Chapter 3

Newton methods

3.1 Linear and nonlinear eigenvalue problems

When solving linear eigenvalue problems we want 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 want to find values \( \lambda \in \mathbb{C} \) such that there is a nontrivial (nonzero) \( x \) that satisfies

\[
(A - \lambda I)x = 0 \iff Ax = \lambda x.
\]

In the linear eigenvalue problem (3.1) the eigenvalue \( \lambda \) appears linearly. However, as the unknown \( \lambda \) is multiplied with the unknown vector \( x \), 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 \( x \) is not determined. Therefore, we add to (3.1) a further equation that fixes this. The straightforward condition is

\[
\|x\|^2 = x^*x = 1,
\]

that determines \( x \) up to a complex scalar of modulus 1, in the real case \( \pm 1 \). Another condition to normalize \( x \) is by requesting that

\[
c^T x = 1, \quad \text{for some } c.
\]

Eq. (3.3) is linear in \( x \) and thus simpler. However, the combined equations (3.1)–(3.3) are nonlinear anyway. Furthermore, \( c \) 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

\[
A(\lambda)x = 0
\]

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

\[
A(\lambda) = \sum_{k=0}^{d} \lambda^k A_k, \quad A_k \in \mathbb{F}^{n \times n}.
\]

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

\[
A(\lambda) = A_0 - \lambda A_1, \quad A_0 = A, \quad A_1 = I.
\]
QuadraC eigenvalue problems are matrix polynomials where \( \lambda \) appears squared [6],
\begin{equation}
(3.6) \quad Ax + \lambda Kx + \lambda^2 Mx = 0.
\end{equation}
Matrix polynomials can be linearized, i.e., they can be transformed in a linear eigenvalue of size \( dn \) where \( d \) is the degree of the polynomial and \( n \) is the size of the matrix. Thus, the quadratic eigenvalue problem (3.6) can be transformed in a linear eigenvalue problem of size \( 2n \). Setting \( y = \lambda x \) we get
\begin{equation}
\begin{pmatrix} A & O \\ O & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} -K & -M \\ I & O \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\end{equation}
or
\begin{equation}
\begin{pmatrix} A & K \\ O & I \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \lambda \begin{pmatrix} O & -M \\ I & O \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}.
\end{equation}
Notice that 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
\begin{equation}
(3.7) \quad f(\lambda) := \det A(\lambda) = 0
\end{equation}
and apply some zero finder. Here the question arises how to compute \( f(\lambda) \) and in particular \( f'(\lambda) = \frac{d}{d\lambda} \det A(\lambda) \). This is what we discuss next.

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
\begin{equation}
(3.8) \quad P(\lambda)A(\lambda) = L(\lambda)U(\lambda),
\end{equation}
where \( P(\lambda) \) is the permutation matrix due to partial pivoting, \( L(\lambda) \) is a lower \textit{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
\begin{equation}
(3.9) \quad f(\lambda) = \det A(\lambda) = \pm 1 \cdot \prod_{i=1}^{n} u_{ii}(\lambda).
\end{equation}
The derivative of \( \det A(\lambda) \) is
\begin{equation}
(3.10) \quad f'(\lambda) = \pm 1 \cdot \sum_{i=1}^{n} u_{ii}'(\lambda) \prod_{j \neq i}^{n} u_{jj}(\lambda)
\end{equation}
\[ = \pm 1 \cdot \sum_{i=1}^{n} \frac{u_{ii}'(\lambda)}{u_{ii}(\lambda)} \prod_{j=1}^{n} u_{jj}(\lambda) = \sum_{i=1}^{n} \frac{u_{ii}'(\lambda)}{u_{ii}(\lambda)} f(\lambda). \]
How can we compute the derivatives \( u_{ii}' \) of the diagonal elements of \( U(\lambda) \)?
3.2. ZEROS OF THE DETERMINANT

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 \( \lambda \). 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

\[
\begin{align*}
p_{n} & := c_{n}, \\
p_{i} & := z p_{i+1} + c_{i}, \quad i = n - 1, n - 2, \ldots, 0, \\
f(z) & := p_{0}.
\end{align*}
\]

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 \( dp_{i} \),

\[
\begin{align*}
dp_{n} & := 0, \quad p_{n} := c_{n}, \\
dp_{i} & := p_{i+1} + z dp_{i+1}, \quad p_{i} := z p_{i+1} + c_{i}, \quad i = n - 1, n - 2, \ldots, 0, \\
f'(z) & := dp_{0}, \quad f(z) := p_{0}.
\end{align*}
\]

Here, to each equation in the original algorithm its derivation is prepended. We can proceed in a similar fashion for computing \( \det \ A(\lambda) \). We need to be able to compute the derivatives \( a'_{ij} \), though. Then, we can derive each single assignment in the GEPP algorithm.

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

3.2.2 Hyman’s algorithm

In a Newton iteration we have to compute the determinant for possibly many values \( \lambda \). 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,

\[
h_{ij} = 0, \quad i > j + 1.
\]

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

$$Ax = \lambda x \iff Hy = \lambda y, \quad x = Sy.$$  

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 $Hx = \lambda x$ in smaller problems.

Let $\lambda$ be an eigenvalue of $H$ and

$$(3.11) \quad (H - \lambda I)x = 0,$$

i.e., $x$ is an eigenvector of $H$ associated with the eigenvalue $\lambda$. Then the last component of $x$ 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 the $H$ is unreduced, i.e., that $h_{n,n-1} \neq 0$. 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 the problem size $n = 4$. If $\lambda$ is an eigenvalue then there are $x_i$, $1 \leq i < n$, such that

$$(3.12) \quad (h_{11} - \lambda \ h_{12} \ h_{13} \ h_{14})
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  1 \\
\end{pmatrix} =
\begin{pmatrix}
  0 \\
  0 \\
  0 \\
  0 \\
\end{pmatrix}.$$

If $\lambda$ is not an eigenvalue then we determine the $x_i$ such that

$$(3.13) \quad (h_{11} - \lambda \ h_{12} \ h_{13} \ h_{14})
\begin{pmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  1 \\
\end{pmatrix} =
\begin{pmatrix}
  p(\lambda) \\
  0 \\
  0 \\
  0 \\
\end{pmatrix}.$$

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

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

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

$$(3.14) \quad (h_{1,1} - \lambda) x_1 + h_{1,2} x_2 + \cdots + h_{1,n} x_n \equiv p(\lambda).$$
3.2. ZEROS OF THE DETERMINANT

Eq. (3.13) is obtained from looking at the last column of

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

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 \(\lambda\), we can get \(p'(\lambda)\) by algorithmic differentiation.

For \(i = n - 1, \ldots, 1\) we have

\[
x_i' = \frac{1}{h_{i+1,i}} \left( -x_{i+1} + (h_{i+1,i+1} - \lambda) x_{i+1}' + h_{i+1,i+2} x_{i+2}' + \cdots + h_{i+1,n-1} x_{n-1}' \right).
\]

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 3.1.2 implements Hyman’s algorithm that returns \(p(\lambda)\) and \(p'(\lambda)\) given an input parameter \(\lambda\) [7].

**Algorithm 3.1. Hyman’s algorithm**

1. Choose a value \(\lambda\).
2. \(x_n := 1; dx_n := 0;\)
3. **for** \(i = n - 1\) **down to** 1 **do**
4. \(s = (\lambda - h_{i+1,i+1}) x_{i+1}; ds = x_{i+1} + (\lambda - h_{i+1,i+1}) dx_{i+1};\)
5. **for** \(j = i + 2\) **to** \(n\) **do**
6. \(s = s - h_{i+1,j} x_j; ds = ds - h_{i+1,j} dx_j;\)
7. **end** **for**
8. \(x_i = s/h_{i+1,i}; dx_i = ds/h_{i+1,i};\)
9. **end** **for**
10. \(s = -(\lambda - h_{1,1}) x_1; ds = -x_1 - (\lambda - h_{1,1}) dx_1;\)
11. **for** \(i = 2\) **to** \(n\) **do**
12. \(s = s + h_{1,i} x_i; ds = ds + h_{1,i} dx_i;\)
13. **end** **for**
14. \(p(\lambda) := s; p'(\lambda) := ds;\)

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

\[
\lambda_{k+1} = \lambda_k - \frac{p(\lambda_k)}{p'(\lambda_k)}, \quad k = 0, 1, \ldots
\]
CHAPTER 3. NEWTON METHODS

The factor \( c \) cancels. An initial guess \( \lambda_0 \) 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 \( x = (x_1, x_2, \ldots, x_{n-1}, 1)^T \) is a good approximation of the corresponding eigenvector.

**Remark 3.1.** Higher order derivatives 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

\[
(3.15) \quad f_1(x) := \frac{f(x)}{x - z},
\]

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 explicitly.

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

\[
\frac{f'_1(x)}{f_1(x)} = \frac{f'(x)}{x - z} - \frac{f(x)}{(x - z)^2} = \frac{f'(x)}{f(x)} \frac{1}{x - z}.
\]

Then a Newton correction becomes

\[
(3.16) \quad x^{(k+1)} = x_k - \frac{1}{f'_1(x_k) - \frac{1}{x_k - z}}
\]

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),

\[
(3.17) \quad T(\lambda)x = 0, \\
\quad c^T x = 1,
\]

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

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

\[
(3.18) \quad f(x, \lambda) = \begin{pmatrix} T(\lambda)x \\ c^T x - 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.
\]
3.3. NEWTON METHODS FOR THE CONSTRAINED MATRIX PROBLEM

To apply Newton’s zero finding method to (3.18) we need the Jacobian $J$ of $f$,

\[(3.19)\quad J(x, \lambda) = \frac{\partial f(x, \lambda)}{\partial (x, \lambda)} = \begin{pmatrix} T(\lambda) & T'(\lambda)x \\ c^T & 0 \end{pmatrix}.\]

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

\[(3.20)\quad \begin{pmatrix} x_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ \lambda_k \end{pmatrix} - J(x_k, \lambda_k)^{-1} f(x_k, \lambda_k),\]

or, with the abbreviations $T_k := T(\lambda_k)$ and $T'_k := T'(\lambda_k)$, an equation for the Newton corrections is obtained,

\[(3.21)\quad \begin{pmatrix} T_k & T'_k x_k \\ c^T & 0 \end{pmatrix} \begin{pmatrix} x_{k+1} - x_k \\ \lambda_{k+1} - \lambda_k \end{pmatrix} = \begin{pmatrix} -T_k x_k \\ 1 - c^T x_k \end{pmatrix}.\]

If $x_k$ is normalized, $c^T x_k = 1$, then the second equation in (3.21) yields

\[(3.22)\quad c^T (x_{k+1} - x_k) = 0 \iff c^T x_{k+1} = 1.\]

The first equation in (3.21) gives

\[T_k (x_{k+1} - x_k) + (\lambda_{k+1} - \lambda_k) T'_k x_k = -T_k x_k \iff T_k x_{k+1} = -(\lambda_{k+1} - \lambda_k) T'_k x_k.\]

We introduce the auxiliary vector $u_{k+1}$ by

\[(3.23)\quad T_k u_{k+1} = T'_k x_k.\]

Note that

\[(3.24)\quad x_{k+1} = -(\lambda_{k+1} - \lambda_k) u_{k+1}.\]

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

\[1 = c^T x_{k+1} = -(\lambda_{k+1} - \lambda_k) c^T u_{k+1},\]

or

\[(3.25)\quad \lambda_{k+1} = \lambda_k - \frac{1}{c^T u_{k+1}}.\]

In summary, we get the following procedure.

If the linear eigenvalue problem is solved by Algorithm 3.3 then $T'(\lambda)x = x$. 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 $x$. Problem (3.17) becomes

\[(3.26)\quad T(\lambda) x = 0, \quad \|x\|^2 = 1,\]

with the corresponding nonlinear system of equations

\[(3.27)\quad f(x, \lambda) = \begin{pmatrix} T(\lambda) x \\ \frac{1}{2}(x^T x - 1) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.\]
CHAPTER 3. NEWTON METHODS

Algorithm 3.2 Newton iteration for solving (3.18)

1: Choose a starting vector \( x_0 \in \mathbb{R}^n \) with \( c^T x_0 = 1 \). Set \( k := 0 \).
2: \textbf{repeat}
3: \hspace{1em} Solve \( T(\lambda_k) u_{k+1} := T'(\lambda_k) x_k \) for \( u_{k+1} \); \hfill (3.23)
4: \hspace{1em} \( \mu_k := c^T u_{k+1} \); \hfill (Normalize \( u_{k+1} \))
5: \hspace{1em} \( x_{k+1} := u_{k+1}/\mu_k \); \hfill (3.25)
6: \hspace{1em} \( \lambda_{k+1} := \lambda_k - 1/\mu_k \);
7: \hspace{1em} \( k := k + 1 \);
8: \textbf{until} some convergence criterion is satisfied

The Jacobian now is

\[
J(x, \lambda) = \begin{bmatrix} \frac{\partial f(x, \lambda)}{\partial x} & \frac{\partial f(x, \lambda)}{\partial \lambda} \end{bmatrix} = \begin{bmatrix} T(\lambda) & T'(\lambda) x \end{bmatrix}. \tag{3.28}
\]

The Newton step (3.20) is changed into

\[
\begin{bmatrix} T_k & T_k' x_k \\ x_k^T & 0 \end{bmatrix} \begin{bmatrix} x_{k+1} - x_k \\ \lambda_{k+1} - \lambda_k \end{bmatrix} = \begin{bmatrix} -T_k x_k \\ \frac{1}{2} (1 - x_k^T x_k) \end{bmatrix}. \tag{3.29}
\]

If \( x_k \) is normalized, \( \|x_k\| = 1 \), then the second equation in (3.29) gives

\[
x_k^T (x_{k+1} - x_k) = 0 \iff x_k^T x_{k+1} = 1. \tag{3.30}
\]

The correction \( \Delta x_k := x_{k+1} - x_k \) is orthogonal to the actual approximation. The first equation in (3.29) is identical with the first equation in (3.21). Again, we employ the auxiliary vector \( u_{k+1} \) defined in (3.23). Premultiplying (3.24) by \( x_k^T \) and using (3.30) gives

\[
1 = x_k^T x_{k+1} = - (\lambda_{k+1} - \lambda_k) x_k^T u_{k+1},
\]
or

\[
\lambda_{k+1} = \lambda_k - \frac{1}{x_k^T u_{k+1}}. \tag{3.31}
\]

The next iterate \( x_{k+1} \) is obtained by normalizing \( u_{k+1} \),

\[
x_{k+1} = u_{k+1}/\|u_{k+1}\|. \tag{3.32}
\]

Algorithm 3.3 Newton iteration for solving (3.27)

1: Choose a starting vector \( x_0 \in \mathbb{R}^n \) with \( \|x_0\| = 1 \). Set \( k := 0 \).
2: \textbf{repeat}
3: \hspace{1em} Solve \( T(\lambda_k) u_{k+1} := T'(\lambda_k) x_k \) for \( u_{k+1} \); \hfill (3.23)
4: \hspace{1em} \( \mu_k := x_k^T u_{k+1} \);
5: \hspace{1em} \( \lambda_{k+1} := \lambda_k - 1/\mu_k \); \hfill (3.31)
6: \hspace{1em} \( x_{k+1} := u_{k+1}/\|u_{k+1}\| \); \hfill (Normalize \( u_{k+1} \))
7: \hspace{1em} \( k := k + 1 \);
8: \textbf{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 $\lambda_k$.

\begin{equation}
T(\lambda)x \approx (T(\lambda_k) - \vartheta T'(\lambda_k))x = 0, \quad \lambda = \lambda_k - \vartheta.
\end{equation}

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

\begin{equation}
T(\lambda_k)x = \vartheta T'(\lambda_k)x
\end{equation}

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

**Algorithm 3.4** Algorithm of successive linear problems

1. Start with approximation $\lambda_1$ of an eigenvalue of $T(\lambda)$.
2. for $k = 1, 2, \ldots$ do
3. Solve the linear eigenvalue problem $T(\lambda_k)u = \vartheta T'(\lambda_k)u$.
4. Choose an eigenvalue $\vartheta$ smallest in modulus.
5. $\lambda_{k+1} := \lambda_k - \vartheta$;
6. if $\vartheta$ is small enough then
7. Return with eigenvalue $\lambda_{k+1}$ and associated eigenvector $u/\|u\|$.
8. end if
9. end for

**Remark 3.2.** If $T$ is twice continuously differentiable, and $\lambda$ 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 $\lambda$. □

**Bibliography**


