Daniel Leitner

CO-MAT2

Lecture Notes

Wintersemester 2014/15

Computational Science Center
Universität Wien
Oskar-Morgenstern-Platz 1
A-1090 Wien

Contents

Chapter 1

Eigenvalues

In this chapter we will consider a matrix $A \in \mathbb{C}^{n\times n}$ and numerically find its eigenvalues and eigenvectors. We use the more general case of complex matrices because even for real matrices eigenvalues and eigenvectors can be complex.

The section is widely based on the lecture notes [1], and contains additional material material from [2].

1.1 Eigenvalue problems

For a given matrix A we search for pairs of *eigenvalues* $\lambda \in \mathbb{C}$ and *eigenvectors* $x \in \mathbb{C}^n$ that satisfy

$$Ax = \lambda x \quad x \in \mathbb{C}^n \setminus \{0\},\ \lambda \in \mathbb{C}\ .$$

We can find the eigenvalues by the roots λ_i of the *characteristic polynomial*

$$\chi(z) = \det(A - zI) = (z - \lambda_1)(z - \lambda_2)\dots(z - \lambda_n)\ ,$$

which is a non-linear problem. We define the *algebraic multiplicity* of an eigenvalue λ to be its multiplicity as a root of $\chi(z)$.

For each eigenvalue λ we can calculate the corresponding eigenvectors solving the linear equation

$$(A - \lambda I)\, v_j = 0\ .$$

The linear space spanned by the eigenvectors v_j is called the *eigenspace* and is denoted with E_λ. The number of linear independent eigenvectors (i.e. the rank of E_λ) is called the *geometric multiplicity* of the eigenvalue λ.

Example 1.1.1. Consider the two matrices

$$A = \begin{bmatrix} 2 & & \\ & 2 & \\ & & 2 \end{bmatrix},\quad B = \begin{bmatrix} 2 & 1 & \\ & 2 & 1 \\ & & 2 \end{bmatrix}\ .$$

What is the characteristic polynomial, eigenvalues, algebraic and geometric multiplicities?

An eigenvalue whose algebraic multiplicity is greater than its geometric multiplicity is called a *defective eigenvalue*. A matrix with at least one defective eigenvalue is called a *defective matrix*.

We call the matrices A and B *similar* if there exists a non-singular $X \in \mathbb{C}^{n\times n}$ such that $B = XAX^{-1}$. Similar matrices have the same characteristic polynomial, eigenvalues, and algebraic and geometric multiplicities.

Numerical algorithms to find eigenvalues are **not** based on finding roots of the characteristic polynomial, since this is a highly unstable problem. Eigenvalues are either computed by power iteration (see Section 1.1.2) or by eigenvalue revealing factorizations (see Section 1.1.3).

Lemma 1.1.2. *The most important matrix factorizations are:*

1. *If A is nondefective a diagonalisation $A = X\Lambda X^{-1}$ exists, where Λ is a diagonal matrix.*
2. *If A is normal (i.e. $AA^* = A^*A$) an unitary diagonalisation $A = Q\Lambda Q^*$ exists, where Q is an unitary matrix (i.e. $Q^* = Q^{-1}$), and Λ is a diagonal matrix. Note that all hermitian matrices are normal.*
3. *An unitary triangulation (*Schur factorization*) $A = QTQ^*$ always exists, where Q is unitary and T is upper triangular.*

Typical applications of eigenvalues are situations where we are interested in $A^k = X\Lambda^k X^{-1}$ or $e^A = Xe^\Lambda X^{-1}$ for a nondefective matrix A. The second expression is especially useful for analysing systems of linear differential equations.

1.1.1 Estimation of eigenvalues

For a quick estimation of the eigenvalues we can use

Theorem 1.1.3. (Gershgorin circle theorem) *Given a matrix $A = [a_{ij}] \in \mathbb{C}^{n\times n}$, then for every eigenvalue λ of A, holds*

$$\lambda \in \bigcup_{i=1}^{n} \mathcal{K}_i,$$

with the Gershgorin discs

$$\mathcal{K}_i := \{\zeta \in \mathbb{C} : |\zeta - a_{ii}| \leq \sum_{j\neq i} |a_{ij}|\}.$$

Proof. Let $Ax = \lambda x$ with $x = [x_i] \neq 0$. Then there exists an index i, so that $|x_j| \leq |x_i|$ for all $j \neq i$. $(Ax)_i$ denotes the i-th component of Ax, then

$$\lambda x_i = (Ax)_i = \sum_{j=1}^{n} a_{ij}x_j$$

and therefore,

$$|\lambda - a_{ii}| = \left|\sum_{j\neq i} a_{ij}\frac{x_j}{x_i}\right| \leq \sum_{j\neq i} |a_{ij}| \ .$$

Therfore, $\lambda \in \mathcal{K}_i \subseteq \bigcup_{j=1}^{n} \mathcal{K}_j$. □

Example 1.1.4. What are the Gershgorin discs of

$$A = \begin{bmatrix} 4 & 0 & -3 \\ 0 & -1 & 1 \\ -1 & 1 & 0 \end{bmatrix} ?$$

We can further confine the range in which the eigenvalues are using the *Rayleigh quotient*. The Rayleigh quotient of a vector x is given by

$$r(x) := \frac{x^* A x}{x^* x}, \quad x \in \mathbb{C}^n \backslash \{0\}.$$

Note that $r(v_i) = \lambda_i$, and for each vector x the Rayleigh quotient gives the value $\alpha := r(x)$ which acts most like an eigenvalue (i.e. minimizes $\|Ax - \alpha x\|$).

The range of $r(x)$

$$\mathcal{W}(A) := \left\{ \frac{x^* A x}{x^* x} : x \in \mathbb{C}^n \backslash \{0\} \right\} \subseteq \mathbb{C} \, .$$

is called the *numerical range* of the matrix A. Notably, the eigenvalues of A lie in the numerical range of A.

Further properties of $\mathcal{W}(\mathcal{A})$ are

1. $\mathcal{W}(A)$ is connected.
2. If A is hermitian, then $\mathcal{W}(A)$ is the real interval $[\lambda_{\min}, \lambda_{\max}]$.
3. If A is *skew-symmetric* (i.e. $A^* = -A$), then $\mathcal{W}(A)$ is an imaginary interval, i.e. the convex hull ($\subseteq \mathbb{C}$) of all eigenvalues of A.

See [1] for a proof.

Every matrix $A \in \mathbb{C}^{n \times n}$ can be spilt into an hermitian part (first term) and a skew symmetric part (second term):

$$A = \frac{A + A^*}{2} + \frac{A - A^*}{2}.$$

From this split and the above properties we directly derive:

Theorem 1.1.5. (Theorem of Bendixon)

$$\sigma(A) \subseteq \mathcal{W}\left(\frac{A + A^*}{2}\right) \oplus \mathcal{W}\left(\frac{A - A^*}{2}\right),$$

where $\sigma(A)$ is the spectrum of $A \in \mathbb{C}^{n \times n}$ (i.e. the set containing all eigenvalues of A).

Example 1.1.6. We use the theorem to further confine the range of Example 1.1.4. First we calculate the hermitian and skew-symmetric part

$$H = \frac{A + A^t}{2} = \begin{bmatrix} 4 & 0 & -2 \\ 0 & -1 & 1 \\ -2 & 1 & 0 \end{bmatrix}, \quad S = \frac{A - A^t}{2} = \begin{bmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}.$$

The spectra of H and S can be estimated by the theorem of Gerschgorin: This yields

$$\mathcal{R} = [-3, 6] \times [-i, i] \supset \mathcal{W}(A) \supset \sigma(A) \; .$$

The spectrum of A must lie in the intersection of $\mathcal{R}$ and the Gerschgorin discs of Example 1.1.4. The actual spectrum of A is

$$\sigma(A) = \{-1.7878, 0.1198, 4.6679\} \; .$$

1.1.2 Power iteration

In this section we numerically calculate eigenvalues and eigenvectors by a method called *power iteration*. For simplicity, we only consider symmetric real matrices. Such matrices are diagonalisable and their eigenvectors form an orthonormal basis. Furthermore, we sort the absolute values of the eigenvalues in descending orders $|\lambda_1| \geq |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$.

We consider the sequence $z^{(k+1)} = Az^{(k)}/\|Az^{(k)}\|$. Under certain assumptions the sequence converges to the eigenvector corresponding to the largest eigenvalue of A. The approach is outlined in Algorithm 1.

Data: Matrix A, initial eigenvector $z^{(0)}$ with $\|z^{(0)}\| = 1$
Initialisation: set $k = 1$
while *convergence criterion not satisfied* **do**

$\tilde{z}^{(k)} := Az^{(k-1)}$ apply A
$z^{(k)} := \frac{\tilde{z}^{(k)}}{\|\tilde{z}^{(k)}\|}$ normalise
$k \leftarrow k + 1$

end

Algorithm 1: Power iteration

The estimation for the eigenvalue can be found by applying the Rayleigh quotient to the resulting $z^{(k)}$,

$$\lambda^{(k)} := z^{(k)T} A z^{(k)}.$$

We analyse power iteration by writing the initial guess $z^{(0)}$ as a linear combination of the orthonormal eigenvectors q_i:

$$z^{(0)} = a_1 q_1 + a_2 q_2 + \cdots + a_n q_n \tag{1.1}$$

The value $z^{(k)}$ is a multiple of $A^k z^{(0)}$, therefore we write

$$z^{(k)} = c_k A^k z^{(0)} \; ,$$

where c_k is some scalar constant. Note that $Aq_1 = \lambda_1 q_1$, thus inserting Eqn (1.1) yields

$$\begin{aligned} z^{(k)} &= c_k(a_1\lambda_1^k q_1 + a_2\lambda_2^k q_2 \cdots + a_n\lambda_n^k q_n) \\ z^{(k)} &= c_k\lambda_1^k(a_1 q_1 + a_2(\lambda_2/\lambda_1)^k q_2 + \cdots + a_n(\lambda_n/\lambda_1)^k q_n) \end{aligned}$$

If k becomes big, the terms (λ_j/λ_1) for $j = 2 \dots n$ become small if $|\lambda_1| > |\lambda_j|$, and therefore $z^{(k)} \to (c_k \lambda_1^k a_1) q_1$. Thus for all $z^{(0)}$, where $a_1 \neq 0$ the proposed Algorithm 1 will converge to the eigenvector corresponding to the largest eigenvalue of A.

The algorithm is very simple, but of limited use, since it only finds the eigenvector corresponding to the largest eigenvalue, and if λ_1 is not much larger than λ_2 convergence is very slow. To speed things up we would need the absolute value of λ_1 to be large.

We improve the algorithm using the following idea: For any $\mu \in \mathbb{R}$ that is not an eigenvalue of A, the eigenvectors of A are the same as of $(A - \mu I)^{-1}$. Furthermore, if λ_i is an eigenvalue of A then $(\lambda_i - \mu)^{-1}$ is an eigenvalue of $(A - \mu I)^{-1}$.

Consequently, if we choose a value μ that is close to an eigenvalue λ_j. Then $(\lambda_i - \mu)^{-1}$ will be large, and therefore the eigenvector of $A - \mu I$ can be computed very fast. This leads to the method called *inverse iteration* which is described in Algorithm 2. The algorithm computes the eigenvector corresponding to the eigenvalue nearest to μ.

Data: Matrix A, initial μ close to the desired eigenvalue, initial vector $z^{(0)}$ with $\|z^{(0)}\| = 1$
Initialisation: set $k = 1$
while *convergence criterion not satisfied* **do**

solve $(A - \mu I)w = z^{(k-1)}$ for w	apply $(A - \mu I)^{-1}$
$z^{(k)} := \frac{w}{\|w\|}$	normalise
$k \leftarrow k + 1$	

end

Algorithm 2: Inverse iteration

We can further improve inverse iteration by continuously improving the eigenvalue estimate μ in each step to increase the rate of convergence. Therefore, we use the Rayleigh quotient

$$r(z) := \frac{z^T A z}{z^T z}, \quad z \in \mathbb{C}^n \backslash \{0\}.$$

to estimate the eigenvalue λ from the estimated (normed) eigenvector z, thus $\lambda = z^T A z$ (this minimizes $\|Ax - \lambda x\|$). This leads to the *Rayleigh quotient iteration* outlined in Algorithm 3. This algorithm is very fast and shows a cubic convergence rate.

1.1.3 QR algorithm

The QR algorithm is a stable and simple procedure for calculating all eigenvalues and eigenvectors. The algorithm uses the QR factorization and the next iterate is a recombination of the factors in reverse order (see Algorithm 4).

Data: Matrix A, initial vector $z^{(0)}$ with $\|z^{(0)}\| = 1$
Initialisation: set $k = 1$, let $\lambda^{(0)} := (z^{(0)})^T A z^{(0)}$
while *convergence criterion not satisfied* **do**

solve $(A - \lambda^{(k-1)} I)w = z^{(k-1)}$ for w	apply $(A - \lambda^{(k-1)} I)^{-1}$
$z^{(k)} := \frac{w}{\|w\|}$	normalise
$\lambda^{(k)} := (z^{(k)})^T A z^{(k)}$	Rayleigh quotient
$k \leftarrow k + 1$	

end

Algorithm 3: Rayleigh quotient iteration

First, we note that in each iteration of the algorithm we use only (stable) similarity transformation. Thus $A^{(k)}$ and $A^{(k-1)}$ are similar, since $R^{(k)} = (Q^{(k)})^* A^{(k-1)}$, and therefore $A^{(k)} = (Q^{(k)})^* A^{(k-1)} Q^{(k)}$. Thus, if $A^{(k)}$ converges, this matrix has the same eigenvalues and eigenvectors as $A^{(0)}$.

Second, QR algorithm can be seen as a more sophisticated variation of power iteration (Algorithm 1). Instead of using one single vector, QR algorithm works with a complete orthonormal basis of vectors, using the QR decomposition to orthogonalize. For a hermitian matrix A the algorithm convergences to a diagonal matrix of eigenvalues. The matrix Q is orthogonal, and the columns of Q are the eigenvectors of A. For the non-hermitian case, the algorithm converges to the a Schur factorisation of the matrix (see Lemma 1.1.2).

Data: Matrix A
Initialisation: set $k = 1$, $A^{(0)} = A$
while *convergence criterion not satisfied* **do**

$Q^{(k)} R^{(k)} = A^{(k-1)}$	QR factorization of $A^{(k-1)}$
$A^{(k)} = R^{(k)} Q^{(k)}$	recombine in reverse order
$k \leftarrow k + 1$	

end

Algorithm 4: QR algorithm

QR algorithm is the standard algorithm for computing all eigenvalues of a matrix. The algorithm can achieve cubic convergence for hermitian matrices, given the following enhancements:

1. Initially, A is reduced to tridiagonal form using Householder transformations.

2. Instead of $A^{(k)}$, the shifted matrix $A^{(k)} - \mu_k I$ is factored, where μ_k is an eigenvalue estimate (same idea as Rayleigh quotient iteration, Algorithm 3).

3. When possibly $A^{(k)}$ is split into submatrices (a divide and conquer strategy).

1.2 Generalized eigenvalue problem

The problem of finding a vector $v_i \in \mathbb{C}^n \setminus \{0\}$ that obeys

$$Av_i = \lambda_i B v_i$$

for $A, B \in \mathbb{C}^{n\times n}$ is called a *general eigenvalue problem* and the *generalised eigenvalue* $\lambda_i \in C$ obeys the equation

$$\det(A - \lambda_i B) = 0.$$

In this case we can find n linearly independent vectors v_i so that $Av_i = \lambda_i B v_i$ the following equality holds

$$A = BX\Lambda X^{-1},$$

where X is the matrix composed of the eigenvectors v_i and Λ is a diagonal matrix of the eigenvalues λ_i.

For generalised eigenvalue problem the Rayleigh quotient is defined as

$$r(x) := \frac{x^* Ax}{x^* Bx}.$$

With this definition it is easy to generalize the Rayleigh quotient iteration (Algorithm 3) to calculate generalized eigenvectors.

Example 1.2.1. Consider the matrices

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

Then, the characteristic polynomial takes the form $\chi(z) = z^2 + 1$, which results to the set of the eigenvalues $\sigma(A, B) = \pm i$. Even though both matrices are symmetric the pair of them exhibits complex conjugate eigenvalues.

1.3 Singular values

Let $A \in \mathbb{C}^{m\times n}$ with $\text{range}(A) = p \leq \min\{m, n\}$, then there exists a unique factorisation called *singular value decomposition* (SVD) such that

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

where $U := [u_1, \ldots, u_m] \in \mathbb{C}^{m\times m}$ is unitary ($U^*U = UU^* = I$), and $V := [v_1, \ldots, v_n] \in \mathbb{C}^{n\times n}$ is unitary, with $\{u_j\}_{j=1}^m$ and $\{v_j\}_{j=1}^n$ being orthonormal bases on $\mathbb{C}^m$ and $\mathbb{C}^n$, respectively. $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal

$$\Sigma := \left[\begin{array}{ccc|c} \sigma_1 & & 0 & 0 \\ & \ddots & & \vdots \\ 0 & & \sigma_p & 0 \\ \hline 0 & \cdots & 0 & 0 \end{array}\right],$$

with real values $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p > 0$.

A geometrical interpretation of the SVD is that the image of a unit sphere under any $m \times n$ matrix is a hyperellipse, where the vectors $\sigma_i u_i$ are the principal semiaxis. Note that

$$Av_j = \sigma_j u_j \quad \text{for} \quad j = 1 \dots p \,,$$

and

$$U^* A V = \Sigma.$$

The SVD shows that all matrices are diagonal under the proper bases for the domain and range spaces.

There are fundamental differences between singular values and eigenvalue decompositions (compare with Lemma 1.1.2):

- SVD uses two different bases, eigenvalue decomposition just one.
- In a SVD the two bases are always orthogonal, in eigenvalue decomposition this is generally not the case (only if A is normal).
- Not all matrices have an eigenvalue decomposition, but all (even rectangular) matrices have a SVD.

Theorem 1.3.1. *Nonzero singular values of A are the square roots of the nonzero eigenvalues of A^*A and AA^*.*

Theorem 1.3.2. *If A is hermitian (i.e. $A = A^*$) the singular values of A are the absolute values of the eigenvalues of A.*

Thus, numerical computation of singular values could be based on calculating the eigenvalues of AA^*. However, this method is unstable, since the condition number of AA^* might be much bigger than that of A.

Stable SVD algorithms are based on finding the eigenvalues of the self-adjoint block matrix

$$M = \begin{pmatrix} 0 & A \\ A^* & 0 \end{pmatrix},$$

which are the singular values of A with positive and negative sign. For a proof refer to [2].

Chapter 2

Iteration methods

Iterative methods try to find an approximate solution $x \in \mathbb{R}^n$ to the linear equation $Ax = b$ or the eigenvalue problem $Ax = \lambda x$, where $A \in \mathbb{R}^{n \times n}$ is very large and typically sparse. Such matrices appear frequently in the applied sciences e.g. as stiffness matrices from partial differential equations.

Section 2.1 is based on the lecture notes [1], with additional notes from [4]. Section 2.2 summarizes the corresponding chapters from [2], with additional comments from lecture notes [3], and [5].

2.1 Fixed point iteration

We will construct iterative methods based on an important result from analysis:

Theorem 2.1.1. (Banach fixed-point theorem) *Let $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ be a contraction, i.e.*

$$\|\Phi(x) - \Phi(y)\| \leq q\|x - y\| \text{ for } q < 1 \text{ for all } x, y \in \mathbb{R}^n \, .$$

The fix point equation $x = \Phi(x)$ has exactly one solution $\hat{x} \in \mathbb{R}^n$, and the iteration $\{x^{(k)}\}$ with $x^{(0)} \in \mathbb{R}^n$, $x^{(k+1)} = \Phi(x^{(k)})$ for $k = 0, 1, 2, \ldots$, converges to the solution $\hat{x}$ for $k \to \infty$. Furthermore, for $k \geq 1$

1. $\|x^{(k)} - \hat{x}\| \leq q\|x^{(k-1)} - \hat{x}\|$ *(monotony)*
2. $\|x^{(k)} - \hat{x}\| \leq \frac{q^k}{1-q}\|x^{(1)} - x^{(0)}\|$ *(a-priori bound)*
3. $\|x^{(k)} - \hat{x}\| \leq \frac{q}{1-q}\|x^{(k)} - x^{(k-1)}\|$ *(a-posteriori bound)*

See [1] for a proof.

The basic idea is that we choose a splitting

$$A = M - N,$$

with a matrix M that can be easily inverted. With this splitting we write $Ax = b$ as fix point equation

$$\begin{aligned} Mx &= Nx + b \, , \text{or} \\ x &= Tx + c \, , \end{aligned}$$

where $T = M^{-1}N$ and $c = M^{-1}b$. Thus, the iteration

$$x^{(k+1)} = \Phi(x^{(k)})$$

is given by

$$\Phi(x) = Tx + c\,.$$

By choosing M we can construct different iterative methods to solve $Ax = b$, and we can use Theorem 2.1.1 to analyse its behaviour (e.g. it shows that such methods typically have a linear convergence rate). Note that it is essential that M^{-1} is easy to compute.

The framework is outlined in Algorithm 5. Choices of M and N are presented in the following sections.

Data: Matrix $A = [a_{ij}]$, initial vector $x^{(0)}$
Initialisation: set $k = 0$
while *convergence criterion not satisfied* **do**
  for $i = 1 : n$ **do**
    $\tilde{x}_i^{(k+1)} := \Phi(x^k)$ with Φ according to Eqns (2.1),(2.2), or (2.3)
  end
  $k \leftarrow k + 1$
end

Algorithm 5: Stationary iterative methods

2.1.1 Jacobi method

Jacobi methods splits the matrix $A = [a_{ij}]$ into its diagonal elements $M = \mathrm{diag}(\mathrm{a}_{11}, \ldots, \mathrm{a}_{\mathrm{nn}})$ and $N = M - A$, thus the inverse can be easily calculated: $M^{-1} = \mathrm{diag}(\frac{1}{\mathrm{a}_{11}}, \ldots, \frac{1}{\mathrm{a}_{\mathrm{nn}}})$. Therefore, we write

$$x^{(k+1)} = M^{-1}(b + Nx^{(k)})$$

or component-wise

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right). \tag{2.1}$$

From fix-point theorem 2.1.1 we know that the iteration converges if we can find a $q < 1$ such that $\|\Phi(x) - \Phi(y)\| = \|M^{-1}N(x - y)\| \leq q\|x - y\|$. From $\|M^{-1}N(x-y)\| \leq \|M^{-1}N\|\|x-y\|$ we conclude that $\|M^{-1}N\| \leq q$. Therefore, if $\|M^{-1}N\| < 1$ the method will converge. Especially, this is the case when A is a diagonal dominant matrix.

Note that Jacobi iteration is very simple to parallelise.

2.1.2 Gauss-Seidel method

Gauss-Seidel method accelerates convergence of the Jacobi method by using the already computed vector components $x_j^{(k+1)}$ for $j < i$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right). \tag{2.2}$$

We rewrite this equation as

$$a_{ii} x_i^{(k+1)} + \sum_{j<i} a_{ij} x_j^{(k+1)} = b_i - \sum_{j>i} a_{ij} x_j^{(k)}$$

and derive the following matrix form

$$(D - L)x^{(k+1)} = b + Ux^{(k)} ,$$

where D is a diagonal matrix and $D = \mathrm{diag}(\mathrm{a}_{11}, \ldots, \mathrm{a}_{\mathrm{nn}})$, L a strictly lower triangular matrix, and U a strictly upper triangular matrix (both with negative entries a_{ij}).

Thus, we choose $M = D - L$, where M is a lower triangular matrix, and therefore easy to invert. Furthermore, $N = M - A = U$, and the fix-point theorem 2.1.1 can be used to analyse convergence.

2.1.3 Successive over-relaxation (SOR)

The SOR is a variant of Gauss-Seidel method that improves convergence rate by using a linear inter- or extrapolation between the last iterate $x^{(k)}$ and the Gauss-Seidel iterate $x^{(k+1)}$ (Eqn 2.2). The method uses the relaxation parameter $\omega \in (0, 2)$, which is often chosen heuristically in dependence of the specific problem. The iteration is given by

$$x_i^{(k+1)} = x_i^{(k)}(1 - \omega) + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right). \tag{2.3}$$

The same approach can be applied to the Jacobi method. This is sometimes called *JOR method*.

2.2 Krylov subspace methods

The idea of all Krylov subspace methods is to project a n-dimensional problem into a lower-dimensional m-Krylov subspace (where $n >> m$).

Definition 2.2.1. Let $b \in \mathbb{R}^n$. The m^{th} *Krylov-space* of A with respect to b is defined as

$$\mathcal{K}_m(A, b) := \left\langle b, Ab, A^2 b, \ldots, A^{m-1} b \right\rangle .$$

That is, $\mathcal{K}_m(A, b)$ is the space spanned by the vectors that one obtains by iteratively applying the matrix A to b. In particular, we set $\mathcal{K}_1(A, b) := \mathbb{R}b$ and $\mathcal{K}_0(A, b) := \{0\}$.
Furthermore, we define the *Krylov matrix* by

$$K_m(A, b) = \left[b, Ab, A^2 b, \ldots A^{m-1} b \right] .$$

In the following subsections we describe methods for solving the eigenvalue problem $Ax = \lambda x$ and linear equations $Ax = b$ for hermitian and non-hermitian matrices $A \in \mathbb{C}^{n\times n}$. The following table states the names or acronyms of the corresponding methods.

	$Ax = \lambda x$	$Ax = b$
$A \neq A^*$	Arnoldi (Section 2.2.1)	GMRES (Section 2.2.3)
$A = A^*$	Lanczos (Section 2.2.2)	CG (Section 2.2.4)

2.2.1 Arnoldi iteration

We first consider the Arnoldi iteration which is a method of computing a *Hessenberg reduction* $A = QHQ^*$, where H has the structure

$$H = \begin{bmatrix} \times & & \dots & \times \\ \times & \times & & \\ & \ddots & \ddots & \vdots \\ & & \times & \times \end{bmatrix}.$$

Arnoldi iteration is an important ingredient in many numerical methods (e.g. GMRES in Section 2.2.3). In the end of this section we will show how we can use Arnoldi iteration to approximate eigenvalues.

In principle a Hessenberg reduction can be easily achieved by Housholder transformations. The advantage of Arnoldi iteration is that it can be stopped part way, which yields a reduced Hessenberg reduction of the first m columns of A.

Hessenberg reduction can be written in the form

$$AQ = QH.$$

If we consider the first m columns of this matrix equation we obtain

$$AQ_m = Q_{m+1}\hat{H}_m, \tag{2.4}$$

where $Q_m \in \mathbb{C}^{n\times m}$ is the matrix with the first m columns of Q:

$$Q_m = [q_1, q_2, \dots, q_m],$$

and $\hat{H}_m \in \mathbb{C}^{(m+1)\times m}$ is the reduced matrix with the first m columns of H (i.e. the rows at the bottom containing only zeros are removed):

$$\hat{H}_m = \begin{bmatrix} h_{11} & & \dots & h_{1m} \\ h_{21} & h_{22} & & \vdots \\ & \ddots & \ddots & \vdots \\ & & h_{m(m-1)} & h_{mm} \\ & & 0 & h_{(m+1)m} \end{bmatrix}.$$

Considering the last column of Eqn (2.4) we obtain the following recurrence relation:

$$Aq_m = h_{1m}q_1 + h_{2m}q_2 + \cdots + h_{mm}q_m + h_{(m+1)m}q_{m+1}. \tag{2.5}$$

Progressively solving this equation for q_{m+1} yields the Arnoldi iteration (see Algorithm 6).

Data: Matrix A, vector b, number n
Initialisation: $q_1 = b/\|b\|$
for $m = 1 \dots n$ **do**
  $v = Aq_m$
  for $i = 1 \dots m$ **do**
    $h_{im} = q_i^* v$
    $v = v - h_{im} q_i$
  end
  $h_{(m+1)m} = \|v\|$
  $q_{m+1} = v/h_{(m+1)m}$
end

Algorithm 6: Arnoldi iteration

Theorem 2.2.2. *The matrices Q_m generated by the Arnoldi iteration have the following properties*

1. *Q_m are the reduced QR factors of the Krylov matrix $K_m(A, b)$:*

$$K_m = Q_m R_m.$$

   Therefore, the Arnoldi process offers a systematic construction of orthonormal bases for successive Krylov subspaces.

2. *The Hessenberg matrices H_m are the projections of A onto the m-dimensional Krylov subspace $\mathcal{K}_m(A, b)$*

$$H_m = Q_m^* A Q_m.$$

   Since H_m is a projection of A the eigenvalues of H_m are related to those of A. The eigenvalues θ_j of the matrix H_m are called Arnoldi estimates *or* Ritz values.

See [2] for a proof.

Arnoldi estimates are most accurate approximations of eigenvalues. Therefore, an obvious way of approximating the eigenvalues *of* $A \in C^{n\times n}$ where n is large is to

1. Calculate the matrix H_m using Algorithm 6 for some $m << n$.

2. Calculate the eigenvalues of H_m ($\in \mathbb{C}^{m\times m}$) with a standard method (e.g. QR algorithm 4).

The Arnoldi process has interesting connection to polynomial approximation theory. This can help us to analyse in which way Arnoldi approximates eigenvalues, and which eigenvalues it will find (typically, it will detect only a few eigenvalues, since $m << n$).

If a vector is in a Krylov subspace of dimension $m+1$, i.e. $x \in \mathcal{K}_{m+1}(A, b)$, it can be expressed as

$$x = c_0 b + c_1 Ab + \dots c_m A^m b,$$

or if $q(z) = c_0 + c_1 z + \dots c_m z^m$ in polynomial form as

$$x = q(A)b.$$

Lets define P_m as the set of all monic (i.e the polynomials, where the highest coefficient $c_m = 1$) polynomials of degree m:

Lemma 2.2.3. *The characteristic polynomial of the Hessenberg matrix H_m is the unique solution of the minimization problem to find $p_m \in P_m$ such that*

$$\|p_m(A)b\|$$

is minimal.

This relates the characteristic polynomial of H_m, which acts as a 'pseudo minimal polynomial', to the characteristic polynomial of A which is the exact minimal polynomial.

2.2.2 Lanzcos iteration

Arnoldi iteration is simplified if the matrix A is symmetric or hermitian. In this case, the matrix H will not only have Hessenberg structure but can be tridiagonal.

Therefore the recurrence equation (Eqn 2.5) turns into the simpler recurrence

$$Aq_m = h_{(m-1)m} q_{m-1} + h_{mm} q_m + h_{(m+1)m} q_{m+1} \tag{2.6}$$

that is used in the hermitian case.

2.2.3 Generalized minimal residuals (GMRES)

The Arnoldi process can be used to approximately solve linear equations $Ax = b$. Lets assume $A \in \mathbb{R}^{n \times n}$ to be a regular matrix and consider the m-th Krylov subspaces $\mathcal{K}_m(A, b)$.

The idea of generalized minimal residuals (GMRES) is to choose each iterate $x^{(m)} \in \mathcal{K}_m(A, b)$ in order to minimize the norm of the residual $r^{(m)} = b - Ax^{(m)}$. The iterate $x^{(m)} \in \mathbb{R}^n$ can be written as $x^{(m)} = K_m c$ where $K_m \in \mathbb{R}^{n \times m}$ is the m-th Krylov matrix and $c \in \mathbb{R}^m$. Finding the minimal residuum is a least square problem:

$$c = \arg\min \|AK_m c - b\|, \quad x^{(m)} = K_m c.$$

This method is numerically unstable since K_m is typically ill conditioned. But, we can easily fix this by taking Q_m from Section 2.2.1 instead of K_m, which is as basis for the Krylov subspace $\mathcal{K}_m(A, b)$. The matrices Q_m can be iteratively created using Arnoldi iteration (see Therorem 2.2.2). Therefore, we write $x^{(m)} = Q_m y$ and minimize

$$y = \arg\min \|AQ_m y - b\|, \quad x^{(m)} = Q_m y.$$

Applying Eqn (2.4) yields

$$y = \arg\min \|Q_{m+1} \hat{H}_m y - b\|, \quad x^{(m)} = Q_m y,$$

and multiplying the least square problem by Q_{m+1}^* gives

$$y = \arg\min \|\hat{H}_m y - Q_{m+1}^* b\|, \quad x^{(m)} = Q_m y.$$

Finally, we arrive at Algorithm 7 after inserting $\|b\|e_1 = Q_{m+1}^* b$. This can be easily verified, since $q_1 = b/\|b\|$ (see Algorithm 6) and q_j are orthonormal vectors.

Data: Matrix A, vector b, number n
Initialisation: $k = 1$
while *convergence criterion not satisfied* **do**

$Q_k, \hat{H}_k =$ Arnoldi iteration	one step from Alg. 6
$y = \arg\min \|\hat{H}_k y - \|b\|e_1\|$	least square problem
$x^{(k)} = Q_k y$	
$k \leftarrow k + 1$	

end

Algorithm 7: GMRES algorithm

2.2.4 Conjugate gradients (CG)

The conjugate gradient (CG) method is a very effective way to solve the linear equation $Ax = b$, if $A \in \mathbb{R}^{n\times n}$ is a symmetric and positive definite matrix.

The mapping $\langle \cdot, \cdot \rangle_A \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$,

$$\langle x, y \rangle_A := x^T A y = \langle x, Ay \rangle$$

defines a scalar product on $\mathbb{R}^n$ (because of positive definiteness). The corresponding norm

$$\|x\|_A := \sqrt{\langle x, x \rangle_A} = \sqrt{x^T A x}$$

is called the *energy norm* induced by A.

The CG method picks the iterate $x^{(m)} \in \mathcal{K}_m(A, b)$ to minimize the energy norm of the *error* $e^{(m)} := x^{(m)} - \hat{x}$, where $\hat{x} := A^{-1}b$ is the exact solution, thus

$$x^{(m)} \in \mathcal{K}_m(A, b), \text{ such that } \|e^{(m)}\|_A = \text{minimum}.$$

In the following we will construct the iterative method. First, we notice that the equation can be reformulated as a quadratic optimisation problem. We define the mapping $\Phi \colon \mathbb{R}^n \to \mathbb{R}$ as

$$\Phi(x) := \frac{1}{2} x^T A x - x^T b \,. \tag{2.7}$$

Then $\nabla\Phi(x) = Ax - b$, which shows that $\hat{x} \in \mathbb{R}^n$ is a critical point of Φ, if and only if $\hat{x}$ satisfies $A\hat{x} = b$. Moreover, the Hessian of Φ is simply the positive definite matrix A, showing that Φ is strictly convex and therefore $\hat{x} := A^{-1}b$ is its unique minimiser.

Rewriting the functional Φ, we show that

$$\begin{aligned}\Phi(x) - \Phi(\hat{x}) &= \frac{1}{2}x^T Ax - x^T b - \frac{1}{2}\hat{x}^T A\hat{x} + \hat{x}^T b \\ &= \frac{1}{2}(x-\hat{x})^T A(x-\hat{x}) + \underbrace{\frac{1}{2}\left(x^T A\hat{x} + \hat{x}^T Ax\right) - \hat{x}^T A\hat{x} - x^T b + \hat{x}^T b}_{=0,\text{ since } A\hat{x}=b} \\ &= \frac{1}{2}(x-\hat{x})^T A(x-\hat{x}) \\ &= \frac{1}{2}\|x-\hat{x}\|_A^2\end{aligned}$$

Thus, minimising Φ is equivalent to minimising the error $e^{(m)}$ in the energy norm.

We approximate $\hat{x}$ by successively minimizing the functional Φ. Let $x^{(k)}$ the actual approximation, then we find the next iterate by

$$x^{(k+1)} = x^{(k)} + \alpha^{(k)} d^{(k)}, \tag{2.8}$$

where $d^{(k)} \neq 0$ is the search direction and $\alpha^{(k)}$ is magnitude we 'move' into this direction.

For a given $d^{(k)}$ we determine α by minimizing along the direction $d^{(k)}$. First we define the mapping $\alpha \to F(\alpha)$ by

$$\begin{aligned}F(\alpha) &:= \Phi(x^{(k)} + \alpha d^{(k)}) \\ &= \Phi(x^{(k)}) + \alpha d^{(k)T} Ax^{(k)} + \frac{1}{2}\alpha^2 d^{(k)T} Ad^{(k)} - \alpha d^{(k)T} b\,.\end{aligned}$$

Then, $F(\alpha)$ is minimal if $F'(\alpha^{(k)}) = 0$, thus

$$\alpha^{(k)} = \frac{(b - Ax^{(k)})^T d^{(k)}}{d^{(k)T} Ad^{(k)}} =: \frac{r^{(k)T} d^{(k)}}{d^{(k)T} Ad^{(k)}}\,, \tag{2.9}$$

where $r^{(k)} = b - Ax^{(k)}$ is the residuum at the k-th iterate.

In the CG method, the direction $d^{(k+1)}$ is chosen to be a linear combination of the previous direction $d^{(k)}$ and the residual $r^{(k+1)}$ in such a way that $d^{(k+1)}$ and $d^{(k)}$ are orthogonal with respect to A

$$d^{(k+1)} := r^{(k+1)} + \beta_k d^{(k)}\,,$$

with β_k such that

$$\langle d^{(k)}, d^{(k-1)}\rangle_A = d^{(k)T} Ad^{(k-1)} = 0\,.$$

We calculate

$$0 = d^{(k+1)T} Ad^{(k)} = r^{(k+1)T} Ad^{(k)} + \beta^{(k)} d^{(k)T} Ad^{(k)}\,,$$

and therefore

$$\beta^{(k)} = -\frac{r^{(k+1)T} Ad^{(k)}}{d^{(k)T} Ad^{(k)}} = -\frac{\langle r^{(k+1)}, d^{(k)}\rangle_A}{\langle d^{(k)}, d^{(k)}\rangle_A}\,. \tag{2.10}$$

With Eqns (2.9) and (2.10) the CG iteration of Eqn 2.8 is well defined. We summarize:

Theorem 2.2.4. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric and positive definite and let $\Phi\colon \mathbb{R}^n \to \mathbb{R}$ be as in (2.7). Let $x^{(0)} = 0$ and consider the iteration (the CG method)*

$$x^{(k+1)} = x^{(k)} - \frac{\langle r^{(k)}, d^{(k)}\rangle}{\langle d^{(k)}, d^{(k)}\rangle_A} d^{(k)}$$

with $r^{(k)} := b - Ax^{(k)}$, $d^{(0)} = r^{(0)}$, and

$$d^{(k)} = r^{(k)} - \frac{\langle r^{(k)}, d^{(k-1)}\rangle_A}{\langle d^{(k-1)}, d^{(k-1)}\rangle_A} d^{(k-1)}\,,$$

where we assume that the iteration is stopped if $r^{(k)} = 0$.

If $r^{(k)} = 0$, then $x^{(k)} = \hat{x} = A^{-1}b$. Else, the Krylov space $\mathcal{K}_k(A, b)$ has dimension k and $x^{(k)}$ minimises Φ on the space $\mathcal{K}_k(A, b)$.

Some properties of the CG method are described by the following lemma:

Lemma 2.2.5. *Let $x^{(0)}$ an arbitrary initial vector, and $d^{(0)} = r^{(0)}$. If $x^{(k)} \neq \hat{x}$ for $k = 0, \ldots, m$, then*

1.
$$r^{(m)T} d^{(j)} = 0\,, \quad \forall 0 \leq j < m\,,$$

2.
$$r^{(m)T} r^{(j)} = 0\,, \quad \forall 0 \leq j < m\,,$$

3.
$$\langle d^{(m)}, d^{(j)}\rangle_A = 0\,, \quad \forall 0 \leq j < m\,.$$

The first equation shows that the residuals are orthogonal to all prior search directions. Thus, to reach the exact solution it is enough to go into each direction only once.

The second equation shows that the residuals are orthogonal to each other. This means especially that the residuals are linear independent, and therefore the CG method finds the solution $\hat{x}$ in at most n steps.

The third equation shows that all search directions are orthogonal to each other regarding the inner product $\langle\cdot,\cdot\rangle_A$.

See [1] for a proof of the lemma.

The following reformulations of $\alpha^{(k)}$ (Eqn 2.9), and $\beta^{(k)}$ (Eqn 2.10) yields the method described in Algorithm 8:

- We calculate

$$\begin{aligned} r^{(k)T} d^{(k)} &= r^{(k)T} r^{(k)} + \beta^{(k-1)} r^{(k)T} d^{(k-1)} \\ &= r^{(k)T} r^{(k)} \quad \text{(by Lemma 2.2.5)}\ . \end{aligned}$$

  Inserting into Eqn (2.9) yields

$$\alpha^{(k)} = \frac{\|r^{(k)}\|_2^2}{\langle d^{(k)}, d^{(k)}\rangle_A}\ . \tag{2.11}$$

- First we notice that $-\alpha A d^{(k)} = r^{(k+1)} - r^{(k)}$. This is easily verified by multiplying the equation by A^{-1} from left, yielding Eqn (2.8). Then we can calculate

$$\begin{aligned} r^{(k+1)t} A d^{(k)} &= -\frac{1}{\alpha^{(k)}} (r^{(k+1)t} r^{(k+1)} - r^{(k+1)t} r^{(k)}) \\ &= -\frac{1}{\alpha^{(k)}} \|r^{(k+1)}\|_2^2 \quad \text{(by Lemma 2.2.5)} \\ &= -\frac{\|r^{(k+1)}\|_2^2}{\|r^{(k)}\|_2^2} d^{(k)t} A d^{(k)} \ . \end{aligned}$$

  Inserting into Eqn (2.10) yields

$$\beta^{(k)} = \frac{\|r^{(k+1)}\|_2^2}{\|r^{(k)}\|_2^2} \ . \tag{2.12}$$

Data: a positive definite and symmetric matrix $A \in \mathbb{R}^{n \times n}$, some $b \in \mathbb{R}^n$
Result: the solution $x^* \in \mathbb{R}^n$ of $Ax = b$
Initialisation: set $x^{(0)} = 0$, $r^{(0)} = b$, $d^{(0)} = r^{(0)}$, $k = 0$
while *convergence criterion not satisfied* **do**

$$\begin{aligned} \alpha^{(k)} &= \frac{\|r^{(k)}\|^2}{\langle d^{(k)}, d^{(k)} \rangle_A} \\ x^{(k+1)} &= x^{(k)} + \alpha^{(k)} d^{(k)} \\ r^{(k+1)} &= r^{(k)} - \alpha^{(k)} A d^{(k)} \\ \beta^{(k)} &= \frac{\|r^{(k+1)}\|^2}{\|r^{(k)}\|^2} \\ d^{(k+1)} &= r^{(k+1)} + \beta^{(k)} d^{(k)} \end{aligned}$$

  $k \leftarrow k + 1$;
end
define $\hat{x} := x^{(k)}$

Algorithm 8: Conjugate gradient method.

Theoretically, solving an equation with the conjugate gradient method is not very efficient, because it requires more operations than for instance a Cholesky or LDL decomposition. In addition, the convergence after n steps is purely theoretical, as it only holds when all the computations are performed in exact arithmetic without any rounding errors. In practise, however, it performs much better, because one can, and should, make use of the advantage of iterative methods: One need not stop only when the residual equals zero, but rather when the residual is sufficiently small.

One problem of CG and GMRES methods is that the convergence speed depends heavily on the condition of the matrix A. In order to improve the performance, usually the method is not applied directly to the equation $Ax = b$,

but rather to a transformed problem $M^{-1}Ax = M^{-1}b$, where the (easily invertible) matrix M is chosen in such a way that $M^{-1}A$ is much better conditioned than A. This approach is called *preconditioning*.

In theory, a slightly different approach is required, as the matrix $M^{-1}A$ will not be symmetric any more. To that end one can consider a Cholesky factorisation $M = LL^T$ of the positive definite and symmetric matrix M and then apply the CG method to the equation $L^{-1}AL^{-T}\hat{x} = L^{-1}b$ and solve $L^Tx = \hat{x}$. It is possible to rewrite the resulting algorithm in a such a way that one only needs the original matrix M and never its factorisation.

Chapter 3

Nonlinear systems of equations

In this chapter we solve the equation $f(x) = 0$ numerically for arbitrary f in one and higher dimensions. This will be achieved by methods that are based on the Newton's method.

Obviously, solving such equations has many application. A frequent application arises from unconstrained optimisation, since the derivative of the function is set to zero in order to find extremal points.

The section is closely based on [5], with additional comments from [4].

3.1 Newton's method

3.1.1 One-dimensional geometric motivation

We want to numerically find the root of a dimensional function, thus for a given function $f : \mathbb{R} \to \mathbb{R}$ we search for $x \in \mathbb{R}$ so that $f(x) = 0$. Furthermore, we assume f to be continuously differentiable.

We start with an initial guess x_0 and iteratively improve it. The idea of the method is to locally approximate the function by its tangent around the guess $x^{(k)}$:

$$f(x^{(k)} + h) = f(x^{(k)}) + hf'(x^{(k)}) + O(h^2).$$

In the next step the x-intercept of the tangent is computed ($0 = f(x^{(k)}) + hf'(x^{(k)})$). The value $x^{(k+1)} = x^{(k)} + h$ will typically be a better approximation. Thus we obtain the following iterative method

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}. \tag{3.1}$$

In this way $x^{(k)}$ is improved in every step (illustrated in Figure 3.1).

Newton's method is frequently used in solving equations as well as in optimisation. It is a very powerful approach since its convergence rate is quadratic. However, the Newton method has two main difficulties: First, if a stationary point of the function f is encountered, i.e. $f'(x^{(k)}) = 0$, we cannot calculate

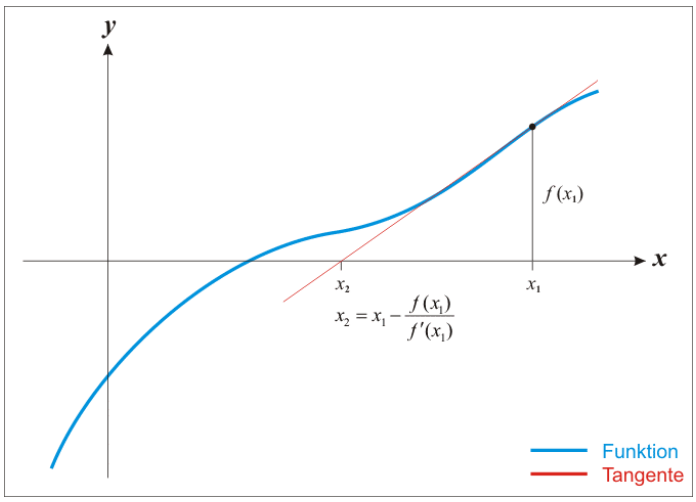


Figure 3.1: One iteration of the one-dimensional Newton method.

Eqn 3.1. Second, Newton's method does not globally converge. Indeed, for arbitrary initialisation x_{init}, we can expect Newton's method to diverge.

One possibility for obtaining a higher probability of convergence is the combination of Newton's method with one of the line search algorithms of Section 3.3.

3.1.2 Higher-dimensional generalisation

In this section we consider the case where we have n equations and n unknowns, thus for a given function $f : \mathbb{R}^n \to \mathbb{R}^n$ we search for $x \in \mathbb{R}^n$ so that $f(x) = 0$. Again, we assume f to be continuously differentiable.

We use the same concept as in the one-dimensional case. We first linearise around the point $x^{(k)}$:

$$f(x^{(k)} + d) \approx f(x^{(k)}) + J_f(x^{(k)})d + O(\|d\|^2),$$

with $x^{(k)} \in \mathbb{R}^n$ and $d^{(k)} \in \mathbb{R}^n$, and where $J_f(x^{(k)}) \in \mathbb{R}^{n\times n}$ is the Jacobian matrix.

As in the one-dimensional case we set the linearised equation to zero, i.e. $f(x^{(k)}) + J_f(x^{(k)})d = 0$, and solve for d, thus $d = -J_f^{-1}(x^{(k)})f(x^{(k)})$, where $J_f^{-1}(x^{(k)})$ is the inverse of the Jacobian matrix. This yields the following iteration

$$x^{(k+1)} = x^{(k)} - J_f^{-1}(x^{(k)})f(x^{(k)}).$$

The equation corresponds to the one-dimensional case (see Eqn 3.1), but the method is **not** implemented in this way. Since the calculation of the inverse Jacobian is computationally expensive, d is calculated directly from the linear system of equations

$$\begin{aligned} J_f(x^{(k)})d &= -f(x^{(k)}) \ , \\ x^{(k+1)} &= x^{(k)} + d. \end{aligned}$$

This method is described in Algorithm 9.

Generally, the computation time of d has to be taken into account when one studies the efficiency of the method. For decomposition methods the computation time is of order n^3. Iterative methods as described in Chapter 2 may often be faster depending how accurate d is computed.

Data: function $f\colon \mathbb{R}^n \to \mathbb{R}^n$, initial guess x_{init} $(\in \mathbb{R}^n)$;
Initialisation: set $x_1 := x_{\text{init}}$, $k = 1$;
while *convergence criterion not satisfied* **do**

solve $J_f(x_k)\, d = -f(x_k)$ for d
$x^{(k+1)} := x^{(k)} + d$
$k \leftarrow k + 1$

end

Algorithm 9: Newton's method.

The Newton method in higher dimensions has similar drawbacks than in one dimension: Problems will arise when the matrix $J_f(x^{(k)})$ is ill conditioned, because then an accurate solution of the equation $J_f(x^{(k)})d = -f(x^{(k)})$ is difficult. Furthermore, we can not guarantee convergence for arbitrary initial values.

Additionally, the computation of the Jacobian $J_f(x^{(k)})$ in each Newton step is computational expensive for large systems of equations. In the following two sections we discuss methods that have been developed specifically to counter these problems.

3.2 Quasi-Newton methods

3.2.1 One-dimensional motivation: Secant method

For a given function $f : \mathbb{R} \to \mathbb{R}$ and we search for $x \in \mathbb{R}$ so that $f(x) = 0$. As in the one-dimensional Newton method the idea is to employ an iterative algorithm, defining the next iterate $x^{(k+1)}$ by approximating the function f around $x^{(k)}$ linearly by $f(x) \approx f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)})$, and then solving the linear equation $f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) = 0$ for x.

Since $f'(x^{(k)})$ may not exist, or may be zero, or could be difficult to calculate numerically, we avoid direct calculation, and use the following approximation instead:

$$f'(x^{(k)}) = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}}.$$

Note that the approximation becomes more exact as the method converges (i.e. $x^{(k)} - x^{(k-1)} \to 0$).

Inserting this additional approximation into Eqn (3.1)

$$x^{(k+1)} = x^{(k)} - \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}\, f(x^{(k)})$$

is obtained.

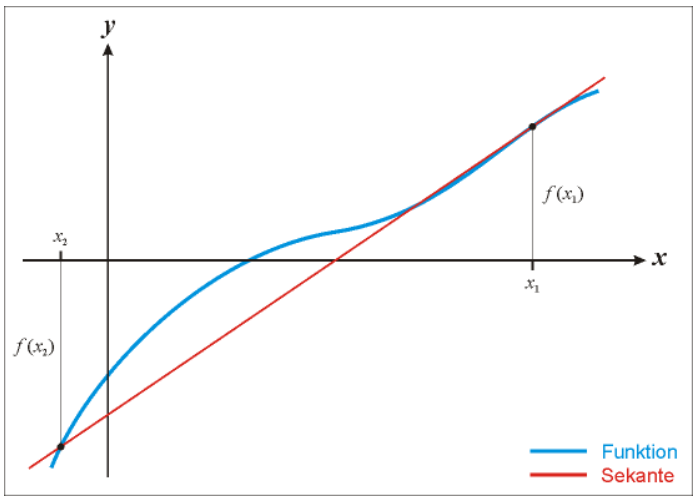


Figure 3.2: One iteration of the one-dimensional secant method.

More intuitively, the idea behind this approach is to approximate the function f (or its graph) near $x^{(k)}$ by the secant through the points $\left(x^{(k)}, f(x^{(k)})\right)$ and $\left(x^{(k-1)}, f(x^{(k-1)})\right)$ and then to find the zero of this line. Hence the name: *secant method*. See also Figure 3.2.

Using this approach we do not have to calculate the derivative $f'(x^{(k)})$. This simplification, however, comes at a price: instead of quadratic convergence, one only has super-linear convergence (more precisely, the convergence rate equals the golden section $(\sqrt{5}+1)/2$).

Note, that the secant method has an interpretation that is very similar to that of Newton's method. In Newton's method, we have approximated the function f by its linearisation around $x^{(k)}$. Here, we use the following approximation $\tilde{f}$ of f satisfying $\tilde{f}(x^{(k)}) = f(x^{(k)})$ and the one-dimensional *secant equation*

$$\tilde{f}'(x^{(k)})(x^{(k)} - x^{(k-1)}) = f(x^{(k)}) - f(x^{(k-1)}) \; . \tag{3.2}$$

3.2.2 Higher-dimensional generalisation

For generalising the one-dimensional secant method to higher dimensions we first start analogously to Section 3.1.2 and linearise the function $f : \mathbb{R}^n \to \mathbb{R}^n$ around $x^{(k)} \in \mathbb{R}^n$ by

$$\tilde{f}(x^{(k)} + d) = f(x^{(k)}) + B_k d$$

where $B_k \in \mathbb{R}^{n \times n}$ is an approximation of the Jacobian $J_f(x^{(k)})$. We calculate d from $\tilde{f} = 0$, which can be computed by solving the system of linear equations $B_k d = -f(x^{(k)})$.

This approach makes sense, if the following conditions are satisfied:

- The matrix B_k must be an estimate of $J_f(x^{(k)})$ in some sense. Otherwise, we cannot expect good convergence rates.
- The equation $B_k d = -f(x^{(k)})$ can be solved without too much effort. This could be achieved by approximating B_k^{-1} instead of B_k.

Quasi-Newton methods choose the estimates B_k of the Jacobian $J_f(x^{(k)})$ to fit the *secant equation* for more dimensions which is given by

$$B_k(x^{(k)} - x^{(k-1)}) = f(x^{(k)}) - f(x^{(k-1)}). \tag{3.3}$$

For more than one dimension this system of equations is highly under determined (n equations, $n \times n$ unknowns). As a result the choice of B_k is not unique.

Brodyden's method uses an initial estimate B_k and improves it by taking the solution of the secant equation which is a minimal modification to B_k. By a minimal modification we assume that the new estimate B_{k+1} is close to the original estimate B_k according to the Frobenius norm (thus minimize $\|B_{k+1} - B_k\|_F$).

We construct a matrix B_{k+1} such that

$$B_{k+1}s^{(k+1)} = y^{(k+1)} \text{ , and} \tag{3.4}$$

$$B_{k+1}u = B_k u \text{ for all } u \text{ orthogonal to } s^{(k+1)} \text{ ,} \tag{3.5}$$

where

$$s^{(k+1)} := x^{(k+1)} - x^{(k)} \text{ , and}$$
$$y^{(k+1)} := f(x^{(k+1)}) - f(x^{(k)}).$$

Thus, Eqn (3.4) is just a reformulation of the secant equation (Eqn 3.3).

The construction of B_k is achieved by the following rank one update:

$$B_{k+1} = B_k + \frac{\left(y^{(k+1)} - B_k s^{(k+1)}\right) s^{(k+1)T}}{s^{(k+1)T} s^{(k+1)}}. \tag{3.6}$$

We show that B_{k+1} fits into Eqns (3.4) and (3.5): This is done by multiplying Eqn (3.6) by a u, such that $\langle s^{(k+1)}, u\rangle = 0$. Then, the second term of the right hand side vanishes, and therefore $B_{k+1}u = B_k u$ (Eqn 3.5).

Furthermore, if we multiply Eqn (3.6) by $s^{(k+1)}$, we obtain $B_{k+1}s^{(k+1)} = B_k s^{(k+1)} + y^{(k+1)} - B_k s^{(k+1)} = y^{(k+1)}$, yielding Eqn (3.4).

Sherman-Morrison formula is used to update the inverse of the Jacobian matrix directly, yielding

$$B_{k+1}^{-1} = B_k^{-1} + \frac{s^{(k+1)} - B_k^{-1}y^{(k+1)}}{s^{(k+1)T}B_k^{-1}y^{(k+1)}} \left(s^{(k+1)T}B_k^{-1}\right) \tag{3.7}$$

The method is referred to as *good Broyden's method* and is illustrated in Algorithm 10. For initialisation of B_1^{-1} the inverted Jacobian $J_f(x_{init})$ is calculated exactly.

Generally, for arbitrary initial values all Quasi-Newton methods (like Newton methods) do not always converge. One possibility for obtaining a higher probability of convergence is using line search as described in the following section.

Data: function $f\colon \mathbb{R}^n \to \mathbb{R}^n$, initial guess x_{init}
Initialisation: set $x_1 := x_{\text{init}}$, $k = 1$, $B_1^{-1} = J_f(x_1)^{-1}$
while *convergence criterion not satisfied* **do**

$d := -B_k^{-1} f(x^{(k)})$
$x^{(k+1)} := x^{(k)} + d$
compute B_{k+1}^{-1} according to Eqn (3.7)
$k \leftarrow k + 1$

end

Algorithm 10: Good Broyden's method

3.3 Basic line search concepts

In this section, we assume that we have already calculated a direction $d \in \mathbb{R}^n$, such that

$$x^{(k+1)} = x^{(k)} + d\,, \tag{3.8}$$

see Sections 3.1.2 and 3.2.2. Instead of using Eqn (3.8) we will use

$$x^{(k+1)} = x^{(k)} + td\,, \tag{3.9}$$

and try to find find a good step size t, that minimizes the function f along the line $x^{(k)} + td$, e.g.

$$g(t) := \frac{1}{2}\|f(x^{(k)} + td)\|^2\,.$$

Line search algorithms use the following framework of Algorithm 11 that consist of two sub-algorithms:

First, an algorithm that, given the values $t > 0$, $g(t)$, and, possibly, $g'(t)$, decides whether the step size t is too large, too small, or acceptable.

Second, an algorithm that computes a new candidate for the step size if the former candidate has been rejected.

Initialisation: set $t_L = 0$ and $t_R = +\infty$, choose some initial $t > 0$
while t *not satisfactory* **do**
  if t *is too small* **then**
    $t_L \leftarrow t$
  else
    $t_R \leftarrow t$
  end
  compute new $t \in (t_L, t_R)$
end
define $t^* := t$

Algorithm 11: Sketch of a line search algorithm.

First we note that the classification sub-algorithm has to satisfy at least the following properties.

1. For each $t > 0$, either t is classified as too small, or t is classified as too large, or t is accepted.

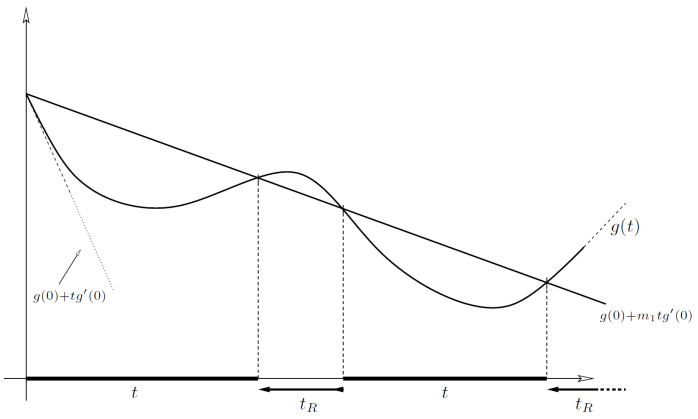


Figure 3.3: Armijo's rule. A step size is declared too large, if the actual decrease of the function value is much smaller than the predicted decrease.

2. There is some upper bound $t_{\max}$ such that every $t > t_{\max}$ is classified as too large. Thus it cannot happen that the step size increases to $+\infty$ and the line search fails to terminate (note that the upper bound need not be given explicitly).

3. Whenever t_L is classified as too small and t_R is classified as too large, there exists a non-empty open interval $I \subset [t_L, t_R]$ such that *every* element in I is classified as satisfactory (thus the algorithm has a chance to terminate for suitable updates of t).

Note that these three properties do not imply that the result will be a good choice for a step size; they are merely required for obtaining any result at all.

3.3.1 Armijo

One main criterion in all modern line search algorithms is that the actual decrease of the function g is (at least) of the same order as the expected decrease.

The expected decrease for a step size $t > 0$ is given by the derivative of g at zero, multiplied by t. Thus we should consider a step size too large, if the difference $g(t) - g(0)$ is much larger than $tg'(0)$ (note that $g'(0)$ is assumed to be negative!).

In practise, this means that we choose some $0 < m_1 < 1$ and say that a step size t is too large, if

$$g(t) > g(0) + m_1 t g'(0) \; . \tag{3.10}$$

The condition (3.10) is called *Armijo's rule*. For an illustration see Figure 3.3. In principle, this condition alone can already be used for the classification step in a line search algorithm. Then we end up with the simple Algorithm 12.

The usage of Armijo's is limited, because it never declares a step size to be too small. However, this only works, if we either have a good understanding of the function we want to optimise, or the algorithm that determines the search direction at the same time yields a step length. This is for instance the case in

Initialisation: choose some $t > 0$ and $0 < m_1 < 1$
while $g(t) > g(0) + m_1 t g'(0)$ **do**
  decrease t
end
define $t^* := t$

Algorithm 12: Line search with Armijo's rule.

the Newton method and its derivatives, where the step length $t = 1$ is asymptotically optimal and the main task of the line search is to increase the region of convergence of the method.

3.3.2 Goldstein and Price

The second classification sub-algorithm we discuss is based on Armijo's rule, but, in addition, introduces a criterion that decides whether a step size is too small. Again this criterion compares the actual decrease with the expected decrease. The difference is now that we declare the step size too small if the actual decrease is not much smaller than the expected one. The idea is that, in this case, it should be possible to decrease the value of g further by increasing t.

In practise, this means that we choose two numbers $0 < m_1 < m_2 < 1$ and say that:

- t is too large if $g(t) > g(0) + m_1 t g'(0)$,
- t is too small if $g(t) < g(0) + m_2 t g'(0)$,
- t is acceptable, if

$$m_2 g'(0) \leq \frac{g(t) - g(0)}{t} \leq m_1 g'(0) \; .$$

These three conditions are called the rule of *Goldstein and Price*. An interpretation of these condition is shown in Figure 3.4. The method is summarised in Algorithm 13.

3.3.3 Wolfe

The two methods discussed above only use the function values $g(0)$ and $g(t)$, as well as the derivative $g'(0)$ for determining the step length, but not the derivative of g at other points. It is reasonable to assume that the additional usage of gradient information may lead to better results of the line search provided that the cost of computing derivatives is not too large. We will, however, still base the decision whether a step size is too large on Armijo's rule and only use the gradient information for declaring step sizes too small.

We choose two numbers $0 < m_1 < m_2 < 1$ and say that:

- t is too large if $g(t) > g(0) + m_1 t g'(0)$,
- t is too small if $g(t) \leq g(0) + m_1 t g'(0)$ and $g'(t) < m_2 g'(0)$,

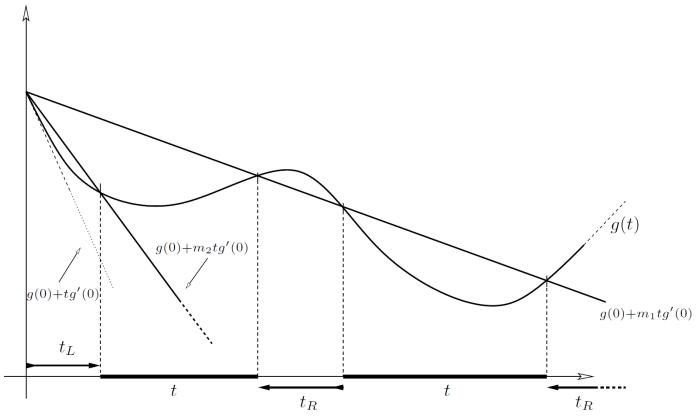


Figure 3.4: Goldstein and Price line search. In addition to Armijo's rule, a step size is declared too small, if the actual decrease of the function value is not much smaller than the predicted decrease.

Initialisation: set $t_L = 0$ and $t_R = +\infty$ choose some initial $t > 0$;
declare t unacceptable, fix $0 < m_1 < m_2 < 1$;
while *t is unacceptable* **do**
  if $g(t) > g(0) + m_1 t g'(0)$ **then**
    set $t_R \leftarrow t$;
    choose new $t \in (t_L, t_R)$
  else if $g(t) < g(0) + m_2 t g'(0)$ **then**
    set $t_L \leftarrow t$;
    choose new $t \in (t_L, t_R)$
  else
    declare t acceptable
  end
end
define $t^* := t$

Algorithm 13: Line search according to Goldstein and Price.

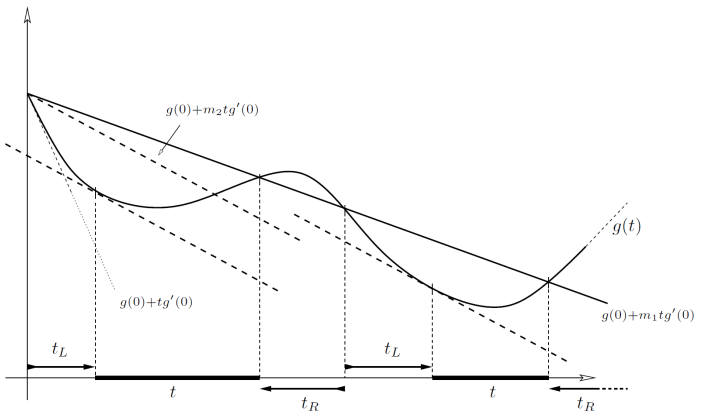


Figure 3.5: Wolfe's line search. In addition to Armijo's rule, one compares the derivative of g at t with the derivative of g at the origin.

- t is acceptable, if $g(t) \leq g(0) + m_1 t g'(0)$ and $g'(t) \geq m_2 g'(0)$.

This leads to *Wolfe's line search*. An interpretation of these conditions is provided in Figure 3.5. The method is summarised in Algorithm 14.

Initialisation: set $t_L = 0$ and $t_R = +\infty$ choose some initial $t > 0$;
declare t unacceptable, fix $0 < m_1 \leq m_2 < 1$;
while t *is unacceptable* **do**
  if $g(t) > g(0) + m_1 t g'(0)$ **then**
    set $t_R \leftarrow t$;
    choose new $t \in (t_L, t_R)$
  else if $g'(t) < m_2 g'(0)$ **then**
    set $t_L \leftarrow t$;
    choose new $t \in (t_L, t_R)$
  else
    declare t acceptable
  end
end
define $t^* := t$;

Algorithm 14: Wolfe's line search.

While in general Wolfe's line search should be preferred over the other methods, in situations where the evaluation of g' takes considerably more time than the evaluation of g alone the method of Goldstein and Price should take precedence. Note moreover that Wolfe's line search is very well suited for the Quasi-Newton methods and ensures super-linear convergence of the algorithm.

Chapter 4

Discrete-time Markov chains

The following presentation of Markov chains is based on the MIT lectures of John Tsitsiklis. Additional information can be found in [6].

4.1 Modelling with Markov chains

Most generally, mathematical modelling of physical processes is done by deriving a new state x_{new} in dependence of a old state x_{old}. Additionally some random processes ($noise$) could be included:

$$x_{new} = f(x_{old}, noise).$$

In the following chapter we use discrete time steps and discrete state space. The modelling approach is illustrated by the following example of a checkout counter.

Example 4.1.1. We model a single checkout counter of a supermarket:

- We use discrete time steps $n = 0, 1, 2, \ldots$ (e.g. hours).
- Customer arrivals happen according to a Bernoulli process (i.e. with chance p a customer arrives at each time step)
- Customer leave after a customer service time, which is a random amount of time calculated according to a geometric distribution (i.e. The probability distribution of the number X of Bernoulli trials with chance q needed to get one success).
- State of the system X_n is the number of customers at time n.

The model can be visualized by a graph, see Figure 4.1. The states are the nodes of the graph, and the transitions between the states are the graph's edges. Edge weights are the transition probabilities.

For $X_n = 1 \ldots N - 1$ the probabilities are given by: (a) $q(1 - p)$ someone leaves, and no one arrives, (b) $p(1 - q)$ someone arrives, and no one leaves, and

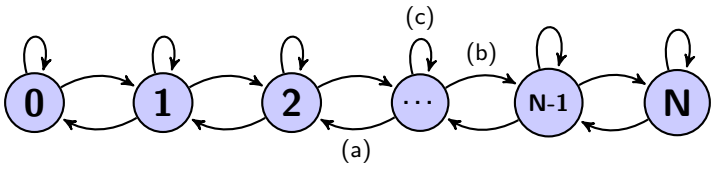


Figure 4.1: Illustration of the checkout counter example: (a) $q(1-p)$, (b) $p(1-q)$, and (c) $(1-p)(1-q)+pq$.

(c) $(1-p)(1-q)+pq$ nothing happens: no one leaves, and no one arrives, or someone arrives, and someone leaves.

In the following we will make probabilistic predictions how the model behaves, e.g. how many people will be at supermarket checkout counter when it closes at 7pm.

Definition 4.1.2. A *Markov chain* is a discrete time stochastic process $(X_n, n \geq 0)$ such that each random variable X_n takes values in a discrete set S and

$$\mathbb{P}(X_{n+1} = j | X_n = i, X_{n-1} = i_{n-1}, \ldots, X_0 = i_0) = \mathbb{P}(X_{n+1} = j | X_n = i).$$

This equation describes the *Markov property*: The transition probability from X_n is independent of the way you got to X_n, i.e. the process does not remember the past.

Modelling with Markov chains includes

1. Choosing states: The state space must be choosen carefully (a state must include everything that is relevant for the future)
2. Define possible transitions
3. Set transition probabilities

We use the notation

$$p_{ij}(n) := \mathbb{P}(X_{n+1} = j | X_n = i).$$

Thus p_{ij} denotes the *transition probability* from state i to state j at time n. If the transition probabilities are independent of the time, the Markov chain is called *homogeneous*. In the following we will focus on homogeneous Markov chains.

A homogeneous Markov chain is characterized by its transition matrix $P = [p_{ij}]$, which has often high dimension and is sparse (depending on the application). Obvious properties of the transition matrix are:

$$p_{ij} \geq 0 \qquad \forall i, j \in \{1 \ldots N\}, \tag{4.1}$$

$$\sum_{j=1}^{N} p_{ij} = 1 \qquad \forall i \in \{1 \ldots N\}. \tag{4.2}$$

This means that all the probabilities of all transitions leaving node i are positive and sum up to one.

Markov chains are used for probabilistic modelling. In the following we want to predict how the stochastic process will behave in the future.

4.2 Probabilistic predictions

Definition 4.2.1. We define the *n-step probabilities* r_{ij}, as

$$r_{ij}(n) = \mathbb{P}(X_n = j | X_0 = i).$$

This is the probability that, if we initially start at node i we will reach node j after n steps. Note that $r_{ij}(0) = \delta_{ij}$, and 1-step probabilities are exactly the transition probabilities $r_{ij}(1) = p_{ij}$.

For $n > 1$ the $n-$step probability will be the sum the probabilities of all possible paths the process might take, and we can derive the following recursions:

$$r_{ij}(n) = \sum_{k=1}^{m} r_{ik}(n-1)p_{kj} \tag{4.3}$$

or

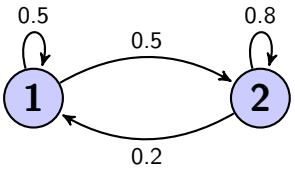
$$r_{ij}(n) = \sum_{k=1}^{m} p_{ik}r_{kj}(n-1). \tag{4.4}$$

Using n-step probabilities we arrive at

$$\mathbb{P}(X_n = j) = \sum_{i=1}^{m} \mathbb{P}(X_0 = i)r_{ij}(n),$$

and we are interested in a long term prediction (i.e. $n >> 0$).

Example 4.2.2. Consider the following Markov chain:



Calculate the n-step probabilities. Will the state be more likely be in 1 or 2, and with which probabilities (in the long run)?

In this example after a certain time the state of the chain does not depend on the initial condition any more.

Definition 4.2.3. If the limit

$$\lim_{n\to\infty} r_{ij}(n) = \pi_j$$

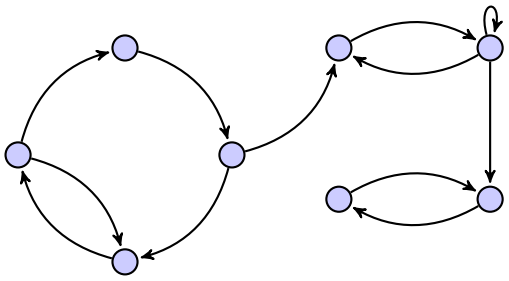
exists and it is independent of the initial state i, then π_j is called the steady state probabilities of state j.

In the following we will analyse in which situations (a) the initial state does not matter, and (b) the limit exists.

Definition 4.2.4. A state is called *recurrent*, if wherever we go from the state, there is a way back. If the state is not recurrent, it is called a *transient* state. Two states i and j *communicate* if there is a way from i to j and from j to i. Note that 'to communicate' is an equivalence relation.

We can label each node to be recurrent or transient and produce equivalence classes of recurrent and transient states regarding if the states communicate or not. For a long term prediction we know that eventually the state will leave the transient classes and enter one of the recurrent classes. The state will stay in this recurrent class.

Example 4.2.5. Label the states as transient or recurrent, and make equivalence classes



Definition 4.2.6. A state i is periodic with period $d > 1$, if d is the smallest integer such that $r_{ii}(n) = 0$ for all n which are not multiples of d. In case $d = 1$ the state i is not periodic.

Lemma 4.2.7. *If the Markov chain has a single recurrent class that is not periodic, then the n-step probabilities $r_{ij}(n)$ converge to the steady state probabilities π_j for all state $j = 1 \dots N$.*

If there is a single non-periodic recurrent class, we can take the limit of the recursion (Eqn 4.3) on both sides, it yields the following equation.

$$\pi_j = \sum_{k=1}^{N} \pi_k p_{kj} \tag{4.5}$$

or in matrix notation

$$\pi^T = \pi^T P, \text{ or } \pi = P^T \pi$$

Since the transition matrix is singular, there is a non-trivial solution. If additionally,

$$\sum_{j=1}^{N} \pi_j = 1$$

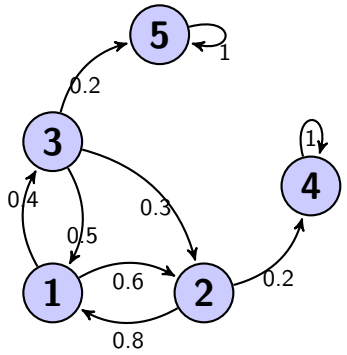
the solution is unique.

Eqns 4.5 are called the *balance equations*, and are used to calculate the *steady state probabilities* π_j.

Generally wherever we start in a Markov chain, after a certain time the state will end up in in a recurrent class. With an initial value X_0 in a transient class, we are interested in two questions:

- What is the absorption probability of each recurrent class (i.e. the probability the state ends up in this class)?
- What is the expected number of transitions, until the state enters the recurrent class?

Example 4.2.8. Consider the following example:



The nodes 4, and 5 are transient, and nodes 1, 2 and 3 are recurrent. Lets denote the chance to end up in node 4 starting from node i as *adsorption probability* a_i. Therefore $a_4 = 1$, and $a_5 = 0$. We are interested in a_1, a_2, and a_3. If we start at node 2 the chance to end up in node 4 is $a_2 = 0.2a_4 + 0.8a_1$, because 0.2 is the chance to move from node 2 to node 1, and 0.8 is the chance to visit node 1. Per definition a_1 the chance to end up in node 4 from node 1. If we do this for every node i this yields a system of linear equations, that enable us to calculate all a_i.

Generalising the idea from Example 4.2.8 the adsorption probability of a recurrent class RC starting at a transient node i can be calculated according to

$$a_i = \sum_{j=1}^{N} p_{ij} a_j \quad \forall i \in \{\text{transient states}\},$$

for the recurrent nodes i

$$a_i = \begin{cases} 1 & \text{if } i \in RC \\ 0 & \text{if } i \notin RC \end{cases}.$$

The question about the expected number of transitions until the state enters a recurrent class can be answered by similar arguments. Consider the Markov chain of Example 4.2.8: Lets denote the expected number of transitions to enter the recurrent node 4 from node i as μ_i. If we start at node 1 this number will be $\mu_1 = 0.6\mu_2 + 0.4\mu_3 + 1$. Therefore, we make one transition, and add the expected number of transition from node 2 and 3. Generalising the idea yields:

$$\mu_i = 1 + \sum_{j=1}^{N} p_{ij} \mu_j \quad \forall i \in \{\text{transient states}\},$$

and for the recurrent nodes i

$$\mu_i = \begin{cases} 0 & \text{if } i \in RC \\ \infty & \text{if } i \notin RC \end{cases}.$$

Chapter 5

Random numbers and Monte Carlo Simulation

Bibliography

[1] O. Scherzer *Numerische Mathematik.* Lecture notes 2013.

[2] N. Trefethen and D. Bau III. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[3] H. Schichl *Numerik 2.* Lecture notes 2009/10.

[4] A. Quarteroni, R. Sacco, and F. Saleri *Numerical Mathematics.* Springer Verlag, Berlin, 2000.

[5] M. Grasmair *Continuous Optimisation.* Lecture notes 2012.

[6] W.K Ching, X. Huang, M.K. Ng and T.-K. Siu *Markov Chains - Models, Algorithms and Appllications.* Springer Verlag, Berlin, 2013.