonances eigenvalues have to be known. Possible remedies are changing the domain (the structure) or parameters (the materials).

Remark 1.1. Vibrating membranes satisfy the wave equation, too. In general the boundary conditions are different from (1.30). If the membrane (of a drum) is fixed at its boundary, the condition

$$u(\mathbf{x}, t) = 0 \tag{1.39}$$

is imposed. These boundary conditions are called *Dirichlet boundary conditions*. The boundary conditions in (1.21) and (1.30) are called *Neumann boundary conditions*. Combinations of these two can occur. □

1.6 Numerical methods for solving the Laplace eigenvalue problem in 2D

In this section we again consider the eigenvalue problem

$$-\Delta u(\mathbf{x}) = \lambda u(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \tag{1.40}$$

with the more general boundary conditions

$$u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Gamma_1 \subset \partial\Omega, \tag{1.41}$$

$$\frac{\partial u}{\partial n}(\mathbf{x}) + \alpha(\mathbf{x})u(\mathbf{x}) = 0, \qquad \mathbf{x} \in \Gamma_2 \subset \partial\Omega. \tag{1.42}$$

Here, Γ_1 and Γ_2 are *disjoint* subsets of $\partial\Omega$ with $\Gamma_1 \cup \Gamma_2 = \partial\Omega$. We restrict ourselves in the following 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 (1.40)–(1.42) exactly (analytically). Therefore, one has to resort to numerical approximations. Because we cannot compute with infinitely many variables we have to construct a finite-dimensional eigenvalue problem that represents the given problem as well as possible, i.e., that yields good approximations for the desired eigenvalues and eigenvectors. Since finite-dimensional eigenvalue problem only have a finite number of eigenvalues one cannot expect to get good approximations for all eigenvalues of (1.40)–(1.42).

Two methods for the discretization of eigenvalue problems of the form (1.40)–(1.42) are the *Finite Difference Method* [11, 16, 9] and the *Finite Element Method (FEM)* [6, 15, 8]. We briefly introduce these methods in the following subsections.

1.6.1 The finite difference method

In this section we just want to mediate some impression what the finite difference method is about. Therefore 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

$$\begin{aligned} -\Delta u(x,y) &= \lambda u(x,y), && 0 < x,y < 1 \\ u(0,y) = u(1,y) = u(x,0) &= 0, && 0 < x,y < 1, \\ \frac{\partial u}{\partial n}(x,1) &= 0, && 0 < x < 1. \end{aligned} \tag{1.43}$$

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 that for sufficiently smooth functions u

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

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

$$4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = \lambda h^2 u_{i,j}, \quad 0 < i,j < N. \tag{1.44}$$

We consider the unknown variables $u_{i,j}$ as approximations of the eigenfunctions at the grid points (i,j):

$$u_{i,j} \approx u(x_i, x_j). \tag{1.45}$$

The Dirichlet boundary conditions are replaced by the equations

$$u_{i,0} = u_{i,N} = u_{0,i}, \qquad 0 < i < N. \tag{1.46}$$

At the points at the upper boundary of Ω we first take the difference equation (1.44)

$$4u_{i,N} - u_{i-1,N} - u_{i+1,N} - u_{i,N-1} - u_{i,N+1} = \lambda h^2 u_{i,N}, \quad 0 \le i \le N. \tag{1.47}$$

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 (1.47) and multiplying the new equation by the factor $1/2$ gives

$$2u_{i,N} - \frac{1}{2}u_{i-1,N} - \frac{1}{2}u_{i+1,N} - u_{i,N-1} = \frac{1}{2}\lambda h^2 u_{i,N}, \quad 0 < i < N. \tag{1.48}$$

In summary, from (1.44) and (1.48), taking into account that (1.46) we get the matrix equation

$$\begin{aligned}
&\begin{pmatrix}
4 & -1 & 0 & -1 & & & & & & & & \\
-1 & 4 & -1 & 0 & -1 & & & & & & & \\
0 & -1 & 4 & 0 & 0 & -1 & & & & & & \\
-1 & 0 & 0 & 4 & -1 & 0 & -1 & & & & & \\
 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & & & & \\
 & & -1 & 0 & -1 & 4 & 0 & 0 & -1 & & & \\
 & & & -1 & 0 & 0 & 4 & -1 & 0 & -1 & & \\
 & & & & -1 & 0 & -1 & 4 & -1 & 0 & -1 & \\
 & & & & & -1 & 0 & -1 & 4 & 0 & 0 & -1 \\
 & & & & & & -1 & 0 & 0 & 2 & -\frac{1}{2} & 0 \\
 & & & & & & & -1 & 0 & -\frac{1}{2} & 2 & -\frac{1}{2} \\
 & & & & & & & & -1 & 0 & -\frac{1}{2} & 2
\end{pmatrix}
\begin{pmatrix}
u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3}
\end{pmatrix} \\
&= \lambda h^2
\begin{pmatrix}
1 & & & & & & & & & & & \\
 & 1 & & & & & & & & & & \\
 & & 1 & & & & & & & & & \\
 & & & 1 & & & & & & & & \\
 & & & & 1 & & & & & & & \\
 & & & & & 1 & & & & & & \\
 & & & & & & 1 & & & & & \\
 & & & & & & & 1 & & & & \\
 & & & & & & & & 1 & & & \\
 & & & & & & & & & \frac{1}{2} & & \\
 & & & & & & & & & & \frac{1}{2} & \\
 & & & & & & & & & & & \frac{1}{2}
\end{pmatrix}
\begin{pmatrix}
u_{1,1} \\ u_{1,2} \\ u_{1,3} \\ u_{2,1} \\ u_{2,2} \\ u_{2,3} \\ u_{3,1} \\ u_{3,2} \\ u_{3,3} \\ u_{4,1} \\ u_{4,2} \\ u_{4,3}
\end{pmatrix}.
\end{aligned} \tag{1.49}$$

For arbitrary $N > 1$ we define

$$\mathbf{u}_i := \begin{pmatrix} u_{i,1} \\ u_{i,2} \\ \vdots \\ u_{i,N-1} \end{pmatrix} \in \mathbb{R}^{N-1},$$

$$T := \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \in \mathbb{R}^{(N-1)\times(N-1)},$$

$$I := \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} \in \mathbb{R}^{(N-1)\times(N-1)}.$$

In this way we obtain from (1.44), (1.46), (1.48) the discrete eigenvalue problem

$$\begin{pmatrix} T & -I & & \\ -I & T & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & \frac{1}{2}T \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_3 \\ \mathbf{u}_4 \end{pmatrix} = \lambda h^2 \begin{pmatrix} I & & & \\ & \ddots & & \\ & & I & \\ & & & \frac{1}{2}I \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{N-1} \\ \mathbf{u}_N \end{pmatrix} \tag{1.50}$$

of size $N \times (N-1)$. This is a **matrix eigenvalue problem** of the form

$$A\mathbf{x} = \lambda M\mathbf{x}, \tag{1.51}$$

where A and M are **symmetric** and M additionally is **positive definite**. If M is the identity matrix, we call (1.51) a *special* and otherwise a *generalized* eigenvalue problem. In these lecture notes we deal with numerical methods, to solve eigenvalue problems like these.

In the case (1.50) it is easy to obtain a special (symmetric) eigenvalue problem by a simple transformation: By left multiplication by

$$\begin{pmatrix} I & & & \\ & I & & \\ & & I & \\ & & & \sqrt{2}I \end{pmatrix}$$

we obtain from (1.50)

$$\begin{pmatrix} T & -I & & \\ -I & T & -I & \\ & -I & T & -\sqrt{2}I \\ & & -\sqrt{2}I & T \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \frac{1}{\sqrt{2}}\mathbf{u}_4 \end{pmatrix} = \lambda h^2 \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \frac{1}{\sqrt{2}}\mathbf{u}_4 \end{pmatrix}. \tag{1.52}$$

A property common to matrices obtained by the finite difference method are its **sparsity**. Sparse matrices have only very few nonzero elements.

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

Nevertheless, problems that are posed on rectangular grids can be solved very efficiently. Therefore, tricks are used to deal with irregular boundaries. The solution of the problem may be extended artificially beyond the boundary, see e.g. [1, 17, 9]. Similar techiques, so-called *immersed boundary conditions* are applied at (irregular) interfaces where, e.g., equations or parameters change [11].

1.6.2 The finite element method (FEM)

Let $(\lambda, u) \in \mathbb{R} \times V$ be an eigenpair of problem (1.40)–(1.42). Then

$$\int_\Omega (\Delta u + \lambda u)v \, dx \, dy = 0, \quad \forall v \in V, \tag{1.53}$$

where V is vector space of bounded twice differentiable functions that satisfy the boundary conditions (1.41)–(1.42). By partial integration (Green's formula) this becomes

$$\int_\Omega \nabla u \nabla v \, dx \, dy + \int_{\Gamma_2} \alpha \, u \, v \, ds = \lambda \int_\Omega u \, v \, dx \, dy, \quad \forall v \in V, \tag{1.54}$$

or

$$a(u,v) = (u,v), \qquad \forall v \in V \tag{1.55}$$

where

$$a(u,v) = \int\limits_{\Omega} \nabla u \, \nabla v \, dx \, dy + \int\limits_{\Gamma_2} \alpha \, u \, v \, ds, \quad \text{and} \quad (u,v) = \int_{\Omega} u \, v \, dx \, dy.$$

We complete the space V with respect to the Sobolev norm [8, 3]

$$\sqrt{\int_{\Omega} (u^2 + |\nabla u|^2) \, dx \, dy}$$

to become a Hilbert space H [3, 19]. H is the space of quadratic integrable functions with quadratic integrable first derivatives that satisfy the Dirichlet boundary conditions (1.41)

$$u(x,y) = 0, \qquad (x,y) \in \Gamma_1.$$

(Functions in H in general do not satisfy the so-called *natural* boundary conditions (1.42).) One can show [19] that the eigenvalue problem (1.40)–(1.42) is equivalent with the eigenvalue problem

$$\begin{array}{l} \text{Find } (\lambda, u) \in \mathbb{R} \times H \text{ such that} \\ a(u,v) = \lambda(u,v) \quad \forall v \in H. \end{array} \tag{1.56}$$

(The essential point is to show that the eigenfunctions of (1.56) are elements of V.)

The Rayleigh–Ritz–Galerkin method

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

$$\phi_1(x,y), \cdots, \phi_n(x,y) \in H, \tag{1.57}$$

are chosen. These functions span a *subspace* S of H. Then, problem (1.56) is solved where H is replaced by S.

$$\boxed{\begin{array}{l} \text{Find } (\lambda, u) \in \mathbb{R} \times S \text{ such that} \\ a(u,v) = \lambda(u,v) \quad \forall v \in S. \end{array}} \tag{1.58}$$

With the Ritz ansatz [15]

$$u = \sum_{i=1}^{n} x_i \phi_i, \tag{1.59}$$

equation (1.58) becomes

$$\begin{array}{l} \text{Find } (\lambda, \mathbf{x}) \in \mathbb{R} \times \mathbb{R}^n \text{ such that} \\ \sum\limits_{i=1}^{n} x_i a(\phi_i, v) = \lambda \sum\limits_{i=1}^{n} x_i (\phi_i, v), \quad \forall v \in S. \end{array} \tag{1.60}$$

Eq. (1.60) must hold *for all* $v \in S$, in particular for $v = \phi_1, \cdots, \phi_n$. But since the $\phi_i, 1 \le i \le n$, form a basis of S, equation (1.60) is equivalent with

$$\sum_{i=1}^{n} x_i a(\phi_i, \phi_j) = \lambda \sum_{i=1}^{n} x_i (\phi_i, \phi_j), \quad 1 \le j \le n. \tag{1.61}$$

This is a matrix eigenvalue problem of the form

$$A\mathbf{x} = \lambda M \mathbf{x} \tag{1.62}$$

where

$$\mathbf{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \quad A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{pmatrix}, \quad M = \begin{pmatrix} m_{11} & \cdots & m_{1n} \\ \vdots & \ddots & \vdots \\ m_{n1} & \cdots & m_{nn} \end{pmatrix} \tag{1.63}$$

with

$$a_{ij} = a(\phi_i, \phi_j) = \int_\Omega \nabla\phi_i \, \nabla\phi_j \, dx \, dy + \int_{\Gamma_2} \alpha \, \phi_i \, \phi_j \, ds$$

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 S and in particular the basis $\{\phi_i\}$ is 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. Fig 1.5. (This means that the boundary is composed entirely of straight line segments.) This domain is now partitioned into triangular subdomains

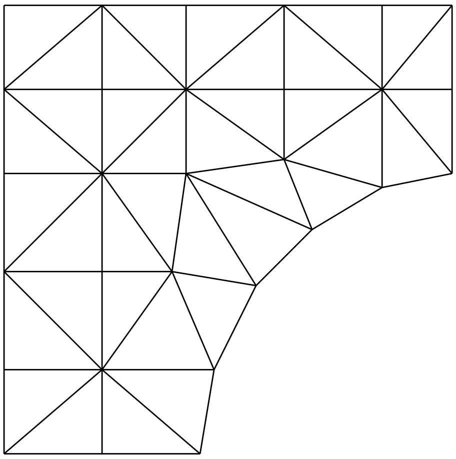


Figure 1.5: Triangulation of a domain Ω

$T_1, \cdots, T_N$, so-called *elements*, such that

$$T_i \cap T_j = \emptyset \quad \text{for all } i \ne j, \quad \text{and} \quad \bigcup_e \overline{T_e} = \overline{\Omega}. \tag{1.64}$$

Finite element spaces for solving (1.40)–(1.42) 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*. Notice that this construction provides a subspace of the Hilbert space H but not of V, i.e., the functions in the finite element space are not very smooth and the natural boundary conditions are not satisfied.

An essential issue is the selection of the *basis* of the finite element space S. If $S_1 \subset H$ 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 Γ_1, be numbered from 1 to n, see Fig. 1.6. Let the coordinates of the i-th node be (x_i, y_i). Then $\phi_i(x, y) \in S_1$ is defined by

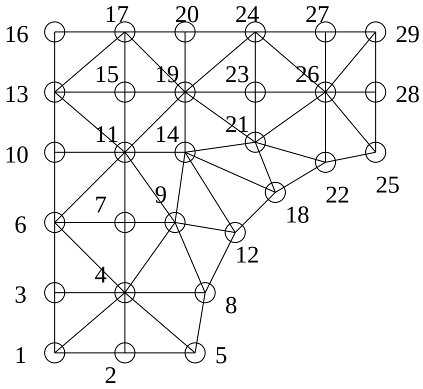


Figure 1.6: Numbering of nodes on Ω (piecewise linear polynomials)

$$\phi_i((x_j, y_j)) := \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \tag{1.65}$$

A typical basis function ϕ_i is sketched in Figure 1.7.

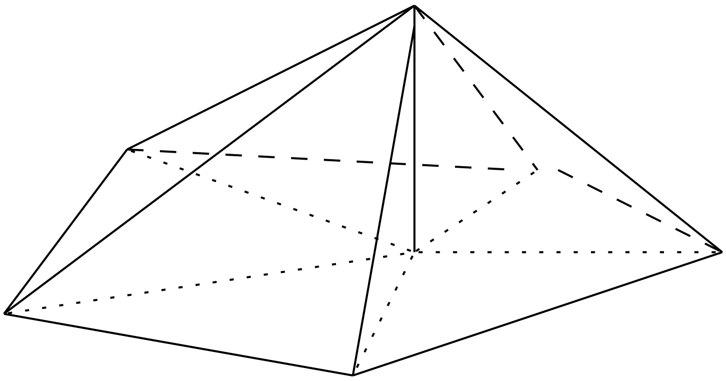


Figure 1.7: A piecewise linear basis function (or hat function)

Another often used finite element element space is $S_2 \subset H$, 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 (1.65). There are two kinds of basis functions ϕ_i now, first those that are 1 at a vertex and second those that are 1 at an edge midpoint, cf. Fig. 1.8. One immediately sees that for most $i \neq j$

$$a(\phi_i, \phi_j) = 0, \quad (\phi_i, \phi_j) = 0. \tag{1.66}$$

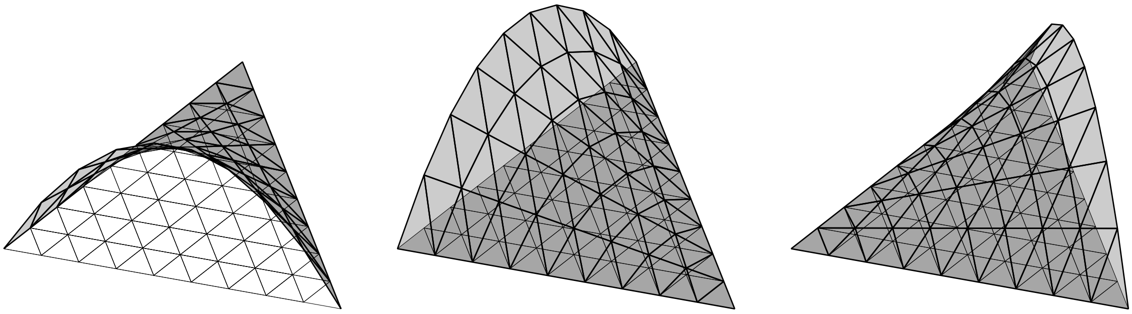


Figure 1.8: The piecewise quadratic basis functions corresponding to the edge midpoints [5]

Therefore the matrices A and M in (1.62)–(1.63) will be **sparse**. The matrix M is positive definite as

$$\mathbf{x}^T M\mathbf{x} = \sum_{i,j=1}^{N} x_i x_j m_{ij} = \sum_{i,j=1}^{N} x_i x_j (\phi_i, \phi_j) = (u,u) > 0, \quad u = \sum_{i=1}^{N} x_i \phi_i \neq 0, \tag{1.67}$$

because the ϕ_i are linearly independent and because $||u|| = \sqrt{(u,u)}$ is a norm. Similarly it is shown that

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

It is possible to have $\mathbf{x^T A x} = 0$ for a nonzero vector $\mathbf{x}$. This is the case if the constant function $u = 1$ is contained in S. This happens 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 $\mathbf{x^T A x} = 0$ for $\mathbf{x} = [1, 1, \ldots, 1]$.

1.6.3 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 (1.24) together with Neumann boundary conditions. The domain is given in Fig. 1.9 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 polynomials are listed in Table 1.2. From the results we notice the quadratic convergence 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. 1.10. 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 noticable.

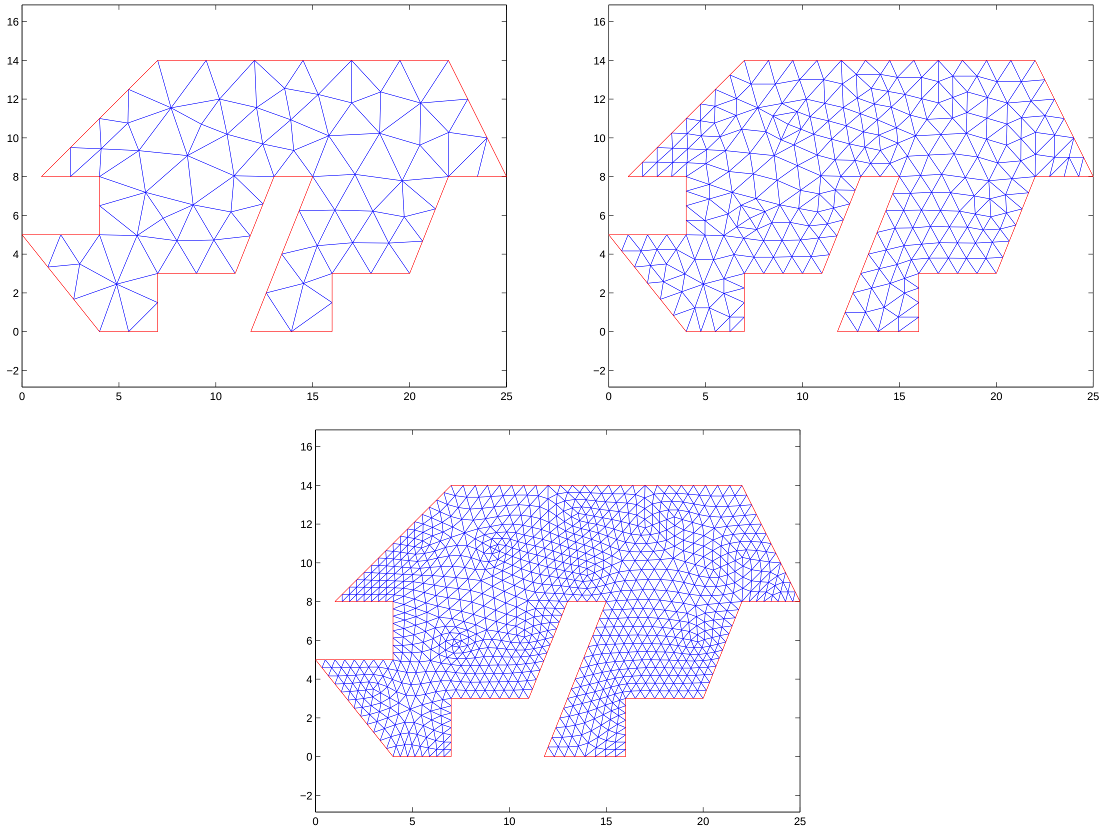


Figure 1.9: Three meshes for the car length cut

1.7 Cavity resonances in particle accelerators

The Maxwell equations in vacuum are given by

$$\begin{aligned}
\mathbf{curl}\,\mathbf{E}(\mathbf{x},t) &= -\frac{\partial \mathbf{B}}{\partial t}(\mathbf{x},t), && \text{(Faraday's law)}\\
\mathbf{curl}\,\mathbf{H}(\mathbf{x},t) &= \frac{\partial \mathbf{D}}{\partial t}(\mathbf{x},t) + \mathbf{j}(\mathbf{x},t), && \text{(Maxwell–Ampère law)}\\
\operatorname{div}\mathbf{D}(\mathbf{x},t) &= \rho(\mathbf{x},t), && \text{(Gauss's law)}\\
\operatorname{div}\mathbf{B}(\mathbf{x},t) &= 0. && \text{(Gauss's law – magnetic)}
\end{aligned}$$

where $\mathbf{E}$ is the electric field intensity, $\mathbf{D}$ is the electric flux density, $\mathbf{H}$ is the magnetic field intensity, $\mathbf{B}$ is the magnetic flux density, $\mathbf{j}$ is the electric current density, and ρ is the electric charge density. Often the “optical” problem is analyzed, i.e. the situation when the cavity is not driven (cold mode), hence $\mathbf{j}$ and ρ are assumed to vanish.

Again by separating variables, i.e. assuming a *time harmonic* behavior of the fields, e.g.,

$$\mathbf{E}(\mathbf{x},t) = \mathbf{e}(\mathbf{x})e^{i\omega t},$$

and by using the constitutive relations

$$\mathbf{D} = \varepsilon\mathbf{E}, \quad \mathbf{B} = \mu\mathbf{H}, \quad \mathbf{j} = \sigma\mathbf{E},$$

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

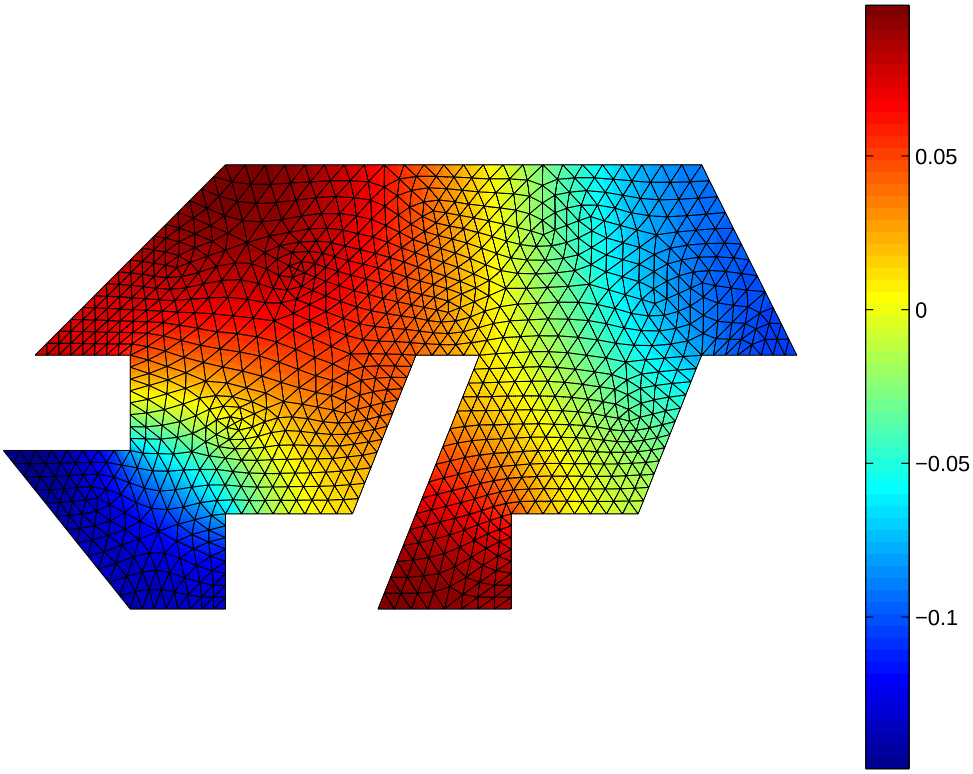


Figure 1.10: Fourth eigenmode of the acoustic vibration problem

one obtains after elimination of the magnetic field intensity the so called **time-harmonic Maxwell equations**

$$
\begin{aligned}
\mathbf{curl}\,\mu^{-1}\mathbf{curl}\,\mathbf{e}(\mathbf{x}) &= \lambda\,\varepsilon\,\mathbf{e}(\mathbf{x}), & \mathbf{x} &\in \Omega,\\
\operatorname{div}\varepsilon\,\mathbf{e}(\mathbf{x}) &= 0, & \mathbf{x} &\in \Omega,\\
\mathbf{n}\times\mathbf{e} &= 0, & \mathbf{x} &\in \partial\Omega.
\end{aligned}
\tag{1.68}
$$

Here, additionally, the cavity boundary $\partial\Omega$ is assumed to be *perfectly electrically conducting*, i.e. $\mathbf{E}(\mathbf{x},t)\times\mathbf{n}(\mathbf{x})=\mathbf{0}$ for $\mathbf{x}\in\partial\Omega$.

The eigenvalue problem (1.68) is a *constrained eigenvalue problem.* Only functions are taken into account that are divergence-free. This constraint is enforced by Lagrange multipliers. A weak formulation of the problem is then

Find $(\lambda,\mathbf{e},p)\in\mathbb{R}\times H_0(\mathbf{curl};\Omega)\times H_0^1(\Omega)$ *such that* $\mathbf{e}\neq\mathbf{0}$ *and*

(a) $(\mu^{-1}\mathbf{curl}\,\mathbf{e},\mathbf{curl}\,\mathbf{\Psi})+(\mathbf{grad}\,p,\mathbf{\Psi})=\lambda(\varepsilon\,\mathbf{e},\mathbf{\Psi}),\qquad \forall\mathbf{\Psi}\in H_0(\mathbf{curl};\Omega),$

(b) $(\mathbf{e},\mathbf{grad}\,q)=0,\qquad \forall q\in H_0^1(\Omega).$

With the correct finite element discretization this problem turns in a matrix eigenvalue problem of the form

$$
\begin{bmatrix} A & C\\ C^T & O\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}=\lambda\begin{bmatrix} M & O\\ O & O\end{bmatrix}\begin{bmatrix}\mathbf{x}\\ \mathbf{y}\end{bmatrix}.
$$

The solution of this matrix eigenvalue problem correspond to vibrating electric fields. A possible shape of domain Ω is given in Figure 1.11.

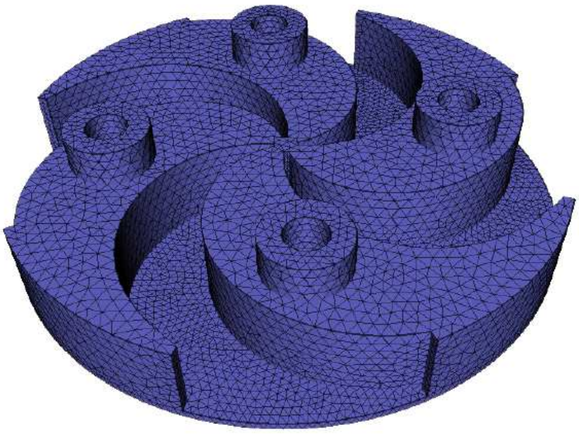


Figure 1.11: Comet cavity of Paul Scherrer Institute

1.8 Spectral clustering[3]

The goal of *clustering* is to group a given set of data points $\mathbf{x}_1,\ldots,\mathbf{x}_n$ 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.

[3] This section is based on a tutorial by von Luxburg [12]. Thanks to Daniel Kressner for compiling it!

A popular approach to clustering is based on *similarity graphs*. For this purpose, we need to assume some notion of similarity $s(\mathbf{x}_i, \mathbf{x}_j) \geq 0$ between pairs of data points $\mathbf{x}_i$ and $\mathbf{x}_j$. An undirected graph $G = (V, E)$ is constructed such that its vertices correspond to the data points: $V = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$. Two vertices $\mathbf{x}_i, \mathbf{x}_j$ are connected by an edge if the similarity s_{ij} between $\mathbf{x}_i$ and $\mathbf{x}_j$ 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 $w_{ij} = s(\mathbf{x}_i, \mathbf{x}_j)$. 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 $s(\mathbf{x}_i, \mathbf{x}_j) = \exp\big(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\big)$, where $\|\mathbf{x}_i - \mathbf{x}_j\|$ is some distance measure (e.g., Euclidean distance) and σ is some parameter controlling how strongly locality is enforced.

k-nearest neighbors Two vertices $\mathbf{x}_i, \mathbf{x}_j$ are connected if $\mathbf{x}_i$ is among the k-nearest neighbors of $\mathbf{x}_j$ or if $\mathbf{x}_j$ is among the k-nearest neighbors of $\mathbf{x}_i$ (in the sense of some distance measure). The weight of the edge between connected vertices $\mathbf{x}_i, \mathbf{x}_j$ is set to the similarity function $s(\mathbf{x}_i, \mathbf{x}_j)$.

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

Assuming that the similarity function is symmetric ($s(\mathbf{x}_i, \mathbf{x}_j) = s(\mathbf{x}_j, \mathbf{x}_i)$ for all $\mathbf{x}_i, \mathbf{x}_j$) 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.

1.8.1 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 $\mathbf{x}_i$ is defined as

$$d_i = \sum_{j=1}^{n} w_{ij}. \tag{1.69}$$

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

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

The *graph Laplacian* is then defined as

$$L = D - W. \tag{1.70}$$

By (1.69), 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 $\{\mathbf{x}_1, \dots, \mathbf{x}_k\}$, $\{\mathbf{x}_{k+1}, \dots, \mathbf{x}_n\}$, 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.

The observation above leads to the basic idea behind spectral graph partitioning: If the vertices of the graph decompose into k connected components $V_1, \dots, V_k$ there are k zero eigenvalues and the associated invariant subspace is spanned by the vectors

$$\chi_{V_1}, \chi_{V_2}, \dots, \chi_{V_k}, \tag{1.71}$$

where χ_{V_j} is the indicator vector having a 1 at entry i if $\mathbf{x}_i \in V_j$ and 0 otherwise.

1.8.2 Spectral clustering

On a first sight, it may seem that (1.71) solves the graph clustering problem. One simply computes the eigenvectors belonging to the k zero eigenvalues of the graph Laplacian and the zero structure (1.71) of the eigenvectors can be used to determine the vertices belonging to each component. Each component gives rise to a cluster.

This tempting idea has two flaws. First, one cannot expect the eigenvectors to have the structure (1.71). Any computational method will yield an arbitrary eigenbasis, e.g., arbitrary linear combinations of $\chi_{V_1}, \chi_{V_2}, \dots, \chi_{V_k}$. In general, the method will compute an orthonormal basis U with

$$U = \big(\mathbf{v}_1, \dots, \mathbf{v}_k\big)Q, \tag{1.72}$$

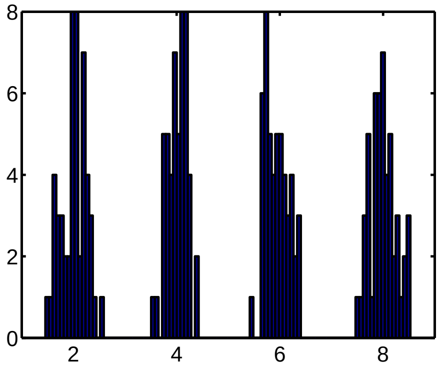
where Q is an arbitrary orthogonal $k \times k$ matrix and $\mathbf{v}_j = \chi_{V_j}/|V_j|$ with the cardinality $|V_j|$ of V_j. Second and more importantly, the goal of graph clustering is not to detect connected components of a graph[4]. Requiring the components to be completely disconnected from each other is too strong and will usually not lead to a meaningful clustering. For example, when using a fully connected similarity graph only one eigenvalue will be zero and the corresponding eigenvector $\mathbf{e}$ yields one component, which is the graph itself! Hence, instead of computing an eigenbasis belonging to zero eigenvalues, one determines an eigenbasis belonging to the k smallest eigenvalues.

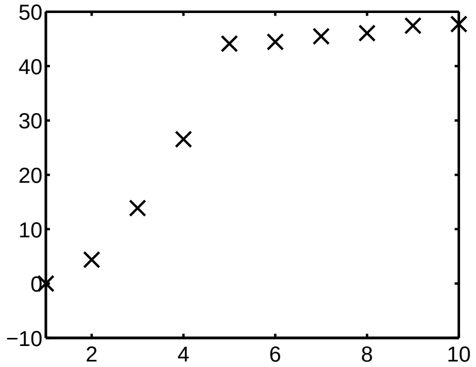
Example 1.1 200 real numbers are generated by superimposing samples from 4 Gaussian distributions with 4 different means:

```
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];
```

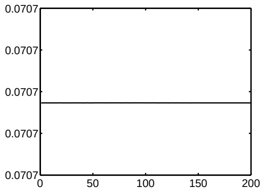
The following two figures show the histogram of the distribution of the entries of $\mathbf{x}$ and the eigenvalues of the graph Laplacian for the fully connected similarity graph with similarity function $s(\mathbf{x}_i, \mathbf{x}_j) = \exp\big(-\frac{|\mathbf{x}_i - \mathbf{x}_j|^2}{2}\big)$:

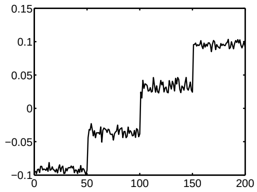
[4] There are more efficient algorithms for finding connected components, e.g., breadth-first and depth-first search.

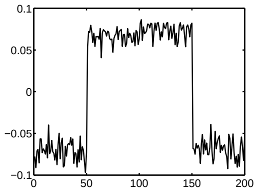


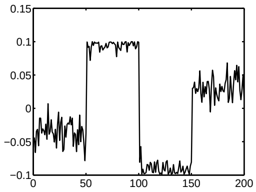


As expected, one eigenvalue is (almost) exactly zero. Additionally, the four smallest eigenvalues have a clearly visible gap to the other eigenvalues. The following four figures show the entries of the 4 eigenvectors belonging to the 4 smallest eigenvalues of L:









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 that the eigenbasis (1.72) may be transformed by an arbitrary orthogonal matrix, we "transpose" the basis and consider the row vectors of U:

$$U^T = (u_1, u_2, \dots, u_n), \quad u_i \in \mathbb{R}^k.$$

If U contained indicator vectors then each of the short vectors u_i would be a unit vector e_j for some $1 \le j \le k$ (possibly divided by $|V_j|$). In particular, the u_i would separate very well into k different clusters. The latter property does not change if the vectors u_i undergo an orthogonal transformation Q^T. Hence, applying a clustering algorithm to $u_1, \dots, u_n$ allows us to detect the membership of u_i independent of the orthogonal transformation. The key point is that the short vectors $u_1, \dots, u_n$ are much better separated than the original data $\mathbf{x}_1, \dots, \mathbf{x}_n$. 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 u_i randomly[5] to a cluster ℓ with $1 \le \ell \le k$ and then iteratively proceeds as follows:

1. Compute cluster centers c_ℓ as cluster means:

$$c_\ell = \sum_{i \text{ in cluster } \ell} u_i \Big/ \sum_{i \text{ in cluster } \ell} 1.$$

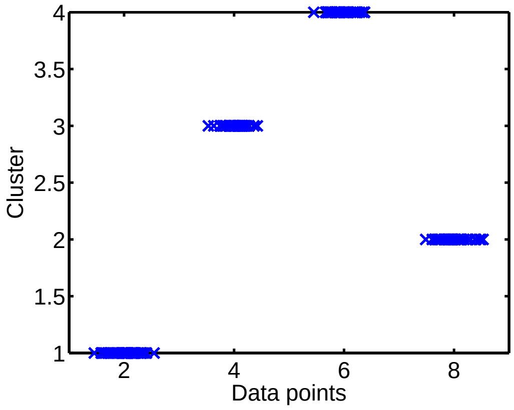
2. Assign each u_i to the cluster with the nearest cluster center.

3. Goto Step 1.

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

[5] 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.

Example 1.1 (cont'd). The k-means algorithm applied to the eigenbasis from Example 1.1 converges after 2 iterations and results in the following clustering:



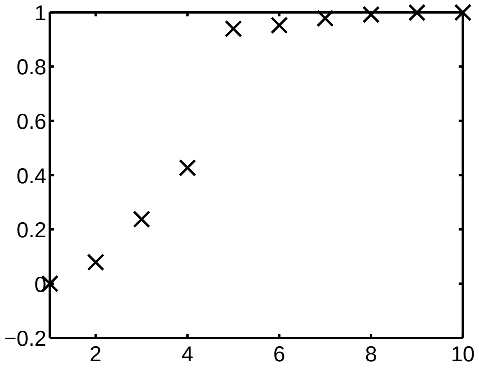
1.8.3 Normalized graph Laplacians

It is sometimes advantageous to use a normalized Laplacian

$$D^{-1}L = I - D^{-1}W \tag{1.73}$$

instead of the standard Laplacians. Equivalently, this means that we compute the eigenvectors belonging to the smallest eigenvalues of the generalized eigenvalue problem $W\mathbf{x} = \lambda D\mathbf{x}$. Alternatively, one may also compute the eigenvalues from the symmetric matrix $D^{-1/2}WD^{-1/2}$ but the eigenvectors need to be adjusted to compensate this transformation.

Example 1.1 (cont'd). The eigenvalues of the normalized Laplacian for Example 1.1 are shown below:



In comparison to the eigenvalues of the standard Laplacian, the four smallest eigenvalues of the are better separated from the rest. Otherwise, the shape of the eigenvectors is similar and the resulting clustering is identical with the one obtained with the standard Laplacian.

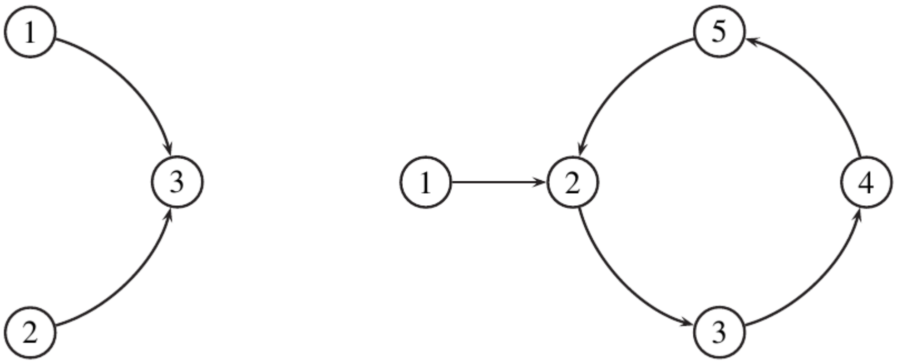


Figure 1.12: Things that can go wrong with the basic model: left is a dangling node, right a terminal strong component featuring a cyclic path. Figures are from [2]

1.9 Google's PageRank[6]

One of the reasons why Google is such an effective search engine is the *PageRank* that determines the importance of a web page [2, 10, 13]. The PageRank is determined entirely by the link structure of the World Wide Web. For any particular query, Google finds the pages on the Web that match that query and lists those pages in the order of their PageRank. Let's imagine a surfer brachiate through pages of the world wide web *randomly* choosing an outgoing link from one page to get to the next. This can lead to dead ends at pages with no outgoing links, or cycles around cliques of interconnected pages. So, every once in a while, simply choose a random page from the Web. This theoretical random walk is known as a *Markov chain* or *Markov process*. The limiting probability that an infinitely dedicated random surfer visits any particular page is its PageRank. A page has high rank if other pages with high rank link to it.

Let W be the set of (reachable) web pages and let $n = |W|$. On `WorldWideWebSize.com`[7] it is estimated that Google's index contains around to 49.5 billion pages.

The elements of the *connectivity matrix* $G \in \mathbb{R}^{n\times n}$ is defined by

$$g_{ij} = \begin{cases} 1 & \text{there is a hyperlink } j \mapsto i, \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, this is an extremely sparse matrix. The number of its nonzero elements `nnz(G)` equals the number of hyperlinks in W. Let r_i and c_j be the row and column sums of G,

$$r_i = \sum_j g_{ij}, \qquad c_j = \sum_i g_{ij}.$$

Then r_j is called the *in-degree* and c_j is called the *out-degree* of the jth page. $c_j = 0$ means a dead end.

In Fig. 1.13 we see the example of a tiny web with just $n = 6$ nodes. The nodes α, β, γ, δ, ρ, σ correspond to labels 1 to 6 in the matrix notation, in this sequence.

[6] Here we closely follow Section 2.11 in Moler's MATLAB introduction [13].

[7] http://www.worldwidewebsize.com/ accessed on Feb. 18, 2016.

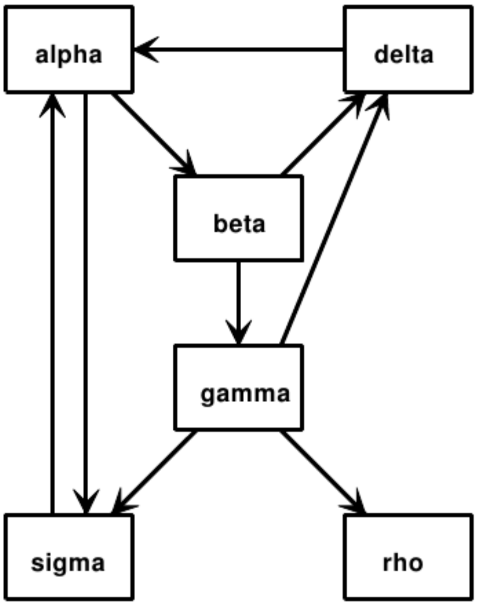


Figure 1.13: A small web with 6 nodes.

Then the connectivity matrix for the small web is given by

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

Notice the zero 5th column of G. This column corresponds to the dead end at the dangling node ρ.

Let A be the matrix with elements

$$a_{ij} = \begin{cases} g_{ij}/c_j & \text{if } c_j \neq 0 \\ 1/n & \text{if } c_j = 0 \text{ (dead end).} \end{cases}$$

In the small web example above,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & \frac{1}{6} & 1 \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{6} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{6} & 0 \\ 0 & \frac{1}{2} & \frac{1}{3} & 0 & \frac{1}{6} & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{6} & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 & \frac{1}{6} & 0 \end{bmatrix}.$$

The entries in A's column j indicate the probabilities of jumping from the jth page to the other pages on the web. Column 3, e.g., tells that starting from node 3 ($= \gamma$) nodes δ, ρ, σ are chosen with equal probability 1/3. Note that we choose any page of the web with equal probability when we land at a dead end.

To not be stuck to much in parts of the web, we follow the links only with probability α. With probability $1-\alpha$ we choose a random page. Therefore, we replace A by the matrix

$$\tilde{A} = \alpha A + (1-\alpha)\mathbf{p}\mathbf{e}^T,$$

where $\mathbf{p}$ is a *personalization* vector and $\mathbf{e} = (1,1,\ldots,1)^T$. ($\mathbf{p}$ has nonnegative elements that sum to 1, $\|\mathbf{p}\|_1 = 1$.) Note that $\mathbf{p}$ may have zero entries indicating, e.g., uncongenial, discredited, or discriminated web pages. We assume an innocent web and set $\mathbf{p} = \mathbf{e}/n$. Since $n \approx 5 \cdot 10^{10}$ in the real WWW, a typical entry of $\mathbf{p}$ is about $2 \cdot 10^{-11}$.

Note that

$$\mathbf{e}^T \tilde{A} = \mathbf{e}^T.$$

So, $1 \in \sigma(A^T) = \sigma(A)$, i.e., 1 is an eigenvalue of A with left eigenvector $\mathbf{e}$. Since the matrix norm

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}| = 1,$$

A cannot have an eigenvalue larger than 1 in modulus. The *Perron–Frobenius theorem* for matrices with nonnegative entries states that such matrices have a simple real eigenvalue of largest modulus [4]. Therefore, the eigenvalue 1 is in fact the largest eigenvalue of A. We are not interested in the left eigenvector $\mathbf{e}$ but in the right eigenvector $\mathbf{x}$,

$$\mathbf{x} = \tilde{A}\mathbf{x}.$$

The Perron–Frobenius theory confirms that $\mathbf{x}$ can be chosen such that all its entries are nonnegative. If $\mathbf{x}$ is scaled such that

$$\sum_{i=1}^{n} x_i = 1$$

then $\mathbf{x}$ is the state vector of the Markov chain and is Google's PageRank.

The computation of the PageRank amounts to determining the largest eigenvalue and corresponding eigenvector of a matrix. It can be determined by vector iteration. The computation gets easier the smaller the damping factor α is chosen. However, small α means small weight is given to the structure of the web. In [13] a Matlab routine `pagerankpow.m` is provided to compute the PageRank exploiting the sparsity structure of G.

1.10 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 the smallest 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. The principal component analysis (PCA), which plays an important role in a large variety of applications, requires the computation of the largest eigenvalues (or rather singular values). 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). It should also guide the choice of algorithm for solving an eigenvalue problem.

Saad [14] discusses further interesting sources of eigenvalue problems like electronic structure calculations, the stability of dynamical systems, or Markov chain models similar as Google's PageRank.

Bibliography

[1] A. Adelmann, P. Arbenz, and Y. Ineichen, *A fast parallel Poisson solver on irregular domains applied to beam dynamics simulations*, J. Comput. Phys., 229 (2010), pp. 4554–4566.

[2] U. M. Ascher and C. Greif, *A First Course in Numerical Methods*, SIAM, Philadelphia, PA, 2011.

[3] O. Axelsson and V. Barker, *Finite Element Solution of Boundary Value Problems*, Academic Press, Orlando, FL, 1984.

[4] A. Berman and R. J.Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979. (Republished by SIAM, Philadelphia, 1994.).

[5] O. Chinellato, *The complex-symmetric Jacobi–Davidson algorithm and its application to the computation of some resonance frequencies of anisotropic lossy axisymmetric cavities*, PhD Thesis No. 16243, ETH Zürich, 2005. (doi:10.3929/ethz-a-005067691).

[6] P. G. Ciarlet, *The Finite Element Method for Elliptic Problems*, North-Holland, Amsterdam, 1978. (Studies in Mathematics and its Applications, 4).

[7] R. Courant and D. Hilbert, *Methoden der Mathematischen Physik*, Springer, Berlin, 1968.

[8] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford University Press, Oxford, 2005.

[9] G. E. Forsythe and W. R. Wasow, *Finite-difference methods for partial differential equations*, Wiley, New York, 1960.

[10] A. N. Langville and C. D. Meyer, *Google's pagerank and beyond: the science of search engine rankings*, Princeton University Press, Princeton, N.J., 2006.

[11] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, SIAM, Philadelphia, PA, 2007.

[12] U. von Luxburg, *A tutorial on spectral clustering*, Stat. Comput., 17 (2007), pp. 395–416.

[13] C. B. Moler, *Numerical Computing with* Matlab, SIAM, Philadelphia, PA, 2004.

[14] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, SIAM, Philadelphia, PA, 2011.

[15] H. R. Schwarz, *Methode der finiten Elemente*, Teubner, Stuttgart, 3rd ed., 1991.

[16] ——, *Numerische Mathematik*, Teubner, Stuttgart, 3rd ed. ed., 1993.

[17] G. H. Shortley and R. Weller, *The numerical solution of Laplace's equation*, J. Appl. Phys., 9 (1939), pp. 334–344.

[18] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, 1986.

[19] H. F. Weinberger, *Variational Methods for Eigenvalue Approximation*, Regional Conference Series in Applied Mathematics 15, SIAM, Philadelphia, PA, 1974.

Chapter 2

Basics

2.1 Notation

The fields of real and complex numbers are denoted by $\mathbb{R}$ and $\mathbb{C}$, respectively. Elements in $\mathbb{R}$ and $\mathbb{C}$, *scalars*, are denoted by lowercase letters, a, b, $c, \ldots$, and α, β, $\gamma, \ldots$

Vectors are denoted by boldface lowercase letters, $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}, \ldots$, and $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $\boldsymbol{\gamma}$, ... We denote the space of vectors of n *real* components by $\mathbb{R}^n$ and the space of vectors of n *complex* components by $\mathbb{C}^n$.

$$\mathbf{x} \in \mathbb{R}^n \Longleftrightarrow \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad x_i \in \mathbb{R}. \tag{2.1}$$

We often make statements that hold for real or complex vectors or matrices. Then we write, e.g., $\mathbf{x} \in \mathbb{F}^n$.

The **inner product** of two n-vectors in $\mathbb{C}$ is defined as

$$(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} x_i \bar{y}_i = \mathbf{y}^* \mathbf{x}, \tag{2.2}$$

that is, we require linearity in the first component and anti-linearity in the second.

$\mathbf{y}^* = (\bar{y}_1,\ \bar{y}_2, \ldots, \bar{y}_n)$ denotes conjugate transposition of complex vectors. To simplify notation we denote real transposition by an asterisk as well.

Two vectors $\mathbf{x}$ and $\mathbf{y}$ are called **orthogonal**, $\mathbf{x} \perp \mathbf{y}$, if $\mathbf{x}^*\mathbf{y} = 0$.

The inner product (2.2) induces a **norm** in $\mathbb{F}$,

$$\|\mathbf{x}\| = \sqrt{(\mathbf{x}, \mathbf{x})} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2}. \tag{2.3}$$

This norm is often called Euclidean norm or 2-norm.

The set of m-by-n **matrices** with components in the field $\mathbb{F}$ is denoted by $\mathbb{F}^{m \times n}$,

$$A \in \mathbb{F}^{m \times n} \Longleftrightarrow A = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{pmatrix}, \quad a_{ij} \in \mathbb{F}. \tag{2.4}$$

The matrix $A^* \in \mathbb{F}^{n\times m}$,

$$A^* = \begin{pmatrix} \bar{a}_{11} & \bar{a}_{21} & \dots & \bar{a}_{m1} \\ \bar{a}_{12} & \bar{a}_{22} & \dots & \bar{a}_{m2} \\ \vdots & \vdots & & \vdots \\ \bar{a}_{1n} & \bar{a}_{2n} & \dots & \bar{a}_{nm} \end{pmatrix} \tag{2.5}$$

is the **Hermitian transpose** of A. Notice, that with this notation n-vectors can be identified with n-by-1 matrices.

The following classes of square matrices are of particular importance:

- $A \in \mathbb{F}^{n\times n}$ is called **Hermitian** if and only if $A^* = A$.
- A *real* Hermitian matrix is called **symmetric**.
- $U \in \mathbb{F}^{n\times n}$ is called **unitary** if and only if $U^{-1} = U^*$.
- *Real* unitary matrices are called **orthogonal**.
- $A \in \mathbb{F}^{n\times n}$ is called **normal** if $A^*A = AA^*$. Both, Hermitian and unitary matrices are normal.

We define the norm of a matrix to be the norm induced by the vector norm (2.3),

$$\|A\| := \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|. \tag{2.6}$$

The condition number of a nonsingular matrix is defined as $\kappa(A) = \|A\|\|A^{-1}\|$. It is easy to show that if U is unitary then $\|U\mathbf{x}\| = \|\mathbf{x}\|$ for all $\mathbf{x}$. Thus the condition number of a unitary matrix is 1.

2.2 Statement of the problem

The **(standard) eigenvalue problem** is as follows.

Given a square matrix $A \in \mathbb{F}^{n\times n}$.
Find scalars $\lambda \in \mathbb{C}$ and vectors $\mathbf{x} \in \mathbb{C}^n$, $\mathbf{x} \neq \mathbf{0}$, such that

$$A\mathbf{x} = \lambda\mathbf{x}, \tag{2.7}$$

i.e., such that

$$(A - \lambda I)\mathbf{x} = \mathbf{0} \tag{2.8}$$

has a nontrivial (nonzero) solution.

So, we are looking for numbers λ such that $A - \lambda I$ is *singular*.

Definition 2.1 Let the pair $(\lambda, \mathbf{x})$ be a solution of (2.7) or (2.8), respectively. Then

- λ is called an **eigenvalue** of A,
- $\mathbf{x}$ is called an **eigenvector** corresponding to λ

- $(\lambda, \mathbf{x})$ is called **eigenpair** of A.
- The set $\sigma(A)$ of *all* eigenvalues of A is called **spectrum** of A.
- The set of all eigenvectors corresponding to an eigenvalue λ together with the vector $\mathbf{0}$ form a linear subspace of $\mathbb{C}^n$ called the **eigenspace** of λ. As the eigenspace of λ is the null space of $\lambda I - A$ we denote it by $\mathcal{N}(\lambda I - A)$.
- The dimension of $\mathcal{N}(\lambda I - A)$ is called **geometric multiplicity** $g(\lambda)$ of λ.
- An eigenvalue λ is a zero of the **characteristic polynomial**

  $$\chi(\lambda) := \det(\lambda I - A) = \lambda^n + a_{n-1}\lambda^{n-1} + \cdots + a_0.$$

  The multiplicity of λ as a zero of χ is called the **algebraic multiplicity** $m(\lambda)$ of λ. We will later see that

  $$1 \le g(\lambda) \le m(\lambda) \le n, \qquad \lambda \in \sigma(A), \quad A \in \mathbb{F}^{n \times n}.$$

Remark 2.1. A nontrivial solution solution $\mathbf{y}$ of

$$\mathbf{y}^* A = \lambda \mathbf{y}^* \tag{2.9}$$

is called **left eigenvector** corresponding to λ. A left eigenvector of A is a right eigenvector of A^*, corresponding to the eigenvalue $\bar{\lambda}$, $A^*\mathbf{y} = \bar{\lambda}\mathbf{y}$. □

Problem 2.2 Let $\mathbf{x}$ be a (right) eigenvector of A corresponding to an eigenvalue λ and let $\mathbf{y}$ be a left eigenvector of A corresponding to a *different* eigenvalue $\mu \neq \lambda$. Show that $\mathbf{x}^*\mathbf{y} = 0$.

Remark 2.2. Let A be an **upper triangular** matrix,

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ & a_{22} & \dots & a_{2n} \\ & & \ddots & \vdots \\ & & & a_{nn} \end{pmatrix}, \qquad a_{ik} = 0 \text{ for } i > k. \tag{2.10}$$

Then we have

$$\det(\lambda I - A) = \prod_{i=1}^{n} (\lambda - a_{ii}).$$

□

Problem 2.3 Let $\lambda = a_{ii}$, $1 \le i \le n$, be an eigenvalue of A in (2.10). Can you give a corresponding eigenvector? Can you detect a situation where $g(\lambda) < m(\lambda)$?

The **(generalized) eigenvalue problem** is as follows.

Given two square matrices $A, B \in \mathbb{F}^{n \times n}$.
Find scalars $\lambda \in \mathbb{C}$ and vectors $\mathbf{x} \in \mathbb{C}$, $\mathbf{x} \neq \mathbf{0}$, such that

$$A\mathbf{x} = \lambda B\mathbf{x}, \tag{2.11}$$

or, equivalently, such that

$$(A - \lambda B)\mathbf{x} = \mathbf{0} \tag{2.12}$$

has a nontrivial solution.

Definition 2.4 Let the pair $(\lambda, \mathbf{x})$ be a solution of (2.11) or (2.12), respectively. Then

- λ is called an **eigenvalue** of A **relative to** B,
- $\mathbf{x}$ is called an **eigenvector** of A **relative to** B corresponding to λ.
- $(\lambda, \mathbf{x})$ is called an **eigenpair** of A **relative to** B,
- The set $\sigma(A; B)$ of *all* eigenvalues of (2.11) is called the **spectrum** of A **relative to** B.

Let us look at some examples.

- Let B be nonsingular. Then

  $$A\mathbf{x} = \lambda B\mathbf{x} \Longleftrightarrow B^{-1}A\mathbf{x} = \lambda\mathbf{x} \tag{2.13}$$

- Let both A and B be Hermitian, $A = A^*$ and $B = B^*$. Let further be B positive definite and $B = LL^*$ be its Cholesky factorization. Then

  $$A\mathbf{x} = \lambda B\mathbf{x} \Longleftrightarrow L^{-1}AL^{-*}\mathbf{y} = \lambda\mathbf{y}, \quad \mathbf{y} = L^*\mathbf{x}. \tag{2.14}$$

- Let A be invertible. Then $A\mathbf{x} = \mathbf{0}$ implies $\mathbf{x} = \mathbf{0}$. That is, $0 \notin \sigma(A; B)$. Therefore,

  $$A\mathbf{x} = \lambda B\mathbf{x} \Longleftrightarrow \mu\mathbf{x} = A^{-1}B\mathbf{x}, \quad \mu = \frac{1}{\lambda} \tag{2.15}$$

- Let $A = B \in \mathbb{R}^{n\times n}$ be invertible. Then

  $$A\mathbf{x} = \lambda B\mathbf{x} \Longleftrightarrow B^{-1}A\mathbf{x} = I\mathbf{x} = \mathbf{x}.$$

  Therefore, $\sigma(A; B) = \{1\}$. The associated eigenspace is $\mathbb{R}^n$. Every nonzero vector $\mathbf{x}$ is an eigenvector.

- *Difficult situation*: both A and B are singular.

  1. Let, e.g.,

     $$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

     Then,

     $$A\mathbf{e}_2 = \mathbf{0} = 0 \cdot B\mathbf{e}_2 = 0 \cdot \mathbf{e}_2,$$

     such that 0 is an eigenvalue of A relative to B. Since

     $$A\mathbf{e}_1 = \mathbf{e}_1 = \lambda B\mathbf{e}_1 = \lambda\mathbf{0}$$

     $\mathbf{e}_1$ cannot be an eigenvector of A relative to B.
     As in (2.15) we may swap the roles of A and B. Then

     $$B\mathbf{e}_1 = \mathbf{0} = \mu A\mathbf{e}_1 = 0\mathbf{e}_1.$$

     So, $\mu = 0$ is an eigenvalue of B relative to A. We therefore say, informally, that $\lambda = \infty$ is an eigenvalue of A relative to B. So, $\sigma(A; B) = \{0, \infty\}$.
  2. Let

     $$A = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} = A.$$

     Then,

     $$\begin{aligned} A\mathbf{e}_1 &= 1 \cdot B\mathbf{e}_1, \\ A\mathbf{e}_2 &= \mathbf{0} = \lambda B\mathbf{e}_2 = \lambda\mathbf{0}, \text{ for } \textbf{all } \lambda \in \mathbb{C}. \end{aligned}$$

     Therefore, in this case, $\sigma(A; B) = \mathbb{C}$.

2.3 Similarity transformations

Definition 2.5 A matrix $A \in \mathbb{F}^{n\times n}$ is **similar** to a matrix $C \in \mathbb{F}^{n\times n}$, $A \sim C$, if and only if there is a nonsingular matrix S such that

$$S^{-1}AS = C. \tag{2.16}$$

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

Theorem 2.6 *Similar matrices have equal eigenvalues with equal multiplicities. If $(\lambda, \mathbf{x})$ is an eigenpair of A and $C = S^{-1}AS$ then $(\lambda, S^{-1}\mathbf{x})$ is an eigenpair of C.*

Proof. $A\mathbf{x} = \lambda\mathbf{x}$ and $C = S^{-1}AS$ imply that

$$CS^{-1}\mathbf{x} = S^{-1}ASS^{-1}\mathbf{x} = S^{-1}\lambda\mathbf{x}.$$

Hence, A and C have equal eigenvalues and their geometric multiplicity is not changed by the similarity transformation. From

$$\begin{aligned}\det(\lambda I - C) &= \det(\lambda S^{-1}S - S^{-1}AS)\\ &= \det(S^{-1}(\lambda I - A)S) = \det(S^{-1})\det(\lambda I - A)\det(S) = \det(\lambda I - A)\end{aligned}$$

it follows that the characteristic polynomials of A and C are equal and hence also the algebraic eigenvalue multiplicities are equal. ■

Similarity transformations are used to transform matrices into similar matrices from which eigenvalues can be easily read. Diagonal matrices are the preferred matrix structure. However, not all matrices are diagonalizable. There is, e.g., no invertible matrix S that diagonalizes the matrix

$$\begin{bmatrix}1 & 1\\ 0 & 1\end{bmatrix}.$$

In the Jordan normal form introduced in section 2.8 the transformation is into a bidiagonal matrix. In the Schur normal form, see section 2.4 the transformation is into an upper tridiagonal matrix, but with an unitary S.

Definition 2.7 Two matrices A and B are called **unitarily similar** if S in (2.16) is unitary. If the matrices are real the term orthogonally similar is used.

Unitary similarity transformations are very important in numerical computations. Let U be unitary. Then $\|U\| = \|U^{-1}\| = 1$, the condition number of U is therefore $\kappa(U) = 1$. Hence, if $C = U^{-1}AU$ then $C = U^*AU$ and $\|C\| = \|A\|$. In particular, if A is disturbed by δA (e.g., roundoff errors introduced when storing the entries of A in finite-precision arithmetic) then

$$U^*(A + \delta A)U = C + \delta C, \qquad \|\delta C\| = \|\delta A\|.$$

Hence, errors (perturbations) in A are not amplified by a unitary similarity transformation. This is in contrast to arbitrary similarity transformations. However, as we will see later, small eigenvalues may still suffer from large relative errors.

Another reason for the importance of unitary similarity transformations is the preservation of symmetry: If A is symmetric then $U^{-1}AU = U^*AU$ is symmetric as well.

For generalized eigenvalue problems, similarity transformations are not so crucial since we can operate with different matrices from both sides. If S and T are nonsingular

$$A\mathbf{x} = \lambda B\mathbf{x} \quad \Longleftrightarrow \quad TAS^{-1}S\mathbf{x} = \lambda TBS^{-1}S\mathbf{x}.$$

This sometimes called *equivalence transformation* of A, B. Thus, $\sigma(A; B) = \sigma(TAS^{-1}; TBS^{-1})$. Let us consider a special case: let B be invertible and let $B = LU$ be the LU-factorization of B. Then we set $S = U$ and $T = L^{-1}$ and obtain $TBU^{-1} = L^{-1}LUU^{-1} = I$. Thus, $\sigma(A; B) = \sigma(L^{-1}AU^{-1}; I) = \sigma(L^{-1}AU^{-1})$.

2.4 Schur decomposition

Theorem 2.8 ***(Schur decomposition)*** *If $A \in \mathbb{C}^{n\times n}$ then there is a unitary matrix $U \in \mathbb{C}^{n\times n}$ such that*

$$U^*AU = T \tag{2.17}$$

is upper triangular. The diagonal elements of T are the eigenvalues of A.

Proof. The proof is by induction. For $n = 1$, the theorem is obviously true.

Assume that the theorem holds for matrices of order $\leq n-1$. Let $(\lambda, \mathbf{x})$, $\|\mathbf{x}\| = 1$, be an eigenpair of A, $A\mathbf{x} = \lambda\mathbf{x}$. We construct a unitary matrix U_1 with first column $\mathbf{x}$ (e.g. the Householder reflector U_1 with $U_1\mathbf{x} = \mathbf{e}_1$). Partition $U_1 = [\mathbf{x}, \overline{U}]$. Then

$$U_1^*AU_1 = \begin{bmatrix} \mathbf{x}^*A\mathbf{x} & \mathbf{x}^*A\overline{U} \\ \overline{U}^*A\mathbf{x} & \overline{U}^*A\overline{U} \end{bmatrix} = \begin{bmatrix} \lambda & \times\cdots\times \\ \mathbf{0} & \hat{A} \end{bmatrix}$$

as $A\mathbf{x} = \lambda\mathbf{x}$ and $\overline{U}^*\mathbf{x} = \mathbf{0}$ by construction of U_1. By assumption, there exists a unitary matrix $\hat{U} \in \mathbb{C}^{(n-1)\times(n-1)}$ such that $\hat{U}^*\hat{A}\hat{U} = \hat{T}$ is upper triangular. Setting $U := U_1(1 \oplus \hat{U})$, we obtain (2.17). ■

Notice , that this proof is not constructive as we assume the knowledge of an eigenpair $(\lambda, \mathbf{x})$. So, we cannot employ it to actually compute the Schur form. The QR algorithm is used for this purpose. We will discuss this basic algorithm in Chapter 4.

Let $U^*AU = T$ be a Schur decomposition of A with $U = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_n]$. The Schur decomposition can be written as $AU = UT$. The k-th column of this equation is

$$A\mathbf{u}_k = \lambda\mathbf{u}_k + \sum_{i=1}^{k-1} t_{ik}\mathbf{u}_i, \qquad \lambda_k = t_{kk}. \tag{2.18}$$

This implies that

$$A\mathbf{u}_k \in \operatorname{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}, \quad \forall k. \tag{2.19}$$

Thus, the first k **Schur vectors** $\mathbf{u}_1, \ldots, \mathbf{u}_k$ form an **invariant subspace**[1] for A. From (2.18) it is clear that the *first* Schur vector is an eigenvector of A. The other columns of U, however, are in general *not* eigenvectors of A. Notice, that the Schur decomposition is not unique. In the proof we have chosen *any* eigenvalue λ. This indicates that the eigenvalues can be arranged in any order in the diagonal of T. This also indicates that the order with which the eigenvalues appear on T's diagonal can be manipulated.

Problem 2.9 Let

$$A = \begin{bmatrix} \lambda_1 & \alpha \\ 0 & \lambda_2 \end{bmatrix}.$$

[1] A subspace $\mathcal{V} \subset \mathbb{F}^n$ is called invariant for A if $A\mathcal{V} \subset \mathcal{V}$.

Find an orthogonal 2×2 matrix Q such that

$$Q^* A Q = \begin{bmatrix} \lambda_2 & \beta \\ 0 & \lambda_1 \end{bmatrix}.$$

Hint: the first column of Q must be a normalized eigenvector of A corresponding to eigenvalue λ_2. Why?

2.5 The real Schur decomposition

Real matrices can have complex eigenvalues. If complex eigenvalues exist, then they occur in *complex conjugate pairs*! That is, if λ is an eigenvalue of the real matrix A, then also $\bar{\lambda}$ is an eigenvalue of A. The following theorem indicates that complex computation can be avoided.

Theorem 2.10 ***(Real Schur decomposition)*** *If $A \in \mathbb{R}^{n\times n}$ then there is an orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ such that*

$$Q^T A Q = \begin{bmatrix} R_{11} & R_{12} & \cdots & R_{1m} \\ & R_{22} & \cdots & R_{2m} \\ & & \ddots & \vdots \\ & & & R_{mm} \end{bmatrix} \tag{2.20}$$

is upper quasi-triangular. The diagonal blocks R_{ii} are either 1×1 or 2×2 matrices. A 1×1 block corresponds to a real eigenvalue, a 2×2 block corresponds to a pair of complex conjugate eigenvalues.

Remark 2.3. The matrix

$$\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}, \quad \alpha, \beta \in \mathbb{R},$$

has the eigenvalues $\alpha + i\beta$ and $\alpha - i\beta$. □

Proof. Let $\lambda = \alpha + i\beta$, $\beta \neq 0$, be an eigenvalue of A with eigenvector $\mathbf{x} = \mathbf{u} + i\mathbf{v}$. Then $\bar{\lambda} = \alpha - i\beta$ is an eigenvalue corresponding to $\bar{\mathbf{x}} = \mathbf{u} - i\mathbf{v}$. To see this we first observe that

$$\begin{aligned} A\mathbf{x} &= A(\mathbf{u} + i\mathbf{v}) = A\mathbf{u} + iA\mathbf{v}, \\ \lambda\mathbf{x} &= (\alpha + i\beta)(\mathbf{u} + i\mathbf{v}) = (\alpha\mathbf{u} - \beta\mathbf{v}) + i(\beta\mathbf{u} + \alpha\mathbf{v}). \end{aligned}$$

Thus,

$$\begin{aligned} A\bar{\mathbf{x}} &= A(\mathbf{u} - i\mathbf{v}) = A\mathbf{u} - iA\mathbf{v}, \\ &= (\alpha\mathbf{u} - \beta\mathbf{v}) - i(\beta\mathbf{u} + \alpha\mathbf{v}) \\ &= (\alpha - i\beta)\mathbf{u} - i(\alpha - i\beta)\mathbf{v} = (\alpha - i\beta)(\mathbf{u} - i\mathbf{v}) = \bar{\lambda}\bar{\mathbf{x}}. \end{aligned}$$

Now, the actual proof starts. Let k be the number of complex conjugate pairs. We prove the theorem by induction on k.

First we consider the case $k = 0$. In this case A has real eigenvalues and eigenvectors. It is clear that we can repeat the proof of the Schur decomposition of Theorem 2.8 in real arithmetic to get the decomposition (2.17) with $U \in \mathbb{R}^{n\times n}$ and $T \in \mathbb{R}^{n\times n}$. So, there are n diagonal blocks R_{jj} in (2.20) all of which are 1×1.

Let us now assume that the theorem is true for all matrices with fewer than k complex conjugate pairs. Then, with $\lambda = \alpha + i\beta$, $\beta \neq 0$ and $\mathbf{x} = \mathbf{u} + i\mathbf{v}$, as previously, we have

$$A[\mathbf{u}, \mathbf{v}] = [\mathbf{u}, \mathbf{v}] \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}.$$

Let $\{\mathbf{x}_1, \mathbf{x}_2\}$ be an orthonormal basis of span($[\mathbf{u}, \mathbf{v}]$). Then, since $\mathbf{u}$ and $\mathbf{v}$ are linearly independent[2], there is a nonsingular 2×2 real square matrix C with

$$[\mathbf{x}_1, \mathbf{x}_2] = [\mathbf{u}, \mathbf{v}]C.$$

Now,

$$\begin{aligned} A[\mathbf{x}_1, \mathbf{x}_2] = A[\mathbf{u}, \mathbf{v}]C = A[\mathbf{u}, \mathbf{v}] \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} C \\ = [\mathbf{x}_1, \mathbf{x}_2]C^{-1} \begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix} C =: [\mathbf{x}_1, \mathbf{x}_2]S. \end{aligned}$$

S and $\begin{bmatrix} \alpha & \beta \\ -\beta & \alpha \end{bmatrix}$ are similar and therefore have equal eigenvalues. Now we construct an orthogonal matrix $[\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n] =: [\mathbf{x}_1, \mathbf{x}_2, W]$. Then

$$\big[[\mathbf{x}_1, \mathbf{x}_2], W\big]^T A \big[[\mathbf{x}_1, \mathbf{x}_2], W\big] = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ W^T \end{bmatrix} \big[[\mathbf{x}_1, \mathbf{x}_2]S, AW\big] = \begin{bmatrix} S & [\mathbf{x}_1, \mathbf{x}_2]^T AW \\ O & W^T AW \end{bmatrix}.$$

The matrix $W^T AW$ has less than k complex-conjugate eigenvalue pairs. Therefore, by the induction assumption, there is an orthogonal $Q_2 \in \mathbb{R}^{(n-2)\times(n-2)}$ such that the matrix

$$Q_2^T (W^T AW) Q_2$$

is quasi-triangular. Thus, the orthogonal matrix

$$Q = [\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_n] \begin{pmatrix} I_2 & O \\ O & Q_2 \end{pmatrix}$$

transforms A similarly to quasi-triangular form. ■

2.6 Normal matrices

Definition 2.11 A matrix $A \in \mathbb{F}^{n\times n}$ is called **normal** if

$$AA^* = A^*A. \tag{2.21}$$

Let $A = URU^*$ be the Schur decomposition of A. Then,

$$RR^* = U^*AUU^*A^*U = U^*AA^*U = U^*A^*AU = U^*A^*UU^*AU = R^*R.$$

Therefore, also the upper triangular R is normal. We look at the (1,1)-elements of RR^* and R^*R that evidently must be equal. On one hand we have

$$(R^*R)_{11} = \bar{r}_{11}r_{11} = |r_{11}|^2,$$

on the other hand

$$(RR^*)_{11} = \sum_{j=1}^{n} r_{1j}\bar{r}_{1j} = |r_{11}|^2 + \sum_{j=2}^{n} |r_{1j}|^2.$$

Therefore, the latter sum must vanish, i.e., $r_{1j} = 0$ for $j = 2, \ldots, n$. Comparing the (2,2)-elements, (3,3)-elements, etc., of RR^* and R^*R, we see that R is diagonal. In this way we arrive at

Theorem 2.12 *A matrix is normal if and only if it is diagonalizable by a unitary matrix.* ■

(Note that unitarily diagonalizable matrices are trivially normal.)

[2] If u and v were linearly dependent then it follows that β must be zero.

2.7 Hermitian matrices

Definition 2.13 A matrix $A \in \mathbb{F}^{n\times n}$ is **Hermitian** if

$$A = A^*. \tag{2.22}$$

The Schur decomposition for Hermitian matrices is particularly simple. We first note that A being Hermitian implies that the upper triangular Λ in the Schur decomposition $A = U\Lambda U^*$ is Hermitian and thus diagonal. In fact, because

$$\overline{\Lambda} = \Lambda^* = (U^*AU)^* = U^*A^*U = U^*AU = \Lambda,$$

each diagonal element λ_i of Λ satisfies $\overline{\lambda_i} = \lambda_i$. So, Λ has to be *real*. In summary have the following result.

Theorem 2.14 ***(Spectral theorem for Hermitian matrices)*** *Let A be Hermitian. Then there is a unitary matrix U and a real diagonal matrix Λ such that*

$$A = U\Lambda U^* = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^*. \tag{2.23}$$

The columns $\mathbf{u}_1, \ldots, \mathbf{u}_n$ of U are eigenvectors corresponding to the eigenvalues $\lambda_1, \ldots, \lambda_n$. They form an orthonormal basis for $\mathbb{F}^n$.

The decomposition (2.23) is called a *spectral decomposition* of A.

As the eigenvalues are real we can sort them with respect to their magnitude. We can, e.g., arrange them in ascending order such that $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$.

If $\lambda_i = \lambda_j$, then any nonzero linear combination of $\mathbf{u}_i$ and $\mathbf{u}_j$ is an eigenvector corresponding to λ_i,

$$A(\mathbf{u}_i\alpha + \mathbf{u}_j\beta) = \mathbf{u}_i\lambda_i\alpha + \mathbf{u}_j\lambda_j\beta = (\mathbf{u}_i\alpha + \mathbf{u}_j\beta)\lambda_i.$$

However, eigenvectors corresponding to *different* eigenvalues are orthogonal. Let $A\mathbf{u} = \mathbf{u}\lambda$ and $A\mathbf{v} = \mathbf{v}\mu$, $\lambda \neq \mu$. Then

$$\lambda\mathbf{u}^*\mathbf{v} = (\mathbf{u}^*A)\mathbf{v} = \mathbf{u}^*(A\mathbf{v}) = \mathbf{u}^*\mathbf{v}\mu,$$

and thus

$$(\lambda - \mu)\mathbf{u}^*\mathbf{v} = 0,$$

from which we deduce $\mathbf{u}^*\mathbf{v} = 0$ as $\lambda \neq \mu$.

In summary, the eigenvectors corresponding to a particular eigenvalue λ form a subspace, the *eigenspace* $\{\mathbf{x} \in \mathbb{F}^n, A\mathbf{x} = \lambda\mathbf{x}\} = \mathcal{N}(A - \lambda I)$. They are perpendicular to the eigenvectors corresponding to all the other eigenvalues. Therefore, the spectral decomposition (2.23) is unique up to ± signs if all the eigenvalues of A are distinct. In case of multiple eigenvalues, we are free to choose any orthonormal basis for the corresponding eigenspace.

Remark 2.4. The notion of Hermitian or symmetric has a wider background. Let $\langle\mathbf{x}, \mathbf{y}\rangle$ be an inner product on $\mathbb{F}^n$. Then a matrix A is symmetric with respect to this inner product if $\langle A\mathbf{x}, \mathbf{y}\rangle = \langle\mathbf{x}, A\mathbf{y}\rangle$ for all vectors $\mathbf{x}$ and $\mathbf{y}$. For the ordinary Euclidean inner product $(\mathbf{x}, \mathbf{y}) = \mathbf{x}^*\mathbf{y}$ we arrive at the element-wise Definition 2.7 if we set $\mathbf{x}$ and $\mathbf{y}$ equal to coordinate vectors.

It is important to note that all the properties of Hermitian matrices that we will derive subsequently hold similarly for matrices symmetric with respect to a certain inner product.
□

Example 2.15 We consider the one-dimensional Sturm-Liouville eigenvalue problem

$$-u''(x) = \lambda u(x), \quad 0 < x < \pi, \quad u(0) = u(\pi) = 0, \tag{2.24}$$

that models the vibration of a homogeneous string of length π that is *fixed* at both ends. The eigenvalues and eigenvectors or eigenfunctions of (2.24) are

$$\lambda_k = k^2, \quad u_k(x) = \sin kx, \qquad k \in \mathbb{N}.$$

Let $u_i^{(n)}$ denote the approximation of an (eigen)function u at the grid point x_i,

$$u_i \approx u(x_i), \quad x_i = ih, \quad 0 \le i \le n+1, \quad h = \frac{\pi}{n+1}.$$

We approximate the second derivative of u at the *interior* grid points by finite differences [3, 7]

$$\frac{1}{h^2}(-u_{i-1} + 2u_i - u_{i+1}) = \lambda u_i, \qquad 1 \le i \le n. \tag{2.25}$$

Collecting these equations and taking into account the boundary conditions, $u_0 = 0$ and $u_{n+1} = 0$, we get a (matrix) eigenvalue problem

$$T_n \mathbf{x} = \lambda \mathbf{x} \tag{2.26}$$

where

$$T_n := \frac{(n+1)^2}{\pi^2} \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{n \times n}.$$

The matrix eigenvalue problem (2.26) can be solved explicitly [9, p.229]. Eigenvalues and eigenvectors are given by

$$\begin{aligned} \lambda_k^{(n)} &= \frac{(n+1)^2}{\pi^2}(2 - 2\cos\phi_k) = \frac{4(n+1)^2}{\pi^2}\sin^2\frac{k\pi}{2(n+1)}, \\ \mathbf{u}_k^{(n)} &= \left(\frac{2}{n+1}\right)^{1/2} [\sin\phi_k, \sin 2\phi_k, \ldots, \sin n\phi_k]^T, \qquad \phi_k = \frac{k\pi}{n+1}. \end{aligned} \tag{2.27}$$

Clearly, $\lambda_k^{(n)}$ converges to λ_k as $n \to \infty$. (Note that $\sin\xi \to \xi$ as $\xi \to 0$.) When we identify $\mathbf{u}_k^{(n)}$ with the piecewise linear function that takes on the values given by $\mathbf{u}_k^{(n)}$ at the grid points x_i then this function evidently converges to $\sin kx$.

Let $p(\lambda)$ be a polynomial of degree d, $p(\lambda) = \alpha_0 + \alpha_1\lambda + \alpha_2\lambda^2 + \cdots + \alpha_d\lambda^d$. As $A^j = (U\Lambda U^*)^j = U\Lambda^j U^*$ we can define a *matrix polynomial* as

$$p(A) = \sum_{j=0}^{d} \alpha_j A^j = \sum_{j=0}^{d} \alpha_j U\Lambda^j U^* = U\left(\sum_{j=0}^{d} \alpha_j \Lambda^j\right) U^*. \tag{2.28}$$

This equation shows that $p(A)$ has the same eigenvectors as the original matrix A. The eigenvalues are modified though, λ_k becomes $p(\lambda_k)$. Similarly, more complicated functions of A can be computed if the function is defined on spectrum of A.

2.8 The Jordan normal form

Theorem 2.16 ***(Jordan normal form)*** *For every $A \in \mathbb{F}^{n\times n}$ there is a nonsingular matrix $X \in \mathbb{F}^{n\times n}$ such that*

$$X^{-1}AX = J = \operatorname{diag}(J_1, J_2, \ldots, J_p), \tag{2.29}$$

where

$$J_k = J_{m_k}(\lambda_k) = \begin{bmatrix} \lambda_k & 1 & & \\ & \lambda_k & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_k \end{bmatrix} \in \mathbb{F}^{m_k\times m_k} \tag{2.30}$$

*are called **Jordan blocks** and $m_1 + \cdots + m_p = n$. The values λ_k need not be distinct. The Jordan matrix J is unique up to the ordering of the blocks. The transformation matrix X is not unique.*

A matrix is diagonalizable if all Jordan blocks are 1 × 1, i.e., $m_k = 1$ for all k[3]. In this case the columns of X are eigenvectors of A.

More generally, there is one eigenvector associated with each Jordan block, e.g.,

$$J_2(\lambda)\mathbf{e}_1 = \begin{bmatrix} \lambda & 1 \\ 0 & \lambda \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \lambda\, \mathbf{e}_1.$$

Nontrivial Jordan blocks give rise to so-called generalized eigenvectors $\mathbf{e}_2, \ldots, \mathbf{e}_{m_k}$ since

$$(J_k(\lambda) - \lambda I)\mathbf{e}_{j+1} = \mathbf{e}_j, \quad j = 1, \ldots, m_k - 1.$$

This choice of generalized eigenvectors is not unique though, as $(J_k(\lambda) - \lambda I)(\mathbf{e}_2 + \alpha\mathbf{e}_1) = \mathbf{e}_1$ for any α. This is one of the reasons for the non-uniqueness of the transformation matrix X in Theorem 2.16.

From the Jordan blocks we can read geometric and algebraic multiplicity of an eigenvalue: The number of Jordan blocks associated with a particular eigenvalue give the geometric multiplicity; the sum of its orders gives the algebraic multiplicity.

Numerically the size of the Jordan blocks cannot be determined stably as the following example shows. Let

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & -\varepsilon \end{bmatrix} \approx \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = J_2(0)$$

be the approximation for $J_2(0)$ that some numerical algorithm has computed. This matrix has two distinct eigenvalues and thus two eigenvectors,

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

For small ε the two eigenvectors are very close. They even collaps when $\varepsilon \to 0$. A numerical code cannot differ between the two cases ($\varepsilon = 0$, $\varepsilon \neq 0$) that have a completely different structure.

[3] 1 × 1 Jordan blocks are called trivial.

Let $Y := X^{-*}$ and let $X = [X_1, X_2, \ldots, X_p]$ and $Y = [Y_1, Y_2, \ldots, Y_p]$ be partitioned according to J in (2.29), meaning that $X_j, Y_j \in \mathbb{F}^{n \times m_j}$. Then,

$$A = XJY^* = \sum_{k=1}^{p} X_k J_k Y_k^* = \sum_{k=1}^{p} (\lambda_k X_k Y_k^* + X_k N_k Y_k^*), \tag{2.31}$$

where $N_k = J_{m_k}(0)$. If $m_k = 1$ then N_k is zero. We define the matrices $P_k := X_k Y_k^*$ and $D_k := X_k N_k Y_k^*$. Then, since $P_k^2 = P_k$, P_k is a *projector* on $\mathcal{R}(P_k) = \mathcal{R}(X_k)$. It is called a **spectral projector**. From (2.31) we immediately obtain [8]

$$A = \sum_{k=1}^{p} (\lambda_k P_k + D_k). \tag{2.32}$$

Since $I_{m_k} N_k = N_k I_{m_k} = N_k$, we have

$$\begin{aligned} P_k D_\ell &= D_\ell P_k = \delta_{k\ell} D_\ell, \\ A P_k &= P_k A = P_k A P_k = \lambda_k P_k + D_k, \\ A^j P_k &= P_k A^j = P_k A^j P_k = P_k(\lambda_k I_n + D_k)^j = (\lambda_k I_n + D_k)^j P_k. \end{aligned}$$

The Jordan normal form can be computed from the Schur decomposition $A = U^* T U$, see, e.g., [2], although it is not recommended in general to do so.

1. Group equal eigenvalues on the diagonal of the triangular T. This is a generalization of the solution of Problem 2.4.

2. Let

$$T = \begin{bmatrix} T_1 & T_{12} & \cdots & T_{1s} \\ & T_2 & \cdots & T_{2s} \\ & & \ddots & \vdots \\ & & & T_s \end{bmatrix} \tag{2.33}$$

   where the s diagonal blocks T_k are related to the s distinct eigenvalues of T. The off-diagonal blocks $T_{j\ell}$ are zeroed one after the other. Each steps requires the solution of a **Sylvester equation** $T_{j\ell} = T_j Y - Y T_\ell$.

   ***Exercise*:** Consider the case of two (simple or multiple) eigenvalues,

$$T = \begin{bmatrix} T_1 & T_{12} \\ & T_2 \end{bmatrix}.$$

   Apply a similarity transformation with the matrix

$$X = \begin{bmatrix} I_1 & Y \\ & I_2 \end{bmatrix}.$$

   Determine Y? How can this be extended to the case (2.33) with s diagonal blocks?

3. The diagonal blocks $T_1, \ldots, T_s$ are brought to Jordan form.

The Jordan normal form can be nicely employed to define matrix functions, see [5].

2.9 Projections

Definition 2.17 A matrix P that satisfies

$$P^2 = P \tag{2.34}$$

is called a **projection**.

Obviously, a projection is a square matrix. If P is a projection then $P\mathbf{x} = \mathbf{x}$ for all $\mathbf{x}$ in the range $\mathcal{R}(P)$ of P. In fact, if $\mathbf{x} \in \mathcal{R}(P)$ then $\mathbf{x} = P\mathbf{y}$ for some $\mathbf{y} \in \mathbb{F}^n$ and $P\mathbf{x} = P(P\mathbf{y}) = P^2\mathbf{y} = P\mathbf{y} = \mathbf{x}$.

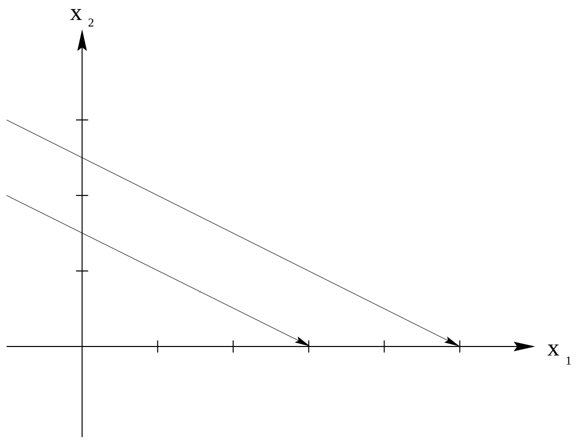


Figure 2.1: Oblique projection of example 2.9

Example 2.18 Let

$$P = \begin{pmatrix} 1 & 2 \\ 0 & 0 \end{pmatrix}.$$

The range of P is $\mathcal{R}(P) = \mathbb{F} \times \{\mathbf{0}\}$. The effect of P is depicted in Figure 2.1: All points $\mathbf{x}$ that lie on a line parallel to $\text{span}\{(2, -1)^*\}$ are mapped on the same point on the $\mathbf{x}_1$ axis. So, the projection is *along* $\text{span}\{(2, -1)^*\}$ which is the null space $\mathcal{N}(P)$ of P.

Example 2.19 Let $\mathbf{x}$ and $\mathbf{y}$ be arbitrary vectors such that $\mathbf{y}^*\mathbf{x} \neq 0$. Then

$$P = \frac{\mathbf{x}\mathbf{y}^*}{\mathbf{y}^*\mathbf{x}} \tag{2.35}$$

is a projection. Notice that the projector of the previous example can be expressed in the form (2.35).

Problem 2.20 Let $X, Y \in \mathbb{F}^{n \times p}$ such that Y^*X is nonsingular. Show that

$$P := X(Y^*X)^{-1}Y^*$$

is a projection.

Example 2.21 The spectral projectors $X_kY_k^*$ introduced in (2.31) are projectors. Their range is the span of all eigenvectors and generalized eigenvectors associated with the eigenvalue λ_k.

If P is a projection then $I-P$ is a projection as well. In fact, $(I-P)^2 = I-2P+P^2 = I-2P+P = I-P$. If $P\mathbf{x} = \mathbf{0}$ then $(I-P)\mathbf{x} = \mathbf{x}$. Therefore, the range of $I-P$ coincides with the null space of P, $\mathcal{R}(I-P) = \mathcal{N}(P)$. It can be shown that $\mathcal{R}(P) = \mathcal{N}(P^*)^\perp$.

Notice that $\mathcal{R}(P)\cap\mathcal{R}(I-P) = \mathcal{N}(I-P)\cap\mathcal{N}(P) = \{\mathbf{0}\}$. For, if $P\mathbf{x} = \mathbf{0}$ then $(I-P)\mathbf{x} = \mathbf{x}$, which can only be zero if $\mathbf{x} = \mathbf{0}$. So, any vector $\mathbf{x}$ can be uniquely decomposed into

$$\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2, \qquad \mathbf{x}_1 \in \mathcal{R}(P), \quad \mathbf{x}_2 \in \mathcal{R}(I-P) = \mathcal{N}(P). \tag{2.36}$$

The most interesting situation occurs if the decomposition is orthogonal, i.e., if $\mathbf{x}_1^*\mathbf{x}_2 = 0$ for all $\mathbf{x}$.

Definition 2.22 A matrix P is called an **orthogonal projection** if

$$\begin{array}{ll} \text{(i)} & P^2 = P \\ \text{(ii)} & P^* = P. \end{array} \tag{2.37}$$

Proposition 2.23 *Let P be a projection. Then the following statements are equivalent.*
(i) $P^* = P$,
(ii) $\mathcal{R}(I-P) \perp \mathcal{R}(P)$, *i.e.* $(P\mathbf{x})^*(I-P)\mathbf{y} = 0$ *for all* $\mathbf{x}, \mathbf{y}$.

Proof. (ii) follows trivially from (i) and (2.34).

Now, let us assume that (ii) holds. Then

$$\begin{aligned} \mathbf{x}^*P^*\mathbf{y} = (P\mathbf{x})^*\mathbf{y} &= (P\mathbf{x})^*(P\mathbf{y} + (I-P)\mathbf{y}) \\ &= (P\mathbf{x})^*(P\mathbf{y}) \\ &= (P\mathbf{x} + (I-P)\mathbf{x})(P\mathbf{y}) = \mathbf{x}^*(P\mathbf{y}). \end{aligned}$$

This equality holds for any $\mathbf{x}$ and $\mathbf{y}$ and thus implies (i). ■

Example 2.24 Let $\mathbf{q}$ be an arbitrary vector of norm 1, $\|\mathbf{q}\| = \mathbf{q}^*\mathbf{q} = 1$. Then $P = \mathbf{q}\mathbf{q}^*$ is the orthogonal projection onto span$\{\mathbf{q}\}$.

Example 2.25 Let $Q \in \mathbb{F}^{n\times p}$ with $Q^*Q = I_p$. Then QQ^* is the orthogonal projector onto $\mathcal{R}(Q)$, which is the space spanned by the columns of Q.

Problem 2.26 Let $Q, Q_1 \in \mathbb{F}^{n\times p}$ with $Q^*Q = Q_1^*Q_1 = I_p$ such that $\mathcal{R}(Q) = \mathcal{R}(Q_1)$. This means that the columns of Q and Q_1, respectively, are orthonormal bases of the *same* subspace of $\mathbb{F}^n$. Show that the projector does not depend on the basis of the subspace, i.e., that $QQ^* = Q_1Q_1^*$.

Problem 2.27 Let $Q = [Q_1, Q_2]$, $Q_1 \in \mathbb{F}^{n\times p}$, $Q_2 \in \mathbb{F}^{n\times(n-p)}$ be a unitary matrix. Q_1 contains the first p columns of Q, Q_2 the last $n-p$. Show that $Q_1Q_1^* + Q_2Q_2^* = I$. Hint: Use $QQ^* = I$. Notice, that if $P = Q_1Q_1^*$ then $I - P = Q_2Q_2^*$.

Problem 2.28 What is the form of the orthogonal projection onto span$\{\mathbf{q}\}$ if the inner product is defined as $\langle \mathbf{x}, \mathbf{y}\rangle := \mathbf{y}^*M\mathbf{x}$ where M is a symmetric positive definite matrix?

2.10 The Rayleigh quotient

Definition 2.29 The quotient

$$\rho(\mathbf{x}) := \frac{\mathbf{x}^* A\mathbf{x}}{\mathbf{x}^*\mathbf{x}}, \qquad \mathbf{x} \neq \mathbf{0},$$

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

Notice, that $\rho(\mathbf{x}\alpha) = \rho(\mathbf{x})$, $\alpha \neq 0$. Hence, the properties of the Rayleigh quotient can be investigated by just considering its values on the unit sphere. Using the spectral decomposition $A = U\Lambda U^*$, we get

$$\mathbf{x}^* A\mathbf{x} = \mathbf{x}^* U\Lambda U^*\mathbf{x} = \sum_{i=1}^{n} \lambda_i |\mathbf{u}_i^*\mathbf{x}|^2.$$

Similarly, $\mathbf{x}^*\mathbf{x} = \sum_{i=1}^n |\mathbf{u}_i^*\mathbf{x}|^2$. With $\lambda_1 \le \lambda_2 \le \cdots \le \lambda_n$, we have

$$\lambda_1 \sum_{i=1}^{n} |\mathbf{u}_i^*\mathbf{x}|^2 \le \sum_{i=1}^{n} \lambda_i |\mathbf{u}_i^*\mathbf{x}|^2 \le \lambda_n \sum_{i=1}^{n} |\mathbf{u}_i^*\mathbf{x}|^2.$$

So,

$$\lambda_1 \le \rho(\mathbf{x}) \le \lambda_n, \qquad \text{for all } \mathbf{x} \neq \mathbf{0}.$$

As

$$\rho(\mathbf{u}_k) = \lambda_k,$$

the extremal values λ_1 and λ_n are actually attained for $\mathbf{x} = \mathbf{u}_1$ and $\mathbf{x} = \mathbf{u}_n$, respectively. Thus we have proved the following theorem.

Theorem 2.30 *Let A be Hermitian. Then the Rayleigh quotient satisfies*

$$\lambda_1 = \min_{\mathbf{x}} \rho(\mathbf{x}), \qquad \lambda_n = \max_{\mathbf{x}} \rho(\mathbf{x}). \tag{2.38}$$

As the Rayleigh quotient is a continuous function it attains *all* values in the closed interval $[\lambda_1, \lambda_n]$.

The next theorem generalizes the above theorem to interior eigenvalues. The following theorems is attributed to Poincaré, Fischer and Pólya.

Theorem 2.31 ***(Minimum-maximum principle)*** *Let A be Hermitian. Then*

$$\lambda_p = \min_{X\in\mathbb{F}^{n\times p},\ \mathrm{rank}(X)=p} \max_{\mathbf{x}\neq\mathbf{0}} \rho(X\mathbf{x}) \tag{2.39}$$

Proof. Let $U_{p-1} = [\mathbf{u}_1, \ldots, \mathbf{u}_{p-1}]$. For every $X \in \mathbb{F}^{n\times p}$ with full rank we can choose $\mathbf{x} \neq \mathbf{0}$ such that $U_{p-1}^* X\mathbf{x} = \mathbf{0}$. Then $\mathbf{0} \neq \mathbf{z} := X\mathbf{x} = \sum_{i=p}^n z_i\mathbf{u}_i$. As in the proof of the previous theorem we obtain the inequality

$$\rho(\mathbf{z}) \ge \lambda_p.$$

To prove that equality holds in (2.39) we choose $X = [\mathbf{u}_1, \ldots, \mathbf{u}_p]$. Then

$$U_{p-1}^* X\mathbf{x} = \begin{bmatrix} 1 & & & 0 \\ & \ddots & & \vdots \\ & & 1 & 0 \end{bmatrix} \mathbf{x} = \mathbf{0}$$

implies that $\mathbf{x} = \mathbf{e}_p$, i.e., that $\mathbf{z} = X\mathbf{x} = \mathbf{u}_p$. So, $\rho(\mathbf{z}) = \lambda_p$. ■

An important consequence of the minimum-maximum principle is the following

Theorem 2.32 ***(Monotonicity principle)*** *Let A be Hermitian and let $\mathbf{q}_1, \ldots, \mathbf{q}_p$ be normalized, mutually orthogonal vectors. Set $Q := [\mathbf{q}_1, \ldots, \mathbf{q}_p]$ and $A' := Q^*AQ \in \mathbb{F}^{p\times p}$. Then the p eigenvalues $\lambda'_1 \le \cdots \le \lambda'_p$ of A' satisfy*

$$\lambda_k \le \lambda'_k, \qquad 1 \le k \le p. \tag{2.40}$$

Proof. Let $\mathbf{w}_1, \ldots, \mathbf{w}_p \in \mathbb{F}^p$ be the eigenvectors of A',

$$A'\mathbf{w}_i = \lambda'_i \mathbf{w}_i, \qquad 1 \le i \le p, \tag{2.41}$$

with $\mathbf{w}_i^*\mathbf{w}_j = \delta_{ij}$. Then the vectors $Q\mathbf{w}_1, \ldots, Q\mathbf{w}_p$ are normalized and mutually orthogonal. Therefore, we can construct a normalized vector $\mathbf{x}_0$ with $\|\mathbf{x}_0\| = 1$,

$$\mathbf{x}_0 := a_1 Q\mathbf{w}_1 + \cdots + a_k Q\mathbf{w}_k = Q(a_1\mathbf{w}_1 + \cdots + a_k\mathbf{w}_k) = Q\mathbf{a},$$

that is orthogonal to the first $k-1$ eigenvectors of A,

$$\mathbf{x}_0^*\mathbf{u}_i = 0, \qquad 1 \le i \le k-1.$$

(Note, that $\|\mathbf{x}_0\| = 1$ implies $\|\mathbf{a}\| = 1$.) Then, with the minimum-maximum principle we get

$$\lambda_k = \min_{\substack{\mathbf{x}\neq\mathbf{0} \\ \mathbf{x}^*\mathbf{u}_1 = \cdots = \mathbf{x}^*\mathbf{u}_{k-1} = 0}} R(\mathbf{x}) \le R(\mathbf{x}_0) = \mathbf{x}_0^*A\mathbf{x}_0 = \mathbf{a}^*Q^*AQ\mathbf{a} = \sum_{i=1}^{k} |a|_i^2 \lambda'_i \le \lambda'_k.$$

■

***Exercise*:** It is possible to prove the inequalities (2.40) without assuming that the $\mathbf{q}_1, \ldots, \mathbf{q}_p$ are orthonormal. But then one has to use the eigenvalues λ'_k of

$$A'\mathbf{x} = \lambda' B\mathbf{x}, \quad B' = Q^*Q,$$

instead of (2.41). Prove this. □

Remark 2.5. Let $\mathbf{q}_i = \mathbf{e}_{j_i}$, $1 \le i \le k$. This means that we extract rows and columns $j_1, \ldots, j_k$ to construct A'. (The indices j_i are assumed to be distinct.) □

Remark 2.6. Let's remove a single row/column (with equal index) from A. Then $k = n-1$ in Remark 2.5 and the index set $j_1, \ldots, j_{n-1}$ contains all but one of the integers $1, \ldots, n$.

If we formulate a monotonicity principle based on the eigenvalues $\lambda_n, \lambda_{n-1}, \ldots$ as consecutive maxima of the Rayleigh quotient, then we arrive at the **interlacing property**

$$\lambda_k \le \lambda'_k \le \lambda_{k+1}, \qquad 1 \le k < n. \tag{2.42}$$

This interlacing property can be generalized, see, e.g., [6, 4]. □

The *trace* of a matrix $A \in \mathbb{F}^{n\times n}$ is defined to be the sum of the diagonal elements of a matrix. Matrices that are similar have equal trace. Hence, by the spectral theorem,

$$\operatorname{trace}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i. \tag{2.43}$$

The following theorem is proved in a similar way as the minimum-maximum theorem.

Theorem 2.33 ***(Trace theorem)***

$$\lambda_1 + \lambda_2 + \cdots + \lambda_p = \min_{X\in\mathbb{F}^{n\times p},\ X^*X = I_p} \operatorname{trace}(X^*AX) \tag{2.44}$$

2.11 Cholesky factorization

Definition 2.34 A Hermitian matrix is called **positive definite** (**positive semi-definite**) if all its eigenvalues are positive (nonnegative).

For a Hermitian positive definite matrix A, the LU decomposition can be written in a particular form reflecting the symmetry of A.

Theorem 2.35 ***(Cholesky factorization)*** *Let $A \in \mathbb{F}^{n\times n}$ be Hermitian positive definite. Then there is a* lower triangular *matrix L such that*

$$A = LL^*. \tag{2.45}$$

L is unique if we choose its diagonal elements to be positive.

Proof. We prove the theorem by giving an algorithm that computes the desired factorization.

Since A is positive definite, we have $a_{11} = \mathbf{e}_1^* A \mathbf{e}_1 > 0$. Therefore we can form the matrix

$$L_1 = \begin{bmatrix} l_{11}^{(1)} & & & \\ l_{21}^{(1)} & 1 & & \\ \vdots & & \ddots & \\ l_{n1}^{(1)} & & & 1 \end{bmatrix} = \begin{bmatrix} \sqrt{a_{11}} & & & \\ \frac{a_{21}}{\sqrt{a_{1,1}}} & 1 & & \\ \vdots & & \ddots & \\ \frac{a_{n1}}{\sqrt{a_{1,1}}} & & & 1 \end{bmatrix}.$$

We now form the matrix

$$A_1 = L_1^{-1} A L_1^{-1*} = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & a_{22} - \frac{a_{21}a_{12}}{a_{11}} & \dots & a_{2n} - \frac{a_{21}a_{1n}}{a_{11}} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2} - \frac{a_{n1}a_{12}}{a_{11}} & \dots & a_{nn} - \frac{a_{n1}a_{1n}}{a_{11}} \end{bmatrix}.$$

This is the first step of the algorithm. Since positive definiteness is preserved by a congruence transformation X^*AX (see also Theorem 2.37 below), A_1 is again positive definite. Hence, we can proceed in a similar fashion factorizing $A_1(2\!:\!n, 2\!:\!n)$, etc.

Collecting $L_1, L_2, \dots,$ we obtain

$$I = L_n^{-1} \cdots L_2^{-1} L_1^{-1} A (L_1^*)^{-1} (L_2^*)^{-1} \cdots (L_n^*)^{-1}$$

or

$$(L_1 L_2 \cdots L_n)(L_n^* \cdots L_2^* L_1^*) = A.$$

which is the desired result. It is easy to see that $L_1 L_2 \cdots L_n$ is a triangular matrix and that

$$L_1 L_2 \cdots L_n = \begin{bmatrix} l_{11}^{(1)} & & & & \\ l_{21}^{(1)} & l_{22}^{(2)} & & & \\ l_{31}^{(1)} & l_{32}^{(2)} & l_{33}^{(3)} & & \\ \vdots & \vdots & \vdots & \ddots & \\ l_{n1}^{(1)} & l_{n2}^{(2)} & l_{n3}^{(3)} & \dots & l_{nn}^{(n)} \end{bmatrix}$$

■

Remark 2.7. When working with symmetric matrices, one often stores only half of the matrix, e.g. the lower triangle consisting of all elements including and below the diagonal. The L-factor of the Cholesky factorization can overwrite this information in-place to save memory. □

Definition 2.36 The **inertia** of a Hermitian matrix is the triple (ν, ζ, π) where ν, ζ, π is the number of negative, zero, and positive eigenvalues.

Theorem 2.37 ***(Sylvester's law of inertia)*** *If $A \in \mathbb{C}^{n\times n}$ is Hermitian and $X \in \mathbb{C}^{n\times n}$ is nonsingular then A and X^*AX have the same inertia.*

Proof. The proof is given, for example, in [4]. ■

Remark 2.8. Two matrices A and B are called congruent if there is a nonsingular matrix X such that $B = X^*AX$. Thus, Sylvester's law of inertia can be stated in the following form: *The inertia is invariant under congruence transformations.* □

2.12 The singular value decomposition (SVD)

Theorem 2.38 ***(Singular value decomposition)*** *If $A \in \mathbb{C}^{m\times n}$ then there exist unitary matrices $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ such that*

$$U^*AV = \Sigma = \begin{pmatrix} \operatorname{diag}(\sigma_1, \ldots, \sigma_p) & 0 \\ 0 & 0 \end{pmatrix}, \qquad p = \min(m, n), \tag{2.46}$$

where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_p \geq 0$.

Proof. If $A = O$, the theorem holds with $U = I_m, V = I_n$ and Σ equal to the $m \times n$ zero matrix.

We now assume that $A \neq O$. Let $\mathbf{x}$, $\|\mathbf{x}\| = 1$, be a vector that maximizes $\|A\mathbf{x}\|$ and let $A\mathbf{x} = \sigma\mathbf{y}$ where $\sigma = \|A\| = \|A\mathbf{x}\|$ and $\|\mathbf{y}\| = 1$. As $A \neq O$, $\sigma > 0$. Consider the scalar function

$$f(\alpha) := \frac{\|A(\mathbf{x} + \alpha\mathbf{y})\|^2}{\|\mathbf{x} + \alpha\mathbf{y}\|^2} = \frac{(\mathbf{x} + \alpha\mathbf{y})^*A^*A(\mathbf{x} + \alpha\mathbf{y})}{(\mathbf{x} + \alpha\mathbf{y})^*(\mathbf{x} + \alpha\mathbf{y})}$$

Because of the extremality of $A\mathbf{x}$, the derivative $f'(\alpha)$ of $f(\alpha)$ must vanish at $\alpha = 0$. This holds *for all* $\mathbf{y}$! We have

$$\frac{df}{d\alpha}(\alpha) = \frac{(\mathbf{x}^*A^*A\mathbf{y} + \bar{\alpha}\mathbf{y}^*A^*A\mathbf{y})\|\mathbf{x} + \alpha\mathbf{y}\|^2 - (\mathbf{x}^*\mathbf{y} + \bar{\alpha}\mathbf{y}^*\mathbf{y})\|A(\mathbf{x} + \alpha\mathbf{y})\|^2}{\|\mathbf{x} + \alpha\mathbf{y}\|^4}$$

Thus, we have for all $\mathbf{y}$,

$$\left.\frac{df}{d\alpha}(\alpha)\right|_{\alpha=0} = \frac{\mathbf{x}^*A^*A\mathbf{y}\|\mathbf{x}\|^2 - \mathbf{x}^*\mathbf{y}\|A(\mathbf{x})\|^2}{\|\mathbf{x}\|^4} = 0.$$

As $\|\mathbf{x}\| = 1$ and $\|A\mathbf{x}\| = \sigma$, we have

$$(\mathbf{x}^*A^*A - \sigma^2\mathbf{x}^*)\mathbf{y} = (A^*A\mathbf{x} - \sigma^2\mathbf{x})^*\mathbf{y} = 0, \qquad \text{for all } \mathbf{y},$$

from which

$$A^*A\mathbf{x} = \sigma^2\mathbf{x}$$

follow. Multiplying $A\mathbf{x} = \sigma\mathbf{y}$ from the left by A^* we get $A^*A\mathbf{x} = \sigma A^*\mathbf{y} = \sigma^2\mathbf{x}$ from which

$$A^*\mathbf{y} = \sigma\mathbf{x}$$

and $AA^*\mathbf{y} = \sigma A\mathbf{x} = \sigma^2\mathbf{y}$ follows. Therefore, $\mathbf{x}$ is an eigenvector of A^*A corresponding to the eigenvalue σ^2 and $\mathbf{y}$ is an eigenvector of AA^* corresponding to the same eigenvalue.

Now, we construct a unitary matrix U_1 with first column $\mathbf{y}$ and a unitary matrix V_1 with first column $\mathbf{x}$, $U_1 = [\mathbf{y}, \bar{U}]$ and $V_1 = [\mathbf{x}, \bar{V}]$. Then

$$U_1^* A V_1 = \left[\begin{array}{cc} \mathbf{y}^* A\mathbf{x} & \mathbf{y}^* A\overline{V} \\ \overline{U}^* A\mathbf{x} & \overline{U}^* A\overline{V} \end{array}\right] = \left[\begin{array}{cc} \sigma & \sigma\mathbf{x}^*\overline{V} \\ \sigma\overline{U}^*\mathbf{y} & \overline{U}^* A\overline{V} \end{array}\right] = \left[\begin{array}{cc} \sigma & \mathbf{0}^* \\ \mathbf{0} & \hat{A} \end{array}\right]$$

where $\hat{A} = \overline{U}^* A \overline{V}$. ■

The proof above is due to W. Gragg. It nicely shows the relation of the singular value decomposition with the spectral decomposition of the Hermitian matrices A^*A and AA^*,

$$A = U\Sigma V^* \quad \Longrightarrow \quad A^*A = V\Sigma^2 V^*. \qquad AA^* = U\Sigma^2 U^*, \tag{2.47}$$

Note that the proof given in [4] is shorter and maybe more elegant.

The SVD of dense matrices is computed in a way that is very similar to the dense Hermitian eigenvalue problem. However, in the presence of roundoff error, it is not advisable to make use of the matrices A^*A and AA^*. Instead, let us consider the $(n+m)\times(n+m)$ Hermitian matrix

$$\begin{bmatrix} O & A \\ A^* & O \end{bmatrix}. \tag{2.48}$$

Making use of the SVD (2.46) we immediately get

$$\begin{bmatrix} O & A \\ A^* & O \end{bmatrix} = \begin{bmatrix} U & O \\ O & V \end{bmatrix} \begin{bmatrix} O & \Sigma \\ \Sigma^T & O \end{bmatrix} \begin{bmatrix} U^* & O \\ O & V^* \end{bmatrix}.$$

Now, let us assume that $m \geq n$. Then we write $U = [U_1, U_2]$ where $U_1 \in \mathbb{F}^{m\times n}$ and $\Sigma = \left[\begin{array}{c} \Sigma_1 \\ O \end{array}\right]$ with $\Sigma_1 \in \mathbb{R}^{n\times n}$. Then

$$\begin{bmatrix} O & A \\ A^* & O \end{bmatrix} = \begin{bmatrix} U_1 & U_2 & O \\ O & O & V \end{bmatrix} \begin{bmatrix} O & O & \Sigma_1 \\ O & O & O \\ \Sigma_1 & O & O \end{bmatrix} \begin{bmatrix} U_1^* & O \\ U_2^* & O \\ O & V^* \end{bmatrix} = \begin{bmatrix} U_1 & O & U_2 \\ O & V & O \end{bmatrix} \begin{bmatrix} O & \Sigma_1 & O \\ \Sigma_1 & O & O \\ O & O & O \end{bmatrix} \begin{bmatrix} U_1^* & O \\ O & V^* \\ U_2^* & O \end{bmatrix}.$$

The first and third diagonal zero blocks have order n. The middle diagonal block has order $n - m$. Now we employ the fact that

$$\left[\begin{array}{cc} 0 & \sigma \\ \sigma & 0 \end{array}\right] = \frac{1}{\sqrt{2}} \left[\begin{array}{cc} 1 & 1 \\ 1 & -1 \end{array}\right] \left[\begin{array}{cc} \sigma & 0 \\ 0 & -\sigma \end{array}\right] \frac{1}{\sqrt{2}} \left[\begin{array}{cc} 1 & 1 \\ 1 & -1 \end{array}\right]$$

to obtain

$$\begin{bmatrix} O & A \\ A^* & O \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{2}}U_1 & \frac{1}{\sqrt{2}}U_1 & U_2 \\ \frac{1}{\sqrt{2}}V & -\frac{1}{\sqrt{2}}V & O \end{bmatrix} \begin{bmatrix} \Sigma_1 & O & O \\ O & -\Sigma_1 & O \\ O & O & O \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}}U_1^* & \frac{1}{\sqrt{2}}V^* \\ \frac{1}{\sqrt{2}}U_1^* & -\frac{1}{\sqrt{2}}V^* \\ U_2^* & O \end{bmatrix}. \tag{2.49}$$

Thus, there are three ways how to treat the computation of the singular value decomposition as an eigenvalue problem. One of the two forms in (2.47) is used *implicitly* in the QR algorithm for dense matrices A, see [4],[1]. The form (2.48) is suited if A is a sparse matrix.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide – Release 2.0*, SIAM, Philadelphia, PA, 1994. (Software and guide are available from Netlib at URL http://www.netlib.org/lapack/).

[2] R. Fletcher and D. C. Sorensen, *An algorithmic derivation of the Jordan canonical form*, Amer. Math. Monthly, 90 (1983), pp. 12–16.

[3] G. E. Forsythe and W. R. Wasow, *Finite-difference methods for partial differential equations*, Wiley, New York, 1960.

[4] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[5] N. J. Higham, *Functions of matrices*, SIAM, Philadelphia, PA, 2008.

[6] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, 1985.

[7] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, SIAM, Philadelphia, PA, 2007.

[8] Y. Saad, *Numerical Methods for Large Eigenvalue Problems*, SIAM, Philadelphia, PA, 2011.

[9] R. Zurmühl, *Matrizen und ihre technischen Anwendungen*, Springer, Berlin, 4th ed., 1964.

Chapter 3

Newton methods

3.1 Linear and nonlinear eigenvalue problems

In **linear** eigenvalue problems we have to find values $\lambda \in \mathbb{C}$ such that $\lambda I - A$ is **singular**. Here $A \in \mathbb{F}^{n\times n}$ is a given real or complex matrix. Equivalently, we have to find values $\lambda \in \mathbb{C}$ such that there is a nontrivial (nonzero) $\mathbf{x}$ that satisfies

$$(A - \lambda I)\mathbf{x} = \mathbf{0} \quad \Longleftrightarrow \quad A\mathbf{x} = \lambda\mathbf{x}. \tag{3.1}$$

In the linear eigenvalue problem (3.1) the eigenvalue λ appears linearly. However, as the unknown λ is multiplied with the unknown vector $\mathbf{x}$, the problem is in fact nonlinear. We have $n+1$ unknowns $\lambda, x_1, \ldots, x_n$ that are not uniquely defined by the n equations in (3.1). We have noticed earlier, that the *length* of the eigenvector $\mathbf{x}$ is not determined. This can be rectified by adding a further equation that fixes the length of $\mathbf{x}$. The straightforward condition is

$$\|\mathbf{x}\|^2 = \mathbf{x}^*\mathbf{x} = 1, \tag{3.2}$$

that determines $\mathbf{x}$ up to a complex scalar of modulus 1, in the real case ±1. Another condition to normalize $\mathbf{x}$ is by requesting that

$$\mathbf{c}^T\mathbf{x} = 1, \qquad \text{for some } \mathbf{c}. \tag{3.3}$$

Eq. (3.3) is linear in $\mathbf{x}$ and thus simpler. However, the combined equations (3.1)–(3.3) are nonlinear anyway. Furthermore, $\mathbf{c}$ must be chosen such that it has a strong component in the (unknown) direction of the searched eigenvector. This requires some knowledge about the solution.

In **nonlinear** eigenvalue problems we have to find values $\lambda \in \mathbb{C}$ such that

$$A(\lambda)\mathbf{x} = \mathbf{0} \tag{3.4}$$

where $A(\lambda)$ is a matrix the elements of which depend on λ in a nonlinear way. An example is a matrix polynomial,

$$A(\lambda) = \sum_{k=0}^{d} \lambda^k A_k, \qquad A_k \in \mathbb{F}^{n\times n}. \tag{3.5}$$

The linear eigenvalue problem (3.1) is a special case with $d = 1$,

$$A(\lambda) = A_0 - \lambda A_1, \qquad A_0 = A, \quad A_1 = I.$$

Quadratic eigenvalue problems of the form

$$A\mathbf{x} + \lambda K\mathbf{x} + \lambda^2 M\mathbf{x} = \mathbf{0}. \tag{3.6}$$

Matrix polynomials can be linearized, i.e., they can be transformed in a linear eigenvalue of bigger size. The quadratic eigenvalue problem (3.6) can be transformed in a linear eigenvalue problem of size $2n$. Setting $\mathbf{y} = \lambda\mathbf{x}$ we get

$$\begin{pmatrix} A & O \\ O & I \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \lambda \begin{pmatrix} -K & -M \\ I & O \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$$

or

$$\begin{pmatrix} A & K \\ O & I \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} = \lambda \begin{pmatrix} O & -M \\ I & O \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}.$$

Notice that many other linearizations are possible [2, 6]. Notice also the relation with the transformation of high order to first order ODE's [5, p. 478].

Instead of looking at the nonlinear system (3.1) (complemented with (3.3) or (3.2)) we may look at the nonlinear scalar equation

$$f(\lambda) := \det\, A(\lambda) = 0 \tag{3.7}$$

and apply some zero finder. Here the question arises how to compute $f(\lambda)$ and in particular $f'(\lambda) = \frac{d}{d\lambda} \det\, A(\lambda)$.

3.2 Zeros of the determinant

We first consider the computation of eigenvalues and subsequently eigenvectors by means of computing zeros of the determinant $\det(A(\lambda))$.

Gaussian elimination with partial pivoting (GEPP) applied to $A(\lambda)$ provides the decomposition

$$P(\lambda)A(\lambda) = L(\lambda)U(\lambda), \tag{3.8}$$

where $P(\lambda)$ is the permutation matrix due to partial pivoting, $L(\lambda)$ is a lower unit triangular matrix, and $U(\lambda)$ is an upper triangular matrix. From well-known properties of the determinant function, equation (3.8) gives

$$\det P(\lambda) \cdot \det A(\lambda) = \det L(\lambda) \cdot \det U(\lambda).$$

Taking the particular structures of the factors in (3.8) into account, we get

$$f(\lambda) = \det A(\lambda) = \pm 1 \cdot \prod_{i=1}^{n} u_{ii}(\lambda). \tag{3.9}$$

The derivative of $\det A(\lambda)$ is

$$\begin{aligned} f'(\lambda) &= \pm 1 \cdot \sum_{i=1}^{n} u'_{ii}(\lambda) \prod_{j \neq i}^{n} u_{jj}(\lambda) \\ &= \pm 1 \cdot \sum_{i=1}^{n} \frac{u'_{ii}(\lambda)}{u_{ii}(\lambda)} \prod_{j=1}^{n} u_{jj}(\lambda) = \sum_{i=1}^{n} \frac{u'_{ii}(\lambda)}{u_{ii}(\lambda)} f(\lambda). \end{aligned} \tag{3.10}$$

How can we compute the derivatives u'_{ii} of the diagonal elements of $U(\lambda)$?

3.2.1 Algorithmic differentiation

A clever way to compute derivatives of a function is by **algorithmic differentiation**, see e.g., [1]. Here we assume that we have an algorithm available that computes the value $f(\lambda)$ of a function f, given the input argument λ. By algorithmic differentiation a new algorithm is obtained that computes besides $f(\lambda)$ the derivative $f'(\lambda)$.

The idea is easily explained by means of the Horner scheme to evaluate polynomials. Let

$$f(z) = \sum_{i=1}^{n} c_i z^i.$$

be a polynomial of degree n. $f(z)$ can be written in the form

$$f(z) = c_0 + z\,(c_1 + z\,(c_2 + \cdots + z\,(c_n)\cdots))$$

which gives rise to the recurrence

$$\begin{aligned} p_n &:= c_n, \\ p_i &:= z\,p_{i+1} + c_i, \qquad i = n-1, n-2, \dots, 0, \\ f(z) &:= p_0. \end{aligned}$$

Note that each of the p_i can be considered as a function (polynomial) in z. We use the above recurrence to determine the derivatives dp_i,

$$\begin{aligned} dp_n &:= 0, \quad p_n := c_n, \\ dp_i &:= p_{i+1} + z\,dp_{i+1}, \quad p_i := z\,p_{i+1} + c_i, \quad i = n-1, n-2, \dots, 0, \\ f'(z) &:= dp_0, \quad f(z) := p_0. \end{aligned}$$

We can proceed in a similar fashion for computing $\det A(\lambda)$. We however need to be able to compute the derivatives a'_{ij}. Then, we can derive each single assignment in the GEPP algorithm.

If we restrict ourselves to the standard eigenvalue problem $A\mathbf{x} = \lambda\mathbf{x}$ then $A(\lambda) = A - \lambda I$. Then, $a'_{ij} = \delta_{ij}$, the Kronecker δ.

3.2.2 Hyman's algorithm

In a Newton iteration we have to compute the determinant for possibly many values λ. Using the factorization (3.8) leads to computational costs of $\frac{2}{3}n^3$ flops (floating point operations) for each factorization, i.e., per iteration step. If this algorithm was used to compute all eigenvalues then an excessive amount of flops would be required. Can we do better?

The strategy is to transform A by a similarity transformation to a **Hessenberg matrix**, i.e., a matrix H whose entries below the lower off-diagonal are zero,

$$h_{ij} = 0, \qquad i > j + 1.$$

Any matrix A can be transformed into a similar Hessenberg matrix H by means of a sequence of elementary unitary matrices called **Householder transformations**. The details are given in Section 4.3.

Let $S^*AS = H$, where S is unitary. S is the product of the just mentioned Householder transformations. Then

$$A\mathbf{x} = \lambda\mathbf{x} \quad\Longleftrightarrow\quad H\mathbf{y} = \lambda\mathbf{y}, \qquad \mathbf{x} = S\mathbf{y}.$$

So, A and H have equal eigenvalues (A and H are similar) and the eigenvectors are transformed by S. We now assume that H is **unreduced**, i.e., $h_{i+1,i} \neq 0$ for all i. Otherwise we can split $H\mathbf{x} = \lambda\mathbf{x}$ in smaller problems.

Let λ be an eigenvalue of H and

$$(H - \lambda I)\mathbf{x} = \mathbf{0}, \tag{3.11}$$

i.e., $\mathbf{x}$ is an eigenvector of H associated with the eigenvalue λ. Then the last component of $\mathbf{x}$ cannot be zero, $x_n \neq 0$. The proof is by contradiction. Let $x_n = 0$. Then (for $n = 4$)

$$\begin{pmatrix} h_{11}-\lambda & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22}-\lambda & h_{23} & h_{24} \\ & h_{32} & h_{33}-\lambda & h_{34} \\ & & h_{43} & h_{44}-\lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

The last equation reads

$$h_{n,n-1}x_{n-1} + (h_{nn} - \lambda)\cdot 0 = 0$$

from which $x_{n-1} = 0$ follows since we assumed $h_{n,n-1} \neq 0$. In the exact same procedure we obtain $x_{n-2} = 0, \ldots, x_1 = 0$. But the zero vector cannot be an eigenvector. Therefore, x_n must not be zero. Without loss of generality we can set $x_n = 1$.

We continue to expose the procedure with a problem size $n = 4$. If λ is an eigenvalue then there are x_i, $1 \leq i < n$, such that

$$\begin{pmatrix} h_{11}-\lambda & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22}-\lambda & h_{23} & h_{24} \\ & h_{32} & h_{33}-\lambda & h_{34} \\ & & h_{43} & h_{44}-\lambda \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.12}$$

If λ is **not** an eigenvalue then we determine the x_i such that

$$\left(\begin{array}{ccc|c} h_{11}-\lambda & h_{12} & h_{13} & h_{14} \\ \hline h_{21} & h_{22}-\lambda & h_{23} & h_{24} \\ & h_{32} & h_{33}-\lambda & h_{34} \\ & & h_{43} & h_{44}-\lambda \end{array}\right) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ 1 \end{pmatrix} = \begin{pmatrix} p(\lambda) \\ 0 \\ 0 \\ 0 \end{pmatrix}. \tag{3.13}$$

We determine the $n-1$ numbers $x_{n-1}, x_{n-2}, \ldots, x_1$ by

$$x_i = \frac{-1}{h_{i+1,i}}\big((h_{i+1,i+1} - \lambda)\,x_{i+1} + h_{i+1,i+2}\,x_{i+2} + \cdots + h_{i+1,n}\underbrace{x_n}_{1}\big), \quad i = n-1, \ldots, 1.$$

The x_i are functions of λ, in fact, $x_i \in \mathbb{P}_{n-i}$. The first equation in (3.13) gives

$$(h_{1,1} - \lambda)\,x_1 + h_{1,2}\,x_2 + \cdots + h_{1,n}\,x_n = p(\lambda). \tag{3.14}$$

Eq. (3.13) can be understood by the factorization

$$\begin{pmatrix} h_{11}-\lambda & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22}-\lambda & h_{23} & h_{24} \\ & h_{32} & h_{33}-\lambda & h_{34} \\ & & h_{43} & h_{44}-\lambda \end{pmatrix} \begin{pmatrix} 1 & & & x_1 \\ & 1 & & x_2 \\ & & 1 & x_3 \\ & & & 1 \end{pmatrix}$$
$$= \begin{pmatrix} h_{11}-\lambda & h_{12} & h_{13} & p(\lambda) \\ h_{21} & h_{22}-\lambda & h_{23} & 0 \\ & h_{32} & h_{33}-\lambda & 0 \\ & & h_{43} & 0 \end{pmatrix}.$$

The last column of this equation corresponds to (3.13). Taking determinants yields

$$\det(H-\lambda I) = (-1)^{n-1} \left(\prod_{i=1}^{n-1} h_{i+1,i} \right) p(\lambda) = c \cdot p(\lambda).$$

So, $p(\lambda)$ is a constant multiple of the determinant of $H-\lambda I$. Therefore, we can solve $p(\lambda)=0$ instead of $\det(H-\lambda I)=0$.

Since the quantities $x_1, x_2, \ldots, x_n$ and thus $p(\lambda)$ are differentiable functions of λ, we can algorithmically differentiate to get $p'(\lambda)$.

For $i = n-1, \ldots, 1$ we have

$$x_i' = \frac{-1}{h_{i+1,i}} \big(-x_{i+1} + (h_{i+1,i+1}-\lambda)\, x_{i+1}' + h_{i+1,i+2}\, x_{i+2}' + \cdots + h_{i+1,n-1} x_{n-1}' \big).$$

Finally,

$$c \cdot f'(\lambda) = -x_1 + (h_{1,n}-\lambda)\, x_1' + h_{1,2}\, x_2' + \cdots + h_{1,n-1} x_{n-1}'.$$

Algorithm 3.2.2 implements Hyman's algorithm that returns $p(\lambda)$ and $p'(\lambda)$ given an input parameter λ [7].

Algorithm 3.1 Hyman's algorithm

```
Choose a value $\lambda$.
$x_n := 1$; $dx_n := 0$;
for $i = n-1$ downto $1$ do
    $s = (\lambda - h_{i+1,i+1})\, x_{i+1}$; $ds = x_{i+1} + (\lambda - h_{i+1,i+1})\, dx_{i+1}$;
    for $j = i+2$ to $n$ do
        $s = s - h_{i+1,j} x_j$; $ds = ds - h_{i+1,j} dx_j$;
    end for
    $x_i = s/h_{i+1,i}$; $dx_i = ds/h_{i+1,i}$;
end for
$s = -(\lambda - h_{1,1}) x_1$; $ds = -x_1 - (\lambda - h_{1,1}) dx_1$;
for $i = 2$ to $n$ do
    $s = s + h_{1,i} x_i$; $ds = ds + h_{1,i} dx_i$;
end for
$p(\lambda) := s$; $p'(\lambda) := ds$;
```

This algorithm computes $p(\lambda) = c' \cdot \det(H(\lambda))$ and its derivative $p'(\lambda)$ of a Hessenberg matrix H in $\mathcal{O}(n^2)$ operations. Inside a Newton iteration the new iterate is obtained by

$$\lambda_{k+1} = \lambda_k - \frac{p(\lambda_k)}{p'(\lambda_k)}, \qquad k = 0, 1, \ldots$$

The factor c' cancels. An initial guess λ_0 has to be chosen to start the iteration. It is clear that a good guess reduces the iteration count of the Newton method. The iteration is considered converged if $f(\lambda_k) \approx 0$. The vector $\mathbf{x} = (x_1, x_2, \ldots, x_{n-1}, 1)^T$ is a good approximation of the corresponding eigenvector.

Remark 3.1. Higher order deriatives of f can be computed in an analogous fashion. Higher order zero finders (e.g. Laguerre's zero finder) are then applicable [3]. □

3.2.3 Computing multiple zeros

If we have found a zero z of $f(x) = 0$ and want to compute another one, we want to avoid recomputing the already found z.

We can **explicitly deflate** the zero by defining a new function

$$f_1(x) := \frac{f(x)}{x - z}, \tag{3.15}$$

and apply our method of choice to f_1. This procedure can in particular be done with polynomials. The coefficients of f_1 are however very sensitive to inaccuracies in z. We can proceed similarly for multiple zeros $z_1, \ldots, z_m$. Explicit deflation is not recommended and often not feasible since f is not given explicitely.

For the reciprocal Newton correction for f_1 in (3.15) we get

$$\frac{f_1'(x)}{f_1(x)} = \frac{\frac{f'(x)}{x-z} - \frac{f(x)}{(x-z)^2}}{\frac{f(x)}{x-z}} = \frac{f'(x)}{f(x)} - \frac{1}{x - z}.$$

Then a Newton correction becomes

$$x^{(k+1)} = x_k - \frac{1}{\frac{f'(x_k)}{f(x_k)} - \frac{1}{x_k - z}} \tag{3.16}$$

and similarly for multiple zeros $z_1, \ldots, z_m$. Working with (3.16) is called **implicit deflation**. Here, f is not modified. In this way errors in z are not propagated to f_1

3.3 Newton methods for the constrained matrix problem

We consider the nonlinear eigenvalue problem (3.4) equipped with the normalization condition (3.3),

$$\begin{aligned} T(\lambda)\,\mathbf{x} &= \mathbf{0}, \\ \mathbf{c}^T\mathbf{x} &= 1, \end{aligned} \tag{3.17}$$

where $\mathbf{c}$ is some given vector. At a solution $(\mathbf{x}, \lambda)$, $\mathbf{x} \neq \mathbf{0}$, $T(\lambda)$ is singular. Note that $\mathbf{x}$ is defined only up to a (nonzero) multiplicative factor. $\mathbf{c}^T\mathbf{x} = 1$ is just a way to normalize $\mathbf{x}$. Another one would be $\|\mathbf{x}\|_2 = 1$, cf. the next section.

Solving (3.17) is equivalent with finding a zero of the nonlinear function $\mathbf{f}(\mathbf{x}, \lambda)$,

$$\mathbf{f}(\mathbf{x}, \lambda) = \begin{pmatrix} T(\lambda)\,\mathbf{x} \\ \mathbf{c}^T\mathbf{x} - 1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}. \tag{3.18}$$

To apply Newton's zero finding method we need the Jacobian of $\mathbf{f}$,

$$J(\mathbf{x}, \lambda) \equiv \frac{\partial \mathbf{f}(\mathbf{x}, \lambda)}{\partial(\mathbf{x}, \lambda)} = \begin{pmatrix} T(\lambda) & T'(\lambda)\mathbf{x} \\ \mathbf{c}^T & 0 \end{pmatrix}. \tag{3.19}$$

Here, $T'(\lambda)$ denotes the (elementwise) derivative of T with respect to λ. Then, a step of Newton's iteration is given by

$$\begin{pmatrix}\mathbf{x}_{k+1}\\ \lambda_{k+1}\end{pmatrix} = \begin{pmatrix}\mathbf{x}_{k}\\ \lambda_{k}\end{pmatrix} - J(\mathbf{x}_k,\lambda_k)^{-1}\mathbf{f}(\mathbf{x}_k,\lambda_k), \tag{3.20}$$

or, with the abbreviations $T_k := T(\lambda_k)$ and $T'_k := T'(\lambda_k)$,

$$\begin{pmatrix}T_k & T'_k\,\mathbf{x}_k\\ \mathbf{c}^T & 0\end{pmatrix}\begin{pmatrix}\mathbf{x}_{k+1}-\mathbf{x}_k\\ \lambda_{k+1}-\lambda_k\end{pmatrix} = \begin{pmatrix}-T_k\,\mathbf{x}_k\\ 1-\mathbf{c}^T\mathbf{x}_k\end{pmatrix}. \tag{3.21}$$

If $\mathbf{x}_k$ is normalized, $\mathbf{c}^T\mathbf{x}_k = 1$, then the second equation in (3.21) yields

$$\mathbf{c}^T(\mathbf{x}_{k+1}-\mathbf{x}_k) = 0 \quad\Longleftrightarrow\quad \mathbf{c}^T\mathbf{x}_{k+1} = 1. \tag{3.22}$$

The first equation in (3.21) gives

$$T_k\,(\mathbf{x}_{k+1}-\mathbf{x}_k) + (\lambda_{k+1}-\lambda_k)\,T'_k\,\mathbf{x}_k = -T_k\,\mathbf{x}_k \quad\Longleftrightarrow\quad T_k\,\mathbf{x}_{k+1} = -(\lambda_{k+1}-\lambda_k)\,T'_k\,\mathbf{x}_k.$$

We introduce the auxiliary vector $\mathbf{u}_{k+1}$ by

$$T_k\,\mathbf{u}_{k+1} = T'_k\,\mathbf{x}_k. \tag{3.23}$$

Note that

$$\mathbf{x}_{k+1} = -(\lambda_{k+1}-\lambda_k)\,\mathbf{u}_{k+1}. \tag{3.24}$$

So, $\mathbf{u}_{k+1}$ points in the desired direction; it just needs to be normalized. Premultiplying (3.24) by $\mathbf{c}^T$ and using (3.22) gives

$$1 = \mathbf{c}^T\mathbf{x}_{k+1} = -(\lambda_{k+1}-\lambda_k)\,\mathbf{c}^T\mathbf{u}_{k+1},$$

or

$$\lambda_{k+1} = \lambda_k - \frac{1}{\mathbf{c}^T\mathbf{u}_{k+1}}. \tag{3.25}$$

In summary, we get the following procedure.

Algorithm 3.2 Newton iteration for solving (3.18)

```
Choose a starting vector $\mathbf{x}_0 \in \mathbb{R}^n$ with $\mathbf{c}^T\mathbf{x}_0 = 1$. Set $k := 0$.
repeat
    Solve $T(\lambda_k)\,\mathbf{u}_{k+1} := T'(\lambda_k)\,\mathbf{x}_k$ for $\mathbf{u}_{k+1}$;        (3.23)
    $\mu_k := \mathbf{c}^T\mathbf{u}_{k+1}$;
    $\mathbf{x}_{k+1} := \mathbf{u}_{k+1}/\mu_k$;        (Normalize $\mathbf{u}_{k+1}$)
    $\lambda_{k+1} := \lambda_k - 1/\mu_k$;        (3.25)
    $k := k + 1$;
until some convergence criterion is satisfied
```

If the linear eigenvalue problem is solved by Algorithm 3.3 then $T'(\lambda)\mathbf{x} = \mathbf{x}$. In each iteration step a linear system has to be solved which requires the factorization of a matrix.

We now change the way we normalize $\mathbf{x}$. Problem (3.17) becomes

$$T(\lambda)\,\mathbf{x} = \mathbf{0}, \qquad \|\mathbf{x}\|^2 = 1, \tag{3.26}$$

with the corresponding nonlinear system of equations

$$\mathbf{f}(\mathbf{x},\lambda) = \begin{pmatrix} T(\lambda)\,\mathbf{x} \\ \frac{1}{2}(\mathbf{x}^T\mathbf{x}-1) \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}. \tag{3.27}$$

The Jacobian now is

$$J(\mathbf{x},\lambda) \equiv \frac{\partial \mathbf{f}(\mathbf{x},\lambda)}{\partial(\mathbf{x},\lambda)} = \begin{pmatrix} T(\lambda) & T'(\lambda)\,\mathbf{x} \\ \mathbf{x}^T & 0 \end{pmatrix}. \tag{3.28}$$

The Newton step (3.20) is changed into

$$\begin{pmatrix} T_k & T'_k\,\mathbf{x}_k \\ \mathbf{x}_k^T & 0 \end{pmatrix} \begin{pmatrix} \mathbf{x}_{k+1}-\mathbf{x}_k \\ \lambda_{k+1}-\lambda_k \end{pmatrix} = \begin{pmatrix} -T_k\,\mathbf{x}_k \\ \frac{1}{2}(1-\mathbf{x}_k^T\mathbf{x}_k) \end{pmatrix}. \tag{3.29}$$

If $\mathbf{x}_k$ is normalized, $\|\mathbf{x}_k\| = 1$, then the second equation in (3.29) gives

$$\mathbf{x}_k^T(\mathbf{x}_{k+1}-\mathbf{x}_k) = 0 \quad \Longleftrightarrow \quad \mathbf{x}_k^T\mathbf{x}_{k+1} = 1. \tag{3.30}$$

The correction $\Delta\mathbf{x}_k := \mathbf{x}_{k+1} - \mathbf{x}_k$ is orthogonal to the actual approximation. The first equation in (3.29) is the same as in (3.21). Again, we employ the auxiliary vector $\mathbf{u}_{k+1}$ defined in (3.23). Premultiplying (3.24) by $\mathbf{x}_k^T$ and using (3.30) gives

$$1 = \mathbf{x}_k^T\mathbf{x}_{k+1} = -(\lambda_{k+1}-\lambda_k)\,\mathbf{x}_k^T\mathbf{u}_{k+1},$$

or

$$\lambda_{k+1} = \lambda_k - \frac{1}{\mathbf{x}_k^T\mathbf{u}_{k+1}}. \tag{3.31}$$

The next iterate $\mathbf{x}_{k+1}$ is obtained by normalizing $\mathbf{u}_{k+1}$,

$$\mathbf{x}_{k+1} = \mathbf{u}_{k+1}/\|\mathbf{u}_{k+1}\|. \tag{3.32}$$

Algorithm 3.3 Newton iteration for solving (3.27)

```
1: Choose a starting vector $\mathbf{x}_0 \in \mathbb{R}^n$ with $\|\mathbf{x}^{(0)}\| = 1$. Set $k := 0$.
2: repeat
3:    Solve $T(\lambda_k)\,\mathbf{u}_{k+1} := T'(\lambda_k)\,\mathbf{x}_k$ for $\mathbf{u}_{k+1}$;        (3.23)
4:    $\mu_k := \mathbf{x}_k^T\mathbf{u}_{k+1}$;
5:    $\lambda_{k+1} := \lambda_k - 1/\mu_k$;        (3.31)
6:    $\mathbf{x}_{k+1} := \mathbf{u}_{k+1}/\|\mathbf{u}_{k+1}\|$;        (Normalize $\mathbf{u}_{k+1}$)
7:    $k := k+1$;
8: until some convergence criterion is satisfied
```

3.4 Successive linear approximations

Ruhe [4] suggested the following method which is not derived as a Newton method. It is based on an expansion of $T(\lambda)$ at some approximate eigenvalue λ_k.

$$T(\lambda)\mathbf{x} \approx (T(\lambda_k) - \vartheta T'(\lambda_k))\mathbf{x} = \mathbf{0}, \qquad \lambda = \lambda_k - \vartheta. \tag{3.33}$$

Algorithm 3.4 Algorithm of successive linear problems

```
Start with approximation $\lambda_1$ of an eigenvalue of $T(\lambda)$.
for $k = 1, 2, \ldots$ do
    Solve the linear eigenvalue problem $T(\lambda)\mathbf{u} = \vartheta T'(\lambda)\mathbf{u}$.
    Choose an eigenvalue $\vartheta$ smallest in modulus.
    $\lambda_{k+1} := \lambda_k - \vartheta$;
end for
```

Equation (3.33) is a generalized eigenvalue problem with eigenvalue ϑ. If λ_k is a good approximation of an eigenvalue, then it is straightforward to compute the smallest eigenvalue ϑ of

$$T(\lambda_k)\mathbf{x} = \vartheta\, T'(\lambda_k)\mathbf{x} \tag{3.34}$$

and update λ_k by $\lambda_{k+1} = \lambda_k - \vartheta$.

Remark: If T is twice continuously differentiable, and λ is an eigenvalue of problem (1) such that $T'(\lambda)$ is singular and 0 is an algebraically simple eigenvalue of $T'(\lambda)^{-1}T(\lambda)$, then the method in Algorithm 3.4 converges quadratically towards λ.

Bibliography

[1] P. Arbenz and W. Gander, *Solving nonlinear eigenvalue problems by algorithmic differentiation*, Computing, 36 (1986), pp. 205–215.

[2] D. S. Mackey, N. Mackey, C. Mehl, and V. Mehrmann, *Structured polynomial eigenvalue problems: Good vibrations from good linearizations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 1029–1051.

[3] B. Parlett, *Laguerre's method applied to the matrix eigenvalue problem*, Math. Comput., 18 (1964), pp. 464–485.

[4] A. Ruhe, *Algorithms for the nonlinear eigenvalue problem*, SIAM J. Numer. Anal., 10 (1973), pp. 674–689.

[5] G. Strang, *Introduction to Applied Mathematics*, Wellesley-Cambridge Press, Wellesley, 1986.

[6] F. Tisseur and K. Meerbergen, *The quadratic eigenvalue problem*, SIAM Rev., 43 (2001), pp. 235–286.

[7] R. C. Ward, *The QR algorithm and Hyman's method on vector computers*, Math. Comput., 30 (1976), pp. 132–142.

Chapter 4

The QR Algorithm

The QR algorithm computes a Schur decomposition of a matrix. It is certainly one of the most important algorithm in eigenvalue computations [9]. However, it is applied to *dense* (or: *full*) matrices only.

The QR algorithm consists of two separate stages. First, by means of a similarity transformation, the original matrix is transformed in a finite number of steps to Hessenberg form or – in the Hermitian/symmetric case – to real tridiagonal form. This first stage of the algorithm prepares its second stage, the actual QR iterations that are applied to the Hessenberg or tridiagonal matrix. The overall complexity (number of floating points) of the algorithm is $\mathcal{O}(n^3)$, which we will see is not entirely trivial to obtain.

The major limitation of the QR algorithm is that already the first stage generates usually complete fill-in in general sparse matrices. It can therefore not be applied to large sparse matrices, simply because of excessive memory requirements. On the other hand, the QR algorithm computes all eigenvalues (and eventually eigenvectors) which is rarely desired in sparse matrix computations anyway.

The treatment of the QR algorithm in these lecture notes on large scale eigenvalue computation is justified in two respects. First, there are of course large or even huge *dense* eigenvalue problems. Second, the QR algorithm is employed in most other algorithms to solve 'internal' small auxiliary eigenvalue problems.

4.1 The basic QR algorithm

In 1958 Rutishauser [10] of ETH Zurich experimented with a similar algorithm that we are going to present, but based on the LR factorization, i.e., based on Gaussian elimination without pivoting. That algorithm was not successful as the LR factorization (nowadays called LU factorization) is not stable without pivoting. Francis [5] noticed that the QR factorization would be the preferred choice and devised the QR algorithm with many of the bells and whistles used nowadays.

Before presenting the complete picture, we start with a basic iteration, given in Algorithm 4.1, discuss its properties and improve on it step by step until we arrive at Francis' algorithm.

We notice first that

$$A_k = R_k Q_k = Q_k^* A_{k-1} Q_k, \tag{4.1}$$

and hence A_k and A_{k-1} are unitarily similar. The matrix sequence $\{A_k\}$ converges (under certain assumptions) towards an upper triangular matrix [11]. Let us assume that the

Algorithm 4.1 Basic QR algorithm

```
Let $A \in \mathbb{C}^{n\times n}$. This algorithm computes an upper triangular matrix $T$ and a unitary
matrix $U$ such that $A = UTU^*$ is the Schur decomposition of $A$.
Set $A_0 := A$ and $U_0 = I$.
for $k = 1, 2, \ldots$ do
    $A_{k-1} =: Q_k R_k$; /* QR factorization */
    $A_k := R_k Q_k$;
    $U_k := U_{k-1} Q_k$; /* Update transformation matrix */
end for
Set $T := A_\infty$ and $U := U_\infty$.
```

eigenvalues are mutually different in magnitude and we can therefore number the eigenvalues such that $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. Then – as we will show in Chapter 8 – the elements of A_k below the diagonal converge to zero like

$$|a_{ij}^{(k)}| = \mathcal{O}(|\lambda_i/\lambda_j|^k), \qquad i > j. \tag{4.2}$$

From (4.1) we see that

$$A_k = Q_k^* A_{k-1} Q_k = Q_k^* Q_{k-1}^* A_{k-2} Q_{k-1} Q_k = \cdots = Q_k^* \cdots Q_1^* A_0 \underbrace{Q_1 \cdots Q_k}_{U_k}. \tag{4.3}$$

With the same assumption on the eigenvalues, A_k tends to an upper triangular matrix and U_k converges to the matrix of Schur vectors.

4.1.1 Numerical experiments

We conduct two MATLAB experiments to illustrate the convergence rate given in (4.2). To that end, we construct a random 4×4 matrix with eigenvalues 1, 2, 3, and 4.

```
D = diag([4 3 2 1]);
rand('seed',0);
format short e
S=rand(4); S = (S - .5)*2;
A = S*D/S      % A_0 = A = S*D*S^{-1}
for i=1:20,
   [Q,R] = qr(A);  A = R*Q
end
```

This yields the matrix sequence

```
A( 0) =  [ -4.4529e-01   4.9063e+00  -8.7871e-01   6.3036e+00]
         [ -6.3941e+00   1.3354e+01   1.6668e+00   1.1945e+01]
         [  3.6842e+00  -6.6617e+00  -6.0021e-02  -7.0043e+00]
         [  3.1209e+00  -5.2052e+00  -1.4130e+00  -2.8484e+00]

A( 1) =  [  5.9284e+00   1.6107e+00   9.3153e-01  -2.2056e+01]
         [ -1.5294e+00   1.8630e+00   2.0428e+00   6.5900e+00]
         [  1.9850e-01   2.5660e-01   1.7088e+00   1.2184e+00]
         [  2.4815e-01   1.5265e-01   2.6924e-01   4.9975e-01]

A( 2) =  [  4.7396e+00   1.4907e+00  -2.1236e+00   2.3126e+01]
```

```
          [ -4.3101e-01   2.4307e+00   2.2544e+00  -8.2867e-01]
          [  1.2803e-01   2.4287e-01   1.6398e+00  -1.8290e+00]
          [ -4.8467e-02  -5.8164e-02  -1.0994e-01   1.1899e+00]

A( 3) =   [  4.3289e+00   1.0890e+00  -3.9478e+00  -2.2903e+01]
          [ -1.8396e-01   2.7053e+00   1.9060e+00  -1.2062e+00]
          [  6.7951e-02   1.7100e-01   1.6852e+00   2.5267e+00]
          [  1.3063e-02   2.2630e-02   7.9186e-02   1.2805e+00]

A( 4) =   [  4.1561e+00   7.6418e-01  -5.1996e+00   2.2582e+01]
          [ -9.4175e-02   2.8361e+00   1.5788e+00   2.0983e+00]
          [  3.5094e-02   1.1515e-01   1.7894e+00  -2.9819e+00]
          [ -3.6770e-03  -8.7212e-03  -5.7793e-02   1.2184e+00]

A( 5) =   [  4.0763e+00   5.2922e-01  -6.0126e+00  -2.2323e+01]
          [ -5.3950e-02   2.9035e+00   1.3379e+00  -2.5358e+00]
          [  1.7929e-02   7.7393e-02   1.8830e+00   3.2484e+00]
          [  1.0063e-03   3.2290e-03   3.7175e-02   1.1372e+00]

A( 6) =   [  4.0378e+00   3.6496e-01  -6.4924e+00   2.2149e+01]
          [ -3.3454e-02   2.9408e+00   1.1769e+00   2.7694e+00]
          [  9.1029e-03   5.2173e-02   1.9441e+00  -3.4025e+00]
          [ -2.6599e-04  -1.1503e-03  -2.1396e-02   1.0773e+00]

A( 7) =   [  4.0189e+00   2.5201e-01  -6.7556e+00  -2.2045e+01]
          [ -2.1974e-02   2.9627e+00   1.0736e+00  -2.9048e+00]
          [  4.6025e-03   3.5200e-02   1.9773e+00   3.4935e+00]
          [  6.8584e-05   3.9885e-04   1.1481e-02   1.0411e+00]

A( 8) =   [  4.0095e+00   1.7516e-01  -6.8941e+00   2.1985e+01]
          [ -1.5044e-02   2.9761e+00   1.0076e+00   2.9898e+00]
          [  2.3199e-03   2.3720e-02   1.9932e+00  -3.5486e+00]
          [ -1.7427e-05  -1.3602e-04  -5.9304e-03   1.0212e+00]

A( 9) =   [  4.0048e+00   1.2329e-01  -6.9655e+00  -2.1951e+01]
          [ -1.0606e-02   2.9845e+00   9.6487e-01  -3.0469e+00]
          [  1.1666e-03   1.5951e-02   1.9999e+00   3.5827e+00]
          [  4.3933e-06   4.5944e-05   3.0054e-03   1.0108e+00]

A(10) =   [  4.0024e+00   8.8499e-02  -7.0021e+00   2.1931e+01]
          [ -7.6291e-03   2.9899e+00   9.3652e-01   3.0873e+00]
          [  5.8564e-04   1.0704e-02   2.0023e+00  -3.6041e+00]
          [ -1.1030e-06  -1.5433e-05  -1.5097e-03   1.0054e+00]

A(11) =   [  4.0013e+00   6.5271e-02  -7.0210e+00  -2.1920e+01]
          [ -5.5640e-03   2.9933e+00   9.1729e-01  -3.1169e+00]
          [  2.9364e-04   7.1703e-03   2.0027e+00   3.6177e+00]
          [  2.7633e-07   5.1681e-06   7.5547e-04   1.0027e+00]

A(12) =   [  4.0007e+00   4.9824e-02  -7.0308e+00   2.1912e+01]
          [ -4.0958e-03   2.9956e+00   9.0396e-01   3.1390e+00]
          [  1.4710e-04   4.7964e-03   2.0024e+00  -3.6265e+00]
          [ -6.9154e-08  -1.7274e-06  -3.7751e-04   1.0014e+00]

A(13) =   [  4.0003e+00   3.9586e-02  -7.0360e+00  -2.1908e+01]
          [ -3.0339e-03   2.9971e+00   8.9458e-01  -3.1558e+00]
          [  7.3645e-05   3.2052e-03   2.0019e+00   3.6322e+00]
          [  1.7298e-08   5.7677e-07   1.8857e-04   1.0007e+00]

A(14) =   [  4.0002e+00   3.2819e-02  -7.0388e+00   2.1905e+01]
```

```
               [ -2.2566e-03   2.9981e+00   8.8788e-01   3.1686e+00]
               [  3.6855e-05   2.1402e-03   2.0014e+00  -3.6359e+00]
               [ -4.3255e-09  -1.9245e-07  -9.4197e-05   1.0003e+00]

      A(15) =  [  4.0001e+00   2.8358e-02  -7.0404e+00  -2.1902e+01]
               [ -1.6832e-03   2.9987e+00   8.8305e-01  -3.1784e+00]
               [  1.8438e-05   1.4284e-03   2.0010e+00   3.6383e+00]
               [  1.0815e-09   6.4192e-08   4.7062e-05   1.0002e+00]

      A(16) =  [  4.0001e+00   2.5426e-02  -7.0413e+00   2.1901e+01]
               [ -1.2577e-03   2.9991e+00   8.7953e-01   3.1859e+00]
               [  9.2228e-06   9.5295e-04   2.0007e+00  -3.6399e+00]
               [ -2.7039e-10  -2.1406e-08  -2.3517e-05   1.0001e+00]

      A(17) =  [  4.0000e+00   2.3503e-02  -7.0418e+00  -2.1900e+01]
               [ -9.4099e-04   2.9994e+00   8.7697e-01  -3.1917e+00]
               [  4.6126e-06   6.3562e-04   2.0005e+00   3.6409e+00]
               [  6.7600e-11   7.1371e-09   1.1754e-05   1.0000e+00]

      A(18) =  [  4.0000e+00   2.2246e-02  -7.0422e+00   2.1899e+01]
               [ -7.0459e-04   2.9996e+00   8.7508e-01   3.1960e+00]
               [  2.3067e-06   4.2388e-04   2.0003e+00  -3.6416e+00]
               [ -1.6900e-11  -2.3794e-09  -5.8750e-06   1.0000e+00]

      A(19) =  [  4.0000e+00   2.1427e-02  -7.0424e+00  -2.1898e+01]
               [ -5.2787e-04   2.9997e+00   8.7369e-01  -3.1994e+00]
               [  1.1535e-06   2.8265e-04   2.0002e+00   3.6421e+00]
               [  4.2251e-12   7.9321e-10   2.9369e-06   1.0000e+00]

      A(20) =  [  4.0000e+00   2.0896e-02  -7.0425e+00   2.1898e+01]
               [ -3.9562e-04   2.9998e+00   8.7266e-01   3.2019e+00]
               [  5.7679e-07   1.8846e-04   2.0002e+00  -3.6424e+00]
               [ -1.0563e-12  -2.6442e-10  -1.4682e-06   1.0000e+00]
```

Looking at the element-wise quotients of the last two matrices one recognizes the convergence rates claimed in (4.2).

```
      A(20)./A(19) =  [  1.0000    0.9752    1.0000   -1.0000]
                      [  0.7495    1.0000    0.9988   -1.0008]
                      [  0.5000    0.6668    1.0000   -1.0001]
                      [ -0.2500   -0.3334   -0.4999    1.0000]
```

The elements above and on the diagonal are relatively stable.

If we run the same little MATLAB script but with the initial diagonal matrix D replaced by

```
      D = diag([5 2 2 1]);
```

then we obtain

```
      A(19) =  [  5.0000e+00   4.0172e+00  -9.7427e+00  -3.3483e+01]
               [ -4.2800e-08   2.0000e+00   2.1100e-05  -4.3247e+00]
               [  1.3027e-08   7.0605e-08   2.0000e+00   2.1769e+00]
               [  8.0101e-14  -2.4420e-08   4.8467e-06   1.0000e+00]

      A(20) =  [  5.0000e+00   4.0172e+00  -9.7427e+00   3.3483e+01]
               [ -1.7120e-08   2.0000e+00   1.0536e-05   4.3247e+00]
               [  5.2106e-09   3.3558e-08   2.0000e+00  -2.1769e+00]
               [ -1.6020e-14   1.2210e-08  -2.4234e-06   1.0000e+00]
```

So, again the eigenvalues are visible on the diagonal of A_{20}. The element-wise quotients of A_{20} relative to A_{19} are

```
A(20)./A(19) =  [  1.0000   1.0000   1.0000  -1.0000]
                [  0.4000   1.0000   0.4993  -1.0000]
                [  0.4000   0.4753   1.0000  -1.0000]
                [ -0.2000  -0.5000  -0.5000   1.0000]
```

Notice that (4.2) does not state a rate for the element at position $(3,2)$.

These little numerical tests are intended to demonstrate that the convergence rates given in (4.2) are in fact seen in a real run of the basic QR algorithm. The conclusions we can draw are the following:

1. The convergence of the algorithm is slow. In fact it can be arbitrarily slow if eigenvalues are very close to each other.

2. The algorithm is expensive. Each iteration step requires the computation of the QR factorization of a full $n \times n$ matrix, i.e., each single iteration step has a complexity $\mathcal{O}(n^3)$. Even if we assume that the number of steps is proportional to n we would get an $\mathcal{O}(n^4)$ complexity. The latter assumption is not even assured, see point 1 of this discussion.

In the following we want to improve on both issues. First we want to find a matrix structure that is preserved by the QR algorithm and that lowers the cost of a single iteration step. Then, we want to improve on the convergence properties of the algorithm.

4.2 The Hessenberg QR algorithm

A matrix structure that is close to upper triangular form and that is preserved by the QR algorithm is the Hessenberg form.

Definition 4.1 A matrix H is a Hessenberg matrix if its elements below the lower off-diagonal are zero,

$$h_{ij} = 0, \qquad i > j + 1.$$

Theorem 4.2 *The Hessenberg form is preserved by the QR algorithms.*

Proof. We give a constructive proof, i.e., given a Hessenberg matrix H with QR factorization $H = QR$, we show that $\overline{H} = RQ$ is again a Hessenberg matrix.

The **Givens rotation** or **plane rotation** $G(i, j, \vartheta)$ is defined by

$$G(i,j,\vartheta) := \begin{array}{cc}
\begin{bmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & & \vdots & & \vdots \\
0 & \cdots & c & \cdots & s & \cdots & 0 \\
\vdots & & \vdots & \ddots & \vdots & & \vdots \\
0 & \cdots & -s & \cdots & c & \cdots & 0 \\
\vdots & & \vdots & & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1
\end{bmatrix}
&
\begin{array}{c}
\\ \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \\
\end{array}
\\
\begin{array}{ccccccc}
 & & \uparrow & & \uparrow & & \\
 & & i & & j & &
\end{array}
&
\end{array} \tag{4.4}$$

where $c = \cos(\vartheta)$ and $s = \sin(\vartheta)$. Pre-multiplication by $G(i, j, \vartheta)$ amounts to a counterclockwise rotation by ϑ radians in the (i, j) coordinate plane. Clearly, a Givens rotation is an orthogonal matrix. For a unitary version see [4]. If $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} = G(i,j,\vartheta)^*\mathbf{x}$, then

$$y_k = \begin{cases} cx_i - sx_j, & k = i \\ sx_i + cx_j, & k = j \\ x_k, & k \neq i,j \end{cases}$$

We can force y_j to be zero by setting

$$c = \frac{x_i}{\sqrt{|x_i|^2 + |x_j|^2}}, \qquad s = \frac{-x_j}{\sqrt{|x_i|^2 + |x_j|^2}}. \tag{4.5}$$

Thus, it is a simple matter to zero a *single specific* entry in a vector by using a Givens rotation[1].

Now, let us look at a Hessenberg matrix H. We can show the principle procedure by means of a 4×4 example.

$$H = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(1,2,\vartheta_1)^*\cdot} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix}$$

$$\xrightarrow{G(2,3,\vartheta_2)^*\cdot} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} \xrightarrow{G(3,4,\vartheta_3)^*\cdot} \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} = R$$

So, with $G_k = G(k,k+1,\vartheta_k)$, we get

$$\underbrace{G_3^* G_2^* G_1^*}_{Q^*} H = R \qquad \Longleftrightarrow \qquad H = QR.$$

Multiplying Q and R in reversed order gives

$$\overline{H} = RQ = RG_1G_2G_3,$$

or, pictorially,

$$R = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{\cdot G(1,2,\vartheta_1)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & 0 & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix}$$

$$\xrightarrow{\cdot G(2,3,\vartheta_2)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & 0 & \times \end{bmatrix} \xrightarrow{\cdot G(3,4,\vartheta_1)} \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ 0 & \times & \times & \times \\ 0 & 0 & \times & \times \end{bmatrix} = \overline{H}$$

More generally, if H is $n \times n$, $n-1$ Givens rotations $G_1, \ldots, G_{n-1}$ are needed to transform H to upper triangular form. Applying the rotations from the right restores the Hessenberg form. ■

Remark 4.1. The Hessenberg nonzero pattern isn't the only pattern that is preserved by the QR algoritm, see [2], however it is the most simple one. □

[1] For a stable way to compute Givens rotations see Algorithm 5.1.3 in [6].

4.2.1 A numerical experiment

We repeat one of the previous two MATLAB experiments

```
D = diag([4 3 2 1]);
rand('seed',0);
S=rand(4); S = (S - .5)*2;
A = S*D/S     % A_0 = A = S*D*S^{-1}
H = hess(A);  % built-in MATLAB function

for i=1:30,
   [Q,R] = qr(H);  H = R*Q
end
```

This yields the matrix sequence

```
H( 0) =  [ -4.4529e-01  -1.8641e+00  -2.8109e+00   7.2941e+00]
         [  8.0124e+00   6.2898e+00   1.2058e+01  -1.6088e+01]
         [  0.0000e+00   4.0087e-01   1.1545e+00  -3.3722e-01]
         [  0.0000e+00   0.0000e+00  -1.5744e-01   3.0010e+00]

H( 5) =  [  4.0763e+00  -2.7930e+00  -7.1102e+00   2.1826e+01]
         [  5.6860e-02   2.4389e+00  -1.2553e+00  -3.5061e+00]
         [              -2.0209e-01   2.5681e+00  -2.1805e+00]
         [                            4.3525e-02   9.1667e-01]

H(10) =  [  4.0024e+00  -6.2734e-01  -7.0227e+00  -2.1916e+01]
         [  7.6515e-03   2.9123e+00  -9.9902e-01   3.3560e+00]
         [              -8.0039e-02   2.0877e+00   3.3549e+00]
         [                           -7.1186e-04   9.9762e-01]

H(15) =  [  4.0001e+00  -1.0549e-01  -7.0411e+00   2.1902e+01]
         [  1.6833e-03   2.9889e+00  -8.9365e-01  -3.2181e+00]
         [              -1.2248e-02   2.0111e+00  -3.6032e+00]
         [                            2.0578e-05   9.9993e-01]

H(20) =  [  4.0000e+00  -3.1163e-02  -7.0425e+00  -2.1898e+01]
         [  3.9562e-04   2.9986e+00  -8.7411e-01   3.2072e+00]
         [              -1.6441e-03   2.0014e+00   3.6377e+00]
         [                           -6.3689e-07   1.0000e-00]

H(25) =  [  4.0000e+00  -2.1399e-02  -7.0428e+00   2.1897e+01]
         [  9.3764e-05   2.9998e+00  -8.7056e-01  -3.2086e+00]
         [              -2.1704e-04   2.0002e+00  -3.6423e+00]
         [                            1.9878e-08   1.0000e-00]

H(30) =  [  4.0000e+00  -2.0143e-02  -7.0429e+00  -2.1897e+01]
         [  2.2247e-05   3.0000e+00  -8.6987e-01   3.2095e+00]
         [              -2.8591e-05   2.0000e+00   3.6429e+00]
         [                           -6.2108e-10   1.0000e-00]
```

Finally we compute the element-wise quotients of the last two matrices.

```
H(30)./H(29) =  [  1.0000    0.9954    1.0000   -1.0000]
                [  0.7500    1.0000    0.9999   -1.0000]
                [            0.6667    1.0000   -1.0000]
                [                     -0.5000    1.0000]
```

Again the elements in the lower off-diagonal reflect nicely the convergence rates in (4.2).

4.2.2 Complexity

We give the algorithm for a single Hessenberg-QR-step in a MATLAB-like way, see Algorithm 4.2. By

$$H_{k:j,m:n}$$

we denote the submatrix of H consisting of rows k through j and columns m through n.

Algorithm 4.2 A Hessenberg QR step

```
Let $H \in \mathbb{C}^{n\times n}$ be an upper Hessenberg matrix. This algorithm overwrites $H$ with
  $\overline{H} = RQ$ where $H = QR$ is a QR factorization of $H$.
for $k = 1, 2, \ldots, n-1$ do
    /* Generate $G_k$ and then apply it: $H = G(k, k+1, \vartheta_k)^* H$ */
    $[c_k, s_k] :=$ givens$(H_{k,k}, H_{k+1,k})$;
    $H_{k:k+1,k:n} = \begin{bmatrix} c_k & -s_k \\ s_k & c_k \end{bmatrix} H_{k:k+1,k:n}$;
end for
for $k = 1, 2, \ldots, n-1$ do
    /* Apply the rotations $G_k$ from the right */
    $H_{1:k+1,k:k+1} = H_{1:k+1,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$;
end for
```

If we neglect the determination of the parameters c_k and s_k, see (4.5), then *each* of the two loops requires

$$\sum_{i=1}^{n-1} 6i = 6\frac{n(n-1)}{2} \approx 3n^2 \quad \text{flops.}$$

A **flop** is a floating point operation $(+, -, \times, /)$. We do not distinguish between them, although they may slightly differ in their execution time on a computer. Optionally, we also have to execute the operation $U_k := U_{k-1}Q_k$ of Algorithm 4.1. This is achieved by a loop similar to the second loop in Algorithm 4.2. Since all the rows and columns of U are involved, executing the loop costs

```
for k=1,2,...,n-1 do
    $U_{1:n,k:k+1} = U_{1:n,k:k+1} \begin{bmatrix} c_k & s_k \\ -s_k & c_k \end{bmatrix}$;
end for
```

$$\sum_{i=1}^{n-1} 6n \approx 6n^2 \quad \text{flops.}$$

Altogether, a QR step with a Hessenberg matrix, including the update of the unitary transformation matrix, requires $12n^2$ floating point operations. This has to be set in relation to a QR step with a full matrix that costs $\frac{7}{3}n^3$. Consequently, we have gained a factor of $\mathcal{O}(n)$ in terms of operations by moving from dense to Hessenberg form. However, we may still have very slow convergence if one of the quotients $|\lambda_k|/|\lambda_{k+1}|$ is close to 1.

4.3 The Householder reduction to Hessenberg form

In the previous section we discovered that it is a good idea to perform the QR algorithm with Hessenberg matrices instead of full matrices. But we have not discussed how we transform a full matrix (by means of similarity transformations) into Hessenberg form. We catch up on this issue in this section.

4.3.1 Householder reflectors

Givens rotations are designed to zero a single element in a vector. Householder reflectors are more efficient if a number of elements of a vector are to be zeroed at once. Here, we follow the presentation given in [6].

Definition 4.3 A matrix of the form

$$P = I - 2\mathbf{u}\mathbf{u}^*, \qquad \|\mathbf{u}\| = 1,$$

is called a **Householder reflector**.

It is easy to verify that Householder reflectors are *Hermitian* and that $P^2 = I$. From this we deduce that P is *unitary*. It is clear that we only have to store the **Householder vector u** to be able to multiply a vector (or a matrix) with P,

$$P\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^*\mathbf{x}). \tag{4.6}$$

This multiplication only costs $4n$ flops where n is the length of the vectors.

A task that we repeatedly want to carry out with Householder reflectors is to transform a vector $\mathbf{x}$ on a multiple of $\mathbf{e}_1$,

$$P\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^*\mathbf{x}) = \alpha\mathbf{e}_1.$$

Since P is unitary, we must have $\alpha = \rho\|\mathbf{x}\|$, where $\rho \in \mathbb{C}$ has absolute value one. Therefore,

$$\mathbf{u} = \frac{\mathbf{x} - \rho\|\mathbf{x}\|\mathbf{e}_1}{\|\mathbf{x} - \rho\|\mathbf{x}\|\mathbf{e}_1\|} = \frac{1}{\|\mathbf{x} - \rho\|\mathbf{x}\|\mathbf{e}_1\|}\begin{bmatrix} x_1 - \rho\|\mathbf{x}\| \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

We can freely choose ρ provided that $|\rho| = 1$. Let $x_1 = |x_1|e^{i\phi}$. To avoid numerical cancellation we set $\rho = -e^{i\phi}$.

In the real case, one commonly sets $\rho = -\mathrm{sign}(x_1)$. If $x_1 = 0$ we can set ρ in any way.

4.3.2 Reduction to Hessenberg form

Now we show how to use Householder reflectors to reduce an arbitrary square matrix to Hessenberg form. We show the idea by means of a 5×5 example. In the first step of the reduction we introduce zeros in the first column below the second element,

$$A = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_1 *} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{* P_1} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = P_1^* A P_1.$$

Notice that $P_1 = P_1^*$ since it is a Householder reflector! It has the structure

$$P_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & I_4 - 2\mathbf{u}_1\mathbf{u}_1^* \end{bmatrix}.$$

The Householder vector $\mathbf{u}_1$ is determined such that

$$(I - 2\mathbf{u}_1\mathbf{u}_1^*) \begin{bmatrix} a_{21} \\ a_{31} \\ a_{41} \\ a_{51} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{with} \quad \mathbf{u}_1 = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix}.$$

The multiplication of P_1 from the left inserts the desired zeros in column 1 of A. The multiplication from the right is necessary in order to have similarity. Because of the nonzero structure of P_1 the first column of $P_1 A$ is not affected. Hence, the zeros stay there.

The reduction continues in a similar way:

$$P_1 A P_1 = \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \end{bmatrix} \xrightarrow{P_2 * / * P_2} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & \times & \times & \times \end{bmatrix}$$

$$\xrightarrow{P_3 * / * P_3} \begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix} = P_3 P_2 P_1 A \underbrace{P_1 P_2 P_3}_{U}.$$

Algorithm 4.3 gives the details for the general $n \times n$ case. In step 4 of this algorithm, the Householder reflector is generated such that

$$(I - 2\mathbf{u}_k\mathbf{u}_k^*) \begin{bmatrix} a_{k+1,k} \\ a_{k+2,k} \\ \vdots \\ a_{n,k} \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \text{with} \quad \mathbf{u}_k = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-k} \end{bmatrix} \quad \text{and} \quad |\alpha| = \|\mathbf{x}\|$$

according to the considerations of the previous subsection. The Householder vectors are stored at the locations of the zeros. Therefore the matrix $U = P_1 \cdots P_{n-2}$ that effects the similarity transformation from the full A to the Hessenberg H is computed after all Householder vectors have been generated, thus saving $(2/3)n^3$ flops. The overall complexity of the reduction is

- Application of P_k from the left: $\sum\limits_{k=1}^{n-2} 4(n-k-1)(n-k) \approx \frac{4}{3}n^3$
- Application of P_k from the right: $\sum\limits_{k=1}^{n-2} 4(n)(n-k) \approx 2n^3$

Algorithm 4.3 Reduction to Hessenberg form

```
This algorithm reduces a matrix $A \in \mathbb{C}^{n\times n}$ to Hessenberg form $H$ by a sequence of
  Householder reflections. $H$ overwrites $A$.
for $k = 1$ to $n-2$ do
  Generate the Householder reflector $P_k$;
  /* Apply $P_k = I_k \oplus (I_{n-k} - 2\mathbf{u_k}\mathbf{u_k}^*)$ from the left to $A$ */
  $A_{k+1:n,k:n} := A_{k+1:n,k:n} - 2\mathbf{u_k}(\mathbf{u_k}^* A_{k+1:n,k:n})$;
  /* Apply $P_k$ from the right, $A := AP_k$ */
  $A_{1:n,k+1:n} := A_{1:n,k+1:n} - 2(A_{1:n,k+1:n}\mathbf{u_k})\mathbf{u_k}^*$;
end for
if eigenvectors are desired form $U = P_1 \cdots P_{n-2}$ then
  $U := I_n$;
  for $k = n-2$ downto 1 do
    /* Update $U := P_k U$ */
    $U_{k+1:n,k+1:n} := U_{k+1:n,k+1:n} - 2\mathbf{u_k}(\mathbf{u_k}^* U_{k+1:n,k+1:n})$;
  end for
end if
```

- Form $U = P_1 \cdots P_{n-2}$: $\sum\limits_{k=1}^{n-2} 4(n-k)(n-k) \approx \frac{4}{3}n^3$

Thus, the reduction to Hessenberg form costs $\frac{10}{3}n^3$ flops without forming the transformation matrix and $\frac{14}{3}n^3$ including forming this matrix.

4.4 Improving the convergence of the QR algorithm

We have seen how the QR algorithm for computing the Schur form of a matrix A can be executed more economically if the matrix A is first transformed to Hessenberg form. Now we want to show how the convergence of the Hessenberg QR algorithm can be improved dramatically by introducing **(spectral) shifts** into the algorithm.

Lemma 4.4 *Let H be an **irreducible** Hessenberg matrix, i.e., $h_{i+1,i} \neq 0$ for all $i = 1, \ldots, n-1$. Let $H = QR$ be the QR factorization of H. Then for the diagonal elements of R we have*

$$|r_{kk}| > 0, \qquad \textit{for all } k < n.$$

Thus, if H is singular then $r_{nn} = 0$.

Proof. Let us look at the k-th step of the Hessenberg QR factorization. For illustration, let us consider the case $k = 3$ in a 5×5 example, where the matrix has the structure

$$\begin{bmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \times & \times \end{bmatrix}.$$

The plus-signs indicate elements that have been modified. In step 3, the (nonzero) element h_{43} will be zeroed by a Givens rotation $G(3, 4, \varphi)$ that is determined such that

$$\begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix} \begin{bmatrix} \tilde{h}_{kk} \\ h_{k+1,k} \end{bmatrix} = \begin{bmatrix} r_{kk} \\ 0 \end{bmatrix}.$$

Because the Givens rotation preserves vector lengths, we have

$$|r_{kk}|^2 = |\tilde{h}_{kk}|^2 + |h_{k+1,k}|^2 \geq |h_{k+1,k}|^2 > 0,$$

which confirms the claim. ■

We apply this Lemma to motivate a further strategy to speed up the convergence of the QR algorithm.

Let λ be an eigenvalue of the irreducible Hessenberg matrix H. Let us check what happens it we perform

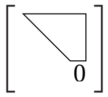
```
$H - \lambda I = QR$ /* QR factorization */
$\overline{H} = RQ + \lambda I$
```

First we notice that $\overline{H} \sim H$. In fact,

$$\overline{H} = Q^*(H - \lambda I)Q + \lambda I = Q^* H Q.$$

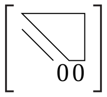
Second, by Lemma 4.4 we have

$$H - \lambda I = QR, \qquad \text{with} \qquad R =$$



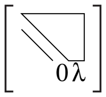
Thus,

$$RQ =$$



and

$$\overline{H} = RQ + \lambda I = \quad = \begin{bmatrix} \overline{H}_1 & \mathbf{h}_1 \\ \mathbf{0}^T & \lambda \end{bmatrix}.$$



So, if we apply a QR step with a **perfect shift** to a Hessenberg matrix, the eigenvalue drops out. We then could **deflate**, i.e., proceed the algorithm with the smaller matrix $\overline{H}_1$.

Remark 4.2. We could prove the existence of the Schur decomposition in the following way. (1) transform the arbitrary matrix to Hessenberg form. (2) Do the perfect shift Hessenberg QR with the eigenvalues which we known to exist one after the other. □

4.4.1 A numerical example

We use a matrix of a previous MATLAB experiments to show that perfect shifts actually work.

```
D = diag([4 3 2 1]); rand('seed',0);
S=rand(4); S = (S - .5)*2;
A = S*D/S;
format short e
H = hess(A)
[Q,R] = qr(H - 2*eye(4))
H1 = R*Q + 2*eye(4)
format long
lam = eig(H1(1:3,1:3))
```

Matlab produces the output

```
     H =  [ -4.4529e-01  -1.8641e+00  -2.8109e+00   7.2941e+00]
          [  8.0124e+00   6.2898e+00   1.2058e+01  -1.6088e+01]
          [               4.0087e-01   1.1545e+00  -3.3722e-01]
          [                           -1.5744e-01   3.0010e+00]

     Q =  [ -2.9190e-01  -7.6322e-01  -4.2726e-01  -3.8697e-01]
          [  9.5645e-01  -2.3292e-01  -1.3039e-01  -1.1810e-01]
          [               6.0270e-01  -5.9144e-01  -5.3568e-01]
          [                           -6.7130e-01   7.4119e-01]

     R =  [  8.3772e+00   4.6471e+00   1.2353e+01  -1.7517e+01]
          [               6.6513e-01  -1.1728e+00  -2.0228e+00]
          [                            2.3453e-01  -1.4912e+00]
          [                                        -2.4425e-14]

    H1 =  [  3.9994e+00  -3.0986e-02   2.6788e-01  -2.3391e+01]
          [  6.3616e-01   1.1382e+00   1.9648e+00  -9.4962e-01]
          [               1.4135e-01   2.8623e+00  -1.2309e+00]
          [                            1.6396e-14   2.0000e+00]

   lam =  [9.99999999999993e-01 4.00000000000003e+00 3.00000000000000e+00]
```

4.4.2 QR algorithm with shifts

This considerations indicate that it may be good to introduce shifts into the QR algorithm. However, we cannot choose perfect shifts because we do not know the eigenvalues of the matrix! We therefore need heuristics how to *estimate* eigenvalues. One such heuristic is the **Rayleigh quotient shift**: Set the shift σ_k in the k-th step of the QR algorithm equal to the last diagonal element:

$$\sigma_k := h_{n,n}^{(k-1)} = \mathbf{e}_n^* H^{(k-1)} \mathbf{e}_n. \tag{4.7}$$

Algorithm 4.4 The Hessenberg QR algorithm with Rayleigh quotient shift

```
Let $H_0 = H \in \mathbb{C}^{n\times n}$ be an upper Hessenberg matrix. This algorithm computes its
  Schur normal form $H = UTU^*$.
$k := 0$;
for m=n,n-1,...,2 do
  repeat
    $k := k + 1$;
    $\sigma_k := h_{m,m}^{(k-1)}$;
    $H_{k-1} - \sigma_k I =: Q_k R_k$;
    $H_k := R_k Q_k + \sigma_k I$;
    $U_k := U_{k-1} Q_k$;
  until $|h_{m,m-1}^{(k)}|$ is sufficiently small
end for
$T := H_k$;
```

Algorithm 4.4 implements this heuristic. Notice that the shift changes in each iteration step! Notice also that **deflation** is incorporated in Algorithm 4.4. As soon as the last lower off-diagonal element is sufficiently small, it is *declared* zero, and the algorithm proceeds with a smaller matrix. In Algorithm 4.4 the 'active portion' of the matrix is $m \times m$.

Lemma 4.4 guarantees that a zero is produced at position $(n, n-1)$ in the Hessenberg matrix H if the shift equals an eigenvalue of H. What happens, if $h_{n,n}$ is a *good* approximation to an eigenvalue of H? Let us assume that we have an irreducible Hessenberg matrix

$$\begin{bmatrix} \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times \\ 0 & 0 & \times & \times & \times \\ 0 & 0 & 0 & \varepsilon & h_{n,n} \end{bmatrix},$$

where ε is a small quantity. If we perform a shifted Hessenberg QR step, we first have to factor $H - h_{n,n}I$, $QR = H - h_{n,n}I$. After $n-2$ steps of this factorization the R-factor is almost upper triangular,

$$\begin{bmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & \alpha & \beta \\ 0 & 0 & 0 & \varepsilon & 0 \end{bmatrix}.$$

From (4.5) we see that the last Givens rotation has the nontrivial elements

$$c_{n-1} = \frac{\alpha}{\sqrt{|\alpha|^2 + |\varepsilon|^2}}, \qquad s_{n-1} = \frac{-\varepsilon}{\sqrt{|\alpha|^2 + |\varepsilon|^2}}.$$

Applying the Givens rotations from the right one sees that the last lower off-diagonal element of $\overline{H} = RQ + h_{n,n}I$ becomes

$$\bar{h}_{n,n-1} = \frac{\varepsilon^2 \beta}{\alpha^2 + \varepsilon^2}. \tag{4.8}$$

So, we have *quadratic convergence* unless α is also tiny.

A second even more often used shift strategy is the **Wilkinson shift**:

$$\sigma_k := \text{eigenvalue of } \begin{bmatrix} h^{(k-1)}_{n-1,n-1} & h^{(k-1)}_{n-1,n} \\ h^{(k-1)}_{n,n-1} & h^{(k-1)}_{n,n} \end{bmatrix} \text{ that is closer to } h^{(k-1)}_{n,n}. \tag{4.9}$$

4.4.3 A numerical example

We give an example for the Hessenberg QR algorithm with shift, but without deflation. The Matlab code

```
D = diag([4 3 2 1]);
rand('seed',0);
S=rand(4); S = (S - .5)*2;
A = S*D/S;
H = hess(A)

for i=1:8,
   [Q,R] = qr(H-H(4,4)*eye(4)); H = R*Q+H(4,4)*eye(4);
end
```

produces the output

```
H( 0) =  [ -4.4529e-01  -1.8641e+00  -2.8109e+00   7.2941e+00]
         [  8.0124e+00   6.2898e+00   1.2058e+01  -1.6088e+01]
         [  0.0000e+00   4.0087e-01   1.1545e+00  -3.3722e-01]
         [  0.0000e+00   0.0000e+00  -1.5744e-01   3.0010e+00]

H( 1) =  [  3.0067e+00   1.6742e+00  -2.3047e+01  -4.0863e+00]
         [  5.2870e-01   8.5146e-01   1.1660e+00  -1.5609e+00]
         [              -1.7450e-01   3.1421e+00  -1.1140e-01]
         [                           -1.0210e-03   2.9998e+00]

H( 2) =  [  8.8060e-01  -4.6537e-01   9.1630e-01   1.6146e+00]
         [ -1.7108e+00   5.3186e+00   2.2839e+01  -4.0224e+00]
         [              -2.2542e-01   8.0079e-01   5.2445e-01]
         [                           -1.1213e-07   3.0000e+00]

H( 3) =  [  1.5679e+00   9.3774e-01   1.5246e+01   1.2703e+00]
         [  1.3244e+00   2.7783e+00   1.7408e+01   4.1764e+00]
         [               3.7230e-02   2.6538e+00  -7.8404e-02]
         [                            8.1284e-15   3.0000e+00]

H( 4) =  [  9.9829e-01  -7.5537e-01  -5.6915e-01   1.9031e+00]
         [ -3.2279e-01   5.1518e+00   2.2936e+01  -3.9104e+00]
         [              -1.6890e-01   8.4993e-01   3.8582e-01]
         [                           -5.4805e-30   3.0000e+00]

H( 5) =  [  9.3410e-01  -3.0684e-01   3.0751e+00  -1.2563e+00]
         [  3.5835e-01   3.5029e+00   2.2934e+01   4.1807e+00]
         [               3.2881e-02   2.5630e+00  -7.2332e-02]
         [                            1.1313e-59   3.0000e+00]

H( 6) =  [  1.0005e+00  -8.0472e-01  -8.3235e-01   1.9523e+00]
         [ -7.5927e-02   5.1407e+00   2.2930e+01  -3.8885e+00]
         [              -1.5891e-01   8.5880e-01   3.6112e-01]
         [                          -1.0026e-119   3.0000e+00]

H( 7) =  [  9.7303e-01  -6.4754e-01  -8.9829e-03  -1.8034e+00]
         [  8.2551e-02   3.4852e+00   2.3138e+01   3.9755e+00]
         [               3.3559e-02   2.5418e+00  -7.0915e-02]
         [                           3.3770e-239   3.0000e+00]

H( 8) =  [  1.0002e+00  -8.1614e-01  -8.9331e-01   1.9636e+00]
         [ -1.8704e-02   5.1390e+00   2.2928e+01  -3.8833e+00]
         [              -1.5660e-01   8.6086e-01   3.5539e-01]
         [                                     0   3.0000e+00]
```

The numerical example shows that the shifted Hessenberg QR algorithm can work very nicely. In this example the (4,3) element is about 10^{-30} after 3 steps. (We could stop there.) The example also nicely shows a quadratic convergence rate.

4.5 The double shift QR algorithm

The shifted Hessenberg QR algorithm does not always work so nicely as in the previous example. If α in (4.8) is $\mathcal{O}(\varepsilon)$ then $h_{n,n-1}$ can be large. (A small α indicates a near singular $H_{1:n-1,1:n-1}$.)

Another problem occurs if real Hessenberg matrices have complex eigenvalues. We know that for reasonable convergence rates the shifts must be complex. If an eigenvalue λ has been found we can execute a single perfect shift with $\bar{\lambda}$. It is (for rounding errors) unprobable however that we will get back to a real matrix.

Since the eigenvalues come in complex conjugate pairs it is natural to search for a pair of eigenvalues right-away. This is done by collapsing two shifted QR steps in one **double step** with the two shifts being complex conjugates of each other.

Let σ_1 and σ_2 be two eigenvalues of the real matrix (cf. Wilkinson shift (4.9))

$$G = \begin{bmatrix} h_{n-1,n-1}^{(k-1)} & h_{n-1,n}^{(k-1)} \\ h_{n,n-1}^{(k-1)} & h_{n,n}^{(k-1)} \end{bmatrix} \in \mathbb{R}^{2\times 2}.$$

If $\sigma_1 \in \mathbb{C}\backslash\mathbb{R}$ then $\sigma_2 = \bar{\sigma}_1$. Let us perform two QR steps using σ_1 and σ_2 as shifts. Setting $k = 1$ for convenience we get

$$\begin{aligned} H_0 - \sigma_1 I &= Q_1 R_1, \\ H_1 &= R_1 Q_1 + \sigma_1 I, \\ H_1 - \sigma_2 I &= Q_2 R_2, \\ H_2 &= R_2 Q_2 + \sigma_2 I. \end{aligned} \tag{4.10}$$

From the second and third equation in (4.10) we obtain

$$R_1 Q_1 + (\sigma_1 - \sigma_2) I = Q_2 R_2.$$

Multiplying this equation with Q_1 from the left and with R_1 from the right we get

$$\begin{aligned} Q_1 R_1 Q_1 R_1 + (\sigma_1 - \sigma_2) Q_1 R_1 &= Q_1 R_1 (Q_1 R_1 + (\sigma_1 - \sigma_2) I) \\ &= (H_0 - \sigma_1 I)(H_0 - \sigma_2 I) = Q_1 Q_2 R_2 R_1. \end{aligned}$$

Because $\sigma_2 = \bar{\sigma}_1$ we have

$$M := (H_0 - \sigma_1 I)(H_0 - \bar{\sigma}_1 I) = H_0^2 - 2\mathrm{Re}(\sigma) H_0 + |\sigma|^2 I = Q_1 Q_2 R_2 R_1.$$

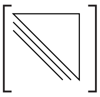
Therefore, $(Q_1 Q_2)(R_2 R_1)$ is the QR factorization of a *real matrix*. We can choose (scale) Q_1 and Q_2 such that $Z := Q_1 Q_2$ is real orthogonal. (Then also $R_2 R_1$ is real.) By consequence,

$$H_2 = (Q_1 Q_2)^* H_0 (Q_1 Q_2) = Z^T H_0 Z$$

is real.

A procedure to compute H_2 by avoiding complex arithmetic could consist of three steps:

1. Form the real matrix $M = H_0^2 - sH_0 + tI$ with $s = 2\mathrm{Re}(\sigma) = \mathrm{trace}(G) = h_{n-1,n-1}^{(k-1)} + h_{n,n}^{(k-1)}$ and $t = |\sigma|^2 = \det(G) = h_{n-1,n-1}^{(k-1)} h_{n,n}^{(k-1)} - h_{n-1,n}^{(k-1)} h_{n,n-1}^{(k-1)}$. Notice that M has two lower off-diagonals,

   $M =$  .

2. Compute the QR factorization $M = ZR$,

3. Set $H_2 = Z^T H_0 Z$.

This procedure is however too expensive since item 1, i.e., forming H^2 requires $\mathcal{O}(n^3)$ flops.

A remedy for the situation is provided by the Implicit Q Theorem.

Theorem 4.5 *(The implicit Q theorem) Let $A \in \mathbb{R}^{n\times n}$. Let $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ and $V = [\mathbf{v}_1, \ldots, \mathbf{v}_n]$ be orthogonal matrices that both similarly transform A to Hessenberg form, $H = Q^TAQ$ and $G = V^TAV$. Let k denote the smallest positive integer for which $h_{k+1,k} = 0$, with $k = n$ if H is irreducible.*

If $\mathbf{q}_1 = \mathbf{v}_1$ then $\mathbf{q}_i = \pm\mathbf{v}_i$ and $|h_{i,i-1}| = |g_{i,i-1}|$ for $i = 2, \ldots, k$. If $k < n$, then $g_{k+1,k} = 0$.

Proof. [6] Let $W = V^TQ$. Clearly, W is orthogonal, and $GW = WH$.

We first show that the first k columns of W form an upper triangular matrix, i.e.,

$$\mathbf{w}_i = W\mathbf{e}_i \in \operatorname{span}\{\mathbf{e}_1, \ldots, \mathbf{e}_i\}, \qquad i \le k. \tag{4.11}$$

(Notice that orthogonal upper triangular matrices are diagonal with diagonal entries ±1.)

This is proced inductively. For $i = 1$ we have $\mathbf{w}_1 = \mathbf{e}_1$ by the assumption that $\mathbf{q}_1 = \mathbf{v}_1$. For $1 < i \le k$ we assume that (4.11) is true for $\mathbf{w}_i$ and use the equality $GW = WH$. The $(i-1)$-th column of this equation reads

$$G\mathbf{w}_{i-1} = GW\mathbf{e}_{i-1} = WH\mathbf{e}_{i-1} = \sum_{j=1}^{i} \mathbf{w}_j h_{j,i-1}.$$

Since $h_{i,i-1} \neq 0$ we have

$$\mathbf{w}_i h_{i,i-1} = G\mathbf{w}_{i-1} - \sum_{j=1}^{i-1} \mathbf{w}_j h_{j,i-1} \in \operatorname{span}\{\mathbf{e}_1, \ldots \mathbf{e}_i\},$$

as G is a Hessenberg matrix. So, the upper-left $k \times k$ block of W is upper triangular. Since the columns of W are orthogonal we conclude that $\mathbf{w}_i = \pm\mathbf{e}_i$, $i \le k$.

Since $\mathbf{w}_i = \pm V^TQ\mathbf{e}_i = V^T\mathbf{q}_i = \pm\mathbf{e}_i$ we see that $\mathbf{q}_i$ is orthogonal to all columns of V except the i-th. Therefore, we must have $\mathbf{q}_i = \pm\mathbf{v}_i$. Further,

$$h_{i,i-1} = \mathbf{e}_i^T H\mathbf{e}_{i-1} = \mathbf{e}_i^T Q^TAQ\mathbf{e}_{i-1} = \mathbf{e}_i^T Q^TVGV^TQ\mathbf{e}_{i-1} = \mathbf{w}_i^T G\mathbf{w}_{i-1} = \pm g_{i,i-1},$$

thus, $|h_{i,i-1}| = |g_{i,i-1}|$. If $h_{k+1,k} = 0$ then

$$g_{k+1,k} = \mathbf{e}_{k+1}^T G\mathbf{e}_k = \pm\,\mathbf{e}_{k+1}^T GW\mathbf{e}_k = \pm\,\mathbf{e}_{k+1}^T WH\mathbf{e}_k = \pm\,\mathbf{e}_{k+1}^T \sum_{j=1}^{k} \mathbf{w}_j h_{j,k} = 0.$$

since $\mathbf{e}_{k+1}^T\mathbf{w}_j = \pm\mathbf{e}_{k+1}^T\mathbf{e}_j = 0$ for $j \le k$. ■

Golub and van Loan [6, p.347] write that *"The gist of the implicit Q theorem is that if $Q^TAQ = H$ and $Z^TAZ = G$ are both unreduced Hessenberg matrices and Q and Z have the same first column, then G and H are "essentially equal" in the sense that $G = DHD$ with $D = \operatorname{diag}(\pm1, \ldots, \pm1)$."*

We apply the Implicit Q Theorem in the following way: We want to compute the Hessenberg matrix $H_{k+1} = Z^TH_{k-1}Z$ where ZR is the QR factorization of $M = H_{k-1}^2 - sH_{k-1} + tI$. The Implicit Q Theorem now tells us that we essentially get H_{k+1} by any orthogonal similarity transformation $H_{k-1} \to Z_1^*H_{k-1}Z_1$ provided that $Z_1^*HZ_1$ is Hessenberg and $Z_1\mathbf{e}_1 = Z\mathbf{e}_1$.

Let P_0 be the Householder reflector with

$$P_0^T M\mathbf{e}_1 = P_0^T(H_{k-1}^2 - 2\mathrm{Re}(\sigma)H_{k-1} + |\sigma|^2 I)\,\mathbf{e}_1 = \alpha\,\mathbf{e}_1.$$

Since only the first three elements of the first column $M\mathbf{e}_1$ of M are nonzero, P_0 has the structure

$$P_0 = \begin{bmatrix} \times & \times & \times & & & \\ \times & \times & \times & & & \\ \times & \times & \times & & & \\ & & & 1 & & \\ & & & & \ddots & \\ & & & & & 1 \end{bmatrix}.$$

So,

$$H'_{k-1} := P_0^T H_{k-1} P_0 = \left[\begin{array}{ccc|cccc} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ + & \times & \times & \times & \times & \times & \times \\ \hline + & + & \times & \times & \times & \times & \times \\ & & & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{array}\right].$$

We now reduce $P_0^T H_{k-1} P_0$ similarly to Hessenberg form the same way as we did earlier, by a sequence of Householder reflectors $P_1, \ldots, P_{n-2}$. However, $P_0^T H_{k-1} P_0$ is a Hessenberg matrix up to the **bulge** at the top left. We take into account this structure when forming the $P_i = I - 2\mathbf{p}_i\mathbf{p}_i^T$. So, the structures of P_1 and of $P_1^T P_0^T H_{k-1} P_0 P_1$ are

$$P_1 = \begin{bmatrix} 1 & & & & & & \\ & \times & \times & \times & & & \\ & \times & \times & \times & & & \\ & \times & \times & \times & & & \\ & & & & 1 & & \\ & & & & & 1 & \\ & & & & & & 1 \end{bmatrix}, \quad H''_{k-1} = P_1^T H'_{k-1} P_1 = \left[\begin{array}{c|ccc|ccc} \times & \times & \times & \times & \times & \times & \times \\ \hline \times & \times & \times & \times & \times & \times & \times \\ 0 & \times & \times & \times & \times & \times & \times \\ 0 & + & \times & \times & \times & \times & \times \\ \hline & + & + & \times & \times & \times & \times \\ & & & & \times & \times & \times \\ & & & & & \times & \times \end{array}\right].$$

The transformation with P_1 has chased the bulge one position down the diagonal. The consecutive reflectors push it further by one position each until it falls out of the matrix at the end of the diagonal. Pictorially, we have

$$H'''_{k-1} = P_2^T H''_{k-1} P_2 = \left[\begin{array}{cc|ccc|cc} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ \hline & \times & \times & \times & \times & \times & \times \\ & 0 & \times & \times & \times & \times & \times \\ & 0 & + & \times & \times & \times & \times \\ \hline & & + & + & \times & \times & \times \\ & & & & & \times & \times \end{array}\right]$$

$$H''''_{k-1} = P_3^T H'''_{k-1} P_3 = \left[\begin{array}{ccc|ccc|c} \times & \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times & \times \\ \hline & & \times & \times & \times & \times & \times \\ & & 0 & \times & \times & \times & \times \\ & & 0 & + & \times & \times & \times \\ \hline & & & + & + & \times & \times \end{array}\right]$$

$$H_{k-1}^{'''''} = P_4^T H_{k-1}^{''''} P_4 = \left[\begin{array}{cccc|ccc}
\times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times \\
\hline
 & & & \times & \times & \times & \times \\
 & & & 0 & \times & \times & \times \\
 & & & 0 & + & \times & \times
\end{array}\right]$$

$$H_{k-1}^{''''''} = P_5^T H_{k-1}^{'''''} P_5 = \left[\begin{array}{ccccc|cc}
\times & \times & \times & \times & \times & \times & \times \\
\times & \times & \times & \times & \times & \times & \times \\
 & \times & \times & \times & \times & \times & \times \\
 & & \times & \times & \times & \times & \times \\
 & & & \times & \times & \times & \times \\
\hline
 & & & & \times & \times & \times \\
 & & & & 0 & \times & \times
\end{array}\right]$$

It is easy to see that the Householder vector $\mathbf{p}_i$, $i < n-2$, has only three nonzero elements at position $i+1, i+2, i+3$. Of $\mathbf{p}_{n-2}$ only the last two elements are nonzero. Clearly, $P_0 P_1 \cdots P_{n-2}\mathbf{e}_1 = P_0\mathbf{e}_1 = M\mathbf{e}_1/\alpha$.

Remark 4.3. Notice that in Algorithm 4.5 a double step is taken also if the eigenvalues of

$$G = \begin{bmatrix} h_{qq} & h_{qp} \\ h_{pq} & h_{pp} \end{bmatrix}$$

are real. As in the complex case we set $s = \text{trace}(G)$ and $t = \det(G)$. □

4.5.1 A numerical example

We consider a simple MATLAB implementation of the Algorithm 4.5 to compute the eigenvalues of the real matrix

$$A = \begin{bmatrix} 7 & 3 & 4 & -11 & -9 & -2 \\ -6 & 4 & -5 & 7 & 1 & 12 \\ -1 & -9 & 2 & 2 & 9 & 1 \\ -8 & 0 & -1 & 5 & 0 & 8 \\ -4 & 3 & -5 & 7 & 2 & 10 \\ 6 & 1 & 4 & -11 & -7 & -1 \end{bmatrix}$$

that has the spectrum

$$\sigma(A) = \{1 \pm 2i, 3, 4, 5 \pm 6i\}.$$

The intermediate output of the code was (after some editing) the following:

```
>> H=hess(A)

H(0) =

    7.0000    7.2761    5.8120   -0.1397    9.0152    7.9363
   12.3693    4.1307   18.9685   -1.2071   10.6833    2.4160
         0   -7.1603    2.4478   -0.5656   -4.1814   -3.2510
         0         0   -8.5988    2.9151   -3.4169    5.7230
         0         0         0    1.0464   -2.8351  -10.9792
         0         0         0         0    1.4143    5.3415
```

Algorithm 4.5 The Francis double step QR algorithm

```
Let $H_0 = H \in \mathbb{R}^{n\times n}$ be an upper Hessenberg matrix. This algorithm computes its real Schur form $H = UTU^T$ using the Francis double step QR algorithm. $T$ is a quasi upper triangular matrix.
$p := n$; /* $p$ indicates the ‘active’ matrix size. */
while $p > 2$ do
  $q := p - 1$;
  $s := H_{q,q} + H_{p,p}$;   $t := H_{q,q}H_{p,p} - H_{q,p}H_{p,q}$;
  /* compute first 3 elements of first column of $M$ */
  $x := H_{1,1}^2 + H_{1,2}H_{2,1} - sH_{1,1} + t$;
  $y := H_{2,1}(H_{1,1} + H_{2,2} - s)$;
  $z := H_{2,1}H_{3,2}$;
  for $k = 0$ to $p - 3$ do
    Determine the Householder reflector $P$ with $P^T\ [x; y; z]^T = \alpha\mathbf{e}_1$;
    $r := \max\{1, k\}$;
    $H_{k+1:k+3,r:n} := P^T H_{k+1:k+3,r:n}$;
    $r := \min\{k + 4, p\}$;
    $H_{1:r,k+1:k+3} := H_{1:r,k+1:k+3}P$;
    $x := H_{k+2,k+1}$;   $y := H_{k+3,k+1}$;
    if $k < p - 3$ then
      $z := H_{k+4,k+1}$;
    end if
  end for
  Determine the Givens rotation $P$ with $P^T\ [x; y]^T = \alpha\mathbf{e}_1$;
  $H_{q:p,p-2:n} := P^T H_{q:p,p-2:n}$;
  $H_{1:p,p-1:p} := H_{1:p,p-1:p}P$;
  /* check for convergence */
  if $|H_{p,q}| < \varepsilon\ (|H_{q,q}| + |H_{p,p}|)$ then
    $H_{p,q} := 0$;   $p := p - 1$;   $q := p - 1$;
  else if $|H_{p-1,q-1}| < \varepsilon\ (|H_{q-1,q-1}| + |H_{q,q}|)$ then
    $H_{p-1,q-1} := 0$;   $p := p - 2$;   $q := p - 1$;
  end if
end while
```

```
>> PR=qr2st(H)

[it_step, p = n_true, H(p,p-1), H(p-1,p-2)]

     1     6  -1.7735e-01  -1.2807e+00
     2     6  -5.9078e-02  -1.7881e+00
     3     6  -1.6115e-04  -5.2705e+00
     4     6  -1.1358e-07  -2.5814e+00
     5     6   1.8696e-14   1.0336e+01
     6     6  -7.1182e-23  -1.6322e-01


H(6) =

    5.0000    6.0000    2.3618    5.1837  -13.4434   -2.1391
   -6.0000    5.0000    2.9918   10.0456   -8.7743  -21.0094
    0.0000   -0.0001   -0.9393    3.6939   11.7357    3.8970
    0.0000   -0.0000   -1.9412    3.0516    2.9596  -10.2714
         0    0.0000    0.0000   -0.1632    3.8876    4.1329
         0         0         0    0.0000   -0.0000    3.0000


    7      5    1.7264e-02   -7.5016e-01
    8      5    2.9578e-05   -8.0144e-01
    9      5    5.0602e-11   -4.6559e+00
   10      5   -1.3924e-20   -3.1230e+00


H(10) =

    5.0000    6.0000   -2.7603    1.3247   11.5569   -2.0920
   -6.0000    5.0000  -10.7194    0.8314   11.8952   21.0142
   -0.0000   -0.0000    3.5582    3.3765    5.9254   -8.5636
   -0.0000   -0.0000   -3.1230   -1.5582  -10.0935   -6.3406
         0         0         0    0.0000    4.0000    4.9224
         0         0         0    0.0000         0    3.0000


   11      4    1.0188e+00   -9.1705e-16


H(11) =

    5.0000    6.0000  -10.2530    4.2738  -14.9394  -19.2742
   -6.0000    5.0000   -0.1954    1.2426    7.2023   -8.6299
   -0.0000   -0.0000    2.2584   -5.4807  -10.0623    4.4380
    0.0000   -0.0000    1.0188   -0.2584   -5.9782   -9.6872
         0         0         0         0    4.0000    4.9224
         0         0         0    0.0000         0    3.0000
```

4.5.2 The complexity

We first estimate the complexity of a single step of the double step Hessenberg QR algorithm. The most expensive operations are the applications of the 3 × 3 Householder reflectors in steps 13 and 15 of Algorithm 4.5. Let us first count the flops for applying the Householder reflector to a 3-vector,

$$\mathbf{x} := (I - 2\mathbf{u}\mathbf{u}^T)\mathbf{x} = \mathbf{x} - \mathbf{u}(2\mathbf{u}^T\mathbf{x}).$$

The inner product $\mathbf{u}^T\mathbf{x}$ costs 5 flops, multiplying with 2 another one. The operation $\mathbf{x} := \mathbf{x} - \mathbf{u}\gamma$, $\gamma = 2\mathbf{u}^T\mathbf{x}$, cost 6 flops, altogether 12 flops.

In the k-th step of the loop there are $n-k$ of these application from the left in step 13 and $k+4$ from the right in step 15. In this step there are thus about $12n+\mathcal{O}(1)$ flops to be executed. As k is running from 1 to $p-3$. We have about $12pn$ flops for this step. Since p runs from n down to about 2 we have $6n^3$ flops. If we assume that two steps are required per eigenvalue the flop count for Francis' double step QR algorithm to compute *all* eigenvalues of a *real* Hessenberg matrix is $12n^3$. If also the eigenvector matrix is accumulated the two additional statements have to be inserted into Algorithm 4.5. After step 15 we have

```
$Q_{1:n,k+1:k+3} := Q_{1:n,k+1:k+3}P$;
```

and after step 23 we introduce

```
1: $Q_{1:n,p-1:p} := Q_{1:n,p-1:p}P$;
```

which costs another $12n^3$ flops.

We earlier gave the estimate of $6n^3$ flops for a Hessenberg QR step, see Algorithm 4.2. If the latter has to be spent in *complex* arithmetic then the single shift Hessenberg QR algorithm is more expensive than the double shift Hessenberg QR algorithm that is executed in real arithmetic.

Remember that the reduction to Hessenberg form costs $\frac{10}{3}n^3$ flops without forming the transformation matrix and $\frac{14}{3}n^3$ if this matrix is formed.

4.6 The symmetric tridiagonal QR algorithm

The QR algorithm can be applied straight to Hermitian or symmetric matrices. By (4.1) we see that the QR algorithm generates a sequence $\{A_k\}$ of symmetric matrices. Taking into account the symmetry, the performance of the algorithm can be improved considerably. Furthermore, from Theorem 2.14 we know that Hermitian matrices have a real spectrum. Therefore, we can restrict ourselves to single shifts.

4.6.1 Reduction to tridiagonal form

The reduction of a full Hermitian matrix to Hessenberg form produces a Hermitian Hessenberg matrix, which (up to rounding errors) is a real symmetric tridiagonal matrix. Let us consider how to take into account symmetry. To that end let us consider the first reduction step that introduces $n-2$ zeros into the first column (and the first row) of $A = A^* \in \mathbb{C}^{n\times n}$. Let

$$P_1 = \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & I_{n-1} - 2\mathbf{u}_1\mathbf{u}_1^* \end{bmatrix}, \qquad \mathbf{u}_1 \in \mathbb{C}^n, \quad \|\mathbf{u}_1\| = 1.$$

Then,

$$\begin{aligned} A_1 := P_1^* A P_1 &= (I - 2\mathbf{u}_1\mathbf{u}_1^*)A(I - 2\mathbf{u}_1\mathbf{u}_1^*) \\ &= A - \mathbf{u}_1\underbrace{(2\mathbf{u}_1^* A - 2(\mathbf{u}_1^* A\mathbf{u}_1)\mathbf{u}_1^*)}_{\mathbf{v}_1^*} - \underbrace{(2A\mathbf{u}_1 - 2\mathbf{u}_1(\mathbf{u}_1^* A\mathbf{u}_1))}_{\mathbf{v}_1}\mathbf{u}_1^* \\ &= A - \mathbf{u}_1\mathbf{v}_1^* - \mathbf{v}_1\mathbf{u}_1^*. \end{aligned}$$

In the k-th step of the reduction we similarly have

$$A_k = P_k^* A_{k-1} P_k = A_{k-1} - \mathbf{u}_{k-1}\mathbf{v}_{k-1}^* - \mathbf{v}_{k-1}\mathbf{u}_{k-1}^*,$$

where the last $n-k$ elements of $\mathbf{u}_{k-1}$ and $\mathbf{v}_{k-1}$ are nonzero. Forming

$$\mathbf{v}_{k-1} = 2A_{k-1}\mathbf{u}_{k-1} - 2\mathbf{u}_{k-1}(\mathbf{u}_{k-1}^* A_{k-1}\mathbf{u}_{k-1})$$

costs $2(n-k)^2 + \mathcal{O}(n-k)$ flops. This complexity results from $A_{k-1}\mathbf{u}_{k-1}$. The rank-2 update of A_{k-1},

$$A_k = A_{k-1} - \mathbf{u}_{k-1}\mathbf{v}_{k-1}^* - \mathbf{v}_{k-1}\mathbf{u}_{k-1}^*,$$

requires another $2(n-k)^2 + \mathcal{O}(n-k)$ flops, taking into account symmetry. By consequence, the transformation to tridiagonal form can be accomplished in

$$\sum_{k=1}^{n-1} \big(4(n-k)^2 + \mathcal{O}(n-k)\big) = \frac{4}{3}n^3 + \mathcal{O}(n^2)$$

floating point operations.

4.6.2 The tridiagonal QR algorithm

In the symmetric case the Hessenberg QR algorithm becomes a tridiagonal QR algorithm. This can be executed in an **explicit** or an **implicit** way. In the explicit form, a QR step is essentially

```
1: Choose a shift $\mu$
2: Compute the QR factorization $A - \mu I = QR$
3: Update $A$ by $A = RQ + \mu I$.
```

Of course, this is done by means of plane rotations and by respecting the symmetric tridiagonal structure of A.

In the more elegant implicit form of the algorithm we first compute the first Givens rotation $G_0 = G(1, 2, \vartheta)$ of the QR factorization that zeros the $(2, 1)$ element of $A - \mu I$,

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix} \begin{bmatrix} a_{11} - \mu \\ a_{21} \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}, \qquad c = \cos(\vartheta_0), \quad s = \sin(\vartheta_0). \tag{4.12}$$

Performing a similary transformation with G_0 we have $(n = 5)$

$$G_0^* A G_0 = A' = \begin{bmatrix} \times & \times & + & & \\ \times & \times & \times & & \\ + & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

Similar as with the double step Hessenberg QR algorithm we chase the bulge down the diagonal. In the 5×5 example this becomes

$$A \xrightarrow[= G(1,2,\vartheta_0)]{G_0} \begin{bmatrix} \times & \times & + & & \\ \times & \times & \times & & \\ + & \times & \times & \times & \\ & & \times & \times & \times \\ & & & \times & \times \end{bmatrix} \xrightarrow[= G(2,3,\vartheta_1)]{G_1} \begin{bmatrix} \times & \times & 0 & & \\ \times & \times & \times & + & \\ 0 & \times & \times & \times & \\ & + & \times & \times & \times \\ & & & \times & \times \end{bmatrix}$$

$$\xrightarrow[= G(3,4,\vartheta_2)]{G_2} \begin{bmatrix} \times & \times & 0 & & \\ \times & \times & \times & & \\ & \times & \times & \times & + \\ & 0 & \times & \times & \times \\ & & + & \times & \times \end{bmatrix} \xrightarrow[= G(4,5,\vartheta_3)]{G_3} \begin{bmatrix} \times & \times & & & \\ \times & \times & \times & & \\ & \times & \times & \times & 0 \\ & & \times & \times & \times \\ & & 0 & \times & \times \end{bmatrix} = \overline{A}.$$

The full step is given by

$$\overline{A} = Q^* A Q, \qquad Q = G_0 \, G_1 \, \cdots \, G_{n-2}.$$

Because $G_k \mathbf{e}_1 = \mathbf{e}_1$ for $k > 0$ we have

$$Q \, \mathbf{e}_1 = G_0 \, G_1 \cdots G_{n-2} \, \mathbf{e}_1 = G_0 \, \mathbf{e}_1.$$

Both explicit and implicit QR step form the same first plane rotation G_0. By referring to the Implicit Q Theorem 4.5 we see that explicit and implicit QR step compute *essentially* the same $\overline{A}$.

Algorithm 4.6 Symmetric tridiagonal QR algorithm with implicit Wilkinson shift

```
Let $T \in \mathbb{R}^{n\times n}$ be a symmetric tridiagonal matrix with diagonal entries $a_1, \ldots, a_n$ and off-diagonal entries $b_2, \ldots, b_n$.
This algorithm computes the eigenvalues $\lambda_1, \ldots, \lambda_n$ of $T$ and corresponding eigenvectors $\mathbf{q}_1, \ldots, \mathbf{q}_n$. The eigenvalues are stored in $a_1, \ldots, a_n$. The eigenvectors are stored in the matrix $Q$, such that $TQ = Q\ \mathrm{diag}(a_1, \ldots, a_n)$.
$m = n$ /* Actual problem dimension. $m$ is reduced in the convergence check. */
while $m > 1$ do
  $d := (a_{m-1} - a_m)/2$; /* Compute Wilkinson's shift */
  if $d = 0$ then
    $s := a_m - |b_m|$;
  else
    $s := a_m - b_m^2/(d + \mathrm{sign}(d)\sqrt{d^2 + b_m^2})$;
  end if
  $x := a(1) - s$; /* Implicit QR step begins here */
  $y := b(2)$;
  for $k = 1$ to $m - 1$ do
    if $m > 2$ then
      $[c, s] :=$ givens$(x, y)$;
    else
      Determine $[c, s]$ such that $\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} a_1 & b_2 \\ b_2 & a_2 \end{bmatrix} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$ is diagonal
    end if
    $w := cx - sy$;
    $d := a_k - a_{k+1}$;    $z := (2cb_{k+1} + ds)s$;
    $a_k := a_k - z$;    $a_{k+1} := a_{k+1} + z$;
    $b_{k+1} := dcs + (c^2 - s^2)b_{k+1}$;
    $x := b_{k+1}$;
    if $k > 1$ then
      $b_k := w$;
    end if
    if $k < m - 1$ then
      $y := -sb_{k+2}$;    $b_{k+2} := cb_{k+2}$;
    end if
    $Q_{1:n;k:k+1} := Q_{1:n;k:k+1} \begin{bmatrix} c & s \\ -s & c \end{bmatrix}$;
  end for/* Implicit QR step ends here */
  if $|b_m| < \varepsilon(|a_{m-1}| + |a_m|)$ then /* Check for convergence */
    $m := m - 1$;
  end if
end while
```

Algorithm 4.6 shows the implicit symmetric tridiagonal QR algorithm. The shifts are chosen acording to Wilkinson. An issue not treated in this algorithm is **deflation**. Deflation is of big practical importance. Let us consider the following 6 × 6 situation

$$T = \begin{bmatrix} a_1 & b_2 & & & & \\ b_2 & a_2 & b_3 & & & \\ & b_3 & a_3 & 0 & & \\ & & 0 & a_4 & b_5 & \\ & & & b_5 & a_5 & b_6 \\ & & & & b_6 & a_6 \end{bmatrix}.$$

The shift for the next step is determined from elements a_5, a_6, and b_6. According to (4.12) the first plane rotation is determined from the shift and the elements a_1 and b_1. The implicit shift algorithm then chases the bulge down the diagonal. In this particular situation, the procedure finishes already in row/column 4 because $b_4 = 0$. Thus the shift which is an approximation to an eigenvalue of the second block (rows 4 to 6) is applied to the wrong first block (rows 1 to 3). Clearly, this shift does not improve convergence.

If the QR algorithm is applied in its explicit form, then still the first block is not treated properly, i.e. with a (probably) wrong shift, but at least the second block is diagonalized rapidly.

Deflation is done as indicated in Algorithm 4.6:

$$\textbf{if} \quad |b_k| < \varepsilon(|a_{k-1}| + |a_k|) \quad \textbf{then} \quad \text{deflate.}$$

Deflation is particularly simple in the symetric case since it just means that a tridiagonal eigenvalue problem decouples in two (or more) smaller tridiagonal eigenvalue problems. Notice, however, that the eigenvectors are still n elements long.

4.7 Research

Still today the QR algorithm computes the Schur form of a matrix and is by far the most popular approach for solving dense nonsymmetric eigenvalue problems. Multishift and aggressive early deflation techniques have led to significantly more efficient sequential implementations of the QR algorithm during the last decade. For a brief survey and a discussion of the parallelization of the QR algorithm, see [7].

The three steps of the presented symmetric QR algorithm are (1) reducion of the original matrix to tridiagonal form, (2) computation of the eigenpairs of the tridiagonal matrix, and (3) back-transformation of the eigenvectors. In the ELPA project the first step has been successfully replaced by a two-stage procedure: transformation full to banded, and banded to tridiagonal. This approach improves the utilization of memory hierarchies [8, 3].

4.8 Summary

The QR algorithm is a very powerful algorithm to stably compute the eigenvalues and (if needed) the corresponding eigenvectors or Schur vectors. All steps of the algorithm cost $\mathcal{O}(n^3)$ floating point operations, see Table 4.1. The one exception is the case where only eigenvalues are desired of a symmetric tridiagonal matrix. The linear algebra software package LAPACK [1] contains subroutines for all possible ways the QR algorithm may be employed.

	nonsymmetric case		symmetric case	
	without	with	without	with
	Schurvectors		eigenvectors	
transformation to Hessenberg/tridiagonal form	$\frac{10}{3}n^3$	$\frac{14}{3}n^3$	$\frac{4}{3}n^3$	$\frac{8}{3}n^3$
real double step Hessenberg/tridiagonal QR algorithm (2 steps per eigenvalues assumed)	$\frac{20}{3}n^3$	$\frac{50}{3}n^3$	$24n^2$	$6n^3$
total	$10n^3$	$25n^3$	$\frac{4}{3}n^3$	$9n^3$

Table 4.1: Complexity in flops to compute eigenvalues and eigenvectors/Schur vectors of a real $n \times n$ matrix

We finish by repeating, that the QR algorithm is a method for *dense* matrix problems. The reduction of a sparse matrix to tridiagonal or Hessenberg form produces **fill-in**, thus destroying the sparsity structure which one almost always tries to preserve.

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide – Release 2.0*, SIAM, Philadelphia, PA, 1994. (Software and guide are available from Netlib at URL http://www.netlib.org/lapack/).

[2] P. Arbenz and G. H. Golub, *Matrix shapes invariant under the symmetric QR algorithm*, Numer. Linear Algebra Appl., 2 (1995), pp. 87–93.

[3] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, and P. Willems, *Developing algorithms and software for the parallel solution of the symmetric eigenvalue problem*, J. Comput. Sci., 2 (2011), pp. 272–278.

[4] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[5] J. G. F. Francis, *The QR transformation – Parts 1 and 2*, Comput. J., 4 (1961-1962), pp. 265–271 and 332–345.

[6] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[7] B. Kågström, D. Kressner, and M. Shao, *On aggressive early deflation in parallel variants of the QR algorithm*, in Applied Parallel and Scientific Computing (PARA 2010), K. Jónasson, ed., Heidelberg, 2012, Springer, pp. 1–10. (Lecture Notes in Computer Science, 7133).

[8] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, *The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science*, J. Phys.: Condens. Matter, 26 (2014), p. 213201.

[9] B. N. Parlett, *The QR algorithm*, Computing Sci. Eng., 2 (2000), pp. 38–42.

[10] H. Rutishauser, *Solution of eigenvalue problems with the LR-transformation*, NBS Appl. Math. Series, 49 (1958), pp. 47–81.

[11] J. H. Wilkinson, *The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford, 1965.

Chapter 5

Cuppen's Divide and Conquer Algorithm

In this chapter we deal with an algorithm that is designed for the efficient solution of the symmetric tridiagonal eigenvalue problem

$$
T\mathbf{x} = \lambda\mathbf{x}, \qquad T = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & a_2 & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix}. \tag{5.1}
$$

We noticed from Table 4.1 that the reduction of a full symmetric matrix to a similar tridiagonal matrix requires about $\frac{8}{3}n^3$ while the tridiagonal QR algorithm needs an estimated $6n^3$ floating operations (flops) to converge. Because of the importance of this subproblem a considerable effort has been put into finding faster algorithms than the QR algorithms to solve the tridiagonal eigenvalue problem. In the mid-1980's Dongarra and Sorensen [4] promoted an algorithm originally proposed by Cuppen [2]. This algorithm was based on a divide and conquer strategy. However, it took ten more years until a stable variant was found by Gu and Eisenstat [5, 6]. Today, a stable implementation of this latter algorithm is available in LAPACK [1].

5.1 The divide and conquer idea

Divide and conquer is an old strategy in military to defeat an enemy going back at least to Caesar. In computer science, divide and conquer (D&C) is an important algorithm design paradigm. It works by recursively breaking down a problem into two or more subproblems of the same (or related) type, until these become simple enough to be solved directly. The solutions to the subproblems are then combined to give a solution to the original problem. Translated to our problem the strategy becomes

1. Partition the tridiagonal eigenvalue problem into two (or more) smaller tridiagonal eigenvalue problems.

2. Solve the two smaller problems.

3. Combine the solutions of the smaller problems to get the desired solution of the overall problem.

Evidently, this strategy can be applied recursively.

5.2 Partitioning the tridiagonal matrix

Partitioning the *irreducible* tridiagonal matrix is done in the following way. We write

$$
\begin{aligned}
T &= \left[\begin{array}{cccc|cccc}
a_1 & b_1 & & & & & & \\
b_1 & a_2 & \ddots & & & & & \\
 & \ddots & \ddots & b_{m-1} & & & & \\
 & & b_{m-1} & a_m & b_m & & & \\
\hline
 & & & b_m & a_{m+1} & b_{m+1} & & \\
 & & & & b_{m+1} & a_{m+2} & \ddots & \\
 & & & & & \ddots & \ddots & b_{n-1} \\
 & & & & & & b_{n-1} & a_n
\end{array}\right] \\
&= \left[\begin{array}{cccc|cccc}
a_1 & b_1 & & & & & & \\
b_1 & a_2 & \ddots & & & & & \\
 & \ddots & \ddots & b_{m-1} & & & & \\
 & & b_{m-1} & a_m \mp b_m & & & & \\
\hline
 & & & & a_{m+1} \mp b_m & b_{m+1} & & \\
 & & & & b_{m+1} & a_{m+2} & \ddots & \\
 & & & & & \ddots & \ddots & b_{n-1} \\
 & & & & & & b_{n-1} & a_n
\end{array}\right]
+ \left[\begin{array}{cc|cc}
 & & & \\
 & \pm b_m & b_m & \\
\hline
 & b_m & \pm b_m & \\
 & & &
\end{array}\right] \\
&= \left[\begin{array}{c|c}
T_1 & \\
\hline
 & T_2
\end{array}\right] + \rho \mathbf{u}\mathbf{u}^T \qquad \text{with } \mathbf{u} = \left[\begin{array}{c} \pm\mathbf{e}_m \\ \hline \mathbf{e}_1 \end{array}\right] \text{ and } \rho = \pm b_m,
\end{aligned}
\tag{5.2}
$$

where $\mathbf{e}_m$ is a vector of length $m \approx \frac{n}{2}$ and $\mathbf{e}_1$ is a vector of length $n-m$. Notice that the most straightforward way to partition the problem without modifying the diagonal elements leads to a rank-two modification. With the approach of (5.2) we have the original T as a sum of two smaller tridiagonal systems plus a *rank-one modification*.

5.3 Solving the small systems

We solve the half-sized eigenvalue problems,

$$T_i = Q_i \Lambda_i Q_i^T, \qquad Q_i^T Q_i = I, \qquad i = 1, 2. \tag{5.3}$$

These two spectral decompositions can be computed by any algorithm, in particular also by this divide and conquer algorithm by which the T_i would be further split. It is clear that by this partitioning an large number of small problems can be generated that can be potentially solved in parallel. For a parallel algorithm, however, the further phases of the algorithm must be parallelizable as well.

Plugging (5.3) into (5.2) gives

$$\left[\begin{array}{c|c} Q_1^T & \\ \hline & Q_2^T \end{array}\right] \left(\left[\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right] + \rho \mathbf{u}\mathbf{u}^T \right) \left[\begin{array}{c|c} Q_1 & \\ \hline & Q_2 \end{array}\right] = \left[\begin{array}{c|c} \Lambda_1 & \\ \hline & \Lambda_2 \end{array}\right] + \rho \mathbf{v}\mathbf{v}^T \tag{5.4}$$

with

$$\mathbf{v} = \left[\begin{array}{c|c} Q_1^T & \\ & Q_2^T \end{array}\right] \mathbf{u} = \left[\begin{array}{c} \pm Q_1^T \mathbf{e}_m \\ Q_2^T \mathbf{e}_1 \end{array}\right] = \left[\begin{array}{c} \pm \text{last row of } Q_1 \\ \text{first row of } Q_2 \end{array}\right]. \tag{5.5}$$

Now we have arrived at the eigenvalue problem

$$(D + \rho \mathbf{v}\mathbf{v}^T)\mathbf{x} = \lambda \mathbf{x}, \qquad D = \Lambda_1 \oplus \Lambda_2 = \operatorname{diag}(\lambda_1, \ldots, \lambda_n). \tag{5.6}$$

That is, we have to compute the spectral decomposition of a matrix that is a **diagonal plus a rank-one update**. Let

$$D + \rho \mathbf{v}\mathbf{v}^T = Q\Lambda Q^T \tag{5.7}$$

be this spectral decomposition. Then, the spectral decomposition of the tridiagonal T is

$$T = \left[\begin{array}{c|c} Q_1 & \\ & Q_2 \end{array}\right] Q\Lambda Q^T \left[\begin{array}{c|c} Q_1^T & \\ & Q_2^T \end{array}\right]. \tag{5.8}$$

Forming the product $(Q_1 \oplus Q_2)Q$ will turn out to be *the most expensive step of the algorithm.* It costs $n^3 + \mathcal{O}(n^2)$ floating point operations

5.4 Deflation

There are certain solutions of (5.7) that can be given immediately, by just looking carefully at the equation.

If there are zero entries in $\mathbf{v}$ then we have

$$\big(v_i = 0 \Leftrightarrow \mathbf{v}^T\mathbf{e}_i = 0\big) \quad \Longrightarrow \quad (D + \rho \mathbf{v}\mathbf{v}^T)\mathbf{e}_i = d_i \mathbf{e}_i. \tag{5.9}$$

Thus, if an entry of $\mathbf{v}$ vanishes we can read the eigenvalue from the diagonal of D at once and the corresponding eigenvector is a coordinate vector.

If identical entries occur in the diagonal of D, say $d_i = d_j$, with $i < j$, then we can find a plane rotation $G(i, j, \phi)$ (see (4.4)) such that it introduces a zero into the j-th position of $\mathbf{v}$,

$$G^T\mathbf{v} = G(i,j,\varphi)^T\mathbf{v} = \begin{array}{c c} \begin{bmatrix} \times \\ \vdots \\ \sqrt{{v_i}^2 + {v_j}^2} \\ \vdots \\ 0 \\ \vdots \\ \times \end{bmatrix} & \begin{array}{c} \\ \\ \leftarrow i \\ \\ \leftarrow j \\ \\ \\ \end{array} \end{array}$$

Notice, that (for *any* φ),

$$G(i,j,\varphi)^T D G(i,j,\varphi) = D, \qquad d_i = d_j.$$

So, if there are multiple eigenvalues in D we can reduce all but one of them by introducing zeros in $\mathbf{v}$ and then proceed as previously in (5.9).

When working with floating point numbers we deflate if

$$|v_i| < C\varepsilon \|T\| \qquad \text{or} \qquad |d_i - d_j| < C\varepsilon\|T\|, \qquad (\|T\| = \|D + \rho \mathbf{v}\mathbf{v}^T\|) \tag{5.10}$$

where C is a small constant. Deflation changes the eigenvalue problem for $D + \rho\mathbf{v}\mathbf{v}^T$ into the eigenvalue problem for

$$\begin{bmatrix} D_1 + \rho \mathbf{v}_1\mathbf{v}_1^T & O \\ O & D_2 \end{bmatrix} = G^T(D + \rho\mathbf{v}\mathbf{v}^T)G + E, \qquad \|E\| < C\varepsilon\sqrt{\|D\|^2 + |\rho|^2\|\mathbf{v}\|^4}, \tag{5.11}$$

where D_1 has no multiple diagonal entries and $\mathbf{v}_1$ has no zero entries. So, we have to compute the spectral decomposition of the matrix in (5.11) which is similar to a slight perturbation of the original matrix. G is the product of Givens rotations.

5.4.1 Numerical examples

Let us first consider

$$
T = \left[\begin{array}{ccc|ccc} 1 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & 3 & 1 & & \\ \hline & & 1 & 4 & 1 & \\ & & & 1 & 5 & 1 \\ & & & & 1 & 6 \end{array}\right] = \left[\begin{array}{ccc|ccc} 1 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & 2 & 0 & & \\ \hline & & 0 & 3 & 1 & \\ & & & 1 & 5 & 1 \\ & & & & 1 & 6 \end{array}\right] + \left[\begin{array}{ccc|ccc} 0 & & & & & \\ & 0 & & & & \\ & & 1 & 1 & & \\ \hline & & 1 & 1 & & \\ & & & & 0 & \\ & & & & & 0 \end{array}\right]
$$

$$
= \left[\begin{array}{ccc|ccc} 1 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & 2 & & & \\ \hline & & & 3 & 1 & \\ & & & 1 & 5 & 1 \\ & & & & 1 & 6 \end{array}\right] + \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \end{array}\right] \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \end{array}\right]^T = T_0 + \mathbf{u}\mathbf{u}^T.
$$

Then a little MATLAB experiment shows that

$$
Q_0^T T Q_0 = \begin{bmatrix} 0.1981 & & & & & \\ & 1.5550 & & & & \\ & & 3.2470 & & & \\ & & & 2.5395 & & \\ & & & & 4.7609 & \\ & & & & & 6.6996 \end{bmatrix} + \begin{bmatrix} 0.3280 \\ 0.7370 \\ 0.5910 \\ 0.9018 \\ -0.4042 \\ 0.1531 \end{bmatrix} \begin{bmatrix} 0.3280 \\ 0.7370 \\ 0.5910 \\ 0.9018 \\ -0.4042 \\ 0.1531 \end{bmatrix}^T
$$

with

$$
Q_0 = \begin{bmatrix} 0.7370 & -0.5910 & 0.3280 & & & \\ -0.5910 & -0.3280 & 0.7370 & & & \\ 0.3280 & 0.7370 & 0.5910 & & & \\ & & & 0.9018 & -0.4153 & 0.1200 \\ & & & -0.4042 & -0.7118 & 0.5744 \\ & & & 0.1531 & 0.5665 & 0.8097 \end{bmatrix}
$$

Here it is not possible to deflate.

Let us now look at an example with more symmetry,

$$
T = \left[\begin{array}{ccc|ccc} 2 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & 2 & 1 & & \\ \hline & & 1 & 2 & 1 & \\ & & & 1 & 2 & 1 \\ & & & & 1 & 2 \end{array}\right] = \left[\begin{array}{ccc|ccc} 2 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & 1 & 0 & & \\ \hline & & 0 & 1 & 1 & \\ & & & 1 & 2 & 1 \\ & & & & 1 & 2 \end{array}\right] + \left[\begin{array}{ccc|ccc} 0 & & & & & \\ & 0 & & & & \\ & & 1 & 1 & & \\ \hline & & 1 & 1 & & \\ & & & & 0 & \\ & & & & & 0 \end{array}\right]
$$

$$
= \left[\begin{array}{ccc|ccc} 2 & 1 & & & & \\ 1 & 2 & 1 & & & \\ & 1 & 1 & & & \\ \hline & & & 1 & 1 & \\ & & & 1 & 2 & 1 \\ & & & & 1 & 2 \end{array}\right] + \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \end{array}\right] \left[\begin{array}{c} 0 \\ 0 \\ 1 \\ \hline 1 \\ 0 \\ 0 \end{array}\right]^T = T_0 + \mathbf{u}\mathbf{u}^T.
$$

Now, MATLAB gives

$$Q_0^T T Q_0 = \begin{bmatrix} 0.1981 &&&&& \\ & 1.5550 &&&& \\ && 3.2470 &&& \\ &&& 0.1981 && \\ &&&& 1.5550 & \\ &&&&& 3.2470 \end{bmatrix} + \begin{bmatrix} 0.7370 \\ -0.5910 \\ 0.3280 \\ 0.7370 \\ -0.5910 \\ 0.3280 \end{bmatrix} \begin{bmatrix} 0.7370 \\ -0.5910 \\ 0.3280 \\ 0.7370 \\ -0.5910 \\ 0.3280 \end{bmatrix}^T$$

with

$$Q_0 = \begin{bmatrix} 0.3280 & 0.7370 & 0.5910 &&& \\ -0.5910 & -0.3280 & 0.7370 &&& \\ 0.7370 & -0.5910 & 0.3280 &&& \\ &&& 0.7370 & -0.5910 & 0.3280 \\ &&& -0.5910 & -0.3280 & 0.7370 \\ &&& 0.3280 & 0.7370 & 0.5910 \end{bmatrix}$$

In this example we have *three* double eigenvalues. Because the corresponding components of $\mathbf{v}$ (v_i and v_{i+1}) are equal we define

$$\begin{aligned} G &= G(1,4,\pi/4)G(2,5,\pi/4)G(3,6,\pi/4) \\ &= \left[\begin{array}{ccc|ccc} 0.7071 &&& 0.7071 && \\ & 0.7071 &&& 0.7071 & \\ && 0.7071 &&& 0.7071 \\ \hline -0.7071 &&& 0.7071 && \\ & -0.7071 &&& 0.7071 & \\ && -0.7071 &&& 0.7071 \end{array}\right]. \end{aligned}$$

Then,

$$\begin{aligned} G^T Q_0^T T Q_0 G &= G^T Q_0^T T_0 Q_0 G + G^T\mathbf{v}(G^T\mathbf{v})^T = D + G^T\mathbf{v}(G^T\mathbf{v})^T \\ &= \begin{bmatrix} 0.1981 &&&&& \\ & 1.5550 &&&& \\ && 3.2470 &&& \\ &&& 0.1981 && \\ &&&& 1.5550 & \\ &&&&& 3.2470 \end{bmatrix} + \begin{bmatrix} 1.0422 \\ -0.8358 \\ 0.4638 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix} \begin{bmatrix} 1.0422 \\ -0.8358 \\ 0.4638 \\ 0.0000 \\ 0.0000 \\ 0.0000 \end{bmatrix}^T \end{aligned}$$

Therefore, (in this example) $\mathbf{e}_4$, $\mathbf{e}_5$, and $\mathbf{e}_6$ are eigenvectors of

$$D + G^T\mathbf{v}(G^T\mathbf{v})^T = D + G^T\mathbf{v}\mathbf{v}^T G$$

corresponding to the eigenvalues d_4, d_5, and d_6, respectively. The eigenvectors of T corresponding to these three eigenvalues are the last three columns of

$$Q_0 G = \begin{bmatrix} 0.2319 & -0.4179 & 0.5211 & 0.5211 & -0.4179 & 0.2319 \\ 0.5211 & -0.2319 & -0.4179 & -0.4179 & -0.2319 & 0.5211 \\ 0.4179 & 0.5211 & 0.2319 & 0.2319 & 0.5211 & 0.4179 \\ -0.2319 & 0.4179 & -0.5211 & 0.5211 & -0.4179 & 0.2319 \\ -0.5211 & 0.2319 & 0.4179 & -0.4179 & -0.2319 & 0.5211 \\ -0.4179 & -0.5211 & -0.2319 & 0.2319 & 0.5211 & 0.4179 \end{bmatrix}.$$

5.5 The eigenvalue problem for $D + \rho\mathbf{v}\mathbf{v}^T$

We know that $\rho \neq 0$. Otherwise there is nothing to be done. Furthermore, after deflation, we know that all elements of $\mathbf{v}$ are nonzero and that the diagonal elements of D are all distinct, in fact,

$$|d_i - d_j| > C\varepsilon\|T\|.$$

We order the diagonal elements of D such that

$$d_1 < d_2 < \cdots < d_n.$$

Notice that this procedure permutes the elements of $\mathbf{v}$ as well. Let $(\lambda, \mathbf{x})$ be an eigenpair of

$$(D + \rho\mathbf{v}\mathbf{v}^T)\mathbf{x} = \lambda\mathbf{x}. \tag{5.12}$$

Then,

$$(D - \lambda I)\mathbf{x} = -\rho\mathbf{v}\mathbf{v}^T\mathbf{x}. \tag{5.13}$$

λ cannot be equal to one of the d_i. If $\lambda = d_k$ then the k-th element on the left of (5.13) vanishes. But then either $v_k = 0$ or $\mathbf{v}^T\mathbf{x} = 0$. The first cannot be true for our assumption about $\mathbf{v}$. If on the other hand $\mathbf{v}^T\mathbf{x} = 0$ then $(D - d_k I)\mathbf{x} = \mathbf{0}$. Thus $\mathbf{x} = \mathbf{e}_k$ and $\mathbf{v}^T\mathbf{e}_k = v_k = 0$, which cannot be true. Therefore $D - \lambda I$ is nonsingular and

$$\mathbf{x} = \rho(\lambda I - D)^{-1}\mathbf{v}(\mathbf{v}^T\mathbf{x}). \tag{5.14}$$

This equation shows that $\mathbf{x}$ is proportional to $(\lambda I - D)^{-1}\mathbf{v}$. If we require $\|\mathbf{x}\| = 1$ then

$$\mathbf{x} = \frac{(\lambda I - D)^{-1}\mathbf{v}}{\|(\lambda I - D)^{-1}\mathbf{v}\|}. \tag{5.15}$$

Multiplying (5.14) by $\mathbf{v}^T$ from the left we get

$$\mathbf{v}^T\mathbf{x} = \rho\mathbf{v}^T(\lambda I - D)^{-1}\mathbf{v}(\mathbf{v}^T\mathbf{x}). \tag{5.16}$$

Since $\mathbf{v}^T\mathbf{x} \neq 0$, λ is an eigenvalue of (5.12) if and only if

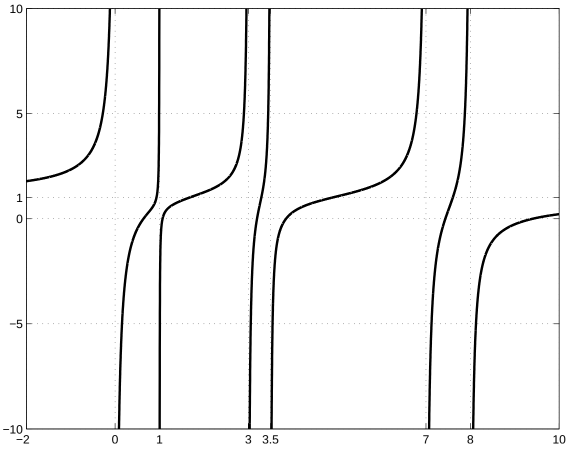


Figure 5.1: Graph of $1 + \frac{1}{0-\lambda} + \frac{0.2^2}{1-\lambda} + \frac{0.6^2}{3-\lambda} + \frac{0.5^2}{3.5-\lambda} + \frac{0.9^2}{7-\lambda} + \frac{0.8^2}{8-\lambda}$

$$\boxed{f(\lambda) := 1 - \rho\mathbf{v}^T(\lambda I - D)^{-1}\mathbf{v} = 1 - \rho\sum_{k=1}^{n}\frac{v_k^2}{\lambda - d_k} = 0.} \tag{5.17}$$

This equation is called **secular equation**. The secular equation has **poles** at the eigenvalues of D and **zeros** at the eigenvalues of $D + \rho\mathbf{v}\mathbf{v}^T$. Notice that

$$f'(\lambda) = \rho \sum_{k=1}^{n} \frac{v_k^2}{(\lambda - d_k)^2}.$$

Thus, the derivative of f is positive if $\rho > 0$ wherever it has a finite value. If $\rho < 0$ the derivative of f is negative (almost) everywhere. A typical graph of f with $\rho > 0$ is depicted in Fig. 5.1. (If ρ is negative the image can be flipped left to right.) The secular equation implies the **interlacing property** of the eigenvalues of D and of $D + \rho\mathbf{v}\mathbf{v}^T$,

$$d_1 < \lambda_1 < d_2 < \lambda_2 < \cdots < d_n < \lambda_n, \qquad \rho > 0. \tag{5.18}$$

or

$$\lambda_1 < d_1 < \lambda_2 < d_2 < \cdots < \lambda_n < d_n, \qquad \rho < 0. \tag{5.19}$$

So, we have to compute one eigenvalue in each of the intervals (d_i, d_{i+1}), $1 \le i < n$, and a further eigenvalue in (d_n, ∞) or $(-\infty, d_1)$. The corresponding eigenvector is then given by (5.15). Evidently, these tasks are easy to parallelize.

Equations (5.17) and (5.15) can also been obtained from the relations

$$\begin{aligned}
\begin{bmatrix} \frac{1}{\rho} & \mathbf{v}^T \\ \mathbf{v} & \lambda I - D \end{bmatrix} &= \begin{bmatrix} 1 & \mathbf{0}^T \\ \rho\mathbf{v} & I \end{bmatrix} \begin{bmatrix} \frac{1}{\rho} & \mathbf{0}^T \\ \mathbf{0} & \lambda I - D - \rho\mathbf{v}\mathbf{v}^T \end{bmatrix} \begin{bmatrix} 1 & \rho\mathbf{v}^T \\ \mathbf{0} & I \end{bmatrix} \\
&= \begin{bmatrix} 1 & \mathbf{v}^T(\lambda I - D)^{-1} \\ \mathbf{0} & I \end{bmatrix} \begin{bmatrix} \frac{1}{\rho} - \mathbf{v}^T(\lambda I - D)^{-1}\mathbf{v} & \mathbf{0}^T \\ \mathbf{0} & \lambda I - D \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0}^T \\ (\lambda I - D)^{-1}\mathbf{v} & I \end{bmatrix}.
\end{aligned}$$

These are simply block LDL^T factorizations of the first matrix. The first is the well-known one where the factorization is started with the $(1,1)$ block. The second is a 'backward' factorization that is started with the $(2,2)$ block. Because the determinants of the tridiagonal matrices are all unity, we have

$$\frac{1}{\rho}\det(\lambda I - D - \rho\mathbf{v}\mathbf{v}^T) = \frac{1}{\rho}(1 - \rho\mathbf{v}^T(\lambda I - D)^{-1}\mathbf{v})\det(\lambda I - D). \tag{5.20}$$

Denoting the eigenvalues of $D + \rho\mathbf{v}\mathbf{v}^T$ again by $\lambda_1 < \lambda_2 < \cdots < \lambda_n$ this implies

$$\begin{aligned}
\prod_{j=1}^{n}(\lambda - \lambda_j) &= (1 - \rho\mathbf{v}^T(\lambda I - D)^{-1}\mathbf{v}) \prod_{j=1}^{n}(\lambda - d_j) \\
&= \left(1 - \rho\sum_{k=1}^{n}\frac{v_k^2}{\lambda - d_k}\right)\prod_{j=1}^{n}(\lambda - d_j) \\
&= \prod_{j=1}^{n}(\lambda - d_j) - \rho\sum_{k=1}^{n} v_k^2 \prod_{j\neq k}(\lambda - d_j)
\end{aligned} \tag{5.21}$$

Setting $\lambda = d_k$ gives

$$\prod_{j=1}^{n}(d_k - \lambda_j) = -\rho v_k^2 \prod_{\substack{j=1 \\ j\neq i}}^{n}(d_k - d_j) \tag{5.22}$$

or

$$
\begin{aligned}
v_k^2 &= \frac{-1}{\rho}\,\frac{\displaystyle\prod_{j=1}^{n}(d_k-\lambda_j)}{\displaystyle\prod_{\substack{j=1\\ j\neq i}}^{n}(d_k-d_j)} = \frac{-1}{\rho}\,\frac{\displaystyle\prod_{j=1}^{k-1}(d_k-\lambda_j)}{\displaystyle\prod_{j=1}^{k-1}(d_k-d_j)}\,\frac{\displaystyle\prod_{j=k}^{n}(\lambda_j-d_k)(-1)^{n-k+1}}{\displaystyle\prod_{j=k+1}^{n}(d_j-d_k)(-1)^{n-k}} \\
&= \frac{1}{\rho}\,\frac{\displaystyle\prod_{j=1}^{k-1}(d_k-\lambda_j)}{\displaystyle\prod_{j=1}^{k-1}(d_k-d_j)}\,\frac{\displaystyle\prod_{j=k}^{n}(\lambda_j-d_k)}{\displaystyle\prod_{j=k+1}^{n}(d_j-d_k)} > 0.
\end{aligned}
\tag{5.23}
$$

Therefore, the quantity on the right side is positive, so

$$
v_k = \sqrt{\frac{\displaystyle\prod_{j=1}^{k-1}(d_k-\lambda_j)\prod_{j=k}^{n}(\lambda_j-d_k)}{\displaystyle\rho\prod_{j=1}^{k-1}(d_k-d_j)\prod_{j=k+1}^{n}(d_j-d_k)}}. \tag{5.24}
$$

(Similar arguments hold if $\rho < 0$.) Thus, we have the solution of the following **inverse eigenvalue problem**:

> Given $D = \operatorname{diag}(d_1, \ldots, d_n)$ and values $\lambda_1, \ldots, \lambda_n$ that satisfy (5.18). Find a vector $\mathbf{v} = [v_1, \ldots, v_n]^T$ with positive components v_k such that the matrix $D + \mathbf{v}\mathbf{v}^T$ has the *prescribed* eigenvalues $\lambda_1, \ldots, \lambda_n$.

The solution is given by (5.24). The positivity of the v_k makes the solution unique.

5.6 Solving the secular equation

In this section we follow closely the exposition of Demmel [3]. We consider the computation of the zero of $f(\lambda)$ in the interval (d_i, d_{i+1}). We assume that $\rho = 1$.

We may simply apply Newton's iteration to solve $f(\lambda) = 0$. However, if we look carefully at Fig. 5.1 then we notice that the tangent at certain points in (d_i, d_{i+1}) crosses the real axis outside this interval. This happens in particular if the weights v_i or v_{i+1} are small. Therefore that zero finder has to be adapted in such a way that it captures the poles at the interval endpoints. It is relatively straightforward to try the ansatz

$$
h(\lambda) = \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda} + c_3. \tag{5.25}
$$

Notice that, given the coefficients c_1, c_2, and c_3, the equation $h(\lambda) = 0$ can easily be solved by means of the equivalent quadratic equation

$$
c_1(d_{i+1} - \lambda) + c_2(d_i - \lambda) + c_3(d_i - \lambda)(d_{i+1} - \lambda) = 0. \tag{5.26}
$$

This equation has two zeros. Precisly one of them is inside (d_i, d_{i+1}).

The coefficients c_1, c_2, and c_3 are computed in the following way. Let us assume that we have available an approximation λ_j to the zero in (d_i, d_{i+1}). We request that $h(\lambda_j) = f(\lambda_j)$ and $h'(\lambda_j) = f'(\lambda_j)$. The exact procedure is as follows. We write

$$
f(\lambda) = 1 + \underbrace{\sum_{k=1}^{i} \frac{v_k^2}{d_k - \lambda}}_{\psi_1(\lambda)} + \underbrace{\sum_{k=i+1}^{n} \frac{v_k^2}{d_k - \lambda}}_{\psi_2(\lambda)} = 1 + \psi_1(\lambda) + \psi_2(\lambda). \tag{5.27}
$$

$\psi_1(\lambda)$ is a sum of positive terms and $\psi_2(\lambda)$ is a sum of negative terms. Both $\psi_1(\lambda)$ and $\psi_2(\lambda)$ can be computed accurately, whereas adding them would likely provoke cancellation and loss of relative accuracy. We now choose c_1 and $\hat{c}_1$ such that

$$h_1(\lambda) := \hat{c}_1 + \frac{c_1}{d_i - \lambda} \text{ satisfies } h_1(\lambda_j) = \psi_1(\lambda_j) \text{ and } h_1'(\lambda_j) = \psi_1'(\lambda_j). \tag{5.28}$$

This means that the graphs of h_1 and of ψ_1 are tangent at $\lambda = \lambda_j$. This is similar to Newton's method. However in Newton's method a straight line is fitted to the given function. The coefficients in (5.28) are given by

$$\begin{aligned} c_1 &= \psi_1'(\lambda_j)(d_i - \lambda_j)^2 > 0, \\ \hat{c}_1 &= \psi_1(\lambda_j) - \psi_1'(\lambda_j)(d_i - \lambda_j) = \sum_{k=1}^{i} v_k^2 \frac{d_k - d_i}{(d_k - \lambda_j)^2} \leq 0. \end{aligned}$$

Similarly, the two constants c_2 and $\hat{c}_2$ are determined such that

$$h_2(\lambda) := \hat{c}_2 + \frac{c_2}{d_{i+1} - \lambda} \text{ satisfies } h_2(\lambda_j) = \psi_2(\lambda_j) \text{ and } h_2'(\lambda_j) = \psi_2'(\lambda_j) \tag{5.29}$$

with the coefficients

$$\begin{aligned} c_2 &= \psi_2'(\lambda_j)(d_{i+1} - \lambda_j)^2 > 0, \\ \hat{c}_2 &= \psi_2(\lambda_j) - \psi_2'(\lambda_j)(d_{i+1} - \lambda_j) = \sum_{k=i+1}^{n} v_k^2 \frac{d_k - d_{i+1}}{(d_k - \lambda)^2} \geq 0. \end{aligned}$$

Finally, we set

$$h(\lambda) = 1 + h_1(\lambda) + h_2(\lambda) = \underbrace{(1 + \hat{c}_1 + \hat{c}_2)}_{c_3} + \frac{c_1}{d_i - \lambda} + \frac{c_2}{d_{i+1} - \lambda}. \tag{5.30}$$

This zerofinder is converging quadratically to the desired zero [7]. Usually 2 to 3 steps are sufficient to get the zero to machine precision. Therefore finding a zero only requires $\mathcal{O}(n)$ flops. Thus, finding all zeros costs $\mathcal{O}(n^2)$ floating point operations.

5.7 A first algorithm

We are now ready to give the divide and conquer algorithm, see Algorithm 5.1.
All steps except step 10 require $\mathcal{O}(n^2)$ operations to complete. The step 10 costs n^3 flops. Thus, the full divide and conquer algorithm, requires

$$\begin{aligned} T(n) &= n^3 + 2 \cdot T(n/2) = n^3 + 2\left(\frac{n}{2}\right)^3 + 4T(n/4) \\ &= n^3 + \frac{n^3}{4} + 4\left(\frac{n}{4}\right)^3 + 8T(n/8) = \cdots = \frac{4}{3}n^3. \end{aligned} \tag{5.31}$$

This *serial* complexity of the algorithm very often overestimates the computational costs of the algorithm due to significant deflation that is observed surprisingly often.

Algorithm 5.1 The tridiagonal divide and conquer algorithm

```
Let $T \in \mathbb{C}^{n\times n}$ be a real symmetric tridiagonal matrix. This algorithm computes
the spectral decomposition of $T = Q\Lambda Q^T$, where the diagonal $\Lambda$ is the matrix of
eigenvalues and $Q$ is orthogonal.
if $T$ is $1 \times 1$ then
    return $(\Lambda = T; Q = 1)$
else
    Partition $T = \begin{bmatrix} T_1 & O \\ O & T_2 \end{bmatrix} + \rho\mathbf{u}\mathbf{u}^T$ according to (5.2)
    Call this algorithm with $T_1$ as input and $Q_1$, $\Lambda_1$ as output.
    Call this algorithm with $T_2$ as input and $Q_2$, $\Lambda_2$ as output.
    Form $D + \rho\mathbf{v}\mathbf{v}^T$ from $\Lambda_1, \Lambda_2, Q_1, Q_2$ according to (5.4)–(5.6).
    Find the eigenvalues $\Lambda$ and the eigenvectors $Q'$ of $D + \rho\mathbf{v}\mathbf{v}^T$.
    Form $Q = \begin{bmatrix} Q_1 & O \\ O & Q_2 \end{bmatrix} \cdot Q'$ which are the eigenvectors of $T$.
    return $(\Lambda; Q)$
end if
```

5.7.1 A numerical example

Let A be a 4×4 matrix

$$A = D + \mathbf{v}\mathbf{v}^T = \begin{bmatrix} 0 & & & \\ & 2-\beta & & \\ & & 2+\beta & \\ & & & 5 \end{bmatrix} + \begin{bmatrix} 1 \\ \beta \\ \beta \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \beta & \beta & 1 \end{bmatrix}. \tag{5.32}$$

In this example (that is similar to one in [8]) we want to point at a problem that the divide and conquer algorithm possesses as it is given in Algorithm 5.1, namely the loss of orthogonality among eigenvectors.

Before we do some MATLAB tests let us look more closely at D and $\mathbf{v}$ in (5.32). This example becomes difficult to solve if β gets very small. In Figures 5.2 to 5.5 we see graphs of the function $f_\beta(\lambda)$ that appears in the secular equation for $\beta = 1$, $\beta = 0.1$, and $\beta = 0.01$. The critical zeros move towards 2 from both sides. The weights $v_2^2 = v_3^2 = \beta^2$ are however not so small that they should be deflated.

The following MATLAB code shows the problem. We execute the commands for $\beta = 10^{-k}$ for $k = 0, 1, 2, 4, 8$.

```
v = [1 beta beta 1]';          % rank-1 modification
d = [0, 2-beta, 2+beta, 5]';   % diagonal matrix

L = eig(diag(d) + v*v')        % eigenvalues of the modified matrix
e = ones(4,1);
q = (d*e'-e*L').\(v*e');       % unnormalized eigenvectors cf. (5.15)

Q = sqrt(diag(q'*q));
q = q./(e*Q');                 % normalized eigenvectors

norm(q'*q-eye(4))              % check for orthogonality
```

We do not bother how we compute the eigenvalues. We simply use MATLAB's built-in function `eig`. We get the results of Table 5.1.

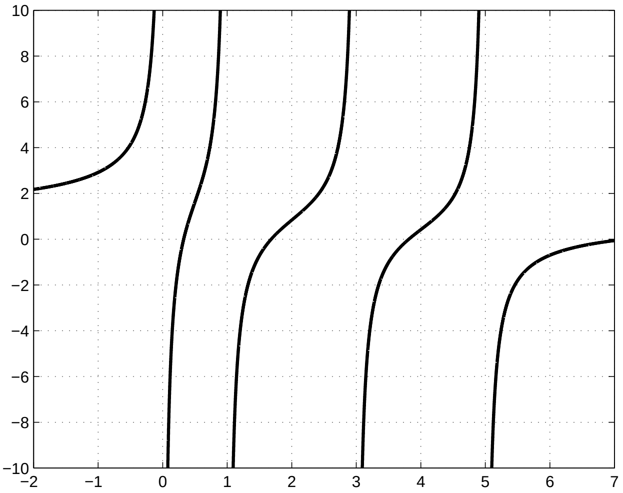


Figure 5.2: Secular equation corresponding to (5.32) for $\beta = 1$

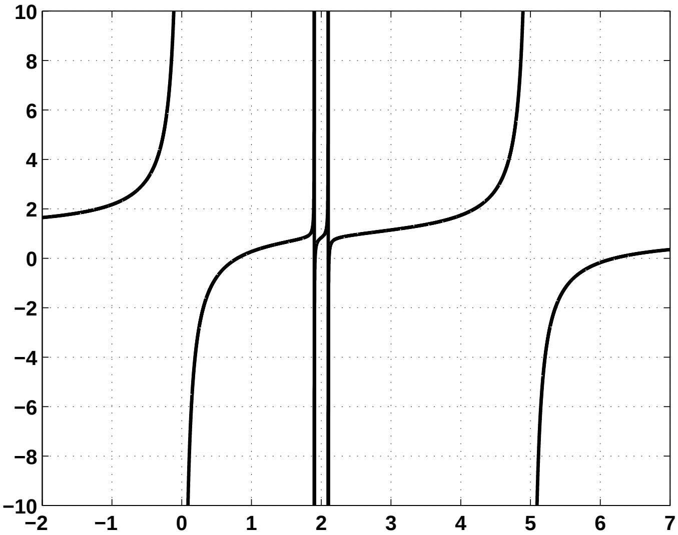


Figure 5.3: Secular equation corresponding to (5.32) for $\beta = 0.1$

We observe loss of orthogonality among the eigenvectors as the eigenvalues get closer and closer. This may not be surprising as we compute the eigenvectors by formula (5.15)

$$\mathbf{x} = \frac{(\lambda I - D)^{-1}\mathbf{v}}{\|(\lambda I - D)^{-1}\mathbf{v}\|}.$$

If $\lambda = \lambda_2$ and $\lambda = \lambda_3$ which are almost equal, $\lambda_2 \approx \lambda_3$ then intuitively one expects almost the same eigenvectors. We have in fact

$$Q^T Q - I_4 = \begin{bmatrix} -2.2204 \cdot 10^{-16} & 4.3553 \cdot 10^{-8} & 1.7955 \cdot 10^{-8} & -1.1102 \cdot 10^{-16} \\ 4.3553 \cdot 10^{-8} & 0 & -5.5511 \cdot 10^{-8} & -1.8298 \cdot 10^{-8} \\ 1.7955 \cdot 10^{-8} & -5.5511 \cdot 10^{-8} & -1.1102 \cdot 10^{-16} & -7.5437 \cdot 10^{-9} \\ -1.1102 \cdot 10^{-16} & -1.8298 \cdot 10^{-8} & -7.5437 \cdot 10^{-9} & 0 \end{bmatrix}.$$

Orthogonality is lost only with respect to the vectors corresponding to the eigenvalues close to 2.

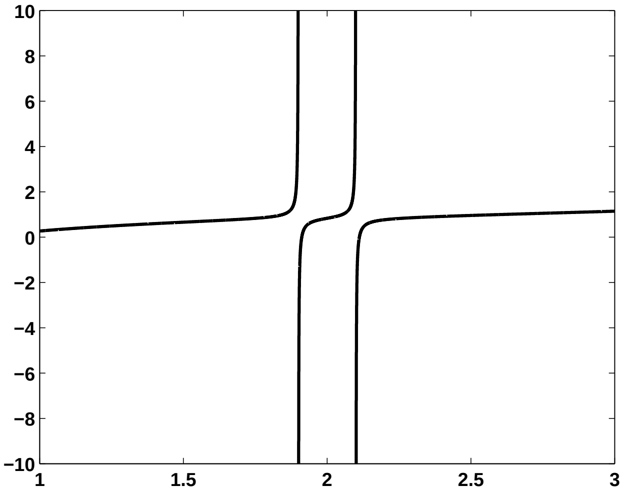


Figure 5.4: Secular equation corresponding to (5.32) for $\beta = 0.1$ for $1 \le \lambda \le 3$

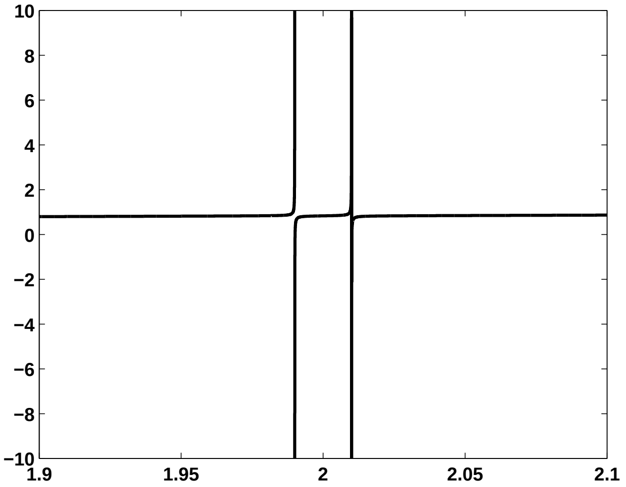


Figure 5.5: Secular equation corresponding to (5.32) for $\beta = 0.01$ for $1.9 \le \lambda \le 2.1$

Already Dongarra and Sorensen [4] analyzed this problem. In their formulation they normalize the vector $\mathbf{v}$ of $D + \rho\mathbf{v}\mathbf{v}^T$ to have norm unity, $\|\mathbf{v}\| = 1$. They stated

Lemma 5.1 *Let*

$$\mathbf{q}_\lambda^T = \left(\frac{v_1}{d_1 - \lambda}, \frac{v_2}{d_2 - \lambda}, \ldots, \frac{v_n}{d_n - \lambda}\right)\left[\frac{\rho}{f'(\lambda)}\right]^{1/2}. \tag{5.33}$$

Then for any $\lambda, \mu \notin \{d_1, \ldots, d_n\}$ *we have*

$$|\mathbf{q}_\lambda^T \mathbf{q}_\mu| = \frac{1}{|\lambda - \mu|}\frac{|f(\lambda) - f(\mu)|}{[f'(\lambda)f'(\mu)]^{1/2}}. \tag{5.34}$$

Proof. Observe that

$$\frac{\lambda - \mu}{(d_j - \lambda)(d_j - \mu)} = \frac{1}{d_j - \lambda} - \frac{1}{d_j - \mu}.$$

Then the proof is straightforward. ■

β	λ_1	λ_2	λ_3	λ_4	$\|Q^TQ - I\|$
1	0.325651	1.682219	3.815197	7.176933	$5.6674 \cdot 10^{-16}$
0.1	0.797024	1.911712	2.112111	6.199153	$3.4286 \cdot 10^{-15}$
0.01	0.807312	1.990120	2.010120	6.192648	$3.9085 \cdot 10^{-14}$
10^{-4}	0.807418	1.999900	2.000100	6.192582	$5.6767 \cdot 10^{-12}$
10^{-8}	0.807418	1.99999999000000	2.00000001000000	6.192582	$8.3188 \cdot 10^{-08}$

Table 5.1: Loss of orthogonality among the eigenvectors computed by (5.15)

Formula (5.34) indicates how problems may arise. In exact arithmetic, if λ and μ are eigenvalues then $f(\lambda) = f(\mu) = 0$. However, in floating point arithmetic this values may be small but nonzero, e.g., $\mathcal{O}(\varepsilon)$. If $|\lambda - \mu|$ is very small as well then we may have trouble! So, a remedy for the problem was for a long time to compute the eigenvalues in doubled precision, so that $f(\lambda) = \mathcal{O}(\varepsilon^2)$. This would counteract a potential $\mathcal{O}(\varepsilon)$ of $|\lambda - \mu|$.

This solution was quite unsatisfactory because doubled precision is in general very slow since it is implemented in software. It took a decade until a proper solution was found.

5.8 The algorithm of Gu and Eisenstat

Computing eigenvector according to the formula

$$\mathbf{x} = \alpha(\lambda I - D)^{-1}\mathbf{v} = \alpha \begin{pmatrix} \dfrac{v_1}{\lambda - d_1} \\ \vdots \\ \dfrac{v_n}{\lambda - d_n} \end{pmatrix}, \qquad \alpha = \|(\lambda I - D)^{-1}\mathbf{v}\|, \tag{5.35}$$

is bound to fail if λ is very close to a pole d_k and the difference $\lambda - d_k$ has an error of size $\mathcal{O}(\varepsilon|d_k|)$ instead of only $\mathcal{O}(\varepsilon|d_k - \lambda|)$. To resolve this problem Gu and Eisenstat [5] found a trick that is at the same time ingenious and simple.

They observed that the v_k in (5.24) are very accurately determined by the data d_i and λ_i. Therefore, once the eigenvalues are computed *accurately* a vector $\hat{\mathbf{v}}$ could be computed such that the λ_i are *accurate* eigenvalues of $D + \hat{\mathbf{v}}\hat{\mathbf{v}}$. If $\hat{\mathbf{v}}$ approximates well the original $\mathbf{v}$ then the new eigenvectors will be the exact eigenvectors of a slightly modified eigenvalue problem, which is all we can hope for.

The zeros of the secular equation can be computed accurately by the method presented in section 5.6. However, a shift of variables is necessary. In the interval (d_i, d_{i+1}) the origin of the real axis is moved to d_i if λ_i is closer to d_i than to d_{i+1}, i.e., if $f((d_i + d_{i+1})/2) > 0$. Otherwise, the origin is shifted to d_{i+1}. This shift of the origin avoids the computation of the smallest difference $d_i - \lambda$ (or $d_{i+1} - \lambda$) in (5.35), thus avoiding cancellation in this most sensitive quantity. Equation (5.26) can be rewritten as

$$\underbrace{(c_1\Delta_{i+1} + c_2\Delta_i + c_3\Delta_i\Delta_{i+1})}_{b} - \underbrace{(c_1 + c_2 + c_3(\Delta_i + \Delta_{i+1}))}_{-a}\eta + \underbrace{c_3}_{c}\eta^2 = 0, \tag{5.36}$$

where $\Delta_i = d_i - \lambda_j$, $\Delta_{i+1} = d_{i+1} - \lambda_j$, and $\lambda_{j+1} = \lambda_j + \eta$ is the next approximate zero. With equations (5.28)–(5.30) the coefficients in (5.36) get

$$\begin{aligned} a &= c_1 + c_2 + c_3(\Delta_i + \Delta_{i+1}) = (1 + \Psi_1 + \Psi_2)(\Delta_i + \Delta_{i+1}) - (\Psi_1' + \Psi_2')\Delta_i\Delta_{i+1}, \\ b &= c_1\Delta_{i+1} + c_2\Delta_i + c_3\Delta_i\Delta_{i+1} = \Delta_i\Delta_{i+1}(1 + \Psi_1 + \Psi_2), \\ c &= c_3 = 1 + \Psi_1 + \Psi_2 - \Delta_i\Psi_1' - \Delta_{i+1}\Psi_2'. \end{aligned} \tag{5.37}$$

If we are looking for a zero that is closer to d_i than to d_{i+1} then we move the origin to λ_j, i.e., we have e.g. $\Delta_i = -\lambda_j$. The solution of (5.36) that lies inside the interval is [7]

$$\eta = \begin{cases} \dfrac{a - \sqrt{a^2 - 4bc}}{2c}, & \text{if } a \le 0, \\ \dfrac{2b}{a + \sqrt{a^2 - 4bc}}, & \text{if } a > 0. \end{cases} \tag{5.38}$$

The following algorithm shows how step 9 of the tridiagonal divide and conquer algorithm 5.1 must be implemented.

Algorithm 5.2 A stable eigensolver for $D + \mathbf{v}\mathbf{v}^T$

```
This algorithm stably computes the spectral decomposition of $D+\mathbf{v}\mathbf{v}^T = Q\Lambda Q^T$ where
    $D = \mathrm{diag}(d_1, \dots d_n)$, $\mathbf{v} = [v_1, \dots, v_n] \in \mathbb{R}^n$, $\Lambda = \mathrm{diag}(\lambda_1, \dots \lambda_n)$, and $Q = [\mathbf{q}_1, \dots, \mathbf{q}_n]$.
$d_{i+1} = d_n + \|\mathbf{v}\|^2$.
In each interval $(d_i, d_{i+1})$ compute the zero $\lambda_i$ of the secular equation $f(\lambda) = 0$.
Use the formula (5.24) to compute the vector $\hat{\mathbf{v}}$ such that the $\lambda_i$ are the ‘exact’
    eigenvalues of $D + \hat{\mathbf{v}}\hat{\mathbf{v}}$.
In each interval $(d_i, d_{i+1})$ compute the eigenvectors of $D + \hat{\mathbf{v}}\hat{\mathbf{v}}$ according to (5.15),
    $\mathbf{q}_i = \frac{(\lambda_i I - D)^{-1}\hat{\mathbf{v}}}{\|(\lambda_i I - D)^{-1}\hat{\mathbf{v}}\|}$.
return $(\Lambda; Q)$
```

5.8.1 A numerical example [continued]

We continue the discussion of the example on page 100 where the eigenvalue problem of

$$A = D + \mathbf{v}\mathbf{v}^T = \begin{bmatrix} 0 & & & \\ & 2-\beta & & \\ & & 2+\beta & \\ & & & 5 \end{bmatrix} + \begin{bmatrix} 1 \\ \beta \\ \beta \\ 1 \end{bmatrix} \begin{bmatrix} 1 & \beta & \beta & 1 \end{bmatrix}. \tag{5.39}$$

The MATLAB code that we showed did not give orthogonal eigenvectors. We show in the following script that the formulae (5.24) really solve the problem.

```
dlam = zeros(n,1);
for k=1:n,
  [dlam(k), dvec(:,k)] = zerodandc(d,v,k);
end

V = ones(n,1);
for k=1:n,
  V(k) = prod(abs(dvec(k,:)))/prod(d(k) - d(1:k-1))/prod(d(k+1:n) - d(k));
  V(k) = sqrt(V(k));
end

Q = (dvec).\(V*e’);
diagq = sqrt(diag(Q’*Q));
Q = Q./(e*diagq’);
```

```
for k=1:n,
  if dlam(k)>0,
    dlam(k) = dlam(k) + d(k);
  else
    dlam(k) = d(k+1) + dlam(k);
  end
end

norm(Q'*Q-eye(n))
norm((diag(d) + v*v')*Q - Q*diag(dlam'))
```

A zero finder returns for each interval the quantity $\lambda_i - d_i$ and the vector $[d_1 - \lambda_i, \ldots, d_n - \lambda_i]^T$ to high precision. These vector elements have been computed as $(d_k - d_i) - (\lambda_i - d_i)$. The zerofinder of Li [7] has been employed here. At the end of this section we list the zerofinder written in MATLAB that was used here. The formulae (5.37) and (5.38) have been used to solve the quadratic equation (5.36). Notice that only one of the `while` loops is traversed, depending on if the zero is closer to the pole on the left or to the right of the interval. The v_k of formula (5.24) are computed next. Q contains the eigenvectors.

β	Algorithm	$\|Q^TQ - I\|$	$\|AQ - Q\Lambda\|$
0.1	I	$3.4286 \cdot 10^{-15}$	$5.9460 \cdot 10^{-15}$
	II	$2.2870 \cdot 10^{-16}$	$9.4180 \cdot 10^{-16}$
0.01	I	$3.9085 \cdot 10^{-14}$	$6.9376 \cdot 10^{-14}$
	II	$5.5529 \cdot 10^{-16}$	$5.1630 \cdot 10^{-16}$
10^{-4}	I	$5.6767 \cdot 10^{-12}$	$6.3818 \cdot 10^{-12}$
	II	$2.2434 \cdot 10^{-16}$	$4.4409 \cdot 10^{-16}$
10^{-8}	I	$8.3188 \cdot 10^{-08}$	$1.0021 \cdot 10^{-07}$
	II	$2.4980 \cdot 10^{-16}$	$9.4133 \cdot 10^{-16}$

Table 5.2: Loss of orthogonality among the eigenvectors computed by the straightforward algorithm (I) and the Gu-Eisenstat approach (II)

Again we ran the code for $\beta = 10^{-k}$ for $k = 0, 1, 2, 4, 8$. The numbers in Table 5.2 confirm that the new formulae are much more accurate than the straight forward ones. The norms of the errors obtained for the Gu-Eisenstat algorithm always are in the order of machine precision, i.e., 10^{-16}.

Note that there may be errors in the following code! The vector `v` is squared on the 3rd line, and later (assigning values to `di1`, `psi1`, `psi2`) squared again!

```
function [lambda,dl] = zerodandc(d,v,i)
% ZERODANDC - Computes eigenvalue lambda in the i-th interval
%             (d(i), d(i+1)) with Li's 'middle way' zero finder
%             dl is the n-vector [d(1..n) - lambda]

n = length(d);
di = d(i);
v = v.^2;
if i < n,
  di1 = d(i+1);  lambda = (di + di1)/2;
else
  di1 = d(n) + norm(v)^2; lambda = di1;
```

```
end
eta = 1;
psi1 = sum((v(1:i).^2)./(d(1:i) - lambda));
psi2 = sum((v(i+1:n).^2)./(d(i+1:n) - lambda));

if  1 + psi1 + psi2 > 0,  %  zero is on the left half of the interval

  d = d - di;  lambda = lambda - di; di1 = di1 - di; di = 0;

  while  abs(eta) > 10*eps
    psi1 = sum(v(1:i)./(d(1:i) - lambda));
    psi1s = sum(v(1:i)./((d(1:i) - lambda)).^2);

    psi2 = sum((v(i+1:n))./(d(i+1:n) - lambda));
    psi2s = sum(v(i+1:n)./((d(i+1:n) - lambda)).^2);

    % Solve for zero
    Di = -lambda; Di1 = di1 - lambda;
    a = (Di + Di1)*(1 + psi1 + psi2) - Di*Di1*(psi1s + psi2s);
    b = Di*Di1*(1 + psi1 + psi2);
    c = (1 + psi1 + psi2) - Di*psi1s - Di1*psi2s;
    if a > 0,
      eta = (2*b)/(a + sqrt(a^2 - 4*b*c));
    else
      eta = (a - sqrt(a^2 - 4*b*c))/(2*c);
    end
    lambda = lambda + eta;
  end

else  %  zero is on the right half of the interval

  d = d - di1; lambda = lambda - di1; di = di - di1; di1 = 0;

  while  abs(eta) > 10*eps
    psi1 = sum(v(1:i)./(d(1:i) - lambda));
    psi1s = sum(v(1:i)./((d(1:i) - lambda)).^2);

    psi2 = sum((v(i+1:n))./(d(i+1:n) - lambda));
    psi2s = sum(v(i+1:n)./((d(i+1:n) - lambda)).^2);

    % Solve for zero
    Di = di - lambda; Di1 = - lambda;
    a = (Di + Di1)*(1 + psi1 + psi2) - Di*Di1*(psi1s + psi2s);
    b = Di*Di1*(1 + psi1 + psi2);
    c = (1 + psi1 + psi2) - Di*psi1s - Di1*psi2s;
    if a > 0,
      eta = (2*b)/(a + sqrt(a^2 - 4*b*c));
    else
      eta = (a - sqrt(a^2 - 4*b*c))/(2*c);
    end
    lambda = lambda + eta;
  end

end
```

```
    dl = d - lambda;

    return
```

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide - Release 2.0*, SIAM, Philadelphia, PA, 1994. (Software and guide are available from Netlib at URL http://www.netlib.org/lapack/).

[2] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numer. Math., 36 (1981), pp. 177–195.

[3] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, PA, 1997.

[4] J. J. Dongarra and D. C. Sorensen, *A fully parallel algorithm for the symmetric eigenvalue problem*, SIAM J. Sci. Stat. Comput., 8 (1987), pp. s139–s154.

[5] M. Gu and S. C. Eisenstat, *A stable and efficient algorithm for the rank-one modification of the symmetric eigenproblem*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 1266–1276.

[6] ——, *A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 172–191.

[7] R.-C. Li, *Solving secular equations stably and efficiently*, Technical Report UT-CS-94-260, University of Tennessee, Knoxville, TN, Nov. 1994. LAPACK Working Note No. 89.

[8] D. C. Sorensen and P. T. P. Tang, *On the orthogonality of eigenvectors computed by divide-and-conquer techniques*, SIAM J. Numer. Anal., 28 (1991), pp. 1752–1775.

Chapter 6

LAPACK and the BLAS

6.1 LAPACK

(This section is essentially compiled from the LAPACK User's Guide [1] that is available online from http://www.netlib.org/lapack/lug/.)

LAPACK [1] is a library of Fortran 77 subroutines for solving the most commonly occurring problems in numerical linear algebra. It has been designed to be efficient on a wide range of modern high-performance computers. The name LAPACK is an acronym for **L**inear **A**lgebra **PACK**age.

LAPACK can solve systems of linear equations, linear least squares problems, eigenvalue problems and singular value problems. LAPACK can also handle many associated computations such as matrix factorizations or estimating condition numbers.

LAPACK contains **driver routines** for solving standard types of problems, **computational routines** to perform a distinct computational task, and **auxiliary routines** to perform a certain subtask or common low-level computation. Each driver routine typically calls a sequence of computational routines. Taken as a whole, the computational routines can perform a wider range of tasks than are covered by the driver routines. Many of the auxiliary routines may be of use to numerical analysts or software developers, so we have documented the Fortran source for these routines with the same level of detail used for the LAPACK routines and driver routines.

Dense and banded matrices are provided for, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices.

LAPACK is designed to give high efficiency on vector processors, high-performance "super-scalar" workstations, and shared memory multiprocessors. It can also be used satisfactorily on all types of scalar machines (PC's, workstations, mainframes). A distributed-memory version of LAPACK, **ScaLAPACK** [2], has been developed for other types of parallel architectures (for example, massively parallel SIMD machines, or distributed memory machines).

LAPACK has been designed to supersede LINPACK [3] and EISPACK [10, 8], principally by restructuring the software to achieve much greater efficiency, where possible, on modern high-performance computers; also by adding extra functionality, by using some new or improved algorithms, and by integrating the two sets of algorithms into a unified package.

LAPACK routines are written so that as much as possible of the computation is performed by calls to the Basic Linear Algebra Subprograms (**BLAS**) [9, 6, 5]. Highly efficient machine-specific implementations of the BLAS are available for many modern high-performance computers. The BLAS enable LAPACK routines to achieve high performance with portable code.

The BLAS are not strictly speaking part of LAPACK, but Fortran 77 code for the BLAS is distributed with LAPACK, or can be obtained separately from `netlib` where "model implementations" are found.

The model implementation is not expected to perform as well as a specially tuned implementation on most high-performance computers – on some machines it may give much worse performance – but it allows users to run LAPACK codes on machines that do not offer any other implementation of the BLAS.

The complete LAPACK package or individual routines from LAPACK are freely available from the World Wide Web or by anonymous ftp. The LAPACK homepage can be accessed via the URL `http://www.netlib.org/lapack/`.

6.2 BLAS

By 1976 it was clear that some standardization of basic computer operations on vectors was needed [9]. By then it was already known that coding procedures that worked well on one machine might work very poorly on others. In consequence of these observations, Lawson, Hanson, Kincaid and Krogh proposed a limited set of **B**asic **L**inear **A**lgebra **S**ubprograms (**BLAS**) to be (hopefully) optimized by hardware vendors, implemented in assembly language if necessary, that would form the basis of comprehensive linear algebra packages [9]. These so-called Level 1 BLAS consisted of vector operations and some attendant co-routines. The first major package which used these BLAS kernels was LINPACK [3]. Soon afterward, other major software libraries such as the IMSL library and NAG rewrote portions of their existing codes and structured new routines to use these BLAS. Early in their development, vector computers saw significant optimizations using the BLAS. Soon, however, such machines were clustered together in tight networks and somewhat larger kernels for numerical linear algebra were developed [6, 7] to include matrix-vector operations (Level 2 BLAS). Additionally, FORTRAN compilers were by then optimizing vector operations as efficiently as hand coded Level 1 BLAS. Subsequently, in the late 1980s, distributed memory machines were in production and shared memory machines began to have significant numbers of processors. A further set of matrix-matrix operations was proposed [4] and soon standardized [5] to form a Level 3. The first major package for linear algebra which used the Level 3 BLAS was LAPACK [1] and subsequently a scalable (to large numbers of processors) version was released as ScaLAPACK [2]. Vendors focused on Level 1, Level 2, and Level 3 BLAS which provided an easy route to optimizing LINPACK, then LAPACK. LAPACK not only integrated pre-existing solvers and eigenvalue routines found in EISPACK [10] (which did not use the BLAS) and LINPACK (which used Level 1 BLAS), but incorporated the latest dense and banded linear algebra algorithms available. It also used the Level 3 BLAS which were optimized by much vendor effort. Later, we will illustrate several BLAS routines. Conventions for different BLAS are indicated by

- A **root** operation. For example, `_axpy` for the operation

$$\mathbf{y} := a{\cdot}\mathbf{x} + \mathbf{y} \tag{6.1}$$

- A prefix (or combination prefix) to indicate the datatype of the operands, for example `saxpy` for single precision `_axpy` operation, or `isamax` for the index of the maximum absolute element in an array of type **single**.

- a suffix if there is some qualifier, for example `cdotc` or `cdotu` for conjugated or unconjugated complex dot product, respectively:

$$\texttt{cdotc(n,x,1,y,1)} = \sum_{i=0}^{n-1} x_i \bar{y}_i$$
$$\texttt{cdotu(n,x,1,y,1)} = \sum_{i=0}^{n-1} x_i y_i$$

  where both $\mathbf{x}, \mathbf{y}$ are vectors of complex elements.

Tables 6.1 and 6.2 give the prefix/suffix and root combinations for the BLAS, respectively.

Prefixes:	
S	REAL
D	DOUBLE PRECISION
C	COMPLEX
Z	DOUBLE COMPLEX
Suffixes:	
U	transpose
C	Hermitian conjugate

Table 6.1: Basic Linear Algebra Subprogram prefix/suffix conventions.

6.2.1 Typical performance numbers for the BLAS

Let us look at typical representations of all three levels of the BLAS, `daxpy`, `ddot`, `dgemv`, and `dgemm`, that perform some basic operations. Additionally, we look at the rank-1 update routine `dger`. An overview on the number of memory accesses and floating point operations is given in Table 6.3. The Level 1 BLAS comprise basic vector operations. A call of one of the Level 1 BLAS thus gives rise to $\mathcal{O}(n)$ floating point operations and $\mathcal{O}(n)$ memory accesses. Here, n is the vector length. The Level 2 BLAS comprise operations that involve matrices *and* vectors. If the involved matrix is n-by-n then both the memory accesses and the floating point operations are of $\mathcal{O}(n^2)$. In contrast, the Level 3 BLAS have a higher order of floating point operations than memory accesses. The most prominent operation of the Level 3 BLAS, matrix-matrix multiplication costs $\mathcal{O}(n^3)$ floating point operations while there are only $\mathcal{O}(n^2)$ reads and writes. The last column in Table 6.3 shows the crucial difference between the Level 3 BLAS and the rest.

Table 6.4 gives some performance numbers for the five BLAS of Table 6.3. Notice that the timer has a resolution of only 1 μsec! Therefore, the numbers in Table 6.4 have been obtained by timing a loop inside of which the respective function is called many times. The Mflop/s rates of the Level 1 BLAS `ddot` and `daxpy` quite precisely reflect the ratios of the memory accesses of the two routines, $2n$ vs. $3n$. The high rates are for vectors that can be held in the on-chip cache of 512 MB. The low 240 and 440 Mflop/s with the very long vectors are related to the memory bandwidth of about 1900 MB/s.

The Level 2 BLAS `dgemv` has about the same performance as `daxpy` if the matrix can be held in cache ($n = 100$). Otherwise it is considerably reduced. `dger` has a high volume of read and write operations, while the number of floating point operations is limited.

Level 1 BLAS	
`_rotg`, `_rot`	Generate/apply plane rotation
`_rotmg`, `_rotm`	Generate/apply modified plane rotation
`_swap`	Swap two vectors: $\mathbf{x} \leftrightarrow \mathbf{y}$
`_scal`	Scale a vector: $\mathbf{x} \leftarrow \alpha\mathbf{x}$
`_copy`	Copy a vector: $\mathbf{x} \leftarrow \mathbf{y}$
`_axpy`	_axpy operation: $\mathbf{y} \leftarrow \mathbf{y} + \alpha\mathbf{x}$
`_dot_`	Dot product: $s \leftarrow \mathbf{x} \cdot \mathbf{y} = \mathbf{x}^*\mathbf{y}$
`_nrm2`	2-norm: $s \leftarrow \|\mathbf{x}\|_\mathbf{2}$
`_asum`	1-norm: $s \leftarrow \|\mathbf{x}\|_\mathbf{1}$
`i_amax`	Index of largest vector element: first i such $\|x_i\| \geq \|x_k\|$ for all k
Level 2 BLAS	
`_gemv`, `_gbmv`	General (banded) matrix-vector multiply: $\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$
`_hemv`, `_hbmv`, `_hpmv`	Hermitian (banded, packed) matrix-vector multiply: $\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$
`_semv`, `_sbmv`, `_spmv`	Symmetric (banded, packed) matrix-vector multiply: $\mathbf{y} \leftarrow \alpha A\mathbf{x} + \beta\mathbf{y}$
`_trmv`, `_tbmv`, `_tpmv`	Triangular (banded, packed) matrix-vector multiply: $\mathbf{x} \leftarrow A\mathbf{x}$
`_trsv`, `_tbsv`, `_tpsv`	Triangular (banded, packed) system solves (forward/backward substitution): $\mathbf{x} \leftarrow A^{-1}\mathbf{x}$
`_ger`, `_geru`, `_gerc`	Rank-1 updates: $A \leftarrow \alpha\mathbf{x}\mathbf{y}^* + A$
`_her`, `_hpr`, `_syr`, `_spr`	Hermitian/symmetric (packed) rank-1 updates: $A \leftarrow \alpha\mathbf{x}\mathbf{x}^* + A$
`_her2`, `_hpr2`,`_syr2`, `_spr2`	Hermitian/symmetric (packed) rank-2 updates: $A \leftarrow \alpha\mathbf{x}\mathbf{y}^* + \alpha^*\mathbf{y}\mathbf{x}^* + A$
Level 3 BLAS	
`_gemm`, `_symm`, `_hemm`	General/symmetric/Hermitian matrix-matrix multiply: $C \leftarrow \alpha AB + \beta C$
`_syrk`, `_herk`	Symmetric/Hermitian rank-k update: $C \leftarrow \alpha AA^* + \beta C$
`_syr2k`, `_her2k`	Symmetric/Hermitian rank-k update: $C \leftarrow \alpha AB^* + \alpha^* BA^* + \beta C$
`_trmm`	Multiple triangular matrix-vector multiplies: $B \leftarrow \alpha AB$
`_trsm`	Multiple triangular system solves: $B \leftarrow \alpha A^{-1}B$

Table 6.2: Summary of the Basic Linear Algebra Subroutines.

	read	write	flops	flops / mem access
`ddot`	$2n$	1	$2n$	1
`daxpy`	$2n$	n	$2n$	2/3
`dgemv`	n^2+n	n	$2n^2$	2
`dger`	n^2+2n	n^2	$2n^2$	1
`dgemm`	$2n^2$	n^2	$2n^3$	$2n/3$

Table 6.3: Number of memory references and floating point operations for vectors of length n.

	$n = 100$	500	2’000	10’000’000
`ddot`	1480	1820	1900	440
`daxpy`	1160	1300	1140	240
`dgemv`	1370	740	670	—
`dger`	670	330	320	—
`dgemm`	2680	3470	3720	—

Table 6.4: Some performance numbers for typical BLAS in Mflop/s for a 2.4 GHz Pentium 4.

This leads to a very low performance rate. The Level 3 BLAS `dgemm` performs at a good fraction of the peak performance of the processor (4.8Gflop/s). The performance increases with the problem size. We see from Table 6.3 that the ratio of computation to memory accesses increases with the problem size. This ratio is analogous to a volume to surface area effect.

6.3 Blocking

In the previous section we have seen that it is important to use Level 3 BLAS. However, in the algorithm we have treated so far, there were no blocks. For instance, in the reduction to Hessenberg form we applied Householder (elementary) reflectors from left and right to a matrix to introduce zeros in one of its columns.

The essential point here is to *gather* a number of reflectors to a single block transformation. Let $P_i = I - 2\mathbf{u}_i\mathbf{u}_i^*$, $i = 1, 2, 3$, be three Householder reflectors. Their product is

$$\begin{aligned} P = P_3P_2P_1 &= (I - 2\mathbf{u}_3\mathbf{u}_3^*)(I - 2\mathbf{u}_2\mathbf{u}_2^*)(I - 2\mathbf{u}_1\mathbf{u}_1^*) \\ &= I - 2\mathbf{u}_3\mathbf{u}_3^* - 2\mathbf{u}_2\mathbf{u}_2^* - 2\mathbf{u}_1\mathbf{u}_1^* + 4\mathbf{u}_3\mathbf{u}_3^*\mathbf{u}_2\mathbf{u}_2^* + 4\mathbf{u}_3\mathbf{u}_3^*\mathbf{u}_1\mathbf{u}_1^* + 4\mathbf{u}_2\mathbf{u}_2^*\mathbf{u}_1\mathbf{u}_1^* \\ &\qquad + 8\mathbf{u}_3\mathbf{u}_3^*\mathbf{u}_2\mathbf{u}_2^*\mathbf{u}_1\mathbf{u}_1^* \\ &= I - [\mathbf{u}_1\mathbf{u}_2\mathbf{u}_3] \begin{bmatrix} 2 & & \\ 4\mathbf{u}_2^*\mathbf{u}_1 & 2 & \\ 4\mathbf{u}_3^*\mathbf{u}_1 + 8(\mathbf{u}_3^*\mathbf{u}_2)(\mathbf{u}_2^*\mathbf{u}_1) & 4\mathbf{u}_3^*\mathbf{u}_2 & 2 \end{bmatrix} [\mathbf{u}_1\mathbf{u}_2\mathbf{u}_3]^*. \end{aligned} \tag{6.2}$$

So, if e.g. three rotations are to be applied on a matrix in blocked fashon, then the three Householder vectors $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$ have to be found first. To that end the rotations are first applied only on the first three columns of the matrix, see Fig. 6.1. Then, the blocked rotation is applied to the rest of the matrix.

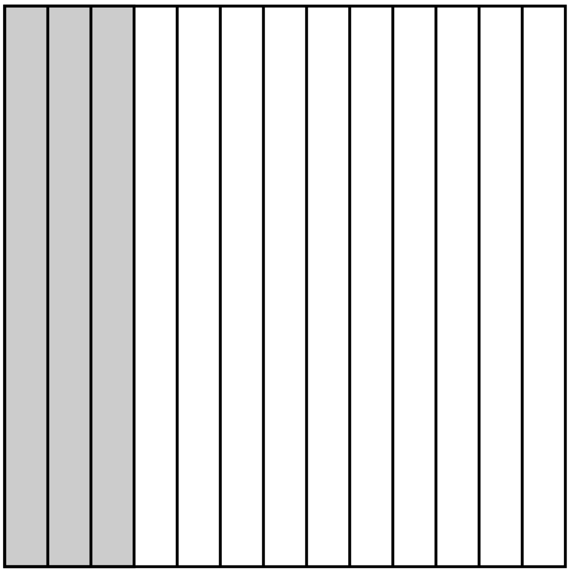


Figure 6.1: Blocking Householder reflections

Remark 6.1. Notice that a similar situation holds for **Gaussian elimination** because

$$\begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & & 1 & & \\ \vdots & & & \ddots & \\ l_{n1} & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & l_{32} & 1 & & \\ & \vdots & & \ddots & \\ & l_{n2} & & & 1 \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ \vdots & \vdots & & \ddots & \\ l_{n1} & l_{n2} & & & 1 \end{bmatrix}.$$

However, things are a complicated because of pivoting. □

6.4 LAPACK solvers for the symmetric eigenproblems

To give a feeling how LAPACK is organized we consider solvers for the symmetric eigenproblem (SEP). Except for this problem there are driver routines for linear systems, least squares problems, nonsymmetric eigenvalue problems, the computation of the singular value decomposition (SVD).

The basic task of the symmetric eigenproblem routines is to compute values of λ and, optionally, corresponding vectors $\mathbf{z}$ for a given matrix A.

There are four types of driver routines for symmetric and Hermitian eigenproblems. Originally LAPACK had just the simple and expert drivers described below, and the other two were added after improved algorithms were discovered. Ultimately we expect the algorithm in the most recent driver (called RRR below) to supersede all the others, but in LAPACK 3.0 the other drivers may still be faster on some problems, so we retain them.

- A simple driver computes all the eigenvalues and (optionally) eigenvectors.

- An expert driver computes all or a selected subset of the eigenvalues and (optionally) eigenvectors. If few enough eigenvalues or eigenvectors are desired, the expert driver is faster than the simple driver.

- A divide-and-conquer driver solves the same problem as the simple driver. It is much faster than the simple driver for large matrices, but uses more workspace. The name divide-and-conquer refers to the underlying algorithm.

- A relatively robust representation (RRR) driver computes all or (in a later release) a subset of the eigenvalues, and (optionally) eigenvectors. It is the fastest algorithm of all (except for a few cases), and uses the least workspace. The name RRR refers to the underlying algorithm.

This computation proceeds in the following stages:

1. The real symmetric or complex Hermitian matrix A is reduced to real tridiagonal form T. If A is real symmetric this decomposition is $A = QTQ^T$ with Q orthogonal and T symmetric tridiagonal. If A is complex Hermitian, the decomposition is $A = QTQ^H$ with Q unitary and T, as before, real symmetric tridiagonal.

2. Eigenvalues and eigenvectors of the real symmetric tridiagonal matrix T are computed. If all eigenvalues and eigenvectors are computed, this is equivalent to factorizing T as $T = S\Lambda S^T$, where S is orthogonal and Λ is diagonal. The diagonal entries of Λ are the eigenvalues of T, which are also the eigenvalues of A, and the columns of S are the eigenvectors of T; the eigenvectors of A are the columns of $Z = QS$, so that $A = Z\Lambda Z^T$ ($Z\Lambda Z^H$ when A is complex Hermitian).

In the real case, the decomposition $A = QTQ^T$ is computed by one of the routines `_sytrd`, `_sptrd`, or `_sbtrd`, depending on how the matrix is stored. The complex analogues of these routines are called `_hetrd`, `_hptrd`, and `_hbtrd`. The routine `_sytrd` (or `_hetrd`) represents the matrix Q as a product of elementary reflectors. The routine `_orgtr` (or in the complex case `_unmtr`) is provided to form Q explicitly; this is needed in particular before calling `_steqr` to compute all the eigenvectors of A by the QR algorithm. The routine `_ormtr` (or in the complex case `_unmtr`) is provided to multiply another matrix by Q without forming Q explicitly; this can be used to transform eigenvectors of T computed by `_stein`, back to eigenvectors of A.

For the names of the routines for packed and banded matrices, see [1].

There are several routines for computing eigenvalues and eigenvectors of T, to cover the cases of computing some or all of the eigenvalues, and some or all of the eigenvectors. In addition, some routines run faster in some computing environments or for some matrices than for others. Also, some routines are more accurate than other routines.

`_steqr` This routine uses the implicitly shifted QR algorithm. It switches between the QR and QL variants in order to handle graded matrices. This routine is used to compute all the eigenvalues and eigenvectors.

`_sterf` This routine uses a square-root free version of the QR algorithm, also switching between QR and QL variants, and can only compute all the eigenvalues. This routine is used to compute all the eigenvalues and no eigenvectors.

`_stedc` This routine uses Cuppen's divide and conquer algorithm to find the eigenvalues and the eigenvectors. `_stedc` can be many times faster than `_steqr` for large matrices but needs more work space ($2n^2$ or $3n^2$). This routine is used to compute all the eigenvalues and eigenvectors.

`_stegr` This routine uses the relatively robust representation (RRR) algorithm to find eigenvalues and eigenvectors. This routine uses an LDL^T factorization of a number of translates $T - \sigma I$ of T, for one shift σ near each cluster of eigenvalues. For each translate the algorithm computes very accurate eigenpairs for the tiny eigenvalues. `_stegr` is faster than all the other routines except in a few cases, and uses the least workspace.

`_stebz` This routine uses bisection to compute some or all of the eigenvalues. Options provide for computing all the eigenvalues in a real interval or all the eigenvalues from the ith to the jth largest. It can be highly accurate, but may be adjusted to run faster if lower accuracy is acceptable.

`_stein` Given accurate eigenvalues, this routine uses inverse iteration to compute some or all of the eigenvectors.

6.5 Generalized Symmetric Definite Eigenproblems (GSEP)

Drivers are provided to compute all the eigenvalues and (optionally) the eigenvectors of the following types of problems:

1. $A\mathbf{z} = \lambda B\mathbf{z}$
2. $AB\mathbf{z} = \lambda\mathbf{z}$
3. $BA\mathbf{z} = \lambda\mathbf{z}$

where A and B are symmetric or Hermitian and B is positive definite. For all these problems the eigenvalues λ are real. The matrices Z of computed eigenvectors satisfy $Z^T AZ = \Lambda$ (problem types 1 and 3) or $Z^{-1}AZ^{-T} = I$ (problem type 2), where Λ is a diagonal matrix with the eigenvalues on the diagonal. Z also satisfies $Z^T BZ = I$ (problem types 1 and 2) or $Z^T B^{-1}Z = I$ (problem type 3).

There are three types of driver routines for generalized symmetric and Hermitian eigenproblems. Originally LAPACK had just the simple and expert drivers described below, and the other one was added after an improved algorithm was discovered.

- a simple driver computes all the eigenvalues and (optionally) eigenvectors.
- an expert driver computes all or a selected subset of the eigenvalues and (optionally) eigenvectors. If few enough eigenvalues or eigenvectors are desired, the expert driver is faster than the simple driver.
- a divide-and-conquer driver solves the same problem as the simple driver. It is much faster than the simple driver for large matrices, but uses more workspace. The name divide-and-conquer refers to the underlying algorithm.

6.6 An example of a LAPACK routines

The double precision subroutine `dsytrd.f` implements the reduction to tridiagonal form. We give it here in full length.

```
      SUBROUTINE DSYTRD( UPLO, N, A, LDA, D, E, TAU, WORK, LWORK, INFO )
*
*  -- LAPACK routine (version 3.0) --
*     Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd.,
*     Courant Institute, Argonne National Lab, and Rice University
*     June 30, 1999
```

```
*
*     .. Scalar Arguments ..
      CHARACTER          UPLO
      INTEGER            INFO, LDA, LWORK, N
*     ..
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), D( * ), E( * ), TAU( * ),
     $                   WORK( * )
*     ..
*
*  Purpose
*  =======
*
*  DSYTRD reduces a real symmetric matrix A to real symmetric
*  tridiagonal form T by an orthogonal similarity transformation:
*  Q**T * A * Q = T.
*
*  Arguments
*  =========
*
*  UPLO    (input) CHARACTER*1
*          = 'U':  Upper triangle of A is stored;
*          = 'L':  Lower triangle of A is stored.
*
*  N       (input) INTEGER
*          The order of the matrix A.  N >= 0.
*
*  A       (input/output) DOUBLE PRECISION array, dimension (LDA,N)
*          On entry, the symmetric matrix A.  If UPLO = 'U', the leading
*          N-by-N upper triangular part of A contains the upper
*          triangular part of the matrix A, and the strictly lower
*          triangular part of A is not referenced.  If UPLO = 'L', the
*          leading N-by-N lower triangular part of A contains the lower
*          triangular part of the matrix A, and the strictly upper
*          triangular part of A is not referenced.
*          On exit, if UPLO = 'U', the diagonal and first superdiagonal
*          of A are overwritten by the corresponding elements of the
*          tridiagonal matrix T, and the elements above the first
*          superdiagonal, with the array TAU, represent the orthogonal
*          matrix Q as a product of elementary reflectors; if UPLO
*          = 'L', the diagonal and first subdiagonal of A are over-
*          written by the corresponding elements of the tridiagonal
*          matrix T, and the elements below the first subdiagonal, with
*          the array TAU, represent the orthogonal matrix Q as a product
*          of elementary reflectors. See Further Details.
*
*  LDA     (input) INTEGER
*          The leading dimension of the array A.  LDA >= max(1,N).
*
*  D       (output) DOUBLE PRECISION array, dimension (N)
*          The diagonal elements of the tridiagonal matrix T:
*          D(i) = A(i,i).
*
*  E       (output) DOUBLE PRECISION array, dimension (N-1)
*          The off-diagonal elements of the tridiagonal matrix T:
*          E(i) = A(i,i+1) if UPLO = 'U', E(i) = A(i+1,i) if UPLO = 'L'.
*
*  TAU     (output) DOUBLE PRECISION array, dimension (N-1)
*          The scalar factors of the elementary reflectors (see Further
*          Details).
```

```
*
*  WORK    (workspace/output) DOUBLE PRECISION array, dimension (LWORK)
*          On exit, if INFO = 0, WORK(1) returns the optimal LWORK.
*
*  LWORK   (input) INTEGER
*          The dimension of the array WORK.  LWORK >= 1.
*          For optimum performance LWORK >= N*NB, where NB is the
*          optimal blocksize.
*
*          If LWORK = -1, then a workspace query is assumed; the routine
*          only calculates the optimal size of the WORK array, returns
*          this value as the first entry of the WORK array, and no error
*          message related to LWORK is issued by XERBLA.
*
*  INFO    (output) INTEGER
*          = 0:  successful exit
*          < 0:  if INFO = -i, the i-th argument had an illegal value
*
*  Further Details
*  ===============
*
*  If UPLO = ’U’, the matrix Q is represented as a product of elementary
*  reflectors
*
*     Q = H(n-1) . . . H(2) H(1).
*
*  Each H(i) has the form
*
*     H(i) = I - tau * v * v’
*
*  where tau is a real scalar, and v is a real vector with
*  v(i+1:n) = 0 and v(i) = 1; v(1:i-1) is stored on exit in
*  A(1:i-1,i+1), and tau in TAU(i).
*
*  If UPLO = ’L’, the matrix Q is represented as a product of elementary
*  reflectors
*
*     Q = H(1) H(2) . . . H(n-1).
*
*  Each H(i) has the form
*
*     H(i) = I - tau * v * v’
*
*  where tau is a real scalar, and v is a real vector with
*  v(1:i) = 0 and v(i+1) = 1; v(i+2:n) is stored on exit in A(i+2:n,i),
*  and tau in TAU(i).
*
*  The contents of A on exit are illustrated by the following examples
*  with n = 5:
*
*  if UPLO = ’U’:                       if UPLO = ’L’:
*
*    (  d   e   v2  v3  v4 )              (  d                  )
*    (      d   e   v3  v4 )              (  e   d              )
*    (          d   e   v4 )              (  v1  e   d          )
*    (              d   e  )              (  v1  v2  e   d      )
*    (                  d  )              (  v1  v2  v3  e   d  )
*
*  where d and e denote diagonal and off-diagonal elements of T, and vi
*  denotes an element of the vector defining H(i).
```

```
*
*  =====================================================================
*
*     .. Parameters ..
      DOUBLE PRECISION   ONE
      PARAMETER          ( ONE = 1.0D+0 )
*     ..
*     .. Local Scalars ..
      LOGICAL            LQUERY, UPPER
      INTEGER            I, IINFO, IWS, J, KK, LDWORK, LWKOPT, NB,
     $                   NBMIN, NX
*     ..
*     .. External Subroutines ..
      EXTERNAL           DLATRD, DSYR2K, DSYTD2, XERBLA
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          MAX
*     ..
*     .. External Functions ..
      LOGICAL            LSAME
      INTEGER            ILAENV
      EXTERNAL           LSAME, ILAENV
*     ..
*     .. Executable Statements ..
*
*     Test the input parameters
*
      INFO = 0
      UPPER = LSAME( UPLO, 'U' )
      LQUERY = ( LWORK.EQ.-1 )
      IF( .NOT.UPPER .AND. .NOT.LSAME( UPLO, 'L' ) ) THEN
         INFO = -1
      ELSE IF( N.LT.0 ) THEN
         INFO = -2
      ELSE IF( LDA.LT.MAX( 1, N ) ) THEN
         INFO = -4
      ELSE IF( LWORK.LT.1 .AND. .NOT.LQUERY ) THEN
         INFO = -9
      END IF
*
      IF( INFO.EQ.0 ) THEN
*
*        Determine the block size.
*
         NB = ILAENV( 1, 'DSYTRD', UPLO, N, -1, -1, -1 )
         LWKOPT = N*NB
         WORK( 1 ) = LWKOPT
      END IF
*
      IF( INFO.NE.0 ) THEN
         CALL XERBLA( 'DSYTRD', -INFO )
         RETURN
      ELSE IF( LQUERY ) THEN
         RETURN
      END IF
*
*     Quick return if possible
*
      IF( N.EQ.0 ) THEN
         WORK( 1 ) = 1
         RETURN
      END IF
*
      NX = N
      IWS = 1
      IF( NB.GT.1 .AND. NB.LT.N ) THEN
*
*        Determine when to cross over from blocked to unblocked code
*        (last block is always handled by unblocked code).
*
         NX = MAX( NB, ILAENV( 3, 'DSYTRD', UPLO, N, -1, -1, -1 ) )
         IF( NX.LT.N ) THEN
*
*           Determine if workspace is large enough for blocked code.
*
            LDWORK = N
            IWS = LDWORK*NB
            IF( LWORK.LT.IWS ) THEN
*
*              Not enough workspace to use optimal NB:  determine the
*              minimum value of NB, and reduce NB or force use of
*              unblocked code by setting NX = N.
*
               NB = MAX( LWORK / LDWORK, 1 )
               NBMIN = ILAENV( 2, 'DSYTRD', UPLO, N, -1, -1, -1 )
               IF( NB.LT.NBMIN )
     $            NX = N
            END IF
         ELSE
            NX = N
         END IF
      ELSE
         NB = 1
      END IF
*
      IF( UPPER ) THEN
*
*        Reduce the upper triangle of A.
*        Columns 1:kk are handled by the unblocked method.
*
         KK = N - ( ( N-NX+NB-1 ) / NB )*NB
         DO 20 I = N - NB + 1, KK + 1, -NB
*
*           Reduce columns i:i+nb-1 to tridiagonal form and form the
*           matrix W which is needed to update the unreduced part of
*           the matrix
*
            CALL DLATRD( UPLO, I+NB-1, NB, A, LDA, E, TAU, WORK,
     $                   LDWORK )
*
*           Update the unreduced submatrix A(1:i-1,1:i-1), using an
*           update of the form:  A := A - V*W' - W*V'
*
            CALL DSYR2K( UPLO, 'No transpose', I-1, NB, -ONE, A( 1, I ),
     $                   LDA, WORK, LDWORK, ONE, A, LDA )
*
*           Copy superdiagonal elements back into A, and diagonal
*           elements into D
*
            DO 10 J = I, I + NB - 1
```

```
               A( J-1, J ) = E( J-1 )
               D( J ) = A( J, J )
   10       CONTINUE
   20    CONTINUE
*
*        Use unblocked code to reduce the last or only block
*
         CALL DSYTD2( UPLO, KK, A, LDA, D, E, TAU, IINFO )
      ELSE
*
*        Reduce the lower triangle of A
*
         DO 40 I = 1, N - NX, NB
*
*           Reduce columns i:i+nb-1 to tridiagonal form and form the
*           matrix W which is needed to update the unreduced part of
*           the matrix
*
            CALL DLATRD( UPLO, N-I+1, NB, A( I, I ), LDA, E( I ),
     $                   TAU( I ), WORK, LDWORK )
*
*           Update the unreduced submatrix A(i+ib:n,i+ib:n), using
*           an update of the form:  A := A - V*W' - W*V'
*
            CALL DSYR2K( UPLO, 'No transpose', N-I-NB+1, NB, -ONE,
     $                   A( I+NB, I ), LDA, WORK( NB+1 ), LDWORK, ONE,
     $                   A( I+NB, I+NB ), LDA )
*
*           Copy subdiagonal elements back into A, and diagonal
*           elements into D
*
            DO 30 J = I, I + NB - 1
               A( J+1, J ) = E( J )
               D( J ) = A( J, J )
   30       CONTINUE
   40    CONTINUE
*
*        Use unblocked code to reduce the last or only block
*
         CALL DSYTD2( UPLO, N-I+1, A( I, I ), LDA, D( I ), E( I ),
     $                TAU( I ), IINFO )
      END IF
*
      WORK( 1 ) = LWKOPT
      RETURN
*
*     End of DSYTRD
*
      END
```

Notice that most of the lines (indicated by '$*$') contain comments. The initial comment lines also serve as manual pages. Notice that the code only looks at one half (upper or lower triangle) of the symmetric input matrix. The other triangle is used to store the Householder vectors. These are normed such that the first component is one,

$$I - 2\mathbf{u}\mathbf{u}^* = I - 2|u_1|^2(\mathbf{u}/u_1)(\mathbf{u}/u_1)^* = I - \tau\mathbf{v}\mathbf{v}^*.$$

In the main loop of `dsytrd` there is a call to a subroutine `dlatrd` that generates a block reflektor. (The blocksize is NB.) Then the block reflector is applied by the routine dsyr2k.

Directly after the loop there is a call to the ‘unblocked dsytrd’ named dsytd2 to deal with the first/last few (<NB) rows/columns of the matrix. This excerpt concerns the situation when the upper triangle of the matrix A is stored. In that routine the mentioned loop looks very much the way we derived the formulae.

```
      ELSE
*
*        Reduce the lower triangle of A
*
         DO 20 I = 1, N - 1
*
*           Generate elementary reflector H(i) = I - tau * v * v'
*           to annihilate A(i+2:n,i)
*
            CALL DLARFG( N-I, A( I+1, I ), A( MIN( I+2, N ), I ), 1,
     $                   TAUI )
            E( I ) = A( I+1, I )
*
            IF( TAUI.NE.ZERO ) THEN
*
*              Apply H(i) from both sides to A(i+1:n,i+1:n)
*
               A( I+1, I ) = ONE
*
*              Compute  x := tau * A * v  storing y in TAU(i:n-1)
*
               CALL DSYMV( UPLO, N-I, TAUI, A( I+1, I+1 ), LDA,
     $                     A( I+1, I ), 1, ZERO, TAU( I ), 1 )
*
*              Compute  w := x - 1/2 * tau * (x'*v) * v
*
               ALPHA = -HALF*TAUI*DDOT( N-I, TAU( I ), 1, A( I+1, I ),
     $                 1 )
               CALL DAXPY( N-I, ALPHA, A( I+1, I ), 1, TAU( I ), 1 )
*
*              Apply the transformation as a rank-2 update:
*                 A := A - v * w' - w * v'
*
               CALL DSYR2( UPLO, N-I, -ONE, A( I+1, I ), 1, TAU( I ), 1,
     $                     A( I+1, I+1 ), LDA )
*
               A( I+1, I ) = E( I )
            END IF
            D( I ) = A( I, I )
            TAU( I ) = TAUI
   20    CONTINUE
         D( N ) = A( N, N )
      END IF
```

Bibliography

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide - Release 2.0*, SIAM, Philadelphia, PA, 1994. (Software and guide are available from Netlib at URL http://www.netlib.org/lapack/).

[2] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, 1997. (Software and guide are available at URL http://www.netlib.org/scalapack/).

[3] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.

[4] J. J. Dongarra, J. D. Croz, I. Duff, and S. Hammarling, *A proposal for a set of level 3 basic linear algebra subprograms*, ACM SIGNUM Newsletter, 22 (1987).

[5] ——, *A set of level 3 basic linear algebra subprograms*, ACM Trans. Math. Softw., 16 (1990), pp. 1–17.

[6] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson, *An extended set of fortran basic linear algebra subprograms*, ACM Trans. Math. Softw., 14 (1988), pp. 1–17.

[7] ——, *An extended set of fortran basic linear algebra subprograms: Model implementation and test programs*, ACM Transactions on Mathematical Software, 14 (1988), pp. 18–32.

[8] B. S. Garbow, J. M. Boyle, J. J. Dongarra, and C. B. Moler, *Matrix Eigensystem Routines – EISPACK Guide Extension*, Lecture Notes in Computer Science 51, Springer-Verlag, Berlin, 1977.

[9] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, *Basic linear algebra subprograms for Fortran usage*, ACM Trans. Math. Softw., 5 (1979), pp. 308–325.

[10] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines – EISPACK Guide*, Lecture Notes in Computer Science 6, Springer-Verlag, Berlin, 2nd ed., 1976.

Chapter 7

Vector iteration (power method)

7.1 Simple vector iteration

In this chapter we consider the simplest method to compute a single extremal eigenvalue, called *vector iteration* or *power method* [2, 5]. Let $A \in \mathbb{F}^{n\times n}$. Starting with an arbitrary initial vector $\mathbf{x}^{(0)} \in \mathbb{F}^n$ we form the vector sequence $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ by defining

$$\mathbf{x}^{(k)} := A\mathbf{x}^{(k-1)}, \qquad k = 1, 2, \ldots \tag{7.1}$$

Clearly,

$$\mathbf{x}^{(k)} := A^k\mathbf{x}^{(0)}. \tag{7.2}$$

The hope is that the $\mathbf{x}^{(k)}$ converge to an eigenvector associated with the eigenvalue of largest magnitude. As we are interested only in the direction but not in the length of the eigenvector, there is no need to normalize the iterates in (7.1), well at least *in theory*. In practice, $\mathbf{x}^{(k)}$ may either underflow (if $\|A\| < 1$) or overflow (if $\|A\| > 1$) for large k. Thus, one usually adds a normalization step to (7.1), leading to Algorithm 7.1.

Algorithm 7.1 Simple vector iteration (power method)

```
Choose a starting vector $\mathbf{x}^{(0)} \in \mathbb{F}^n$ with $\|\mathbf{x}^{(0)}\| = 1$.
$k = 0$.
repeat
    $k := k + 1$;
    $\mathbf{y}^{(k)} := A\mathbf{x}^{(k-1)}$;
    $\mu_k := \|\mathbf{y}^{(k)}\|$;
    $\mathbf{x}^{(k)} := \mathbf{y}^{(k)}/\mu_k$;
until a convergence criterion is satisfied
```

The vectors $\mathbf{x}^{(k)}$ generated by Algorithm 7.1 have all norm (length) one. That is, $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ is a sequence on the unit sphere in $\mathbb{F}^n$.

Let $A = XJY^*$ be the Jordan normal form of A with $Y^* := X^{-1}$, see Section 2.8. Then,

$$Y^*\mathbf{x}^{(k)} := JY^*\mathbf{x}^{(k-1)} \qquad \text{and} \qquad Y^*\mathbf{x}^{(k)} := J^kY^*\mathbf{x}^{(0)}, \tag{7.3}$$

respectively. If the sequence $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ converges to $\mathbf{x}_*$ then the sequence $\left\{\mathbf{y}^{(k)}\right\}_{k=0}^{\infty}$ with $\mathbf{y}^{(k)} = Y^*\mathbf{x}^{(k)}$ converges to $\mathbf{y}_* = Y^*\mathbf{x}_*$. By consequence, for the convergence analysis, we may assume without loss of generality (w.l.o.g.) that A is a Jordan block matrix.

7.2 Angles between vectors

Let $\mathbf{q}_1$ and $\mathbf{q}_2$ be unit vectors, cf. Fig. 7.1. The length of the orthogonal projection of $\mathbf{q}_2$ on $\operatorname{span}\{\mathbf{q}_1\}$ is given by

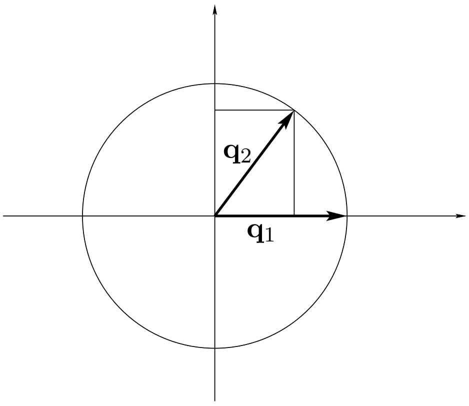


Figure 7.1: Angle between vectors $\mathbf{q}_1$ and $\mathbf{q}_2$

$$c := \|\mathbf{q}_1\mathbf{q}_1{}^*\mathbf{q}_2\| = |\mathbf{q}_1{}^*\mathbf{q}_2| \le 1. \tag{7.4}$$

The length of the orthogonal projection of $\mathbf{q}_2$ on $\operatorname{span}\{\mathbf{q}_1\}^\perp$ is

$$s := \|(\mathbf{I} - \mathbf{q}_1\mathbf{q}_1{}^*)\mathbf{q}_2\|. \tag{7.5}$$

As $\mathbf{q}_1\mathbf{q}_1^*$ is an orthogonal projection we have by Pythagoras' formula that

$$1 = \|\mathbf{q}_2\|^2 = \|\mathbf{q}_1\mathbf{q}_1{}^*\mathbf{q}_2\|^2 + \|(\mathbf{I} - \mathbf{q}_1\mathbf{q}_1{}^*)\mathbf{q}_2\|^2 = s^2 + c^2. \tag{7.6}$$

Alternatively, we can conclude from (7.5) that

$$\begin{array}{rcl} s^2 &=& \|(\mathbf{I} - \mathbf{q}_1\mathbf{q}_1{}^*)\mathbf{q}_2\|^2 \\ &=& \mathbf{q}_2{}^*(\mathbf{I} - \mathbf{q}_1\mathbf{q}_1{}^*)\mathbf{q}_2 \\ &=& \mathbf{q}_2{}^*\mathbf{q}_2 - (\mathbf{q}_2{}^*\mathbf{q}_1)(\mathbf{q}_1{}^*\mathbf{q}_2) \\ &=& 1 - c^2 \end{array} \tag{7.7}$$

So, there is a number, say, ϑ, $0 \le \vartheta \le \frac{\pi}{2}$, such that $c = \cos\vartheta$ and $s = \sin\vartheta$. We call this uniquely determined number ϑ the **angle** between the vectors $\mathbf{q}_1$ and $\mathbf{q}_2$:

$$\vartheta = \angle(\mathbf{q}_1, \mathbf{q}_2).$$

The generalization to arbitrary vectors is straightforward.

Definition 7.1 The **angle** θ between two nonzero vectors $\mathbf{x}$ and $\mathbf{y}$ is given by

$$\vartheta = \angle(\mathbf{x}, \mathbf{y}) = \arcsin\left(\left\|\left(I - \frac{\mathbf{x}\mathbf{x}^*}{\|\mathbf{x}\|^2}\right)\frac{\mathbf{y}}{\|\mathbf{y}\|}\right\|\right) = \arccos\left(\frac{|\mathbf{x}^*\mathbf{y}|}{\|\mathbf{x}\|\|\mathbf{y}\|}\right). \tag{7.8}$$

When investigating the convergence behaviour of eigensolvers we usually show that the angle between the approximating and the desired vector tends to zero as the number of iterations increases. In fact it is more convenient to work with the sine of the angle.

In the formulae above we used the projections P and $I-P$ with $P = \mathbf{q}_1\mathbf{q}_1{}^*$. We would have arrived at the same point if we had exchanged the roles of $\mathbf{q}_1$ and $\mathbf{q}_2$. As

$$\|\mathbf{q}_1\mathbf{q}_1^*\mathbf{q}_2\| = \|\mathbf{q}_2\mathbf{q}_2^*\mathbf{q}_1\| = |\mathbf{q}_2^*\mathbf{q}_1|$$

we get

$$\|(I - \mathbf{q}_1\mathbf{q}_1^*)\mathbf{q}_2\| = \|(I - \mathbf{q}_2\mathbf{q}_2^*)\mathbf{q}_1\|.$$

This immediately leads to

Lemma 7.2 $\sin\angle(\mathbf{q}_1, \mathbf{q}_2) = \|\mathbf{q}_1\mathbf{q}_1^* - \mathbf{q}_2\mathbf{q}_2^*\|.$

7.3 Convergence analysis

Let us now assume that A has Jordan block form,

$$A = \begin{bmatrix} \lambda_1 & \mathbf{0}^* \\ \mathbf{0} & J_2 \end{bmatrix} \tag{7.9}$$

with eigenvalues

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|. \tag{7.10}$$

Then, the eigenvector of A corresponding to its largest eigenvalue λ_1 is $\mathbf{e}_1$. We will now show that the iterates $\mathbf{x}^{(k)}$ converge to $\mathbf{e}_1$. More precisely, we will show that the angle $\angle(\mathbf{x}^{(k)}, \mathbf{e}_1)$ between $\mathbf{x}^{(k)}$ and $\mathbf{e}_1$ goes to zero with $k \to \infty$. Let

$$\mathbf{x}^{(k)} = \begin{pmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{pmatrix} =: \begin{pmatrix} x_1^{(k)} \\ \mathbf{x}_2^{(k)} \end{pmatrix}$$

with $\|\mathbf{x}^{(k)}\| = 1$. Then,

$$\sin\vartheta^{(k)} := \sin(\angle(\mathbf{x}^{(k)}, \mathbf{e}_1)) = \|(I - \mathbf{e}_1\mathbf{e}_1^*)\mathbf{x}^{(k)}\| = \|\mathbf{x}_2^{(k)}\| = \sqrt{\sum_{i=2}^n |x_i^{(k)}|^2}.$$

If we omit the normalization $\|\mathbf{x}^{(k)}\| = 1$, which we will do for convenience, then this becomes

$$\sin\vartheta^{(k)} := \sin(\angle(\mathbf{x}^{(k)}, \mathbf{e}_1)) = \frac{\|\mathbf{x}_2^{(k)}\|}{\|\mathbf{x}^{(k)}\|} = \sqrt{\frac{\sum_{i=2}^n |x_i^{(k)}|^2}{\sum_{i=1}^n |x_i^{(k)}|^2}}.$$

This means that for the convergence analysis we look at the iteration (7.1), while the actual implementation follows closely Algorithm 7.1.

From (7.1) we have

$$\mathbf{x}^{(k)} = \begin{pmatrix} x_1^{(k)} \\ \mathbf{x}_2^{(k)} \end{pmatrix} = \begin{bmatrix} \lambda_1 & \mathbf{0}^* \\ \mathbf{0} & J_2 \end{bmatrix} \begin{pmatrix} x_1^{(k-1)} \\ \mathbf{x}_2^{(k-1)} \end{pmatrix} = \begin{bmatrix} \lambda_1 & \mathbf{0}^* \\ \mathbf{0} & J_2 \end{bmatrix}^k \begin{pmatrix} x_1^{(0)} \\ \mathbf{x}_2^{(0)} \end{pmatrix}.$$

Defining

$$\mathbf{y}^{(k)} := \frac{1}{\lambda_1^k}\mathbf{x}^{(k)} \tag{7.11}$$

we have

$$\mathbf{y}^{(k)} = \begin{bmatrix} 1 & \mathbf{0}^* \\ \mathbf{0} & \frac{1}{\lambda_1}J_2 \end{bmatrix} \mathbf{y}^{(k-1)}.$$

Let us assume that $y_1^{(0)} = 1$. Then $y_1^{(k)} = 1$ for all k and

$$\mathbf{y}_2^{(k)} = \frac{1}{\lambda_1}J_2\mathbf{y}_2^{(k-1)}, \qquad \frac{1}{\lambda_1}J_2 = \begin{bmatrix} \mu_2 & * & & & \\ & \mu_3 & * & & \\ & & \ddots & \ddots & \\ & & & \mu_{n-1} & * \\ & & & & \mu_n \end{bmatrix}, \qquad |\mu_k| = \frac{|\lambda_k|}{|\lambda_1|} < 1.$$

For the sequel we need

Theorem 7.3 *Let* $|||\cdot|||$ *be* any *matrix norm. Then*

$$\lim_{k\to\infty} |||M^k|||^{1/k} = \rho(M) = \max_i |\lambda_i(M)|. \tag{7.12}$$

Proof. See Horn-Johnson [3], pp.297-299. ■

Definition 7.4 $\rho(M)$ in (7.12) is call **spectral radius** of M.

Regarding the convergence of the vector iteration, Theorem 7.3 implies that for *any* $\varepsilon > 0$ there is an integer $K(\varepsilon)$ such that

$$|||M^k|||^{1/k} \le \rho(M) + \varepsilon, \qquad \text{for all } k > K(\varepsilon). \tag{7.13}$$

We will apply this theorem to the case $M = \lambda_1^{-1}J_2$, the matrix norm $|||\cdot|||$ will be the ordinary 2-norm. Thus, for any $\varepsilon > 0$ there is a $K(\varepsilon) \in \mathbb{N}$ with

$$\left\| \left(\frac{1}{\lambda_1}J_2\right)^k \right\|^{1/k} \le |\mu_2| + \varepsilon, \quad \forall\, k > K(\varepsilon). \tag{7.14}$$

We can choose ε such that

$$|\mu_2| + \varepsilon < 1.$$

Then,

$$\sin(\angle(\mathbf{y}^{(k)}, \mathbf{e}_1)) = \frac{\|(I - \mathbf{e}_1\mathbf{e}_1^*)\mathbf{y}^{(k)}\|}{\|\mathbf{y}^{(k)}\|} = \frac{\|\mathbf{y}_2^{(k)}\|}{\|\mathbf{y}^{(k)}\|} = \frac{\|\mathbf{y}_2^{(k)}\|}{\sqrt{1 + \|\mathbf{y}_2^{(k)}\|}}$$
$$\le \|\mathbf{y}_2^{(k)}\| \le \|\frac{1}{\lambda_1^k}J_2^k\|\|\mathbf{y}_2^{(0)}\| \le (|\mu_2| + \varepsilon)^k\|\mathbf{y}_2^{(0)}\|.$$

Thus, the angle between $\mathbf{y}^{(k)}$ and $\mathbf{e}_1$ goes to zero with a rate $\mu_2 + \varepsilon$ for any positive ε. Since $\mathbf{x}^{(k)}$ is a scalar multiple of $\mathbf{y}^{(k)}$ the same holds for the angle between $\mathbf{x}^{(k)}$ and $\mathbf{e}_1$. Since we can choose ε arbitrarily small, we have proved that

$$\sin\vartheta^{(k)} = \sin(\angle(\mathbf{x}^{(k)}, \mathbf{u}_1)) \le c \cdot \left|\frac{\lambda_2}{\lambda_1}\right|^k$$

provided that $x_1^{(0)} = \mathbf{e}_1^* \mathbf{x}^{(0)} \neq 0$.

Returning to a general matrix $A \in \mathbb{F}^{n \times n}$ with Jordan normal form $A = XJY^*$, we employ equation (7.3). The sequence $\mathbf{y}^{(k)} = Y^* \mathbf{x}^{(k)}$ converges to $\mathbf{y}_* = \alpha \mathbf{e}_1$ with $\alpha \neq 0$. Therefore, $\mathbf{x}^{(k)}$ converges to a multiple of $X\mathbf{e}_1$, which is an eigenvector associated with the largest eigenvalue λ_1. The condition $\mathbf{e}_1^* \mathbf{y}^{(0)} \neq 0$ translates into

$$\mathbf{e}_1^*(Y^* \mathbf{x}^{(0)}) = (Y\mathbf{e}_1)^* \mathbf{x}^{(0)} \neq 0.$$

The first column of Y is a *left* eigenvector associated with λ_1. Therefore, we have proved

Theorem 7.5 *Let the eigenvalues of $A \in \mathbb{F}^{n \times n}$ be arranged such that $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$. Let $\mathbf{u}_1$ and $\mathbf{v}_1$ be right and left eigenvectors of A corresponding to λ_1, respectively. Then, the vector sequence generated by Algorithm 7.1 converges to $\mathbf{u}_1$ in the sense that*

$$\sin \vartheta^{(k)} = \sin(\angle(\mathbf{x}^{(k)}, \mathbf{u}_1)) \leq c \cdot \left| \frac{\lambda_2}{\lambda_1} \right|^k \tag{7.15}$$

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

Remark 7.1. The quantity μ_k in Algorithm 7.1 converges to $|\lambda_1|$. The true value $\lambda_1 \in \mathbb{C}$ can be found by comparing single components of $\mathbf{y}^{(k)}$ and $\mathbf{x}^{(k-1)}$. If $\lambda_1 \in \mathbb{R}$ then only the sign of λ_1 is at stake. □

Remark 7.2. The convergence of the vector iteration is faster the smaller the quotient $|\lambda_2|/|\lambda_1|$ is. □

Remark 7.3. From (7.12) we see that the norm of the powers of a matrix goes to zero if all is eigenvalues are smaller than one in modulus. For small powers the norm can initially grow considerably. In Fig. 7.2 we have plotted the norms of B^k with

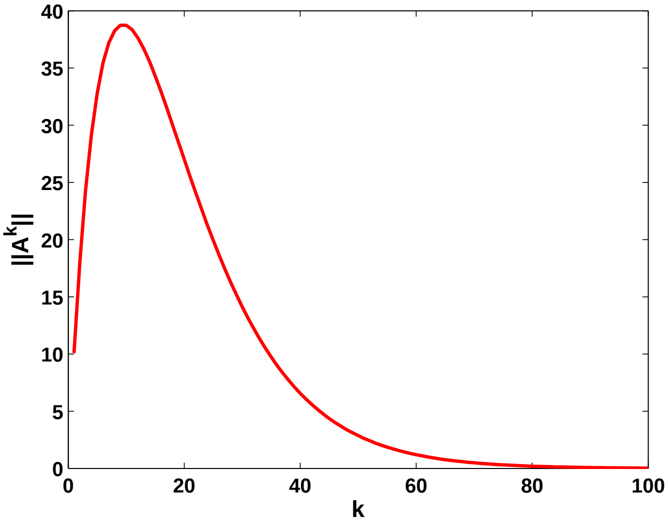


Figure 7.2: Norms of powers of B in (7.16).

$$B = \begin{bmatrix} 0.9 & 5 \\ 0 & 0.9 \end{bmatrix}. \tag{7.16}$$

□

Remark 7.4. If $\mathbf{v}_1^*\mathbf{x}^{(0)} = 0$ then the vector iteration converges to an eigenvector corresponding to the second largest eigenvalue. In practice, rounding errors usually prevent this behaviour of the algorithm. After a long initial phase the $\mathbf{x}^{(k)}$ turn to $\mathbf{u}_1$. □

Remark 7.5. In case that $\lambda_1 \neq \lambda_2$ but $|\lambda_1| = |\lambda_2|$ there may be no convergence at all. An example is

$$A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \qquad \mathbf{x}^{(0)} = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

□

7.4 A numerical example

In the following Matlab script we assume that A is upper triangular and that the largest eigenvalue (in modulus) is at position (1,1), i.e., $|a_{1,1}| > |a_{j,j}|$ for all $j > 1$.

```
%Demo  Simple Vector Iteration
%
n = 6;
randn('state',0);
A = diag([n:-1:1]) + triu(randn(n),1) % upper triangular matrix
x0 = rand(n,1); x0=x0/norm(x0)        % initial vector

e = eye(n,1); % Right eigenvector corresponding to largest
              % eigenvalue A(1,1)
% ----------------------------------------------------------

x=x0;  ang = norm(x - e*(e'*x))

hist = [ang,nan,nan];

if ~exist('tol'), tol = 100*eps; end
oldang = nan;

while ang > tol
    x = A*x;
    mue = norm(x);           % This is an approximation for the
    x = x/mue;               % searched eigenvalue

    ang = norm(x - e*(e'*x));
    hist = [hist; [mue,ang,ang/oldang]];
    oldang = ang;
end
```

Because the solution is known, we can compute the angle between iterate and true solution. We can even use this angle in the stopping criterion. The matrix A is given by

```
A =
    6.0000    1.1892   -0.5883   -0.0956   -0.6918   -0.3999
         0    5.0000    2.1832   -0.8323    0.8580    0.6900
         0         0    4.0000    0.2944    1.2540    0.8156
```

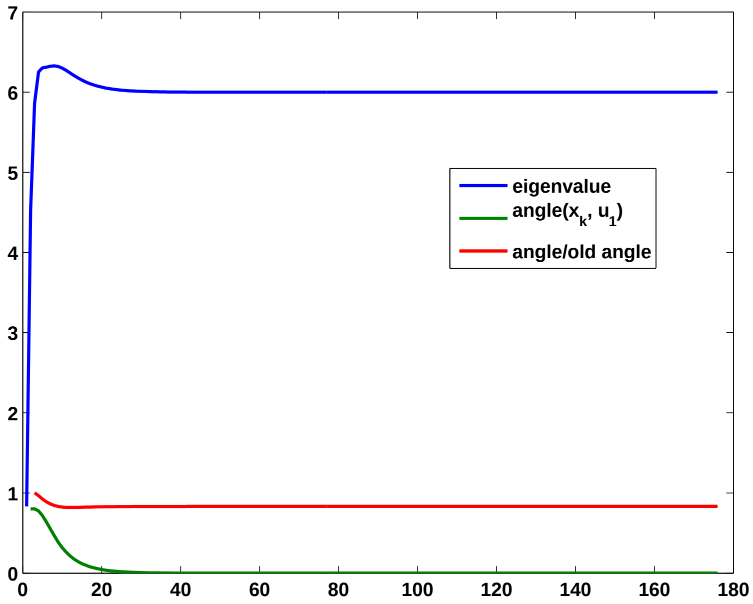


Figure 7.3: Plot of three important quantities: eigenvalue, angle between eigenvector approximation and exact eigenvector, convergence rate of eigenvector

```
         0         0         0    3.0000   -1.5937    0.7119
         0         0         0         0    2.0000    1.2902
         0         0         0         0         0    1.0000
```

The development of three important quantities is given in Fig. 7.3. In Fig. 7.4 the case is depicted when the initial vector is chosen orthogonal to the left eigenvector corresponding to $\lambda_1 = 6$. Initially, the approximated eigenvalue is 5. Because the stopping criterion does not hold, the iteration continues until eventually rounding errors take effect.

7.5 The symmetric case

Let us now consider the Hermitian/symmetric case. We again assume the now real eigenvalues to be arranged as in (7.10). But now the Schur decomposition of A becomes its spectral decomposition,

$$A = U\Lambda U^*, \qquad U = [\mathbf{u}_1, \ldots, \mathbf{u}_n], \quad \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n). \tag{7.17}$$

For the convergence analysis, we assume that A is diagonal, and that

$$\lambda_1 > \lambda_2 \geq \cdots \geq \lambda_n \geq 0. \tag{7.18}$$

Therefore, in (7.9) we have $J_2 = \operatorname{diag}(\lambda_2, \ldots, \lambda_n)$, i.e., all Jordan blocks are 1×1.

In contrast to the general case, in the Hermitian case we approximate the eigenvalue by the Rayleigh quotient of $\mathbf{x}^{(k)}$,

$$\lambda^{(k)} := {\mathbf{x}^{(k)}}^* A \mathbf{x}^{(k)}, \qquad \|\mathbf{x}^{(k)}\| = 1. \tag{7.19}$$

The symmetric algorithm is given in Algorithm 7.2.

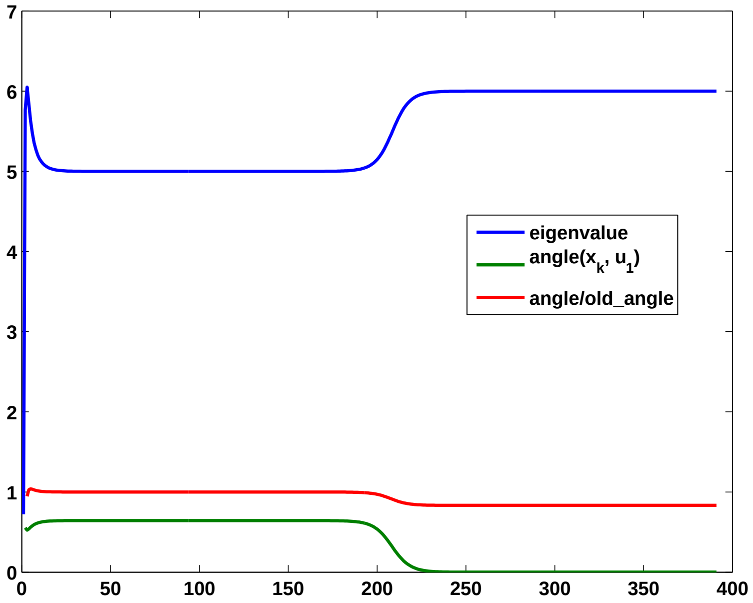


Figure 7.4: Plot of three important quantities: eigenvalue, angle between eigenvector approximation and exact eigenvector, convergence rate of eigenvector. Here, the initial vector is chosen orthogonal to the left eigenvector corresponding to the largest eigenvalue

Algorithm 7.2 Simple vector iteration for Hermitian matrices

```
Choose a starting vector $\mathbf{x}^{(0)} \in \mathbb{F}^n$ with $\|\mathbf{x}^{(0)}\| = 1$.
$\mathbf{y}^{(0)} := A\mathbf{x}^{(0)}$.
$\lambda^{(0)} := \mathbf{y}^{(0)*}\mathbf{x}^{(0)}$.
$k := 0$.
while $\|\mathbf{y}^{(k)} - \lambda^{(k)}\mathbf{x}^{(k)}\| >$ tol do
    $k := k + 1$;
    $\mathbf{x}^{(k)} := \mathbf{y}_{k-1}/\|\mathbf{y}_{k-1}\|$;
    $\mathbf{y}^{(k)} := A\mathbf{x}^{(k)}$;
    $\lambda^{(k)} := \mathbf{y}^{(k)*}\mathbf{x}^{(k)}$;
end while
```

In order to investigate the convergence of the Rayleigh quotient we work with auxiliary vectors

$$\mathbf{y}^{(k)} = \begin{pmatrix} 1 \\ \mathbf{y}_2^{(k)} \end{pmatrix} = \frac{1}{|x_1^{(k)}|}\mathbf{x}^{(k)}. \tag{7.20}$$

Notice, that any 'reasonable' approximation of the first eigenvector $\mathbf{e}_1$ has a nonzero first component. For the Rayleigh quotients we have

$$\rho(\mathbf{y}^{(k)}) = \rho(\mathbf{x}^{(k)}).$$

Now,

$$\lambda^{(k)} = \frac{\mathbf{y}^{(k)*}A\mathbf{y}^{(k)}}{\mathbf{y}^{(k)*}\mathbf{y}^{(k)}} = \frac{(\mathbf{e}_1 + \mathbf{y}_2^{(k)})^* A(\mathbf{e}_1 + \mathbf{y}_2^{(k)})}{1 + \|\mathbf{y}_2^{(k)}\|^2} = \frac{\lambda_1 + \mathbf{y}_2^{(k)*}A\mathbf{y}_2^{(k)}}{1 + \|\mathbf{y}_2^{(k)}\|^2} \tag{7.21}$$

where we used that $\mathbf{e}_1^*\mathbf{y}_2^{(k)} = 0$ and $\mathbf{e}_1^* A\mathbf{y}_2^{(k)} = 0$. Because,

$$\tan\vartheta^{(k)} := \tan(\angle(\mathbf{y}^{(k)}, \mathbf{e}_1)) = \|\mathbf{y}_2^{(k)}\|$$

and

$$1 + \tan^2(\phi) = \frac{1}{1 - \sin^2(\phi)}$$

we get from (7.21) that

$$\lambda^{(k)} = (\lambda_1 + {\mathbf{y}_2^{(k)}}^* A\mathbf{y}_2^{(k)})(1 - \sin^2\vartheta^{(k)}) = \lambda_1 - \lambda_1\sin^2\vartheta^{(k)} + {\mathbf{y}_2^{(k)}}^* A\mathbf{y}_2^{(k)}\cos^2\vartheta^{(k)}. \tag{7.22}$$

Now, since $\lambda_1 > 0$,

$$\begin{aligned} 0 \le \lambda_1 - \lambda^{(k)} &= \lambda_1\sin^2\vartheta^{(k)} - {\mathbf{y}_2^{(k)}}^* A\mathbf{y}_2^{(k)}\cos^2\vartheta^{(k)} \\ &\le \lambda_1\sin^2\vartheta^{(k)} - \lambda_n\|\mathbf{y}_2^{(k)}\|^2\cos^2\vartheta^{(k)} = (\lambda_1 - \lambda_n)\sin^2\vartheta^{(k)}. \end{aligned} \tag{7.23}$$

In summary, we have proved

Theorem 7.6 *Let A be a symmetric matrix with spectral decomposition (7.17)–(7.18). Then, the simple vector iteration of Algorithm 7.2 computes sequences $\left\{\lambda^{(k)}\right\}_{k=0}^{\infty}$ and $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ that converge linearly towards the largest eigenvalue λ_1 of A and the corresponding eigenvector $\mathbf{u}_1$ provided that the initial vector $\mathbf{x}^{(0)}$ has a nonzero component in the direction of $\mathbf{u}_1$, i.e., that $\mathbf{u}_1^*\mathbf{x}^{(0)} \neq 0$. The convergence rates are given by*

$$\sin\vartheta^{(k)} \le \left|\frac{\lambda_2}{\lambda_1}\right|^k \sin\vartheta^{(0)}, \qquad |\lambda_1 - \lambda^{(k)}| \le (\lambda_1 - \lambda_n)\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\sin^2\vartheta^{(0)}.$$

where $\vartheta^{(k)} = \angle(\mathbf{x}^{(k)}, \mathbf{u}_1$. ■

Thus, the speed of convergence is determined by the ratio of the two eigenvalues largest in modulus and the quality of the initial guess $\mathbf{x}^{(0)}$. Both sequences $\left\{\lambda^{(k)}\right\}$ and $\left\{\mathbf{x}^{(k)}\right\}$ converge linearly, but the decisive ratio appears squared in the bound for the approximation error in the eigenvalue. $\lambda_1 - \lambda_n$ is called the **spread** of the spectrum of A. Its occurance in the bound for $\lambda_{\max} - \lambda^{(k)}$ shows that a simple scaling of the matrix does not affect the convergence behavior of the algorithm.

Example 7.7 Let's compute the *smallest* eigenvalue and corresponding eigenvector of the one-dimensional Poisson matrix $T = T_n$ of Example 2.7 with $n = 40$. Let us assume that we know an upper bound τ for the largest eigenvalue λ_n of T then the transformed matrix $\tau I - T$ has the same eigenvectors as T and eigenvalues $\tau - \lambda_n < \tau - \lambda_{n-1} < \cdots < \tau - \lambda_1$. So, we apply vector iteration to compute the desired quantities.

We set $\tau = 4(n+1)^2/\pi^2$ a number that is easily obtained by applying Gerschgorin's circle theorem. We performed a MATLAB experiment starting with a random vector.

```
>> n=40;
>> T = (4*((n+1)^2/pi^2))*eye(n) - ((n+1)^2/pi^2)*p_1d(n);
>> rand('state',0); x0=rand(n,1);
>> [x,lam,nit]=vit(T,x0,1e-4);
>> tau-lam
ans =
    0.9995
>> nit
nit =
        1968
```

In as many as 1968 iteration steps we arrived at an eigenvalue approximation 0.9995. This number is correct to all digits. The difference to the eigenvalue 1 of the continuous eigenvalue problem $-u''(x) = \lambda u(x)$ is due to the discretization error. Figure 7.5 shows the convergence history of this calculation. The straight lines show the actual angle $\vartheta^{(k)}$ between $\mathbf{x}^{(k)}$ and $\mathbf{u}_1$ (above) and the actual error $\lambda^{(k)} - \lambda_1$. These quantities can of course not be computed in general. In this example we know them, see Ex. 2.7. The dotted lines show powers of $q = (\tau - \lambda_2)/(\tau - \lambda_1)$ that indicate the convergence rates given by Theorem 7.6. Here, $q = 0.9956$. Clearly, the convergence is as predicted.

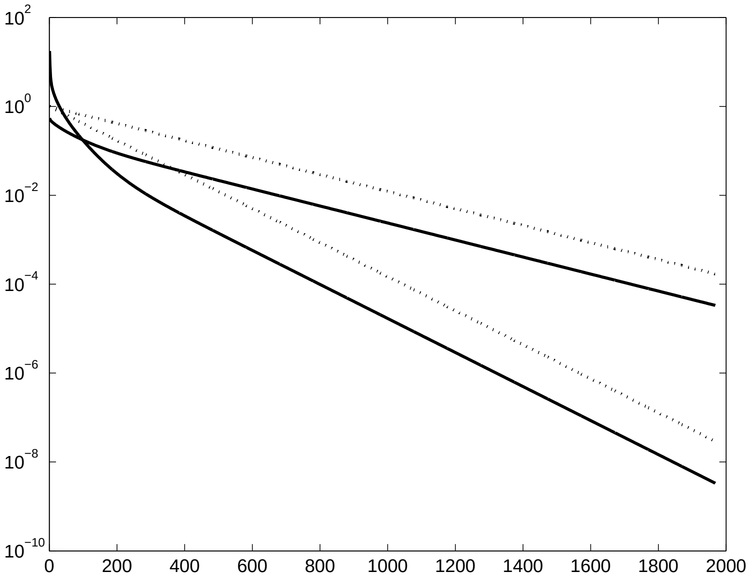


Figure 7.5: Simple vector iteration with $\tau I_{40} - T_{40}$

Example 7.8 We mentioned that a good initial vector can reduce the number of iteration steps. Remember that the smallest eigenfunction is $\sin x$, a function that is positive on the whole interval $(0, \pi)$. Let us therefore set $\mathbf{x}^{(0)}$ to be the vector of all ones.

```
>> x0 = ones(n,1);
>> [x,lam,nit]=vit(T,x0,1e-4);
>> nit
nit =
   866
```

This is a surprisingly high reduction in the number of iteration steps. Figure 7.6 shows the convergence history of this calculation. Again the doted lines indicate the convergence rates that are to be expected. The actual convergence rates are evidently much better. How can that be?

The eigenvectors of T_n resemble the eigenfunctions $\sin kx$ of the continuous eigenvalue problem. Therefore the coefficients corresponding to eigenvectors corresponding to eigenfunctions antisymmetric with respect to the point $\pi/2$ vanish. In particular $x_2 = 0$. Therefore the convergence rate is not $q = (\tau - \lambda_2)/(\tau - \lambda_1)$ but $\hat{q} = (\tau - \lambda_3)/(\tau - \lambda_1)$. This is verified by the numbers given in Fig. 7.7 where the assymptotic corrected convergence rates $\hat{q}$ and $\hat{q}^2$ are indicated.

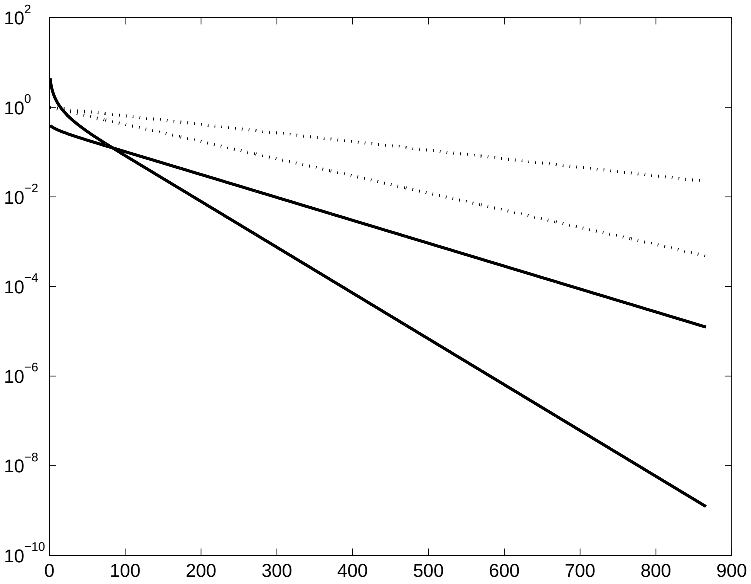


Figure 7.6: Simple vector iteration with $\tau I_{40} - T_{40}$ and starting vector $(1, 1, \ldots, 1)^T$

Problem 7.9 When computing the smallest eigenvalue of T_n by the simple vector iteration we can find a better shift than τ above if the extremal points of the spectrum are known. Determine σ such that $\sigma I_n - T_n$ exhibits the optimal convergence rate. Hint: On the one hand we would like the quotient $(\sigma - \lambda_{n-1})/(\sigma - \lambda_n)$ to be as small as possible. On the other hand $|\sigma - \lambda_1|/(\sigma - \lambda_n)$ must not become to big. Hint: Equate the two quantities.

7.6 Inverse vector iteration

The previous examples have shown that the convergence of simple vector iteration is potentially very slow. The quotient of the second largest to the largest eigenvalue are very close to 1. We noticed this by means of a very simple and small eigenvalue problem. The situation gets much worse if the problems are big.

We have seen in (2.28) that a polynomial in A has the same eigenvectors as A. We therefore may try to find a polynomial that enhances the eigenvalue that we are looking for. This approach is however not successful in the most critical case when the wanted eigenvalue is very close to unwanted. In this situation, the **shift-and-invert** spectral transformation is most appropriate. Instead of a polynomial we transform the matrix by the rational function $f(\lambda) = 1/(\lambda - \sigma)$ where σ is a so-called **shift** that is chosen close to the desired eigenvalue. Simple vector iteration with the matrix $(A - \sigma I)^{-1}$ is referred to as **inverse vector iteration**, see Algorithm 7.6.

$$(7.24) \qquad \mathbf{x}^{(k)} := (A - \sigma I)^{-1}\mathbf{x}^{(k-1)} \quad \Longleftrightarrow \quad (A - \sigma I)\,\mathbf{x}^{(k)} := \mathbf{x}^{(k-1)}, \qquad k = 1, 2, \ldots$$

The iteration converges towards the eigenvector with eigenvalue closest to σ. A linear system of equations has to be solved in each iteration step. Of course only one Cholesky or LU factorization has to be computed as the shift remains constants in all iterations. The stopping criterion is changed into

$$(7.25) \qquad \|\mathbf{x}^{(k)} - \mathbf{y}^{(k)}/\mu^{(k)}\| \le \text{tol}\|\mathbf{y}^{(k)}\|$$

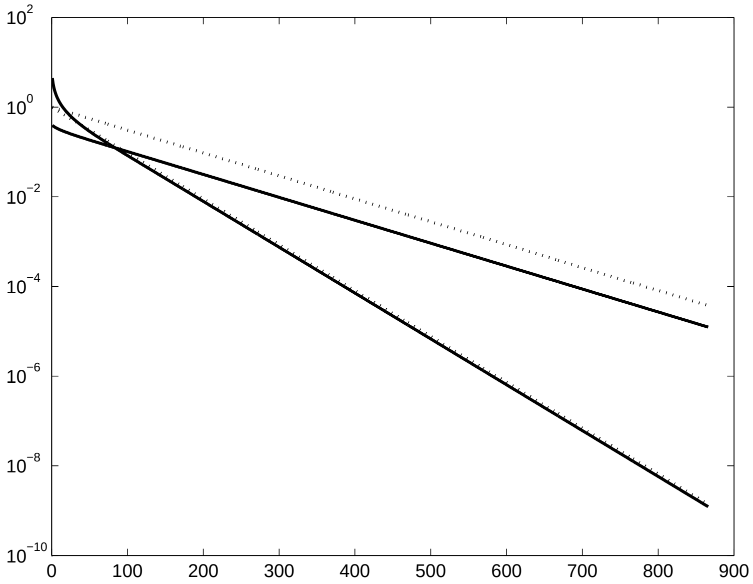


Figure 7.7: Simple vector iteration with $\tau I_{40} - T_{40}$ and starting vector $(1, 1, \ldots, 1)^T$

Algorithm 7.3 Inverse vector iteration

```
1: Choose a starting vector $\mathbf{x}_0 \in \mathbb{F}^n$ and a shift $\sigma$.
2: Compute the LU factorization of $A - \sigma I$: $LU = P(A - \sigma I)$
3: $\mathbf{y}^{(0)} := U^{-1}L^{-1}P\mathbf{x}^{(0)}$. $\mu^{(0)} = {\mathbf{y}^{(0)}}^*\mathbf{x}^{(0)}$,   $\lambda^{(0)} := \sigma + 1/\mu^{(0)}$.   $k := 0$.
4: while $\|\mathbf{x}^{(k)} - \mathbf{y}^{(k)}/\mu^{(k)}\| > \mathrm{tol}\|\mathbf{y}^{(k)}\|$ do
5:     $k := k + 1$.
6:     $\mathbf{x}^{(k)} := \mathbf{y}_{k-1}/\|\mathbf{y}_{k-1}\|$.
7:     $\mathbf{y}^{(k)} := U^{-1}L^{-1}P\mathbf{x}^{(k)}$.
8:     $\mu^{(k)} := {\mathbf{y}^{(k)}}^*\mathbf{x}^{(k)}$,   $\lambda^{(k)} := \sigma + 1/\mu^{(k)}$.
9: end while
```

where we have used

$$A\mathbf{y}^{(k)} - \lambda^{(k)}\mathbf{y}^{(k)} = A\mathbf{y}^{(k)} - \left(\sigma - \frac{1}{\mu^{(k)}}\right)\mathbf{y}^{(k)} = \mathbf{x}^{(k)} - \mathbf{y}^{(k)}/\mu^{(k)}$$

The convergence result of Theorem 7.6 can easily be adapted to the new situation if it is taken into account that $A - \sigma I$ has eigenpairs $(\mu_i, \mathbf{u}_i)$ with $\mu_i = 1/(\sigma - \lambda_i)$.

Theorem 7.10 *Let A be symmetric positive definite with spectral decomposition (7.17). Let $\lambda_1', \ldots, \lambda_n'$ be a renumeration of the eigenvalues in (7.17) such that*

$$\frac{1}{|\lambda_1' - \sigma|} > \frac{1}{|\lambda_2' - \sigma|} \geq \cdots \geq \frac{1}{|\lambda_n' - \sigma|} \tag{7.26}$$

Then, provided that ${\mathbf{u}_1'}^*\mathbf{x}^{(0)} \neq 0$, the inverse vector iteration of Algorithm 7.6 constructs sequences $\left\{\lambda^{(k)}\right\}_{k=0}^{\infty}$ and $\left\{\mathbf{x}^{(k)}\right\}_{k=0}^{\infty}$ that converge linearly towards that eigenvalue λ_1' closest to the shift σ and to the corresponding eigenvector $\mathbf{u}_1'$, respectively. The bounds

$$\sin\vartheta^{(k)} \leq \left|\frac{\lambda_1' - \sigma}{\lambda_2' - \sigma}\right|^k \sin\vartheta^{(0)}, \qquad \lambda^{(k)} - \lambda_1 \leq \delta \left|\frac{\lambda_1' - \sigma}{\lambda_2' - \sigma}\right|^{2k} \sin^2\vartheta^{(0)}.$$

hold with $\vartheta^{(k)} = \angle(\mathbf{x}^{(k)}, \mathbf{u}_1)$ and $\delta = \mathrm{spread}(\sigma((A - \sigma I)^{-1}))$. ■

If the shift σ approximates very well an eigenvalue of A then $\dfrac{\lambda^{(k)}-\sigma}{\lambda_n-\sigma} \ll 1$ and convergence is very rapid.

Example 7.11 Let us now use inverse iteration to compute the smallest eigenvalue and corresponding eigenvector of the one-dimensional Poisson matrix $T = T_n$ of Example 2.7 with $n = 40$. If we assume that we know that the smallest eigenvalue λ_1 is around 1 then a shift $\sigma = .9$ is reasonable, if we want $A - \sigma I$ to still be positive definite. Starting with the vector of all ones three iteration steps suffice to get the desired accuracy of $\mathrm{tol} = 10^{-5}$, see Table 7.1.

k	$\lambda^{(k)} - \lambda_1$	$\sin(\vartheta^{(k)})$
1	2.0188e-02	4.1954e-03
2	1.7306e-06	5.0727e-05
3	2.5289e-10	6.2492e-07

Table 7.1: Computing the lowest eigenvalue of the one-dimensinal Poisson equation by inverse iteration

Example 7.12 We consider the problem of computing the eigenvector corresponding to a *known* eigenvalue. The matrix that we consider is one of the so-called Wilkinson matrices

$$T = \begin{bmatrix} 19 & -1 & & & & & & \\ -1 & 18 & -1 & & & & & \\ & \ddots & \ddots & \ddots & & & & \\ & & -1 & 1 & -1 & & & \\ & & & -1 & 1 & -1 & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & -1 & 19 & -1 \\ & & & & & & -1 & 20 \end{bmatrix}.$$

Wilkinson matrices are irreducible tridiagonal matrices that have very close eigenvalues. This matrix has the eigenvalues

i	λ_i	i	λ_i
1	-1.1254415221199814	11	5.0002362656192743
2	0.2538058170966502	12	5.9999918413270530
3	0.9475343675285830	13	6.0000083521880692
4	1.7893213526669509	14	6.9999997949295611
5	2.1302092192694015	15	7.0000002079042920
6	2.9610588806935558	16	7.9999999961918720
7	3.0430992883895192	17	8.0000000038418246
8	3.9960479973346419	18	8.9999999999455120
9	4.0043538173235769	19	9.0000000000548166
10	4.9997743198148310	20	9.9999999999996234

The following MATLAB code constructs the SPARSE tridiagonal matrix T.

```
n = 40;
e = ones(n,1); f = abs([-n/2+1:n/2]');
T = spdiags([-e f -e], [-1:1], n, n);
lam = sort(eig(T));
```

Computing the 20-th and 21-st eigenvectors could be done in the following way.

```
>> x = (T - lam(20)*eye(n))\e;
>> y = (T - lam(21)*eye(n))\e;
>> x = x/norm(x);  y = y/norm(y);
>> x'*y
ans =
   0.00140329005834
>> norm((T - lam(20)*eye(n))*x)
ans =
     7.325760095786749e-15
>> norm((T - lam(21)*eye(n))*y)
ans =
     7.120036319503636e-15
```

The computed vectors **x** and **y** are good approximations in the sense that they give small residuals. However, the two vectors are nor mutually orthogonal at all. We try to improve orthogonality by apply a second step of inverse iteration

```
>> x = (T - lam(20)*eye(n))\x;
>> y = (T - lam(21)*eye(n))\y;
>> x = x/norm(x);  y = y/norm(y);
>> x'*y
ans =
    -1.313592004487587e-05
```

Things have only slightly improved. Therefore, we orthogonalize **y** *explicitely* against **x**.

```
>> y = y - x*(x'*y);
>> x'*y
ans =
    -2.155571068436496e-17
>> norm((T - lam(21)*eye(n))*y)
ans =
     1.557058217172078e-15
>> norm((T - lam(20)*eye(n))*x)
ans =
     4.117116818055497e-16
```

This helped. The two eigenvectors are now perpendicular on each other, and the residuals are still fine.

Discussion of inverse iteration

We have seen that

- we can compute eigenvectors corresponding to any (simple and well separated) eigenvalue if we choose the shift properly, and that

- we have very good convergence rates, is the shift is close to an eigenvalue.

However, one may feel uncomfortable solving an 'almost singular' system of equations, after all $\sigma \approx \lambda_k$ means that the condition of $A - \sigma I$ is very big. From the analysis of linear systems of equations we know that this means large errors in the solution. Furtunately, the error that is suffered from when solving with $A - \sigma I$ points in the right direction. To see this, assume that the singular value decomposition of $A - \sigma I$ is given by

$$A - \sigma I = U\Sigma V^*, \qquad \Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n), \quad \text{with } \sigma_1 \geq \cdots \geq \sigma_n \geq 0.$$

If $A - \sigma I$ is 'almost' singular then $\sigma_n \ll 1$. If even $\sigma_n = 0$ then $(A - \sigma I)\mathbf{v}_n = \mathbf{0}$, i.e., the last right singular vector is an eigenvector of A corresponding to the eigenvalue σ (the shift).

If $\sigma_n = \mathcal{O}(\varepsilon)$ then

$$(A - \sigma I)\mathbf{z} = U\Sigma V^*\mathbf{z} = \mathbf{y}.$$

Thus,

$$\mathbf{z} = V\Sigma^{-1}U^*\mathbf{y} = \sum_{i=1}^{n} \mathbf{v}_i \frac{\mathbf{u}_i^*\mathbf{y}}{\sigma_i} \overset{\sigma_n \ll \sigma_{n-1}}{\approx} \mathbf{v}_n \frac{\mathbf{u}_n^*\mathbf{y}}{\sigma_n}.$$

The tiny σ_n blows up the component in direction of $\mathbf{v}_n$. So, the vector $\mathbf{z}$ points in the desired 'most singular' direction.

7.7 The generalized eigenvalue problem

Applying the vector iteration (7.1) to the generalized eigenvalue problem $A\mathbf{x} = \lambda B\mathbf{x}$ leads to the iteration

$$\mathbf{x}^{(k)} := B^{-1}A\mathbf{x}^{(k-1)}, \qquad k = 1, 2, \ldots$$

Since the solution of a linear system is required in each iteration step, we can execute an inverse iteration right-away,

$$(A - \sigma B)\,\mathbf{x}^{(k)} := B\mathbf{x}^{(k-1)}, \qquad k = 1, 2, \ldots \tag{7.27}$$

The iteration performs an ordinary vector iteration for the eigenvalue problem

$$(A - \sigma B)^{-1}B\mathbf{x} := \mu\mathbf{x}, \qquad \mu = \frac{1}{\lambda - \sigma}. \tag{7.28}$$

Thus, the iteration (7.27) converges to the largest eigenvector of (7.28), i.e., the eigenvector with eigenvalue closest to the shift σ.

7.8 Computing higher eigenvalues

In order to compute higher eigenvalues $\lambda_2, \lambda_3, \ldots$, we make use of the mutual orthogonality of the eigenvectors of symmetric matrices, see Theorem 2.14. (In the case of Schur vectors we can proceed in a similar way.)

So, in order to be able to compute the second eigenpair $(\lambda_2, \mathbf{u}_2)$ we have to know the eigenvector $\mathbf{u}_1$ corresponding to the lowest eigenvalue. Most probably is has been computed previously. If this is the case we can execute an inverse iteration orthogonal to $\mathbf{u}_1$.

More generally, we can compute the j-th eigenpair $(\lambda_j, \mathbf{u}_j)$ by inverse iteration, keeping the iterated vector $\mathbf{x}^{(k)}$ orthogonal to the already known or computed eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}$.

Algorithm 7.4 Inverse vector iteration for computing $(\lambda_j, \mathbf{u}_j)$

```
1: The LU factorization of $A - \sigma I$: $LU = P(A - \sigma I)$
   and the eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}$ are known.
2: Choose a starting vector $\mathbf{x}^{(0)}$ such that $\mathbf{u}_q^*\mathbf{x}^{(0)} = 0$, $q < j$.
3: Set $k := 0$.
4: while $\|\mathbf{x}^{(k)} - \mathbf{y}^{(k)}/\mu^{(k)}\| > \mathrm{tol}\|\mathbf{y}^{(k)}\|$ do
5:    $k := k + 1$;
6:    $\mathbf{x}^{(k)} := \mathbf{y}^{(k-1)}/\|\mathbf{y}^{(k-1)}\|$;
7:    $\mathbf{y}^{(k)} := U^{-1}L^{-1}P\mathbf{x}^{(k)}$;
8:    $\mu^{(k)} := \mathbf{y}^{(k)*}\mathbf{x}^{(k)}$,    $\lambda^{(k)} := \sigma + 1/\mu^{(k)}$.
9: end while
```

In exact arithmetic, the condition $\mathbf{u}_1^*\mathbf{x}^{(0)} = \cdots = \mathbf{u}_{j-1}^*\mathbf{x}^{(0)} = 0$ implies that all $\mathbf{x}^{(k)}$ are orthogonal to $\mathbf{u}_1, \ldots, \mathbf{u}_{j-1}$. In general, however, one has to expect rounding errors that introduce components in the directions of already computed eigenvectors. Therefore, it is *necessary* to enforce the orthogonality conditions during the iteration.

Assuming exact arithmetic, Theorem 7.10 immediately implies that

$$\sin\angle(\mathbf{x}^{(k)}, \mathbf{x}_j) \le c_1 \left(\frac{\lambda_j}{\lambda_{j'}}\right)^k$$
$$|\lambda^{(k)} - \lambda_j| \le c_2 \left(\frac{\lambda_j}{\lambda_{j'}}\right)^{2k}$$

where j' is the smallest index for which $\lambda_{j'} > \lambda_j$.

7.9 Rayleigh quotient iteration

We now assume that the matrix the eigenpairs of which we want to determine is Hermitian (or symmetric).

We have noticed that inverse iteration is an effective way to compute eigenpairs, if a good approximation of the desired eigenvalue is known. This approximation is used as a shift. However, as we have seen earlier, if a good approximation of an eigenvector is available its Rayleigh quotient gives a very good approximation of its eigenvalue.

Indeed we have the following

Lemma 7.13 *Let* $\mathbf{q}$ *be any nonzero vector. The number* ρ *that minimizes* $\|A\mathbf{q} - \rho\mathbf{q}\|$ *is the* Rayleigh quotient

$$\rho = \frac{\mathbf{q}^*A\mathbf{q}}{\mathbf{q}^*\mathbf{q}}. \tag{7.29}$$

Proof. Let $\rho \in \mathbb{R}$ be the Rayleigh quotient (7.29) of $\mathbf{q} \neq 0$ and let $\tau \in \mathbb{C}$ be any number. Then we have

$$\begin{aligned}
\|A\mathbf{q} - (\rho + \tau)\mathbf{q}\|^2 &= \mathbf{q}^*A^2\mathbf{q} - (2\rho + \tau + \bar{\tau})\,\mathbf{q}^*A\mathbf{q} + |\rho + \tau|^2\mathbf{q}^*\mathbf{q} \\
&= \mathbf{q}^*A^2\mathbf{q} - 2\rho\,\mathbf{q}^*A\mathbf{q} - 2\mathrm{Re}(\tau)\,\mathbf{q}^*A\mathbf{q} + \rho^2\,\mathbf{q}^*\mathbf{q} + 2\rho\mathrm{Re}(\tau)\,\mathbf{q}^*\mathbf{q} + |\tau|^2\mathbf{q}^*\mathbf{q} \\
&= \mathbf{q}^*A^2\mathbf{q} - \frac{(\mathbf{q}^*A\mathbf{q})^2}{\mathbf{q}^*\mathbf{q}} + |\tau|^2\,\mathbf{q}^*\mathbf{q}.
\end{aligned}$$

The last term is smallest if $\tau = 0$. ■

Algorithm 7.5 Rayleigh quotient iteration (RQI)

```
1: Choose a starting vector $\mathbf{y}_0 \in \mathbb{F}^n$ with $\|\mathbf{y}_0\| = 1$ and a tolerance $\varepsilon$.
2: for $k = 1, 2, \dots$ do
3:   $\rho^{(k)} := \mathbf{y}^{(k-1)^*} A \mathbf{y}^{(k-1)}$.
4:   Solve $(A - \rho^{(k)} I)\mathbf{z}^{(k)} = \mathbf{y}^{(k-1)}$ for $\mathbf{z}^{(k)}$.
5:   $\sigma^{(k)} = \|\mathbf{z}^{(k)}\|$.
6:   $\mathbf{y}^{(k)} := \mathbf{z}^{(k)}/\sigma^{(k)}$.
7:   if $\sigma^{(k)} > 10/\varepsilon$ then
8:     return $\{\mathbf{y}^{(k)}\}$
9:   end if
10: end for
```

The following algorithm 7.9 is a modification of inverse iteration. In each iteration step the shift is modified to be the Rayleigh quotient of the most recent eigenvector approximation. This is not a curse but a blessing [4] as we have seen in section 7.6.

In step 4 of this algorithm a close to singular system of equation is solved. This results in a very long solution whose norm is used a the convergence criterion.

The Rayleigh quotient iteration usually converges, however not always towards the desired solution. Therefore, to investigate the convergence rate we make the following

Assumption: $\mathbf{y}^{(k)} \xrightarrow[k\to\infty]{} \mathbf{x}$ with $A\mathbf{x} = \lambda\mathbf{x}$.

This assumption garantees that there is at all convergence towards a certain eigenvector $\mathbf{x}$. Let $\|\mathbf{x}\| = \|\mathbf{y}^{(k)}\| = 1$ and let the angle between this eigenvector and its approximation be $\varphi^{(k)} = \angle(\mathbf{x}, \mathbf{y}^{(k)})$. Then the assumption implies that $\{\varphi^{(k)}\}_{k=1}^{\infty}$ converges to zero. We can write

$$\mathbf{y}^{(k)} = \mathbf{x}\cos\varphi^{(k)} + \mathbf{u}^{(k)}\sin\varphi^{(k)}, \qquad \|\mathbf{x}\| = \|\mathbf{y}^{(k)}\| = \|\mathbf{u}^{(k)}\| = 1.$$

Let

$$\rho^{(k)} = \rho(\mathbf{y}^{(k)}) = \frac{\mathbf{y}^{(k)^*} A \mathbf{y}^{(k)}}{\mathbf{y}^{(k)^*}\mathbf{y}^{(k)}} = \mathbf{y}^{(k)^*} A \mathbf{y}^{(k)}$$

be the Rayleigh quotient of $\mathbf{y}_k$. Then we have

$$\begin{aligned}\lambda - \rho_k &= \lambda - \cos^2\varphi_k \underbrace{\mathbf{x}^* A\mathbf{x}}_{\lambda} - \cos\varphi_k \sin\varphi_k \underbrace{\mathbf{x}^* A\mathbf{u}_k}_{0} - \sin^2\varphi_k \mathbf{u}_k^* A\mathbf{u}_k \\ &= \lambda(1 - \cos^2\varphi_k) - \sin^2\varphi_k \rho(\mathbf{u}_k) \\ &= (\lambda - \rho(\mathbf{u}_k))\sin^2\varphi_k.\end{aligned}$$

We now prove the

Theorem 7.14 (Local convergence of Rayleigh quotient iteration) *With the above assumption we have*

$$\lim_{k\to\infty} \left|\frac{\varphi_{k+1}}{\varphi_k^3}\right| \le 1. \tag{7.30}$$

i.e., RQI converges cubically.

Proof. (The proof follows closely the one given by Parlett [4].) We have

$$\begin{aligned}\mathbf{z}_{k+1} = (A - \rho_k I)^{-1}\mathbf{y}_k &= \mathbf{x}\cos\varphi_k/(\lambda - \rho_k) + (A - \rho_k I)^{-1}\mathbf{u}_k \sin\varphi_k \\ &= \mathbf{x}\underbrace{\cos\varphi_k/(\lambda - \rho_k)}_{\|\mathbf{z_{k+1}}\|\cos\varphi_{k+1}} + \mathbf{u}_{k+1}\underbrace{\sin\varphi_k \|(A - \rho_k I)^{-1}\mathbf{u}_k\|}_{\|\mathbf{z_{k+1}}\|\sin\varphi_{k+1}},\end{aligned}$$

where we set

$$\mathbf{u}_{k+1} := (A - \rho_k I)^{-1}\mathbf{u}_k / \|(A - \rho_k I)^{-1}\mathbf{u}_k\| \tag{7.31}$$

such that $\|\mathbf{u}_{k+1}\| = 1$ and $\mathbf{u}_{k+1}^*\mathbf{x} = 0$. Thus,

$$\begin{aligned}\tan\varphi_{k+1} &= \sin\varphi_{k+1}/\cos\varphi_{k+1}\\ &= \sin\varphi_k\,\|(A - \rho_k I)^{-1}\mathbf{u}_k\|\,(\lambda - \rho_k)/\cos\varphi_k\\ &= (\lambda - \rho_k)\,\|(A - \rho_k I)^{-1}\mathbf{u}_k\|\,\tan\varphi_k\\ &= (\lambda - \rho(\mathbf{u}_k))\,\|(A - \rho_k I)^{-1}\mathbf{u}_k\|\,\sin^2\varphi_k\tan\varphi_k.\end{aligned}$$

So,

$$(A - \rho_k I)^{-1}\mathbf{u}_k = (A - \rho_k I)^{-1}\left(\sum_{\lambda_i \neq \lambda}\beta_i\mathbf{x}_i\right) = \sum_{\lambda_i \neq \lambda}\frac{\beta_i}{\lambda_i - \rho_k}\mathbf{x}_i$$

and taking norms,

$$\|(A - \rho_k I)^{-1}\mathbf{u}_k\|^2 = \sum_{\lambda_i \neq \lambda}\frac{\beta_i^2}{|\lambda_i - \rho_k|^2} \geq \frac{1}{\min_{\lambda_i \neq \lambda}|\lambda_i - \rho_k|^2}\underbrace{\sum_{\lambda_i \neq \lambda}\beta_i^2}_{\|\mathbf{u}_k\|^2 = 1} \tag{7.32}$$

We define the **gap** between the eigenvalue λ and the rest of A's spectrum by

$$\gamma := \min_{\lambda_i \neq \lambda}|\lambda_i - \lambda|.$$

The assumption implies that there must be a $k_0 \in \mathbb{N}$ such that $|\lambda - \rho_k| < \frac{\gamma}{2}$ for all $k > k_0$, and, therefore,

$$|\lambda_i - \rho_k| > \frac{\gamma}{2} \qquad \text{for all } \lambda_i \neq \lambda.$$

Using this in (7.32) gives

$$\|(A - \rho_k I)^{-1}\mathbf{u}_k\| \leq \frac{1}{\min_{\lambda_i \neq \lambda}|\lambda_i - \rho_k|} \leq \frac{2}{\gamma}, \qquad k > k_0.$$

Because $\tan\varphi_k \approx \sin\varphi_k \approx \varphi_k$ if $\varphi_k \ll 1$ we can deduce the cubic convergence rate.

We now look at the sequence $\{\mathbf{u}_k\}$ more closely. We note from (7.31) that this sequence is obtained by "*inverse iteration with variable shift* ρ_k". But since $\rho_k \longrightarrow \lambda$ with a cubic rate of convergence we can for large k assume that $\rho_k = \lambda$ and that $\mathbf{u}_k \perp \mathbf{x}$.

We now consider two cases, either $\{\mathbf{u}_k\}$ converges, or it does not converge.

1. We assume that $\{\mathbf{u}_k\}$ converges. Then the limit vector $\hat{\mathbf{u}}$ must be an eigenvector of A in span$\{\mathbf{x}\}^\perp$. (In general, $\hat{\mathbf{u}}$ is an eigenvector corresponding to the eigenvalue $\hat{\lambda}$ that is closest to λ.) Thus,

$$(\lambda - \rho(\mathbf{u}_k))\|(A - \rho_k I)^{-1}\mathbf{u}_k\| \xrightarrow[k\to\infty]{} \pm|\lambda - \hat{\lambda}| \cdot \|\hat{\mathbf{u}}/(\lambda - \hat{\lambda})\| = \pm 1.$$

2. Now we assume that $\{\mathbf{u}_k\}$ does *not* converge. Then A has two eigenvalues of equal distance to λ and the cluster points of the sequence $\{\mathbf{u}_k\}$ are two vectors in the plane

that is spanned by two eigenvectors corresponding to these two eigenvalues $\lambda \pm \delta$, $\alpha\mathbf{x}_p + \beta\mathbf{x}_q$, where $\alpha \neq 0$, $\beta \neq 0$, and $\alpha^2 + \beta^2 = 1$. Their Rayleigh quotients are

$$\rho(\alpha\mathbf{x}_p + \beta\mathbf{x}_q) = \alpha^2\lambda_p + \beta^2\lambda_q = \alpha^2(\lambda \pm \delta) + \beta^2(\lambda \mp \delta) = \lambda \pm \delta(\alpha^2 - \beta^2).$$

As $k \longrightarrow \infty$ the Rayleigh quotients of $\mathbf{u}_k$ jump between these two values. Hence,

$$(\lambda - \rho(\mathbf{u}_k))\|(A - \rho_k I)^{-1}\mathbf{u}_k\| \longrightarrow \pm\delta(\alpha^2 - \beta^2)/\delta,$$

and, therefore,

$$\left|\frac{\varphi_{k+1}}{\varphi_k^3}\right| \xrightarrow[k\to\infty]{} |\alpha^2 - \beta^2| < 1$$

■

Remark 7.6. Notice that we have not proved global convergence. Regarding this issue consult the book by Parlett [4] that contains all of this and more.

RQI converges 'almost always'. However, it is not clear in general towards which eigenpair the iteration converges. So, it is wise to start RQI only with good starting vectors. An alternative is to first apply inverse vector iteration and switch to Rayleigh quotient iteration as soon as the iterate is close enough to the solution. For references on this technique see [6, 1]. □

Remark 7.7. The Rayleigh quotient iteration is expensive. In every iteration step another system of equations has to be solved, i.e., in every iteration step a matrix has to be factorized. Therefore, RQI is usually applied only to tridiagonal matrices. □

7.9.1 A numerical example

The following MATLAB script demonstrates the power of Rayleigh quotient iteration. It expects as input a matrix `A`, an initial vector `x` of length one.

```
%       Initializations
k = 0; rho = 0; ynorm = 0;

while abs(rho)*ynorm < 1e+15,
   k = k + 1; if k>20, break, end
   rho = x'*A*x;
   y = (A - rho*eye(size(A)))\x;
   ynorm = norm(y);
   x = y/ynorm;
end
```

We invoked this routine with the matrix

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{9\times 9}$$

and the initial vector $\mathbf{x} = [-4, -3, \ldots, 4]^T$. The numbers obtained are

k	rho	ynorm
1	0.6666666666666666	3.1717e+00
2	0.4155307724080958	2.9314e+01
3	0.3820048793104663	2.5728e+04
4	0.3819660112501632	1.7207e+13
5	0.3819660112501051	2.6854e+16

The cubic convergence is evident.

Bibliography

[1] C. Beattie and D. Fox, *Localization criteria and containment for Rayleigh Quotient iteration*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 80–93.

[2] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 4th ed., 2012.

[3] R. A. Horn and C. R. Johnson, *Matrix Analysis*, Cambridge University Press, Cambridge, 1985.

[4] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980. (Republished by SIAM, Philadelphia, 1998.).

[5] H. R. Schwarz, *Methode der finiten Elemente*, Teubner, Stuttgart, 3rd ed., 1991.

[6] D. B. Szyld, *Criteria for combining inverse and Rayleigh Quotient iteration*, SIAM J. Numer. Anal., 25 (1988), pp. 1369–1375.

Chapter 8

Simultaneous vector or subspace iterations

8.1 Basic subspace iteration

We have learned in subsection 7.8 how to compute several eigenpairs of a matrix, one after the other. This turns out to be quite inefficient. Some or several of the quotients λ_{i+1}/λ_i may be close to one. The following algorithm differs from Algorithm 7.8 in that it does not perform p individual iterations for computing the, say, p smallest eigenvalues, but a single iteration with p vectors simultaneously.

Algorithm 8.1 Basic subspace iteration

```
Let $X \in \mathbb{F}^{n\times p}$ be a matrix with orthnormalized columns, $X^*X = I_p$. This algorithmus
computes eigenvectors corresponding to the $p$ largest eigenvalues $\lambda_1 \geq \cdots \geq \lambda_p$ of $A$.
Set $X^{(0)} := X, \quad k = 1$,
while $\|(I - X^{(k)}X^{(k)*})X^{(k-1)}\| > tol$ do
    $k := k + 1$
    $Z^{(k)} := AX^{(k-1)}$
    $X^{(k)}R^{(k)} := Z^{(k)}$/* QR factorization of $Z^{(k)}$ */
end while
```

The QR factorization in step 6 of the algorithm prevents the columns of the $X^{(k)}$ from converging all to an eigenvector of largest modulus.

If the convergence criterion is satisfied then

$$X^{(k)} - X^{(k-1)}(X^{(k-1)^*}X^{(k)}) = E, \quad \text{with} \quad \|E\| \leq tol.$$

Therefore, for the *'residual'*,

$$AX^{(k-1)} - X^{(k-1)}(X^{(k-1)^*}X^{(k)})R^{(k)} = ER^{(k)},$$

Therefore, in case of convergence, $X^{(k)}$ tends to an invariant subspace, say $\mathcal{R}(X_*)$. $X^{(k-1)^*}X^{(k)} \approx I_p$ and $AX^{(k)} = X^{(k)}R^{(k)}$ is an approximation of a *partial Schur decomposition*. We will show that the matrix $X^{(k)}$ converges to the Schur vectors associated with the largest p eigenvalues of A. In the convergence analysis we start from the Schur decomposition $A = UTU^*$ of A.

It is important to notice that in the QR factorization of $Z^{(k)}$ the j-th column affects only the columns to its right. If we would apply Algorithm 8.1 to a matrix $\hat{X} \in \mathbb{F}^{n\times q}$, with

$X\mathbf{e}_i = \hat{X}\mathbf{e}_i$ for $i = 1, \ldots, q$ then, for all k, we would have $X^{(k)}\mathbf{e}_i = \hat{X}^{(k)}\mathbf{e}_i$ for $i = 1, \ldots, j$. This, in particular, means that the first columns $X^{(k)}\mathbf{e}_1$ perform a simple vector iteration.

Problem 8.1 Show by recursion that the QR factorization of $A^k X = A^k X^{(0)}$ is given by

$$A^k X = X^{(k)} R^{(k)} R^{(k-1)} \cdots R^{(1)}.$$

8.2 Angles between subspaces

In the convergence analysis of the subspace iteration we need the notion of an angle between subspaces. Let $Q_1 \in \mathbb{F}^{n\times p}$, $Q_2 \in \mathbb{F}^{n\times q}$ be matrices with orthonormal columns, $Q_1^* Q_1 = I_p, Q_2^* Q_2 = I_q$. Let $S_i = \mathcal{R}(Q_i)$. Then S_1 and S_2 are subspaces of $\mathbb{F}^n$ of dimension p and q, respectively. We want to investigate how we can define a distance or an angle between S_1 and S_2 [2].

It is certainly straightforward to define the angle between the subspaces S_1 and S_2 to be the angle between two vectors $\mathbf{x}_1 \in S_1$ and $\mathbf{x}_2 \in S_2$. It is, however, not clear right-away how these vectors should be chosen. Let us consider the case of two 2-dimensional subspaces in $\mathbb{R}^3$, cf. Fig. (8.1). Let $S_1 = \text{span}\{\mathbf{q}_1,\ \mathbf{q}_2\}$ and $S_2 = \text{span}\{\mathbf{q}_1,\ \mathbf{q}_3\}$ where we assume that $\mathbf{q}_1^*\mathbf{q}_2 = \mathbf{q}_1^*\mathbf{q}_3 = 0$. We might be tempted to define the angle between S_1 and S_2 as the maximal angle between any two vectors in S_1 and S_2,

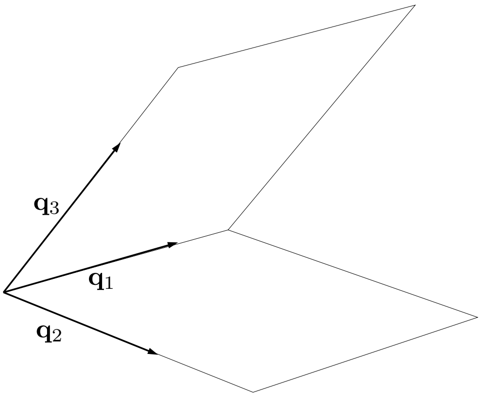


Figure 8.1: Two intersecting planes in 3-space

$$\angle(S_1, S_2) = \max_{\substack{\mathbf{x}_1 \in S_1 \\ \mathbf{x}_2 \in S_2}} \angle(\mathbf{x}_1, \mathbf{x}_2). \tag{8.1}$$

This would give an angle of 90^o as we could chose $\mathbf{q}_1$ in S_1 and $\mathbf{q}_3$ in S_2. This angle would not change as we turn S_2 around $\mathbf{q}_1$. It would even stay the same if the two planes coincided.

What if we would take the minimum in (8.1)? This definition would be equally unsatisfactory as we could chose $\mathbf{q}_1$ in S_1 as well as in S_2 to obtain an angle of 0^o. In fact, any two 2-dimensional subspaces in 3 dimensions would have an angle of 0^o. Of course, we would like to reserve the angle of 0^o to coinciding subspaces.

A way out of this dilemma is to proceed as follows: Take any vector $\mathbf{x}_1 \in S_1$ and determine the angle between $\mathbf{x}_1$ *and its orthogonal projection* $(I - Q_2^* Q_2)\mathbf{x}_1$ on S_2. We now maximize the angle by varying $\mathbf{x}_1$ among all non-zero vectors in S_1. In the above 3-dimensional example we would obtain the angle between $\mathbf{x}_2$ and $\mathbf{x}_3$ as the angle between

S_1 and S_3. Is this a reasonable definition? In particular, is it well-defined in the sense that it does not depend on how we number the two subspaces? Let us now assume that $S_1, S_2 \subset \mathbb{F}^n$ have dimensions p and q. Formally, the above procedure gives an angle ϑ with

$$\begin{aligned}\sin\vartheta := \max_{\substack{\mathbf{r}\in S_1\\ \|\mathbf{r}\|=1}} \|(I_n - Q_2Q_2^*)\mathbf{r}\| = \max_{\substack{\mathbf{a}\in\mathbb{F}^p\\ \|\mathbf{a}\|=1}} \|(I_n - Q_2Q_2^*)Q_1\mathbf{a}\| \\ = \|(I_n - Q_2Q_2^*)Q_1\|.\end{aligned} \tag{8.2}$$

Because $I_n - Q_2Q_2^*$ is an orthogonal projection, we get

$$\begin{aligned}\|(I_n - Q_2Q_2^*)Q_1\mathbf{a}\|^2 &= \mathbf{a}^*Q_1^*(I_n - Q_2Q_2^*)(I_n - Q_2Q_2^*)Q_1\mathbf{a}\\ &= \mathbf{a}^*Q_1^*(I_n - Q_2Q_2^*)Q_1\mathbf{a}\\ &= \mathbf{a}^*(Q_1^*Q_1 - Q_1^*Q_2Q_2^*Q_1)\mathbf{a}\\ &= \mathbf{a}^*(I_p - (Q_1^*Q_2)(Q_2^*Q_1))\mathbf{a}\\ &= \mathbf{a}^*(I_p - W^*W)\mathbf{a}\end{aligned} \tag{8.3}$$

where $W := Q_2^*Q_1 \in \mathbb{F}^{q\times p}$. With (8.2) we obtain

$$\begin{aligned}\sin^2\vartheta &= \max_{\|\mathbf{a}\|=1} \mathbf{a}^*(I_p - W^*W)\mathbf{a}\\ &= \text{largest eigenvalue of } I_p - W^*W\\ &= 1 - \text{smallest eigenvalue of } W^*W.\end{aligned} \tag{8.4}$$

If we change the roles of Q_1 and Q_2 we get in a similar way

$$\sin^2\varphi = \|(I_n - Q_1Q_1^*)Q_2\| = 1 - \text{smallest eigenvalue of } WW^*. \tag{8.5}$$

Notice, that $W^*W \in \mathbb{F}^{p\times p}$ and $WW^* \in \mathbb{F}^{q\times q}$ and that both matrices have equal rank. Thus, if W has full rank and $p < q$ then $\vartheta < \varphi = \pi/2$. However if $p = q$ then W^*W and WW^* have equal eigenvalues, and, thus, $\vartheta = \varphi$. In this most interesting case we have

$$\sin^2\vartheta = 1 - \lambda_{\min}(W^*W) = 1 - \sigma^2_{\min}(W),$$

where $\sigma_{\min}(W)$ is the smallest singular value of W [2, p.16].

For our purposes in the analysis of eigenvalue solvers the following definition is most appropriate.

Definition 8.2 Let $S_1, S_2 \subset \mathbb{F}^n$ be of dimensions p and q and let $Q_1 \in \mathbb{F}^{n\times p}$ and $Q_2 \in \mathbb{F}^{n\times q}$ be matrices the columns of which form orthonormal bases of S_1 and S_2, respectively, i.e. $S_i = \mathcal{R}(Q_i)$, $i = 1, 2$. Then we define the angle ϑ, $0 \le \vartheta \le \pi/2$, between S_1 and S_2 by

$$\sin\vartheta = \sin\angle(S_1, S_2) = \begin{cases}\sqrt{1 - \sigma^2_{\min}(Q_1^*Q_2)} & \text{if } p = q,\\ 1 & \text{if } p \ne q.\end{cases}$$

If $p = q$ the equations (8.2)–(8.4) imply that

$$\begin{aligned}\sin^2\vartheta &= \max_{\|\mathbf{a}\|=1} \mathbf{a}^*(I_p - W^*W)\mathbf{a} = \max_{\|\mathbf{b}\|=1} \mathbf{b}^*(I_p - WW^*)\mathbf{b}\\ &= \|(I_n - Q_2Q_2^*)Q_1\| = \|(I_n - Q_1Q_1^*)Q_2\|\\ &= \|(Q_1Q_1^* - Q_2Q_2^*)Q_1\| = \|(Q_1Q_1^* - Q_2Q_2^*)Q_2\|\end{aligned} \tag{8.6}$$

Let $\mathbf{x} \in S_1 + S_2$. Then $\mathbf{x} = \tilde{\mathbf{q}}_1 + \tilde{\mathbf{q}}_2$ with $\tilde{\mathbf{q}}_i \in S_i$. We write

$$\mathbf{x} = \tilde{\mathbf{q}}_1 + Q_1 Q_1^* \tilde{\mathbf{q}}_2 + (I_n - Q_1 Q_1^*)\tilde{\mathbf{q}}_2 =: \mathbf{q}_1 + \mathbf{q}_2$$

with $\mathbf{q}_1 = Q_1\mathbf{a}$ and $\mathbf{q}_2 = Q_2\mathbf{b} = (I_n - Q_1Q_1^*)Q_2\mathbf{b}$. Then

$$\begin{aligned}
\|(Q_1Q_1^* - Q_2Q_2^*)\mathbf{x}\|^2 &= \|(Q_1Q_1^* - Q_2Q_2^*)(Q_1\mathbf{a} + Q_2\mathbf{b})\|^2 \\
&= \|Q_1\mathbf{a} + Q_2Q_2^*Q_1\mathbf{a} + Q_2\mathbf{b}\|^2 \\
&= \|(I_n - Q_2Q_2^*)Q_1\mathbf{a} + Q_2\mathbf{b}\|^2 \\
&= \mathbf{a}^*Q_1^*(I_n - Q_2Q_2^*)Q_1\mathbf{a} \\
&\qquad + 2\mathrm{Re}(\mathbf{a}^*Q_1^*(I_n - Q_2Q_2^*)Q_2\mathbf{b}) + \mathbf{b}^*Q_2^*Q_2\mathbf{b} \\
\sin^2\vartheta &= \max_{\|\mathbf{a}\|=1} \mathbf{a}^*Q_1^*(I_n - Q_2Q_2^*)Q_1\mathbf{a}, \\
&= \max_{\|\mathbf{a}\|=1} \mathbf{a}^*Q_1^*(Q_1Q_1^* - Q_2Q_2^*)Q_1\mathbf{a}, \\
&= \max_{\mathbf{x}\in S_1\setminus\{\mathbf{0}\}} \frac{\mathbf{x}^*(Q_1Q_1^* - Q_2Q_2^*)\mathbf{x}}{\mathbf{x}^*\mathbf{x}}.
\end{aligned}$$

Thus, $\sin\vartheta$ is the maximum of the Rayleigh quotient $R(\mathbf{x})$ corresponding to $Q_1Q_1^* - Q_2Q_2^*$, that is the largest eigenvalue of $Q_1Q_1^* - Q_2Q_2^*$. As $Q_1Q_1^* - Q_2Q_2^*$ is symmetric and positive semi-definite, its largest eigenvalue equals its norm,

Lemma 8.3 $\sin\angle(S_1, S_2) = \|Q_2Q_2^* - Q_1Q_1^*\|$

Lemma 8.4 $\angle(S_1, S_2) = \angle({S_1}^\perp, {S_2}^\perp)$.

Proof. Because

$$\|Q_2Q_2^* - Q_1Q_1^*\| = \|(I - Q_2Q_2^*) - (I - Q_1Q_1^*)\|$$

the claim immediately follows from Lemma 8.3. ■

8.3 Convergence of basic subspace iteration

In analyzing the convergence of the basic subspace iteration we proceed similarly as in the analysis of the simple vector iteration, exploiting the Jordan normal form $A = XJY^*$ with $Y^* := X^{-1}$. We assume that the p largest eigenvalues of A in modulus are separated from the rest of the spectrum,

$$|\lambda_1| \geq \cdots \geq |\lambda_p| > |\lambda_{p+1}| \geq \cdots \geq |\lambda_n|. \tag{8.7}$$

This means that the eigenvalues are arranged on the diagonal of the Jordan block matrix J in the order given in (8.7).

In fact as we can either analyze the original iteration $\{X^{(k)}\}$ in the canonical coordinate system or the iteration $\{Y^{(k)}\} = \{U^*X^{(k)}\}$ in the coordinate system generated by the (generalized) eigenvectors we assume that A itself is a Jordan block matrix with its diagonal elements arranged as in (8.7).

The invariant subspace of A associated with the p largest (or *dominant*) eigenvalues is given by $\mathcal{R}(E_p)$ where $E_p = [\mathbf{e}_1, \ldots, \mathbf{e}_p]$. We are now going to show that the angle between $\mathcal{R}(X^{(k)})$ and $\mathcal{R}(E_p)$, tends to zero as k goes to ∞.

From Problem 8.1 we know that

$$\vartheta^{(k)} := \angle(\mathcal{R}(E_p), \mathcal{R}(X^{(k)})) = \angle(\mathcal{R}(E_p), \mathcal{R}(A^kX^{(0)})). \tag{8.8}$$

We partition the matrices A and $X^{(k)}$,

$$A = \operatorname{diag}(J_1, J_2), \quad X^{(k)} = \begin{bmatrix} X_1^{(k)} \\ X_2^{(k)} \end{bmatrix}, \qquad J_1, X_1^{(k)} \in \mathbb{F}^{p\times p}.$$

From (8.7) we know that J_1 is nonsingular. Let us also assume that $X_1^{(k)} = E_p^* X^{(k)}$ is invertible. This means, that $X^{(k)}$ has components in the direction of the invariant subspace associated with the p dominant eigenvalues. Then, with Problem 8.1,

$$X^{(k)}R = A^k X^{(0)} = \begin{bmatrix} J_1^k X_1^{(0)} \\ J_2^k X_2^{(0)} \end{bmatrix} = \begin{bmatrix} I_p \\ S^{(k)} \end{bmatrix} J_1^k X_1^{(0)}, \qquad S^{(k)} := J_2^k X_2^{(0)} {X_1^{(0)}}^{-1} J_1^{-k}. \tag{8.9}$$

Notice that $X_1^{(k)}$ is invertible if $X_1^{(0)}$ is so. (8.8) and (8.9) imply that

$$\begin{aligned} \sin\vartheta^{(k)} &= \|(I - E_p E_p^*)X^{(k)}\| \\ &= \left\|(I - E_p E_p^*)\begin{bmatrix} I_p \\ S^{(k)} \end{bmatrix}\right\| \Big/ \left\|\begin{bmatrix} I_p \\ S^{(k)} \end{bmatrix}\right\| = \frac{\|S^{(k)}\|}{\sqrt{1 + \|S^{(k)}\|^2}}. \end{aligned} \tag{8.10}$$

Likewise, we have

$$\cos\vartheta^{(k)} = \|E_p^* X^{(k)}\| = \frac{1}{\sqrt{1 + \|S^{(k)}\|^2}}.$$

Since $\rho(J_2) = |\lambda_{p+1}|$ and $\rho(J_1^{-1}) = 1/|\lambda_p|$ we obtain with (7.13) and a few algebraic manipulations for any $\varepsilon > 0$ that

$$\tan\vartheta^{(k)} = \|S^{(k)}\| \le \|J_2^k\| \|S^{(0)}\| \|J_1^{-k}\| \le \left(\left| \frac{\lambda_{p+1}}{\lambda_p} \right| + \varepsilon \right)^k \tan\vartheta^{(0)}, \tag{8.11}$$

for $k > K(\varepsilon)$. Making a transformation back to a general matrix A as before Theorem 7.5 we get

Theorem 8.5 *Let $U_p, V_p \in \mathbb{F}^{n\times p}$, $U_p^* U_p = V_p^* V_p = I_p$, be matrices that span the right and left invariant subspace associated with the dominant p eigenvalues $\lambda_1, \ldots, \lambda_p$ of A. Let $X^{(0)} \in \mathbb{F}^{n\times p}$ be such that $V_p^* X^{(0)}$ is nonsingular. Then, if $|\lambda_p| < |\lambda_{p+1}|$ and $\varepsilon > 0$, the iterates $X^{(k)}$ of the basic subspace iteration with initial subpace $X^{(0)}$ converges to U_p, and*

$$\tan\vartheta^{(k)} \le \left(\left| \frac{\lambda_{p+1}}{\lambda_p} \right| + \varepsilon \right)^k \tan\vartheta^{(0)}, \qquad \vartheta^{(k)} = \angle(\mathcal{R}(U_p), \mathcal{R}(X^{(k)})) \tag{8.12}$$

for sufficiently large k.

If the matrix A is Hermitian or real-symmetric we can simplify Theorem 8.5.

Theorem 8.6 *Let $U_p := [\mathbf{u}_1, \ldots, \mathbf{u}_p]$ be the matrix formed by the eigenvectors corresponding to the p dominant eigenvalues $\lambda_1, \ldots, \lambda_p$ of A. Let $X^{(0)} \in \mathbb{F}^{n\times p}$ be such that $U_p^* X^{(0)}$ is nonsingular. Then, if $|\lambda_p| < |\lambda_{p+1}|$, the iterates $X^{(k)}$ of the basic subspace iteration with initial subpace $X^{(0)}$ converges to U_p, and*

$$\tan\vartheta^{(k)} \le \left| \frac{\lambda_{p+1}}{\lambda_p} \right|^k \tan\vartheta^{(0)}, \qquad \vartheta^{(k)} = \angle(\mathcal{R}(U_p), \mathcal{R}(X^{(k)})). \tag{8.13}$$

Let us elaborate on this result. (Here we assume that A is Hermitian or real-symmetric. Otherwise the statements are similar modulo ε terms as in (8.12).) Let us assume that not only $W_p := U_p^* X$ is nonsingular but that *each* principal submatrix

$$W_j := \begin{pmatrix} w_{11} & \cdots & w_{1j} \\ \vdots & & \vdots \\ w_{j1} & \cdots & w_{jj} \end{pmatrix}, \quad 1 \le j \le p,$$

of W_p is nonsingular. Then we can apply Theorem 8.6 to each set of columns $[\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_j^{(k)}]$, $1 \le j \le p$, provided that $|\lambda_j| < |\lambda_{j+1}|$. If this is the case, then

$$\tan \vartheta_j^{(k)} \le \left| \frac{\lambda_{j+1}}{\lambda_j} \right|^k \tan \vartheta_j^{(0)}, \tag{8.14}$$

where $\vartheta_j^{(k)} = \angle(\mathcal{R}([\mathbf{u}_1, \ldots, \mathbf{u}_j]), \mathcal{R}([\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_j^{(k)}]))$.

We can even say a little more. We can combine the statements in (8.14) as follows.

Theorem 8.7 *Let $X \in \mathbb{F}^{n\times p}$. Let $|\lambda_{q-1}| > |\lambda_q| \ge \ldots \ge |\lambda_p| > |\lambda_{p+1}|$. Let W_q and W_p be nonsingular. Then*

$$\sin \angle(\mathcal{R}([\mathbf{x}_q^{(k)}, \ldots, \mathbf{x}_p^{(k)}]), \mathcal{R}([\mathbf{u}_q, \ldots, \mathbf{u}_p])) \le c \cdot \max \left\{ \left| \frac{\lambda_q}{\lambda_{q-1}} \right|^k, \left| \frac{\lambda_{p+1}}{\lambda_p} \right|^k \right\}. \tag{8.15}$$

Proof. Recall that the sine of the angle between two subspaces S_1, S_2 of equal dimension is the norm of the projection on $S_2^\perp$ restricted to S_1, see (8.6). Here, $S_1 = \mathcal{R}([\mathbf{x}_q^{(k)}, \ldots, \mathbf{x}_p^{(k)}])$ and $S_2 = \mathcal{R}([\mathbf{u}_q, \ldots, \mathbf{u}_p])$.

Let $\mathbf{x} \in S_1$ with $\|\mathbf{x}\| = 1$. The orthogonal projection of $\mathbf{x}$ on S_2 reflects the fact, that $\mathbf{y} \in \mathcal{R}([\mathbf{u}_q, \ldots, \mathbf{u}_p])$ implies that $\mathbf{y} \in \mathcal{R}([\mathbf{u}_1, \ldots, \mathbf{u}_p])$ as well as $\mathbf{y} \in \mathcal{R}([\mathbf{u}_1, \ldots, \mathbf{u}_q])^\perp$,

$$U_{q-1}U_{q-1}^*\mathbf{x} + (I - U_pU_p^*)\mathbf{x}.$$

To estimate the norm of this vector we make use of Lemmata 8.4 and (8.10),

$$\begin{aligned} \|U_{q-1}U_{q-1}^*\mathbf{x} + (I - U_pU_p^*)\mathbf{x}\| &= \left(\|U_{q-1}U_{q-1}^*\mathbf{x}\|^2 + \|(I - U_pU_p^*)\mathbf{x}\|^2\right)^{1/2} \\ &\le \left(\sin^2 \vartheta_{q-1}^{(k)} + \sin^2 \vartheta_p^{(k)}\right)^{1/2} \le \sqrt{2} \cdot \max\left\{\sin \vartheta_{q-1}^{(k)}, \sin \vartheta_p^{(k)}\right\} \\ &\le \sqrt{2} \cdot \max\left\{\tan \vartheta_{q-1}^{(k)}, \tan \vartheta_p^{(k)}\right\}. \end{aligned}$$

Then, inequality (8.15) is obtained by applying (8.14) that we know to hold true for both $j = q-1$ and $j = p$. ■

Corollary 8.8 *Let $X \in \mathbb{F}^{n\times p}$. Let $|\lambda_{j-1}| > |\lambda_j| > |\lambda_{j+1}|$ and let W_{j-1} and W_j be nonsingular. Then*

$$\sin \angle(\mathbf{x}_j^{(k)}, \mathbf{u}_j) \le c \cdot \max \left\{ \left| \frac{\lambda_j}{\lambda_{j-1}} \right|^k, \left| \frac{\lambda_{j+1}}{\lambda_j} \right|^k \right\}. \tag{8.16}$$

Example 8.9 Let us see how subspace iteration performs with the matrix

$$A = \operatorname{diag}(1, 3, 4, 6, 10, 15, 20, \ldots, 185)^{-1} \in \mathbb{R}^{40\times 40}$$

if we iterate with 5 vectors. The critical quotients appearing in Corollary 8.8 are

j	1	2	3	4	5
$\lvert\lambda_{j+1}\rvert/\lvert\lambda_j\rvert$	$1/3$	$3/4$	$2/3$	$3/5$	$2/3$

.

So, according to (8.16), the first column $\mathbf{x}_1^{(k)}$ of $X^{(k)} \in \mathbb{R}^{40\times 5}$ should converge to the first eigenvector at a rate 1/3, $\mathbf{x}_2^{(k)}$ *and* $\mathbf{x}_3^{(k)}$ should converge at a rate 3/4 and the last two columns should converge at the rate 2/3. The graphs in Figure 8.2 show that convergence takes place in exactly this manner.

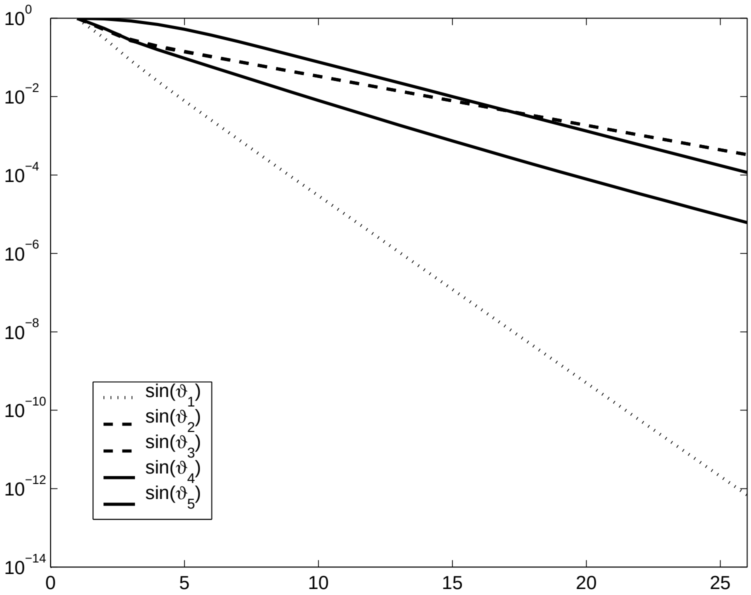


Figure 8.2: Basic subspace iteration with $\tau I_{40} - T_{40}$

Similarly as earlier the eigenvalue approximations $\lambda_j^{(k)}$ approach the desired eigenvalues more rapidly than the eigenvectors. In fact we have

$$\lambda_j^{(k+1)^2} = \|\mathbf{z}_j^{(k+1)}\|^2 = \frac{\mathbf{x}_j^{(k)^*} A^2 \mathbf{x}_j^{(k)}}{\mathbf{x}_j^{(k)^*}\mathbf{x}_j^{(k)}} = \mathbf{x}_j^{(k)^*} A^2 \mathbf{x}_j^{(k)},$$

since $\|\mathbf{x}_j^{(k)}\| = 1$. Let $\mathbf{x}_j^{(k)} = \mathbf{u} + \mathbf{u}^\perp$, where $\mathbf{u}$ is the eigenvalue corresponding to λ_j. Then, since $\mathbf{u} = \mathbf{x}_j^{(k)} \cos\phi$ and $\mathbf{u}^\perp = \mathbf{x}_j^{(k)} \sin\phi$ for a $\phi \le \vartheta^{(k)}$, we have

$$\begin{aligned}
\lambda_j^{(k+1)^2} &= \mathbf{x}_j^{(k)^*} A^2 \mathbf{x}_j^{(k)} = \mathbf{u}^* A\mathbf{u} + \mathbf{u}^{\perp^*} A\mathbf{u}^\perp = \lambda_j^2 \mathbf{u}^*\mathbf{u} + \mathbf{u}^{\perp^*} A\mathbf{u}^\perp \\
&\le \lambda_j^2 \|\mathbf{u}\|^2 + \lambda_1^2 \|\mathbf{u}^\perp\|^2 \\
&\le \lambda_j^2 \cos^2\vartheta^{(k)} + \lambda_1^2 \sin^2\vartheta^{(k)} \\
&= \lambda_j^2(1 - \sin^2\vartheta^{(k)}) + \lambda_1^2 \sin^2\vartheta^{(k)} = \lambda_j^2 + (\lambda_1^2 - \lambda_j^2)\sin^2\vartheta^{(k)}.
\end{aligned}$$

Thus,

$$|\lambda_j^{(k+1)} - \lambda_j| \le \frac{\lambda_1^2 - \lambda_j^{(k+1)^2}}{\lambda_j^{(k+1)} + \lambda_j} \sin^2\vartheta^{(k)} = O\left(\max\left\{\left(\frac{\lambda_j}{\lambda_{j-1}}\right)^k, \left(\frac{\lambda_{j+1}}{\lambda_j}\right)^k\right\}\right).$$

k	$\frac{\lambda_1^{(k-1)}-\lambda_1}{\lambda_1^{(k)}-\lambda_1}$	$\frac{\lambda_2^{(k-1)}-\lambda_2}{\lambda_2^{(k)}-\lambda_2}$	$\frac{\lambda_3^{(k-1)}-\lambda_3}{\lambda_3^{(k)}-\lambda_3}$	$\frac{\lambda_4^{(k-1)}-\lambda_4}{\lambda_4^{(k)}-\lambda_4}$	$\frac{\lambda_5^{(k-1)}-\lambda_5}{\lambda_5^{(k)}-\lambda_5}$
1	0.0002	0.1378	-0.0266	0.0656	0.0315
2	0.1253	0.0806	-0.2545	0.4017	-1.0332
3	0.1921	0.1221	1.5310	0.0455	0.0404
4	0.1940	0.1336	0.7649	-3.0245	-10.4226
5	0.1942	0.1403	0.7161	0.9386	1.1257
6	0.1942	0.1464	0.7002	0.7502	0.9327
7	0.1942	0.1522	0.6897	0.7084	0.8918
8	0.1942	0.1574	0.6823	0.6918	0.8680
9	0.1942	0.1618	0.6770	0.6828	0.8467
10	0.1942	0.1652	0.6735	0.6772	0.8266
11	0.1943	0.1679	0.6711	0.6735	0.8082
12	0.1942	0.1698	0.6694	0.6711	0.7921
13	0.1933	0.1711	0.6683	0.6694	0.7786
14	0.2030	0.1720	0.6676	0.6683	0.7676
15	0.1765	0.1727	0.6671	0.6676	0.7589
16		0.1733	0.6668	0.6671	0.7522
17		0.1744	0.6665	0.6668	0.7471
18		0.2154	0.6664	0.6665	0.7433
19		0.0299	0.6663	0.6664	0.7405
20			0.6662	0.6663	0.7384
21			0.6662	0.6662	0.7370
22			0.6662	0.6662	0.7359
23			0.6661	0.6662	0.7352
24			0.6661	0.6661	0.7347
25			0.6661	0.6661	0.7344
26			0.6661	0.6661	0.7343
27			0.6661	0.6661	0.7342
28			0.6661	0.6661	0.7341
29			0.6661	0.6661	0.7342
30			0.6661	0.6661	0.7342
31			0.6661	0.6661	0.7343
32			0.6661	0.6661	0.7343
33			0.6661	0.6661	0.7344
34			0.6661	0.6661	0.7345
35			0.6661	0.6661	0.7346
36			0.6661	0.6661	0.7347
37			0.6661	0.6661	0.7348
38			0.6661	0.6661	0.7348
39			0.6661	0.6661	0.7349
40			0.6661	0.6661	0.7350

Table 8.1: Example of basic subspace iteration.
The convergence criterion $\|(I - X^{(k-1)}X^{(k-1)^*})X^{(k)}\| < 10^{-6}$ was satisfied after 87 iteration steps

A numerical example

Let us again consider the test example introduced in subsection 1.6.3 that deals with the accustic vibration in the interior of a car. The eigenvalue problem for the Laplacian is solved by the finite element method as introduced in subsection 1.6.2. We use the finest grid in Fig. 1.9. The matrix eigenvalue problem

$$A\mathbf{x} = \lambda B\mathbf{x}, \qquad A, B \in \mathbb{F}^{n\times n}, \tag{8.17}$$

has the order $n = 1095$. Subspace iteration is applied with five vectors as an *inverse iteration* to

$$L^{-1}AL^{-T}(L\mathbf{x}) = \lambda(L\mathbf{x}), \qquad B = LL^T. \quad \text{(Cholesky factorization)}$$

$X^{(0)}$ is chosen to be a random matrix. Here, we number the eigenvalues from small to big. The smallest six eigenvalues of (8.17) shifted by 0.01 to the right are

$$\begin{aligned} \hat\lambda_1 &= 0.01, & \hat\lambda_4 &= 0.066635,\\ \hat\lambda_2 &= 0.022690, & \hat\lambda_5 &= 0.126631,\\ \hat\lambda_3 &= 0.054385, & \hat\lambda_6 &= 0.147592. \end{aligned}$$

and thus the ratios of the eigenvalues that determine the rate of convergence are

$$\begin{aligned} (\hat\lambda_1/\hat\lambda_2)^2 &= 0.194, & (\hat\lambda_4/\hat\lambda_5)^2 &= 0.277,\\ (\hat\lambda_2/\hat\lambda_3)^2 &= 0.174, & (\hat\lambda_5/\hat\lambda_6)^2 &= 0.736,\\ (\hat\lambda_3/\hat\lambda_4)^2 &= 0.666. \end{aligned}$$

So, the numbers presented in Table 8.1 reflect quite accurately the predicted rates. The numbers in column 6 are a little too small, though.

The convergence criterion

$$\max_{1\le i\le p} \|(I - X^{(k)}X^{(k)*})\mathbf{x}_i^{(k-1)}\| \le \epsilon = 10^{-5}$$

was not satisfied after 50 iteration step.

8.4 Accelerating subspace iteration

Subspace iteration potentially converges very slowly. It can be slow even it one starts with a subspace that contains all desired solutions! If, e.g., $\mathbf{x}_1^{(0)}$ and $\mathbf{x}_2^{(0)}$ are both elements in $\mathcal{R}([\mathbf{u}_1, \mathbf{u}_2])$, the vectors $\mathbf{x}_i^{(k)}$, $i = 1, 2, \ldots$, still converge linearly towards $\mathbf{u}_1$ und $\mathbf{u}_2$ although they could be readily obtained from the 2×2 eigenvalue problem,

$$\begin{bmatrix} \mathbf{x}_1^{(0)*} \\ \mathbf{x}_2^{(0)*} \end{bmatrix} A \left[\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}\right] \mathbf{y} = \lambda\mathbf{y}$$

The following theorem gives hope that the convergence rates can be improved if one proceeds in a suitable way.

Theorem 8.10 *Let $X \in \mathbb{F}^{n\times p}$ as in Theorem 8.5. Let $\mathbf{u}_i$, $1 \le i \le p$, be the eigenvectors corresponding to the eigenvalues $\lambda_1, \ldots, \lambda_p$ of A. Then we have*

$$\min_{\mathbf{x}\in\mathcal{R}(X^{(k)})} \sin\angle(\mathbf{u}_i, \mathbf{x}) \le c\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k$$

Proof. In the proof of Theorem 8.5 we have seen that

$$\mathcal{R}(X^{(k)}) = \mathcal{R}\left(U\begin{pmatrix} I_p \\ \mathbf{S}^{(k)}\end{pmatrix}\right), \qquad \mathbf{S}^{(k)} \in \mathbb{F}^{(n-p)\times p},$$

where

$$s_{ij}^{(k)} = s_{ij}\left(\frac{\lambda_j}{\lambda_{p+i}}\right)^k, \qquad 1 \le i \le n-p, \quad 1 \le j \le p.$$

But we have

$$\begin{aligned}
\min_{\mathbf{x}\in\mathcal{R}(X^{(k)})} \sin \angle(\mathbf{u}_i, \mathbf{x}) &\le \sin \angle\left(\mathbf{u}_i, U\begin{pmatrix} I_p \\ \mathbf{S}^{(k)}\end{pmatrix}\mathbf{e}_i\right), \\
&= \left\| (I - \mathbf{u}_i{\mathbf{u}_i}^*)U\begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ s_{1i}(\lambda_i/\lambda_{p+1})^k \\ \vdots \\ s_{n-p,i}(\lambda_i/\lambda_n)^k \end{pmatrix}\right\| \Big/ \left\|\begin{pmatrix} I_p \\ \mathbf{S}^{(k)}\end{pmatrix}\mathbf{e}_i\right\| \\
&\le \left\| (I - \mathbf{u}_i{\mathbf{u}_i}^*)\left(\mathbf{u}_i + \sum_{j=p+1}^{n} s_{j-p,i}\left(\frac{\lambda_i}{\lambda_{p+j}}\right)^k \mathbf{u}_j\right)\right\| \\
&= \sqrt{\sum_{j=1}^{n-p} s_{ji}^2 \frac{\lambda_i^{2k}}{\lambda_{p+j}^{2k}}} \le \left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k \sqrt{\sum_{j=1}^{n-p} s_{ji}^2}.
\end{aligned}$$

■

These considerations lead to the idea to complement Algorithm 8.1 by a so-called **Rayleigh-Ritz step**. Here we give an 'inverted algorithm' to compute the smallest eigenvalues and corresponding eigenvectors of a symmetric/Hermitian matrix. For the corresponding nonsymmetric algorithm see [1].

Algorithm 8.2 Subspace or simultaneous inverse iteration combined with Rayleigh-Ritz step

```
Let $X \in \mathbb{F}^{n\times p}$ with $X^*X = I_p$:
Set $X^{(0)} := X$.
for $k = 1, 2, \ldots$ do
    Solve $AZ^{(k)} := X^{(k-1)}$
    $Q^{(k)}R^{(k)} := Z^{(k)}$    /* QR factorization of $Z^{(k)}$ (or modified Gram–Schmidt) */
    $\hat{H}^{(k)} := Q^{(k)*}AQ^{(k)}$,
    $\hat{H}^{(k)} =: F^{(k)}\Theta^{(k)}F^{(k)*}$    /* Spectral decomposition of $\hat{H}^{(k)} \in \mathbb{F}^{p\times p}$ */
    $X^{(k)} = Q^{(k)}F^{(k)}$.
end for
```

Remark 8.1. The columns $\mathbf{x}_i^{(k)}$ of $X^{(k)}$ are called **Ritz vectors** and the eigenvalues $\vartheta_1^{(k)} \le \cdots \le \vartheta_p^{(k)}$ in the diagonal of Θ are called **Ritz values**. According to the Rayleigh-

Ritz principle 2.32 we have

$$\lambda_i \leq \vartheta_i^{(k)} \qquad 1 \leq i \leq p, \quad k > 0.$$

□

The solution of the *full* eigenvalue problems $\hat{H}^{(k)}\mathbf{y} = \vartheta\mathbf{y}$ is solved by the symmetric QR algorithm.

The computation of the matrix $\hat{H}^{(k)}$ is expensive as matrix-vector products have to be executed. The following considerations simplify matters. We write $X^{(k)}$ in the form

$$X^{(k)} = Z^{(k)}G^{(k)}, \qquad G^{(k)} \in \mathbb{F}^{p\times p}\text{nonsingular}$$

Because $X^{(k)}$ must have orthonormal columns we must have

$$G^{(k)^*}Z^{(k)^*}Z^{(k)}G^{(k)} = I_p. \tag{8.18}$$

Furthermore, the columns of $Z^{(k)}G^{(k)}$ are the Ritz vectors in $\mathcal{R}(A^{-k}X)$ of A^2,

$$G^{(k)^*}Z^{(k)^*}A^2Z^{(k)}G^{(k)} = \Delta^{(k)^{-2}},$$

where $\Delta^{(k)}$ is a diagonal matrix. Using the definition of $Z^{(k)}$ in Algorithm 8.2 we see that

$$G^{(k)^*}X^{(k-1)^*}X^{(k-1)}G^{(k)} = G^{(k)^*}G^{(k)} = \Delta^{(k)^{-2}},$$

and that $Y^{(k)} := G^{(k)}\Delta^{(k)}$ is orthogonal. Substituting into (8.18) gives

$$Y^{(k)^*}Z^{(k)^*}Z^{(k)}Y^{(k)} = \Delta^{(k)^2}.$$

So, the columns of $Y^{(k)}$ are the normalized eigenvectors of $H^{(k)} := Z^{(k)^*}Z^{(k)}$.

Thus we obtain a second variant of the inverse subspace iteration with Rayleigh-Ritz step.

Algorithm 8.3 Subspace or simultaneous inverse vector iteration combined with Rayleigh-Ritz step, version 2

```
Let $X \in \mathbb{F}^{n\times p}$ with $X^*X = I_p$.
Set $X^{(0)} := X$.
for $k = 1, 2, \ldots$ do
    $AZ^{(k)} := X^{(k-1)}$;
    $H^{(k)} := Z^{(k)^*}Z^{(k)}$    /* $= X^{(k-1)^*}A^{-2}X^{(k-1)}$ */
    $H^{(k)} =: Y^{(k)}\Delta^{(k)^2}Y^{(k)^*}$    /* Spectral decomposition of $H^{(k)}$ */
    $X^{(k)} = Z^{(k)}Y^{(k)}\Delta^{(k)^{-1}}$    /* $= Z^{(k)}G^{(k)}$ */
end for
```

Remark 8.2. An alternative to Algorithm 8.3 is the subroutine `ritzit`, that has been programmed by Rutishauser [5] in ALGOL, see also [3, p.293]. □

We are now going to show that the Ritz vectors converge to the eigenvectors, as Theorem 8.10 lets us hope. First we prove

Lemma 8.11 ([3, p.222]) *Let* $\mathbf{y}$ *be a unit vector and* $\vartheta \in \mathbb{F}$*. Let* λ *be the eigenvalue of* A *closest to* ϑ *and let* $\mathbf{u}$ *be the corresponding eigenvector. Let*

$$\gamma := \min_{\lambda_i(A)\neq\lambda} |\lambda_i(A) - \vartheta|$$

and let $\psi = \angle(\mathbf{y},\mathbf{u})$. Then

$$\sin\psi \le \frac{\|\mathbf{r}(\mathbf{y})\|}{\gamma} := \frac{\|A\mathbf{y}-\vartheta\mathbf{y}\|}{\gamma},$$

where $\mathbf{r}(\mathbf{y},\vartheta) = A\mathbf{y}-\vartheta\mathbf{y}$ plays the role of a ***residual****.*

Proof. We write $\mathbf{y} = \mathbf{u}\cos\psi + \mathbf{v}\sin\psi$ with $\|\mathbf{v}\| = 1$. Then

$$\begin{aligned}\mathbf{r}(\mathbf{y},\vartheta) = A\mathbf{y}-\vartheta\mathbf{y} &= (A-\vartheta I)\mathbf{u}\cos\psi + (A-\vartheta I)\mathbf{v}\sin\psi,\\ &= (\lambda-\vartheta)\mathbf{u}\cos\psi + (A-\vartheta I)\mathbf{v}\sin\psi.\end{aligned}$$

Because $\mathbf{u}^*(A-\vartheta I)\mathbf{v} = 0$, Pythagoras' theorem implies

$$\|\mathbf{r}(\mathbf{y},\vartheta)\|^2 = (\lambda-\vartheta)^2\cos^2\psi + \|(A-\vartheta I)\mathbf{v}\|^2\sin^2\psi \ge \gamma^2\|\mathbf{v}\|^2\sin^2\psi. \qquad\blacksquare$$

Theorem 8.12 ([3, p.298]) *Let the assumptions of Theorem 8.5 be satisfied. Let $\mathbf{x}_j^{(k)} = X^{(k)}\mathbf{e}_j$ be the j-th Ritz vector as computed be Algorithm 8.3, and let $\mathbf{y}_i^{(k)} = U\begin{pmatrix} I \\ \mathbf{S}^{(k)}\end{pmatrix}\mathbf{e}_i$ (cf. the proof of Theorem 8.5). Then the following inequality holds*

$$\sin\,\angle(\mathbf{x}_i^{(k)},\mathbf{y}_i^{(k)}) \le c\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k, \qquad 1\le\ i\le p.$$

Proof. The columns of $U\begin{pmatrix} I_p \\ \mathbf{S}^{(k)}\end{pmatrix}$ form a basis of $\mathcal{R}(X^{(k)})$. Therefore, we can write

$$\mathbf{x}_i^{(k)} = U\begin{pmatrix} I_p \\ S^{(k)}\end{pmatrix}\mathbf{t}_i, \qquad \mathbf{t}_i\in\mathbb{F}^p.$$

Instead of the special eigenvalue problem

$$X^{(k-1)^*}A^{-2}X^{(k-1)}\mathbf{y} = H^{(k)}\mathbf{y} = \mu^{-2}\mathbf{y}$$

in the orthonormal 'basis' $X^{(k)}$ we consider the equivalent eigenvalue problem

$$\left[I_p, S^{(k)^*}\right]UA^{-2}U\begin{pmatrix} I_p \\ S^{(k)}\end{pmatrix}\mathbf{t} = \mu^{-2}\left[I_p, S^{(k)^*}\right]\begin{pmatrix} I_p \\ S^{(k)}\end{pmatrix}\mathbf{t}. \tag{8.19}$$

Let $(\mu,\mathbf{t})$ be an eigenpair of (8.19). Then we have

$$\begin{aligned} 0 &= \left[I_p, S^{(k)^*}\right]UA^{-2}U\begin{pmatrix} I_p \\ S^{(k)}\end{pmatrix}\mathbf{t} - \mu^{-2}\left[I_p, S^{(k)^*}\right]\begin{pmatrix} I_p \\ S^{(k)}\end{pmatrix}\mathbf{t}\\ &= \left(\Lambda_1^{-2} + S^{(k)^*}\mathbf{\Lambda}_2^{-2}S^{(k)}\right)\mathbf{t} - \mu^{-2}\left(I_p + S^{(k)^*}S^{(k)}\right)\mathbf{t},\\ &= \left((\Lambda_1^{-2}-\mu^{-2}I) + S^{(k)^*}(\Lambda_2^{-2}-\mu^{-2}I)S^{(k)}\right)\mathbf{t}\\ &= \left((\Lambda_1^{-2}-\mu^{-2}I) + \Lambda_1^kS^{(0)^*}\Lambda_2^{-k}(\Lambda_2^{-2}-\mu^{-2}I)\Lambda_2^{-k}S^{(0)}\Lambda_1^k\right)\mathbf{t}\\ &= \left((\Lambda_1^{-2}-\mu^{-2}I) + \left(\frac{1}{\lambda_{p+1}}\Lambda_1\right)^k H_k\left(\frac{1}{\lambda_{p+1}}\Lambda_1\right)^k\right)\mathbf{t}\end{aligned} \tag{8.20}$$

with

$$H_k = \lambda_{p+1}^{2k}S^{(0)^*}\Lambda_2^{-k}(\Lambda_2^{-2}-\mu^{-2}I)\Lambda_2^{-k}S^{(0)}.$$

As the largest eigenvalue of Λ_2^{-1} is $1/\lambda_{p+1}$, H_k is bounded,

$$\|H_k\| \le c_1 \qquad \forall k > 0.$$

Thus,

$$\left(\left(\frac{1}{\lambda_{p+1}}\Lambda_1\right)^k H_k \left(\frac{1}{\lambda_{p+1}}\Lambda_1\right)^k\right)\mathbf{t} \quad \xrightarrow[\lambda\to\infty]{} \quad 0.$$

Therefore, in (8.20) we can interpret this expression as an perturbation of the diagonal matrix $\Lambda_2^{-2} - \mu^{-2}I$. For sufficiently large k (that may depend on i) there is a μ_i that is close to λ_i, and a $\mathbf{t}_i$ that is close to $\mathbf{e}_i$. We now assume that k is so big that

$$|\mu_i^{-2} - \lambda_i^{-1}| \le \rho := \frac{1}{2} \min_{\lambda_j \ne \lambda_i} |\lambda_i^{-2} - \lambda_j^{-2}|$$

such that μ_i^{-2} is closer to λ_i^{-2} than to any other λ_j^{-2}, $j \ne i$.

We now consider the orthonormal 'basis'

$$B = \begin{pmatrix} I_p \\ S^{(k)} \end{pmatrix} \left(I_p + S^{(k)^*}S^{(k)}\right)^{-1/2}.$$

If $(\mu_i, \mathbf{t}_i)$ is an eigenpair of (8.19) or (8.20), respectively, then $\left(\mu_i^{-2}, \left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{t}_i\right)$ is an eigenpair of

$$B^* A^{-2} B\mathbf{t} = \mu^{-2}\mathbf{t}. \tag{8.21}$$

As, for sufficiently large k, $(\lambda_i^{-2}, \mathbf{e}_i)$ is a good approximation of the eigenpair $(\mu_i^{-2}, \mathbf{t}_i)$ of (8.20), then also $\left(\lambda_i^{-2}, \left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{e}_i\right)$ is a good approximation to the eigenpair $\left(\mu_i^{-2}, \left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{t}_i\right)$ of (8.21). We now apply Lemma 8.11 with

$$\begin{aligned}
\gamma = \rho, \qquad \vartheta = \lambda_i^{-2},& \\
\mathbf{y} = \left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{e}_i \Big/ &\left\|\left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{e}_i\right\|, \\
\mathbf{u} = \left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{t}_i \Big/ &\left\|\left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{t}_i\right\|.
\end{aligned}$$

Now we have

$$\begin{aligned}
\|\mathbf{r}(\mathbf{y})\| &= \left\|(B^* A^{-2^*} B - \lambda_i^{-2} I)\left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{e}_i\right\|\left\|\left(I_p + S^{(k)^*}S^{(k)}\right)^{1/2}\mathbf{e}_i\right\| \\
&\le \left\|\left(I_p + S^{(k)^*}S^{(k)}\right)^{-\frac{1}{2}}\left[\left[I_p, \mathbf{S}^{(k)^*}\right] U A^{-2} U \begin{pmatrix} I_p \\ S^{(k)} \end{pmatrix} - \frac{1}{\lambda_i^2}\left(I_p + S^{(k)^*}S^{(k)}\right)\right]\mathbf{e}_i\right\| \\
&\le \left\|\left(I_p + S^{(k)^*}S^{(k)}\right)^{-1/2}\right\| \left\|\left[\Lambda_1^{-2} - \lambda_i^{-2} I + \left(\lambda_{p+1}^{-1}\mathbf{\Lambda}_1\right)^k H_k \left(\lambda_{p+1}^{-1}\mathbf{\Lambda}_1\right)^k\right]\mathbf{e}_i\right\| \\
&\le \left\|\left(\lambda_{p+1}^{-1}\mathbf{\Lambda}_1\right)^k H_k \left(\lambda_{p+1}^{-1}\Lambda_1\right)^k \mathbf{e}_i\right\| \\
&\le \left\|\lambda_{p+1}^{-1}\Lambda_1\right\|^k \|H_k\| \left\|\left(\lambda_{p+1}^{-1}\Lambda_1\right)^k e_i\right\| \le c_1\left(\frac{\lambda_p}{\lambda_{p+1}}\right)^k\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k.
\end{aligned}$$

Then, Lemma 8.11 implies that

$$\sin\ \angle(x_i^{(k)}, \mathbf{y}_i^{(k)}) = \sin\ \angle\left(\left(I_p + S^{(k)*}S^{(k)}\right)^{1/2}\mathbf{t}_i, \left(I_p + S^{(k)*}S^{(k)}\right)^{1/2}\mathbf{e}_i\right) \le \frac{c_1}{\rho}\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k.$$

■

In the proof of Theorem 8.10 we showed that

$$\angle(\mathbf{u}_i, \mathbf{y}_i^{(k)}) \le c\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k.$$

In the previous theorem we showed that

$$\angle(\mathbf{x}_i^{(k)}, \mathbf{y}_i^{(k)}) \le c_1\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k.$$

By consequence,

$$\angle(\mathbf{x}_i^{(k)}, \mathbf{u}_i) \le c_2\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^k$$

must be true for a constant c_2 independent of k.

As earlier, for the eigenvalues we can show that

$$|\lambda_i - \lambda_i^{(k)}| \le c_3\left(\frac{\lambda_i}{\lambda_{p+1}}\right)^{2k}.$$

A numerical example

For the previous example that is concerned with the accustic vibration in the interior of a car the numbers listed in Table 8.2 are obtained. The quotients $\hat{\lambda}_i^2/\hat{\lambda}_{p+1}^2$, that determine the convergence behavior of the eigenvalues are

$$\begin{array}{ll} (\hat{\lambda}_1/\hat{\lambda}_6)^2 = 0.004513, & (\hat{\lambda}_4/\hat{\lambda}_6)^2 = 0.2045, \\ (\hat{\lambda}_2/\hat{\lambda}_6)^2 = 0.02357, & (\hat{\lambda}_5/\hat{\lambda}_6)^2 = 0.7321. \\ (\hat{\lambda}_3/\hat{\lambda}_6)^2 = 0.1362, & \end{array}$$

The numbers in the table confirm the improved convergence rate. The convergence rates of the first four eigenvalues have improved considerably. The predicted rates are not clearly visible, but they are approximated quite well. The convergence rate of the fifth eigenvalue has not improved. The convergence of the 5-dimensional *subspace* $\mathcal{R}([\mathbf{x}_1^{(k)}, \ldots, \mathbf{x}_5^{(k)}])$ to the searched space $\mathcal{R}([\mathbf{u}_1, \ldots, \mathbf{u}_5])$ has not been accelerated. Its convergence rate is still $\approx \lambda_5/\lambda_6$ according to Theorem 8.5

By means of the Rayleigh-Ritz step we have achieved that the columns $\mathbf{x}_i^{(k)} = \mathbf{x}^{(k)}$ converge in an optimal rate to the individual eigenvectors of A.

8.5 Relation between subspace iteration and QR algorithm

The connection between (simultaneous) vector iteration and the QR algorithm has been investigated by Parlett and Poole [4].

Let $X_0 = I_n$, the $n \times n$ identity matrix.

k	$\frac{\lambda_1^{(k-1)}-\lambda_1}{\lambda_1^{(k)}-\lambda_1}$	$\frac{\lambda_2^{(k-1)}-\lambda_2}{\lambda_2^{(k)}-\lambda_2}$	$\frac{\lambda_3^{(k-1)}-\lambda_3}{\lambda_3^{(k)}-\lambda_3}$	$\frac{\lambda_4^{(k-1)}-\lambda_4}{\lambda_4^{(k)}-\lambda_4}$	$\frac{\lambda_5^{(k-1)}-\lambda_5}{\lambda_5^{(k)}-\lambda_5}$
1	0.0001	0.0017	0.0048	0.0130	0.0133
2	0.0047	0.0162	0.2368	0.0515	0.2662
3	0.0058	0.0273	0.1934	0.1841	0.7883
4	0.0057	0.0294	0.1740	0.2458	0.9115
5	0.0061	0.0296	0.1688	0.2563	0.9195
6		0.0293	0.1667	0.2553	0.9066
7		0.0288	0.1646	0.2514	0.8880
8		0.0283	0.1620	0.2464	0.8675
9		0.0275	0.1588	0.2408	0.8466
10			0.1555	0.2351	0.8265
11			0.1521	0.2295	0.8082
12			0.1490	0.2245	0.7921
13			0.1462	0.2200	0.7786
14			0.1439	0.2163	0.7676
15			0.1420	0.2132	0.7589
16			0.1407	0.2108	0.7522
17			0.1461	0.2089	0.7471
18			0.1659	0.2075	0.7433
19			0.1324	0.2064	0.7405
20				0.2054	0.7384
21				0.2102	0.7370
22				0.2109	0.7359
23					0.7352
24					0.7347
25					0.7344
26					0.7343
27					0.7342
28					0.7341
29					0.7342
30					0.7342
31					0.7343
32					0.7343
33					0.7344
34					0.7345
35					0.7346
36					0.7347
37					0.7348
38					0.7348
39					0.7349
40					0.7350

Table 8.2: Example of accelerated basic subspace iteration.

Then we have

$$\begin{aligned} AI = A_0 = AX_0 = Y_1 &= X_1R_1 && (SVI)\\ A_1 = X_1^*AX_1 &= X_1^*X_1R_1X_1 = R_1X_1 && (QR)\\ AX_1 = Y_2 &= X_2R_2 && (SVI)\\ A_1 &= X_1^*Y_2 = X_1^*X_2R_2 && (QR)\\ A_2 &= R_2X_1^*X_2 && (QR)\\ &= X_2^*X_1\underbrace{X_1^*X_2R_2}_{A_1}X_1^*X_2 = X_2^*AX_2 && (QR) \end{aligned}$$

More generally, by induction, we have

$$\begin{aligned} AX_k &= Y_{k+1} = X_{k+1}R_{k+1} && (SVI)\\ A_k &= X_k^*AX_k = X_k^*Y_{k+1} = X_k^*X_{k+1}R_{k+1}\\ A_{k+1} &= R_{k+1}X_k^*X_{k+1} && (QR)\\ &= X_{k+1}^*\underbrace{X_k\underbrace{X_k^*X_{k+1}R_{k+1}}_{A_k}X_k^*}_{A}X_{k+1} = X_{k+1}^*AX_{k+1} && (QR) \end{aligned}$$

Relation to QR: $Q_1 = X_1$, $Q_k = X_k^*X_{k+1}$.

$$\begin{aligned} A^k &= A^kX_0 = A^{k-1}AX_0 = A^{k-1}X_1R_1\\ &= A^{k-2}AX_1R_1 = A^{k-2}X_2R_2R_1\\ &\vdots\\ &= X_k\underbrace{R_kR_{k-1}\cdots R_1}_{U_k} = X_kU_k && (QR) \end{aligned}$$

Because U_k is *upper triangular* we can write

$$A^k[\mathbf{e}_1,\ldots,\mathbf{e}_p] = X_kU_k[\mathbf{e}_1,\ldots,\mathbf{e}_p] = X_kU_k(:,1:p) = X_k(:,1:p)\begin{bmatrix} u_{11} & \cdots & u_{1p}\\ & \ddots & \vdots\\ & & u_{pp} \end{bmatrix}$$

This holds for all p. We therefore can interpret the QR algorithm as a **nested subspace iteration**. There is also a relation to simultaneous inverse vector iteration! Let us assume that A is invertible. Then we have,[1]

$$\begin{gathered} AX_{k-1} = X_{k-1}A_{k-1} = X_kR_k\\ X_kR_k^{-*} = A^{-*}X_{k-1}, \qquad R_k^{-*} \text{ is lower triangular}\\ X_k\underbrace{R_k^{-*}R_{k-1}^{-*}\cdots R_1^{-*}}_{U_k^{-*}} = \left(A^{-*}\right)^kX_0 \end{gathered}$$

[1] Notice that $A^{-*} = (A^{-1})^* = (A^*)^{-1}$.

Then,

$$X_k[\mathbf{e}_\ell,\ldots,\mathbf{e}_n]\begin{bmatrix}\bar{u}_{\ell,\ell} & & \\ \vdots & \ddots & \\ \bar{u}_{n,\ell} & & \bar{u}_{n,n}\end{bmatrix} = \left(A^{-*}\right)^k X_0[\mathbf{e}_\ell,\ldots,\mathbf{e}_n]$$

By consequence, the last $n-\ell+1$ columns of X_k execute a simultaneous *inverse* vector iteration. This holds for all ℓ. Therefore, the QR algorithm also performs a **nested inverse subspace iteration**.

8.6 Addendum

Let $A = H$ be an *irreducible* Hessenberg matrix and $W_1 = [\mathbf{w}_1,\ldots,\mathbf{w}_p]$ be a basis of the p-th dominant invariant subspace of H^*,

$$H^*W_1 = W_1S, \qquad S \text{ invertible.}$$

Notice that the p-th dominant invariant subspace is unique if $|\lambda_p| > |\lambda_{p+1}|$.

Let further $X_0 = [\mathbf{e}_1,\ldots,\mathbf{e}_p]$. Then we have the

Theorem 8.13 *$W_1^*X_0$ is nonsingular.*

Remark 8.3. If $W_1^*X_0$ is nonsingular then $W_k^*X_0$ is nonsingular for all $k > 0$. □

Proof. If $W_1^*X_0$ were singular then there was a vector $\mathbf{a} \in \mathbb{F}^p$ with $X_0^*W_1\mathbf{a} = \mathbf{0}$. Thus, $\mathbf{w} = W_1\mathbf{a}$ is orthogonal to $\mathbf{e}_1,\ldots,\mathbf{e}_p$. Therefore, the first p components of $\mathbf{w}$ are zero.

From, $H^*W_1 = W_1S$ we have that $(H^*)^k\mathbf{w} \in \mathcal{R}(W_1)$ for all k.

But we have

$$\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \times \\ \vdots \\ \times \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}0\\ \vdots\\ 0\end{matrix}}\right\} p \text{ zeros} \\ \\ \\ \\ \end{matrix} \qquad H^*\mathbf{w} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \times \\ \vdots \\ \times \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}0\\ \vdots\\ 0\end{matrix}}\right\} p-1 \text{ zeros} \\ \\ \\ \\ \end{matrix} \qquad (H^*)^k\mathbf{w} = \begin{bmatrix} \times \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \times \end{bmatrix}$$

These vectors evidently are linearly independent.

So, we have constructed $p+1$ linearly independent vectors $\mathbf{w},\ldots,(H^*)^p\mathbf{w}$ in the p-dimensional subspace $\mathcal{R}(W_1)$. This is a contradiction. ■

Bibliography

[1] Z. Bai and G. W. Stewart, *Algorithm 776. SRRIT — A FORTRAN subroutine to calculate the dominant invariant subspace of a nonsymmetric matrix*, ACM Trans. Math. Softw., 23 (1997), pp. 494–513.

[2] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[3] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980. (Republished by SIAM, Philadelphia, 1998.).

[4] B. N. Parlett and W. G. Poole, *A geometric theory for the QR, LU, and power iterations*, SIAM J. Numer. Anal., 8 (1973), pp. 389–412.

[5] H. Rutishauser, *Simultaneous iteration method for symmetric matrices*, Numer. Math., 16 (1970), pp. 205–223. Reprinted in: Linear Algebra, J.H. Wilkinson, C. Reinsch (eds.), pp. 284–301, Springer, Berlin, 1971.

Chapter 9

Krylov subspaces

9.1 Introduction

In the power method or in the inverse vector iteration we computed, up to normalization, sequences of the form

$$\mathbf{x}, A\mathbf{x}, A^2\mathbf{x}, \ldots$$

The information available at the k-th step of the iteration is the single vector $\mathbf{x}^{(k)} = A^k\mathbf{x}/\|A^k\mathbf{x}\|$. One can pose the question if discarding all the previous information $\left\{\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k-1)}\right\}$ is not a too big waste of information. This question is not trivial to answer. On one hand there is a big increase of memory requirement, on the other hand exploiting all the information computed up to a certain iteration step can give much better approximations to the searched solution. As an example, let us consider the symmetric matrix

$$T = \left(\frac{51}{\pi}\right)^2 \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix} \in \mathbb{R}^{50\times 50}.$$

the lowest eigenvalue of which is around 1. Let us choose $\mathbf{x} = [1, \ldots, 1]^*$ and compute the first three iterates of inverse vector iteration, $\mathbf{x}$, $T^{-1}\mathbf{x}$, and $T^{-2}\mathbf{x}$. We denote their Rayleigh quotients by $\rho^{(1)}$, $\rho^{(2)}$, and $\rho^{(3)}$, respectively. The Ritz values $\vartheta_j^{(k)}$, $1 \le j \le k$, obtained with the Rayleigh-Ritz procedure with $\mathcal{K}_k(\mathbf{x}) = \operatorname{span}(\mathbf{x}, T^{-1}\mathbf{x}, \ldots, T^{1-k}\mathbf{x})$, $k = 1, 2, 3$, are given in Table 9.1. The three smallest eigenvalues of T are 0.999684, 3.994943, and 8.974416. The approximation errors are thus $\rho^{(3)} - \lambda_1 \approx 0.000'14$ and $\vartheta_1^{(3)} - \lambda_1 \approx 0.000'009$, which is 15 times smaller.

k	$\rho^{(k)}$	$\vartheta_1^{(k)}$	$\vartheta_2^{(k)}$	$\vartheta_3^{(k)}$
1	10.541456	10.541456		
2	1.012822	1.009851	62.238885	
3	0.999822	0.999693	9.910156	147.211990

Table 9.1: Ritz values $\vartheta_j^{(k)}$ vs. Rayleigh quotients $\rho^{(k)}$ of inverse vector iterates.

These results immediately show that the cost of three matrix vector multiplications can be much better exploited than with (inverse) vector iteration. We will consider in this section a kind of space that is very often used in the iterative solution of linear systems as well as of eigenvalue problems.

9.2 Definition and basic properties

Definition 9.1 The matrix

$$K^m(\mathbf{x}) = K^m(\mathbf{x}, A) := [\mathbf{x}, A\mathbf{x}, \ldots, \mathbf{A}^{(m-1)}\mathbf{x}] \in \mathbb{F}^{n\times m} \tag{9.1}$$

generated by the vector $\mathbf{x} \in \mathbb{F}^n$ is called a **Krylov matrix**. Its columns span the **Krylov (sub)space**

$$\mathcal{K}^m(\mathbf{x}) = \mathcal{K}^m(\mathbf{x}, A) := \operatorname{span}\left\{\mathbf{x}, A\mathbf{x}, \mathbf{A}^2\mathbf{x}, \ldots, A^{(m-1)}\mathbf{x}\right\} = \mathcal{R}\left(K^m(\mathbf{x})\right) \subset \mathbb{F}^n. \tag{9.2}$$

The **Arnoldi** and **Lanczos algorithms** are methods to compute an orthonormal basis of the Krylov space. Let

$$\left[\mathbf{x}, A\mathbf{x}, \ldots, A^{k-1}\mathbf{x}\right] = \mathbf{Q}^{(k)}\mathbf{R}^{(k)}$$

be the QR factorization of the Krylov matrix $K^m(\mathbf{x})$. The Ritz values and Ritz vectors of A in this space are obtained by means of the $k \times k$ eigenvalue problem

$$\mathbf{Q}^{(k)^*}A\mathbf{Q}^{(k)}\mathbf{y} = \vartheta^{(k)}\mathbf{y}. \tag{9.3}$$

If $(\vartheta_j^{(k)}, \mathbf{y}_j)$ is an eigenpair of (9.3) then $(\vartheta_j^{(k)}, \mathbf{Q}^{(k)}\mathbf{y}_j)$ is a Ritz pair of A in $K^m(\mathbf{x})$.

The following properties of Krylov spaces are easy to verify [1, p.238]

1. *Scaling.* $\mathcal{K}^m(\mathbf{x}, A) = \mathcal{K}^m(\alpha\mathbf{x}, \beta A), \quad \alpha, \beta \neq 0.$
2. *Translation.* $\mathcal{K}^m(\mathbf{x}, A - \sigma\mathbf{I}) = \mathcal{K}^m(\mathbf{x}, A)$.
3. *Change of basis.* If U is unitary then $U\mathcal{K}^m(U^*\mathbf{x}, U^*AU) = \mathcal{K}^m(\mathbf{x}, A)$.

   In fact,

$$\begin{aligned} K^m(\mathbf{x}, A) &= [\mathbf{x}, A\mathbf{x}, \ldots, A^{(m-1)}\mathbf{x}] \\ &= U[U^*\mathbf{x}, (U^*AU)U^*\mathbf{x}, \ldots, (U^*AU)^{m-1}U^*\mathbf{x}], \\ &= UK^m(U^*\mathbf{x}, U^*AU). \end{aligned}$$

Notice that the scaling and translation invariance hold only for the Krylov subspace, not for the Krylov matrices.

What is the dimension of $\mathcal{K}^m(\mathbf{x})$? It is evident that for $n \times n$ matrices A the columns of the Krylov matrix $K^{n+1}(\mathbf{x})$ are linearly dependent. (A subspace of $\mathbb{F}^n$ cannot have a dimension bigger than n.) On the other hand if $\mathbf{u}$ is an eigenvector corresponding to the eigenvalue λ then $A\mathbf{u} = \lambda\mathbf{u}$ and, by consequence, $\mathcal{K}^2(\mathbf{u}) = \operatorname{span}\{\mathbf{u}, A\mathbf{u}\} = \operatorname{span}\{\mathbf{u}\} = \mathcal{K}^1(\mathbf{u})$. So, there is a smallest m, $1 \le m \le n$, *depending on* $\mathbf{x}$ such that

$$\mathcal{K}^1(\mathbf{x}) \subsetneqq \mathcal{K}^2(\mathbf{x}) \subsetneqq \cdots \subsetneqq \mathcal{K}^m(\mathbf{x}) = \mathcal{K}^{m+1}(\mathbf{x}) = \cdots$$

For this number m,

$$K_{m+1}(\mathbf{x}) = [\mathbf{x}, A\mathbf{x}, \ldots, A^m\mathbf{x}] \in \mathbb{F}^{n\times m+1} \tag{9.4}$$

has linearly dependant columns, i.e., there is a *nonzero* vector $\mathbf{a} \in \mathbb{F}^{m+1}$ such that

$$K_{m+1}(\mathbf{x})\mathbf{a} = p(A)\mathbf{x} = \mathbf{0}, \qquad p(\lambda) = a_0 + a_1\lambda + \cdots + a_m\lambda^m. \tag{9.5}$$

The polynomial $p(\lambda)$ is called the minimal polynomial of A relativ to $\mathbf{x}$. By construction, the highest order coefficient $a_m \neq 0$.

If A is *diagonalizable*, then the degree of the minimal polynomial relativ to $\mathbf{x}$ has a simple geometric meaning (which does not mean that it is easily checked). Let

$$\mathbf{x} = \sum_{i=1}^{m} \mathbf{u}_i = [\mathbf{u}_1, \ldots, \mathbf{u}_m] \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix},$$

where the $\mathbf{u}_i$ are eigenvectors of A, $A\mathbf{u}_i = \lambda_i\mathbf{u}_i$, and $\lambda_i \neq \lambda_j$ for $i \neq j$. Notice that we have arranged the eigenvectors such that the coefficients in the above sum are all unity. Now we have

$$A^k\mathbf{x} = \sum_{i=1}^{m} \lambda_i^k \mathbf{u}_i = [\mathbf{u}_1, \ldots, \mathbf{u}_m] \begin{pmatrix} \lambda_1^k \\ \vdots \\ \lambda_m^k \end{pmatrix},$$

and, by consequence,

$$K^j(\mathbf{x}) = \underbrace{[\mathbf{u}_1, \ldots, \mathbf{u}_m]}_{\in \mathbb{C}^{n\times m}} \underbrace{\begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 & \cdots & \lambda_1^{j-1} \\ 1 & \lambda_2 & \lambda_2^2 & \cdots & \lambda_2^{j-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_m & \lambda_m^2 & \cdots & \lambda_m^{j-1} \end{bmatrix}}_{\in \mathbb{C}^{m\times j}}.$$

Since matrices of the form

$$\begin{bmatrix} 1 & \lambda_1 & \cdots & \lambda_1^{s-1} \\ 1 & \lambda_2 & \cdots & \lambda_2^{s-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \lambda_s & \cdots & \lambda_m^{s-1} \end{bmatrix} \in \mathbb{F}^{s\times s}, \quad \lambda_i \neq \lambda_j \text{ for } i \neq j,$$

so-called *Vandermonde matrices*, are *nonsingular* if the λ_i are different (their determinant equals $\prod_{i\neq j}(\lambda_i - \lambda_j)$) the Krylov matrices $K^j(\mathbf{x})$ are nonsingular for $j \leq m$. Thus for diagonalizable matrices A we have

$$\dim \mathcal{K}^j(\mathbf{x}, A) = \min\{j, m\}$$

where m is the number of eigenvectors needed to represent $\mathbf{x}$. The subspace $\mathcal{K}^m(\mathbf{x})$ is the smallest invariant space that contains $\mathbf{x}$.

9.3 Polynomial representation of Krylov subspaces

In this section we assume A to be Hermitian. Let $\mathbf{s} \in \mathcal{K}^j(\mathbf{x})$. Then

$$\mathbf{s} = \sum_{i=0}^{j-1} c_i A^i \mathbf{x} = \pi(A)\mathbf{x}, \qquad \pi(\xi) = \sum_{i=0}^{j-1} c_i \xi^i. \tag{9.6}$$

Let $\mathbb{P}_j$ be the space of polynomials of degree $\leq j$. Then (9.6) becomes

$$\mathcal{K}^j(\mathbf{x}) = \{\pi(A)\mathbf{x} \mid \pi \in \mathbb{P}_{j-1}\}\,. \tag{9.7}$$

Let m be the smallest index for which $\mathcal{K}^m(\mathbf{x}) = \mathcal{K}^{m+1}(\mathbf{x})$. Then, for $j \leq m$ the mapping

$$\mathcal{P}^{j-1} \ni \sum c_i \xi^i \to \sum c_i A^i \mathbf{x} \in \mathcal{K}^j(\mathbf{x})$$

is bijective, while it is only surjective for $j > m$.

Let $Q \in \mathbb{F}^{n\times j}$ be a matrix with orthonormal columns that span $\mathcal{K}^j(\mathbf{x})$, and let $A' = Q^*AQ$. The spectral decomposition

$$A'X' = X'\Theta, \qquad X'^*X' = I, \quad \Theta = \operatorname{diag}(\vartheta_i, \ldots, \vartheta_j),$$

of A' provides the Ritz values of A in $\mathcal{K}^j(\mathbf{x})$. The columns $\mathbf{y}_i$ of $Y = QX'$ are the Ritz vectors.

By construction the Ritz vectors are mutually orthogonal. Furthermore,

$$A\mathbf{y}_i - \vartheta_i \mathbf{y}_i \perp \mathcal{K}^j(\mathbf{x}) \tag{9.8}$$

because

$$Q^*(AQ\mathbf{x}'_i - Q\mathbf{x}'_i\vartheta_i) = Q^*AQ\mathbf{x}'_i - \mathbf{x}'_i\vartheta_i = A'\mathbf{x}'_i - \mathbf{x}'_i\vartheta_i = \mathbf{0}.$$

It is easy to represent a vector in $\mathcal{K}^j(\mathbf{x})$ that is orthogonal to $\mathbf{y}_i$.

Lemma 9.2 *Let $(\vartheta_i, \mathbf{y}_i)$, $1 \leq i \leq j$ be Ritz values and Ritz vectors of A in $\mathcal{K}^j(\mathbf{x}), j \leq m$. Let $\omega \in \mathbb{P}_{j-1}$. Then*

$$\omega(A)\mathbf{x} \perp \mathbf{y}_k \quad \Longleftrightarrow \quad \omega(\vartheta_k) = 0. \tag{9.9}$$

Proof. "$\Longleftarrow$" Let first $\omega \in \mathbb{P}_j$ with $\omega(x) = (x - \vartheta_k)\pi(x), \pi \in \mathbb{P}_{j-1}$. Then

$$\begin{aligned} \mathbf{y}_k^*\omega(A)\mathbf{x} &= \mathbf{y}_k^*(A - \vartheta_k\mathbf{I})\pi(A)\mathbf{x}, \qquad \text{here we use that } A = A^* \\ &= (A\mathbf{y}_k - \vartheta_k\mathbf{y}_k)^*\pi(A)\mathbf{x} \overset{(9.8)}{=} 0. \end{aligned} \tag{9.10}$$

"$\Longrightarrow$" Let $\mathcal{S}_k \subset \mathcal{K}^j(\mathbf{x})$ be defined by

$$\mathcal{S}_k := (A - \vartheta_k I)\,\mathcal{K}^{j-1}(\mathbf{x}) = \{\tau(A)\mathbf{x} \mid \tau \in \mathbb{P}_{j-1}, \tau(\vartheta_k) = 0\}\,.$$

Each polynomial τ in the definition of $\mathcal{S}_k$ has the form $\tau(\vartheta_k) = (x - \vartheta_k)\psi(x)$, $\psi \in \mathbb{P}_{j-2}$. Therefore, $\dim(\mathcal{S}_k) = \dim(\mathbb{P}_{j-2}) = j - 1$. As the dimension of a subspace of $\mathcal{K}^j(\mathbf{x})$ that is orthogonal to $\mathbf{y}_k$ is $j-1$, it must coincide with $\mathcal{S}_k$. ■

Next we define the polynomials

$$\mu(\xi) := \prod_{i=1}^{j}(\xi - \vartheta_i) \in \mathbb{P}_j, \quad \pi_k(\xi) := \frac{\mu(\xi)}{(\xi - \vartheta_k)} = \prod_{\substack{i=1\\ i\neq k}}^{j}(\xi - \vartheta_i) \in \mathbb{P}_{j-1}.$$

Then the Ritz vector $\mathbf{y}_k$ can be represented in the form

$$\mathbf{y}_k = \frac{\pi_k(A)\mathbf{x}}{\|\pi_k(A)\mathbf{x}\|}, \tag{9.11}$$

as $\pi_k(\xi) = 0$ for all $\vartheta_i, i \neq k$. According to Lemma 9.2 $\pi_k(A)\mathbf{x}$ is perpendicular to all $\mathbf{y}_i$ with $i \neq k$. Further,

$$\beta_j := \|\mu(A)\mathbf{x}\| = \min\{\|\omega(\mathbf{A})\mathbf{x}\| \mid \omega \in \mathbb{P}_j \text{ monic}\}. \tag{9.12}$$

(A polynomial in $\mathbb{P}_j$ is monic if its highest coefficients $a_j = 1$.) By the first part of Lemma 9.2 $\mu(A)\mathbf{x} \in \mathcal{K}^{j+1}(\mathbf{x})$ is orthogonal to $\mathcal{K}^j(\mathbf{x})$. As each monic $\omega \in \mathbb{P}_j$ can be written in the form

$$\omega(\xi) = \mu(\xi) + \psi(\xi), \qquad \psi \in \mathbb{P}_{j-1},$$

we have

$$\|\omega(A)\mathbf{x}\|^2 = \|\mu(A)\mathbf{x}\|^2 + \|\psi(A)\mathbf{x}\|^2,$$

as $\psi(A)\mathbf{x} \in \mathcal{K}^j(\mathbf{x})$. Because of property (9.12) μ is called the *minimal polynomial* of $\mathbf{x}$ of degree j. (In (9.5) we constructed the minimal polynomial of degree m in which case $\beta_m = 0$.)

Let $\mathbf{u}_1, \cdots, \mathbf{u}_m$ be the eigenvectors of A corresponding to $\lambda_1 < \cdots < \lambda_m$ that span $\mathcal{K}^m(\mathbf{x})$. We collect the first i of them in the matrix $U_i := [\mathbf{u}_1, \ldots, \mathbf{u}_i]$. Let $\|\mathbf{x}\| = 1$. Let $\varphi := \angle(\mathbf{x}, \mathbf{u}_i)$ and $\psi := \angle(\mathbf{x}, U_iU_i^*\mathbf{x})$ $(\leq \varphi)$. (Remember that $U_iU_i^*\mathbf{x}$ is the orthogonal projection of $\mathbf{x}$ on $\mathcal{R}(U_i)$.)

Let

$$\mathbf{g} := \frac{U_iU_i^*\mathbf{x}}{\|U_iU_i^*\mathbf{x}\|} \qquad \text{and} \qquad \mathbf{h} := \frac{(I - U_iU_i^*)\mathbf{x}}{\|(I - U_iU_i^*)\mathbf{x}\|}.$$

Then we have

$$\|U_iU_i^*\mathbf{x}\| = \cos\psi, \qquad \|(\mathbf{I} - U_iU_i^*)\mathbf{x}\| = \sin\psi.$$

The following Lemma will be used for the estimation of the difference $\vartheta_i^{(j)} - \lambda_i$ of the desired eigenvalue and its approximation from the Krylov subspace.

Lemma 9.3 ([1, p.241]) *For each $\pi \in \mathbb{P}_{j-1}$ and each $i \leq j \leq m$ the Rayleigh quotient*

$$\rho(\pi(A)\mathbf{x}; A - \lambda_i I) = \frac{(\pi(A)\mathbf{x})^*(A - \lambda_i I)(\pi(A)\mathbf{x})}{\|\pi(A)\mathbf{x}\|^2} = \rho(\pi(A)\mathbf{x}; A) - \lambda_i$$

satisfies the inequality

$$\rho(\pi(A)\mathbf{x}; A - \lambda_i I) \leq (\lambda_m - \lambda_i)\left[\frac{\sin\psi}{\cos\varphi}\,\frac{\|\pi(A)\mathbf{h}\|}{\pi(\lambda_i)}\right]^2. \tag{9.13}$$

Proof. With the definitions of $\mathbf{g}$ and $\mathbf{h}$ from above we have

$$\mathbf{x} = U_iU_i^*\mathbf{x} + (I - U_iU_i^*)\mathbf{x} = \cos\psi\,\mathbf{g} + \sin\psi\,\mathbf{h}.$$

which is an orthogonal decomposition. As $\mathcal{R}(U_i)$ is *invariant* under A,

$$\mathbf{s} := \pi(A)\mathbf{x} = \cos\psi\ \pi(A)\mathbf{g} + \sin\psi\ \pi(A)\mathbf{h}$$

is an orthogonal decomposition of $\mathbf{s}$. Thus,

$$\rho(\pi(A)\mathbf{x}; A - \lambda_i I) = \frac{\cos^2\psi\,\mathbf{g}^*(A - \lambda_i I)\pi^2(A)\mathbf{g} + \sin^2\psi\,\mathbf{h}^*(A - \lambda_i I)\pi^2(A)\mathbf{h}}{\|\pi(A)\mathbf{x}\|^2}. \tag{9.14}$$

Since $\lambda_1 < \lambda_2 < \cdots < \lambda_m$, we have

(i) $\mathbf{v}^*(A-\lambda_i I)\mathbf{v} \le 0$ for all $\mathbf{v} \in \mathcal{R}(U_i)$,

(ii) $\mathbf{w}^*(A-\lambda_i I)\mathbf{w} \le (\lambda_m - \lambda_i)\|\mathbf{w}\|^2$ for all $\mathbf{w} \in \mathcal{R}(U_i)^\perp$.

Setting $\mathbf{v} = \pi(A)\mathbf{g}$ and $\mathbf{w} = \pi(A)\mathbf{h}$ we obtain from (9.14)

$$\rho(\mathbf{s}; A-\lambda_i I) \le \sin^2\psi\,(\lambda_m-\lambda_i)\frac{\|\pi(A)\mathbf{h}\|^2}{\|\pi(A)\mathbf{x}\|^2}.$$

With

$$\|\mathbf{s}\|^2 = \|\pi(A)\mathbf{x}\|^2 = \sum_{l=1}^{m} \pi^2(\lambda_l)(\mathbf{x}^*\mathbf{u}_l)^2 \ge \pi^2(\lambda_i)\cos^2\varphi$$

we obtain the claim. ■

9.4 Error bounds of Saad

The error bounds to be presented have been published by Saad [2]. We follow the presentation in Parlett [1]. The error bounds for $\vartheta_i^{(j)} - \lambda_i$ are obtained by carefully selecting the polynomial π in Lemma 9.3. Of course we would like $\pi(A)$ to be as small as possible and $\pi(\lambda_i)$ to be as large as possible. First, by the definition of $\mathbf{h}$, we have

$$\begin{aligned}\|\pi(A)\mathbf{h}\|^2 &= \frac{\|\pi(A)(I-U_iU_i^*)\mathbf{x}\|^2}{\|(I-U_iU_i^*)\mathbf{x}\|^2} = \frac{\|\pi(A)\sum_{l=i+1}^m(\mathbf{u}_l^*\mathbf{x})\mathbf{u}_l\|^2}{\|\sum_{l=i+1}^m(\mathbf{u}_l^*\mathbf{x})\mathbf{u}_l\|^2}\\ &= \frac{\sum_{l=i+1}^m(\mathbf{u}_l^*\mathbf{x})^2\pi^2(\lambda_l)}{\sum_{l=i+1}^m(\mathbf{u}_l^*\mathbf{x})^2} \le \max_{i<l\le m}\pi^2(\lambda_l) \le \max_{\lambda_{i+1}\le\lambda\le\lambda_m}\pi^2(\lambda).\end{aligned}$$

The last inequality is important! In this step the search of a maximum in a few selected points $(\lambda_{i+1},\ldots,\lambda_m)$ is replaced by a search of a maximum in a *whole interval* containing these points. Notice that λ_i is *outside* of this interval. Among all polynomials of a given degree that take a given fixed value $\pi(\lambda_i)$ the Chebyshev polynomial have the smallest maximum. As $\vartheta_i^{(j)}$ is a Ritz value, we know from the monotonicity principle 2.32 that

$$0 \le \vartheta_i^{(j)} - \lambda_i.$$

Further, from the definition of $\vartheta_i^{(j)}$ (as an eigenvalue of A in the subspace $\mathcal{K}^j(\mathbf{x})$),

$$\vartheta_i^{(j)} - \lambda_i \le \rho(\mathbf{s}, A-\lambda_i I) \quad \text{provided that } \mathbf{s}\perp\mathbf{y}_l,\ 1\le l\le i-1.$$

According to Lemma 9.2 $\mathbf{s} = \pi(A)\mathbf{x}$ is orthogonal on $\mathbf{y}_1,\ldots,\mathbf{y}_{i-1}$, if π has the form

$$\pi(\xi) = (\xi-\vartheta_1^{(j)})\cdots(\xi-\vartheta_{i-1}^{(j)})\omega(\xi), \quad \omega\in\mathbb{P}_{j-i}.$$

With this choice of π we get

$$\frac{\|\pi(A)\mathbf{h}\|}{\pi(\lambda_i)} \le \frac{\|(A-\vartheta_1^{(j)}I)\cdots(A-\vartheta_{i-1}^{(j)}I)\|\cdot\|\omega(A)\mathbf{h}\|}{|(\lambda_i-\vartheta_1^{(j)})|\cdots|(\lambda_i-\vartheta_{i-1}^{(j)})|\cdot|\omega(\lambda_i)|} \le \prod_{l=1}^{i-1}\frac{\lambda_m-\vartheta_l^{(j)}}{\lambda_i-\vartheta_l^{(j)}}\max_{\lambda_{i+1}\le\lambda\le\lambda_m}\frac{\omega(\lambda)}{\omega(\lambda_i)}.$$

This expression should be as small as possible. Now we have

$$\begin{aligned}\min_{\omega\in\mathbb{P}_{j-1}}\max_{\lambda_{i+1}\le\lambda\le\lambda_m}\frac{|\omega(\lambda)|}{|\omega(\lambda_i)|} &= \frac{\max\limits_{\lambda_{i+1}\le\lambda\le\lambda_m} T_{j-i}(\lambda;[\lambda_{i+1},\lambda_m])}{T_{j-i}(\lambda_i;[\lambda_{i+1},\lambda_m])}\\ &= \frac{1}{T_{j-i}(\lambda_i;[\lambda_{i+1},\lambda_m])}\\ &= \frac{1}{T_{j-i}(1+2\gamma)}, \qquad \gamma=\frac{\lambda_{i+1}-\lambda_i}{\lambda_m-\lambda_{i+1}}.\end{aligned}$$

$T_{j-i}(1+2\gamma)$ is the value of the Chebyshev polynomial corresponding to the normal interval $[-1,\ 1]$. The point $1+2\gamma$ is obtained if the affine transformation

$$[\lambda_{i+1},\lambda_m]\ni\lambda\longrightarrow\frac{2\lambda-\lambda_{i+1}-\lambda_m}{\lambda_i-\lambda_{i+1}}\in[-1,\ 1]$$

is applied to λ_i.

Thus we have proved the first part of the following

Theorem 9.4 [2] *Let $\vartheta_1^{(j)},\ldots,\vartheta_j^{(j)}$ be the Ritz values of A in $\mathcal{K}^j(\mathbf{x})$ and let $(\lambda_l,\mathbf{u}_l)$, $l=1,\ldots,m$, be the eigenpairs of A (in $\mathcal{K}^m(\mathbf{x})$). Then for all $i\le j$ we have*

$$0\le\vartheta_i^{(j)}-\lambda_i\le(\lambda_m-\lambda_i)\left[\frac{\sin\psi}{\cos\varphi}\cdot\frac{\prod_{l=1}^{i-1}\frac{\lambda_m-\vartheta_l^{(j)}}{\lambda_i-\vartheta_l^{(j)}}}{T_{j-i}(1+2\gamma)}\right]^2, \qquad \gamma=\frac{\lambda_{i+1}-\lambda_i}{\lambda_m-\lambda_{i+1}}, \tag{9.15}$$

and

$$\tan\angle(\mathbf{u}_i,\text{projektion of }\mathbf{u}_i\text{ on }\mathcal{K}^j)\le\frac{\sin\psi}{\cos\varphi}\cdot\frac{\prod_{l=1}^{i-1}\frac{\lambda_m-\lambda_l}{\lambda_i-\lambda_l}}{T_{j-i}(1+2\gamma)}. \tag{9.16}$$

Proof. For proving the second part of the Theorem we write

$$\mathbf{x}=\mathbf{g}\cos\angle(\mathbf{x},U_{i-1}U_{i-1}^*\mathbf{x})+\mathbf{u}_i\cos\underbrace{\angle(\mathbf{x},\mathbf{u}_i)}_{\varphi}+\mathbf{h}\sin\underbrace{\angle(\mathbf{x},U_iU_i^*\mathbf{x})}_{\psi}.$$

We choose π such that $\pi(\lambda_1)=\cdots=\pi(\lambda_{i-1})=0$. Then.

$$\mathbf{s}=\pi(A)\mathbf{x}=\pi(\lambda_i)\mathbf{u}_i\cos\varphi+\pi(A)\mathbf{h}\sin\psi$$

is an orthogonal decomposition of $\mathbf{s}$. By consequence,

$$\tan\angle(\mathbf{s},\mathbf{u}_i)=\frac{\sin\psi\ \|\pi(A)\mathbf{h}\|}{\cos\varphi\ |\pi(\lambda_i)|}.$$

The rest is similar as above. ■

Remark 9.1. Theorem 9.4 does *not* give bounds for the angle between $\angle(\mathbf{u}_i,\mathbf{y}_i)$, an angle that would be more interesting than the abstract angle between $\mathbf{u}_i$ and its projection on $\mathcal{K}^j(\mathbf{x})$. It is possible however to show that [1, p. 246]

$$\sin\angle(\mathbf{u}_i,\mathbf{y}_i)\le\sqrt{1+\frac{\beta_j^2}{\gamma_i^{(j)2}}}\ \sin\angle(\mathbf{u}_i,\text{projection of }\mathbf{u}_i\text{ onto }\mathcal{K}^j)$$

β_j is the number that appeared earlier in the discussion after Lemma 9.2, and

$$\gamma_i^{(j)} = \min_{s \neq i} |\lambda_i - \vartheta_s^{(j)}|$$

□

Theorem 9.4 can easily be rewritten to give error bounds for $\lambda_m - \vartheta_j^{(j)}$, $\lambda_{m-1} - \vartheta_{j-1}^{(j)}$, etc.

We see from this Theorem that the eigenvalues at the beginning and at the end of the spectrum are approximated the quickest. For the first eigenvalue the bound (9.15) simplifies a little,

$$(9.17) \quad 0 \leq \vartheta_1^{(j)} - \lambda_1 \leq (\lambda_m - \lambda_1)\,\frac{\tan^2\varphi_1}{T_{j-i}(1+2\gamma_1)^2}, \qquad \gamma_1 = \frac{\lambda_2 - \lambda_1}{\lambda_m - \lambda_2}, \quad \varphi_1 = \angle(\mathbf{x}, \mathbf{u}_1).$$

Analogously, for the largest eigenvalue we have

$$(9.18) \qquad 0 \leq \lambda_m - \vartheta_j^{(j)} \leq (\lambda_m - \lambda_1)\ \tan^2\varphi_m\,\frac{1}{T_{j-i}(1+2\gamma_m)^2},$$

with

$$\gamma_m = \frac{\lambda_m - \lambda_{m-1}}{\lambda_{m-1} - \lambda_1}, \quad \text{and} \quad \cos\varphi_m = \mathbf{x}^*\mathbf{u}_m.$$

If the Lanczos algorithmus is applied with $(A - \sigma I)^{-1}$ as with the shifted and inverted vector iteration then we form Krylov spaces $\mathcal{K}^j(\mathbf{x}, (A-\sigma I)^{-1})$. Here the largest eigenvalues are $\frac{1}{\hat\lambda_1} \geq \frac{1}{\hat\lambda_2} \geq \cdots \geq \frac{1}{\hat\lambda_j}$, $\hat\lambda_i = \lambda_i - \sigma$.

Eq. (9.18) then becomes

$$0 \leq \frac{1}{\hat\lambda_1} - \frac{1}{\hat\vartheta_j^{(j)}} \leq (\frac{1}{\hat\lambda_1} - \frac{1}{\hat\lambda_j})\,\frac{\tan^2\varphi_1}{T_{j-1}(1+2\hat\gamma_1)^2}, \qquad \hat\gamma_1 = \frac{\frac{1}{\hat\lambda_1} - \frac{1}{\hat\lambda_2}}{\frac{1}{\hat\lambda_2} - \frac{1}{\hat\lambda_j}}.$$

Now, we have

$$1 + 2\hat\gamma_1 = 2(1+\hat\gamma_1) - 1 = 2\left(\frac{\frac{1}{\hat\lambda_1} - \frac{1}{\hat\lambda_j}}{\frac{1}{\hat\lambda_2} - \frac{1}{\hat\lambda_j}}\right) - 1 = 2\frac{\hat\lambda_2}{\hat\lambda_1}\underbrace{\left(\frac{1 - \frac{\hat\lambda_1}{\hat\lambda_j}}{1 - \frac{\hat\lambda_2}{\hat\lambda_j}}\right)}_{>1} - 1 \geq 2\frac{\hat\lambda_2}{\hat\lambda_1} - 1 > 1.$$

Since $|T_{j-1}(\xi)|$ grows rapidly and monotonically outside $[-1, 1]$ we have

$$T_{j-1}(1+2\hat\gamma_1) \geq T_{j-1}(2\frac{\hat\lambda_2}{\hat\lambda_1} - 1),$$

and thus

$$(9.19) \qquad \frac{1}{\hat\lambda_1} - \frac{1}{\hat\vartheta_1^{(j)}} \leq c_1\left(\frac{1}{T_{j-1}(2\frac{\hat\lambda_2}{\hat\lambda_1} - 1)}\right)^2$$

With the simple inverse vector iteration we had

$$(9.20) \qquad \frac{1}{\hat\lambda_1} - \frac{1}{\hat\lambda_1^{(j)}} \leq c_2\left(\frac{\hat\lambda_1}{\hat\lambda_2}\right)^{2(j-1)}$$

In Table 9.2 the numbers

$$\left(\frac{1}{T_{j-1}(2\frac{\hat\lambda_2}{\hat\lambda_1}-1)}\right)^2$$

are compared with

$$\left(\frac{\hat\lambda_1}{\hat\lambda_2}\right)^{2(j-1)}$$

for $\hat\lambda_2/\hat\lambda_1 = 2,\ 1.1,\ 1.01$. If this ratio is large both methods quickly provide the desired results. If however the ratio tends to 1 then a method that computes the eigenvalues by means of Ritz values of Krylov spaces shows an acceptable convergence behaviour whereas vector iteration hardly improves with j. Remember that j is the number of matrix-vector multiplications have been executed, or, with the shift-and-invert spectral transformation, how many systems of equations have been solved.

$\hat\lambda_2/\hat\lambda_1$	$j = 5$	$j = 10$	$j = 15$	$j = 20$	$j = 25$
2.0	$\frac{3.0036e-06}{3.9063e-03}$	$\frac{6.6395e-14}{3.8147e-06}$	$\frac{1.4676e-21}{3.7253e-09}$	$\frac{3.2442e-29}{3.6380e-12}$	$\frac{7.1712e-37}{3.5527e-15}$
1.1	$\frac{2.7152e-02}{4.6651e-01}$	$\frac{5.4557e-05}{1.7986e-01}$	$\frac{1.0814e-07}{6.9343e-02}$	$\frac{2.1434e-10}{2.6735e-02}$	$\frac{4.2482e-13}{1.0307e-02}$
1.01	$\frac{5.6004e-01}{9.2348e-01}$	$\frac{1.0415e-01}{8.3602e-01}$	$\frac{1.4819e-02}{7.5684e-01}$	$\frac{2.0252e-03}{6.8515e-01}$	$\frac{2.7523e-04}{6.2026e-01}$

Table 9.2: Ratio $\dfrac{(1/T_{j-1}(2\hat\lambda_2/\hat\lambda_1-1))^2}{(\hat\lambda_1/\hat\lambda_2)^{2(j-1)}}$ for varying j and ratios $\hat\lambda_2/\hat\lambda_1$.

Bibliography

[1] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980. (Republished by SIAM, Philadelphia, 1998.).

[2] Y. Saad, *On the rates of convergence of the Lanczos and the block Lanczos methods*, SIAM J. Numer. Anal., 17 (1980), pp. 687–706.

Chapter 10

Arnoldi and Lanczos algorithms

10.1 An orthonormal basis for the Krylov space $\mathcal{K}^j(\mathbf{x})$

The natural basis of the Krylov subspace $\mathcal{K}^j(\mathbf{x}) = \mathcal{K}^j(\mathbf{x}, A)$ is evidently $\{\mathbf{x}, A\mathbf{x}, \ldots, A^{j-1}\mathbf{x}\}$. Remember that the vectors $A^k\mathbf{x}$ converge to the direction of the eigenvector corresponding to the largest eigenvalue (in modulus) of A. Thus, this basis tends to be badly conditioned with increasing dimension j. Therefore, the straightforward procedure, the **Gram–Schmidt orthogonalization process**, is applied to the basis vectors in their natural order.

Suppose that $\{\mathbf{q}_1, \ldots, \mathbf{q}_i\}$ is the orthonormal basis for $\mathcal{K}^i(\mathbf{x})$, where $i \leq j$. We construct the vector $\mathbf{q}_{j+1}$ by first orthogonalizing $A^j\mathbf{x}$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$,

$$\mathbf{y}_j := A^j\mathbf{x} - \sum_{i=1}^{j} \mathbf{q}_i\mathbf{q}_i^* A^j\mathbf{x}, \tag{10.1}$$

and then normalizing the resulting vector,

$$\mathbf{q}_{j+1} = \mathbf{y}_j/\|\mathbf{y}_j\|. \tag{10.2}$$

Then $\{\mathbf{q}_1, \ldots, \mathbf{q}_{j+1}\}$ is an orthonormal basis of $\mathcal{K}^{j+1}(\mathbf{x})$, called in general the **Arnoldi basis** or, if the matrix A is real symmetric or Hermitian, the **Lanczos basis**. The vectors $\mathbf{q}_i$ are called **Arnoldi vectors** or **Lanczos vectors**, respectively, see [6, 1].

The vector $\mathbf{q}_{j+1}$ can be computed in a more economical way since

$$\begin{aligned}
\mathcal{K}^{j+1}(\mathbf{x}, A) &= \mathcal{R}\left([\mathbf{x}, A\mathbf{x}, \ldots, A^j\mathbf{x}]\right), && (\mathbf{q}_1 = \mathbf{x}/\|\mathbf{x}\|),\\
&= \mathcal{R}\left([\mathbf{q}_1, A\mathbf{q}_1, \ldots, A^j\mathbf{q}_1]\right) && (A\mathbf{q}_1 = \alpha\mathbf{q}_1 + \beta\mathbf{q}_2,\ \beta \neq 0),\\
&= \mathcal{R}\left([\mathbf{q}_1, \alpha\mathbf{q}_1 + \beta\mathbf{q}_2, A(\alpha\mathbf{q}_1 + \beta\mathbf{q}_2), \ldots, A^{j-1}(\alpha\mathbf{q}_1 + \beta\mathbf{q}_2)]\right),\\
&= \mathcal{R}\left([\mathbf{q}_1, \mathbf{q}_2, A\mathbf{q}_2, \ldots, A^{j-1}\mathbf{q}_2]\right),\\
&\ \ \vdots\\
&= \mathcal{R}\left([\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{j-1}, A\mathbf{q}_j]\right).
\end{aligned}$$

So, instead of orthogonalizing $A^j\mathbf{q}_1$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$, we can orthogonalize $A\mathbf{q}_j$ against $\mathbf{q}_1, \ldots, \mathbf{q}_j$ to obtain $\mathbf{q}_{j+1}$. The component $\mathbf{r}_j$ of $A\mathbf{q}_j$ orthogonal to $\mathbf{q}_1, \ldots, \mathbf{q}_j$ is given by

$$\mathbf{r}_j = A\mathbf{q}_j - \sum_{i=1}^{j} \mathbf{q}_i({\mathbf{q}_i}^* A\mathbf{q}_j). \tag{10.3}$$

If $\mathbf{r}_j = \mathbf{0}$ then the procedure stops which means that we have found an invariant subspace, namely $\operatorname{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_j\}$. If $\|\mathbf{r}_j\| > 0$ we obtain $\mathbf{q}_{j+1}$ by normalizing,

$$\mathbf{q}_{j+1} = \frac{\mathbf{r}_j}{\|\mathbf{r}_j\|}. \tag{10.4}$$

Since, $\mathbf{q}_{j+1}$ and $\mathbf{r}_j$ are aligned, we have

$$\mathbf{q}_{j+1}^* \mathbf{r}_j = \|\mathbf{r}_j\| \overset{(10.3)}{=} \mathbf{q}_{j+1}^* A \mathbf{q}_j. \tag{10.5}$$

The last equation holds since $\mathbf{q}_{j+1}$ (by construction) is orthogonal to all the previous Arnoldi vectors. Let

$$h_{ij} = \mathbf{q}_i^* A \mathbf{q}_j.$$

Then, (10.3)–(10.5) can be written as

$$A\mathbf{q}_j = \sum_{i=1}^{j+1} \mathbf{q}_i h_{ij}. \tag{10.6}$$

We collect the procedure in Algorithm 10.1

Algorithm 10.1 The Arnoldi algorithm for the computation of an orthonormal basis of a Krylov space

```
Let $A \in \mathbb{F}^{n \times n}$. This algorithm computes an orthonormal basis for $\mathcal{K}^k(\mathbf{x})$.
$\mathbf{q}_1 = \mathbf{x}/\|\mathbf{x}\|_2$;
for $j = 1, \ldots$ do
  $\mathbf{r} := A\mathbf{q}_j$;
  for $i = 1, \ldots, j$ do /* Gram-Schmidt orthogonalization */
    $h_{ij} := \mathbf{q}_i^* \mathbf{r}$,   $\mathbf{r} := \mathbf{r} - \mathbf{q}_i h_{ij}$;
  end for
  $h_{j+1,j} := \|\mathbf{r}\|$;
  if $h_{j+1,j} = 0$ then /* Found an invariant subspace */
    return $(\mathbf{q}_1, \ldots, \mathbf{q}_j, H \in \mathbb{F}^{j \times j})$
  end if
  $\mathbf{q}_{j+1} = \mathbf{r}/h_{j+1,j}$;
end for
return $(\mathbf{q}_1, \ldots, \mathbf{q}_{k+1}, H \in \mathbb{F}^{k+1 \times k})$
```

The Arnoldi algorithm returns if $h_{j+1,j} = 0$, in which case j is the degree of the minimal polynomial of A relative to $\mathbf{x}$, cf. (9.5). This algorithm costs k matrix-vector multiplications, $n^2/2 + \mathcal{O}(n)$ inner products, and the same number of _axpy's.

Defining $Q_k = [\mathbf{q}_1, \ldots, \mathbf{q}_k]$, equation (10.6) can be collected for $j = 1, \ldots, k$,

$$AQ_k = Q_k H_k + [\,\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{k-1 \text{ times}}, \mathbf{q}_{k+1} h_{k+1,k}] \tag{10.7}$$

Equation (10.7) is called **Arnoldi relation**. The construction of the Arnoldi vectors is expensive. Most of all, each iteration step becomes more costly as the number of vectors against which $\mathbf{r}$ has to be orthogonalized increases. Therefore, algorithms based on the Arnoldi relation like GMRES or the Arnoldi algorithm itself are restarted. This in general means that the algorithm is repeated with a initial vector that is extracted from previous invocation of the algorithm.

10.2 Arnoldi algorithm with explicit restarts

Algorithm 10.1 stops if $h_{m+1,m} = 0$, i.e., if it has found an invariant subspace. The vectors $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ then form an invariant subspace of A,

$$AQ_m = Q_m H_m, \qquad Q_m = [\mathbf{q}_1, \ldots, \mathbf{q}_m].$$

The eigenvalues of H_m are eigenvalues of A as well and the Ritz vectors are eigenvectors of A.

In general, we cannot afford to store the vectors $\mathbf{q}_1, \ldots, \mathbf{q}_m$ because of limited memory space. Furthermore, the algorithmic complexity increases linearly in the iteration number j. The orthogonalization would cost $2nm^2$ floating point operations.

Often it is possible to extract good approximate eigenvectors from a Krylov space of small dimension. We have seen, that in particular the extremal eigenvalues and corresponding eigenvectors are very well approximated after a few iteration steps. So, if only a small number of eigenpairs is desired, it is usually sufficient to get away with Krylov space of much smaller dimension than m.

Exploiting the Arnoldi relation (10.7) we can get cheap estimates for the eigenvalue/eigenvector residuals. Let $\mathbf{u}_i^{(k)} = Q_k \mathbf{s}_i^{(k)}$ be a Ritz vector with Ritz value $\vartheta_i^{(k)}$. Then

$$A\mathbf{u}_i^{(k)} - \vartheta_i^{(k)} \mathbf{u}_i^{(k)} = AQ_k \mathbf{s}_i^{(k)} - \vartheta_i^{(k)} Q_k \mathbf{s}_i^{(k)} = (AQ_k - Q_k H_k)\mathbf{s}_i^{(k)} = h_{k+1,k} \mathbf{q}_{k+1} \mathbf{e}_k^* \mathbf{s}_i^{(k)}.$$

Therefore,

$$\|(A - \vartheta_i^{(k)} I)\mathbf{u}_i^{(k)}\|_2 = h_{k+1,k} |\mathbf{e}_k^* \mathbf{s}_i^{(k)}|. \tag{10.8}$$

The residual norm is equal to the last component of $\mathbf{s}_i^{(k)}$ multiplied by $h_{k+1,k}$ (which is positive by construction). These residual norms are not always indicative of actual errors in $\lambda_i^{(k)}$, but they can be helpful in deriving stopping procedures.

We now consider an algorithm for computing some of the extremal eigenvalues of a non-Hermitian matrix. The algorithm proceeds by computing one eigenvector or rather Schur vector at the time. For each of them an individual Arnoldi procedure is employed. Let us assume that we have already computed $k-1$ Schur vectors $\mathbf{u}_1, \ldots \mathbf{u}_{k-1}$. To compute $\mathbf{u}_k$ we force the iterates in the Arnoldi process (the Arnoldi vectors) to be orthogonal to U_{k-1} where $U_{k-1} = [\mathbf{u}_1, \ldots \mathbf{u}_{k-1}]$. So, we work essentially with the matrix

$$(I - U_{k-1} U_{k-1}^*)A$$

that has $k-1$ eigenvalues zero which we of course neglect.

The procedure is given in Algorithm 10.2. The Schur vectors $\mathbf{u}_1, \ldots \mathbf{u}_{k-1}$ are kept in the search space, while the Krylov space is formed with the next approximate Schur vector. The search space thus is

$$\text{span}\{\mathbf{u}_1, \ldots \mathbf{u}_{k-1}, \mathbf{u}_k, A\mathbf{u}_k, \ldots A^{m-k}\mathbf{u}_k\}.$$

In Algorithm 10.2 the basis vectors are denoted $\mathbf{v}_j$ with $\mathbf{v}_j = \mathbf{u}_j$ for $j < k$. The vectors $\mathbf{v}_k, \ldots, \mathbf{v}_m$ form an orthonormal basis of $\text{span}\{\mathbf{u}_k, A\mathbf{u}_k, \ldots A^{m-k}\mathbf{u}_k\}$.

The matrix H_m for $k = 2$ has the structure

$$H_m = \left[\begin{array}{cc|cccc} \times & \times & \times & \times & \times & \times \\ & \times & \times & \times & \times & \times \\ \hline & & \times & \times & \times & \times \\ & & \times & \times & \times & \times \\ & & & \times & \times & \times \\ & & & & \times & \times \end{array}\right]$$

Algorithm 10.2 Explicitly restarted Arnoldi algorithm

```
Let $A \in \mathbb{F}^{n\times n}$. This algorithm computes the $n_{\rm ev}$ largest eigenvalues of $A$ together with
  the corresponding Schur vectors.
Set $k = 1$.
loop
  for $j = k, \ldots, m$ do /* Execute $m - k$ steps of Arnoldi */
    $\mathbf{r} := A\mathbf{q}_j$;
    for $i = 1, \ldots, j$ do
      $h_{ij} := \mathbf{q}_i^*\mathbf{r}, \quad \mathbf{r} := \mathbf{r} - \mathbf{q}_i h_{ij}$;
    end for
    $h_{j+1,j} := \|\mathbf{r}\|$;
    $\mathbf{q}_{j+1} = \mathbf{r}/h_{j+1,j}$;
  end for
  Compute approximate eigenvector of $A$ associated with $\lambda_k$ and the corresponding
    residual norm estimate $\rho_k$ according to (10.8).
  Orthogonalize this eigenvector (Ritz vector) against all previous $\mathbf{v}_j$ to get the ap-
    proximate Schur vector $\mathbf{u}_k$. Set $\mathbf{v}_k := \mathbf{u}_k$.
  if $\rho_k$ is small enough then /* accept eigenvalue */
    for $i = 1, \ldots, k$ do
      $h_{ik} := \mathbf{v}_i^* A\mathbf{v}_k$;
    end for
    Set $k := k + 1$.
    if $k \ge n_{\rm ev}$ then
      return $(\mathbf{v}_1, \ldots, \mathbf{v}_k, H \in \mathbb{F}^{k\times k})$
    end if
  end if
end loop
```

where the block in the lower right corresponds to the Arnoldi process for the Krylov space $\mathcal{K}_{m-k}(\mathbf{u}_k, (I - U_{k-1}U_{k-1}^*)A)$.

This algorithm needs at most m basis vectors. As soon as the dimension of the search space reaches m the Arnoldi iteration is **restarted** with the best approximation as the initial vector. The Schur vectors that have already converged are **locked** or **deflated**.

10.3 The Lanczos basis

We have seen that the Lanczos basis is formally constructed in the same way as the Arnoldi basis, however with a Hermitian matrix. It deserves a special name for the simplifications that the symmetry entails.

By multiplying (10.7) with Q_k^* from the left we get

$$Q_k^* A Q_k = Q_k^* Q_k H_k = H_k. \tag{10.9}$$

If A is Hermitian, then so is H_k. This means that H_k is *tridiagonal*. To emphasize this matrix structure, we call this tridiagonal matrix T_k. Due to symmetry, equation (10.3) simplifies considerably,

$$\mathbf{r}_j = A\mathbf{q}_j - \mathbf{q}_i \underbrace{(\mathbf{q}_j^* A\mathbf{q}_j)}_{\alpha_j\in\mathbb{R}} - \mathbf{q}_{j-1} \underbrace{(\mathbf{q}_{j-1}^* A\mathbf{q}_j)}_{\beta_{j-1}\in\mathbb{F}} = A\mathbf{q}_j - \alpha_j\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}. \tag{10.10}$$

Similarly as earlier, we premultiply (10.10) by $\mathbf{q}_{j+1}$ to get

$$\begin{aligned}\|\mathbf{r}_j\| = \mathbf{q}_{j+1}^*\mathbf{r}_j &= \mathbf{q}_{j+1}^*(A\mathbf{q}_j - \alpha_j\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1})\\ &= \mathbf{q}_{j+1}^* A\mathbf{q}_j = \bar{\beta}_j.\end{aligned}$$

From this it follows that $\beta_j \in \mathbb{R}$. Therefore,

$$\beta_j\mathbf{q}_{j+1} = \mathbf{r}_j, \qquad \beta_j = \|\mathbf{r}_j\|. \tag{10.11}$$

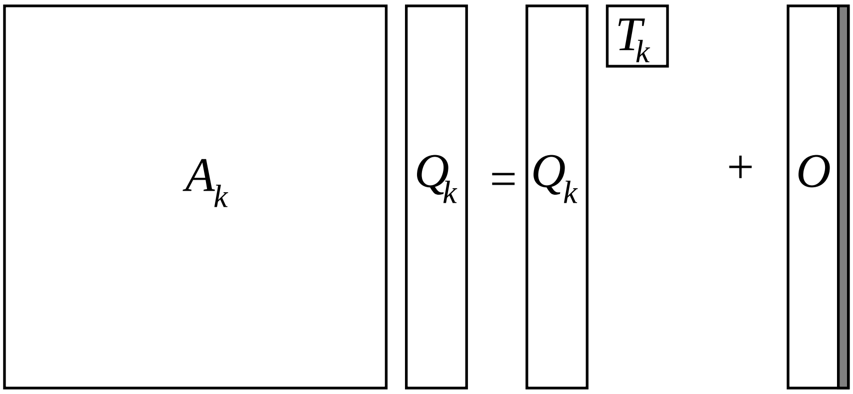
Collecting (10.10)–(10.11) yields

$$A\mathbf{q}_j = \beta_{j-1}\mathbf{q}_{j-1} + \alpha_j\mathbf{q}_j + \beta_j\mathbf{q}_{j+1}. \tag{10.12}$$

Gathering these equations for $j = 1, \ldots, k$ we get

$$AQ_k = Q_k \underbrace{\begin{pmatrix} \alpha_1 & \beta_1 & & & \\ \beta_1 & \alpha_2 & \beta_2 & & \\ & \beta_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{k-1} \\ & & & \beta_{k-1} & \alpha_k \end{pmatrix}}_{\mathbf{T}_k} + \beta_k[\mathbf{0}, \ldots, \mathbf{0}, \mathbf{q}_{k+1}]. \tag{10.13}$$

$T_k \in \mathbb{R}^{k\times k}$ is *real symmetric*. Equation (10.13) is called **Lanczos relation**. Pictorially, this is



The Lanczos algorithm is summarized in Algorithm 10.3. In this algorithm just the three vectors $\mathbf{q}$, $\mathbf{r}$, and $\mathbf{v}$ are employed. In the j-th iteration step (line 8) $\mathbf{q}$ is assigned $\mathbf{q}_j$ and $\mathbf{v}$ stores $\mathbf{q}_{j-1}$. $\mathbf{r}$ stores first (line 9) $A\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}$. Later (step 11), when α_j is available, it stores $\mathbf{r}_j = A\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1} - \alpha_j\mathbf{q}_j$. In the computation of α_j the fact is exploited that $\mathbf{q}_j^*\mathbf{q}_{j-1} = 0$ whence

$$\alpha_j = \mathbf{q}_j^* A\mathbf{q}_j = \mathbf{q}_j^*(A\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1}).$$

In each traversal of the j-loop a column is appended to the matrix Q_{j-1} to become Q_j. If the Lanczos vectors are not desired this statement can be omitted. The Lanczos vectors are required to compute the eigenvectors of A. Algorithm 10.3 returns when $j = m$, where m is the degree of the minimal polynomial of A relative to $\mathbf{x}$. $b_m = 0$ implies

$$AQ_m = Q_m T_m. \tag{10.14}$$

Algorithm 10.3 Basic Lanczos algorithm for the computation of an orthonormal basis for of the Krylov space $\mathcal{K}^m(\mathbf{x})$

```
Let $A \in \mathbb{F}^{n\times n}$ be Hermitian. This algorithm computes the Lanczos relation (10.13), i.e., an orthonormal basis $Q_m = [\mathbf{q}_1, \ldots, \mathbf{q}_m]$ for $\mathcal{K}^m(\mathbf{x})$ where $m$ is the smallest index such that $\mathcal{K}^m(\mathbf{x}) = \mathcal{K}^{m+1}(\mathbf{x})$, and (the nontrivial elements of) the tridiagonal matrix $T_m$.
$\mathbf{q} := \mathbf{x}/\|\mathbf{x}\|;\quad Q_1 = [\mathbf{q}];$
$\mathbf{r} := A\mathbf{q};$
$\alpha_1 := \mathbf{q}^*\mathbf{r};$
$\mathbf{r} := \mathbf{r} - \alpha_1\mathbf{q};$
$\beta_1 := \|\mathbf{r}\|;$
for $j = 2, 3, \ldots$ do
    $\mathbf{v} = \mathbf{q};\quad \mathbf{q} := \mathbf{r}/\beta_{j-1};\quad Q_j := [Q_{j-1}, \mathbf{q}];$
    $\mathbf{r} := A\mathbf{q} - \beta_{j-1}\mathbf{v};$
    $\alpha_j := \mathbf{q}^*\mathbf{r};$
    $\mathbf{r} := \mathbf{r} - \alpha_j\mathbf{q};$
    $\beta_j := \|\mathbf{r}\|;$
    if $\beta_j = 0$ then
        return $(Q \in \mathbb{F}^{n\times j};\ \alpha_1, \ldots, \alpha_j;\ \beta_1, \ldots, \beta_{j-1})$
    end if
end for
```

Let $(\lambda_i, \mathbf{s}_i)$ be an eigenpair of T_m,

$$T_m\mathbf{s}_i^{(m)} = \vartheta_i^{(m)}\mathbf{s}_i^{(m)}. \tag{10.15}$$

Then,

$$AQ_m\mathbf{s}_i^{(m)} = Q_mT_m\mathbf{s}_i^{(m)} = \vartheta_i^{(m)}Q_m\mathbf{s}_i^{(m)}. \tag{10.16}$$

So, the eigenvalues of T_m are also eigenvalues of A. The eigenvector of A corresponding to the eigenvalue ϑ_i is

$$\mathbf{y}_i = Q_m\mathbf{s}_i^{(m)} = [\mathbf{q}_1, \ldots, \mathbf{q}_m]\,\mathbf{s}_i^{(m)} = \sum_{j=1}^{m}\mathbf{q}_j s_{ji}^{(m)}. \tag{10.17}$$

The cost of a single iteration step of Algorithm 10.3 does not depend on the index of the iteration! In a single iteration step we have to execute a matrix-vector multiplication and $7n$ further floating point operations.

Remark 10.1. In certain very big applications the Lanczos vectors cannot be stored for reasons of limited memory. In this situation, the Lanczos algorithm is executed *without* building the matrix Q. When the desired eigenvalues and Ritz vectors have been determined from (10.15) the Lanczos algorithm is repeated and the desired eigenvectors are accumulated on the fly using (10.17). □

10.4 The Lanczos process as an iterative method

The Lanczos Algorithm 10.3 essentially determines an invariant Krylov subspace $\mathcal{K}^m(\mathbf{x})$ of $\mathbb{F}^n$. More precisely, it constructs an orthonormal basis $\{\mathbf{q}_1, \ldots, \mathbf{q}_m\}$ of $\mathcal{K}^m(\mathbf{x})$. The *projection* of A onto this space is a Hessenberg or even a real tridiagonal matrix if A is Hermitian.

We have seen in section 9.4 that the eigenvalues at the end of the spectrum are approximated very quickly in Krylov spaces. Therefore, only a very few iteration steps may be required to get those eigenvalues (and corresponding eigenvectors) within the desired accuracy, i.e., $|\vartheta_i^{(j)} - \lambda_i|$ may be tiny for $j \ll m$.

The Ritz values $\vartheta_i^{(j)}$ are the eigenvalues of the tridiagonal matrices T_j that are generated element by element in the course of the Lanczos algorithm. They can be computed efficiently by, e.g., the tridiagonal QR algorithm in $\mathcal{O}(j^2)$ flops. The cost for computing the eigenvalues of T_j are in general negligible compared with the cost for forming $A\mathbf{q}_j$.

But how can the error $|\vartheta_i^{(j)} - \lambda_i|$ be estimated? We will adapt the following more general lemma to this end.

Lemma 10.1 (**Eigenvalue inclusion of Krylov–Bogoliubov** [5] [7, p.69]) *Let $A \in \mathbb{F}^{n\times n}$ be Hermitian. Let $\vartheta \in \mathbb{R}$ and $\mathbf{x} \in \mathbb{F}^n$ with $\mathbf{x} \neq \mathbf{0}$ be arbitrary. Set $\tau := \|(A - \vartheta I)\mathbf{x}\|/\|\mathbf{x}\|$. Then there is an eigenvalue of A in the interval $[\vartheta - \tau, \vartheta + \tau]$.*

Proof. Let

$$A = U\Lambda U^* = \sum_{i=1}^{n} \lambda_i \mathbf{u}_i \mathbf{u}_i^*$$

be the spectral decomposition of A. Then,

$$(A - \vartheta I)\mathbf{x} = \sum_{i=1}^{n}(\lambda_i \mathbf{u}_i\mathbf{u}_i^* - \vartheta \mathbf{u}_i\mathbf{u}_i^*)\mathbf{x} = \sum_{i=1}^{n}(\lambda_i - \vartheta)(\mathbf{u}_i^*\mathbf{x})\mathbf{u}_i.$$

Taking norms, we obtain

$$\|(A - \vartheta I)\mathbf{x}\|^2 = \sum_{i=1}^{n}|\lambda_i - \vartheta|^2|\mathbf{u}_i^*\mathbf{x}|^2 \geq |\lambda_k - \vartheta|^2 \sum_{i=1}^{n}|\mathbf{u}_i^*\mathbf{x}|^2 = |\lambda_k - \vartheta|^2\|\mathbf{x}\|^*,$$

where λ_k is the eigenvalue closest to ϑ, i.e., $|\lambda_k - \vartheta| \leq |\lambda_i - \vartheta|$ for all i. ■

We want to apply this Lemma to the case where the vector is a Ritz vector $\mathbf{y}_i^{(j)}$ corresponding to the Ritz value $\tau = \vartheta_i^{(j)}$ as obtained in the j-th step of the Lanczos algorithm. Then,

$$\mathbf{y}_i^{(j)} = Q_j\mathbf{s}_i^{(j)}, \qquad T_j\mathbf{s}_i^{(j)} = \vartheta_i^{(j)}\mathbf{s}_i^{(j)}.$$

Thus, by employing the Lanczos relation (10.13),

$$\begin{aligned}\|A\mathbf{y}_i^{(j)} - \vartheta_i^{(j)}\mathbf{y}_i^{(j)}\| &= \|AQ_j\mathbf{s}_i^{(j)} - \vartheta_i^{(j)}Q_j\mathbf{s}_i^{(j)}\| \\ &= \|(AQ_j - Q_jT_j)\mathbf{s}_i^{(j)}\| \\ &= \|\beta_j\mathbf{q}_{j+1}\mathbf{e}_j^*\mathbf{s}_i^{(j)}\| = |\beta_j||\mathbf{e}_j^*\mathbf{s}_i^{(j)}| = |\beta_j||s_{ji}^{(j)}|.\end{aligned}$$

$s_{ji}^{(j)}$ is the j-th, i.e., the last element of the eigenvector matrix S_j of T_j,

$$T_jS_j = S_j\Theta_j, \qquad \Theta_j = \mathrm{diag}(\vartheta_1^{(j)}, \cdots, \vartheta_j^{(j)}).$$

According to Lemma 10.1 there is an eigenvalue λ of A such that

$$|\lambda - \vartheta_i^{(j)}| \leq \beta_j|s_{ji}|. \tag{10.18}$$

Thus, it is possible to get good eigenvalue approximations even if β_j is not small! Further, we know that [7, §11.7]

$$\sin \angle(\mathbf{y}_i^{(j)}, \mathbf{z}) \le \beta_j \frac{|s_{ji}|}{\gamma}, \tag{10.19}$$

where $\mathbf{z}$ is the eigenvector corresponding to λ in (10.18) and γ is the gap between λ and the next eigenvalue $\neq \lambda$ of A. In an actual computation, γ is not known. Parlett suggests to replace γ by the distance of $\vartheta_i^{(j)}$ to the next $\vartheta_k^{(j)}$, $k \neq i$. Because the $\vartheta_i^{(j)}$ converge to eigenvalues of A this substitution will give a reasonable number, at least in the limit.

In order to use the estimate (10.18) we need to compute all eigenvalues of T_j and the last row of S_j. It is possible and in fact straightforward to compute this row without the rest of S_j. The algorithm, a simple modification of the tridiagonal QR algorithm, has been introduced by Golub and Welsch [3] in connection with the computation of interpolation points and weights in Gaussian integration.

A numerical example

This numerical example is intended to show that the implementation of the Lanczos algorithm is not as simple as it seems from the previous. Let

$$A = \text{diag}(0, 1, 2, 3, 4, 100000)$$

and

$$\mathbf{x} = (1, 1, 1, 1, 1, 1)^T.$$

The diagonal matrix A has six simple eigenvalues and $\mathbf{x}$ has a non-vanishing component in the direction of each eigenspace. Thus, the Lanczos algorithm should stop after $m = n = 6$ iteration steps with the complete Lanczos relation. Up to rounding error, we expect that $\beta_6 = 0$ and that the eigenvalues of T_6 are identical with those of A. Let's see what happens if Algorithm 10.3 is applied with these input data. in the sequel we present the numbers that we obtained with a MATLAB implementation of this algorithm.

$\boxed{j = 1}$

$$\alpha_1 = 16668.33333333334, \qquad \beta_1 = 37267.05429136513.$$

$\boxed{j = 2}$

$$\alpha_2 = 83333.66652666384, \qquad \beta_2 = 3.464101610531258.$$

The diagonal of the eigenvalue matrix Θ_2 is:

$$\text{diag}(\Theta_2) = (1.999959999195565, 99999.99989999799)^T.$$

The last row of $\beta_2 S_2$ is

$$\beta_2 S_{2,:} = (1.4142135626139063.162277655014521).$$

The matrix of Ritz vectors $Y_2 = Q_2 S_2$ is

$$\begin{pmatrix} -0.44722 & -2.0000 \cdot 10^{-05} \\ -0.44722 & -9.9998 \cdot 10^{-06} \\ -0.44721 & 4.0002 \cdot 10^{-10} \\ -0.44721 & 1.0001 \cdot 10^{-05} \\ -0.44720 & 2.0001 \cdot 10^{-05} \\ 4.4723 \cdot 10^{-10} & 1.0000 \end{pmatrix}$$

$\boxed{j=3}$

$$\alpha_3 = 2.000112002245340 \qquad \beta_3 = 1.183215957295906.$$

The diagonal of the eigenvalue matrix is

$$\mathrm{diag}(\Theta_3) = (0.5857724375775532, 3.414199561869119, 99999.9999999999)^T.$$

The largest eigenvalue has converged already. This is not surprising as $\lambda_2/\lambda_1 = 4\cdot 10^{-5}$. With simple vector iteration the eigenvalues would converge with the factor $\lambda_2/\lambda_1 = 4\cdot 10^{-5}$.

The last row of $\beta_3 S_3$ is

$$\beta_3 S_{3,:} = \left(0.8366523355001995, 0.8366677176165411, 3.741732220526109 \cdot 10^{-05}\right).$$

The matrix of Ritz vectors $Y_3 = Q_3 S_3$ is

$$\begin{pmatrix} 0.76345 & 0.13099 & 2.0000\cdot 10^{-10} \\ 0.53983 & -0.09263 & -1.0001\cdot 10^{-10} \\ 0.31622 & -0.31623 & -2.0001\cdot 10^{-10} \\ 0.09262 & -0.53984 & -1.0000\cdot 10^{-10} \\ -0.13098 & -0.76344 & 2.0001\cdot 10^{-10} \\ -1.5864\cdot 10^{-13} & -1.5851\cdot 10^{-13} & 1.00000 \end{pmatrix}$$

The largest element (in modulus) of $Y_3^T Y_3$ is $\approx 3\cdot 10^{-12}$.

The Ritz vectors (and thus the Lanczos vectors $\mathbf{q}_i$) are mutually orthogonal up to rounding error.

$\boxed{j=4}$

$$\alpha_4 = 2.000007428756856 \qquad \beta_4 = 1.014186947306611.$$

The diagonal of the eigenvalue matrix is

$$\mathrm{diag}(\Theta_4) = \begin{pmatrix} 0.1560868732577987 \\ 1.999987898940119 \\ 3.843904656006355 \\ 99999.99999999999 \end{pmatrix}.$$

The last row of $\beta_4 S_4$ is

$$\beta_4 S_{4,:} = \left(0.46017, -0.77785, -0.46018, 3.7949\cdot 10^{-10}\right).$$

The matrix of Ritz vectors $Y_4 = Q_4 S_4$ is

$$\begin{pmatrix} -0.82515 & 0.069476 & -0.40834 & -0.18249 \\ -0.034415 & 0.41262 & -0.40834 & -0.18243 \\ 0.37812 & 0.37781 & -0.40834 & -0.18236 \\ 0.41256 & -0.034834 & -0.40834 & -0.18230 \\ 0.069022 & -0.82520 & -0.40834 & -0.18223 \\ -1.3202\cdot 10^{-04} & 1.3211\cdot 10^{-04} & -0.40777 & 0.91308 \end{pmatrix}.$$

The largest element (in modulus) of $Y_4^T Y_4$ is $\approx 2\cdot 10^{-8}$.

We have $\beta_4 s_{4,4} \doteq 4\cdot 10^{-10}$. So, according to our previous estimates $(\vartheta_4, \mathbf{y}_4)$, $\mathbf{y}_4 = Y_4\mathbf{e}_4$ is a very good approximation for an eigenpair of A. This is in fact the case.

Notice that $Y_4^T Y_4$ has off diagonal elements of the order 10^{-8}. These elements are in the last row/column of $Y_4^T Y_4$. This means that all Ritz vectors have a small but not negligible component in the direction of the ‘largest’ Ritz vector.

$\boxed{j=5}$

$$\alpha_5 = 2.363169101109444 \quad \beta_5 = 190.5668098726485.$$

The diagonal of the eigenvalue matrix is

$$\operatorname{diag}(\Theta_5) = \begin{pmatrix} 0.04749223464478182 \\ 1.413262891598485 \\ 2.894172742223630 \\ 4.008220660846780 \\ 9.999999999999999 \cdot 10^4 \end{pmatrix}.$$

The last row of $\beta_5 S_5$ is

$$\beta_5 S_{5,:} = \left(-43.570 - 111.38134.0963.4957.2320 \cdot 10^{-13}\right).$$

The matrix of Ritz vectors Y_5 is

$$\begin{pmatrix} -0.98779 & -0.084856 & 0.049886 & 0.017056 & -1.1424 \cdot 10^{-17} \\ -0.14188 & 0.83594 & -0.21957 & -0.065468 & -7.2361 \cdot 10^{-18} \\ 0.063480 & 0.54001 & 0.42660 & 0.089943 & -8.0207 \cdot 10^{-18} \\ -0.010200 & -0.048519 & 0.87582 & -0.043531 & -5.1980 \cdot 10^{-18} \\ -0.0014168 & -0.0055339 & 0.015585 & -0.99269 & -1.6128 \cdot 10^{-17} \\ 4.3570 \cdot 10^{-4} & 0.0011138 & -0.0013409 & -6.3497 \cdot 10^{-4} & 1.0000 \end{pmatrix}$$

Evidently, the last column of Y_5 is an excellent eigenvector approximation. Notice, however, that all Ritz vectors have a relatively large ($\sim 10^{-4}$) last component. This, gives rise to quite large off-diagonal elements of $Y_5^T Y_5 - I_5 =$

$$\begin{pmatrix} 2.220 \cdot 10^{-16} & -1.587 \cdot 10^{-16} & -3.430 \cdot 10^{-12} & -7.890 \cdot 10^{-9} & -7.780 \cdot 10^{-4} \\ -1.587 \cdot 10^{-16} & -1.110 \cdot 10^{-16} & 1.283 \cdot 10^{-12} & -1.764 \cdot 10^{-8} & -1.740 \cdot 10^{-3} \\ -3.430 \cdot 10^{-12} & 1.283 \cdot 10^{-12} & 0 & 5.6800 \cdot 10^{-17} & -6.027 \cdot 10^{-8} \\ -7.890 \cdot 10^{-9} & -1.764 \cdot 10^{-8} & 5.6800 \cdot 10^{-17} & -2.220 \cdot 10^{-16} & 4.187 \cdot 10^{-16} \\ -7.780 \cdot 10^{-4} & -1.740 \cdot 10^{-3} & -6.027 \cdot 10^{-8} & 4.187 \cdot 10^{-16} & -1.110 \cdot 10^{-16} \end{pmatrix}.$$

Similarly as with $j = 4$, the first four Ritz vectors satisfy the orthogonality condition very well. But they are not perpendicular to the last Ritz vector.

$\boxed{j=6}$

$$\alpha_6 = 99998.06336906151 \qquad \beta_6 = 396.6622037049789$$

The diagonal of the eigenvalue matrix is

$$\operatorname{diag}(\Theta_6) = \begin{pmatrix} 0.02483483859326367 \\ 1.273835519171372 \\ 2.726145019098232 \\ 3.975161765440400 \\ 9.999842654044850 \cdot 10^{+4} \\ 1.000000000000000 \cdot 10^{+5} \end{pmatrix}.$$

The eigenvalues are not the exact ones, as was to be expected. We even have *two* copies of the largest eigenvalue of A in Θ_6! The last row of $\beta_6 S_6$ is

$$\beta_6 S_{6,:} = \left(-0.20603, 0.49322, 0.49323, 0.20604, 396.66, -8.6152 \cdot 10^{-15}\right)$$

although theory predicts that $\beta_6 = 0$. The sixth entry of $\beta_6 S_6$ is very small, which means that the sixth Ritz value and the corresponding Ritz vector are good approximations to an eigenpair of A. In fact, eigenvalue and eigenvector are accurate to machine precision.

$\beta_5 s_{6,5}$ does not predict the fifth column of Y_6 to be a good eigenvector approximation, although the angle between the fifth and sixth column of Y_6 is less than 10^{-3}. The last two columns of Y_6 are

$$\begin{pmatrix} -4.7409 \cdot 10^{-4} & -3.3578 \cdot 10^{-17} \\ 1.8964 \cdot 10^{-3} & -5.3735 \cdot 10^{-17} \\ -2.8447 \cdot 10^{-3} & -7.0931 \cdot 10^{-17} \\ 1.8965 \cdot 10^{-3} & -6.7074 \cdot 10^{-17} \\ -4.7414 \cdot 10^{-4} & -4.9289 \cdot 10^{-17} \\ -0.99999 & 1.0000 \end{pmatrix}.$$

As $\beta_6 \neq 0$ one could continue the Lanczos process and compute ever larger tridiagonal matrices. If one proceeds in this way one obtains multiple copies of certain eigenvalues [2, 2]. The corresponding values $\beta_j s_{ji}^{(j)}$ will be tiny. The corresponding Ritz vectors will be 'almost' linearly dependent.

From this numerical example we see that the problem of the Lanczos algorithm consists in the loss of orthogonality among Ritz vectors which is a consequence of the loss of orthogonality among Lanczos vectors, since $Y_j = Q_j S_j$ and S_j is unitary (up to roundoff).

To verify this diagnosis, we rerun the Lanczos algorithm with *complete reorthogonalization*. This procedure amounts to the Arnoldi algorithm 10.1. It can be accomplished by modifying line 11 in the Lanczos algorithm 10.3, see Algorithm 10.4.

Algorithm 10.4 Lanczos algorithm with full reorthogonalization

```
11: $\mathbf{r} := \mathbf{r} - \alpha_j \mathbf{q}$;    $\mathbf{r} := \mathbf{r} - Q(Q^* \mathbf{r})$;
```

Of course, the cost of the algorithm increases considerably. The j-th step of the algorithm requires now a matrix-vector multiplication and $(2j + \mathcal{O}(1))n$ floating point operations.

A numerical example [continued]

With matrix and initial vector as before Algorithm 10.4 gives the following numbers.

$\boxed{j = 1}$

$$\alpha_1 = 16668.33333333334, \quad \beta_1 = 37267.05429136513.$$

$\boxed{j = 2}$

$$\alpha_2 = 83333.66652666384, \qquad \beta_2 = 3.464101610531258.$$

The diagonal of the eigenvalue matrix Θ_2 is:

$$\mathrm{diag}(\Theta_2) = (1.999959999195565, 99999.99989999799)^T.$$

$\boxed{j = 3}$

$$\alpha_3 = 2.000112002240894 \qquad \beta_3 = 1.183215957295905$$

The diagonal of the eigenvalue matrix is

$$\mathrm{diag}(\Theta_3) = (0.5857724375677908, 3.414199561859357, 100000.0000000000)^T.$$

$\boxed{j = 4}$

$$\alpha_4 = 2.000007428719501 \qquad \beta_4 = 1.014185105707661$$

$$\operatorname{diag}(\Theta_4) = \begin{pmatrix} 0.1560868732475296 \\ 1.999987898917647 \\ 3.843904655996084 \\ 99999.99999999999 \end{pmatrix}$$

The matrix of Ritz vectors $Y_4 = Q_4 S_4$ is

$$\begin{pmatrix} -0.93229 & 0.12299 & 0.03786 & -1.1767 \cdot 10^{-15} \\ -0.34487 & -0.49196 & -0.10234 & 2.4391 \cdot 10^{-15} \\ 2.7058 \cdot 10^{-6} & -0.69693 & 2.7059 \cdot 10^{-6} & 4.9558 \cdot 10^{-17} \\ 0.10233 & -0.49195 & 0.34488 & -2.3616 \cdot 10^{-15} \\ -0.03786 & 0.12299 & 0.93228 & 1.2391 \cdot 10^{-15} \\ 2.7086 \cdot 10^{-17} & 6.6451 \cdot 10^{-17} & -5.1206 \cdot 10^{-17} & 1.00000 \end{pmatrix}$$

The largest off-diagonal element of $|Y_4^T Y_4|$ is about $2 \cdot 10^{-16}$

$\boxed{j = 5}$

$$\alpha_5 = 2.000009143040107 \qquad \beta_5 = 0.7559289460488005$$

$$\operatorname{diag}(\Theta_5) = \begin{pmatrix} 0.02483568754088384 \\ 1.273840384543175 \\ 2.726149884630423 \\ 3.975162614480485 \\ 10000.000000000000 \end{pmatrix}$$

The Ritz vectors are $Y_5 =$

$$\begin{pmatrix} -9.91 \cdot 10^{-01} & -4.62 \cdot 10^{-02} & 2.16 \cdot 10^{-02} & -6.19 \cdot 10^{-03} & -4.41 \cdot 10^{-18} \\ -1.01 \cdot 10^{-01} & 8.61 \cdot 10^{-01} & -1.36 \cdot 10^{-01} & -3.31 \cdot 10^{-02} & 1.12 \cdot 10^{-17} \\ 7.48 \cdot 10^{-02} & 4.87 \cdot 10^{-01} & 4.87 \cdot 10^{-01} & -7.48 \cdot 10^{-02} & -5.89 \cdot 10^{-18} \\ -3.31 \cdot 10^{-02} & -1.36 \cdot 10^{-01} & 8.61 \cdot 10^{-01} & -1.01 \cdot 10^{-01} & 1.07 \cdot 10^{-17} \\ 6.19 \cdot 10^{-03} & 2.16 \cdot 10^{-02} & -4.62 \cdot 10^{-02} & -9.91 \cdot 10^{-01} & 1.13 \cdot 10^{-17} \\ 5.98 \cdot 10^{-18} & 1.58 \cdot 10^{-17} & -3.39 \cdot 10^{-17} & -5.96 \cdot 10^{-17} & 1.000000000000000 \end{pmatrix}$$

Largest off-diagonal element of $|Y_5^T Y_5|$ is about 10^{-16} The last row of $\beta_5 S_5$ is

$$\beta_5 S_{5,:} = \left(-0.20603, -0.49322, 0.49322, 0.20603, 2.8687 \cdot 10^{-15}\right).$$

$\boxed{j = 6}$

$$\alpha_6 = 2.000011428799386 \qquad \beta_6 = 4.178550866749342 \cdot 10^{-28}$$

$$\operatorname{diag}(\Theta_6) = \begin{pmatrix} 7.950307079340746 \cdot 10^{-13} \\ 1.000000000000402 \\ 2.000000000000210 \\ 3.000000000000886 \\ 4.000000000001099 \\ 9.999999999999999 \cdot 10^{4} \end{pmatrix}$$

The Ritz vectors are very accurate. Y_6 is almost the identity matrix are 1.0. The largest off diagonal element of $Y_6^T Y_6$ is about 10^{-16}. Finally,

$$\beta_6 S_{6,:} = \left(4.99 \cdot 10^{-29}, -2.00 \cdot 10^{-28}, 3.00 \cdot 10^{-28}, -2.00 \cdot 10^{-28}, 5.00 \cdot 10^{-29}, 1.20 \cdot 10^{-47}\right).$$

With a much enlarged effort we have obtained the desired result. Thus, the loss of orthogonality among the Lanczos vectors can be prevented by the explicit reorthogonalization against *all* previous Lanczos vectors. This amounts to applying the Arnoldi algorithm. In the sequel we want to better understand when the loss of orthogonality actually happens.

10.5 An error analysis of the unmodified Lanczos algorithm

When the quantities $Q_j, T_j, \mathbf{r}_j$, etc., are computed numerically by using the Lanczos algorithm, they can deviate greatly from their theoretical counterparts. However, despite this gross deviation from the exact model, it nevertheless delivers fully accurate Ritz value and Ritz vector approximations.

In this section $Q_j, T_j, \mathbf{r}_j$ etc. denote the *numerically computed* values and not their theoretical counterparts. So, instead of the Lanczos relation (10.13) we write

$$AQ_j - Q_jT_j = \mathbf{r}_j\mathbf{e}_j^* + F_j \tag{10.20}$$

where the matrix F_j accounts for errors due to roundoff. Similarly, we write

$$I_j - Q_j^*Q_j = C_j^* + \Delta_j + C_j, \tag{10.21}$$

where Δ_j is a diagonal matrix and C_j is a *strictly* upper triangular matrix (with zero diagonal). Thus, $C_j^* + \Delta_j + C_j$ indicates the deviation of the Lanczos vectors from orthogonality.

We make the following **assumptions**

1. The tridiagonal eigenvalue problem can be solved exactly, i.e.,

$$T_j = S_j\Theta_jS_j^*, \quad S_j^* = S_j^{-1}, \quad \Theta_j = \mathrm{diag}(\vartheta_1, \ldots, \vartheta_j). \tag{10.22}$$

2. The orthogonality of the Lanczos vectors holds *locally*, i.e.,

$$\mathbf{q}_{i+1}^*\mathbf{q}_i = 0, \quad i = 1, \ldots, j-1, \quad \text{and} \quad \mathbf{r}_j^*\mathbf{q}_i = 0. \tag{10.23}$$

3. Furthermore,

$$\|\mathbf{q}_i\| = 1. \tag{10.24}$$

So, we assume that the computations that we actually perform (like orthogonalizations or solving the eigenvalue problem) are accurate. These assumptions imply that $\Delta_j = O$ and $c_{i,i+1}^{(j)} = 0$ for $i = 1, \ldots, j-1$.

We premultiply (10.20) by Q_j^* and obtain

$$Q_j^*AQ_j - Q_j^*Q_jT_j = Q_j^*\mathbf{r}_j\mathbf{e}_j^* + Q_j^*F_j \tag{10.25}$$

In order to eliminate A we subtract from this equation its transposed,

$$\begin{aligned} Q_j^*\mathbf{r}_j\mathbf{e}_j^* - \mathbf{e}_j\mathbf{r}_j^*Q_j &= -Q_j^*Q_jT_j + T_jQ_j^*Q_j + Q_j^*F_j - F_j^*Q_j, \\ &= (I - Q_j^*Q_j)T_j - T_j(I - Q_j^*Q_j) + Q_j^*F_j - F_j^*Q_j, \\ &\overset{(10.21)}{=} (C_j + C_j^*)T_j - T_j(C_j + C_j^*) + Q_j^*F_j - F_j^*Q_j, \\ &= \underbrace{(C_jT_j - T_jC_j)}_{\text{upper triangular}} + \underbrace{(C_j^*T_j - T_jC_j^*)}_{\text{lower triangular}} - F_j^*Q_j + Q_j^*F_j. \end{aligned} \tag{10.26}$$

$F_j^* Q_j - Q_j^* F_j$ is skew symmetric. Therefore we have

$$F_j^* Q_j - Q_j^* F_j = -K_j^* + K_j,$$

where K_j is an upper triangular matrix with zero diagonal. Thus, (10.25) has the form

$$\begin{pmatrix} & O & & \times \\ & & & \vdots \\ & & & \times \\ \underbrace{\times \cdots \times}_{\substack{\text{First } j-1 \\ \text{components} \\ \text{of } \mathbf{r}_j^* Q_j.}} & & & 0 \end{pmatrix} = \begin{pmatrix} 0 & & C_jT_j - T_jC_j \\ & \ddots & \\ C_j^*T_j - T_jC_j^* & & 0 \end{pmatrix} + \begin{pmatrix} 0 & & K_j \\ & \ddots & \\ -K_j^* & & 0 \end{pmatrix}.$$

As the last component of $Q_j^* \mathbf{r}_j$ vanishes, we can treat these triangular matrices separately. For the upper triangular matrices we have

$$Q_j^* \mathbf{r}_j \mathbf{e}_j^* = C_jT_j - T_jC_j + K_j.$$

Multiplication by $\mathbf{s}_i^*$ and $\mathbf{s}_i$, respectively, from left and right gives

$$\underbrace{\mathbf{s}_i^* Q_j^*}_{\mathbf{y}_i^*} \underbrace{\mathbf{r}_j}_{\beta_j \mathbf{q}_{j+1}} \underbrace{\mathbf{e}_j^* \mathbf{s}_i}_{s_{ji}} = \mathbf{s}_i^*(C_jT_j - T_jC_j)\mathbf{s}_i + \mathbf{s}_i^* K_j \mathbf{s}_i.$$

Let $G_j := S_i^* K_j S_i$. Then we have

$$\beta_j s_{ji} \mathbf{y}_i^* \mathbf{q}_{j+1} = s_{ji} \mathbf{y}_i^* \mathbf{r}_j = \mathbf{s}_i^* C_j \mathbf{s}_i \vartheta_i - \vartheta_i \mathbf{s}_i^* C_j \mathbf{s}_i + g_{ii}^{(j)} = g_{ii}^{(j)}. \tag{10.27}$$

We now multiply (10.25) with $\mathbf{s}_i^*$ from the left and with $\mathbf{s}_k$ from the right. As $Q_j \mathbf{s}_i = \mathbf{y}_i$, we have

$$\mathbf{y}_i^* A \mathbf{y}_k - \mathbf{y}_i^* \mathbf{y}_k \vartheta_k = \mathbf{y}_i^* \mathbf{r}_j \mathbf{e}_j^* \mathbf{s}_k + \mathbf{s}_i^* Q_j^* F_j \mathbf{s}_k.$$

Now, from this equation we subtract again its transposed, such that A is eliminated,

$$\begin{aligned} \mathbf{y}_i^* \mathbf{y}_k (\vartheta_i - \vartheta_k) &= \mathbf{y}_i^* \mathbf{r}_j \mathbf{e}_j^* \mathbf{s}_k - \mathbf{y}_k^* \mathbf{r}_j \mathbf{e}_j^* \mathbf{s}_i + \mathbf{s}_i^* Q_j^* F_j \mathbf{s}_k - \mathbf{s}_k^* Q_j^* F_j \mathbf{s}_i \\ &\overset{(10.27)}{=} \left(\frac{g_{ii}^{(j)}}{s_{ji}^{(j)}} \right) s_{jk} - \left(\frac{g_{kk}^{(j)}}{s_{jk}^{(j)}} \right) s_{ji} \\ &\quad + \frac{1}{2}(\mathbf{s}_i^* Q_j^* F_j \mathbf{s}_k + \mathbf{s}_k^* F_j^* Q_j \mathbf{s}_i) - \frac{1}{2}(\mathbf{s}_k^* Q_j^* F_j \mathbf{s}_i + \mathbf{s}_i^* F_j^* Q_j \mathbf{s}_k) \\ &= g_{ii}^{(j)} \frac{s_{jk}^{(j)}}{s_{ji}^{(j)}} - g_{kk}^{(j)} \frac{s_{ji}^{(j)}}{s_{jk}^{(j)}} - (g_{ik}^{(j)} - g_{ki}^{(j)}). \end{aligned}$$

Thus we have proved

Theorem 10.2 (Paige, see [7, p.266]) *With the above notations we have*

$$\mathbf{y}_i^{(j)^*} \mathbf{q}_{j+1} = \frac{g_{ii}^{(j)}}{\beta_j s_{ji}^{(j)}} \tag{10.28}$$

$$(\vartheta_i^{(j)} - \vartheta_k^{(j)}) \mathbf{y}_i^{(j)^*} \mathbf{y}_k^{(j)} = g_{ii}^{(j)} \frac{s_{jk}^{(j)}}{s_{ji}^{(j)}} - g_{kk}^{(j)} \frac{s_{ji}^{(j)}}{s_{jk}^{(j)}} - (g_{ik}^{(j)} - g_{ki}^{(j)}). \tag{10.29}$$

■

We can **interpret** these equations in the following way.

- From numerical experiments it is known that equation (10.20) is always satisfied to machine precision. Thus, $\|F_j\| \approx \varepsilon\,\|A\|$. Therefore, $\|G_j\| \approx \varepsilon\,\|A\|$, and, in particular, $|g_{ik}^{(j)}| \approx \varepsilon\,\|A\|$.
- We see from (10.28) that $|\mathbf{y}_i^{(j)^*}\mathbf{q}_{j+1}|$ becomes large if $\beta_j|s_{ji}^{(j)}|$ becomes small, i.e., if the Ritz vector $\mathbf{y}_i^{(j)}$ is a good approximation of the corresponding eigenvector. Thus, each *new* Lanczos vector has a significant component in the direction of converged ('good') Ritz vectors.

  As a consequence: $\boxed{\textbf{convergence} \iff \textbf{loss of orthogonality}}$.
- Let $|s_{ji}^{(j)}| \ll |s_{jk}^{(j)}|$, i.e., $\mathbf{y}_i^{(j)}$ is a 'good' Ritz vector in contrast to $\mathbf{y}_k^{(j)}$ that is a 'bad' Ritz vector. Then in the first term on the right of (10.29) two small ($\mathcal{O}(\varepsilon)$) quantities counteract each other such that the right hand side in (10.29) becomes large, $\mathcal{O}(1)$. If the corresponding Ritz values are well separated, $|\vartheta_i - \vartheta_k| = \mathcal{O}(1)$, then $|\mathbf{y}_i^*\mathbf{y}_k| \gg \varepsilon$. So, in this case also 'bad' Ritz vectors have a significant component in the direction of the 'good' Ritz vectors.
- If $|\vartheta_i - \vartheta_k| = \mathcal{O}(\varepsilon)$ and both $s_{ji}^{(j)}$ and $s_{jk}^{(j)}$ are of $\mathcal{O}(\varepsilon)$ the $s_{ji}^{(j)}/s_{jk}^{(j)} = \mathcal{O}(1)$ such that the right hand side of (10.29) as well as $|\vartheta_i - \vartheta_k|$ is $\mathcal{O}(\varepsilon)$. Therefore, we must have $\mathbf{y}_i^{(j)^*}\mathbf{y}_k^{(j)} = \mathcal{O}(1)$. So, these two vectors are almost parallel.

10.6 Partial reorthogonalization

In Section 10.4 we have learned that the Lanczos algorithm does not yield orthogonal Lanczos vectors as it should in theory due to floating point arithmetic. In the previous section we learned that the loss of orthogonality happens as soon as Ritz vectors have converged accurately enough to eigenvectors. In this section we review an approach how to counteract the loss of orthogonality without executing *full* reorthogonalization [8, 9].

In [7] it is shown that if the Lanczos basis is **semiorthogonal**, i.e., if

$$W_j = Q_j^* Q_j = I_j + E, \qquad \|E\| < \sqrt{\varepsilon_M},$$

then the tridiagonal matrix T_j is the projection of A onto the subspace $\mathcal{R}(V_j)$,

$$T_j = N_j^* A N_j + G, \qquad \|G\| = O((\varepsilon_M)\|A\|),$$

where N_j is an orthonormal basis of $\mathcal{R}(Q_j)$. Therefore, it suffices to have semiorthogonal Lanczos vectors for computing accurate eigenvalues. Our goal is now to enforce semiorthogonality by monitoring the loss of orthogonality and to reorthogonalize if needed.

The computed Lanczos vectors satisfy

$$\beta_j\mathbf{q}_{j+1} = A\mathbf{q}_j - \alpha_j\mathbf{q}_j - \beta_{j-1}\mathbf{q}_{j-1} + \mathbf{f}_j, \tag{10.30}$$

where $\mathbf{f}_j$ accounts for the roundoff errors committed in the j-th iteration step. Let $W_j = ((\omega_{ik}))_{1\le i,\,k\le j}$. Premultiplying equation (10.30) by $\mathbf{q}_k^*$ gives

$$\beta_j\omega_{j+1,k} = \mathbf{q}_k^* A\mathbf{q}_j - \alpha_j\omega_{jk} - \beta_{j-1}\omega_{j-1,k} + \mathbf{q}_k^*\mathbf{f}_j. \tag{10.31}$$

Exchanging indices j and k in the last equation (10.31) gives

$$\beta_k\omega_{j,k+1} = \mathbf{q}_j^* A\mathbf{q}_k - \alpha_k\omega_{jk} - \beta_{k-1}\omega_{j,k-1} + \mathbf{q}_j^*\mathbf{f}_k. \tag{10.32}$$

By subtracting (10.32) from (10.31) we get

$$\beta_j\omega_{j+1,k} = \beta_k\omega_{j,k+1} + (\alpha_k - \alpha_j)\omega_{jk} - \beta_{k-1}\omega_{j,k-1} - \beta_{j-1}\omega_{j-1,k} - \mathbf{q}_j^*\mathbf{f}_k + \mathbf{q}_k^*\mathbf{f}_j. \tag{10.33}$$

Given W_j we employ equation (10.33) to compute the $j+1$-th row of W_{j+1}. However, elements $\omega_{j+1,j}$ and $\omega_{j+1,j+1}$ are not defined by (10.33). We can assign values to these two matrix entries by reasoning as follows.

- We set $\omega_{j+1,j+1} = 1$ because we explicitly normalize $\mathbf{q}_{j+1}$.
- We set $\omega_{j+1,j} = O(\varepsilon_M)$ because we explicitly orthogonalize $\mathbf{q}_{j+1}$ and $\mathbf{q}_j$.

For computational purposes, equation (10.33) is now replaced by

$$\begin{aligned}\tilde{\omega} &= \beta_k\omega_{j,k+1} + (\alpha_k - \alpha_j)\omega_{jk} - \beta_{k-1}\omega_{j,k-1} - \beta_{j-1}\omega_{j-1,k},\\ \omega_{j+1,k} &= (\tilde{\omega} + \operatorname{sign}(\tilde{\omega})\underbrace{2\varepsilon\|A\|}_{\substack{\text{the estimate of}\\ \mathbf{q}_j^*\mathbf{f}_k + \mathbf{q}_k^*\mathbf{f}_j}})/\beta_j.\end{aligned} \tag{10.34}$$

As soon as $\omega_{j+1,k} > \sqrt{\varepsilon_M}$ the vectors $\mathbf{q}_j$ and $\mathbf{q}_{j+1}$ are orthogonalized against *all* previous Lanczos vectors $\mathbf{q}_1, \ldots, \mathbf{q}_{j-1}$. Then the elements of last two lines of W_j are set equal to a number of size $\mathcal{O}(\varepsilon_M)$. Notice that only the last two rows of W_j have to be stored.

Numerical example

We perform the Lanczos algorithm with matrix

$$A = \operatorname{diag}(1, 2, \ldots, 50)$$

and initial vector

$$\mathbf{x} = [1, \ldots, 1]^*.$$

In the first experiment we execute 50 iteration steps. In Table 10.1 the base-10 logarithms of the values $|w_{i,j}|$/macheps are listed where $|w_{i,j}| = |\mathbf{q}_i^*\mathbf{q}_j|$, $1 \le j \le i \le 50$ and macheps $\approx 2.2 \cdot 10^{-16}$. One sees how the $|w_{i,j}|$ steadily grow with increasing i and with increasing $|i-j|$.

In the second experiment we execute 50 iteration steps with partial reorthogonalization turned on. The estimators $\omega_{j,k}$ are computed according to (10.33),

$$\begin{aligned}\omega_{k,k} &= 1, \qquad k = 1, \ldots, j\\ \omega_{k,k-1} &= \psi_k, \qquad k = 2, \ldots, j\\ \omega_{j+1,k} &= \frac{1}{\beta_j}\left[\beta_k\omega_{j,k+1} + (\alpha_k - \alpha_j)\omega_{jk}\right.\\ &\qquad \left.-\beta_{k-1}\omega_{j,k-1} - \beta_{j-1}\omega_{j-1,k}\right] + \vartheta_{i,k}, \qquad 1 \le k \le j.\end{aligned} \tag{10.35}$$

Here, we set $\omega_{j,0} = 0$. The values ψ_k and $\vartheta_{i,k}$ could be defined to be random variables of the correct magnitude, i.e., $\mathcal{O}(\varepsilon k)$. Following a suggestion of Parlett [7] we used

$$\psi_k = \varepsilon\|A\|, \qquad \vartheta_{i,k} = \varepsilon\sqrt{\|A\|}.$$

0																																																
0	0																																															
1	1	0																																														
1	0	1	0																																													
1	1	0	1	0																																												
1	1	1	0	1	0																																											
0	1	1	0	0	0	0																																										
1	0	1	1	0	1	1	0																																									
1	0	0	1	0	1	1	1	0																																								
1	0	0	1	1	1	1	0	1	0																																							
0	1	0	1	1	1	1	0	0	0	0																																						
0	1	0	1	1	0	1	1	0	0	0	0																																					
1	1	1	1	1	1	0	1	1	0	1	1	0																																				
1	1	1	1	1	1	0	0	1	0	1	0	1	0																																			
1	1	0	1	0	1	1	1	1	0	1	1	0	1	0																																		
1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	0																																	
0	1	1	1	1	1	0	1	1	1	0	1	1	0	0	1	0																																
1	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0																															
0	1	1	2	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0																														
1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	0	0	0																													
1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	0	0																												
1	1	1	1	2	1	2	1	1	1	1	1	1	1	1	0	1	1	0	0	1	0																											
0	2	1	2	1	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	0	0																										
1	1	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	0	0																									
1	2	2	2	2	2	2	2	2	2	2	2	1	2	1	2	1	1	1	1	1	0	0	1	0																								
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	1	0																							
2	2	2	2	2	3	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	0	0	0	0																						
2	2	3	3	3	3	3	3	3	2	3	2	2	2	2	2	2	2	2	1	1	1	1	1	1	0	1	0																					
2	3	3	3	3	3	3	3	3	3	3	3	2	3	2	2	2	2	2	2	2	2	1	1	1	0	1	1	0																				
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	2	2	1	2	1	1	1	1	0																			
3	3	3	4	3	4	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	1	1	1	1	1	1	0																		
3	3	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	2	2	2	2	2	2	1	2	1	1	0	1	0																	
4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	2	2	2	2	1	1	1	1	0	0	0																
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	2	2	2	2	1	1	1	1	1	1	0															
4	5	5	5	5	5	5	5	5	5	4	5	4	4	4	4	4	4	4	4	3	3	3	3	3	2	2	2	2	2	1	1	1	1	0														
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	3	3	3	3	3	3	2	2	2	1	1	2	0	0	0													
5	5	5	6	5	6	5	5	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	3	3	3	3	2	2	2	1	1	1	1	0	0												
6	6	6	6	6	6	6	6	6	6	6	5	6	5	5	5	5	5	5	5	5	4	4	4	4	3	3	3	3	2	2	2	2	1	1	1	0	0											
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5	4	4	4	4	4	3	3	3	2	2	2	2	1	1	0	0	0										
6	6	7	7	7	7	7	7	7	6	7	6	6	6	6	6	6	6	6	5	5	5	5	5	5	4	4	4	4	3	3	3	2	2	2	1	1	1	1	0									
7	7	7	7	7	7	7	7	7	7	7	7	7	7	6	7	6	6	6	6	6	6	5	5	5	5	4	4	4	4	3	3	3	3	2	2	1	1	1	1	0								
7	7	8	8	8	8	8	7	8	7	8	7	7	7	7	7	7	7	7	6	6	6	6	6	6	5	5	5	5	4	4	4	3	3	3	2	2	1	2	0	1	0							
8	8	8	8	8	8	8	8	8	8	8	8	8	8	7	8	7	7	7	7	7	7	6	6	6	6	5	5	5	5	4	4	4	4	3	3	2	1	1	1	1	1	0						
8	8	9	9	9	9	9	9	9	8	9	8	8	8	8	8	8	8	8	7	7	7	7	7	7	6	6	6	6	5	5	5	4	4	4	3	3	2	2	1	2	1	1	0					
9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	8	8	8	8	8	8	7	7	7	7	7	6	6	6	5	5	5	5	4	4	3	3	3	2	1	2	1	1	0				
10	10	10	10	10	10	10	10	10	10	10	10	10	9	9	9	9	9	9	9	9	8	8	8	8	7	7	7	7	6	6	6	6	5	5	4	4	4	3	3	2	1	2	1	1	0			
10	11	10	11	10	11	10	11	10	11	10	10	10	10	10	10	10	10	9	9	9	9	9	9	8	8	8	8	7	7	7	7	6	6	5	5	5	4	4	3	3	2	1	2	1	2	0		
11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	10	10	10	10	10	10	9	9	9	9	8	8	8	8	7	7	7	6	6	6	5	5	4	4	3	1	1	2	1	1	0	
12	12	12	12	12	12	12	12	12	12	12	12	12	12	11	12	11	11	11	11	11	11	10	10	10	10	9	9	9	9	8	8	8	8	7	7	6	6	5	5	5	4	3	2	1	2	2	1	0	
13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	12	12	12	12	12	12	11	11	11	11	11	11	10	10	10	9	9	9	8	8	8	7	7	7	6	6	5	4	4	3	1	3	2	1	0

Table 10.1: MATLAB demo on the loss of orthogonality among Lanczos vectors. Unmodified Lanczos. `round(log10(abs(I-`$Q_{50}^{*}Q_{50}$`)/eps))`

Reorthogonalization takes place in the j-th Lanczos step if $\max_k(\omega_{j+1,k}) > \sqrt{\text{macheps}}$. $\mathbf{q}_{j+1}$ is orthogonalized against all vectors $\mathbf{q}_k$ with $\omega_{j+1,k} > \text{macheps}^{3/4}$. In the following iteration step also $\mathbf{q}_{j+2}$ is orthogonalized against these vectors. In Table 10.2 the base-10 logarithms of the values $|w_{i,j}|/\text{macheps}$ obtained with this procedure are listed where $|w_{i,j}| = |\mathbf{q}_i^*\mathbf{q}_j|$, $1 \le j \le i \le 50$ and $\text{macheps} \approx 2.2 \cdot 10^{-16}$. In Table 10.3 the base-10 logarithms of the estimates $|\omega_{i,j}|/\text{macheps}$ are given. The estimates are too high by (only) an order of magnitude. However, the procedure succeeds in that the resulting $\{\mathbf{q}_k\}$ *are* semi-orthogonal.

10.7 Block Lanczos

As we have seen, the Lanczos algorithm produces a sequence $\{\mathbf{q}_i\}$ of orthonormal vectors. These Lanczos vectors build an orthonormal basis for the Krylov subspace $\mathcal{K}^j(\mathbf{x}) = \text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_j\} \subset \mathbb{R}^n$. The restriction of A to $\mathcal{K}^j(\mathbf{x})$ is an unreduced tridiagonal matrix. However the Lanczos algorithm cannot detect the *multiplicity* of the eigenvalues it computes. This limitation prompted the development of the block version of the Lanczos process (*Block Lanczos algorithm*), which is capable of determining multiplicities of eigenvalues up to the block size.

The idea is not to start with a single vector $\mathbf{q}_1 \in \mathbb{R}^n$ but with a set of mutually orthogonal vectors which we take as the columns of the matrix $Q_1 \in \mathbb{R}^{n\times p}$ with the *block size* $p > 1$.

Associated with Q_1 is the 'big' Krylov subspace

$$\mathcal{K}^{jp}(Q_1) = \text{span}\{Q_1, AQ_1, \ldots, A^{j-1}Q_1\}. \tag{10.36}$$

(We suppose, for simplicity, that $A^{j-1}Q_1$ has rank p. Otherwise we would have to consider variable block sizes.)

The approach is similar to the scalar case with $p = 1$: Let $Q_1, \ldots, Q_j \in \mathbb{R}^{n\times p}$ be pairwise orthogonal block matrices ($Q_i^*Q_k = O$ for $i \ne k$) with orthonormal columns ($Q_i^*Q_i = I_p$ for all $i \le j$). Then, in the j-th iteration step, we obtain the matrix AQ_j and orthogonalize it against matrices Q_i, $i \le j$. The columns of the matrices are obtained by means of the QR factorization or with the Gram–Schmidt orthonormalization process. We obtained the following:

Algorithm 10.5 Block Lanczos algorithm

```
Choose $Q_1 \in \mathbb{F}^{n\times p}$ such that $Q_1^*Q_1 = I_p$. Set $j := 0$ and $\mathbb{F}^{n\times p} \ni V := 0$.
This algorithm generates a block tridiagonal matrix $\hat{T}_j$ with the diagonal blocks $A_i$,
$i \le j$, the lower diagonal blocks $B_i, i < j$, and the Krylov basis $[Q_1, \ldots, Q_j]$ of $\mathcal{K}^{jp}(Q_1)$.
for $j \ge 0$ do
  if $j > 0$ then
    $V =: Q_{j+1}B_j$; /* QR decomposition */
    $V := -Q_jB_j^*$;
  end if
  $j := j + 1$;
  $A_j := Q_j^*V$;
  $V := V - Q_jA_j$;
  Test for convergence (Ritz pairs, evaluation of error)
end for
```

```
0
0 0
1 1 0
1 0 1 0
1 1 0 1 0
1 1 1 0 1 0
0 1 1 0 0 0 0
1 0 1 1 0 1 1 0
1 0 0 1 0 1 1 1 0
1 0 0 1 1 1 1 0 1 0
0 1 0 1 1 1 1 0 0 0 0
0 1 0 1 1 0 1 1 0 0 0 0
1 1 1 1 1 1 0 1 1 0 1 1 0
1 1 1 1 1 1 0 0 1 0 1 0 1 0
1 1 0 1 0 1 1 1 1 0 1 1 0 1 0
1 1 1 0 1 1 1 0 1 0 1 1 1 0 1 0
0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 0
1 1 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 0
0 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0
1 1 1 1 2 1 2 1 1 1 1 1 1 1 1 0 1 1 0 0 1 0
0 2 1 2 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 2 2 2 2 2 2 2 2 2 2 2 1 2 1 2 1 1 1 1 1 0 0 1 0
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 0 0 1 0
2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 0 0 0 0
2 2 3 3 3 3 3 3 3 2 3 2 2 2 2 2 2 2 2 1 1 1 1 1 1 0 1 0
2 3 3 3 3 3 3 3 3 3 3 3 2 3 2 2 2 2 2 2 2 2 1 1 1 0 1 1 0
3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 1 2 1 1 1 1 0
3 3 3 4 3 4 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 1 1 1 1 1 1 0
3 3 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 2 2 2 2 2 2 1 2 1 1 0 1 0
4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 2 2 2 2 1 1 1 1 0 0 0
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 2 2 2 2 1 1 1 1 1 1 0
4 5 5 5 5 5 5 5 5 5 4 5 4 4 4 4 4 4 4 4 3 3 3 3 3 2 2 2 2 2 1 1 1 1 0
5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 3 3 3 3 3 3 2 2 2 1 1 2 0 0 0
5 5 5 6 5 6 5 5 5 5 5 5 5 5 5 5 5 5 4 4 4 4 4 4 3 3 3 3 2 2 2 1 1 1 1 0 0
6 6 6 6 6 6 6 6 6 6 6 5 6 5 5 5 5 5 5 5 5 4 4 4 4 3 3 3 3 2 2 2 2 1 1 1 0 0
6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 5 5 5 5 5 4 4 4 4 4 3 3 3 2 2 2 2 1 1 0 0 0
6 6 7 7 7 7 7 7 7 6 7 6 6 6 6 6 6 6 6 5 5 5 5 5 5 4 4 4 4 3 3 3 2 2 2 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 2 2 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 3 2 2 1 2 0 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 3 3 3 3 2 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 3 3 2 2 1 2 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 3 3 3 3 3 4 3 3 3 3 2 1 2 1 1 0
1 1 1 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 1 0 1 0 3 3 4 4 4 4 4 4 4 4 3 3 2 1 1 1 0 0
1 1 2 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 2 1 2 1 1 1 1 2 0 3 4 4 4 4 4 4 4 4 4 4 4 3 3 2 2 2 1 1 0
2 2 2 2 1 1 2 1 2 1 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 4 4 4 5 5 5 5 5 5 5 5 5 4 4 4 3 2 2 2 1 1 0
3 3 3 3 3 2 2 2 2 3 2 3 2 2 2 3 3 3 3 3 3 2 3 3 3 4 5 5 5 5 6 6 6 6 6 5 5 5 5 5 5 4 3 2 2 2 1 1 0
4 4 4 4 3 3 3 3 3 3 4 3 3 3 4 4 4 4 4 4 4 4 3 4 5 5 6 6 6 6 6 7 6 7 6 6 6 6 6 6 5 5 4 4 2 2 2 2 2 0
```

Table 10.2: MATLAB demo on the loss of orthogonality among Lanczos vectors: Lanczos with partial reorthogonalization. `round(log10(abs(I-Q`$^*_{50}$`Q`$_{50}$`)/eps))`

0																																																
2	0																																															
0	2	0																																														
1	0	2	0																																													
0	1	0	2	0																																												
1	0	1	0	2	0																																											
0	1	0	1	0	2	0																																										
1	0	1	0	1	0	2	0																																									
0	1	1	1	0	1	0	2	0																																								
1	1	1	1	1	1	1	0	2	0																																							
1	1	1	1	1	1	1	1	0	2	0																																						
1	1	1	1	1	1	1	1	1	0	2	0																																					
1	1	1	1	1	1	1	1	1	1	0	2	0																																				
1	1	1	1	1	1	1	1	1	1	1	0	2	0																																			
1	1	1	1	1	2	1	1	1	1	1	1	0	2	0																																		
1	1	2	1	2	1	2	1	2	1	1	1	1	0	2	0																																	
1	2	1	2	2	2	2	2	1	2	1	1	1	1	0	2	0																																
1	2	2	2	2	2	2	2	2	1	2	1	2	1	1	0	2	0																															
1	2	2	2	2	2	2	2	2	2	2	2	1	2	1	1	0	2	0																														
2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	1	1	0	2	0																													
2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	1	1	0	2	0																												
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	1	2	1	1	0	2	0																											
2	2	2	3	2	3	2	3	2	2	2	2	2	2	2	2	2	2	1	2	0	2	0																										
2	2	3	3	3	3	3	3	3	2	3	2	2	2	2	2	2	2	2	1	2	0	2	0																									
2	3	3	3	3	3	3	3	3	3	3	3	2	3	2	2	2	2	2	2	1	2	0	2	0																								
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2	1	2	0	2	0																							
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	2	2	2	2	1	2	0	2	0																						
3	3	4	3	4	3	4	3	4	3	3	3	3	3	3	3	3	3	3	2	2	2	2	1	2	0	2	0																					
3	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	2	2	2	2	1	2	0	2	0																				
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	2	2	2	2	1	2	0	2	0																			
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	4	3	3	3	3	2	3	2	2	1	2	0	2	0																		
4	4	5	5	5	5	5	5	5	4	5	4	4	4	4	4	4	4	4	3	3	3	3	3	3	2	2	1	2	0	2	0																	
4	5	5	5	5	5	5	5	5	5	5	5	4	5	4	4	4	4	4	4	3	3	3	3	3	3	2	2	2	2	0	2	0																
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	5	4	4	4	4	4	4	3	3	3	3	2	2	2	2	0	2	0															
5	6	5	6	6	6	6	6	5	6	5	5	5	5	5	5	5	5	4	4	4	4	4	4	3	3	3	3	2	2	2	2	0	2	0														
6	6	6	6	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5	5	5	4	4	4	4	3	3	3	3	2	2	2	2	0	2	0													
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	5	5	5	5	5	5	4	4	4	4	4	4	3	3	2	3	2	2	0	2	0												
6	6	7	7	7	7	7	7	7	7	7	6	6	6	6	6	6	6	6	5	5	5	5	5	5	4	4	4	4	3	3	3	3	2	2	0	2	0											
7	7	7	7	7	7	7	7	7	7	7	7	7	7	6	7	6	6	6	6	6	6	5	5	5	5	4	4	4	4	3	3	3	3	2	2	0	2	0										
7	7	8	7	8	8	8	7	8	7	7	7	7	7	7	7	7	7	7	6	6	6	6	5	5	5	5	5	5	4	4	3	3	3	3	2	2	0	2	0									
8	8	8	8	8	8	8	8	8	8	8	8	8	8	7	7	7	7	7	7	7	7	6	6	6	6	5	5	5	5	4	4	4	4	3	3	2	2	0	2	0								
7	7	8	8	8	8	8	7	8	7	7	7	7	7	7	7	7	7	7	6	6	6	6	5	6	5	5	5	5	4	4	3	3	4	4	3	3	2	2	0	2	0							
2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3	4	4	4	3	3	2	2	0	2	0						
2	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2	4	4	4	4	4	4	4	3	3	2	2	0	2	0					
3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4	4	4	4	4	4	4	4	3	3	2	2	0	2	0				
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	4	4	5	5	5	5	5	5	5	4	4	3	4	2	3	0	2	0			
4	4	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	5	4	5	5	5	5	5	5	5	5	5	5	5	4	4	3	3	0	2	0		
5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	6	5	6	6	6	6	6	6	6	5	5	5	5	4	4	3	3	0	2	0	
6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	5	6	6	6	6	7	6	7	6	7	6	6	6	6	5	5	4	4	3	3	0	2	0	
7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	6	7	6	7	7	7	7	7	8	7	7	7	7	7	6	6	6	6	5	5	3	3	0	2	0

Table 10.3: Matlab demo on the loss of orthogonality among Lanczos vectors: Lanczos with partial reorthogonalization. `round(log10(abs(I-W50)/eps))`

Let $\hat{Q}_j := [Q_1, Q_2, \ldots, Q_j]$ be the Krylov basis generated by Algorithm 10.5. Then, in this basis, the projection of A is the block tridiagonal matrix $\hat{T}_j$

$$\hat{Q}_j^* A \hat{Q}_j = \hat{T}_j = \begin{pmatrix} A_1 & B_1^* & & \\ B_1 & A_2 & \ddots & \\ & \ddots & \ddots & B_{j-1}^* \\ & & B_{j-1} & A_j \end{pmatrix}, \quad A_i, B_i \in \mathbb{R}^{p\times p}.$$

If matrices B_i are chosen to be upper triangular, then $\hat{T}_j$ is a band matrix with bandwidth $2p+1$!

Similarly as in scalar case, in the j-th iteration step we obtain the equation

$$A\hat{Q}_j - \hat{Q}_j\hat{T}_j = Q_{j+1}B_jE_j^* + \hat{F}_j, \qquad E_j = \begin{pmatrix} O \\ \vdots \\ O \\ I_p \end{pmatrix},$$

where $\hat{F}_j$ accounts for the effect of roundoff error. Let $(\vartheta_i, \mathbf{y}_i)$ be a Ritz pair of A in $\mathcal{K}^{jp}(Q_1)$. Then

$$\mathbf{y}_i = \hat{Q}_j\mathbf{s}_i, \qquad \hat{T}_j\mathbf{s}_i = \vartheta_i\mathbf{s}_i.$$

As before, we can consider the residual norm to study the accuracy of the Ritz pair $(\vartheta_i, \mathbf{y}_i)$ of A

$$\|A\mathbf{y}_i - \vartheta_i\mathbf{y}_i\| = \|A\hat{Q}_j\mathbf{s}_i - \vartheta_i\hat{Q}_j\mathbf{s}_i\| \approx \|Q_{j+1}B_jE_j^*\mathbf{s}_i\| = \left\| B_j \begin{pmatrix} s_{j(p-1)+1,i} \\ \vdots \\ s_{jp+1,i} \end{pmatrix} \right\|.$$

We have to compute the bottom p components of the eigenvectors $\mathbf{s}_i$ in order to test for convergence.

Similarly as in the scalar case, the mutual orthogonality of the Lanczos vectors (i.e., the columns of $\hat{Q}_j$) is lost, as soon as convergence sets in. The remedies described earlier are available: full reorthogonalization or selective orthogonalization.

10.8 External selective reorthogonalization

If many eigenvalues are to be computed with the Lanczos algorithm, it is usually advisable to execute shift-and-invert Lanczos with *varying shifts* [4].

In each new start of a Lanczos procedure, one has to prevent the algorithm from finding already computed eigenpairs. We have encountered this problem when we tried to compute multiple eigenpairs by simple vector iteration. Here, the remedy is the same as there. In the second and further runs of the Lanczos algorithm, the starting vectors are made orthogonal to the already computed eigenvectors. We know that in theory all Lanczos vectors will be orthogonal to the previously computed eigenvectors. However, because the previous eigenvectors have been computed only approximately the initial vectors are not orthogonal to the true eigenvectors. Because of this and because of floating point errors loss of orthogonality is observed. The loss of orthogonality can be monitored similarly as with partial reorthogonalization. For details see [4].

Bibliography

[1] W. E. Arnoldi, *The principle of minimized iterations in the solution of the matrix eigenvalue problem*, Quarterly of Applied Mathematics, 9 (1951), pp. 17–29.

[2] J. K. Cullum and R. A. Willoughby, *Lanczos Algorithms for Large Symmetric Eigenvalue Computations*, vol. 1: Theory, Birkhäuser, Boston, 1985.

[3] G. H. Golub and J. H. Welsch, *Calculation of Gauss quadrature rules*, Math. Comp., 23 (1969), pp. 221–230.

[4] R. Grimes, J. G. Lewis, and H. Simon, *A shifted block Lanczos algorithm for solving sparse symmetric generalized eigenproblems*, SIAM J. Matrix Anal. Appl., 15 (1994), pp. 228–272.

[5] N. Krylov and N. Bogoliubov, *Sur le calcul des racines de la transcendante de Fredholm les plus voisines d'une nombre donné par les méthodes des moindres carres et de l'algorithme variationel*, Izv. Akad. Naik SSSR, Leningrad, (1929), pp. 471–488.

[6] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, J. Res. Nat. Bureau Standards, Sec. B, 45 (1950), pp. 255–282.

[7] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice Hall, Englewood Cliffs, NJ, 1980. (Republished by SIAM, Philadelphia, 1998.).

[8] H. Simon, *Analysis of the symmetric Lanczos algorithm with reorthogonalization methods*, Linear Algebra Appl., 61 (1984), pp. 101–132.

[9] ——, *The Lanczos algorithm with partial reorthogonalization*, Math. Comp., 42 (1984), pp. 115–142.

Chapter 11

Restarting Arnoldi and Lanczos algorithms

The number of iteration steps can be very high with the Arnoldi or the Lanczos algorithm. This number is, of course, not predictable. The iteration count depends on properties of the matrix, in particular the distribution of its eigenvalues, but also on the initial vectors.

High iteration counts entail a large memory requirement to store the Arnoldi/Lanczos vectors and a high amount of computation because of growing cost of the reorthogonalization.

The idea behind the implicitely restarted Arnoldi (IRA) and implicitely restarted Lanczos (IRL) algorithms is to reduce these costs by limiting the dimension of the search space. This means that the iteration is stopped after a number of steps (which is bigger than the number of desired eigenvalues), reduce the dimension of the search space without destroying the Krylov space structure, and finally resume the Arnoldi / Lanczos iteration.

The implicitely restarted Arnoldi has first been proposed by Sorensen [7, 8]. It is implemented together with the implicitely restarted Lanczos algorithms in the software package ARPACK [4]. The ARPACK routines are the basis for the sparse matrix eigensolver `eigs` in Matlab.

11.1 The m-step Arnoldi iteration

Algorithm 11.1 The m-step Arnoldi iteration

```
Let $A \in \mathbb{F}^{n\times n}$. This algorithm executes $m$ steps of the Arnoldi algorithm.
$\mathbf{q}_1 = \mathbf{x}/\|\mathbf{x}\|;\quad \mathbf{z} = A\mathbf{q}_1;\quad \alpha_1 = \mathbf{q}_1^*\mathbf{z};$
$\mathbf{r}_1 = \mathbf{w} - \alpha_1\mathbf{q}_1;\quad Q_1 = [\mathbf{q}_1];\quad H_1 = [\alpha_1];$
for $j = 1, \dots, m-1$ do
    $\beta_j := \|\mathbf{r}_j\|;\quad \mathbf{q}_{j+1} = \mathbf{r}_j/\beta_j;$
    $Q_{j+1} := [Q_j, \mathbf{q}_{j+1}];\quad \hat{H}_j := \begin{bmatrix} H_j \\ \beta_j\mathbf{e}_j^T \end{bmatrix} \in \mathbb{F}^{(j+1)\times j};$
    $\mathbf{z} := A\mathbf{q}_j;$
    $\mathbf{h} := Q_{j+1}^*\mathbf{z};\quad \mathbf{r}_{j+1} := \mathbf{z} - Q_{j+1}\mathbf{h};$
    $H_{j+1} := [\hat{H}_j, \mathbf{h}];$
end for
```

We start with the Algorithm 11.1 that is a variant of the Arnoldi Algorithm 10.1. It executes just m Arnoldi iteration steps. We will now show how the dimension of the search space is reduced withouth losing the information regarding the eigenvectors one is looking for.

Remark 11.1. Step 8 in Algorithm 11.1 is classical Gram–Schmidt orthogonalization. As

$$\mathbf{r}_{j+1} = \mathbf{z} - Q_{j+1}\mathbf{h} = \mathbf{z} - Q_{j+1}Q_{j+1}^*\mathbf{z},$$

we formally have $Q_{j+1}^*\mathbf{r}_{j+1} = \mathbf{0}$. However, classical Gram–Schmidt orthogonalization is faster but not so accurate as modified Gram–Schmidt orthogonalization [1]. So, often, $Q_{j+1}^*\mathbf{r}_{j+1}$ is quite large. Therefore, the orthogonalization is iterated to get sufficient orthogonality.

A possible modification of step 8 that incorporates a second iteration is

```
8: $\mathbf{h} := Q_{j+1}^*\mathbf{z}$;   $\mathbf{r}_{j+1} := \mathbf{z} - Q_{j+1}\mathbf{h}$;
   $\mathbf{c} := Q_{j+1}^*\mathbf{r}_{j+1}$;   $\mathbf{r}_{j+1} := \mathbf{r}_{j+1} - Q_{j+1}\mathbf{c}$;   $\mathbf{h} = \mathbf{h} + \mathbf{c}$;
```

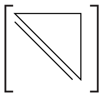
Now we have,

$$\begin{aligned}
\tilde{\mathbf{r}}_{j+1} &= \text{corrected } \mathbf{r}_{j+1}\\
&= \mathbf{r}_{j+1} - Q_{j+1}\underbrace{Q_{j+1}^*\mathbf{r}_{j+1}}_{\mathbf{c}}\\
&= \mathbf{z} - Q_{j+1}\underbrace{Q_{j+1}^*\mathbf{z}}_{\mathbf{h}} - Q_{j+1}\underbrace{Q_{j+1}^*\mathbf{r}_{j+1}}_{\mathbf{c}} = \mathbf{z} - Q_{j+1}(\mathbf{h}+\mathbf{c})
\end{aligned}$$

More iterations are possible but seldom necessary. □

After the execution of Algorithm 11.1 we have the Arnoldi / Lanczos relation

$$AQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*, \qquad H_m = \tag{11.1}$$



available with

$$\mathbf{r}_m = \beta_m \mathbf{q}_{m+1}, \qquad \|\mathbf{q}_{m+1}\| = 1.$$

If $\beta_m = 0$ then $\mathcal{R}(Q_m)$ is invariant under A, i.e., $A\mathbf{x} \in \mathcal{R}(Q_m)$ for all $\mathbf{x} \in \mathcal{R}(Q_m)$. This lucky situation implies that $\sigma(H_m) \subset \sigma_m(A)$. So, the Ritz values and vectors are eigenvalues and eigenvectors of A.

What we can realistically hope for is β_m being small. Then,

$$AQ_m - \mathbf{r}_m\mathbf{e}_m^* = (A - \mathbf{r}_m\mathbf{q}_m^*)Q_m = Q_m H_m.$$

Then, $\mathcal{R}(Q_m)$ is invariant under a matrix $A+E$, that differs from A by a perturbation E with $\|E\| = \|\mathbf{r}_m\| = |\beta_m|$. From general eigenvalue theory we know that in this situation well-conditioned eigenvalues of H_m are good approximations of eigenvalues of A.

In the sequel we investigate how we can find a q_1 such that β_m becomes small?

11.2 Implicit restart

Let us start from the Arnoldi relation

$$AQ_m = Q_m H_m + \mathbf{r}_m\mathbf{e}_m^*, \tag{11.2}$$

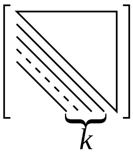
Algorithm 11.2 k implicit QR steps applied to H_m

```
$H_m^+ := H_m$.
for $i := 1, \ldots, k$ do
    $H_m^+ := V_i^* H_m^+ V_i$,    where $H_m^+ - \mu_i I = V_i R_i$   (QR factorization)
end for
```

that is obtained after calling Algorithm 11.1.

We apply $k < m$ implicit QR steps to H_m with shifts $\mu_1, \ldots, \mu_k$, see Algorithm 11.2. Let $V^+ := V_1 V_2 \cdots V_k$. V^+ is the product of k (unitary) Hessenberg matrices whence it has k nonzero off-diagonals below its main diagonal.

$V_m^+ =$



.

We define

$$Q_m^+ := Q_m V^+, \qquad H_m^+ := (V^+)^* H_m V^+.$$

Then, from (11.2) we obtain

$$AQ_m V^+ = Q_m V^+ (V^+)^* H_m V^+ + \mathbf{r}_m \mathbf{e}_m^* V^+,$$

or

$$AQ_m^+ = Q_m^+ H_m^+ + \mathbf{r}_m \mathbf{e}_m^* V^+. \tag{11.3}$$

As V^+ has k nonzero off-diagonals below the main diagonal, the last row of V^+ has the form

$$\mathbf{e}_m^* V^+ = (\underbrace{0, \ldots, 0}_{p-1}, \underbrace{*, \ldots, *}_{k+1}), \qquad k + p = m.$$

We now simply discard the last k columns of (11.3).

$$\begin{aligned} AQ_m^+(:, 1:p) &= Q_m^+ H_m^+(:, 1:p) + \mathbf{r}_m \mathbf{e}_m^* V^+(:, 1:p) \\ &= Q_m^+(:, 1:p) H_m^+(1:p, 1:p) + \underbrace{h_{p+1,p}^+}_{\beta_p^+} \mathbf{q}_{p+1}^+ \mathbf{e}_p^* + v_{m,p}^+ \mathbf{r}_m \mathbf{e}_p^* \\ &= Q_m^+(:, 1:p) H_m^+(1:p, 1:p) + \underbrace{(\mathbf{q}_{p+1}^+ h_{p+1,p}^+ + \mathbf{r}_m v_{m,p}^+)}_{\mathbf{r}_p^+} \mathbf{e}_p^*. \end{aligned}$$

In Algorithm 11.3 we have collected what we have derived so far. We have however left open in step 3 of the algorithm *how* the shifts $\mu_1, \ldots, \mu_k$ should be chosen. In ARPACK [4], all eigenvalues of H_m are computed. Those k eigenvalues that are furthest away from some target value are chosen as shifts. We have not specified how we determine convergence, too.

One can show that a QR step with shift μ_i transforms the vector $\mathbf{q}_1$ in a multiple of $(A - \mu_i I)\mathbf{q}_1$. In fact, a simple modification of the Arnoldi relation (11.2) gives

$$(A - \mu_i I) Q_m = Q_m \underbrace{(H_m - \mu_i I)}_{V_1 R_1} + \mathbf{r}_m \mathbf{e}_m^* = Q_m V_1 R_1 + \mathbf{r}_m \mathbf{e}_m^*.$$

Algorithm 11.3 Implicitely restarted Arnoldi (IRA)

```
Let the Arnoldi relation $AQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*$ be given.
repeat
  Determine $k$ shifts $\mu_1, \ldots, \mu_k$;
  $\mathbf{v}^* := \mathbf{e}_m^*$;
  for $i = 1, \ldots, k$ do
    $H_m - \mu_i I = V_i R_i$; /* QR factorization */
    $H_m := V_i^* H_m V_i$;   $Q_m := Q_m V_i$;
    $\mathbf{v}^* := \mathbf{v}^* V_i$;
  end for
  $\mathbf{r}_p := \mathbf{q}_{p+1}^+ \beta_p^+ + \mathbf{r}_m v_{m,p}^+$;
  $Q_p := Q_m(:, 1:p)$;   $H_p := H_m(1:p, 1:p)$;
  Starting with
      $AQ_p = Q_p H_p + \mathbf{r}_p \mathbf{e}_p^*$
  execute $k$ additional steps of the Arnoldi algorithm until
      $AQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*$.
until convergence
```

Comparing the first columns in this equation gives

$$(A - \mu_i I)\mathbf{q}_1 = Q_m V_1 \mathbf{e}_1 r_{11} + \mathbf{0} = \mathbf{q}_1^{(1)} r_{11}.$$

By consequence, all k steps combined give

$$\mathbf{q}_1 \longleftarrow \Psi(A)\mathbf{q}_1, \qquad \Psi(\lambda) = \prod_{i=1}^{k} (\lambda - \mu_i).$$

If μ_i were an eigenvalue of A then $(A - \mu_i I)\mathbf{q}_1$ removes components of $\mathbf{q}_1$ in the direction of the corresponding eigenvector. More general, if μ_i is close to an eigenvalue of A then $(A - \mu_i I)\mathbf{q}_1$ will have only small components in the direction of eigenvectors corresponding to nearby eigenvalues. Choosing the μ_i equal to Ritz values far away from the desired part of the spectrum thus enhances the desired component. Still there is the danger that in each sweep on Algorithm 11.3 the same undesired Ritz values are recovered. Therefore, other strategies for choosing the shifts have been proposed [2]. Experimental results indicate however, that the original strategy chosen in ARPACK mostly works best.

11.3 Convergence criterion

Let $H_m \mathbf{s} = \mathbf{s}\vartheta$ with $\|\mathbf{s}\| = 1$. Let $\hat{\mathbf{x}} = Q_m \mathbf{s}$. Then we have as earlier

$$\|A\hat{\mathbf{x}} - \vartheta\hat{\mathbf{x}}\| = \|AQ_m \mathbf{s} - Q_m H_m \mathbf{s}\| = \|\mathbf{r}_m\| |\mathbf{e}_m^* \mathbf{s}| = \beta_m |\mathbf{e}_m^* \mathbf{s}|. \tag{11.4}$$

In the Hermitian case, $A = A^*$, the Theorem 10.1 of Krylov–Bogoliubov provides an interval that contains an eigenvalue of A. In the general case, we have

$$(A + E)\hat{\mathbf{x}} = \vartheta\hat{\mathbf{x}}, \qquad E = -\mathbf{r}_m \mathbf{q}_m^*, \quad \|E\| = \|\mathbf{r}_m\| = \beta_m. \tag{11.5}$$

According to an earlier theorem we know that if $\lambda \in \sigma(A)$ is simple and ϑ is the eigenvalue of $A+E$ closest to λ, then

$$|\lambda - \vartheta| \le \frac{\|E\|}{\mathbf{y}^*\mathbf{x}} + \mathcal{O}(\|E\|^2). \tag{11.6}$$

Here, $\mathbf{y}$ and $\mathbf{x}$ are left and right eigenvectors of E corresponding to the eigenvalue λ. A similar statement holds for the eigenvectors, but the distance (gap) to the next eigenvalue comes into play as well.

In ARPACK, a Ritz pair $(\vartheta, \hat{\mathbf{x}})$ is considered converged if

$$\beta_m |\mathbf{e}_m^* \mathbf{s}| \le \max(\varepsilon_M \|H_m\|, \mathrm{tol} \cdot |\vartheta|). \tag{11.7}$$

As $|\vartheta| \le \|H_m\| \le \|A\|$, the inequality $\|E\| \le \mathrm{tol} \cdot \|A\|$ holds at convergence. According to (11.6) well-conditioned eigenvalues are well approximated.

11.4 The generalized eigenvalue problem

Let us consider now the generalized eigenvalue problem

$$A\mathbf{x} = \lambda M \mathbf{x}. \tag{11.8}$$

Applying a shift-and-invert spectral transformation with shift σ transforms (11.8) into

$$S\mathbf{x} = (A - \sigma M)^{-1} M \mathbf{x} = \mu \mathbf{x}, \qquad \mu = \frac{1}{\lambda - \sigma}. \tag{11.9}$$

We now execute an Arnoldi/Lanczos iteration with S to obtain

$$SQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*, \qquad Q_m^* M Q_m = I_m, \quad Q_m^* M \mathbf{r}_m = \mathbf{0}. \tag{11.10}$$

Let $\mathbf{s}$ with $\|\mathbf{s}\| = 1$ be an eigenvector of H_m with Ritz value ϑ. Let $\mathbf{y} = Q_m \mathbf{s}$ be the associated Ritz vector. Then,

$$SQ_m \mathbf{s} = S\mathbf{y} = Q_m H_m \mathbf{s} + \mathbf{r}_m \mathbf{e}_m^* \mathbf{s} = \mathbf{y}\vartheta + \mathbf{r}_m \mathbf{e}_m^* \mathbf{s}. \tag{11.11}$$

So, $\mathbf{y}\vartheta + \mathbf{r}_m \mathbf{e}_m^* \mathbf{s}$ can be considered a vector that is obtained by one step of inverse iteration. This vector is an improved approximation to the desired eigenvector, obtained at negligible cost. This so-called **eigenvector purification** is particularly important if M is singular.

Let us bound the residual norm of the purified vector. With (11.11) we have

$$M\mathbf{y} = (A - \sigma M)\underbrace{(\mathbf{y}\vartheta + \mathbf{r}_m \mathbf{e}_m^* \mathbf{s})}_{\tilde{\mathbf{y}}} \tag{11.12}$$

with

$$\|\tilde{\mathbf{y}}\|_M = \sqrt{\vartheta^2 + \beta_k^2 |\mathbf{e}_m^* \mathbf{s}|^2}.$$

This equality holds as $\mathbf{y} \perp_M \mathbf{r}$. By consequence,

$$\begin{aligned} \|A\tilde{\mathbf{y}} - \lambda M \tilde{\mathbf{y}}\| &= \|(A - \sigma M)\tilde{\mathbf{y}} + M\tilde{\mathbf{y}}\underbrace{(\sigma - \lambda)}_{-\frac{1}{\vartheta}}\| \\ &= \|M\mathbf{y} - M(\mathbf{y}\vartheta + \mathbf{r}_m \mathbf{e}_m^* \mathbf{s})/\vartheta\| = \|M\mathbf{r}\| \; |\mathbf{e}_m^* \mathbf{s}|/|\vartheta|. \end{aligned} \tag{11.13}$$

Since $|\vartheta|$ is large in general, we obtain good bounds for the residual of the purified eigenvectors.

```
EIGS  Find a few eigenvalues and eigenvectors of a matrix using ARPACK
   D = EIGS(A) returns a vector of A’s 6 largest magnitude eigenvalues.
   A must be square and should be large and sparse.

   [V,D] = EIGS(A) returns a diagonal matrix D of A’s 6 largest magnitude
   eigenvalues and a matrix V whose columns are the corresponding
   eigenvectors.

   [V,D,FLAG] = EIGS(A) also returns a convergence flag. If FLAG is 0 then
   all the eigenvalues converged; otherwise not all converged.

   EIGS(A,B) solves the generalized eigenvalue problem A*V == B*V*D. B
   must be symmetric (or Hermitian) positive definite and the same size as
   A. EIGS(A,[],...) indicates the standard eigenvalue problem A*V == V*D.

   EIGS(A,K) and EIGS(A,B,K) return the K largest magnitude eigenvalues.

   EIGS(A,K,SIGMA) and EIGS(A,B,K,SIGMA) return K eigenvalues. If SIGMA is:
      ’LM’ or ’SM’ - Largest or Smallest Magnitude
   For real symmetric problems, SIGMA may also be:
      ’LA’ or ’SA’ - Largest or Smallest Algebraic
      ’BE’ - Both Ends, one more from high end if K is odd
   For nonsymmetric and complex problems, SIGMA may also be:
      ’LR’ or ’SR’ - Largest or Smallest Real part
      ’LI’ or ’SI’ - Largest or Smallest Imaginary part
   If SIGMA is a real or complex scalar including 0, EIGS finds the
   eigenvalues closest to SIGMA. For scalar SIGMA, and when SIGMA = ’SM’,
   B need only be symmetric (or Hermitian) positive semi-definite since it
   is not Cholesky factored as in the other cases.

   EIGS(A,K,SIGMA,OPTS) and EIGS(A,B,K,SIGMA,OPTS) specify options:
   OPTS.issym: symmetry of A or A-SIGMA*B represented by AFUN [{false} | true]
   OPTS.isreal: complexity of A or A-SIGMA*B represented by AFUN [false | {true}]
   OPTS.tol: convergence: Ritz estimate residual <= tol*NORM(A) [scalar | {eps}]
   OPTS.maxit: maximum number of iterations [integer | {300}]
   OPTS.p: number of Lanczos vectors: K+1<p<=N [integer | {2K}]
   OPTS.v0: starting vector [N-by-1 vector | {randomly generated}]
   OPTS.disp: diagnostic information display level [0 | {1} | 2]
   OPTS.cholB: B is actually its Cholesky factor CHOL(B) [{false} | true]
   OPTS.permB: sparse B is actually CHOL(B(permB,permB)) [permB | {1:N}]
   Use CHOL(B) instead of B when SIGMA is a string other than ’SM’.

   EIGS(AFUN,N) accepts the function AFUN instead of the matrix A. AFUN is
   a function handle and Y = AFUN(X) should return
      A*X            if SIGMA is unspecified, or a string other than ’SM’
      A\X            if SIGMA is 0 or ’SM’
      (A-SIGMA*I)\X  if SIGMA is a nonzero scalar (standard problem)
      (A-SIGMA*B)\X  if SIGMA is a nonzero scalar (generalized problem)
   N is the size of A. The matrix A, A-SIGMA*I or A-SIGMA*B represented by
   AFUN is assumed to be real and nonsymmetric unless specified otherwise
   by OPTS.isreal and OPTS.issym. In all these EIGS syntaxes, EIGS(A,...)
   may be replaced by EIGS(AFUN,N,...).

   Example:
      A = delsq(numgrid(’C’,15));  d1 = eigs(A,5,’SM’);

   Equivalently, if dnRk is the following one-line function:
      %----------------------------%
      function y = dnRk(x,R,k)
      y = (delsq(numgrid(R,k))) \ x;
      %----------------------------%

      n = size(A,1);  opts.issym = 1;
      d2 = eigs(@(x)dnRk(x,’C’,15),n,5,’SM’,opts);

   See also eig, svds, ARPACKC, function_handle.
```

11.5 A numerical example

This example is taken from the Matlab document pages regarding `eigs`. `eigs` is the Matlab interface to the ARPACK code, see page 200. The matrix called `west0479` is a 479 × 479 matrix originating in a chemical engineering plant model. The matrix is available from the Matrix Market [5], a web site that provides numerous test matrices. Its nonzero structure is given in Fig. 11.1

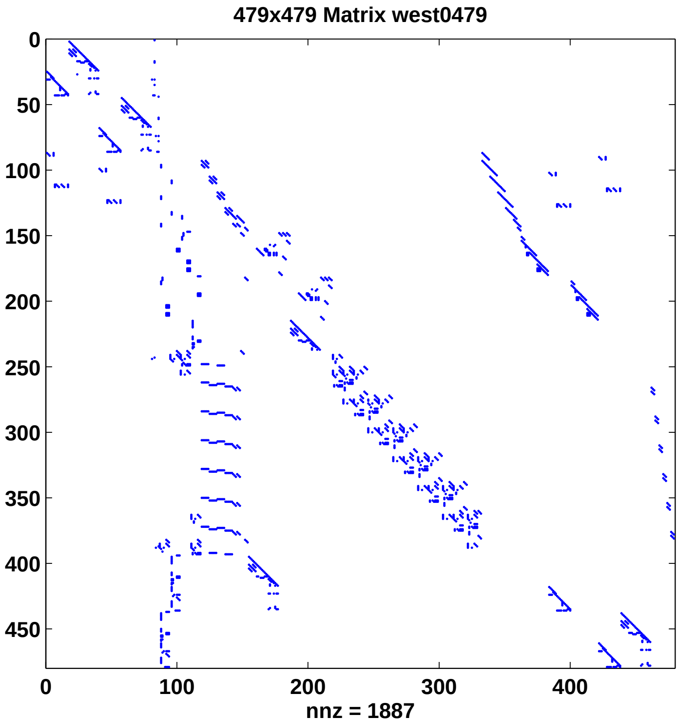


Figure 11.1: Nonzero structure of the 479 × 479 matrix west0479

To compute the eight largest eigenvalues of this matrix we issue the following Matlab commands.

```
>> load west0479
>> d = eig(full(west0479));
>> dlm=eigs(west0479,8);
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0561 - 0.0536i
```

```
   0.1081 + 0.0541i
   0.1081 - 0.0541i
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 3: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0866
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
   0.1081 - 0.0541i
   0.1081 + 0.0541i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 4: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

   0.0614 - 0.0465i
  -0.0072 - 0.1207i
  -0.0072 + 0.1207i
   0.1081 + 0.0541i
   0.1081 - 0.0541i
  -0.1009 + 0.0666i
  -0.1009 - 0.0666i
   0.0000 - 1.7007i
   0.0000 + 1.7007i

Iteration 5: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0808
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
  -0.1009 - 0.0666i
  -0.1009 + 0.0666i
   0.1081 + 0.0541i
   0.1081 - 0.0541i
   0.0000 + 1.7007i
   0.0000 - 1.7007i

Iteration 6: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

   0.0734 - 0.0095i
  -0.0072 + 0.1207i
  -0.0072 - 0.1207i
   0.1081 - 0.0541i
```

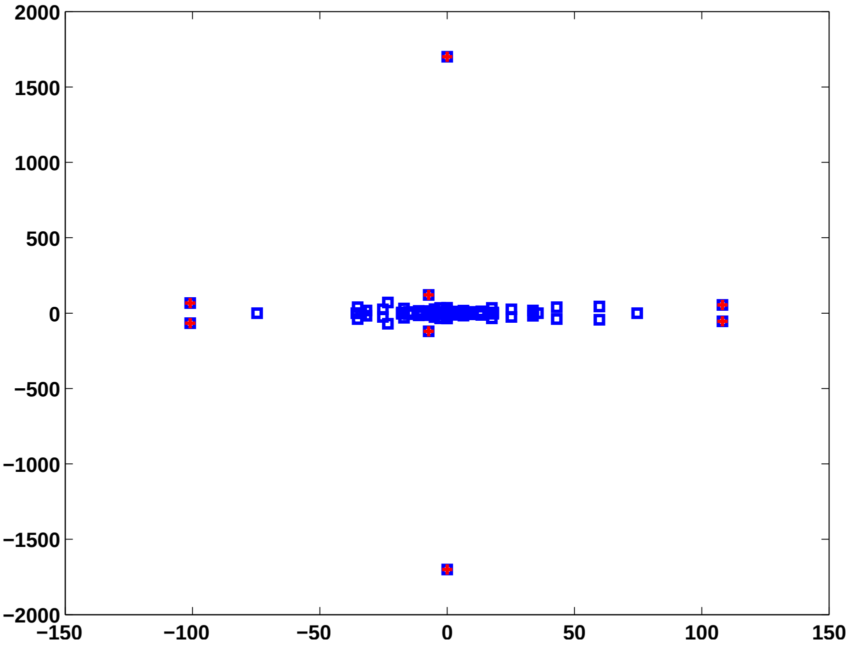


Figure 11.2: Spectrum of the matrix west0479

```
    0.1081 + 0.0541i
   -0.1009 - 0.0666i
   -0.1009 + 0.0666i
    0.0000 - 1.7007i
    0.0000 + 1.7007i

Iteration 7: a few Ritz values of the 20-by-20 matrix:
    1.0e+03 *

   -0.0747
   -0.0072 - 0.1207i
   -0.0072 + 0.1207i
    0.1081 + 0.0541i
    0.1081 - 0.0541i
   -0.1009 + 0.0666i
   -0.1009 - 0.0666i
    0.0000 + 1.7007i
    0.0000 - 1.7007i
```

The output indicates that `eigs` needs seven sweeps to compute the eigenvalues to the default accuracy of macheps$\|A\|$. The Ritz values given are the approximations of the eigenvalues we want to compute. The complete spectrum of `west0479` is given in Fig. 11.2. Notice the different scales of the axes! Fig. 11.3 is a zoom that shows all eigenvalues except the two very large ones. Here the axes are equally scaled. From the two figures it becomes clear that `eigs` has computed the eight eigenvalues (and corresponding eigenvectors) of *largest modulus*.

To compute the eigenvalues smallest in modulus we issue the following command.

```
dsm=eigs(west0479,8,'sm');
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
```

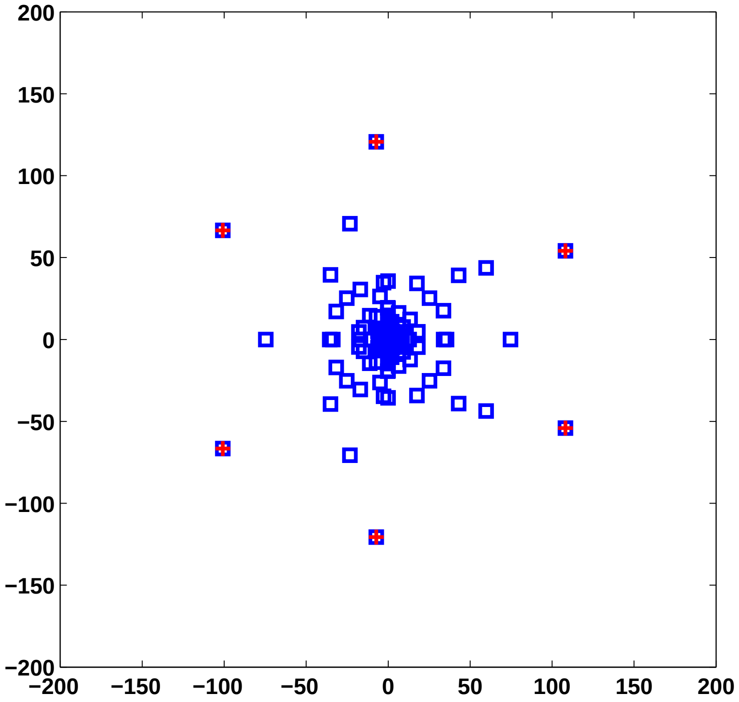


Figure 11.3: A zoom to the center of the spectrum of matrix west0479 that excludes the largest two eigenvalues on the imaginary axis

```
           0
           0
           0
           0
           0
           0
           0
           0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0228 - 0.0334i
   0.0444
  -0.0473
   0.0116 + 0.0573i
   0.0116 - 0.0573i
  -0.0136 - 0.1752i
  -0.0136 + 0.1752i
  -3.4455
   5.8308

Iteration 3: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *
  -0.0228 - 0.0334i
   0.0444
  -0.0473
   0.0116 + 0.0573i
   0.0116 - 0.0573i
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
  -3.4455
   5.8308

Iteration 4: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0228 + 0.0334i
   0.0444
  -0.0473
   0.0116 - 0.0573i
   0.0116 + 0.0573i
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
  -3.4455
   5.8308

Iteration 5: a few Ritz values of the 20-by-20 matrix:
   1.0e+03 *

  -0.0228 + 0.0334i
   0.0444
  -0.0473
   0.0116 - 0.0573i
   0.0116 + 0.0573i
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
  -3.4455
   5.8308

>> dsm

dsm =

   0.0002
  -0.0003
  -0.0004 - 0.0057i
  -0.0004 + 0.0057i
   0.0034 - 0.0168i
   0.0034 + 0.0168i
  -0.0211
   0.0225
```

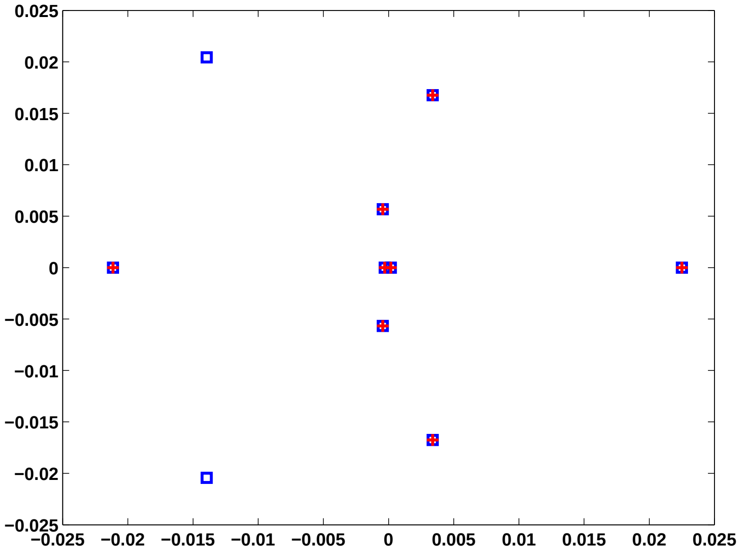


Figure 11.4: Smallest eigenvalues of the matrix west0479

```
>> 1./dsm

ans =

   1.0e+03 *

   5.8308
  -3.4455
  -0.0136 + 0.1752i
  -0.0136 - 0.1752i
   0.0116 + 0.0573i
   0.0116 - 0.0573i
  -0.0473
   0.0444
```

The computed eigenvalues are depicted in Fig. 11.4

11.6 Another numerical example

We revisit the determination the acoustic eigenfrequencies and modes in the interior of a car, see section 1.6.3. The computations are done with the finest grid depicted in Fig. 1.9. We first compute the lowest ten eigenpairs with simultaneous inverse vector iteration (sivit). The dimension of the search space is 15.

```
>> [p,e,t]=initmesh('auto');
>> [p,e,t]=refinemesh('auto',p,e,t);
>> [p,e,t]=refinemesh('auto',p,e,t);
>> p=jigglemesh(p,e,t);
>> [A,M]=assema(p,t,1,1,0);
>> whos
  Name      Size                   Bytes  Class
  A      1095x1095                 91540  double array (sparse)
  M      1095x1095                 91780  double array (sparse)
  e         7x188                  10528  double array
  p         2x1095                 17520  double array
  t         4x2000                 64000  double array

Grand total is 26052 elements using 275368 bytes

>> sigma=-.01;
>> p=10; tol=1e-6; X0=rand(size(A,1),15);
>> [V,L] = sivit(A,M,p,X0,sigma,tol);

 ||Res(0)|| = 0.998973
 ||Res(5)|| = 0.603809
 ||Res(10)|| = 0.0171238
 ||Res(15)|| = 0.00156298
 ||Res(20)|| = 3.69725e-05
 ||Res(25)|| = 7.11911e-07
>> %  25 x 15 =  375 matrix - vektor - multiplications until convergence
>>
>> format long, L

L =

   0.00000000000000
   0.01269007628847
   0.04438457596824
   0.05663501055565
   0.11663116522140
   0.13759210393200
   0.14273438015546
   0.20097619880776
   0.27263682280769
   0.29266080747831

>> format short
>> norm(V'*M*V - eye(10))

ans =

     1.8382e-15
```

Then we use Matlab's solver `eigs`. We set the tolerance and the shift to be the same as with `sivit`. Notice that ARPACK applies a shift-and-invert spectral transformation if a shift is given.

```
>> options.tol=tol; options.issym=1;
>> [v,l,flag]=eigs(A,M,p,sigma,options);
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
    3.3039
    3.5381
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

Iteration 3: a few Ritz values of the 20-by-20 matrix:
    3.3040
    3.5381
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

>> flag

flag =

     0

>> l=diag(l);  l=l(end:-1:1); norm(l-L)

ans =

   3.7671e-14

>> norm(v'*M*v - eye(10))

ans = 8.0575e-15
```

Clearly the eigenvectors are mutually m-orthogonal. Notice that `eigs` returns the eigenvalues sorted from large to small such that they have to be reordered before comparing with those `sivit` computed.

In the next step we compute the largest eigenvalues of the matrix

$$S = R(A - \sigma M)^{-1} R^T, \tag{11.14}$$

where $R^T R = M$ is the Cholesky factorization of M. The matrix in (11.14) is transferred to `eigs` as a function.

```
>> type afun
```

```
function x = afun(x)
global RA RB

x = RB*(RA\(RA’\(RB’*x)));

>> global RA RB
>> RA = chol(A-sigma*M);
>> RB = chol(M);
>> [v,l1,flag]=eigs(’afun’,n,10,’lm’,options);
Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
    3.3030
    3.5380
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

Iteration 3: a few Ritz values of the 20-by-20 matrix:
    3.3040
    3.5381
    4.7399
    6.5473
    6.7754
    7.8970
   15.0071
   18.3876
   44.0721
  100.0000

>> flag

flag =

     0

>> l1 = diag(l1)

l1 =
  100.0000
   44.0721
   18.3876
   15.0071
    7.8970
    6.7754
    6.5473
    4.7399
    3.5381
    3.3040

>> sigma + 1./l1

ans =

    0.0000
    0.0127
    0.0444
    0.0566
    0.1166
    0.1376
    0.1427
    0.2010
    0.2726
    0.2927

>> norm(sigma + 1./l1 - l)

ans =

   4.4047e-14
```

11.7 The Lanczos algorithm with thick restarts

The implicit restarting procedures discussed so far are very clever ways to get rid of unwanted directions in the search space and still keeping a Lanczos or Arnoldi basis. The latter admits to continue the iteration in a known framework. The Lanczos or Arnoldi relations hold that admit very efficient checks for convergence. The restart has the effect of altering the starting vector.

In this and the next section we discuss algorithms that work with Krylov spaces but are not restricted to Krylov or Arnoldi bases. Before continuing we make a step back and consider how we can determine if a given subspace of $\mathbb{F}^n$ is a Krylov space at all.

Let A be an n-by-n matrix and let $\mathbf{v}_1, \ldots, \mathbf{v}_k$ be linearly independent n-vectors. Is the subspace $\mathcal{V} := \text{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ a Krylov space, i.e., is there a vector $\mathbf{q} \in \mathcal{V}$ such that $\mathcal{V} = \mathcal{K}_k(A, \mathbf{q})$? The following theorem gives the answer [3, 10].

Theorem 11.1 *$\mathcal{V} = span\{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ is a Krylov space if and only if there is a k-by-k matrix M such that*

$$R := AV - VM, \qquad V = [\mathbf{v}_1, \ldots, \mathbf{v}_k], \tag{11.15}$$

has rank one and $span\{\mathbf{v}_1, \ldots, \mathbf{v}_k, \mathcal{R}(R)\}$ has dimension $k + 1$.

Proof. Let us first assume that $\mathcal{V} = \mathcal{K}_k(A, \mathbf{q})$ for some $\mathbf{q} \in \mathcal{V}$. Let $Q = [\mathbf{q}_1, \dots, \mathbf{q}_k]$ be the Arnoldi basis of $\mathcal{K}_k(A, \mathbf{q})$. Then $Q = VS$ with S a nonsingular k-by-k matrix. We now multiply the Arnoldi relation

$$AQ = QH + \tilde{\mathbf{q}}_{k+1}\mathbf{e}_k^T, \quad Q^*\tilde{\mathbf{q}}_{k+1} = \mathbf{0}, \qquad H \text{ Hessenberg}.$$

by S^{-1} from the right to get

$$AV = VSHS^{-1} + \tilde{\mathbf{q}}_{k+1}\mathbf{e}_k^* S^{-1}.$$

which is (11.15) with $M = SHS^{-1}$.

Let us now assume that R in (11.15) has rank 1 so that we can write

$$AV = VM + R = VM + \mathbf{v}\mathbf{w}^*, \qquad M \in \mathbb{F}^{k\times k}. \tag{11.16}$$

with some $\mathbf{v} \in \mathbb{F}^n$ and $\mathbf{w} \in \mathbb{F}^k$. Let S_1, $S_1^{-1} = S_1^*$, be the Householder reflector that maps $\mathbf{w}$ onto a multiple of $\mathbf{e}_k$, $S_1^*\mathbf{w} = \gamma\mathbf{e}_k$. Then, (11.16) becomes

$$AVS_1 = VS_1S_1^*MS_1 + \gamma\mathbf{v}\mathbf{e}_k^T.$$

There is another unitary matrix S_2 with $S_2^*\mathbf{e}_k = \mathbf{e}_k$ that transforms $S_1^*MS_1$ similarly to Hessenberg form,

$$S^*MS = H, \qquad H \text{ Hessenberg},$$

where $S = S_1S_2$. S_2 can be formed as the product of Householder reflectors. In contrast to the well-known transformation of full matrices to Hessenberg form, here the zeros are generated row-wise starting with the last in order not to destroy $\mathbf{e}_k$ in the rank-1 term. Thus,

$$AVS = VSH + \gamma\mathbf{v}\mathbf{e}_k^T.$$

So, $\mathcal{V} = \mathcal{K}_k(A, \mathbf{q})$ with $\mathbf{q} = VS\mathbf{e}_1$. ■

We apply this theorem to the case where a subspace is spanned by some Ritz vectors. Let $A = A^*$ and let

$$AQ_k - Q_kT_k = \beta_{k+1}\mathbf{q}_{k+1}\mathbf{e}_k^T \tag{11.17}$$

be a Lanczos relation. Let

$$T_kS_k = S_k\Theta_k, \qquad S_k = [\mathbf{s}_1^{(k)}, \dots, \mathbf{s}_k^{(k)}], \quad \Theta_k = \operatorname{diag}(\vartheta_1, \dots, \vartheta_k).$$

be the spectral decomposition of the tridiagonal matrix T_k. Then, for all i, the Ritz vector

$$\mathbf{y}_i = Q_k\mathbf{s}_i^{(k)} \in \mathcal{K}_k(A, \mathbf{q})$$

gives rise to the residual

$$\mathbf{r}_i = A\mathbf{y}_i - \mathbf{y}_i\vartheta_i = \beta_{k+1}\mathbf{q}_{k+1}\mathbf{e}_k^*\mathbf{s}_i^{(k)} \in \mathcal{K}_{k+1}(A, \mathbf{q}) \ominus \mathcal{K}_k(A, \mathbf{q}).$$

Therefore, for any set of indices $1 \le i_1 < \cdots < i_j \le k$ we have

$$A[\mathbf{y}_{i_1}, \mathbf{y}_{i_2}, \dots, \mathbf{y}_{i_j}] - [\mathbf{y}_{i_1}, \mathbf{y}_{i_2}, \dots, \mathbf{y}_{i_j}] \operatorname{diag}(\vartheta_{i_1}, \dots, \vartheta_{i_j}) = \beta_{k+1}\mathbf{q}_{k+1}[s_{i_1}^{(k)}, s_{i_2}^{(k)}, \dots, s_{i_j}^{(k)}].$$

By Theorem 11.1 we see that any set $[\mathbf{y}_{i_1}, \mathbf{y}_{i_2}, \dots, \mathbf{y}_{i_j}]$ of Ritz vectors forms a Krylov space. Note that the generating vector differs for each set.

Algorithm 11.4 Thick restart Lanczos

```
Let us be given $k$ Ritz vectors $\mathbf{y}_i$ and a residual vector $\mathbf{r}_k$ such that $A\mathbf{y}_i = \vartheta_i\mathbf{y}_i + \sigma_i\mathbf{r}_k$, $i = 1,\dots,k$. The value $k$ may be zero in which case $\mathbf{r}_0$ is the initial guess.
This algorithm computes an orthonormal basis $\mathbf{y}_1,\dots,\mathbf{y}_j,\mathbf{q}_{j+1},\dots,\mathbf{q}_m$ that spans a $m$-dimensional Krylov space whose generating vector is not known unless $k = 0$.
$\mathbf{q}_{k+1} := \mathbf{r}_k/\|\mathbf{r}_k\|$.
$\mathbf{z} := A\mathbf{q}_{k+1}$;
$\alpha_{k+1} := \mathbf{q}_{k+1}^*\mathbf{z}$;
$\mathbf{r}_{k+1} = \mathbf{z} - \alpha_{k+1}\mathbf{q}_{k+1} - \sum_{i=1}^{k}\sigma_i\mathbf{y}_i$
$\beta_{k+1} := \|\mathbf{r}_{k+1}\|$
for $i = k+2,\dots,m$ do
    $\mathbf{q}_i := \mathbf{r}_{i-1}/\beta_{i-1}$.
    $\mathbf{z} := A\mathbf{q}_i$;
    $\alpha_i := \mathbf{q}_i^*\mathbf{z}$;
    $\mathbf{r}_i = \mathbf{z} - \alpha_i\mathbf{q}_i - \beta_{i-1}\mathbf{q}_{i-1}$
    $\beta_i = \|\mathbf{r}_i\|$
end for
```

We now split the indices $1,\dots,k$ in two sets. The first set contains the 'good' Ritz vectors that we want to keep and that we collect in Y_1, the second set contains the 'bad' ones that we want to remove. Those we put in Y_2. In this way we get

$$A[Y_1, Y_2] - [Y_1, Y_2]\begin{bmatrix}\Theta_1 & \\ & \Theta_2\end{bmatrix} = \beta_{k+1}\mathbf{q}_{k+1}[\mathbf{s}_1^*, \mathbf{s}_2^*]. \tag{11.18}$$

Keeping the first set of Ritz vectors and **purging** (deflating) the rest yields

$$AY_1 - Y_1\Theta_1 = \beta_{k+1}\mathbf{q}_{k+1}\mathbf{s}_1^*.$$

We now can restart a Lanczos procedure by orthogonalizing $A\mathbf{q}_{k+1}$ against $Y_1 =: [\mathbf{y}_1^*,\dots,\mathbf{y}_j^*]$ and $\mathbf{q}_{k+1}$. From the equation

$$A\mathbf{y}_i - \mathbf{y}_i\vartheta_i = \mathbf{q}_{k+1}\sigma_i, \qquad \sigma_i = \beta_{k+1}\mathbf{e}_k^*\mathbf{s}_i^{(k)}$$

we get

$$\mathbf{q}_{k+1}^*A\mathbf{y}_\ell = \sigma_\ell,$$

whence

$$\mathbf{r}_{k+1} = A\mathbf{q}_{k+1} - \alpha_{k+1}\mathbf{q}_{k+1} - \sum_{i=1}^{j}\sigma_i\mathbf{y}_i \perp \mathcal{K}_{k+1}(A, \mathbf{q}.) \tag{11.19}$$

From this point on the Lanczos algorithm proceeds with the ordinary three-term recurrence. We finally arrive at a relation similar to (11.17), however, with

$$Q_m = [\mathbf{y}_1,\dots,\mathbf{y}_j,\mathbf{q}_{k+1},\dots,\mathbf{q}_{m+k-j}]$$

and

$$T_m = \begin{pmatrix}
\vartheta_1 & & & \sigma_1 & & \\
& \ddots & & \vdots & & \\
& & \vartheta_j & \sigma_j & & \\
\sigma_1 & \cdots & \sigma_j & \alpha_{k+1} & \ddots & \\
& & & \ddots & \ddots & \beta_{m+k-j-1} \\
& & & & \beta_{m+k-j-1} & \alpha_{m+k-j}
\end{pmatrix}$$

This procedure, called **thick restart**, has been suggested by Wu & Simon [11], see Algorithm 11.4. It allows to restart with any number of Ritz vectors. In contrast to the implicitly restarted Lanczos procedure, here we need the spectral decomposition of T_m. Its computation is not an essential overhead in general. The spectral decomposition admits a simple sorting of Ritz values. We could further split the first set of Ritz pairs into converged and unconveregd ones, depending on the value $\beta_{m+1}|s_{k,i}|$. If this quantity is below a given threshold we set the value to zero and **lock** (deflate) the corresponding Ritz vector, i.e., accept it as an eigenvector.

The procedure is mathematically equivalent with the implicitely restarted Lanczos algorithm. In fact, the generating vector of the Krylov space span$\{\mathbf{y}_1,\ldots,\mathbf{y}_j,\mathbf{q}_{j+1},\ldots,\mathbf{q}_m\}$ that we do not compute is $\mathbf{q}_1' = (A-\vartheta_{j+1}I)\cdots(A-\vartheta_m I)\mathbf{q}_1$. This restarting procedure is probably simpler than with IRL.

The problem of losing orthogonality is similar to plain Lanczos. Wu & Simon [11] investigate the various reorthogonalizing strategies known from plain Lanczos (full, selective, partial). In their numerical experiments the simplest procedure, full reorthogonalization, performs similarly or even faster than the more sophisticated reorthogonalization procedures.

Remark 11.2. The thick restart Lanczos procedure does not need a Krylov basis of span$\{\mathbf{y}_1,\ldots,\mathbf{y}_j\}$ or, equivalently, the tridiagonalization of

$$\begin{pmatrix} \vartheta_1 & & & \sigma_1 \\ & \ddots & & \vdots \\ & & \vartheta_j & \sigma_j \\ \sigma_1 & \cdots & \sigma_j & \alpha_{k+1} \end{pmatrix}.$$

However, at the next restart, the computation of the spectral decomposition will most probably require it.

Question: How can the arrow matrix above be tridiagonalized economically? □

11.8 Krylov–Schur algorithm

The Krylov–Schur algorithm introduced by Stewart [9] is a generalization of the thick-restart procedure for non-Hermitian problems. The Arnoldi algorithm constructs the Arnoldi relation

$$AQ_m = Q_m H_m + \mathbf{r}_m \mathbf{e}_m^*, \tag{11.1}$$

where H_m is Hessenberg and $[Q_m, \mathbf{r}_m]$ has full rank. Let $H_m = S_m T_m S_m^*$ be a Schur decomposition of H_m with unitary S_m and triangular T_m. Then, similarly as in the previous section we have

$$AY_m = Y_m T_m + \mathbf{r}_m \mathbf{s}^*, \qquad Y_m = Q_m S_m, \quad \mathbf{s}^* = \mathbf{e}_m^* S_m. \tag{11.20}$$

The upper trangular form of T_m eases the analysis of the individual Ritz pairs. In particular, it admits moving unwanted Ritz values to the lower-right corner of T_m. (See the subroutine `_trexc` in LAPACK for details.) Similarly as in (11.18) we collect the 'good' and 'bad' Ritz vectors in matrices Y_1 and Y_2, respectively. In this way we get

$$A[Y_1, Y_2] - [Y_1, Y_2]\begin{bmatrix} T_{11} & T_{12} \\ & T_{22} \end{bmatrix} = \beta_{k+1}\mathbf{q}_{k+1}[\mathbf{s}_1^*, \mathbf{s}_2^*]. \tag{11.21}$$

Keeping the first set of Ritz vectors and purging the rest yields

$$AY_1 - Y_1T_{11} = \beta_{k+1}\mathbf{q}_{k+1}\mathbf{s}_1^*.$$

In the thick-restart Lanczos procedure we have found an eigenpair as soon as $\beta_{k+1}|s_{ik}|$ is sufficiently small. The determination of a converged subspace with the general Krylov–Schur procedure is not so easy. However, if we manage to bring $\mathbf{s}_1$ into the form

$$\mathbf{s}_1 = \begin{bmatrix} \mathbf{s}_1' \\ \mathbf{s}_1'' \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \mathbf{s}_1'' \end{bmatrix}$$

then we found an invariant subspace.

$$A[Y_1', Y_1''] - [Y_1', Y_1''] \begin{bmatrix} T_{11}' & T_{12}' \\ & T_{22}' \end{bmatrix} = \beta_{k+1}\mathbf{q}_{k+1}[\mathbf{0}^T, \mathbf{s}_1''^*]$$

i.e.,

$$AY_1' = Y_1'T_{11}'$$

In most cases $\mathbf{s}_1'$ consists of a single *small* element or of two small elements in the case of a complex-conjugate eigenpair of a real nonsymmetric matrix [9]. These small elements are then declared zero and the columns in Y_1' are **locked**, i.e., they are not altered anymore in the future computations. Orthogonality against them has to be enforced in the continuation of the eigenvalue computation though.

11.9 The rational Krylov space method

After having computed a number of eigenvalue–eigen/Schurvector pairs in the neighborhood of some shift σ_1 with the shift-invert Lanczos, Arnoldi, or Krylov–Schur algorithm it may be advisable to restart with a changed shift σ_2. This is in fact possible without discarding the available Krylov space [6]. In this section we consider the generalized eigenvalue problem $A\mathbf{x} = \lambda B\mathbf{x}$.

The rational Krylov space method starts out as a shift-invert Arnoldi iteration with shift σ_1 and starting vector $\mathbf{v}_1$. It computes an orthonormal basis V_j using the basic recurrence,

$$(A - \sigma_1 B)^{-1}BQ_j = Q_jH_j + \mathbf{r}_j\mathbf{e}^T = Q_{j+1}\bar{H}_j. \tag{11.22}$$

or, using the Schur decomposition of H_j, cf. (11.20),

$$(A - \sigma_1 B)^{-1}BY_j = Y_jT_j + \mathbf{r}_j\mathbf{s}^* = Y_{j+1}\begin{bmatrix} T_j \\ \mathbf{s}^* \end{bmatrix}, \qquad Y_{j+1} = [Y_j, \mathbf{r}_j] \tag{11.23}$$

We want to derive a Krylov–Schur relation for a new shift $\sigma_2 \neq \sigma_1$ from (11.23) for the same space $\mathcal{R}(Y_{j+1})$ without accessing the matrices A or B. The tricky thing is to avoid discard all the information gathered in the basis Y_{j+1} that was computed with the old shift σ_1. This is indeed possible if we replace the basis Y_{j+1} with a new basis W_{j+1}, which spans the same subspace as Y_{j+1} but can be interpreted as the orthonormal basis of a Krylov–Schur relation with the new shift σ_2.

We rewrite the relation (11.23) as

$$BY_j = BY_{j+1}\begin{bmatrix} I_j \\ \mathbf{0}^* \end{bmatrix} = (A - \sigma_1 B)Y_{j+1}\begin{bmatrix} T_j \\ \mathbf{s}^* \end{bmatrix}.$$

Introducing the shift σ_2 this becomes

$$BY_{j+1}\left\{\begin{bmatrix} I_j \\ \mathbf{0}^* \end{bmatrix} + (\sigma_1 - \sigma_2)\begin{bmatrix} T_j \\ \mathbf{s}^* \end{bmatrix}\right\} = (A - \sigma_2 B)Y_{j+1}\begin{bmatrix} T_j \\ \mathbf{s}^* \end{bmatrix}. \tag{11.24}$$

To construct a Krylov–Schur relation we must get rid of the last non-zero row of the matrix in braces in (11.24). To that end we use the QR factorization

$$\begin{bmatrix} I_j \\ \mathbf{0}^T \end{bmatrix} + (\sigma_1 - \sigma_2)\begin{bmatrix} T_j \\ \mathbf{s}^* \end{bmatrix} = Q_{j+1}\begin{bmatrix} R_j \\ \mathbf{0}^T \end{bmatrix}.$$

Using it we obtain

$$BY_{j+1}Q_{j+1}\begin{bmatrix} R_j \\ \mathbf{0}^T \end{bmatrix} \equiv BW_{j+1}\begin{bmatrix} R_j \\ \mathbf{0}^T \end{bmatrix} = BW_jR_j = (A - \sigma_2 B)W_{j+1}Q_{j+1}^*\begin{bmatrix} T_j \\ \mathbf{s}^* \end{bmatrix}$$

Multiplying with $(A - \sigma_2 B)^{-1}$ from the left we obtain

$$(A - \sigma_2 B)^{-1}BW_j = W_{j+1}Q_{j+1}^*\begin{bmatrix} T_jR_j^{-1} \\ \mathbf{s}^* \end{bmatrix} = W_{j+1}\begin{bmatrix} M_j \\ \mathbf{t}^* \end{bmatrix} \tag{11.25}$$

or

$$(A - \sigma_2 B)^{-1}BW_j = W_jM_j + \mathbf{w}_{j+1}\mathbf{t}^*. \tag{11.26}$$

This equation can easily been transformed into an Arnoldi or Krylov–Schur relation.

All these transformations can be executed without performing any operations on the large sparse matrices A and B.

In a practical implementation, the mentioned procedure is combined with locking, purging, and implicit restart. First run shifted and inverted Arnoldi with the first shift σ_1. When an appropriate number of eigenvalues around σ_1 have converged, lock these converged eigenvalues and purge those that are altogether outside the interesting region, leaving an Arnoldi (11.22) or Krylov–Schur recursion (11.22) for the remaining vectors. Then introduce the new shift σ_2 and perform the steps above to get a new basis W_{j+1} that replaces V_{j+1}. Start at the new shift by operating on the last vector of this new basis

$$\mathbf{r} := (A - \sigma_2 B)^{-1}B\mathbf{w}_{j+1}$$

and get the next basis vector w_{j+2} in the Arnoldi recurrence with the new shift σ_2. Continue until we get convergence for a set of eigenvalues around σ_2, and repeat the same procedure with new shifts until either all interesting eigenvalues have converged or all the shifts in the prescribed frequency range have been used.

Bibliography

[1] Å. Björck, *Numerics of Gram–Schmidt orthogonalization*, Linear Algebra Appl., 197/198 (1994), pp. 297–316.

[2] D. Calvetti, L. Reichel, and D. C. Sorensen, *An implicitely restarted Lanczos method for large symmetric eigenvalue problems*, Electron. Trans. Numer. Anal., 2 (1994), pp. 1–21.

[3] M. Genseberger and G. L. G. Sleijpen, *Alternative correction equations in the Jacobi–Davidson method*, Numer. Linear Algebra Appl., 6 (1999), pp. 235–253.

[4] R. B. Lehoucq, D. C. Sorensen, and C. Yang, *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems by Implicitely Restarted Arnoldi Methods*, SIAM, Philadelphia, PA, 1998. (The software and this manual are available at URL `http://www.caam.rice.edu/software/ARPACK/`).

[5] *The Matrix Market.* A repository of test data for use in comparative studies of algorithms for numerical linear algebra. Available at URL `http://math.nist.gov/MatrixMarket/`.

[6] A. Ruhe, *Rational Krylov subspace method*, in Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., SIAM, Philadelphia, PA, 2000, pp. 246–249.

[7] D. C. Sorensen, *Implicit application of polynomial filters in a k-step Arnoldi method*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 357–385.

[8] ——, *Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations*, in Parallel Numerical Algorithms, D. E. Keyes, A. Sameh, and V. Venkatakrishnan, eds., Kluwer, Dordrecht, 1997, pp. 119–165. (ICASE/LaRC Interdisciplinary Series in Science and Engineering, 4).

[9] G. W. Stewart, *A Krylov–Schur algorithm for large eigenproblems*, SIAM J. Matrix Anal. Appl., 23 (2001), pp. 601–614.

[10] ——, *Matrix Algorithms II: Eigensystems*, SIAM, Philadelphia, PA, 2001.

[11] K. Wu and H. D. Simon, *Thick-restart Lanczos method for large symmetric eigenvalue problems*, SIAM J. Matrix Anal. Appl., 22 (2000), pp. 602–616.

Chapter 12

The Jacobi-Davidson Method

The Lanczos and Arnoldi methods are very effective to compute extremal eigenvalues provided these are well separated from the rest of the spectrum. Lanczos and Arnoldi methods combined with a shift-and-invert spectral transformation are also efficient to compute eigenvalues in the vicinity of the shift σ. In this case it is necessary to solve a system of equation

$$(A - \sigma I)\mathbf{x} = \mathbf{y}, \qquad \text{or} \qquad (A - \sigma M)\mathbf{x} = \mathbf{y},$$

respectively, in each iteration step. These systems have to be solved very accurately since otherwise the Lanczos or Arnoldi relation does not hold anymore. In most cases the matrix $A - \sigma I$ (or $A - \sigma M$) is LU or Cholesky factored. The Jacobi–Davidson (JD) algorithm is particularly attractive if this factorization is not feasible [11].

12.1 The Davidson algorithm

Let $\mathbf{v}_1, \ldots, \mathbf{v}_m$ be a set of orthonormal vectors, spanning the *search space* $\mathcal{R}(V_m)$ with $V_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$. In the Galerkin approach we are looking for vectors $\mathbf{s} \in \mathbb{F}^m$ such that the *Galerkin condition* holds,

$$AV_m\mathbf{s} - \vartheta V_m\mathbf{s} \perp \mathbf{v}_1, \ldots, \mathbf{v}_m. \tag{12.1}$$

This immediately leads to the (small) eigenvalue problem

$$V_m^* A V_m \mathbf{s} = \vartheta V_m^* V_m \mathbf{s} \tag{12.2}$$

with solutions $(\vartheta_j^{(m)}, \mathbf{s}_j^{(m)})$, $j = 1, \ldots, m$. $\vartheta_j^{(m)}$ is called a Ritz value and $V_m\mathbf{s}_j^{(m)}$ is called a Ritz vector. In the sequel we omit the superscript m for readability. The dimension of the search space should become evident from the context.

Let us consider, say, the Ritz value ϑ_j, its Ritz vector $\mathbf{u}_j = V_m\mathbf{s}_j$ and their residual $\mathbf{r}_j = A\mathbf{u}_j - \vartheta_j\mathbf{u}_j$. Often we are looking for the largest or smallest eigenvalue of A in which case $j = 1$ or $j = m$, respectively. The question immediately arises how we can improve $(\vartheta_j, \mathbf{u}_j)$ if $\|\mathbf{r}_j\|$ is still too large. It is straightforward to try to find a better approximate eigenpair by *expanding* the search space. Davidson, in his original paper [2], suggested to compute a vector $\mathbf{t}$ from

$$(D_A - \vartheta_j I)\mathbf{t} = \mathbf{r}_j, \tag{12.3}$$

where D_A is the *diagonal* of the matrix A. The vector $\mathbf{t}$ is then made orthogonal to the basis vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m$. The resulting vector, after normalization, is chosen as $\mathbf{v}_{m+1}$ by which $\mathcal{R}(V_m)$ is expanded, i.e., $V_{m+1} = [\mathbf{v}_1, \ldots, \mathbf{v}_m, \mathbf{v}_{m+1}]$.

This method is successful in finding dominant eigenvalues of (strongly) diagonally dominant matrices. The matrix $D_A - \vartheta_j I$ has therefore often been viewed as a preconditioner for the matrix $A - \vartheta_j I$. A number of investigations were made with more sophisticated preconditioners $M - \vartheta_j I$, see e.g. [7, 8]. They lead to the conclusion that $M - \vartheta_j I$ should not be too close to $A - \vartheta_j I$ which contradicts the notion of a preconditioner as being an easily invertible (factorizable) approximation of $A - \vartheta_j I$.

12.2 The Jacobi orthogonal component correction

In his seminal paper, Jacobi [6] not only presented the solution of symmetric eigenvalue problems by successive application of (later to be called) Jacobi rotations, but also presented an approach to improve an approximate eigenpair with an iterative procedure. Here, we give Jacobi's approach in a generalized form presented by Sleijpen and van der Vorst [11]. Let $\mathbf{u}_j$ be an approximation to the eigenvector $\mathbf{x}$ of A corresponding to the eigenvalue λ. Jacobi proposed to *correct* $\mathbf{u}_j$ by a vector $\mathbf{t}$, $\mathbf{u}_j \perp \mathbf{t}$, such that

$$A(\mathbf{u}_j + \mathbf{t}) = \lambda(\mathbf{u}_j + \mathbf{t}), \qquad \mathbf{u}_j \perp \mathbf{t}. \tag{12.4}$$

This is called the **Jacobi orthogonal component correction (JOCC)** by Sleijpen & van der Vorst [11]. As $\mathbf{t} \perp \mathbf{u}_j$ we may split equation (12.4) in the part parallel to $\mathbf{u}_j$ and in the part orthogonal to $\mathbf{u}_j$. If $\|\mathbf{u}_j\| = 1$ then the part parallel to $\mathbf{u}_j$ is

$$\mathbf{u}_j {\mathbf{u}_j}^* A(\mathbf{u}_j + \mathbf{t}) = \lambda \mathbf{u}_j {\mathbf{u}_j}^* (\mathbf{u}_j + \mathbf{t}) \tag{12.5}$$

which simplifies to the scalar equation

$$\vartheta_j + {\mathbf{u}_j}^* A\mathbf{t} = \lambda. \tag{12.6}$$

Here ϑ_j is the Rayleigh quotient of $\mathbf{u}_j$, $\vartheta_j = \rho(\mathbf{u}_j)$. The part orthogonal to $\mathbf{u}_j$ is

$$(I - \mathbf{u}_j {\mathbf{u}_j}^*) A(\mathbf{u}_j + \mathbf{t}) = \lambda (I - \mathbf{u}_j {\mathbf{u}_j}^*)(\mathbf{u}_j + \mathbf{t}) \tag{12.7}$$

which is equivalent to

$$\begin{aligned}(I - \mathbf{u}_j {\mathbf{u}_j}^*)(A - \lambda I)\mathbf{t} &= (I - \mathbf{u}_j {\mathbf{u}_j}^*)(-A\mathbf{u}_j + \lambda \mathbf{u}_j) \\ &= -(I - \mathbf{u}_j {\mathbf{u}_j}^*)A\mathbf{u}_j = -(A - \vartheta_j I)\mathbf{u}_j =: -\mathbf{r}_j.\end{aligned}$$

As $(I - \mathbf{u}_j {\mathbf{u}_j}^*)\mathbf{t} = \mathbf{t}$ we can rewrite this equation in symmetrized form as

$$(I - \mathbf{u}_j {\mathbf{u}_j}^*)(A - \lambda I)(I - \mathbf{u}_j {\mathbf{u}_j}^*)\mathbf{t} = -\mathbf{r}_j. \tag{12.8}$$

If A is symmetric then the matrix in (12.8) is symmetric as well.

Unfortunately, we do not know λ! Therefore, we replace λ by ϑ_j to get the **Jacobi–Davidson correction equation**

$$\boxed{(I - \mathbf{u}_j \mathbf{u}_j^*)(A - \vartheta_j I)(I - \mathbf{u}_j \mathbf{u}_j^*)\mathbf{t} = -\mathbf{r}_j = -(A - \vartheta_j I)\mathbf{u}_j, \qquad \mathbf{t} \perp \mathbf{u}_j.} \tag{12.9}$$

As $\mathbf{r}_j \perp \mathbf{u}_j$ (in fact $\mathbf{r}_j \perp \mathcal{V}_m$) this equation is consistent if $A - \vartheta_j I$ is nonsingular.

The correction equation (12.9) is, in general, solved iteratively by the GMRES or MINRES algorithm [1]. Often, only little accuracy in the solution is required.

Once $\mathbf{t}$ is (approximately) known we set

$$\mathbf{u}_{j+1} = \mathbf{u}_j + \mathbf{t}. \tag{12.10}$$

From (12.6) we may then obtain

$$\vartheta_{j+1} = \vartheta_j + \mathbf{u}_j{}^* A\mathbf{t}. \tag{12.11}$$

If A is symmetric ϑ_{j+1} may be set equal to the Rayleigh quotient $\rho(\mathbf{u_{j+1}})$.

Let us analyze (12.9) more closely. Let us first investigate the role of the orthogonality condition $\mathbf{t} \perp \mathbf{u}_j$. If this condition is omitted then the equation to be solved is

$$(I - \mathbf{u}_j\mathbf{u}_j^*)(A - \vartheta_j I)\mathbf{t} = -\mathbf{r}_j = -(A - \vartheta_j I)\mathbf{u}_j. \tag{12.12}$$

This equation has the solution $\mathbf{t} = -\mathbf{u}_j$. Therefore, without the condition $\mathbf{t} \perp \mathbf{u}_j$ there is no progress in solving the eigenvalue problem $A\mathbf{x} = \lambda\mathbf{x}$.

One can argue that this is the approach suggested by Davidson [2]. Davidson approximated A on the left side of (12.12) by an approximation of it, typically the diagonal, say D_A, of A. As his matrices were diagonally dominant, he solved a reasonably good approximation of (12.12). If D_A in (12.3) is considered a preconditioner of A then any matrix closer to A should lead to better performance of the algorithm. In extremis, A should be a possible choice for the matrix on the left. But we have just seen that this leads to a situation without progress. In fact the progess in the iteration deteriorates the better the 'preconditioner' approximates the system matrix. In consequence, D_A in (12.3) must *not* be considered a preconditioner.

Let us now investigate what happens if the correction equation is solved exactly. To that end we write it as

$$(I - \mathbf{u}_j\mathbf{u}_j^*)(A - \vartheta_j I)\mathbf{t} = -\mathbf{r}_j, \qquad \mathbf{t} \perp \mathbf{u}_j,$$

which immediately leads to

$$(A - \vartheta_j I)\mathbf{t} - \mathbf{u}_j \underbrace{\mathbf{u}_j^*(A - \vartheta_j I)\mathbf{t}}_{\alpha \in \mathbb{F}} = -\mathbf{r}_j,$$

or,

$$(A - \vartheta_j I)\mathbf{t} = \alpha\mathbf{u}_j - \mathbf{r}_j.$$

Assuming that ϑ_j is not an eigenvalue of A we get

$$\mathbf{t} = \alpha(A - \vartheta_j I)^{-1}\mathbf{u}_j - (A - \vartheta_j I)^{-1}\mathbf{r}_j.$$

The constraint $\mathbf{u}_j^*\mathbf{t} = 0$ allows us to determine the free variable α,

$$0 = \alpha\mathbf{u}_j^*(A - \vartheta_j I)^{-1}\mathbf{u}_j - \mathbf{u}_j^*(A - \vartheta_j I)^{-1}\mathbf{r}_j,$$

whence

$$\alpha = \frac{\mathbf{u}_j^*(A - \vartheta_j I)^{-1}\mathbf{r}_j}{\mathbf{u}_j^*(A - \vartheta_j I)^{-1}\mathbf{u}_j}.$$

By (12.10), the next approximate is then

$$\mathbf{u}_{j+1} = \mathbf{u}_j + \mathbf{t} = \mathbf{u}_j + \alpha(A - \vartheta_j I)^{-1}\mathbf{u}_j - \underbrace{(A - \vartheta_j I)^{-1}\mathbf{r}_j}_{\mathbf{u}_j} = \alpha(A - \vartheta_j I)^{-1}\mathbf{u}_j \tag{12.13}$$

which is a step of Rayleigh quotient iteration! This implies a fast (quadratic in general, cubic in the Hermitian case) convergence rate of this algorithm.

In general the correction equation

$$\tilde{A}\mathbf{t} = (I - \mathbf{u}_j\mathbf{u}_j^*)(A - \vartheta_j I)(I - \mathbf{u}_j\mathbf{u}_j^*)\mathbf{t} = -\mathbf{r}_j, \qquad \mathbf{t} \perp \mathbf{u}_j, \tag{12.14}$$

is solved iteratively with a Krylov space solver like GMRES or MINRES [1]. To get a decent performance a preconditioner is needed. Sleijpen and van der Vorst suggest preconditioners of the form

$$\tilde{K} = (I - \mathbf{u}_j\mathbf{u}_j^*)K(I - \mathbf{u}_j\mathbf{u}_j^*), \qquad K \approx A - \vartheta_j I. \tag{12.15}$$

We assume that K is (easily) invertible, i.e., that it is computationaly much cheaper to solve a system of equation with K than with A. With this assumption the system of equation

$$\tilde{K}\mathbf{z} = \mathbf{v}, \qquad \mathbf{z} \perp \mathbf{u}_j,$$

can be solved provided that the right-hand side $\mathbf{v}$ is in the range of $\tilde{K}$, i.e., provided that $\mathbf{v} \perp \mathbf{u}_j$. We formally denote the solution by $\mathbf{z} = \tilde{K}^+\mathbf{v}$. So, instead of (12.14) we solve the equation

$$\tilde{K}^+\tilde{A}\mathbf{t} = -\tilde{K}^+\mathbf{r}_j, \qquad \mathbf{t} \perp \mathbf{u}_j. \tag{12.16}$$

Let $\mathbf{t}_0 = \mathbf{0}$ be the initial approximation to the solution of (12.16). (Notice that $\mathbf{t}_0$ trivially satisfies the orthogonality constraint.) Because of the projectors $I - \mathbf{u}_j\mathbf{u}_j^*$ in the definitions of $\tilde{A}$ and $\tilde{K}$ all approximations are orthogonal to $\mathbf{u}_j$.

In each iteration step we have to compute

$$\mathbf{z} = \tilde{K}^+\tilde{A}\mathbf{v}, \qquad \mathbf{z} \perp \mathbf{u}_j$$

where $\mathbf{v} \perp \mathbf{u}_j$. To do this we proceed as follows. First we write

$$\tilde{A}\mathbf{v} = \underbrace{(I - \mathbf{u}_j\mathbf{u}_j^*)(A - \vartheta_j I)\mathbf{v}}_{\mathbf{y}} =: \mathbf{y}.$$

Then,

$$\tilde{K}\mathbf{z} = \mathbf{y}, \qquad \mathbf{z} \perp \mathbf{u}_j.$$

With (12.15) this becomes

$$(I - \mathbf{u}_j\mathbf{u}_j^*)K\mathbf{z} = K\mathbf{z} - \mathbf{u}_j\mathbf{u}_j^*K\mathbf{z} = \mathbf{y},$$

the solution of which is

$$\mathbf{z} = K^{-1}\mathbf{y} - \alpha K^{-1}\mathbf{u}_j,$$

where, formally, $\alpha = -\mathbf{u}_j^*K\mathbf{z}$. Similarly as earlier, we determine the scalar by means of the constraint $\mathbf{z}^*\mathbf{u}_j = 0$. Thus

$$\alpha = \frac{\mathbf{u}_j^*K^{-1}\mathbf{y}}{\mathbf{u}_j^*K^{-1}\mathbf{u}_j}.$$

Remark 12.1. Since $\mathbf{u}_j$ is fixed during the solution of the secular equation, the vector $K^{-1}\mathbf{u}_j$ has to be computed just once. Thus, if the iterative solver needs k steps until convergence, $k + 1$ systems of equations have to be solved with the matrix K. □

Algorithm 12.1 The Jacobi–Davidson algorithm to compute the eigenvalue of A closest to a target value τ

```
 1: Let $A, B \in \mathbb{F}^{n\times n}$. This algorithm computes the eigenvalue of $A$ that is closest to $\tau$.
    Let $\mathbf{t}$ be an initial vector. Set $V_0 = []$, $V_0^A = []$, $m = 0$.
 2: loop
 3:   for $i = 1, \ldots, m-1$ do
 4:     $\mathbf{t} := \mathbf{t} - (\mathbf{v}_i^*\mathbf{t})\mathbf{v}_i$;                    /* $\mathbf{t} = (I - V_{m-1}V_{m-1}^*)\mathbf{t}$ */
 5:   end for
 6:   $\mathbf{v}_m := \mathbf{t}/\|\mathbf{t}\|$;   $\mathbf{v}_m^A := A\mathbf{v}_m$;   $V_m := [V_{m-1}, \mathbf{v}_m]$;   $V_m^A := [V_{m-1}^A, \mathbf{v}_m^A]$;
 7:   for $i = 1, \ldots, m$ do
 8:     $M_{i,m} := \mathbf{v}_i^*\mathbf{v}_m^A$;   $M_{m,i} := \mathbf{v}_m^*\mathbf{v}_i^A$;                    /* $M = V_m^* A V_m$ */
 9:   end for
10:   $M_{m,m} := \mathbf{v}_m^*\mathbf{v}_m^A$;
11:   Compute the eigenvalue $\vartheta$ of $M$ closest to $\tau$ and the      /* Rayleigh Ritz step */
      corresponding eigenvector $\mathbf{s}$: $M\mathbf{s} = \vartheta\mathbf{s}$;   $\|\mathbf{s}\| = 1$;
12:   $\mathbf{u} := V_m\mathbf{s}$;   $\mathbf{u}^A := V_m^A\mathbf{s}$;   $\mathbf{r} := \mathbf{u}^A - \vartheta\mathbf{u}$;
13:   if $\|\mathbf{r}\| <$ tol then
14:     return $(\tilde{\lambda} = \vartheta,\ \tilde{\mathbf{x}} = \mathbf{u})$
15:   end if
16:   (Approximatively) solve the correction equation for $\mathbf{t}$,
          $(I - \mathbf{u}\mathbf{u}^*)(A - \vartheta_j I)(I - \mathbf{u}\mathbf{u}^*)\mathbf{t} = -\mathbf{r}$,     $\mathbf{t} \perp \mathbf{u}$;
17: end loop
```

12.2.1 Restarts

Evidently, in Algorithm 12.1, the dimension m of the search space can get large. To limit memory consumption, we limit m such that $m \le m_{\max}$. As soon as $m = m_{\max}$ we restart: $V_m = V_{m_{\max}}$ is replaced by the q Ritz vectors corresponding to the Ritz values closest to the target τ. Notice that the Schur decomposition of $M = M_{m,m} = V_m^* A V_m$ is computed already in step 11 of the Algorithm. Let $M = S^*TS$ be this Schur decomposition with $|t_{11} - \tau| \le |t_{22} - \tau| \le \cdots$. Then we set $V_q = V_m \cdot S_{:,1:q}$, $V_q^A = V_m^A \cdot S_{:,1:q}$, $M = T \cdot S_{1:q,1:q}$. Notice that the restart is easy because the Jacobi–Davidson algorithm is not a Krylov space method.

12.2.2 The computation of several eigenvalues

Let $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_k$ be already computed eigenvectors or Schur vectors with $\tilde{\mathbf{x}}_i^*\tilde{\mathbf{x}}_j = \delta_{ij}$, $1 \le i, j \le k$. Then

$$AQ_k = Q_kT_k, \qquad Q_k = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_k]. \tag{12.17}$$

is a **partial Schur decomposition** of A [13]. We want to *extend* the partial Schur decomposition by one vector employing the Jacobi–Davidson algorithm. Since Schur vectors are mutually orthogonal we can apply the Jacobi–Davidson algorithm in the orthogonal complement of $\mathcal{R}(Q_k)$, i.e., we apply the Jacobi–Davidson algorithm to the matrix

$$(I - Q_kQ_k^*)A(I - Q_kQ_k^*), \qquad Q_k = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_k]. \tag{12.18}$$

The correction equation gets the form

$$(I - \mathbf{u}_j\mathbf{u}_j^*)(I - Q_kQ_k^*)(A - \vartheta_j I)(I - Q_kQ_k^*)(I - \mathbf{u}_j\mathbf{u}_j^*)\mathbf{t} = -\mathbf{r}_j, \qquad \mathbf{t} \perp \mathbf{u}_j, \mathbf{t} \perp Q_k. \tag{12.19}$$

As $\mathbf{u}_j \perp Q_k$ we have

$$(I - \mathbf{u}_j\mathbf{u}_j^*)(I - Q_kQ_k^*) = I - \tilde{Q}_k\tilde{Q}_k^*, \qquad \tilde{Q}_k = [\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_k, \mathbf{u}_j].$$

Thus, we can write (12.19) in the form

$$(I - \tilde{Q}_k\tilde{Q}_k^*)(A - \vartheta_j I)(I - \tilde{Q}_k\tilde{Q}_k^*)\mathbf{t} = -\mathbf{r}_j, \qquad \tilde{Q}_k^*\mathbf{t} = \mathbf{0}. \tag{12.20}$$

The preconditioner becomes

$$\tilde{K} = (I - \tilde{Q}_k\tilde{Q}_k^*)K(I - \tilde{Q}_k\tilde{Q}_k^*), \qquad K \approx A - \vartheta_j I. \tag{12.21}$$

Similarly as earlier, for solving

$$\tilde{K}\mathbf{z} = \tilde{A}\mathbf{v}, \qquad \tilde{Q}_k^*\mathbf{z} = \tilde{Q}_k^*\mathbf{v} = \mathbf{0},$$

we execute the following steps. Since

$$\tilde{A}\mathbf{v} = (I - \tilde{Q}_k\tilde{Q}_k^*)\underbrace{(A - \vartheta_j I)\mathbf{v}}_{\mathbf{y}} =: (I - \tilde{Q}_k\tilde{Q}_k^*)\mathbf{y} =: \mathbf{y} - \tilde{Q}_k\underbrace{\tilde{Q}_k^*\mathbf{y}}_{\mathbf{a}}.$$

we have to solve

$$\tilde{K}\mathbf{z} = (I - \tilde{Q}_k\tilde{Q}_k^*)K\mathbf{z} = (I - \tilde{Q}_k\tilde{Q}_k^*)\mathbf{y}, \qquad \mathbf{z} \perp \tilde{Q}_k.$$

Thus,

$$\mathbf{z} = K^{-1}\mathbf{y} - K^{-1}\tilde{Q}_k\mathbf{a}.$$

Similarly as earlier, we determine $\mathbf{a}$ by means of the constraint $\tilde{Q}_k^*\mathbf{z} = \mathbf{0}$,

$$\mathbf{a} = (\tilde{Q}_k^*K^{-1}\tilde{Q}_k)^{-1}\tilde{Q}_k^*K^{-1}\mathbf{y}.$$

If the iteration has converged to the vector $\tilde{\mathbf{x}}_{k+1}$ we can extend the partial Schur decomposition (12.17). Setting

$$Q_{k+1} := [Q_k, \tilde{\mathbf{x}}_{k+1}],$$

we get

$$AQ_{k+1} = Q_{k+1}T_{k+1} \tag{12.22}$$

with

$$T_{k+1} = \begin{bmatrix} T_k & Q_k^*A\tilde{\mathbf{x}}_{k+1} \\ 0 & \tilde{\mathbf{x}}_{k+1}^*A\tilde{\mathbf{x}}_{k+1} \end{bmatrix}.$$

12.2.3 Spectral shifts

In the correction equation (12.9) and implicitely in the preconditioner (12.15) a spectral shift ϑ_j appears. Experiments show that it is not wise to always choose the Rayleigh quotient of the recent eigenvector approximation $\mathbf{u}$ as the shift. In particular, far away from convergence, i.e., in the first few iteration steps, the Rayleigh quotient may be far away from the (desired) eigenvalue, and in fact may direct the JD iteration to an unwanted solution. So, one proceeds similarly as in the plain Rayleigh quotient iteration, cf. Remark 7.6 on page 143. Initially, the shift is held fixed, usually equal to the *target value* τ. As soon as the norm of the residual is small enough, the Rayleigh quotient of the actual approximate is chosen as the spectral shift in the correction equation. For efficiency

Algorithm 12.2 The Jacobi–Davidson QR algorithm to compute p of the eigenvalues closest to a target value τ

```
$Q_0 := []$;   $k = 0$.                                   /* Initializations */
Choose $\mathbf{v}_1$ with $\|\mathbf{v}_1\| = 1$.
$\mathbf{w}_1 = A\mathbf{v}_1$;   $H_1 := \mathbf{v}_1^*\mathbf{w}_1$;   $V_1 := [\mathbf{v}_1]$;   $W_1 := [\mathbf{W}_1]$;
$\tilde{\mathbf{q}} = \mathbf{v}_1$;   $\tilde{\vartheta} = \mathbf{v}_1^*\mathbf{w}_1$;   $\mathbf{r} := \mathbf{w}_1 - \tilde{\vartheta}\tilde{\mathbf{q}}$.
$j := 1$;
while $k < p$ do
                                 /* Compute Schur vectors one after the other */
    Approximatively solve the correction equation for $\mathbf{t}$
        $$(I - \tilde{Q}_k\tilde{Q}_k^*)(A - \tilde{\vartheta}I)(I - \tilde{Q}_k\tilde{Q}_k^*)\mathbf{t} = -\mathbf{r}_j, \qquad \tilde{Q}_k^*\mathbf{t} = \mathbf{0}.$$
    where $\tilde{Q}_k = [Q_k, \tilde{\mathbf{q}}]$.
    $\mathbf{v}_j = (I - V_{j-1}V_{j-1}^*)\mathbf{t}/\|(I - V_{j-1}V_{j-1}^*)\mathbf{t}\|$;   $V_j := [V_{j-1}, \mathbf{v}_j]$.
    $\mathbf{w}_j = A\mathbf{v}_j$;   $H_j = \begin{bmatrix} H_{j-1} & V_{j-1}^*\mathbf{w}_j \\ \mathbf{v}_j^*W_{j-1} & \mathbf{v}_j^*\mathbf{w}_j \end{bmatrix}$;   $W_j = [W_{j-1}, \mathbf{w}_j]$.
    Compute the Schur decomposition of
        $H_j =: S_j R_j S_j$
    with the eigenvalues $r_{ii}^{(j)}$ sorted according to their distance to $\tau$.
    /* Test for convergence */
    repeat
        $\tilde{\vartheta} = \lambda_1^{(j)}$;   $\tilde{\mathbf{q}} = V_j\mathbf{s}_1$;   $\tilde{\mathbf{w}} = W_j\mathbf{s}_1$;   $\mathbf{r} = \tilde{\mathbf{w}} - \tilde{\vartheta}\tilde{\mathbf{q}}$
        found := $\|\mathbf{r}\| < \varepsilon$
        if found then
            $Q_{k+1} = [Q_k, \tilde{\mathbf{q}}]$;   $k := k + 1$;
        end if
    until not found
    /* Restart */
    if $j = j_{\max}$ then
        $V_{j_{\min}} := V_j[\mathbf{s}_1, \ldots, \mathbf{s}_{\min}]$;   $T_{j_{\min}} := T_j(1 : j_{\min}, 1 : j_{\min})$;
        $H_{j_{\min}} := T_{j_{\min}}$;   $S_{j_{\min}} := I_{j_{\min}}$;   $J := j_{\min}$
    end if
end while
```

reasons, the spectral shift in the preconditioner K is always fixed. In this way it has to be computed just once. Notice that $\tilde{K}$ is changing with each correction equation.

Remark 12.2. As long as the shift is held fixed Jacobi–Davidson is actually performing a shift-and-invert Arnoldi iteration. □

Algorithm 12.2 gives the framework for an algorithm to compute the partial Schur decomposition of a matrix A. Q_k stores the converged Schur vectors; V_j stores the 'active' search space. This algorithm *does not* take into account some of the just mentioned issues. In particular the shift is always taken to be the Rayleigh quotient of the most recent approximate $\tilde{\mathbf{q}}$.

12.3 The generalized Hermitian eigenvalue problem

We consider the problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \tag{12.23}$$

with A and M $n\times n$ Hermitian, and M additionally positive definite. Then the eigenvectors can be chosen mutually M-orthogonal,

$$\mathbf{x}_i^* M\mathbf{x}_j = \delta_{ij}, \qquad A\mathbf{x}_i = \lambda_i M\mathbf{x}_i, \quad 1 \le i, j \le n, \tag{12.24}$$

where δ_{ij} denotes the Kronecker delta function. Then it makes sense in the Jacobi–Davidson (as in other algorithms) to keep the iterates M-orthogonal.

Let $V_m = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$ be an M-orthogonal basis of the search space $\mathcal{V}_m$. Then the Galerkin condition

$$AV_m\mathbf{s} - \vartheta MV_m\mathbf{s} \perp \mathbf{v}_1, \ldots, \mathbf{v}_m, \tag{12.25}$$

leads to the eigenvalue problem

$$V_m^* AV_m\mathbf{s} = \vartheta V_m^* MV_m\mathbf{s} = \vartheta\mathbf{s}. \tag{12.26}$$

Let $(\tilde{\vartheta}, \tilde{\mathbf{u}} = V_m\tilde{\mathbf{s}})$ be a solution of (12.26). Then the correction $\mathbf{t}$ to $\tilde{\mathbf{u}}$ must be M-orthogonal,

$$\mathbf{t}^* M\tilde{\mathbf{u}} = 0 \quad \Longleftrightarrow \quad (I - \tilde{\mathbf{u}}\tilde{\mathbf{u}}^* M)\mathbf{t} = \mathbf{t}. \tag{12.27}$$

The correction equation in turn becomes

$$(I - M\tilde{\mathbf{u}}\tilde{\mathbf{u}}^*)(A - \tilde{\vartheta}M)(I - \tilde{\mathbf{u}}\tilde{\mathbf{u}}^* M)\mathbf{t} = -(I - \tilde{\mathbf{u}}\tilde{\mathbf{u}}^* M)\tilde{\mathbf{r}}, = -\tilde{\mathbf{r}}, \qquad \mathbf{t} \perp_M \tilde{\mathbf{u}}, \tag{12.28}$$

where $\tilde{\mathbf{r}} = A\tilde{\mathbf{u}} - \tilde{\vartheta}M\tilde{\mathbf{u}}$. Preconditioners for the secular equation are chosen of the form

$$\tilde{K} = (I - M\tilde{\mathbf{u}}\tilde{\mathbf{u}}^*)K(I - \tilde{\mathbf{u}}\tilde{\mathbf{u}}^* M), \tag{12.29}$$

where $K \approx A - \tau M$ and τ is the target value.

12.4 A numerical example

We give a demonstration on how a full-fledged Jacobi–Davidson algorithm works. The code is a MATLAB implementation of a program from the PhD thesis of Geus [4]. It solves the *generalized symmetric* eigenvalue problem as discussed in the previous section. The command `help jdsym` provides the output given on page 225.

As the numerical example we again consider the accustic behaviour in the interior of a car. We compute the five smallest eigenvalues and associated eigenvectors. The preconditioner is chosen to be the diagonal of A. An eigenpair $(\tilde{\lambda}, \tilde{\mathbf{q}}$ is declared converged if the residual norm $\|A\tilde{\mathbf{q}} - \tilde{\lambda}M\tilde{\mathbf{q}}\| < 10^{-8}\|\tilde{\mathbf{q}}\|$. Most of the components of `options` are explained in the help text. The residual norms for each iteration step are plotted in Fig. 12.1. As soon as an eigenpair has converged a new iteration starts. The residual norm then increases by several orders of magnitude.

```
[Q, lambda, it] = jdsym(n, A, B, K, kmax, tau, options)

jdsym is a MATLAB implementation of the JDQR algorithm for symmetric
matrices.

jdsym returns kmax eigenvalues with corresponding eigenvectors of
the matrix A near the target tau. K is a symmetric preconditioner
for A - tau * B.

The arguments A and B both contain either n-by-n symmetric matrices
or a string containing the name of an M-file which applies a
symmetric linear operator to the columns of a given
matrix. Matrix B must be positive definite.

To solve the specialized eigenvalue problem A * x = lambda * x pass
an empty matrix [] for parameter B. If no preconditioner is used
pass an empty matrix [] for parameter K.

 The options structure specifies certain parameters in the algorithm:

 options.tol      convergence tolerance                         1e-10
 options.jmax     maximal dimension of search subspace V        2*kmax
 options.jmin     dimension of search subspace V after restart  kmax
 options.maxit    maximum number of outer iterations            max(100,2*n/jmax)
 options.clvl     verbosity of output (0 means no output)       1
 options.eps_tr   tracing parameter as described in literature  1e-4
 options.toldecay convergence tolerance for inner iteration is  2
                  toldecay ^ (-solvestep)
 options.cgmaxit  maximum number of iterations in linear solver 100
 options.V0       initial search subspace                       rand(n,1)-.5
                  V0 will be orthonormalized by jdsym
 options.linsolv  solver used for corrections equation          1
                  1 -- CGS
                  2 -- SYMMLQ
                  3 -- CGS_OP
                  4 -- CGS mit SYMOP
                  5 -- MINRES
                  6 -- QMR
                  7 -- QMRS
 options.strategy strategy to avoid computation of zero
                  eigenvalues:
                  0 -- standard JD algorithm                    0
                  1 -- never choose Ritz values that are close
                       to zero as best current approximation.
                       Purge Ritz values that are close
                       to zero when restarting
                  2 -- dynamically adjust tau
                  3 -- see (1) and set tau to last converged
                       eigenvalue if it was bigger than the old
                       tau
                  4 -- set tau to last converged eigenvalue if
                       it was bigger than the old tau

The converged eigenvalues and eigenvectors are stored in Q and lambda. The
number of outer JD iterations performed is returned in it.
```

```
    >> K=diag(diag(A));
    >> options

    options =
         linsolv: 6
        strategy: 0

>> options.tol=1e-8

options =

        tol: 1.0000e-08
       jmax: 20
       jmin: 10
       clvl: 1
     optype: 1
    linsolv: 5

>> [Q, lambda, it] = jdsym(n, A, M, K, 5, -0.01, options);
JDSYM     Solving  A*x = lambda*M*x  with preconditioning

  N=            1095  ITMAX=1.095000e+02
  KMAX=  5  JMIN= 10  JMAX= 20  VODIM=  1
  TAU=   -1.0000e-02  JDTOL=   1.0000e-08  STRATEGY=        0
  LINSOLVER=  MINRES  OPTYPE=         SYM
  LINITMAX=      100  EPS_TR=   1.000e-04  TOLDECAY= 2.00e+00

  IT   K   J       RES     CGTHET     CGTOL CGIT     CGERR CGFLG    Ritz values 1-5
-----------------------------------------------------------------------------------
   0   0   1  4.26e+00
   1   0   2  9.33e-01  -1.00e-02  2.50e-01    1  9.74e-01     0
   2   0   3  7.13e-02  -1.00e-02  1.25e-01    4  6.95e-02     0
   3   0   4  4.14e-03  -1.00e-02  6.25e-02   10  4.04e-03     0
   4   0   5  2.01e-04  -1.00e-02  3.12e-02   33  1.22e-04     0
   5   0   6  4.79e-05  -1.00e-02  1.56e-02   71  3.07e-06     0
   6   0   7  3.66e-07   9.33e-08  7.81e-03   88  3.53e-07     0
   7   0   8  1.70e-09   6.39e-12  3.91e-03   74  1.34e-09     0
   7   1   7  5.94e-03
   8   1   8  4.98e-03  -1.00e-02  5.00e-01    4  2.67e-03     0
   9   1   9  2.53e-03  -1.00e-02  2.50e-01   11  1.19e-03     0
  10   1  10  3.38e-04  -1.00e-02  1.25e-01   18  3.06e-04     0
  11   1  11  4.76e-05  -1.00e-02  6.25e-02   27  2.05e-05     0
  12   1  12  1.45e-06   1.27e-02  3.12e-02   26  1.48e-06     0
  13   1  13  1.87e-08   1.27e-02  1.56e-02   38  2.22e-08     0
  14   1  14  9.87e-11   1.27e-02  7.81e-03   60  1.38e-10     0
  14   2  13  4.75e-03
  15   2  14  3.58e-03  -1.00e-02  5.00e-01    5  2.17e-03     0
  16   2  15  1.16e-03  -1.00e-02  2.50e-01    9  8.93e-04     0
  17   2  16  1.59e-04  -1.00e-02  1.25e-01   10  1.24e-04     0
  18   2  17  1.46e-05  -1.00e-02  6.25e-02   14  8.84e-06     0
  19   2  18  4.41e-07   4.44e-02  3.12e-02   21  4.29e-07     0
  20   2  19  7.01e-09   4.44e-02  1.56e-02   29  6.58e-09     0
  20   3  18  4.82e-03
  21   3  19  3.44e-03  -1.00e-02  5.00e-01    3  2.34e-03     0
  22   3  20  8.25e-04  -1.00e-02  2.50e-01    7  7.08e-04     0
  23   3  11  1.57e-04  -1.00e-02  1.25e-01   11  8.91e-05     0
  24   3  12  1.65e-05  -1.00e-02  6.25e-02   14  9.77e-06     0
  25   3  13  4.77e-07   5.66e-02  3.12e-02   31  4.68e-07     0
  26   3  14  6.51e-09   5.66e-02  1.56e-02   32  7.26e-09     0
  26   4  13  1.28e-02
  27   4  14  1.14e-02  -1.00e-02  5.00e-01    3  6.30e-03     0
  28   4  15  3.54e-03  -1.00e-02  2.50e-01    6  2.45e-03     0
  29   4  16  8.00e-04  -1.00e-02  1.25e-01   10  4.19e-04     0
  30   4  17  1.13e-04  -1.00e-02  6.25e-02   12  4.95e-05     0
  31   4  18  1.67e-05  -1.00e-02  3.12e-02   16  3.22e-06     0
  32   4  19  4.23e-07   1.17e-01  1.56e-02   21  2.49e-07     0
  33   4  20  3.20e-09   1.17e-01  7.81e-03   45  3.21e-09     0

JDSYM

IT_OUTER=33   IT_INNER_TOT=764   IT_INNER_AVG=   23.15

Converged eigensolutions in order of convergence:

  I              LAMBDA(I)      RES(I)
---------------------------------------
  1  9.102733263227557e-16  1.70111e-09
  2  1.269007628846320e-02  9.86670e-11
  3  4.438457596823515e-02  7.01153e-09
  4  5.663501055565738e-02  6.50940e-09
  5  1.166311652214006e-01  3.19504e-09
>>
```

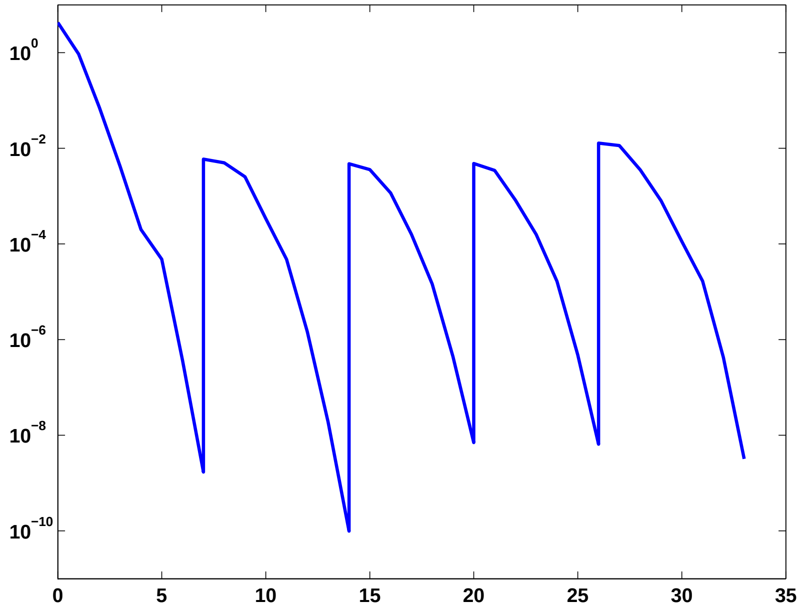


Figure 12.1: Jacobi–Davidson convergence history

12.5 The Jacobi–Davidson algorithm for interior eigenvalues

Interior eigenvalues are eigenvalues that do not lie at the ‘border’ of the convex hull of the spectrum, cf. Fig. 12.2

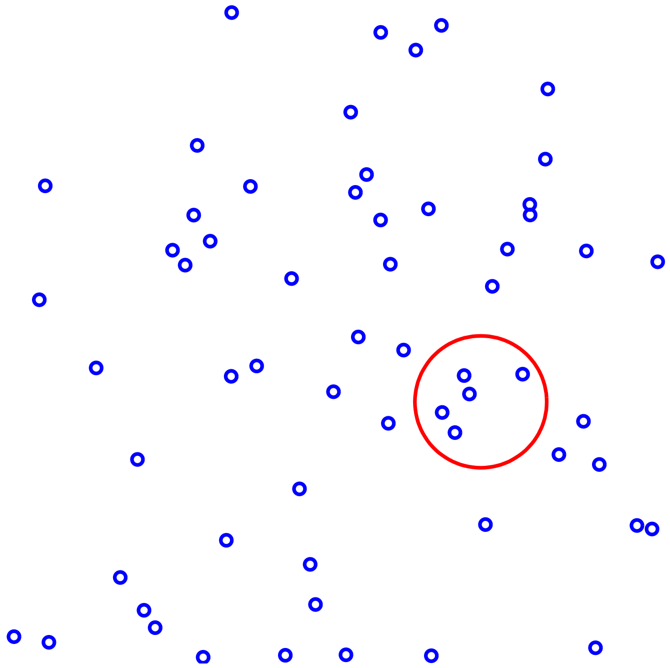


Figure 12.2: View of a spectrum $\sigma(A)$ in the complex plane. The eigenvalues in the red circle are to be computed

The success of the Jacobi–Davidson algorithm depends heavily on the quality of the actual Ritz pair $(\tilde{\vartheta}_j, \tilde{\mathbf{q}})$. However, the Rayleigh–Ritz procedure can lead to problem if it is applied to *interior* eigenvalues. The following simple numerical example shall demonstrate the problem. Let

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \qquad U = \begin{bmatrix} 1 & 0 \\ 0 & \sqrt{0.5} \\ 0 & \sqrt{0.5} \end{bmatrix}.$$

Then,

$$U^*AU = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \qquad U^*U = I_2.$$

So, any linear combination of the columns of U is a Ritz vector corresponding to the Ritz value 0, e.g.,

$$U \begin{pmatrix} \sqrt{0.5} \\ \sqrt{0.5} \end{pmatrix} = \begin{pmatrix} \sqrt{0.5} \\ 0.5 \\ 0.5 \end{pmatrix}.$$

Thus, although the basis contains the correct eigenvalue associated with the eigenvalue 0, the Rayleigh–Ritz procedure fails to find it and, instead, returns a very bad eigenvector approximation.

This example may look contrived. So, we conduct a MATLAB experiment with the same A but with a randomly perturbed U.

```
>> rand('state',0)
>> U1=U+1e-4*rand(size(U)); [U1,dummy]=qr(U1,0); U1=-U1
U1 =
    1.0000   -0.0001
    0.0000    0.7071
    0.0001    0.7071

>> B=U1'*A*U1

B =
   1.0e-04 *
   -0.0000   -0.2656
   -0.2656    0.1828

>> [X,L]=eig(B)

X =
   -0.8140   -0.5808
   -0.5808    0.8140
L =
   1.0e-04 *
   -0.1896         0
         0    0.3723

>> x=U1*-X(:,1)

x =
    0.8140
    0.4107
    0.4107

>> theta=L(1,1)

theta =
  -1.8955e-05

>> norm(A*x-x*theta)

ans =
    0.5808
```

We note that ϑ is a reasonable approximation for the eigenvalue 0. However, as the norm of the residual indicates, the Ritz vector is a bad approximation of the eigenvector.

12.6 Harmonic Ritz values and vectors

In the shift-and-invert Arnoldi algorithm the basic operator is $A - \sigma I$ where σ is some shift. The Arnoldi algorithm finds the largest eigenvalues of $A - \sigma I$, i.e., the eigenvalues of A closest to the shift. One of the reasons for inventing the Jacobi-Davidson algorithm is infeasibility of the factorization of $A - \sigma I$. Therefore, a shift-and-invert approach is not possible.

A clever way out of this dilemma works as follows: We apply the Ritz–Galerkin procedure with the matrix $(A - \sigma I)^{-1}$ and some subspace $\mathcal{R}(V) \subset \mathbb{F}^n$. This leads to the eigenvalues problem

$$V^*(A-\sigma I)^{-1}V\mathbf{s} = \mu V^*V\mathbf{s}. \tag{12.30}$$

The largest Ritz values μ_j approximate the largest eigenvalues of $(A-\sigma I)^{-1}$, i.e.,

$$\mu_j \approx \frac{1}{\lambda_j - \sigma} \Longleftrightarrow \lambda_j \approx \sigma + \frac{1}{\mu_j},$$

where λ_j is an eigenvalue of A close to the shift σ.

The trick is in the choice of V. Let us set $V := (A-\sigma I)U$. Then (12.30) becomes

$$U^*(A-\sigma I)^*U\mathbf{s} = \mu U^*(A-\sigma I)^*(A-\sigma I)U\mathbf{s}, \tag{12.31}$$

or, with $\tau = 1/\mu$,

$$U^*(A-\sigma I)^*(A-\sigma I)U\mathbf{s} = \tau U^*(A-\sigma I)^*U\mathbf{s}. \tag{12.32}$$

With $V = (A-\sigma I)U$ this becomes

$$V^*V\mathbf{s} = \tau V^*U\mathbf{s}. \tag{12.33}$$

If A is nonsymmetric, we compute an orthonormal basis $\tilde{V}$ of $V = (A-\sigma I)U$. Then we can write (12.32) in the nonsymmetric form

$$\tilde{V}^*(A-\sigma I)U\mathbf{s} = \tau \tilde{V}^*U\mathbf{s}. \tag{12.34}$$

We make the following

Definition 12.1 Let $(\tau, \mathbf{s})$ be an eigenpair of (12.32)–(12.34). Then the pair $(\sigma+\tau, U\mathbf{s})$ is called a **harmonic Ritz pair** of A with shift σ.

In practice, we are interested only in the harmonic Ritz pair corresponding to the smallest harmonic Ritz values. In the correction equation of the Jacobi–Davidson algorithm the harmonic Ritz vector is used as the latest eigenvector approximation and the harmonic Ritz values as the shift. In the symmetric case the harmonic Ritz value is replaced by the Rayleigh quotient of the harmonic Ritz vector $\mathbf{x}$, since

$$\|A\mathbf{x} - \rho(\mathbf{x})\mathbf{x}\| \le \|A\mathbf{x} - \mu\mathbf{x}\|, \quad \text{for all } \mu.$$

We continue the previous numerical example regarding the computation of the eigenvalue 0 of $A = \text{diag}(0, 1, -1)$

```
>> V=(A-theta*eye(3))*U1;
>> [v,l]=eig(V'*V, V'*U1)
v =
  -1.000000000000000  -1.000000000000000
   0.000059248824925  -0.713473633096137
l =
   1.0e+17 *
   0.000000000000000                   0
                   0  -1.970695224946170

>> theta + l(1,1)    %  Harmonic Ritz value
ans =
   3.722769433847084e-05
```

```
>> x = U1*v(:,1)   % Harmonic Ritz vector
x =
   1.000000001402380
  -0.000018783973233
   0.000018783630008
>> x'*A*x
ans =
   1.289413628670287e-14
```

The above considerations affect the Jacobi–Davidson algorithm in the extraction phase. Steps 11 and 14 in Algorithm 12.2 become

```
11: Compute the smallest eigenpair $(\tilde{\tau}, \tilde{\mathbf{s}})$ of
        $(W_j^* - \bar{\sigma} V_j^*)(W_j - \sigma V_j)\mathbf{s} = \tau (W_j^* - \bar{\sigma} V_j^*) V_j \mathbf{s}.$
14: Set $\tilde{\mathbf{q}} = V_j \tilde{\mathbf{s}}$, $\tilde{\mathbf{w}} = W_j \tilde{\mathbf{s}}$.    $\tilde{\vartheta} = \sigma + \tau$ or $\tilde{\vartheta} = \tilde{\mathbf{q}}^* A \tilde{\mathbf{q}} / \tilde{\mathbf{q}}^* \tilde{\mathbf{q}}$.
```

To solve the eigenvalue problem (12.34) the QZ algorithm has to be employed, see section 12.8. In the symmetric case (12.33) the symmetric QR algorithm will suffice in general since the matrix on the left is positive definite.

12.7 Refined Ritz vectors

An alternative to harmonic Ritz vectors are refined Ritz vectors [13]. Again we start from the observation that the Ritz values were of good quality. What we need are improved Ritz vectors. Stewart [13] suggested the following procedure.

Definition 12.2 Let μ_ϑ be a Ritz value of A restricted to U_ϑ. A solution of the minimization problem

$$\min_{\hat{\mathbf{x}} \in \mathbf{U}_\vartheta, \|\hat{\mathbf{x}}\| = \mathbf{1}} \|A\hat{\mathbf{x}} - \mu_\vartheta \hat{\mathbf{x}}\| \tag{12.35}$$

is called a **refined Ritz vector**.

How is this minimization problem solved? We write $\hat{\mathbf{x}} = U_\vartheta \mathbf{z}$. Then (12.35) becomes

$$\min_{\|\mathbf{z}\|=1} \|(A - \mu_\vartheta I) U_\vartheta \mathbf{z}\|. \tag{12.36}$$

This minimization problem is solved by the right singular vector corresponding to the smallest singular value of $(A - \mu_\vartheta I)U_\vartheta$ or, equivalently, the eigenvector corresponding to the smallest eigenvalue of

$$U_\vartheta^* (A - \mu_\vartheta I)^* (A - \mu_\vartheta I) U_\vartheta \mathbf{z} = \tau \mathbf{z}.$$

We continue the example of before.

```
>> [u,s,v]=svd((A - 0*eye(3))*U)

u =
         0    1.0000         0
   -0.7071         0    0.7071
    0.7071         0    0.7071

s =
    1.0000         0
         0         0
         0         0

v =
     0     1
    -1     0

>> U*v(:,2)

ans =
     1
     0
     0

>> [u,s,v]=svd((A - L(1,1)*eye(3))*U1)

u =
   -0.0000    0.5810    0.8139
   -0.7071   -0.5755    0.4108
    0.7071   -0.5755    0.4108

s =
    1.0001         0
         0    0.0000
         0         0

v =
   -0.0001    1.0000
   -1.0000   -0.0001

>> format long
>> U1*v(:,2)

ans =
   1.00009500829405
  -0.00001878470226
   0.00001878647014
```

With the refined Ritz vector approach Steps 11 and 14 in Algorithm 12.2 are replaced by

```
11: Compute the Ritzpair $(\tilde{\vartheta}, \tilde{\mathbf{q}})$ of $A$ closest to the target value.
    Compute the smallest singular vector $\tilde{\mathbf{s}}$ of $AV_j - \tilde{\vartheta}V_j$.
14: Replace $\tilde{\mathbf{q}}$ by $V_j\tilde{\mathbf{s}}$.
```

12.8 The generalized Schur decomposition

The QZ algorithm computes the following generalized Schur decomposition.

Theorem 12.3 ***(Generalized Schur decomposition)*** *If $A, B \in \mathbb{C}^{n\times n}$ then there are unitary matrices $Q, Z \in \mathbb{C}^{n\times n}$ such that*

$$Q^*AZ = T^A, \qquad Q^*BZ = T^B, \tag{12.37}$$

are upper triangular. If for some k, $t^A_{kk} = t^B_{kk} = 0$ then $\sigma(A, B) = \mathbb{C}$. Otherwise

$$\sigma(A, B) = \{t^A_{ii}/t^B_{ii} \mid t^B_{ii} \neq 0\}.$$

Proof. See [5] ■

The algorithm starts out with transforming A and B in Hessenberg and upper triangular form, respectively. After defalting zeros in the lower offdiagonal of the Hessenberg matrix and on the diagonal of the upper triangular matrix, the QR algorithm with implicit shifts is applied to AB^{-1}. For details see [5].

Corresponding to the notion of an invariant subspace for a single matrix we have the notion of a **deflating subspace** for the **pencil** $A - \lambda B$. In particular, we say that a k-dimensional subspace $\mathcal{S} \subset \mathbb{F}^n$ is "deflating" for the pencil $A - \lambda B$ if the subspace $\{A\mathbf{x} + B\mathbf{y} \mid \mathbf{x}, \mathbf{y} \in \mathcal{S}\}$ has dimension k or less. Note that the columns of the matrix Z in the generalized Schur decomposition define a family of deflating subspaces, for if $Q = [\mathbf{q}_1, \ldots, \mathbf{q}_n]$ and $Z = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$ then we have $\text{span}\{A\mathbf{z}_1, \ldots, A\mathbf{z}_k\} \subset \text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}$ and $\text{span}\{B\mathbf{z}_1, \ldots, B\mathbf{z}_k\} \subset \text{span}\{\mathbf{q}_1, \ldots, \mathbf{q}_k\}$.

12.9 JDQZ: Computing a partial QZ decomposition by the Jacobi–Davidson algorithm

We now consider the generalized eigenvalue problem

$$A\mathbf{x} = \lambda B\mathbf{x}, \tag{12.38}$$

with *arbitrary* A and B. There is a variant of Jacobi–Davidson called JDQZ that computes a *partial* QZ decomposition of the stencil (A, B). This section follows closely the corresponding section in the eigenvalue templates [12]. Further details are found in [3].

With $\lambda = \alpha/\beta$, the generalized eigenproblem (12.38) is equivalent to the eigenproblem

$$(\beta A - \alpha B)\mathbf{x} = 0, \tag{12.39}$$

where we denote a generalized eigenvalue of the matrix pair $\{A, B\}$ as a pair (α, β). The notation (12.39) is preferred over (12.40), because underflow or overflow for $\lambda = \alpha/\beta$ in finite precision arithmetic may occur when α and/or β are zero or close to zero. It also emphazises the symmetry of the roles of A and B.

A **partial generalized Schur** form of dimension k for a matrix pair $\{A, B\}$ is the decomposition

$$AQ_k = Z_kR^A_k, \quad BQ_k = Z_kR^B_k, \tag{12.40}$$

where Q_k and Z_k are unitary $n \times k$ matrices and R^A_k and R^B_k are upper triangular $k \times k$ matrices. A column $\mathbf{q}_i$ of Q_k is referred to as a generalized Schur vector, and we refer to a pair $((\alpha_i, \beta_i), \mathbf{q}_i)$, with $(\alpha_i, \beta_i) = (R_k^A(i,i), R_k^B(i,i))$ as a generalized Schur pair. It follows that if $((\alpha, \beta), \mathbf{y})$ is a generalized eigenpair of (R_k^A, R_k^B) then $((\alpha, \beta), Q_k\mathbf{y})$ is a generalized eigenpair of $\{A, B\}$.

From the relations (12.40) we see that

$$\beta_i A\mathbf{q}_i - \alpha_i B\mathbf{q}_i \perp \mathbf{z}_i.$$

This somewhat resembles the Schur decomposition, where $A\mathbf{q}_i - \lambda_i\mathbf{q}_i \perp \mathbf{q}_i$. The $\mathbf{z}_i$ on the right hand side suggests that we should follow a *Petrov-Galerkin condition* for the construction of reduced systems. In each step the approximate eigenvector $\mathbf{u}$ is selected from a j-dimensional search subspace $\mathrm{span}(V_j) = \mathrm{span}\{\mathbf{v}_1, \ldots, \mathbf{v}_j\}$. We require that the residual $\eta A\mathbf{u} - \zeta B\mathbf{u}$ is orthogonal to some *other* well-chosen test subspace $\mathrm{span}(W_j) = \mathrm{span}\{\mathbf{w}_1, \ldots, \mathbf{w}_j\}$,

$$\eta\, A\mathbf{u} - \zeta\, B\mathbf{u} \perp \mathrm{span}(W_j). \tag{12.41}$$

Equation (12.41) leads to the projected generalized $j \times j$ eigenproblem

$$(\eta\, W_j^* A V_j - \zeta\, W_j^* B V_j)\, \mathbf{s} = 0. \tag{12.42}$$

The j-dimensional pencil $\eta\, W_j^* A V_j - \zeta\, W_j^* B V_j$ can be reduced by the QZ algorithm (see §12.8) to generalized Schur form. This leads to orthogonal $j \times j$ matrices S^R and S^L and upper triangular $j \times j$ matrices T^A and T^B, such that

$$(S^L)^*(W_j^* A V_j) S^R = T^A \quad \text{and} \quad (S^L)^*(W_j^* B V_j) S^R = T^B. \tag{12.43}$$

This decomposition can be reordered such that the first column of S^R and the $(1,1)$-entries of T^A and T^B represent the wanted Petrov solution [3]. With $\mathbf{s} := \mathbf{s}_1^R := S^R\mathbf{e}_1$ and $\zeta := T_{1,1}^A$, $\eta := T_{1,1}^B$, the Petrov vector is defined as

$$\mathbf{u} := V_j\mathbf{s} = V_j\mathbf{s}_1^R$$

for the associated generalized Petrov value (ζ, η). In an analogous way we can define a *left* Petrov vector as

$$\mathbf{p} := W_j\mathbf{s}_1^L \qquad \mathbf{s}_1^L := S^L\mathbf{e}_1$$

If V_j and W_j are unitary, as in Algorithm 12.3, then $\|\mathbf{s}^R\|_2 = \|\mathbf{s}^L\|_2 = 1$ implies $\|\mathbf{u}\|_2 = 1$.

With the decomposition in (12.43), we construct an approximate partial generalized Schur form (cf. (12.40)): $V_j S^R$ approximates a Q_k, and $W_j S^L$ approximates the associated Z_j.

It is not yet clear how to choose the test space W_j. The equations $\mathrm{span}(Z_j) = \mathrm{span}(AQ_j) = \mathrm{span}(BQ_j)$, cf. (12.40), suggest to choose W_j such that $\mathrm{span}(W_j)$ coincides with $\mathrm{span}(\nu_0 AV_j + \mu_0 BV_j)$ for some suitably chosen ν_0 and μ_0. With the weights ν_0 and μ_0 we can influence the convergence of the Petrov values. If we want eigenpair approximations for eigenvalues λ close to a target τ, then the choice

$$\nu_0 = 1/\sqrt{1 + |\tau|^2}, \qquad \mu_0 = -\tau\nu_0$$

is very effective [3], especially if we want to compute eigenvalues in the interior of the spectrum of $A - \lambda B$. We will call the Petrov approximations for this choice the harmonic Petrov eigenpairs. The Jacobi-Davidson correction equation for the component $\mathbf{t} \perp \mathbf{u}$ for the pencil $\eta A - \zeta B$ becomes

$$(I - \mathbf{p}\mathbf{p}^*)\,(\eta A - \zeta B)\,(I - \mathbf{u}\mathbf{u}^*)\,\mathbf{t} = -\mathbf{r}, \qquad \mathbf{r} := \eta A\mathbf{u} - \zeta B\mathbf{u}. \tag{12.44}$$

Sleijpen et al. [10] have shown that if (12.44) is solved exactly, the convergence to the generalized eigenvalue is quadratic. Usually, this correction equation is solved only approximately, for instance, with a (preconditioned) iterative solver. The obtained vector $\mathbf{t}$ is used for the expansion $\mathbf{v}$ of V_j and $\nu_0 A\mathbf{v} + \mu_0 B\mathbf{v}$ is used for the expansion of W_j. For both spaces we work with orthonormal bases. Therefore, the new columns are orthonormalized with respect to the current basis by a modified Gram-Schmidt orthogonalization process.

12.9.1 Restart

Suppose that the generalized Schur form (12.43) is ordered with respect to τ such that

$$|T^A_{1,1}/T^B_{1,1} - \tau| \le |T^A_{2,2}/T^B_{2,2} - \tau| \le \cdots \le |T^A_{j,j}/T^B_{j,j} - \tau|,$$

where j is the dimension of span(V_j). Then, for $i < j$, the space span$(V_j\mathbf{s}^R_1, \ldots, V_j\mathbf{s}^R_i)$ spanned by the first i columns of $V_j S^R$ contains the i most promising Petrov vectors. The corresponding test subspace is given by span$(W_j\mathbf{s}^L, \ldots, W\mathbf{s}^L_i)$. Therefore, in order to reduce the dimension of the subspaces ("implicit restart") to $j_{\min}$, $j_{\min} < j$, the columns $\mathbf{v}_{j_{\min}+1}$ through $\mathbf{v}_j$ and $\mathbf{w}_{j_{\min}+1}$ through $\mathbf{w}_j$ can simply be discarded and the Jacobi-Davidson algorithm can be continued with

$$V = [V\mathbf{s}^R_1, \ldots, V\mathbf{s}^R_{j_{\min}}] \quad \text{and} \quad W = [W\mathbf{s}^L_1, \ldots, W\mathbf{s}^L_{j_{\min}}].$$

12.9.2 Deflation

Like in the Jacobi-Davidson algorithm for the standard eigenvalue problem, in the Jacobi-Davidson process for the generalized eigenvalue problem found (converged) Ritz (here Petrov) vectors can be *deflated*.

The partial generalized Schur form can be obtained in a number of successive steps. Suppose that we have already available the partial generalized Schur form $AQ_{k-1} = Z_{k-1}R^A_{k-1}$ and $BQ_{k-1} = Z_{k-1}R^B_{k-1}$. We want to expand this partial generalized Schur form with the new right Schur vector $\mathbf{u}$ and the left Schur vector $\mathbf{p}$ to

$$A[Q_{k-1}\mathbf{u}] = [Z_{k-1}\mathbf{p}] \begin{bmatrix} R^A_{k-1} & \mathbf{a} \\ 0 & \alpha \end{bmatrix}$$

and

$$A[Q_{k-1}\mathbf{u}] = [Z_{k-1}\mathbf{p}] \begin{bmatrix} R^B_{k-1} & \mathbf{b} \\ 0 & \beta \end{bmatrix}$$

The new generalized Schur pair $((\alpha, \beta), \mathbf{u})$ satisfies

$$Q^*_{k-1}\mathbf{u} = \mathbf{0} \quad \text{and} \quad (\beta A - \alpha B)\mathbf{u} - Z_{k-1}(\beta\mathbf{a} - \alpha\mathbf{b}) = \mathbf{0},$$

or, since $\beta\mathbf{a} - \alpha\mathbf{b} = Z^*_{k-1}(\beta A - \alpha B)\mathbf{u}$,

$$Q^*_{k-1}\mathbf{u} = \mathbf{0} \quad \text{and} \quad \left(I - Z_{k-1}Z^*_{k-1}\right)(\beta A - \alpha B)\left(I - Q_{k-1}Q^*_{k-1}\right)\mathbf{u} = \mathbf{0}.$$

Hence, the vectors $\mathbf{a}$ and $\mathbf{b}$ can be computed from

$$\mathbf{a} = Z^*_{k-1}A\mathbf{u} \quad \text{and} \quad \mathbf{b} = Z^*_{k-1}B\mathbf{u}.$$

Furthermore, the generalized Schur pair $((\alpha, \beta), \mathbf{u})$ is an eigenpair of the deflated matrix pair

$$\left(\left(I - Z_{k-1}Z_{k-1}^*\right) A \left(I - Q_{k-1}Q_{k-1}^*\right), \left(I - Z_{k-1}Z_{k-1}^*\right) B \left(I - Q_{k-1}Q_{k-1}^*\right)\right).$$

This eigenproblem can be solved again with the Jacobi-Davidson QZ process. In that process we construct vectors v_i that are orthogonal to Q_{k-1} and vectors w_i that are orthogonal to Z_{k-1}. This simplifies the computation of the interaction matrices M^A and M^B, associated with the deflated operators

$$\begin{cases} M^A \equiv W^* \left(I - Z_{k-1}Z_{k-1}*\right) A \left(I - Q_{k-1}Q_{k-1}*\right) V = W^*AV, \\ M^A \equiv W^* \left(I - Z_{k-1}Z_{k-1}*\right) B \left(I - Q_{k-1}Q_{k-1}*\right) V = W^*BV, \end{cases}$$

and M^A and M^B can be simply computed as W^*AV and W^*BV, respectively.

12.9.3 Algorithm

The Jacobi-Davidson algorithm to compute a partial QZ decomposition for a general matrix pencil (A, B) is given in Algorithm 12.3 This algorithm attempts to compute the generalized Schur pairs $((\alpha, \beta), q)$, for which the ratio β/α is closest to a specified target value τ in the complex plane. The algorithm includes restart in order to limit the dimension of the search space, and deflation with already converged left and right Schur vectors.

To apply this algorithm we need to specify a starting vector v_0, a tolerance ϵ, a target value τ, and a number $k_{\max}$ that specifies how many eigenpairs near τ should be computed. The value of $j_{\max}$ specifies the maximum dimension of the search subspace. If it is exceeded then a restart takes place with a subspace of dimension $j_{\min}$.

On completion the $k_{\max}$ generalized eigenvalues close to τ are delivered, and the corresponding reduced Schur form $AQ = ZR^A$, $BQ = ZR^B$, where Q and Z are n by $k_{\max}$ orthogonal and R^A, R^B are $k_{\max}$ by $k_{\max}$ upper triangular. The generalized eigenvalues are the on-diagonals of R^A and R^B. The computed form satisfies $\|A\mathbf{q}_j - ZR^Ae_j\|_2 = O(\epsilon)$, $\|B\mathbf{q}_j - ZR^Be_j\|_2 = O(\epsilon)$, where $\mathbf{q}_j$is the jth column of Q.

12.10 Jacobi-Davidson for nonlinear eigenvalue problems

Nonlinear eigenvalue problems have the form

$$T(\lambda)\mathbf{x} = \mathbf{0} \tag{12.45}$$

where the $n \times n$ matrix $T(\lambda)$ has elements that depend on the scalar parameter λ. For the *linear* eigenvalue problem $T(\lambda) = A - \lambda B$. λ is an eigenvalue of (12.45) if $T(\lambda)$ is singular; a nontrivial solution $\mathbf{x}$ of the singular linear system is a corresponding eigenvector.

For small problems, Newton iteration is applicable. Ruhe [9] suggests to proceed as follows. Complement (12.45) by a normalization condition

$$\mathbf{v}^*\mathbf{x} = 1. \tag{12.46}$$

Then, we solve

$$P\begin{pmatrix} \mathbf{x} \\ \lambda \end{pmatrix} = \begin{pmatrix} T(\lambda)\mathbf{x} \\ \mathbf{v}^*\mathbf{x} - 1 \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}. \tag{12.47}$$

Algorithm 12.3 Jacobi–Davidson QZ method for $k_{\max}$ interior eigenvalues close to τ for the generalized non-Hermitian eigenvalue problem

```
Let $A, B \in \mathbb{F}^{n\times n}$ be non-Hermitian. This algorithm computes $k_{\max}$ interior eigenvalues
of $\alpha A\mathbf{x} = \beta B\mathbf{x}$ close to the target $\tau$.
$\mathbf{t} = \mathbf{v}_0$; $k = 0$; $\nu_0 = 1/\sqrt{1+|\tau|^2}$; $\mu_0 = -\tau\nu_0$; $m = 0$;
$Q = []$; $Z = []$; $S = []$; $T = []$;
while $k < k_{\max}$ do
  Orthogonalize $\mathbf{t} := \mathbf{t} - V_m V_m^*\mathbf{t}$
  $m = m+1$; $\mathbf{v}_m = \mathbf{t}/\|\mathbf{t}\|$; $\mathbf{v}_m^A := A\mathbf{v}_m$; $\mathbf{v}_m^B := B\mathbf{v}_m$; $\mathbf{w} := \nu_0\mathbf{v}_m^A + \mu_0\mathbf{v}_m^B$;
  Orthogonalize $\mathbf{w} := \mathbf{w} - Z_k Z_k^*\mathbf{w}$
  Orthogonalize $\mathbf{w} := \mathbf{w} - W_{m-1}W_{m-1}^*\mathbf{w}$
  $\mathbf{w}_m = \mathbf{w}/\|\mathbf{w}\|$;
  $$M^A := \begin{bmatrix} M^A & W_{m-1}^*\mathbf{v}_m^A \\ \mathbf{w}_m^*V_{m-1}^A & \mathbf{w}_m^*\mathbf{v}_m^A \end{bmatrix}; \qquad M^B := \begin{bmatrix} M^B & W_{m-1}^*\mathbf{v}_m^B \\ \mathbf{w}_m^*V_{m-1}^B & \mathbf{w}_m^*\mathbf{v}_m^B \end{bmatrix};$$
  Compute the QZ decomposition $M^AS^R = S^LT^A$, $M^BS^R = S^LT^B$, such that
  $|T^A_{i,i}/T^B_{i,i} - \tau| \le |T^A_{i+1,i+1}/T^B_{i+1,i+1} - \tau|$        /* Rayleigh Ritz step */
  $\mathbf{u} := V\mathbf{s}_1^R$; $\mathbf{p} := W_j\mathbf{s}_1^L$; $\mathbf{u}^A := V^A\mathbf{s}_1^R$; $\mathbf{u}^B := V^B\mathbf{s}_1^R$; $\zeta = T^A_{1,1}$; $\eta = T^B_{1,1}$;
  $\mathbf{r} = \eta\mathbf{u}^A - \zeta\mathbf{u}^B$; $\tilde{\mathbf{a}} = Z^*\mathbf{u}^A$; $\tilde{\mathbf{b}} = Z^*\mathbf{u}^B$; $\tilde{\mathbf{r}} = \mathbf{r} - Z(\eta\tilde{\mathbf{a}} - \zeta\tilde{\mathbf{b}})$;
  while $\|\tilde{\mathbf{r}}\| < \varepsilon$ do
    $$R^A := \begin{bmatrix} R^A & \tilde{\mathbf{a}} \\ \mathbf{0}^T & \zeta \end{bmatrix}; \qquad R^B := \begin{bmatrix} R^B & \tilde{\mathbf{b}} \\ \mathbf{0}^T & \eta \end{bmatrix};$$
    $Q := [Q, \mathbf{u}]$; $Z := [Z, \mathbf{p}]$; $k := k+1$;
    if $k = k_{\max}$ then
      return $(Q, Z, R^A, R^B)$
    end if
    $m := m-1$;
    for $i = 1, \ldots, m$ do
      $\mathbf{v}_i := V\mathbf{s}_{i+1}^R$; $\mathbf{v}_i^A := V^A\mathbf{s}_{i+1}^R$; $\mathbf{v}_i^B := V^B\mathbf{s}_{i+1}^R$;
      $\mathbf{w}_i := W\mathbf{s}_{i+1}^L$; $\mathbf{s}_i^R := \mathbf{s}_i^L := \mathbf{e}_i$;
    end for
    $M^A$, $M^B$ is the lower $m \times m$ block of $T^A$, $T^B$, resp.
    $\mathbf{u} := \mathbf{u}_1$; $\mathbf{p} := \mathbf{w}_1$; $\mathbf{u}^A := \mathbf{v}_1^A$; $\mathbf{u}^B := \mathbf{v}_1^b$; $\zeta = T^A_{1,1}$; $\eta = T^B_{1,1}$;
    $\mathbf{r} = \eta\mathbf{u}^A - \zeta\mathbf{u}^B$; $\tilde{\mathbf{a}} = Z^*\mathbf{u}^A$; $\tilde{\mathbf{b}} = Z^*\mathbf{u}^B$; $\tilde{\mathbf{r}} = \mathbf{r} - Z(\eta\tilde{\mathbf{a}} - \zeta\tilde{\mathbf{b}})$;
  end while
  if $m \ge m_{\max}$ then
    for $i = 2, \ldots, m_{\min}$ do
      $\mathbf{v}_i := V\mathbf{s}_i^R$; $\mathbf{v}_i^A := V^A\mathbf{s}_i^R$; $\mathbf{v}_i^B := V^B\mathbf{s}_i^R$; $\mathbf{w}_i := W\mathbf{s}_i^L$;
    end for
    $M^A$, $M^B$ is the leading $m_{\min} \times m_{\min}$ block of $T^A$, $T^B$, resp.
    $\mathbf{v}_1 := \mathbf{u}$; $\mathbf{v}_1^A := \mathbf{u}^A$; $\mathbf{v}_1^B := \mathbf{u}^B$; $\mathbf{w}_1 := \mathbf{p}$; $m := m_{\min}$
  end if
  $\tilde{Q} := [Q, \mathbf{u}]$; $\tilde{Z} := [Z, \mathbf{p}]$;
  (Approximatively) solve the correction equation for $\mathbf{t} \perp \tilde{Q}$,
      $(I - \tilde{Z}\tilde{Z}^*)(\eta A - \zeta B)(I - \tilde{Q}\tilde{Q}^*)$
end while
```

For the derivative of P we obtain

$$P' = \begin{bmatrix} T(\lambda) & T'(\lambda)\mathbf{x} \\ \mathbf{v}^* & 0 \end{bmatrix}$$

such that the Newton iteration becomes

$$\begin{pmatrix} \mathbf{x}_{s+1} \\ \lambda_{s+1} \end{pmatrix} = \begin{pmatrix} \mathbf{x}_s \\ \lambda_s \end{pmatrix} - \begin{bmatrix} T(\lambda_s) & T'(\lambda_s)\mathbf{x}_s \\ \mathbf{v}_s^* & 0 \end{bmatrix}^{-1} \begin{pmatrix} T(\lambda_s)\mathbf{x}_s \\ \mathbf{v}_s^*\mathbf{x}_s - 1 \end{pmatrix} \tag{12.48}$$

or

$$\begin{aligned} T(\lambda_s)\mathbf{u}_{s+1} &= T'(\lambda_s)\mathbf{x}_s, \\ \lambda_{s+1} &= \lambda_s - (\mathbf{v}_s^*\mathbf{x}_s)/(\mathbf{v}_s^*\mathbf{x}_{s+1}), \\ \mathbf{x}_{s+1} &= C \cdot \mathbf{u}_{s+1}. \end{aligned} \tag{12.49}$$

Here, C is some normalization constant. The vector $\mathbf{v}_s$ may depend on the iteration step. It can be chosen in a number of ways. It could be constant, e.g., $\mathbf{v}_s = \mathbf{e}_i$. This amounts to keeping one of the entries of $\mathbf{x}_s$ constant. Another choce is

$$\mathbf{v}_s = T(\lambda_s)^*\mathbf{y}_s$$

where $\mathbf{y}_s$ is an approximation to the left eigenvector $\mathbf{y}$.

A Jacobi-Davidson algorithm for large nonlinear eigenvalue problems is given in Algorithm 12.4. This algorithm is by Voss [14]. There are two noteworthy issues.

Algorithm 12.4 Nonlinear Jacobi–Davidson algorithm

```
Start with an initial basis $V$, $V^*V = I$; $m = 1$.
Determine a preconditioner $K \approx T(\sigma)$, $\sigma$ close to the first wanted eigenvalue.
while $m \le$ number of wanted eigenvalues do
    Compute an approximation to the $m$-th wanted eigenvalue $\lambda_m$ and corresponding
    eigenvector $\mathbf{s}_m$ of the projected problem $V^*T(\lambda_m)V\mathbf{s} = \mathbf{0}$.
    Determine the Ritz vector $\mathbf{u} = V\mathbf{s}_m$ and the residual $\mathbf{r} = T(\lambda_m)\mathbf{u}$
    if $\|\mathbf{r}\|/\|\mathbf{u}\| < \varepsilon$ then
        Accept approximate eigenpair $(\lambda_m, \mathbf{u})$; $m := m + 1$;
        Reduce the search space $V$ if necessary
        Choose an approximation $(\lambda_m, \mathbf{u})$ to the next eigenpair.
        Compute the residual $\mathbf{r} = T(\lambda_m)\mathbf{u}$
    end if
    $\mathbf{p} = T'(\lambda_m)\mathbf{x}$;
    (Approximatively) solve the correction equation for $\mathbf{t}$,

    (12.50)    $(I - \frac{\mathbf{p}\mathbf{u}^*}{\mathbf{u}^*\mathbf{p}})T(\sigma)(I - \frac{\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*\mathbf{u}})\mathbf{t} = -\mathbf{r}, \qquad \mathbf{t} \perp \mathbf{u}.$

    Orthogonalize $\mathbf{t} := \mathbf{t} - VV^*\mathbf{t}$, $\mathbf{v} := \mathbf{t}/\|\mathbf{t}\|$, and expand the subspace $[V, \mathbf{v}]$.
    Determine a new preconditioner $K \approx T(\lambda_m)$ if necessary.
    Update the projected problem.
end while
```

1. The projected problem is the *nonlinear* eigenvalue problem $V^*T(\lambda_m)V\mathbf{s} = \mathbf{0}$ where λ_m is an approximation to the wanted eigenvalue.

2. In the expansion of the search space, it is ensured that the Newton iterate is contained in the expanded search space. To this end, assume that $\mathbf{u}$ is the Ritz vector in $\mathcal{R}(V)$ obtained from the projected problem, $\mathbf{u} = V\mathbf{s}$. Set $\mathbf{p} = T'(\lambda_m)\mathbf{u}$. We now solve the correction equation

   $$(I - \frac{\mathbf{p}\mathbf{u}^*}{\mathbf{u}^*\mathbf{p}})T(\lambda_m)(I - \frac{\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*\mathbf{u}})\mathbf{t} = -\mathbf{r} = -T(\lambda_m)\mathbf{u}, \qquad \mathbf{t} \perp \mathbf{u}. \tag{12.50}$$

   This equation can be written as

   $$T(\lambda_m)\mathbf{t} - \alpha\mathbf{p} = -\mathbf{r}, \qquad \alpha = \frac{1}{\mathbf{u}^*\mathbf{p}}\mathbf{u}^*T(\lambda_m)\mathbf{t}.$$

   Using $T(\lambda_m)\mathbf{u} = \mathbf{r}$ we get

   $$\mathbf{t} = -\mathbf{u} + \alpha T(\lambda_m)^{-1}\mathbf{p} = -\mathbf{u} + \alpha T(\lambda_m)^{-1}T'(\lambda_m)\mathbf{u}.$$

   α is determined such that $\mathbf{t} \perp \mathbf{u}$. Since $\mathbf{u} \in \mathcal{R}(V)$, we must have $T(\lambda_m)^{-1}T'(\lambda_m)\mathbf{u} \in \mathcal{R}([V, \mathbf{t}])$. This ensures the quadratic convergence rate of Newton's method.

The correction equation (12.50) in Algorithm 12.4 is typically solved to low accuracy by a preconditioned GMRES iteration where the preconditioner has the form

$$\tilde{K} = (I - \frac{\mathbf{p}\mathbf{u}^*}{\mathbf{u}^*\mathbf{p}})K(I - \frac{\mathbf{u}\mathbf{u}^*}{\mathbf{u}^*\mathbf{u}}), \qquad K \approx T(\sigma). \tag{12.51}$$

Solving with the preconditioner amounts to solving the equation

$$\tilde{K}\mathbf{t} = \mathbf{g}, \qquad \mathbf{t} \perp \mathbf{u}.$$

Bibliography

[1] R. Barret, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, PA, 1994. (Available from Netlib at URL `http://www.netlib.org/templates/index.html`).

[2] E. R. Davidson, *The iterative calculation of a few of the lowest eigenvalues and corresponding eigenvectors of large real-symmetric matrices*, J. Comp. Phys., 17 (1975), pp. 87–94.

[3] D. R. Fokkema, G. L. G. Sleijpen, and H. A. van der Vorst, *Jacobi–Davidson style QR and QZ algorithms for the partial reduction of matrix pencils*, SIAM J. Sci. Comput., 20 (1998), pp. 94–125.

[4] R. Geus, *The Jacobi–Davidson algorithm for solving large sparse symmetric eigenvalue problems*, PhD Thesis No. 14734, ETH Zürich, 2002. (Available at URL `http://e-collection.ethbib.ethz.ch/show?type=diss&nr=14734`).

[5] G. H. Golub and C. F. van Loan, *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD, 2nd ed., 1989.

[6] C. G. J. Jacobi, *Über ein leichtes Verfahren die in der Theorie der Säcularstörungen vorkommenden Gleichungen numerisch aufzulösen*, J. reine angew. Math., 30 (1846), pp. 51–95.

[7] R. B. Morgan, *Davidson's method and preconditioning for generalized eigenvalue problems*, J. Comp. Phys., 89 (1990), pp. 241–245.

[8] R. B. Morgan and D. S. Scott, *Generalizations of Davidson's method for computing eigenvalues of sparse symmetric matrices*, SIAM J. Sci. Stat. Comput., 7 (1986), pp. 817–825.

[9] A. Ruhe, *Algorithms for the nonlinear eigenvalue problem*, SIAM J. Numer. Anal., 10 (1973), pp. 674–689.

[10] G. L. G. Sleijpen, A. G. L. Booten, D. R. Fokkema, and H. A. van der Vorst, *Jacobi–Davidson type methods for generalized eigenproblems and polynomial eigenproblems*, BIT, 36 (1996), pp. 595–633.

[11] G. L. G. Sleijpen and H. A. van der Vorst, *A Jacobi–Davidson iteration method for linear eigenvalue problems*, SIAM J. Matrix Anal. Appl., 17 (1996), pp. 401–425.

[12] G. L. G. Sleijpen and H. A. van der Vorst, *Jacobi–Davidson method*, in Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., SIAM, Philadelphia, PA, 2000, pp. 238–246.

[13] G. W. Stewart, *Matrix Algorithms II: Eigensystems*, SIAM, Philadelphia, PA, 2001.

[14] H. Voss, *A Jacobi–Davidson method for nonlinear eigenproblems*, in Computational Science – ICCS 2004, G. D. van Albada, M. Bubak, P. M. A. Sloot, and J. J. Dongarra, eds., Berlin, 2004, Springer, pp. 34–41. (Lecture Notes in Computer Science, 3037).

Chapter 13

Rayleigh quotient and trace minimization

13.1 Introduction

In this chapter we restrict ourselves to the symmetric/Hermitian eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad A = A^*, \quad M = M^* > 0. \tag{13.1}$$

We want to exploit the property of the Rayleigh quotient that

$$\lambda_1 = \min_{\mathbf{x}\neq\mathbf{0}} \rho(\mathbf{x}) \qquad \rho(\mathbf{x}) = \frac{\mathbf{x}^*A\mathbf{x}}{\mathbf{x}^*M\mathbf{x}}, \tag{13.2}$$

which was proved in Theorem 2.30. The basic idea of Rayleigh quotient minimization is to construct a sequence $\{\mathbf{x}_k\}_{k=1,2,\ldots}$ such that $\rho(\mathbf{x}_{k+1}) < \rho(\mathbf{x}_k)$ for all k. The hope is that the sequence $\{\rho(\mathbf{x}_k)\}$ converges to λ_1 and by consequence the vector sequence $\{\mathbf{x}_k\}$ towards the corresponding eigenvector.

The procedure is as follows: For any given $\mathbf{x}_k$ let us choose a **search direction $\mathbf{p}_k$**, so that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{p}_k. \tag{13.3}$$

The parameter δ_k is determined such that the Rayleigh quotient of the new iterate $\mathbf{x}_{k+1}$ becomes minimal,

$$\rho(\mathbf{x}_{k+1}) = \min_{\delta} \rho(\mathbf{x}_k + \delta\mathbf{p}_k). \tag{13.4}$$

We can write the Rayleigh quotient of the linear combination $\mathbf{x}_k + \delta\mathbf{p}_k$ of two (linearly independant) vectors $\mathbf{x}_k$ and $\mathbf{p}_k$ as

$$\rho(\mathbf{x}_k + \delta\mathbf{p}_k) = \frac{\mathbf{x}_k^*A\mathbf{x}_k + 2\delta\mathbf{x}_k^*A\mathbf{p}_k + \delta^2\mathbf{p}_k^*A\mathbf{p}_k}{\mathbf{x}_k^*M\mathbf{x}_k + 2\delta\mathbf{x}_k^*M\mathbf{p}_k + \delta^2\mathbf{p}_k^*M\mathbf{p}_k} = \frac{\begin{pmatrix}1\\ \delta\end{pmatrix}^* \begin{bmatrix}\mathbf{x}_k^*A\mathbf{x}_k & \mathbf{x}_k^*A\mathbf{p}_k\\ \mathbf{p}_k^*A\mathbf{x}_k & \mathbf{p}_k^*A\mathbf{p}_k\end{bmatrix}\begin{pmatrix}1\\ \delta\end{pmatrix}}{\begin{pmatrix}1\\ \delta\end{pmatrix}^* \begin{bmatrix}\mathbf{x}_k^*M\mathbf{x}_k & \mathbf{x}_k^*M\mathbf{p}_k\\ \mathbf{p}_k^*M\mathbf{x}_k & \mathbf{p}_k^*M\mathbf{p}_k\end{bmatrix}\begin{pmatrix}1\\ \delta\end{pmatrix}}. \tag{13.5}$$

This is the Rayleigh quotient associated with the generalized 2×2 eigenvalue problem

$$\begin{bmatrix}\mathbf{x}_k^*A\mathbf{x}_k & \mathbf{x}_k^*A\mathbf{p}_k\\ \mathbf{p}_k^*A\mathbf{x}_k & \mathbf{p}_k^*A\mathbf{p}_k\end{bmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix} = \lambda\begin{bmatrix}\mathbf{x}_k^*M\mathbf{x}_k & \mathbf{x}_k^*M\mathbf{p}_k\\ \mathbf{p}_k^*M\mathbf{x}_k & \mathbf{p}_k^*M\mathbf{p}_k\end{bmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix}. \tag{13.6}$$

The smaller of the two eigenvalues of (13.6) is the searched value $\rho_{k+1} := \rho(\mathbf{x}_{k+1})$ in (13.4) that minimizes the Rayleigh quotient. The corresponding eigenvector is normalized such that its first component equals one[1]. The second component of this eigenvector is $\delta = \delta_k$. Inserting the solution $[1, \delta_k]^*$ into the second line of (13.6) we obtain

$$\mathbf{p}_k^*(A - \rho_{k+1}M)(\mathbf{x}_k + \delta_k\mathbf{p}_k) = \mathbf{p}_k^*\mathbf{r}_{k+1} = 0. \tag{13.7}$$

So, the ‘next’ residual $\mathbf{r}_{k+1}$ is orthogonal to the actual search direction $\mathbf{p}_k$.

There are various ways how to choose the search direction $\mathbf{p}_k$. A simple way is to cycle through the coordinate vectors, a method that is called coordinate relaxation [3]. It cannot compete with the methods we discuss next; but it has some potential for parallelization.

13.2 The method of steepest descent

Let us make a detour to solving systems of equations

$$A\mathbf{x} = \mathbf{b}, \tag{13.8}$$

where A is symmetric/Hermitian positive definite. Let us define the functional

$$\varphi(\mathbf{x}) \equiv \frac{1}{2}\mathbf{x}^*A\mathbf{x} - \mathbf{x}^*\mathbf{b} + \frac{1}{2}\mathbf{b}^*A^{-1}\mathbf{b} = \frac{1}{2}(A\mathbf{x} - \mathbf{b})^*A^{-1}(A\mathbf{x} - \mathbf{b}). \tag{13.9}$$

The functional φ is minimized at the solution $\mathbf{x}_*$ of (13.8). (With the above definition of φ the minimum is actually zero.) The negative gradient of φ is

$$-\nabla\varphi(\mathbf{x}) = \mathbf{b} - A\mathbf{x} =: \mathbf{r}(\mathbf{x}). \tag{13.10}$$

It is nonzero except at $\mathbf{x}_*$. In the method of steepest descent [2, 3] a sequence of vectors $\{\mathbf{x}_k\}_{k=1,2,\ldots}$ is constructed such that the relation

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{p}_k \tag{13.3}$$

holds among any two consecutive vectors. The search direction $\mathbf{p}_k$ is chosen to be the negative gradient $-\nabla\phi(\mathbf{x}_k) = \mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$. This is the direction in which φ decreases the most. Setting $\mathbf{x}_{k+1}$ as in (13.3) we get

$$0 = \left.\frac{\partial\varphi(\mathbf{x}_{k+1})}{\partial\delta}\right|_{\delta=\delta_k} = \mathbf{p}_k^*(A\mathbf{x}_k - \mathbf{b}) + \delta_k\mathbf{p}_k^*A\mathbf{p}_k = -\mathbf{p}_k^*\mathbf{r}_k + \delta_k\mathbf{p}_k^*A\mathbf{p}_k.$$

Thus,

$$\delta_k = \frac{\mathbf{p}_k^*\mathbf{r}_k}{\mathbf{p}_k^*A\mathbf{p}_k} \tag{13.11}$$

which, for steepest descent, becomes

$$\delta_k = \frac{\mathbf{r}_k^*\mathbf{r}_k}{\mathbf{r}_k^*A\mathbf{r}_k} \tag{13.12}$$

Remark 13.1. Notice that

$$\mathbf{r}_{k+1} = \mathbf{b} - A\mathbf{x}_{k+1} = \mathbf{b} - A(\mathbf{x}_k + \delta_k\mathbf{p}_k) = \mathbf{r}_k - \delta_kA\mathbf{p}_k. \tag{13.13}$$

[1] The first component of this eigenvector is nonzero if it has a component in the direction of the ‘smallest eigenvector’.

Therefore, from (13.11) we have

$$\mathbf{p}_k^*\mathbf{r}_{k+1} = \mathbf{p}_k^*\mathbf{r}_k - \delta_k\mathbf{p}_k^*A\mathbf{p}_k = 0, \tag{13.14}$$

which corresponds to (13.7) in the linear system case. □

For the eigenvalue problem we can proceed similarly by choosing $\mathbf{p}_k$ to be the negative gradient of the Rayleigh quotient ρ,

$$\mathbf{p}_k = -\mathbf{g}_k = -\nabla\rho(\mathbf{x}_k) = -\frac{2}{\mathbf{x}_k^*M\mathbf{x}_k}(A\mathbf{x}_k - \rho(\mathbf{x}_k)M\mathbf{x}_k).$$

Notice that $\mathbf{g}_k$ points in the *same* direction as the residual $\mathbf{r}_k$. (This is in contrast to the linear system case!) Since in eigenvalue problems we only care about directions we can equivalently set

$$\mathbf{p}_k = \mathbf{r}_k = A\mathbf{x}_k - \rho_k M\mathbf{x}_k, \qquad \rho_k = \rho(\mathbf{x}_k). \tag{13.15}$$

With this choice of search direction we immediately have from (13.7) that

$$\mathbf{r}_k^*\mathbf{r}_{k+1} = 0. \tag{13.16}$$

Not surprisingly, the method of steepest descent often converges slowly, as it does for linear systems. This happens if the spectrum is very much spread out, i.e., if the condition number of A relative to M is big.

13.3 The conjugate gradient algorithm

As with linear systems of equations a remedy against the slow convergence of steepest descent are conjugate search directions. So, let's again first look at linear systems [5]. There, we define the search directions as[2]

$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k\mathbf{p}_{k-1}, \qquad k > 0. \tag{13.17}$$

where the coefficient β_k is determined such that $\mathbf{p}_k$ and $\mathbf{p}_{k-1}$ are **conjugate**, i.e.,

$$\mathbf{p}_k^*A\mathbf{p}_{k-1} = -\mathbf{g}_k^*A\mathbf{p}_{k-1} + \beta_k\mathbf{p}_{k-1}^*A\mathbf{p}_{k-1} = 0, \tag{13.18}$$

such that

$$\beta_k = \frac{\mathbf{g}_k^*A\mathbf{p}_{k-1}}{\mathbf{p}_{k-1}^*A\mathbf{p}_{k-1}}. \tag{13.19}$$

Premultiplying (13.17) by $\mathbf{g}_k^*$ gives

$$\mathbf{g}_k^*\mathbf{p}_k = -\mathbf{g}_k^*\mathbf{g}_k + \beta_k\mathbf{g}_k^*\mathbf{p}_{k-1} \overset{(13.14)}{=} -\mathbf{g}_k^*\mathbf{g}_k. \tag{13.20}$$

Furthermore, since $\mathbf{x}_k^*M\mathbf{x}_k = 1$ we have $\mathbf{r}_k = -\mathbf{g}_k$ and

$$\begin{aligned} 0 \overset{(13.14)}{=} \mathbf{g}_{k+1}^*\mathbf{p}_k &\overset{(13.17)}{=} -\mathbf{g}_{k+1}^*\mathbf{g}_k + \beta_k\mathbf{g}_{k+1}^*\mathbf{p}_{k-1} \\ &\overset{(13.13)}{=} -\mathbf{g}_{k+1}^*\mathbf{g}_k + \beta_k\mathbf{g}_k^*\mathbf{p}_{k-1} + \beta_k\delta_k\mathbf{p}_k^*A\mathbf{p}_{k-1}. \end{aligned}$$

[2] In linear systems the residual $\mathbf{r} = \mathbf{b} - A\mathbf{x}$ is defined as the negative gradient whereas in eigenvalue computations it is defined as $\mathbf{r} = A\mathbf{x} - \rho(\mathbf{x})M\mathbf{x}$, i.e., in the same direction as the gradient. To reduce the confusion we proceed using the gradient.

From (13.14) we have that $\mathbf{g}_k^*\mathbf{p}_{k-1} = 0$ and by construction of $\mathbf{p}_k$ and $\mathbf{p}_{k-1}$ being conjugate we have that $\mathbf{p}_k^* A\mathbf{p}_{k-1} = 0$. Thus,

$$\mathbf{g}_{k+1}^*\mathbf{g}_k = 0, \tag{13.21}$$

as with the method of steepest descent. Still in the case of linear systems, using these identities we find formulae equivalent to (13.19),

$$\beta_k = -\frac{\mathbf{g}_k^* A\mathbf{p}_{k-1}}{\mathbf{p}_{k-1}^* A\mathbf{p}_{k-1}} \overset{(13.13)}{=} -\frac{\mathbf{g}_k^*(\mathbf{g}_k - \mathbf{g}_{k-1})}{\mathbf{p}_{k-1}^*(\mathbf{g}_k - \mathbf{g}_{k-1})} \overset{(13.14)}{=} -\frac{\mathbf{g}_k^*(\mathbf{g}_k - \mathbf{g}_{k-1})}{-\mathbf{p}_{k-1}^*\mathbf{g}_{k-1}}$$

$$\overset{(13.20)}{=} \frac{\mathbf{g}_k^*(\mathbf{g}_k - \mathbf{g}_{k-1})}{\mathbf{g}_{k-1}^*\mathbf{g}_{k-1}} \tag{13.22}$$

$$\overset{(13.21)}{=} \frac{\mathbf{g}_k^*\mathbf{g}_k}{\mathbf{g}_{k-1}^*\mathbf{g}_{k-1}}. \tag{13.23}$$

The equivalent identities (13.19), (13.22), and (13.23) can be used to define β_k the most economic being (13.23).

We now look at how a conjugate gradient algorithm for the eigenvalue problem can be devised. The idea is straightforward. The algorithm differs from steepest descent by the choice of the search directions that are kept conjugate, i.e., consecutive search directions satisfy $\mathbf{p}_k^* A\mathbf{p}_{k-1} = 0$.

The crucial difference to linear systems stems from the fact, that the functional that is to be minimized, i.e., the Rayleigh quotient, is not quadratic anymore. (In particular, there is no finite termination property.) The gradient of $\rho(\mathbf{x})$ is

$$\mathbf{g} = \nabla\rho(\mathbf{x}_k) = \frac{2}{\mathbf{x}^* M\mathbf{x}}(A\mathbf{x} - \rho(\mathbf{x})M\mathbf{x}).$$

So, in particular, the equation (13.14), does not hold:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta_k\mathbf{p}_k \quad \not\Longrightarrow \quad \mathbf{g}_{k+1} = \mathbf{g}_k + \delta_k A\mathbf{p}_k.$$

Therefore, in the context of nonlinear systems or eigenvalue problems the formulae in (13.19), (13.22), and (13.23) that define β_k are not equivalent anymore! Feng and Owen [4] extensively compared the three formulae and found that in the context of eigenvalue problems the last identity (13.23) leads to the fastest convergence. So, we opt for this equation and define the search directions according to

$$\begin{cases} \mathbf{p}_0 = -\mathbf{g}_0, & k = 0, \\ \mathbf{p}_k = -\mathbf{g}_k + \dfrac{\mathbf{g}_k^* M\mathbf{g}_k}{\mathbf{g}_{k-1}^* M\mathbf{g}_{k-1}}\mathbf{p}_{k-1}, & k > 0, \end{cases} \tag{13.24}$$

where we have given the formulae for the generalized eigenvalue problem $A\mathbf{x} = \lambda M\mathbf{x}$. The complete procedure is given in Algorithm 13.1

Convergence

The construction of Algorithm 13.1 guarantees that $\rho(\mathbf{x}_{k+1}) < \rho(\mathbf{x}_k)$ unless $\mathbf{r}_k = \mathbf{0}$, in which case $\mathbf{x}_k$ is the searched eigenvector. In general, i.e., if the initial vector $\mathbf{x}_0$ has a nonvanishing component in the direction of the 'smallest' eigenvector $\mathbf{u}_1$, convergence is toward the smallest eigenvalue λ_1. This assumption must also hold for vector iteration or the Lanczos algorithm.

Algorithm 13.1 The Rayleigh quotient algorithm

```
Let $\mathbf{x}_0$ be a unit vector, $\|\mathbf{x}_0\|_M = 1$.
$\mathbf{v}_0 := A\mathbf{x}_0, \quad \mathbf{u}_0 := M\mathbf{x}_0$,
$\rho_0 := \dfrac{\mathbf{v}_0^*\mathbf{x}_0}{\mathbf{u}_0^*\mathbf{x}_0}$,
$\mathbf{g}_0 := 2(\mathbf{v}_0 - \rho_0\mathbf{u}_0)$
while $\|\mathbf{g}_k\| > tol$ do
  if $k = 1$ then
    $\mathbf{p}_k := -\mathbf{g}_{k-1}$;
  else
    $\mathbf{p}_k := -\mathbf{g}_{k-1} + \dfrac{\mathbf{g}_{k-1}^*M\mathbf{g}_{k-1}}{\mathbf{g}_{k-2}^*M\mathbf{g}_{k-2}}\mathbf{p}_{k-1}$;
  end if
  Determine the smallest Ritz value $\rho_k$ and corresponding Ritz vector $\mathbf{x}_k$ of $(A, M)$
  in $\mathcal{R}([\mathbf{x}_{k-1}, \mathbf{p}_k])$
  $\mathbf{v}_k := A\mathbf{x}_k, \quad \mathbf{u}_k := M\mathbf{x}_k$
  $\rho_k := \mathbf{x}_k^*\mathbf{v}_k/\mathbf{x}_k^*\mathbf{u}_k$
  $\mathbf{g}_k := 2(\mathbf{v}_k - \rho_k\mathbf{u}_k)$
end while
```

Let

$$\mathbf{x}_k = \cos\vartheta_k\mathbf{u}_1 + \sin\vartheta_k\mathbf{z}_k =: \cos\vartheta_k\mathbf{u}_1 + \mathbf{w}_k, \tag{13.25}$$

where $\|\mathbf{x}_k\|_M = \|\mathbf{u}_1\|_M = \|\mathbf{z}_k\|_M = 1$ and $\mathbf{u}_1^*M\mathbf{z}_k = 0$. Then we have

$$\begin{aligned}\rho(\mathbf{x}_k) &= \cos^2\vartheta_k\lambda_1 + 2\cos\vartheta_k\sin\vartheta_k\mathbf{u}_1^*A\mathbf{z}_k + \sin^2\vartheta_k\mathbf{z}_k^*A\mathbf{z}_k \\ &= \lambda_1(1-\sin^2\vartheta_k) + \sin^2\vartheta_k\rho(\mathbf{z}_k),\end{aligned}$$

or,

$$\rho(\mathbf{x}_k) - \lambda_1 = \sin^2\vartheta_k\,(\rho(\mathbf{z}_k) - \lambda_1) \le (\lambda_n - \lambda_1)\sin^2\vartheta_k. \tag{13.26}$$

As seen earlier, in symmetric eigenvalue problems, the eigenvalues are much more accurate than the eigenvectors.

Let us now suppose that the eigenvalues have already converged, i.e.,

$$\rho(\mathbf{x}_k) = \rho_k \cong \lambda_1,$$

while the eigenvectors are not yet as accurate as desired. Then we can write

$$\mathbf{r}_k = (A - \rho_k M)\mathbf{x}_k \cong (A - \lambda_1 M)\mathbf{x}_k = \sum_{j=1}^{n}(\lambda_j - \lambda_1)M\mathbf{u}_j\mathbf{u}_j^*M\mathbf{x}_k, \tag{13.27}$$

which entails $\mathbf{u}_1^*\mathbf{r}_k = 0$ since the first summand on the right of (13.27) vanishes. From (13.25) we have $\mathbf{w}_k = \sin\vartheta_k\mathbf{z}_k \perp_M \mathbf{u}_1$. Thus,

$$\begin{cases} (A - \lambda_1 M)\mathbf{w}_k = (A - \lambda_1 M)\mathbf{x}_k = \mathbf{r}_k \perp \mathbf{u}_1 \\ \qquad \mathbf{w}_k^*M\mathbf{u}_1 = 0 \end{cases} \tag{13.28}$$

If λ_1 is a simple eigenvalue of the pencil $(A;B)$ then $A - \lambda_1 M$ is a bijective mapping of $\mathcal{R}(\mathbf{u}_1)^{\perp_M}$ onto $\mathcal{R}(\mathbf{u}_1)^\perp$. If $\mathbf{r} \in \mathcal{R}(\mathbf{u}_1)^\perp$ then the equation

$$(A - \lambda_1 M)\mathbf{w} = \mathbf{r} \tag{13.29}$$

has a *unique* solution in $\mathcal{R}(\mathbf{u}_1)^{\perp_M}$.

So, close to convergence, Rayleigh quotient minimization does nothing else but solving equation (13.29) by the conjugate gradient algorithm. The convergence of the Rayleigh quotient minimization is determined by the condition number of $A-\lambda_1 M$ (as a mapping of $\mathcal{R}(\mathbf{u}_1)^{\perp_M}$ onto $\mathcal{R}(\mathbf{u}_1)^{\perp}$), according to the theory of conjugate gradients for linear system of equations. This condition number is

$$\kappa_0 = \mathcal{K}(A - \lambda_1 M)\Big|_{\mathcal{R}(\mathbf{u}_1)^{\perp_M}} = \frac{\lambda_n - \lambda_1}{\lambda_2 - \lambda_1}, \tag{13.30}$$

and the rate of convergence is given by

$$\frac{\sqrt{\kappa_0} - 1}{\sqrt{\kappa_0} + 1}. \tag{13.31}$$

A high condition number implies slow convergence. We see from (13.31) that the condition number is high if the distance of λ_1 and λ_2 is much smaller than the spread of the spectrum of $(A;B)$. This happens more often than not, in particular with FE discretizations of PDE's.

Preconditioning

In order to reduce the condition number of the eigenvalue problem we change

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

into

$$\tilde{A}\tilde{\mathbf{x}} = \tilde{\lambda}\tilde{M}\tilde{\mathbf{x}}, \tag{13.32}$$

such that

$$\kappa(\tilde{A} - \tilde{\lambda}_1\tilde{M}) \ll \kappa(A - \lambda_1 M). \tag{13.33}$$

To further investigate this idea, let C be a nonsingular matrix, and let $\mathbf{y} = C\mathbf{x}$. Then,

$$\rho(\mathbf{x}) = \frac{\mathbf{x}^*A\mathbf{x}}{\mathbf{x}^*M\mathbf{x}} = \frac{\mathbf{y}^*C^{-*}AC^{-1}\mathbf{y}}{\mathbf{y}^*C^{-*}MC^{-1}\mathbf{y}} = \frac{\mathbf{y}^*\tilde{A}\mathbf{y}}{\tilde{\mathbf{y}}^*\tilde{M}\mathbf{y}} = \tilde{\rho}(\mathbf{y}) \tag{13.34}$$

Thus,

$$\tilde{A} - \lambda_1\tilde{M} = C^{-*}(A - \lambda_1 M)C^{-1},$$

or, after a similarity transformation,

$$C^{-1}(\tilde{A} - \lambda_1\tilde{M})C = (C^*C)^{-1}(A - \lambda_1 M).$$

How should we choose C to satisfy (13.33)? Let us tentatively set $C^*C = A$. Then we have

$$(C^*C)^{-1}(A - \lambda_1 M)\mathbf{u}_j = A^{-1}(A - \lambda_1 M)\mathbf{u}_j = (I - \lambda_1 A^{-1}M)\mathbf{u}_j = \left(1 - \frac{\lambda_1}{\lambda_j}\right)\mathbf{u}_j.$$

Note that

$$0 \le 1 - \frac{\lambda_1}{\lambda_j} < 1.$$

Dividing the largest eigenvalue of $A^{-1}(A-\lambda_1 M)$ by the smallest *positive* gives the condition number

$$\kappa_1 := \kappa\left(A^{-1}(A-\lambda_1 M)\big|_{\mathcal{R}(\mathbf{u}_1)^{\perp_M}}\right) = \frac{1-\frac{\lambda_1}{\lambda_n}}{1-\frac{\lambda_1}{\lambda_2}} = \frac{\lambda_2}{\lambda_n}\frac{\lambda_n-\lambda_1}{\lambda_2-\lambda_1} = \frac{\lambda_2}{\lambda_n}\kappa_0. \tag{13.35}$$

If $\lambda_2 \ll \lambda_n$ then the condition number is much reduced. Further, κ_1 is bounded independently of n,

$$\kappa_1 = \frac{1-\lambda_1/\lambda_n}{1-\lambda_1/\lambda_2} < \frac{1}{1-\lambda_1/\lambda_2}. \tag{13.36}$$

So, with this particular preconditioner, κ_1 does not dependent on the choice of the meshwidth h in the FEM application.

The previous discussion suggests to choose C in such way that $C^*C \cong A$. C could, for instance, be obtained form an Incomplete Cholesky decomposition. We make this choice in the numerical example below.

Notice that the transformation $\mathbf{x} \longrightarrow \mathbf{y} = C\mathbf{x}$ need not be made explicitly. In particular, the matrices $\tilde{A}$ and $\tilde{M}$ must not be formed. As with the preconditioned conjugate gradient algorithm for linear systems there is an additional step in the algorithm where the *preconditioned residual* is computed, see Fig. 13.1 on page 248.

13.4 Locally optimal PCG (LOPCG)

The parameter δ_k in the RQMIN und (P)CG algorithms is determined such that

$$\rho(\mathbf{x}_{k+1}) = \rho(\mathbf{x}_k + \delta_k \mathbf{p}_k), \quad \mathbf{p}_k = -\mathbf{g}_k + \alpha_k \mathbf{p}_{k-1} \tag{13.37}$$

is minimized. α_k is chosen to make consecutive search directions conjugate. Knyazev [6] proposed to optimize both parameters, α_k and δ_k, at once.

$$\rho(\mathbf{x}_{k+1}) = \min_{\delta,\gamma} \rho(\mathbf{x}_k - \delta\mathbf{g}_k + \gamma\mathbf{p}_{k-1}) \tag{13.38}$$

This results in potentially smaller values for the Rayleigh quotient, as

$$\min_{\delta,\gamma} \rho\big(\mathbf{x}_k - \delta\mathbf{g}_k + \gamma\mathbf{p}_{k-1}\big) \le \min_{\delta}\big(\mathbf{x}_k - \delta(\mathbf{g}_k - \alpha_k\mathbf{p}_k)\big).$$

Hence, Knyazev coined the notion "locally optimal".

$\rho(\mathbf{x}_{k+1})$ in (13.38) is the minimal eigenvalue of the 3×3 eigenvalue problem

$$\begin{bmatrix}\mathbf{x}_k^*\\ -\mathbf{g}_k^*\\ \mathbf{p}_{k-1}^*\end{bmatrix} A[\mathbf{x}_k, -\mathbf{g}_k, \mathbf{p}_{k-1}]\begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} = \lambda \begin{bmatrix}\mathbf{x}_k^*\\ -\mathbf{g}_k^*\\ \mathbf{p}_{k-1}^*\end{bmatrix} M[\mathbf{x}_k, -\mathbf{g}_k, \mathbf{p}_{k-1}]\begin{pmatrix}\alpha\\ \beta\\ \gamma\end{pmatrix} \tag{13.39}$$

We normalize the eigenvector corresponding to the smallest eigenvalue such that its first component becomes 1,

$$[1, \delta_k, \gamma_k] := [1, \beta/\alpha, \gamma/\alpha].$$

These values of δ_k and γ_k are the parameters that minimize the right hand side in (13.38). Then we can write

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \delta_k\mathbf{g}_k + \gamma_k\mathbf{p}_{k-1} = \mathbf{x}_k + \delta_k \underbrace{\left(-\mathbf{g}_k + (\gamma_k/\delta_k)\mathbf{p}_{k-1}\right)}_{=:\mathbf{p}_k} = \mathbf{x}_k + \delta_k\mathbf{p}_k. \tag{13.40}$$

We can consider $\mathbf{x}_{k+1}$ as having been obtained by a Rayleigh quotient minimization from $\mathbf{x}_k$ along $\mathbf{p}_k = -\mathbf{g}_k + (\gamma_k/\delta_k)\mathbf{p}_{k-1}$. Notice that this direction is needed in the next iteration step. (Otherwise it is not of a particular interest.)

```
function [x,rho,log] = rqmin1(A,M,x,tol,C)
%RQMIN1    [x,rho] = rqmin1(A,M,x0,tol,C)
%       cg-Rayleigh quotient minimization for the computation
%       of the smallest eigenvalue of  A*x = lambda*M*x,
%       A and M are symmetric, M spd.  x0 initial vector
%       C'*C preconditioner
%       tol: convergence criterium:
%            ||2*(C'*C)\(A*x - lam*M*x)|| < tol

% PA 16.6.2000

u = M*x;
q = sqrt(x'*u);
x = x/q; u = u/q;
v = A*x;
rho = x'*v;

k = 0; g = x; gnorm = 1;  log=[]; % Initializations

while gnorm > tol,
  k = k + 1;
  galt = g;
  if exist('C'),
    g = 2*(C\(C'\(v - rho*u)));   %  preconditioned gradient
  else
    g = 2*(v - rho*u);      %  gradient
  end
  if k == 1,
    p = -g;
  else
    p = -g + (g'*M*g)/(galt'*M*galt)*p;
  end

  [qq,ll] = eig([x p]'*[v A*p],[x p]'*[u M*p]);
  [rho,ii] = min(diag(ll));
  delta = qq(2,ii)/qq(1,ii);

  x = x + delta*p;
  u = M*x;
  q = sqrt(x'*u);
  x = x/q; u = u/q;
  v = A*x;
  gnorm = norm(g);
  if nargout>2, log = [log; [k,rho,gnorm]]; end
end
```

Figure 13.1: Matlab code RQMIN: Rayleigh quotient minimization

```
function [x,rho,log] = lopcg(A,M,x,tol,C)
%RQMIN1 [x,rho] = lopcg(A,M,x0,tol,C)
%       Locally Optimal Proconditioned CG algorithm for
%       computing the smallest eigenvalue of  A*x = lambda*M*x,f
%       where A and M are symmetrisch, M spd.
%       x0 initial vektor
%       C'*C preconditioner
%       tol: stopping criterion:
%            (C'*C)\(A*x - lam*M*x) < tol

% PA 2002-07-3

n = size(M,1);
u = M*x;
q = sqrt(x'*u);
x = x/q; u = u/q;
v = A*x;
rho = x'*v;

k = 0;  gnorm = 1;  log=[]; % initializations

while gnorm > tol,
  k = k + 1;
  g = v - rho*u;        %  gradient
  gnorm = norm(g);
  if exist('C'),
    g = (C\(C'\g));   %  preconditioned gradient
  end
  if k == 1, p = zeros(n,0); end

  aa = [x -g p]'*[v A*[-g p]]; aa = (aa+aa')/2;
  mm = [x -g p]'*[u M*[-g p]]; mm = (mm+mm')/2;
  [qq,ll] = eig(aa,mm);
  [rho,ii] = min(diag(ll));
  delta = qq(:,ii);

  p = [-g p]*delta(2:end);
  x = delta(1)*x + p;
  u = M*x;
  q = sqrt(x'*u);
  x = x/q; u = u/q;
  v = A*x;
  if nargout>2, log = [log; [k,rho,gnorm]]; end
end
```

Figure 13.2: MATLAB code LOPCG: Locally Optimal Preconditioned Conjugate Gradient algorithm

13.5 The block Rayleigh quotient minimization algorithm (BRQMIN)

The above procedures converge *very* slowly if the eigenvalues are clustered. Hence, these methods should be applied only in **blocked** form.

Longsine and McCormick [8] suggested several variants for blocking Algorithm 13.1. See [1] for a recent numerical investigation of this algorithm.

13.6 The locally-optimal block preconditioned conjugate gradient method (LOBPCG)

In BRQMIN the Rayleigh quotient is minimized in the $2q$-dimensional subspace generated by the eigenvector approximations X_k and the search directions $P_k = -H_k + P_{k-1}B_k$, where the H_k are the preconditioned residuals corresponding to X_k and B_k is chosen such that the *block* of search directions is conjugate. Instead, Knyazev [6] suggests that the space for the minimization be augmented by the q-dimensional subspace $\mathcal{R}(H_k)$. The resulting algorithm is deemed 'locally-optimal' because $\rho(\mathbf{x})$ is minimized with respect to all available vectors.

Algorithm 13.2 The locally-optimal block preconditioned conjugate gradient method (LOBPCG) for solving $A\mathbf{x} = \lambda M\mathbf{x}$ with preconditioner N of [1]

```
Choose random matrix $X_0 \in \mathbb{R}^{n\times q}$ with $X_0^T M X_0 = I_q$. Set $Q := [\,]$.
Compute $(X_0^T A X_0)S_0 = S_0\Theta_0$          /* (Spectral decomposition) */
where $S_0^T S_0 = I_q$,   $\Theta_0 = \operatorname{diag}(\vartheta_1, \ldots, \vartheta_q)$,   $\vartheta_1 \le \ldots \le \vartheta_q$.
$X_0 := X_0 S_0$;   $R_0 := AX_0 - MX_0\Theta_0$;   $P_0 := [\,]$;   $k := 0$.
while $\operatorname{rank}(Q) < p$ do
   Solve the preconditioned linear system $NH_k = R_k$
   $H_k := H_k - Q(Q^T M H_k)$.
   $\widetilde{A} := [X_k, H_k, P_k]^T A [X_k, H_k, P_k]$.
   $\widetilde{M} := [X_k, H_k, P_k]^T M [X_k, H_k, P_k]$.
   Compute $\widetilde{A}\widetilde{S}_k = \widetilde{M}\widetilde{S}_k\widetilde{\Theta}_k$          /* (Spectral decomposition) */
   where $\widetilde{S}_k^T \widetilde{M}\widetilde{S}_k = I_{3q}$, $\widetilde{\Theta}_k = \operatorname{diag}(\vartheta_1, \ldots, \vartheta_{3q})$, $\vartheta_1 \le \ldots \le \vartheta_{3q}$.
   $S_k := \widetilde{S}_k[\mathbf{e}_1, \ldots, \mathbf{e}_q]$, $\Theta := \operatorname{diag}(\vartheta_1, \ldots, \vartheta_q)$.
   $P_{k+1} := [H_k, P_k]\, S_{k,2}$; $X_{k+1} := X_k S_{k,1} + P_{k+1}$.
   $R_{k+1} := AX_{k+1} - MX_{k+1}\Theta_k$.
   $k := k + 1$.
   for $i = 1, \ldots, q$ do
      /* (Convergence test) */
      if $\|R_k \mathbf{e}_i\| < \mathrm{tol}$ then
         $Q := [Q, X_k\mathbf{e}_i]$;   $X_k\mathbf{e}_i := \mathbf{t}$,  with $\mathbf{t}$ a random vector.
         $M$-orthonormalize the columns of $X_k$.
      end if
   end for
end while
```

If $\mathbf{d}_j = [\mathbf{d}_{1j}^T, \mathbf{d}_{2j}^T, \mathbf{d}_{3j}^T]^T$, $\mathbf{d}_{ij} \in \mathbb{R}^q$, is the eigenvector corresponding to the j-th eigenvalue of (13.1) restricted to $\mathcal{R}([X_k, H_k, P_{k-1}])$, then the j-th column of X_{k+1} is the corresponding

Ritz vector

$$X_{k+1}\mathbf{e}_j := [X_k, H_k, P_{k-1}]\ \mathbf{d}_j = X_k\mathbf{d}_{1j} + P_k\mathbf{e}_j, \tag{13.41}$$

with

$$P_k\mathbf{e}_j := H_k\mathbf{d}_{2j} + P_{k-1}\mathbf{d}_{3j}.$$

Notice that P_0 is an empty matrix such that the eigenvalue problem in step (8) of the locally-optimal block preconditioned conjugate gradient method (LOBPCG), displayed in Algorithm 13.2, has order $2q$ only for $k = 0$.

The algorithm as proposed by Knyazev [6] was designed to compute just a few eigenpairs and so a memory efficient implementation was not presented. For instance, in addition to X_k, R_k, H_k, P_k, the matrices MX_k, MH_k, MP_k and AX_k, AH_k, AP_k are also stored. The resulting storage needed is prohibitive if more than a handful of eigenpairs are needed.

A more memory efficient implementation results when we iterate with blocks of width q in the space orthogonal to the already computed eigenvectors. The computed eigenvectors are stored in Q and neither MQ nor AQ are stored. Hence only storage for $(p + 10q)n + \mathcal{O}(q^2)$ numbers is needed.

Here, the columns of $[X_k, H_k, P_k]$ may become (almost) linearly dependent leading to ill-conditioned matrices $\widetilde{A}$ and $\widetilde{M}$ in step (9) of the LOBPCG algorithm. If this is the case we simply restart the iteration with random X_k orthogonal to the computed eigenvector approximations. More sophisticated restarting procedures that retain X_k but modify H_k and/or P_k were much less stable in the sense that the search space basis again became linearly dependent within a few iterations. Restarting with random X_k is a rare occurrence and in our experience, has little effect on the overall performance of the algorithm.

13.7 A numerical example

We again look at the determination the acoustic eigenfrequencies and modes in the interior of a car, see section 1.6.3. The computations are done with the finest grid depicted in Fig. 1.9. We compute the smallest eigenvalue of the problem with RQMIN and LOPCG, with preconditioning and without. The preconditioner we chose was the incomplete Cholesky factorization without fill-in, usually denoted IC(0). This factorization is implemented in the MATLAB routine `ichol`.

```
>> [p,e,t]=initmesh('auto');
>> [p,e,t]=refinemesh('auto',p,e,t);
>> [p,e,t]=refinemesh('auto',p,e,t);
>> p=jigglemesh(p,e,t);
>> [A,M]=assema(p,t,1,1,0);
>> whos
  Name      Size                    Bytes  Class

  A      1095x1095                  91540  double array (sparse)
  M      1095x1095                  91780  double array (sparse)
  e         7x188                   10528  double array
  p         2x1095                  17520  double array
  t         4x2000                  64000  double array

Grand total is 26052 elements using 275368 bytes

>> n=size(A,1);
>> R=ichol(A)';   % Incomplete Cholesky factorization
>> x0=rand(n,1)-.5;
>> tol=1e-6;
>> [x,rho,log0] = rqmin1(A,M,x0,tol);
>> [x,rho,log1] = rqmin1(A,M,x0,tol,R);
>> [x,rho,log2] = lopcg(A,M,x0,tol);
>> [x,rho,log3] = lopcg(A,M,x0,tol,R);
>> whos log*
  Name       Size                    Bytes  Class

  log0     346x3                      8304  double array
  log1     114x3                      2736  double array
  log2     879x3                     21096  double array
  log3     111x3                      2664  double array

Grand total is 4350 elements using 34800 bytes

>> L = sort(eig(full(A),full(M)));
>> format short e, [L(1) L(2) L(n)], format

ans =

  -7.5901e-13   1.2690e-02   2.6223e+02

>> k0= L(n)/L(2);
>> (sqrt(k0) - 1)/(sqrt(k0) + 1)

ans =

    0.9862

>> l0=log0(end-6:end-1,2).\log0(end-5:end,2);
>> l1=log1(end-6:end-1,2).\log1(end-5:end,2);
>> l2=log2(end-6:end-1,2).\log2(end-5:end,2);
>> l3=log3(end-6:end-1,2).\log3(end-5:end,2);
>> [l0 l1 l2 l3]

ans =

    0.9292    0.8271    0.9833    0.8046
    0.9302    0.7515    0.9833    0.7140
    0.9314    0.7902    0.9837    0.7146
    0.9323    0.7960    0.9845    0.7867
    0.9320    0.8155    0.9845    0.8101
    0.9301    0.7955    0.9852    0.8508

>> semilogy(log0(:,1),log0(:,3)/log0(1,3),log1(:,1),log1(:,3)/log1(1,3),...
   log2(:,1),log2(:,3)/log2(1,3),log3(:,1),log3(:,3)/log3(1,3),'LineWidth',2)
>> legend('rqmin','rqmin + prec','lopcg','lopcg + prec')
```

The convergence histories in Figure 13.3 for RQMIN and LOPCG show that preconditioning helps very much in reducing the iteration count.

In Figure 13.4 the convergence histories of LOBPCG for computing ten eigenvalues is shown. In 43 iteration steps all *ten* eigenvalues have converged to the desired accuracy ($\varepsilon = 10^{-5}$). Clearly, the iteration count has been decreased drastically. Note however, that each

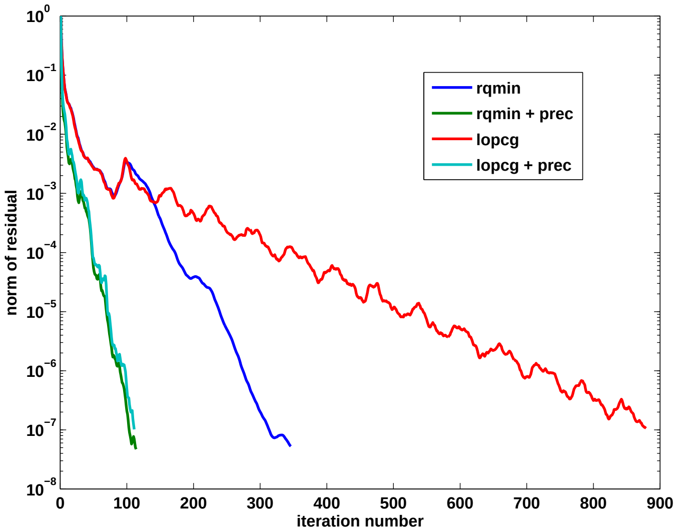


Figure 13.3: Convergence of variants of Rayleigh quotient minimization

iteration step requires solving ten systems of equation resulting in 430 system solves. (In fact, if converged eigenvectors are locked, only 283 systems had to be solved.) Nevertheless, when comparing with Fig. 13.3 one should remember that in the LOBPCG computation ten eigenpairs have been computed. If a single eigenpair is required then a blocksize of 10 is too big, but a smaller blocksize may reduce the execution time. If a small number of eigenvalues is desired then a blocksize equal or slightly bigger than theis number is certainly advantageous. Not that in step (5) of Algorithm 13.2 q linear systems of equations are solved concurrently. An efficient implementation accesses the preconditioner N only once. The MATLAB code does this naturally. A parallel implementation of LOBPCG can be found in the software package Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) [7].

13.8 Trace minimization

Remember that the **trace** of a matrix $A \in \mathbb{F}^{n\times n}$ is defined to be the sum of its diagonal elements. Similar matrices have equal trace. Hence, by the spectral theorem 2.14, we have

$$\text{trace}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i. \tag{13.42}$$

The following theorem [11] generalizes the trace theorem 2.33 for the generalized eigenvalue problem

$$A\mathbf{x} = \lambda M\mathbf{x}, \qquad A = A^*, \quad M = M^* > 0. \tag{13.1}$$

Theorem 13.1 ***(Trace theorem for the generalized eigenvalue problem)*** *Let A and M be as in (13.1). Then,*

$$\lambda_1 + \lambda_2 + \cdots + \lambda_p = \min_{X\in\mathbb{F}^{n\times p},\ X^*MX=I_p} \mathit{trace}(X^*AX) \tag{13.43}$$

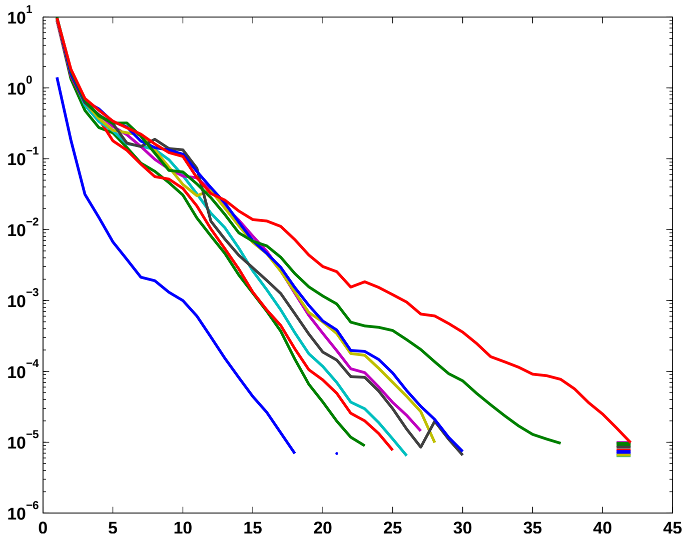


Figure 13.4: Convergence of 10 eigenvalues with LOBPCG preconditioned by IC(0)

where $\lambda_1, \ldots, \lambda_n$ *are the eigenvalues of problem* (13.1)*. Equality holds in* (13.43) *if and only if the columns of the matrix* X *that achieves the minimum span the eigenspace corresponding to the smallest* p *eigenvalues.*

Sameh and coworkers [11, 10] suggested an algorithm to exploit this property of the trace, following the lines of Rayleigh quotient minimization. Let $X_k \in \mathbb{F}^{n \times p}$ with $X_k^* M X_k = I_p$ and

$$X_k^* A X_k = \Sigma_k = \operatorname{diag}(\sigma_1^{(k)}, \ldots, \sigma_p^{(k)}).$$

We want to construct the next iterate X_{k+1} by setting

$$X_{k+1} = (X_k - \Delta_k) S_k \tag{13.44}$$

such that

$$X_{k+1}^* M X_{k+1} = I_p, \tag{13.45}$$

$$X_{k+1}^* A X_{k+1} = \Sigma_{k+1} = \operatorname{diag}(\sigma_1^{(k+1)}, \ldots, \sigma_p^{(k+1)}), \tag{13.46}$$

$$\operatorname{trace}(X_{k+1}^* A X_{k+1}) < \operatorname{trace}(X_k^* A X_k). \tag{13.47}$$

S_k in (13.44) is needed to enforce the orthogonality of X_{k+1}. We choose the correction Δ_k to be orthogonal to X_k,

$$\Delta_k^* M X_k = 0. \tag{13.48}$$

Similarly as in Jacobi–Davidson [13] this choice of Δ_k is no loss of generality. We first assume that we have found the Δ_k that minimizes

$$\operatorname{trace}((X_k - \Delta_k)^* A (X_k - \Delta_k)) \tag{13.49}$$

under the constraint (13.48). Let $Z_{k+1} = X_k - \Delta_k$ be the solution of (13.49). Then, by construction,

$$\operatorname{trace}(Z_{k+1}^* A Z_{k+1}) \leq \operatorname{trace}(X_k^* A X_k).$$

Furthermore,

$$Z_{k+1}^* M Z_{k+1} \overset{(13.48)}{=} X_k^* M X_k + \Delta_k^* M \Delta_k \geq X_k^* M X_k = I_p.$$

From this it follows that Z_{k+1} has maximal rank and that all eigenvalues of $Z_{k+1}^* M Z_{k+1}$ are ≥ 1. Therefore, the spectral decomposition of $Z_{k+1}^* M Z_{k+1}$ can be written in the form

$$Z_{k+1}^* M Z_{k+1} = U D^2 U^*, \qquad U^* U = I_p, \quad D = \operatorname{diag}(\delta_1, \ldots, \delta_p), \quad \delta_i \geq 1.$$

This implies that the columns of $Z_{k+1} U D^{-1}$ are M-orthogonal. Let the spectral decomposition of $D^{-1} U^* Z_{k+1}^* A Z_{k+1} U D^{-1}$ be given by

$$D^{-1} U^* Z_{k+1}^* A Z_{k+1} U D^{-1} = V \Sigma_{k+1} V^*, \qquad V^* V = I_p.$$

Then,

$$V^* \underbrace{D^{-1} U^* Z_{k+1}^* M Z_{k+1} U D^{-1}}_{I_p} V = I_p, \tag{13.50}$$

$$V^* D^{-1} U^* Z_{k+1}^* A Z_{k+1} U D^{-1} V = \Sigma_{k+1}. \tag{13.51}$$

So, if we have found $Z_{k+1} = X_k - \Delta_k$ then S_k in (13.44) is given by

$$S_k = U D^{-1} V.$$

Thus, with $X_{k+1} = Z_{k+1} S_k$ we have

$$\begin{aligned}
\operatorname{trace}(X_{k+1}^* A X_{k_1}) = \operatorname{trace}(\Sigma_{k+1}) &= \operatorname{trace}(V^* D^{-1} U^* Z_{k+1}^* A Z_{k+1} U D^{-1} V) \\
&= \operatorname{trace}(D^{-1} \underbrace{U^* Z_{k+1}^* A Z_{k+1} U}_{W} D^{-1}) \\
&= \sum_{i=1}^{p} w_{ii}/\delta_i^2 \\
&\leq \sum_{i=1}^{p} w_{ii} \\
&= \operatorname{trace}(U^* Z_{k+1}^* A Z_{k+1} U) \\
&\leq \operatorname{trace}(X_k^* A X_k)
\end{aligned}$$

Equality can only hold if $\Delta_k = 0$.

To solve the minimization problem (13.49) we write

$$\begin{aligned}
\operatorname{trace}((X_k - \Delta_k)^* A (X_k - \Delta_k)) &= \sum_{i=1}^{p} \mathbf{e}_i^* (X_k - \Delta_k)^* A (X_k - \Delta_k) \mathbf{e}_i \\
&= \sum_{i=1}^{p} (\mathbf{x}_i - \mathbf{d}_i)^* A (\mathbf{x}_i - \mathbf{d}_i)
\end{aligned}$$

where $\mathbf{x}_i = X_k \mathbf{e}_i$ and $\mathbf{d}_i = \Delta_k \mathbf{e}_i$. Interestingly, these are p individual minimization problems, one for each $\mathbf{d}_i$!

$$\text{Minimize} \quad (\mathbf{x}_i - \mathbf{d}_i)^* A (\mathbf{x}_i - \mathbf{d}_i) \quad \text{subject to } X_k^* M \mathbf{d}_i = \mathbf{0}, \quad i = 1, \ldots, p. \tag{13.52}$$

To solve (13.52) we define the functional

$$f(\mathbf{d}, \mathbf{l}) := (\mathbf{x}_i - \mathbf{d})^* A(\mathbf{x}_i - \mathbf{d}) + \mathbf{l}^* X_k^* M\mathbf{d}.$$

Here, the vector $\mathbf{l}$ contains the Lagrange multipliers. A necessary condition for $\mathbf{d}$ to be a solution of (13.52) is

$$\begin{aligned} \partial_{\mathbf{d}} f = \mathbf{0} \quad &\Longleftrightarrow \quad A(\mathbf{x}_i - \mathbf{d}) + MX_k\mathbf{l} = \mathbf{0}, \\ \partial_{\mathbf{l}} f = \mathbf{0} \quad &\Longleftrightarrow \quad X_k^* M\mathbf{d} = \mathbf{0}. \end{aligned}$$

In matrix form this is

$$\begin{bmatrix} A & MX_k \\ X_k^*M & O \end{bmatrix} \begin{pmatrix} \mathbf{d} \\ \mathbf{l} \end{pmatrix} = \begin{pmatrix} A\mathbf{x}_i \\ 0 \end{pmatrix}.$$

We can collect all p equations in one,

$$\begin{bmatrix} A & MX_k \\ X_k^*M & O \end{bmatrix} \begin{pmatrix} \Delta_k \\ L \end{pmatrix} = \begin{pmatrix} AX_k \\ O \end{pmatrix}. \tag{13.53}$$

Using the LU factorization

$$\begin{bmatrix} A & MX_k \\ X_k^*M & O \end{bmatrix} = \begin{bmatrix} I & 0 \\ X_k^*MA^{-1} & I \end{bmatrix} \begin{bmatrix} A & MX_k \\ O & -X_k^*MA^{-1}MX_k \end{bmatrix}$$

we obtain

$$\begin{bmatrix} A & MX_k \\ O & -X_k^*MA^{-1}MX_k \end{bmatrix} \begin{pmatrix} \Delta_k \\ L \end{pmatrix} = \begin{bmatrix} I & 0 \\ -X_k^*MA^{-1} & I \end{bmatrix} \begin{pmatrix} AX_k \\ O \end{pmatrix} = \begin{pmatrix} AX_k \\ -X_k^*MX_k \end{pmatrix}.$$

Since $X_k^*MX_k = I_p$, L in (13.53) becomes

$$L = (X_k^*MA^{-1}MX_k)^{-1}.$$

Multiplying the first equation in (13.53) by A^{-1} we get

$$\Delta_k + A^{-1}MX_kL = X_k,$$

such that

$$Z_{k+1} = X_k - \Delta_k = A^{-1}MX_kL = A^{-1}MX_k(X_k^*MA^{-1}MX_k)^{-1}.$$

Thus, one step of the above trace minimization algorithm amounts to one step of subspace iteration with shift $\sigma = 0$. This proves convergence of the algorithm to the smallest eigenvalues of (13.1). Remember the similar equation (12.13) for the Jacobi–Davidson iteration and Remark 12.2.

Let P be the orthogonal projection onto $\mathcal{R}(MX_k)^\perp$,

$$P = I - (MX_k)((MX_k)^*(MX_k))^{-1}(MX_k)^* = I - MX_k(X_k^*M^2X_k)^{-1}X_k^*M. \tag{13.54}$$

Then the linear systems of equations (13.53) and

$$PAP\Delta_k = PAX_k, \qquad X_k^*M\Delta_k = 0, \tag{13.55}$$

are equivalent, i.e., they have the same solution Δ_k. In fact, let $\begin{pmatrix} \Delta_k \\ L \end{pmatrix}$ be the solution of (13.53). Then, from $X_kM\Delta_k = 0$ we get $P\Delta_k = \Delta_k$. Equation (13.55) is now obtained

Algorithm 13.3 Trace minimization algorithm to compute p eigenpairs of $A\mathbf{x} = \lambda M\mathbf{x}$.

```
Choose random matrix $V_1 \in \mathbb{R}^{n\times q}$ with $V_1^T M V_1 = I_q$, $q \geq p$.
for $k = 1, 2, \ldots$ until convergence do
    Compute $W_k = AV_k$ and $H_k := V_k^* W_k$.
    Compute spectral decomposition $H_k = U_k \Theta_k U_k^*$,
    with $\Theta_k = \operatorname{diag}(\vartheta_1^{(k)}, \ldots, \vartheta_q^{(k)})$, $\vartheta_1^{(k)} \leq \ldots \leq \vartheta_q^{(k)}$.
    Compute Ritz vectors $X_k = V_k U_k$ and residuals $R_k = W_k U_k - M X_k \Theta_k$
    For $i = 1, \ldots, q$ solve approximatively
        $P(A - \sigma_i^{(k)} M) P \mathbf{d}_i^{(k)} = P\mathbf{r}_i, \qquad \mathbf{d}_i^{(k)} \perp_M X_k$
    by some modified PCG solver.
    Compute $V_{k+1} = [X_k - \Delta_k, R_k]$, $\Delta_k = \mathbf{d}_1^{(k)}, \ldots, \mathbf{d}_q^{(k)}]$, by a $M$-orthogonal modified
    Gram-Schmidt procedure.
end for
```

by multiplying the first equation in (13.53) by P. On the other hand, let Δ_k be the solution of (13.55). Since $P(AP\Delta_k - AX_k) = 0$ we must have $AP\Delta_k - AX_k = MX_k L$ for some L. As $X_k M \Delta_k = 0$ we get $P\Delta_k = \Delta_k$ and thus the first equation in (13.53).

As PAP is positive semidefinite, equation (13.55) is easier to solve than equation (13.53) which is an indefinite system of equations. (13.55) can be solved by the (preconditioned) conjugate gradient method (PCG). The iteration has to be started by a vector $\mathbf{z}_0$ that satisfies the constrains $X_k^* M \mathbf{z}_0$. A straightforward choice is $\mathbf{z}_0 = \mathbf{0}$

reducion factor	10^{-4}		10^{-2}		0.5	
	#its	A mults	#its	A mults	#its	A mults
	59	6638	59	4263	77	4030

Table 13.1: The basic trace minimization algorithm (Algorithm 13.3). The inner systems are solved by the CG scheme which is terminated such that the 2-norm of the residual is reduced by a specified factor. The number of outer iterations (#its) and the number of multiplications with matrix A (A mults) are listed for different residual reduction factors.

In practice, we do not solve the p linear systems

$$P(A - \sigma_i^{(k)} M) P \mathbf{d}_i^{(k)} = P\mathbf{r}_i, \qquad \mathbf{d}_i^{(k)} \perp_M X_k \tag{13.56}$$

to high accuracy. In Table 13.1 the number of outer iterations (#its) are given and the number of multiplications of the matrix A with a vector for various relative stopping criteria for the inner iteration (reduction factor) [10].

Acceleration techniques

Sameh & Tong [10] investigate a number of ways to accelerate the convergence of the trace minimization algorithm 13.3.

1. *Simple shifts.* Choose a shift $\sigma_1 \leq \lambda_1$ until the first eigenpair is found. Then proceed with the shift $\sigma_2 \leq \lambda_2$ and *lock* the first eigenvector. In this way PCG can be used to solve the linear systems as before.

2. *Multiple dynamic shifts.* Each linear system (13.56) is solved with an individual shift. The shift is ‘turned on’ close to convergence. Since the linear systems are indefinite, PCG has to be adapted.

3. *Preconditioning.* The linear systems (13.56) can be preconditioned, e.g., by a matrix of the form $M = CC^*$ where $CC^* \approx A$ is an incomplete Cholesky factorization. One then solves

$$\tilde{P}(\tilde{A} - \sigma_i^{(k)}\tilde{M})\tilde{P}\tilde{\mathbf{d}}_i^{(k)} = \tilde{P}\tilde{\mathbf{r}}_i, \qquad \tilde{X}_k^*\tilde{M}\tilde{\mathbf{d}}_i^{(k)} = \mathbf{0} \tag{13.57}$$

with $\tilde{A} = C^{-1}AC^{-*}, \tilde{M} = C^{-1}MC^{-*}, \tilde{\mathbf{d}}_i^{(k)} = C^*\mathbf{d}_i^{(k)}, \tilde{X}_k = C^*X_k, \tilde{\mathbf{x}}_i^{(k)} = C^*\mathbf{x}_i^{(k)}$, and $\tilde{P} = I - \tilde{M}\tilde{X}_k(\tilde{X}_k^*\tilde{M}^2\tilde{X}_k)^{-1}\tilde{X}_k^*\tilde{M}$.

In Table 13.2 results are collected for some problems in the Harwell–Boeing collection [10]. These problems are diffcult because the gap ratios for the smallest eigenvalues are extremely small due to the huge span of the spectra. Without preconditioning, none of these problems can be solved with a reasonable cost. In the experiments, the incomplete Cholesky factorization (IC(0)) of A was used as the preconditioner for all the matrices of the form $A - \sigma B$.

Problem	Size	Max #	Block Jacobi–Davidson			Davidson-type tracemin		
		inner its	#its	A mults	time[sec]	#its	A mults	time[sec]
BCSST08	1074	40	34	3954	4.7	10	759	0.8
BCSST09	1083	40	15	1951	2.2	15	1947	2.2
BCSST11	1473	100	90	30990	40.5	54	20166	22.4
BCSST21	3600	100	40	10712	35.1	39	11220	36.2
BCSST26	1922	100	60	21915	32.2	39	14102	19.6

Table 13.2: Numerical results for problems from the Harwell–Boeing collection with four processors (reproduced from [10])

The Davidson-type trace minimization algorithm with multiple dynamic shifts works better than the block Jacobi–Davidson algorithm for three of the five problems. For the other two, the performance for both algorithms is similar.

Bibliography

[1] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. Tuminaro, *A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods*, Internat. J. Numer. Methods Eng., 64 (2005), pp. 204–236.

[2] O. Axelsson and V. Barker, *Finite Element Solution of Boundary Value Problems*, Academic Press, Orlando FL, 1984.

[3] D. K. Faddeev and V. N. Faddeeva, *Computational Methods of Linear Algebra*, Freeman, San Francisco, 1963.

[4] Y. T. Feng and D. R. J. Owen, *Conjugate gradient methods for solving the smallest eigenpair of large symmetric eigenvalue problems*, Internat. J. Numer. Methods Eng., 39 (1996), pp. 2209–2229.

[5] M. R. Hestenes and E. Stiefel, *Methods of conjugent gradients for solving linear systems*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 409–436.

[6] A. V. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[7] A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov, *Block locally optimal preconditioned eigenvalue Xolvers (BLOPEX) in hypre and PETSc*, SIAM J. Sci. Comput., 29 (2007), pp. 2224–2239.

[8] D. E. Longsine and S. F. McCormick, *Simultaneous Rayleigh–quotient minimization methods for $Ax = \lambda Bx$*, Linear Algebra Appl., 34 (1980), pp. 195–234.

[9] A. Ruhe, *Computation of eigenvalues and vectors*, in Sparse Matrix Techniques, V. A. Barker, ed., Lecture Notes in Mathematics 572, Berlin, 1977, Springer, pp. 130–184.

[10] A. Sameh and Z. Tong, *The trace minimization method for the symmetric generalized eigenvalue problem*, J. Comput. Appl. Math., 123 (2000), pp. 155–175.

[11] A. H. Sameh and J. A. Wisniewski, *A trace minimization algorithm for the generalized eigenvalue problem*, SIAM J. Numer. Anal., 19 (1982), pp. 1243–1259.

[12] H. R. Schwarz, *Rayleigh–Quotient–Minimierung mit Vorkonditionierung*, in Numerical Methods of Approximation Theory, Vol. 8, L. Collatz, G. Meinardus, and G. Nürnberger, eds., vol. 81 of International Series of Numerical Mathematics (ISNM), Basel, 1987, Birkhäuser, pp. 229–45.

[13] G. L. G. Sleijpen and H. A. van der Vorst, *Jacobi–Davidson method*, in Templates for the solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., SIAM, Philadelphia, PA, 2000, pp. 238–246.