# TMA4215 Numerical Mathematics: Collection of lecture notes.

November 15, 2013

## Contents

## 1 Solution of systems of nonlinear equations

Given a system of nonlinear equations

$$\mathbf{f}(\mathbf{x}) = 0, \qquad \mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m \tag{1}$$

for which we assume that there is (at least) one solution **xi**. The idea is to rewrite this system into the form

$$\mathbf{x} = \mathbf{g}(\mathbf{x}), \qquad \mathbf{g} : \mathbb{R}^m \to \mathbb{R}^m. \tag{2}$$

The solution $\xi$ of (1) should satisfy $\xi = \mathbf{g}(\xi)$, and is thus called a *fixed point* of $\mathbf{g}$. The iteration schemes becomes: given an initial guess $\mathbf{x}^{(0)}$, the *fixed point iterations* becomes

$$\mathbf{x}^{(k+1)} = \mathbf{g}(\mathbf{x}^{(k)}), \qquad k = 1, 2, \dots . \tag{3}$$

The following questions arise:

(i) How to find a suitable function $\mathbf{g}$?
(ii) Under what conditions will the sequence $\mathbf{x}^{(k)}$ converge to the fixed point $\xi$?
(iii) How quickly will the sequence $\mathbf{x}^{(k)}$ converge?

Point (ii) can be answered by Banach fixed point theorem:

**Theorem 1.1.** *Let $D \subseteq \mathbb{R}^m$ be a convex*[1] *and closed set. If*

$$\mathbf{g}(D) \subseteq D \tag{4a}$$

*and*

$$\|\mathbf{g}(\mathbf{y}) - \mathbf{g}(\mathbf{v})\| \le L\|\mathbf{y} - \mathbf{v}\|, \qquad \textit{with } L < 1 \ \textit{ for all } \ \mathbf{y}, \mathbf{v} \in D, \tag{4b}$$

*then $G$ has a unique fixed point in $D$ and the fixed point iterations* (3) *converges for all $\mathbf{x}^{(0)} \in D$. Further,*

$$\|\mathbf{x}^{(k)} - \xi\| \le \frac{L^k}{1-L}\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|. \tag{4c}$$

*Proof.* The proof is based on the *Cauchy Convergence theorem*, saying that a sequence $\{\mathbf{x}^{(k)}\}_{k=0}^{\infty}$ converges to some $\xi$ if and only if for every $\varepsilon > 0$ there is an $N$ such that

$$\|\mathbf{x}^{(l)} - \mathbf{x}^{(k)}\| < \varepsilon \qquad \text{for all} \quad l, k > N. \tag{5}$$

Assumption (4a) ensures $\mathbf{x}^{(k)} \in D$ as long as $\mathbf{x}^{(0)} \in D$. From (3) and (4b) we get:

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| = \|\mathbf{g}(\mathbf{x}^{(k)}) - \mathbf{g}(\mathbf{x}^{(k-1)})\| \le L\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\| \le L^k\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|.$$

We can write $\mathbf{x}^{(k+p)} - \mathbf{x}^{(k)} = \sum_{i=1}^{p}(\mathbf{x}^{(k+i)} - \mathbf{x}^{(k+i-1)})$, thus

$$\begin{aligned}
\|\mathbf{x}^{(k+p)} - \mathbf{x}^{(k)}\| &\le \sum_{i=1}^{p}\|\mathbf{x}^{(k+i)} - \mathbf{x}^{(k+i-1)}\| \\
&= (L^{p-1} + L^{p-2} + \cdots + 1)\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\| \le \frac{L^k}{1-L}\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|,
\end{aligned}$$

[1] $D$ is convex if $\theta y + (1-\theta)v \in D$ for all $y, v \in D$ and $\theta \in [0, 1]$.

since $L < 1$. For the same reason, the sequence satisfy (5), so the sequence converges to some $\xi \in D$. Since the inequality is true for all $p > 0$ it is also true for $\xi$, proving (4c).

To prove that the fixed point is unique, let $\xi$ and $\eta$ be two different fixed points in $D$. Then

$$\|\xi - \eta\| = \|\mathbf{g}(\xi) - \mathbf{g}(\eta)\| < \|\xi - \eta\|$$

which is impossible. □

For a given problem, it is not necessarily straightforward to justify the two assumptions of the theorem. But it is sufficient to find some $L$ satisfying the condition $L < 1$ in some norm to prove convergence.

Let $\mathbf{x} = [x_1, \dots, x_m]^T$ and $\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \dots, g_m(\mathbf{x})]^T$. Let $\mathbf{y}, \mathbf{v} \in D$, and let $\mathbf{x}(\theta) = \theta\mathbf{y} + (1-\theta)\mathbf{v}$ be the straight line between $\mathbf{y}$ and $\mathbf{v}$. According to the mean value theorem for functions, for each $g_i$ there exist at $\tilde{\theta}_i$ such that

$$\begin{aligned} g_i(\mathbf{y}) - g_i(\mathbf{v}) = g_i(\mathbf{x}(1)) - g_i(\mathbf{x}(0)) &= \frac{dg_i}{d\theta}(\tilde{\theta}_i)(1-0), && \tilde{\theta}_i \in (0,1) \\ &= \sum_{j=1}^{m} \frac{\partial g_i}{\partial x_j}(\tilde{\mathbf{x}}_i)(y_j - v_j), && \tilde{\mathbf{x}}_i = \tilde{\theta}_i\mathbf{y} + (1-\tilde{\theta}_i)\mathbf{v} \end{aligned}$$

since $dx_j(\theta)/d\theta = y_j - v_j$. Then

$$|g_i(\mathbf{y}) - g_i(\mathbf{v})| \le \sum_{j=1}^{m} |\frac{\partial g_i}{\partial x_j}(\tilde{\mathbf{x}}_i)| \cdot |y_j - v_j| \le \left( \sum_{j=1}^{m} |\frac{\partial g_i}{\partial x_j}(\tilde{\mathbf{x}}_i)| \right) \max_l |y_l - v_l|.$$

If we let $\bar{g}_{ij}$ be some upper bound for each of the partial derivatives, that is

$$|\frac{\partial g_i}{\partial x_j}(\mathbf{x})| \le \bar{g}_{ij}, \quad \text{for all } \mathbf{x} \in D.$$

then

$$\|\mathbf{g}(\mathbf{y}) - \mathbf{g}(\mathbf{v})\|_\infty = \left( \max_i \sum_{j=1}^{m} \bar{g}_{ij} \right) \|\mathbf{y} - \mathbf{v}\|_\infty.$$

We can then conclude that (4b) is satisfied if

$$\max_i \sum_{j=1}^{m} \bar{g}_{ij} < 1.$$

## Newton's method

Newton's method is a fixed point iterations for which

$$\mathbf{g}(\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} - J_f(\mathbf{x}^{(k)})^{-1}\mathbf{f}(\mathbf{x}^{(k)}), \tag{6}$$

where the *Jacobian* is the matrix function

$$J_f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_m}(\mathbf{x}) \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_m}(\mathbf{x}) \end{pmatrix}.$$

The Newton method can be derived as follow: Consider element $i$ in $\mathbf{f}$, that is $f_i(\mathbf{x})$. Do a multidimensional Taylor expansion of $f_i(\xi)$ around the vector $\mathbf{x}^{(k)}$, using $\mathbf{e}^{(k)} = \xi - \mathbf{x}^{(k)}$ This gives

$$0 = f_i(x_1^{(k)} + e_1^{(k)}, \dots, x_m^{(k)} + e_m^{(k)}) = f_i + \frac{\partial f_i}{\partial x_1} e_1^{(k)} + \cdots + \frac{\partial f_i}{\partial x_m} e_m^{(k)} + R_i$$

The function and all the derivatives are evaluated in $\mathbf{x}^{(k)}$. The remainder term $R_i$ consists of quadratic terms like $\mathcal{O}(e_i^{(k)} e_j^{(k)})$. If the error is small, this term is even smaller, so let us now ignore it and replace the errors $e_i^{(k)}$ with an approximation to the error $\Delta x_i^{(k)}$ to compensate. Doing so for each $i = 1, 2, \dots, m$ gives us the following system of linear equations,

$$f_i + \frac{\partial f_i}{\partial x_1} \Delta x_1^{(k)} + \cdots + \frac{\partial f_i}{\partial x_m} \Delta x_m^{(k)} = 0, \qquad i = 1, 2, \dots, m.$$

which is

$$\mathbf{f}(\mathbf{x}^{(k)}) + J_f(\mathbf{x}^{(k)}) \cdot \Delta x^{(k)} = \mathbf{0}.$$

Solve this with repect to $\Delta \mathbf{x}^{(k)}$. Remember that $\Delta \mathbf{x}^{(k)} \approx \xi - \mathbf{x}_k^{(k)}$ it seems reasonable to update our iterate with this amount, thus

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \Delta x_k^{(k)}$$

which finally results in (6).

It is possible to prove that if *i)* (1) has a solution $\xi$, *ii)* $J_f(\mathbf{x})$ is nonsingular in some open neighbourhood around $\xi$ and *iii)* the initial guess $\mathbf{x}^{(0)}$ is sufficiently close to $\xi$, the Newton iterations will converge to $\xi$ and

$$\|\xi - \mathbf{x}^{(k+1)}\| \leq K \|\xi - \mathbf{x}^{(k)}\|^2$$

for some positive constant $K$. We say that the convergence is *quadratic.*

## 2 Linear algebra

Author: Elena Celledoni.

### 2.1 Introduction

We consider the approximation of the solution of linear systems of algebraic equations with $n$ equations and $n$ unknowns:

$$\begin{array}{rcl} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n & = & b_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n & = & b_2 \\ \vdots & \vdots & \vdots \\ a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n & = & b_n \end{array} \tag{7}$$

where $(x_1, \cdots, x_n)$ are the unknowns. Given $(a_{i,j})_{1 \leq i,j \leq n}$, and $b_i$, $i = 1, \cdots, n$.

We can rewrite (7) in a matrix form by defining:

$$A := \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & \cdots & a_{2,n} \\ \vdots & \vdots & \cdots & \cdots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & \cdots & a_{n,n} \end{bmatrix}, \quad x := \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b := \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

so we have that (7) becomes

$$A \cdot x = b. \tag{8}$$

Linear systems can be easily solved when $A$ has a special structure, for example when $A$ is a diagonal matrix or triangular matrix and $a_{i,i} \neq 0$ for $i = 1, \ldots, n$:

- $A$ diagonal matrix

$$A = \begin{bmatrix} a_{1,1} & 0 & \cdots & \cdots & 0 \\ 0 & a_{2,2} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \cdots & a_{n,n} \end{bmatrix} \Rightarrow \begin{array}{ccccccc} a_{1,1}x_1 & = & b_1 & & x_1 & = & \frac{b_1}{a_{1,1}} \\ a_{2,2}x_2 & = & b_2 & & x_2 & = & \frac{b_2}{a_{2,2}} \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ a_{n,n}x_n & = & b_n & & x_n & = & \frac{b_n}{a_{n,n}} \end{array}$$

- $A$ triangular matrix

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & \cdots & a_{1,n} \\ 0 & a_{2,2} & a_{2,3} & \cdots & a_{2,n} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & 0 & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \cdots & 0 & a_{n,n} \end{bmatrix} \Rightarrow$$

$$\begin{array}{rcl} a_{1,1}x_1 + a_{1,2}x_2 + \cdots + \cdots + a_{1,n}x_n & = & b_1 \\ a_{2,2}x_2 + \cdots + \cdots + a_{2,n}x_n & = & b_2 \\ \cdots & \vdots & \vdots \quad \Rightarrow \\ a_{n-1,n-1}x_{n-1} + subsubsection**a_{n-1,n}x_n & = & b_{n-1} \\ a_{n,n}x_n & = & b_n \end{array}$$

$$\begin{array}{ccc} x_1 & = & \frac{b_1 - \sum_{j=2}^{n} a_{1,j}x_j}{a_{1,1}} \\ x_2 & = & \frac{b_2 - \sum_{j=3}^{n} a_{2,j}x_j}{a_{2,2}} \\ \vdots & \vdots & \vdots \\ x_{n-1} & = & \frac{b_{n-1} - a_{n-1,n}x_n}{a_{n-1,n-1}} \\ x_n & = & \frac{b_n}{a_{n,n}} \end{array} \tag{9}$$

The formula (9) is called back substitution algorithm. For general maurices other techniques based on matrix factorizations are in general used. If the matrix is of large size and *sparse* (i.e. it has a relatively small number of elements different from zero), then it is more convenient to use iterative techniques. We will describe here briefly both approaches. In the next section we consider the stability of linear systems, that is the sensitivity of the output with respect to perturbations in the input data.

## 2.2 Stability of linear systems

### Preliminaries

A vector norm is a function $\|\cdot\| : \mathbf{R}^n \to \mathbf{R}$ satisfying three axioms:

1. $\|u\| \geq 0$ for all $u \in \mathbf{R}^n$ and $\|u\| = 0$ if and only if $u = 0$.
2. $\|\lambda u\| = |\lambda| \|u\|$for all vectors $u \in \mathbf{R}^n$ and scalars $\lambda \in \mathbf{R}$.
3. $\|u + v\| \leq \|u\| + \|v\|$ for all $u \in \mathbf{R}^n$ and $v \in \mathbf{R}^n$.

If $u$ is a vector with $n$ components, examples of norms are:

- $\|u\|_1 := |u_1| + \cdots + |u_n|$, norm-1;
- $\|u\|_\infty := \max_{1\leq i \leq n} |u_i|$, max-norm;
- $\|u\|_2 := \left(|u_1|^2 + \cdots + |u_n|^2\right)^{\frac{1}{2}} = \left(u^T u\right)^{\frac{1}{2}}$, norm-2.

We will also make use of matrix-norms. Matrix-norms satisfy an extra axiom compared to vector norms; this axiom has to do with the product of matrices, so if $A$ and $B$ are $n \times n$ matrices, for a vector-norm according to this axiom we have that

$$\|AB\| \leq \|A\| \|B\|.$$

Assume $U$ is a matrix with elements $u_{i,j}$, examples of matrix-norms are:

- $\|U\|_F = \sqrt{\sum_{i=1}^n \sum_{j=1}^n u_{i,j}^2}$, Frobenius norm;
- $\|U\|_\infty := \max_i \sum_{j=1}^n |u_{i,j}|$ max-norm;
- $\|U\|_1 := \max_j \sum_{i=1}^n |u_{i,j}|$ norm-1.

For example for

$$U := \begin{bmatrix} 3 & 2 \\ -1 & 0 \end{bmatrix},$$

we get

$$\begin{aligned} \|U\|_F &= \sqrt{3^2 + 2^2 + (-1)^2} = \sqrt{14} = 3.74165738677394, \\ \|U\|_\infty &= \max\{|3| + |2|, |-1| + |0|\} = 5, \\ \|U\|_1 &= \max\{|3| + |-1|, |2| + |0|\} = 4. \end{aligned}$$

**Stability analysis**

Consider $Ax = b$, $A$ invertible. Let $x(\varepsilon)$ be the solution of the linear system

$$(A + \varepsilon F)x(\varepsilon) = (b + \varepsilon f).$$

We are intersted in the case when $\varepsilon$ tends to zero. Here $f \in \mathbf{R}^n$ and $F$ is $n \times n$ matrix. We seek for bounds for the relative error

$$\frac{\|x - x(\varepsilon)\|}{\|x\|},$$

for a chosen vector-norm $\|\cdot\|$. We will also make use of the so called subordinate matrix-norm deduced from $\|\cdot\|$, that is for a given $n \times n$ matrix $B$:

$$\|B\| := \max_{x \neq 0,\, x \in \mathbf{R}^n} \frac{\|Bx\|}{\|x\|}.$$

**Proposition 2.1.** *Assume $A$ is invertible. For all $\varepsilon$ such that $\varepsilon \leq C_\varepsilon$, $A+\varepsilon F$ is also invertible.*

**Proposition 2.2.** *For all $\varepsilon$ such that $\varepsilon \leq \tilde{C}_\varepsilon$, here exists a unique $x(\varepsilon)$ and its components, $x_i(\varepsilon)$ are continuous functions of $\varepsilon$.*

**Lemma 2.3.**
$$\frac{d}{d\varepsilon} x(\varepsilon)\bigg|_{\varepsilon=0} = A^{-1}(f - Fx).$$

*Proof.* By differentiation.

We now use Taylor theorem and obtain

$$x(\varepsilon) = x(0) + \varepsilon x'(0) + \mathcal{O}(\varepsilon^2),$$

where $x(0) = x$. So
$$\|x(\varepsilon) - x\| \leq \varepsilon \|x'(0)\| + \mathcal{O}(\varepsilon^2),$$

and using the lemma, we obtain

$$\|x(\varepsilon) - x\| \leq \varepsilon \|A^{-1}(f - Fx)\| + \mathcal{O}(\varepsilon^2).$$

This leads to

$$\frac{\|x(\varepsilon) - x\|}{\|x\|} \leq \varepsilon \frac{\|A^{-1}(f - Fx)\|}{\|x\|} + \mathcal{O}(\varepsilon^2) \leq \varepsilon \|A^{-1}\| \left( \frac{\|f\|}{\|x\|} + \frac{\|Fx\|}{\|x\|} \right) + \mathcal{O}(\varepsilon^2).$$

Now $Ax = b$ implies $\|b\| \leq \|A\|\|x\|$, and this is equivalent to

$$\frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|}.$$

We also have
$$\frac{\|Fx\|}{\|x\|} \leq \|F\| = \frac{\|F\|\|A\|}{\|A\|}.$$

So proceeding in the estimation of the relative error we get

$$\frac{\|x(\varepsilon)-x\|}{\|x\|} \leq \varepsilon \|A^{-1}\| \left( \frac{\|f\|\|A\|}{\|b\|} + \frac{\|F\|\|A\|}{\|A\|} \right) + \mathcal{O}(\varepsilon^2),$$

and we obtain

$$\frac{\|x(\varepsilon)-x\|}{\|x\|} \leq \varepsilon \|A^{-1}\|\|A\| \left( \frac{\|f\|}{\|b\|} + \frac{\|F\|}{\|b\|} \right) + \mathcal{O}(\varepsilon^2).$$

The real number

$$\mathcal{K}(A) := \|A^{-1}\| \, \|A\|$$

is called **condition number** of $A$. The condition number depends on $A$ and on the matrix-norm used to measure the relative error. The final bound of the relative error is then

$$\frac{\|x(\varepsilon)-x\|}{\|x\|} \leq \varepsilon \, \mathcal{K}(A) \left( \frac{\|f\|}{\|b\|} + \frac{\|F\|}{\|b\|} \right) + \mathcal{O}(\varepsilon^2),$$

and we clearly see that the condition number gives a bound of the leading error term (in $\varepsilon$) of the relative error in the output data by means of the relative error in the input data.
We observe that since $I = A^{-1}A$, then

$$\|I\| \leq \|A^{-1}\| \, \|A\| = \mathcal{K}(A),$$

and for all subordinate matrix-norms

$$\|I\| = \max_{x \neq 0, \, x \in \mathbf{R}^n} \frac{\|I\,x\|}{\|x\|} = 1,$$

so

$$1 \leq \mathcal{K}(A).$$

## 2.3 Gaussian elimination

**Definition**. Two linear systems of equations are said to be equivalent if they have the same solution.

Gaussian elimination is an algorithm where in $n-1$ steps, $Ax = b$ is transformed in an equivalent system $Ux = f$ and $U$ is an upper triangular matrix. The advantage is that triangular systems can be easily solved using the formula (9).
In particular we have

$$Ax = b \rightarrow A^{(1)}x = b^{(1)} \rightarrow \cdots \rightarrow A^{(k)}x = b^{(k)} \rightarrow \cdots \rightarrow A^{(n-1)}x = b^{(n-1)}$$

all intermediate linear systems $A^{(k)}x = b^{(k)}$ $k = 1, \ldots, n-1$ are equivalent to $Ax = b$. The last system $A^{(n-1)}x = b^{(n-1)}$, is in upper triangular form, and system number $k$ is

$$A^{(k)} = \begin{bmatrix} \tilde{a}_{1,1} & \cdots & \tilde{a}_{1,k} & \tilde{a}_{1,k+1} & \cdots & \tilde{a}_{1,n} \\ 0 & \ddots & \vdots & \vdots & \cdots & \vdots \\ \vdots & \ddots & \tilde{a}_{k,k} & \tilde{a}_{k,k+1} & \cdots & \tilde{a}_{k,n} \\ \vdots & \vdots & 0 & \tilde{a}_{k+1,k+1} & \cdots & \tilde{a}_{k+1,n} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & 0 & \tilde{a}_{n,k+1} & \cdots & \tilde{a}_{n,n} \end{bmatrix}.$$

We obtain $A^{(1)}x = b^{(1)}$ from $Ax = b$ by replacing the den 2nd, 3rd, .., nth equation in $Ax = b$ with corresponding linear combinations of the first equation with the 2nd, 3rd, .., nth equation. To obtain $A^{(2)}x = b^{(2)}$ (and the subsequent linear systems) the same process is repeated considering only prosessen er repetert med å ta hensyn til kolonnene fra the columns form the 2nd to the nth and the rows from the 2nd to the nth.

**Example**

Given:

$$\begin{aligned} x_1 + 4x_2 + x_3 &= 6 \\ 2x_1 - x_2 - 2x_3 &= 3 \\ x_1 + 3x_2 + 2x_3 &= 5 \end{aligned} \qquad A = \begin{bmatrix} 1 & 4 & 1 \\ 2 & -1 & -2 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 5 \end{bmatrix} \tag{10}$$

we want to reduce it t a triangular form.
We start with replacing the second equation with a linear combination of the first two equations, that is we replace
$2x_1 - x_2 - 2x_3 = 3$ with

$$(2x_1 - x_2 - 2x_3) + (-2) \cdot (x_1 + 4x_2 + x_3) = 3 + (-2) \cdot 6, \Rightarrow -9x_2 - 4x_3 = -9.$$

We then replace the 3rd equation with a linear combination of the 3rd and 1st equation:

$$(x_1 + 3x_2 + 2x_3) + (-1) \cdot (x_1 + 4x_2 - x_3) = 5 + (-1) \cdot 6, \Rightarrow -x_2 + x_3 = -1.$$

This way we get the following new system

$$\begin{aligned} x_1 + 4x_2 + x_3 &= 6 \\ -9x_2 - 4x_3 &= -9 \\ -x_2 + x_3 &= -1 \end{aligned} \qquad A^{(1)} = \begin{bmatrix} 1 & 4 & 1 \\ 0 & -9 & -4 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -9 \\ -1 \end{bmatrix} \tag{11}$$

The coefficients used in the linear combination of the equations are $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$, and they are chosen so that in the new system we get that the second and third element of the first column vanish, this way we *eliminate* to coefficients in the linear system.
Now we work only with the two last equations:

$$\begin{aligned} -9x_2 - 4x_3 &= -9 \\ -x_2 + x_3 &= -1 \end{aligned} \tag{12}$$

We replace the last equaltion with

$$(-x_2 + x_3) + (-\frac{1}{9}) \cdot (-9x_2 - 4x_3) = -1 + \left(-\frac{1}{9}\right) \cdot (-9) \Rightarrow \frac{13}{9}x_3 = 0.$$

The coefficient used for the linear combination is $\frac{1}{9}$.
In the end we get the linear system

$$\begin{aligned} x_1 + 4x_2 + x_3 &= 6 \\ -9x_2 - 4x_3 &= -9 \\ \tfrac{13}{9}x_3 &= 0 \end{aligned} \qquad A^{(2)} = \begin{bmatrix} 1 & 4 & 1 \\ 0 & -9 & -4 \\ 0 & 0 & \frac{13}{9} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -9 \\ 0 \end{bmatrix} \tag{13}$$

and the solution can be computed by the backward substitution algorithm. Starting form the last equation $x_3 = 0$, and proceeding upwards to solve $-9x_2 = -9 \Rightarrow x_2 = 1$ og $x_1 + 4 = 6 \Rightarrow x_1 = 2$, we get

$$x = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix}.$$

Note now that when we *eliminated* the first column we used the two coefficients:

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix},$$

and for the second we used

$$\frac{1}{9}.$$

By using these coefficients we construct a triangular matrix $L$:

$$L := \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & \frac{1}{9} & 1 \end{bmatrix}.$$

Note also that if from (13) we define now $U := A^{(2)}$ and we compute $L \cdot U$, we get $A$ back.

In general when we perform the Gaussian elimination, we compute simultaneously a factorization of the matrix $A = LU$, where $L$ and $U$ are two triangular matrices.

This is very handy if we want to compute solution of two or more systems with the same coefficient matrix $A$ but different right hand sides, $b$, $\hat{b}$ and so on. In this case one can use the same factorization $LU = A$ two (or more) times ganger and compute to different backward substitutions, one with $b$ and one with $\hat{b}$.

Such factorization is also used for computing the determinant of $A$, because $\det(A) = \prod_{i=1}^{n} u_{i,i}$. Analogously one can use the factorization to find the inverse $A^{-1}$. In our case we have

$$\det(A) = 1 \cdot (-9) \cdot \frac{13}{9} = -13$$

and

$$A^{-1} = \begin{bmatrix} 1 & 4 & 1 \\ 0 & -9 & -4 \\ 0 & 0 & \frac{13}{9} \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & \frac{1}{9} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 1 & \frac{4}{9} & \frac{7}{13} \\ 0 & -\frac{1}{9} & -\frac{4}{13} \\ 0 & 0 & \frac{9}{13} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -\frac{7}{9} & -\frac{1}{9} & 1 \end{bmatrix}$$

Note that it is much easier to compute the inverse of a triangular matrix than the inverse of a general invertible but unstructured matrix.

**Gauss elimination: general algorithm**

For the general matrix (7) we have:

$$Ax = b \rightarrow A^{(1)}x = b^{(1)}$$

if $a_{1,1} \neq 0$

$$\left.\begin{array}{lllllll} l_{2,1} & := & \frac{a_{2,1}}{a_{1,1}} & a_{2,p}^{(1)} & := & a_{2,p} - l_{2,1}a_{1,p} & b_2^{(1)} := b_2 - l_{2,1}b_1 \\ l_{3,1} & := & \frac{a_{3,1}}{a_{1,1}} & a_{3,p}^{(1)} & := & a_{3,p} - l_{3,1}a_{1,p} & b_3^{(1)} := b_3 - l_{3,1}b_1 \\ \vdots & & & & & & \\ l_{n,1} & := & \frac{a_{n,1}}{a_{1,1}} & a_{n,p}^{(1)} & := & a_{n,p} - l_{n,1}a_{1,p} & b_n^{(1)} := b_n - l_{n,1}b_1 \end{array}\right\} p = 2, \dots, n.$$

Now we define

$$A := A^{(1)}, b := b^{(1)}$$

and continue with

$$Ax = b \to A^{(2)}x = b^{(2)}.$$

If $a_{2,2} \neq 0$, we obtain, for $j = 3, \dots, n$,

$$l_{j,2} \;:=\; \frac{a_{j,2}}{a_{2,2}} \quad a_{j,p}^{(2)} \;:=\; a_{j,p} - l_{j,2}a_{2,p} \quad b_j^{(2)} := b_j - l_{j,2}b_2 \quad p = 3, \dots, n,$$

and then we define

$$A := A^{(2)}, b := b^{(2)}.$$

In general for $A^{(k)}$ we have,

$$A := A^{(k-1)}, b := b^{(k-1)}$$

and

$$Ax = b \to A^{(k)}x = b^{(k)}$$

with

$$l_{j,k} \;:=\; \frac{a_{j,k}}{a_{k,k}} \quad a_{j,p}^{(k)} \;:=\; a_{j,p} - l_{j,k}a_{k,p} \quad b_j^{(k)} := b_j - l_{j,k}b_k \quad p = k+1, \dots, n.$$

for $j = k+1, \dots n$, and assuming $a_{k,k} \neq 0$. In the end we obtain the following algorithm:

**Gaussian elimination**

```
For $k = 1, \dots, n-1$
    For $j = k+1, \dots, n$
        $l_{j,k} := \frac{a_{j,k}}{a_{k,k}}$
        For $p = k+1, \dots, n+1$,
            $a_{j,p} := a_{j,p} - l_{j,k} a_{k,p}$
        End
    End
End
```

And with $U := A^{(n-1)}$ we also obtain the following, so called $LU$**-factorization** for $A$,

$$A = LU.$$

**Gaussian elimination with partial pivoting**

In Gaussisan elimination (G-E), as described on the general algorithm, we divide always by $a_{k,k}$ (the so called pivot element). Obviously we might get problems when such value is zero or very small. To avoid such problems we can perform systematic permutationns of the rows of the linear system. This procedure is called partial pivoting.

**Two exmples: Gaussian elimination with partial pivoting**

Consider the linear system

$$\begin{array}{rcl} x_1 + x_2 + x_3 + x_4 &=& 1 \\ x_1 + x_2 + 2x_3 - x_4 &=& 1 \\ x_1 + 2x_2 - x_3 - x_4 &=& 1 \\ x_1 - x_2 + x_3 - x_4 &=& 1 \end{array} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & -1 \\ 1 & 2 & -1 & -1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{14}$$

We eliminate the first column by subtracting the first row from the other three. We obtain

$$\begin{array}{rcl} x_1 + x_2 + x_3 + x_4 &=& 1 \\ x_3 - 2x_4 &=& 0 \\ x_2 - 2x_3 - 2x_4 &=& 0 \\ -2x_2 - 2x_4 &=& 0 \end{array} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & -2 \\ 0 & 1 & -2 & -2 \\ 0 & -2 & 0 & -2 \end{bmatrix} \tag{15}$$

now we can not proceede, because there is no $\alpha$ which could be used to eliminate $x_2$

$$(x_2 - 2x_3 - 2x_4) - \alpha \cdot (x_3 - 2x_4) = 0.$$

The only option is to permute the rows. To minimize roundoff error propagation it pays off to exchange the second row with the row having the biggest coefficient in absolute value for $x_2$, that is the third. Then we get

$$\begin{array}{ccc} x_1 + x_2 + x_3 + x_4 & = & 1 \\ -2x_2 - 2x_4 & = & 0 \\ x_2 - 2x_3 - 2x_4 & = & 0 \\ x_3 - 2x_4 & = & 0 \end{array} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \\ 0 & 1 & -2 & -2 \\ 0 & 0 & 1 & -2 \end{bmatrix} \tag{16}$$

and now we continue the Gaussian elimination as usual. We obtain:

$$\begin{array}{ccc} x_1 + x_2 + x_3 + x_4 & = & 1 \\ -2x_2 - 2x_4 & = & 0 \\ -2x_3 - 3x_4 & = & 0 \\ x_3 - 2x_4 & = & 0 \end{array} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & -3 \\ 0 & 0 & 1 & -2 \end{bmatrix} \tag{17}$$

and in the end:

$$\begin{array}{ccc} x_1 + x_2 + x_3 + x_4 & = & 1 \\ -2x_2 - 2x_4 & = & 0 \\ -2x_3 - 3x_4 & = & 0 \\ -\frac{7}{2}x_4 & = & 0 \end{array} \qquad \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & -2 \\ 0 & 0 & -2 & -3 \\ 0 & 0 & 0 & -\frac{7}{2} \end{bmatrix} \tag{18}$$

which, by the backward substitution algorithm, gives the following solution

$$x = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

In the second example given:

$$\begin{array}{ccc} x_1 + 3x_2 + 2x_3 & = & 5 \\ 2x_1 - x_2 - 2x_3 & = & 3 \\ x_1 + 4x_2 + x_3 & = & 6 \end{array} \qquad A = \begin{bmatrix} 1 & 3 & 2 \\ 2 & -1 & -2 \\ 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ 6 \end{bmatrix} \tag{19}$$

we want to reduce the system to triangular form using Gaussian elimination with partial pivoting.

We exchange rows:

$$\begin{array}{ccc} 2x_1 - x_2 - 2x_3 & = & 3 \\ x_1 + 3x_2 + 2x_3 & = & 5 \\ x_1 + 4x_2 + x_3 & = & 6 \end{array} \tag{20}$$

then we eliminate $x_1$,

$$\begin{array}{lcc} 2x_1 - x_2 - 2x_3 & = & 3 \\ 3.5x_2 + 3x_3 & = & 3.5 \\ 4.5x_2 + 2x_3 & = & 4.5 \end{array} \tag{21}$$

we exchange rows once more:

$$\begin{array}{lcc} 2x_1 - x_2 - 2x_3 & = & 3 \\ 4.5x_2 + 2x_3 & = & 4.5 \\ 3.5x_2 + 3x_3 & = & 3.5 \end{array} \tag{22}$$

and eliminate $x_2$ from the last equation,

$$\begin{array}{lcr} 2x_1 - x_2 - 2x_3 & = & 3 \\ 4.5x_2 + 2x_3 & = & 4.5 \\ 1.4444x_3 & = & 0 \end{array} \tag{23}$$

with the backward substitution algorithm we obtain the solution $x_3 = 0$, $x_2 = 1$, $x_1 = 2$.

### G-E with partial pivoting: general algorithm

In general we get the following Gaussian elimination algorithm with partial pivoting: first one initializes the vector $\pi$ (pivot vektor) so that $\pi_i := i$ for $i = 1, \ldots n$,

**Gaussian elimination with partial pivoting**

```
For $k = 1, \dots, n-1$
$a := |a_{k,k}|$
    For $j = k+1, \dots, n$
        if ( $a < |a_{j,k}|$ )
            $a := |a_{j,k}|$
            $\pi_k := j$
        End
    End
    if ( $\pi_k \neq k$ )
        $s := \pi_k$
        For $p = k, \dots, n$
            $r := a_{k,p}$
            $a_{k,p} := a_{s,p}$
            $a_{s,p} := r$
        End
    End
    For $j = k+1, \dots, n$
        $l_{j,k} := \frac{a_{j,k}}{a_{k,k}}$
        For $p = k+1, \dots, n+1,$
            $a_{j,p} := a_{j,p} - l_{j,k} a_{k,p}$
        End
    End
End
```

**Complexity of Gaussian elimination**

One can prove that for a $n \times n$ matrix the complexity of Gaussian elimination is of

$$\frac{1}{3}n^3 - \frac{1}{3}n,$$

operations (additions and multiplications). For big $n$ $\frac{1}{3}n^3$ dominates the cost. We say that Gaussian elimination has complexity $\mathcal{O}(n^3)$. The backward substitution algorithm, (9)) has lower cost, that is $\mathcal{O}(n^2)$.

## 2.4 Other matrix factorizations

Bisides the $LU$-factorizations there are many other important matrix factorizations which is useful to know about.

Recall that an eigenvalue of $A$ is a real or complex value such that, there is $u$ vector such that

$$Au = \lambda u.$$

$u$ is called eigenvector of $A$.

1. ***QR*- factorization**: any real matrix $A$ $n \times p$ can be factorized in the form

   $$A = QR$$

   where $Q$ is $n \times n$ orthogonal (i.e. $Q^TQ = QQ^T = I$ where $I$ is the identity matrix) and $R$ is $n \times p$ is upper triangular.

2. **Polar decomposition**: any matrix $A$ $n \times n$ can be factorized in the form

   $$A = QS$$

   where $Q$ is $n \times n$ orthogonal and $S$ is $n \times n$ is symmetric.

3. **Schur canonical form**: for any matrix $A$ $n \times n$ there exists a matrix $P$ (complex) unitary (i.e. $P^HP = PP^H = I$ and $P^H$ the transpose-conjugate of $P$) such that

   $$P^HAP = T$$

   where $T$ is upper triangular.

   As a consequence, if $A$ is Hermitian (i.e. $A = A^H$) then

   $$P^HAP = P^HA^HP = T$$

   and $T$ is Hermitian and triangular and therefore is diagonal.

4. **Singular value decomposition**. For all $A$ $n \times n$ real matrices, $A$ can be factorized as

   $$A = U\Sigma V^T$$

   where $\Sigma$ is diagonal, $U$ and $V$ are $n \times n$ orthogonal matrices. The diagonal elements of $\Sigma$ are the singular values of $A$, i.e. the square roots of the eigenvalues of $A^TA$. This factorization has analogs for $n \times p$ matrices and for matrices with complex entries.

5. **Jordan canonical form**. For any $A$ real $n \times n$ (or $A \in \mathbb{C}^{n\times n}$) it exists a matrix $M \in \mathbb{C}^{n\times n}$ invertible, such that

$$M^{-1}AM = J = \begin{bmatrix} J_1 & & & \\ & J_2 & & \\ & & \ddots & \\ & & & J_k \end{bmatrix}, \qquad \text{(block-diagonal).} \tag{24}$$

Here $J_i$ is a $m_i \times m_i$-matrix, and $\sum_{i=1}^{k} m_i = n$. The *Jordan-blocks* $J_i$ have the form

$$J_i = \begin{bmatrix} \lambda_i & 1 & & \\ & \lambda_i & \ddots & \\ & & \ddots & 1 \\ & & & \lambda_i \end{bmatrix}, \quad \text{if } m_i \geq 2$$

and $J_i = [\lambda_i]$ if $m_i = 1$. If all $m_i = 1$, then $k = n$ and the matrix is diagonalizable. If $A$ has $n$ distinct egenvalues, it is always diagonalizable. The converse is not true, that is a matrix can be diagonalizable even if it has multiple eigenvalues.

## 2.5 Symmetric matrices

When we talk about symmetric matrices, we mean normally *real* symmetric matrices. The *transpose* $A^T$ of a $m \times n$-matrix $A$, is a $n \times m$-matrix with $a_{ji}$ as the $(ij)$-element (a matrix whose columns are the rows of $A$). A $n \times n$ matrix is symmetric if $A^T = A$.

A symmetric $n \times n$ matrix has real eigenvalues $\lambda_1, \ldots, \lambda_n$ and a set of real orthonormal eigenvectors $x_1, \ldots, x_n$. Let $\langle \cdot, \cdot \rangle$ denote the standard inner-product on $\mathbb{C}^n$, then $\langle x_i, x_j \rangle = \delta_{ij}$ (Kronecker-delta).

A consequence of this is that the matrix of eigenvectors $X = [x_1, \ldots, x_n]$ is real and orthogonal and its inverse is therefore the transpose

$$X^{-1} = X^T.$$

The diagonalization of $A$ is given by

$$\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n), \quad X = [x_1, \ldots, x_n], \quad X^TX = I, \quad X^TAX = \Lambda \Leftrightarrow A = X\Lambda X^T$$

### Positive definite matrices

If $A$ is symmetric and $\langle x, Ax \rangle = x^TAx > 0$ for all $0 \neq x \in \mathbb{R}^n$ $A$ is called *positive definite*. Here we denote with $\langle \cdot, \cdot \rangle$ the Euclidean inner product.

$A$ (symmetric) is positive semi-definite if $\langle x, Ax \rangle \geq 0$ for all $x \in \mathbb{R}^n$ and $\langle x, Ax \rangle = 0$ for at least a $x \neq 0$.

$A$ positive definite $\Leftrightarrow$ $A$ has only positive eigenvalues.

$A$ positive semi-definite $\Leftrightarrow$ $A$ has only non-negative eigenvalues, and at least a 0-eigenvalue.

## 2.6 Gershgorin's theorem

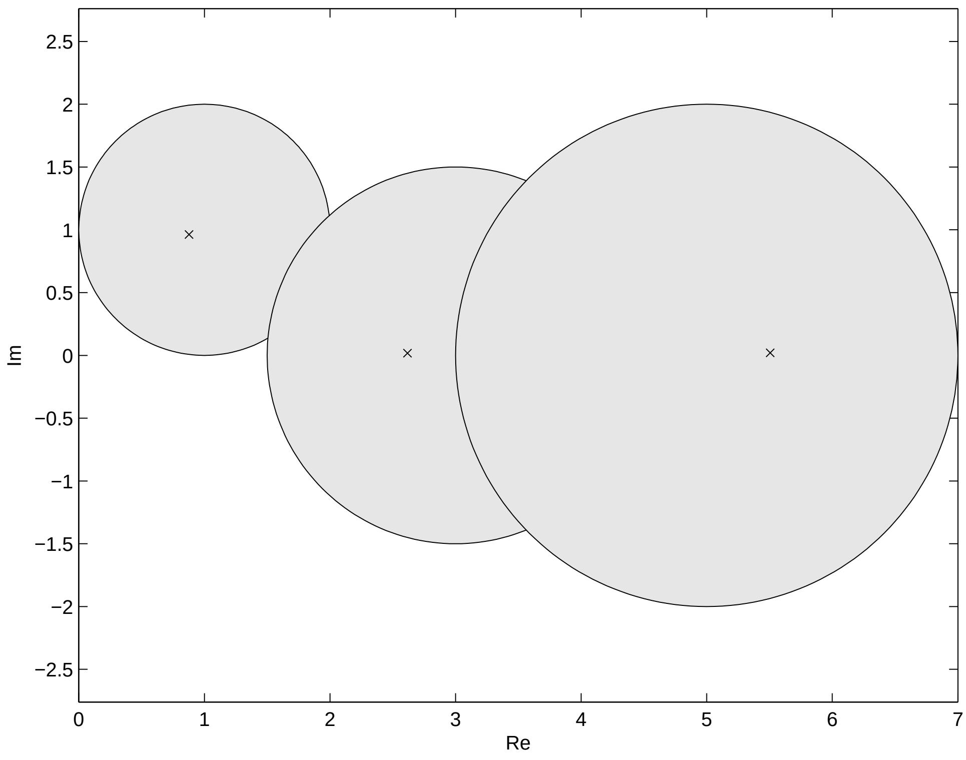
**Gershgorin's theorem.** Is given $A = (a_{ik}) \in \mathbb{C}^{n\times n}$. Define $n$ disks $S_j$ in the complex plane by

$$S_j = \left\{ z \in \mathbb{C} : |z - a_{jj}| \le \sum_{k\neq j} |a_{jk}| \right\}.$$

The union $S = \bigcup_{j=1}^{n} S_j$ contains all the eigenvalues of $A$. For every eigenvalue $\lambda$ of $A$ there is a $j$ such that $\lambda \in S_j$.

**Example.**

$$A = \begin{bmatrix} 1+i & 1 & 0 \\ 0.5 & 3 & 1 \\ 1 & 1 & 5 \end{bmatrix}.$$



□

*Proof of Gershgorin's theorem:* Let $\lambda$ be a eigenvalue with associate eigenvector $x = [\xi_1, \ldots, \xi_n]^T \neq 0$. Choose $\ell$ among the indexes $1, \ldots, n$ such that $|\xi_\ell| \ge |\xi_k|,\ k = 1, \ldots, n$, and so $|\xi_\ell| > 0$. The equation $Ax = \lambda x$ has component $\ell$:

$$\sum_{k=1}^{n} a_{\ell k}\xi_k = \lambda\,\xi_\ell \ \Rightarrow\ (\lambda - a_{\ell\ell})\xi_\ell = \sum_{k\neq\ell} a_{\ell k}\xi_k$$

Divide by $|\xi_\ell|$ on each side and take the absolute value

$$|\lambda - a_{\ell\ell}| \quad = \quad \left|\sum_{k\neq\ell} a_{\ell k}\frac{\xi_k}{\xi_\ell}\right| \quad \leq \quad \sum_{k\neq\ell}|a_{\ell k}|\frac{|\xi_k|}{|\xi_\ell|} \quad \leq \quad \sum_{k\neq\ell}|a_{\ell k}|$$

Then we get $\lambda \in S_\ell$.

**Example.** Diagonally dominant matrices with positive diagonal elements are positive definite. Why?
thm

## 2.7 Solution of linear systems by iteration

To approximate $x$ in the numerical solution of $Ax = b$, given $x^{(0)}$, we construct a sequence of vectors $x^{(1)}, \dots, x^{(n)}, \dots$. A way to do this is by fixed-point iteration.
We consider an equivalent formulation of $Ax = b$ as fix-point equaltion. For example:

$$Ax = b \Leftrightarrow x = (I - A)x + b$$

where $I$ is the $n \times n$ identity matrix. Given $x_0$, we then obtain the iteration:

$$x^{(n+1)} = (I - A)x^{(n)} + b.$$

In general one can obtain an iteration as follows:

- write $A$ as a sum of two terms: $A = M - N$, choose $M$ invertible;
- from $(M - N)x = b$ one gets $Mx = Nx + b$ and
$$x = M^{-1}Nx + M^{-1}b;$$
- so given $x^0$ one builds the iteration:
$$x^{(n+1)} = M^{-1}Nx^{(n)} + M^{-1}b.$$

Typically $M$ is chosen such that $M^{-1}$ is easy to compute, for example $M$ can be diagonal or triangular.

**Example**

We want to solve with fix-point-iteration the system

$$\begin{array}{rcl} x_1 - x_2 &=& 1 \\ -x_1 + 2x_2 &=& -1 \end{array} \quad A := \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \tag{25}$$

and start with

$$x^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The solution is

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

We take

$$M := \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \qquad N = M - A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}.$$

We have $Mx = Nx + b$, and $x = M^{-1}Nx + M^{-1}b$, i.e.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

By using $x^{(0)}$ we compute

$$x^{(1)} = M^{-1}Nx^{(0)} + M^{-1}b,$$

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{1}{2} \end{bmatrix}.$$

We continue $x^{(2)} = M^{-1}Nx^{(1)} + M^{-1}b$:

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} + \begin{bmatrix} 1 \\ -\frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} \\ 0 \end{bmatrix}$$

and more

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} 1 \\ -0.2500 \end{bmatrix}, \quad \begin{bmatrix} x_1^{(4)} \\ x_2^{(4)} \end{bmatrix} = \begin{bmatrix} 0.7500 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} x_1^{(5)} \\ x_2^{(5)} \end{bmatrix} = \begin{bmatrix} 1 \\ -0.1250 \end{bmatrix},$$

$$\begin{bmatrix} x_1^{(6)} \\ x_2^{(6)} \end{bmatrix} = \begin{bmatrix} 0.8750 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} x_1^{(28)} \\ x_2^{(28)} \end{bmatrix} = \begin{bmatrix} 1.0000 \\ -0.0000 \end{bmatrix}.$$

**Example**

In the system (25) we take

$$M := \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix} \qquad N = M - A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

and with the same $x^{(0)}$ we compute $x^{(1)} = M^{-1}Nx^{(0)} + M^{-1}b$:

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

and we get the true solution of the system already at the first iteration.

In general if $M$ is the diagonal part of $A$, i.e.

$$m_{i,i} = a_{i,i}, \quad i = 1, \ldots, n, \quad m_{i,j} = 0, \quad i \neq j,$$

(where $m_{i,j}$ are the elements of $M$, and $a_{i,j}$ are the elements of $A$), then we obtain the so called **Jacobi method**.

If $M$ is the lower triangular part of $A$, i.e.

$$m_{i,j} = a_{i,j}, \quad i = 1, \ldots, n, \ j = 1, \ldots, i, \quad m_{i,j} = 0, \quad i = 1, \ldots, n, \ j = i+1, \ldots, n,$$

then the method is called **Gauss-Seidel method**.

**Convergence**

To measure the extent to which $x^n$ has converged to $x$ we use vector norms: $\|x - x^{(n)}\|$.

We write the iteration in the following general way

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

by defining $C := M^{-1}N$ and $g := M^{-1}b$ we can write:

$$x^{(n+1)} = Cx^{(n)} + g. \tag{26}$$

**Theorem 2.4.** *If there is a matrix-norm $\|\cdot\|$ such that $\|C\| < 1$ the iteration (26) converges for all $x^{(0)}$.*

**Example**

Consider

$$B = \begin{bmatrix} 3 & 1 \\ -1 & 2.5 \end{bmatrix}, M = \begin{bmatrix} 3 & 0 \\ 0 & 2.5 \end{bmatrix}, N = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

then

$$C_J = M^{-1}N = \begin{bmatrix} 0 & -0.3333 \\ 0.4000 & 0 \end{bmatrix}$$

and $\|C_J\|_F = 0.5207$, $\|C_J\|_1 = \|C_J\|_{\max} = 0.4000$, so given $f$, the Jacobi method converges for $Bx = f$ for any $x^{(0)}$.
For the Gauss-Seidel method we have

$$C_{GS} = \begin{bmatrix} 3 & 0 \\ -1 & 2.5 \end{bmatrix}^{-1} \begin{bmatrix} 0 & -1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -0.3333 \\ 0 & -0.1333 \end{bmatrix}$$

and $\|C_{GS}\|_F = 0.3590$, $\|C_{GS}\|_1 = 0.4667$ and $\|C_{GS}\|_{\max} = 0.3333$, so Gauss-Seidel method converges for $Bx = f$ and for any $x^{(0)}$.

**Example**

Consider the matrix from the example (25):

$$A := \begin{bmatrix} 1 & -1 \\ -1 & 2 \end{bmatrix},$$

Both Jacobi and Gauss-Seidel converge. For the Jacobi method we have:

$$C_J = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ \frac{1}{2} & 0 \end{bmatrix}$$

with norms $\|C_J\|_F \geq 1$ $\|C_J\|_{\max} \geq 1$ $\|C_J\|_1 \geq 1$. For Gauss-Seidel

$$C_{GS} = \begin{bmatrix} 1 & 0 \\ -1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \frac{1}{2} \end{bmatrix}$$

and $\|C_{GS}\|_F \geq 1$ $\|C_{GS}\|_{\max} \geq 1$ $\|C_{GS}\|_1 \geq 1$.
Here the hypothesis of the previous theorem are not satisfied, even if in our numerical experiments the iteration converges. We should use another technique to prove convergence. Let $\sigma(C)$ be the set of eigenvalues of $C$, then we can define

$$\rho(C) := \max_{\lambda \in \sigma(C)} |\lambda|$$

$\rho(C)$ is named spectral radius of $C$, and it is a positive number.
One can show that

**Theorem 2.5.** $\rho(C) < 1$ *if and only if the iterative method (26) converges for all* $x^{(0)}$.

In our example we must look at the eigenvalues of $C_J$ and $C_{GS}$. Note that

$$Cu = \lambda u \Leftrightarrow (C - \lambda I)u = 0 \Leftrightarrow \det(C - \lambda I) = 0,$$

where $I$ is the identity matrix and

$$\det(U) = \det\left(\begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \end{bmatrix}\right) = u_{1,1} \cdot u_{2,2} - u_{1,2} \cdot u_{2,1}.$$

So

$$\det(C_J - \lambda I) = \det\left(\begin{bmatrix} 0 & 1 \\ \frac{1}{2} & 0 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix}\right) = \det\left(\begin{bmatrix} -\lambda & 1 \\ \frac{1}{2} & -\lambda \end{bmatrix}\right) = \lambda^2 - \frac{1}{2},$$

and $\lambda^J_{1,2} = \pm\sqrt{\frac{1}{2}} \approx \pm 0.7071$. So $\rho(C) = 0.7071$ and $\rho(C) < 1$.
For $C_{GS}$ we have

$$\det(C_J - \lambda I) = \det\left(\begin{bmatrix} -\lambda & 1 \\ 0 & \frac{1}{2} - \lambda \end{bmatrix}\right) = \lambda\left(\frac{1}{2} - \lambda\right),$$

and $\lambda^{GS}_1 = 0$ and $\lambda^{GS}_2 = \frac{1}{2}$ and $\rho(C_{GS}) = \frac{1}{2}$.
The reason why Gauss-Seidel converges faster than Jacobi is that $\rho(C_{GS}) \leq \rho(C_J)$.

## 3 Adaptive Simpson

We now want to find an approximation $Q(a, b)$ to the integral $I(a, b) = \int_a^b f(x)dx$ so that the error is below a given tolerance $\epsilon$, thus

$$|I(a, b) - Q(a, b)| \lesssim \epsilon,$$

To do this, we will need a strategy for

- estimating the error $I(a, b) - Q(a, b)$.
- Divivde the interval $[a, b]$ interval into subintervals $a = X_0 < X_1 < \cdots < X_M = b$ so that
$$|I(X_k, X_{k+1}) - Q(X_k, X_{k+1})| \lesssim \frac{X_{k+1} - X_k}{b - a}\,\epsilon.$$

And we will use Simpsons method as an example here, but the strategy is applicable for other methods as well.

## Error estimate

Simpsons formula is given by

$$S(a,b) = \frac{b-a}{6}\Big(f(a) + 4f(c) + f(b)\Big), \qquad c = \frac{a+b}{2}.$$

and the error is given by

$$I(a,b) - S(a,b) = -\frac{(b-a)^5}{2880} f^{(4)}(\xi), \qquad \xi \in (a,b).$$

Assume now that the interval $[a,b]$ is small, so that $f^{(4)}$ is almost constant over this interval. In that case, the formula above becomes

$$I(a,b) - S(a,b) \approx C(b-a)^5, \qquad C \approx -\frac{f^{(4)}}{2880} \tag{27}$$

Next, let us split the interval in two equal parts: $[a,c]$ and $[c,b]$ and integrate each subinterval by Simpsons method:

$$S_2 = S(a,c)+S(c,b) = \frac{(b-a)}{2\cdot 6}\Big(f(a)+4f(d)+2f(c)+4f(e)+f(b)\Big), \qquad d = \frac{a+c}{2},\ e = \frac{c+b}{2}$$

and the error is

$$I(a,b) - S_2 = -\frac{(b-a)^5}{2^5\cdot 2880} f^{(4)}(\xi_1) - \frac{(b-a)^5}{2^5\cdot 2880} f^{(4)}(\xi_2) \approx C\cdot\frac{1}{2^4}(b-a)^5 \tag{28}$$

For notational convenience, use $S_1 = S(a,b)$. Multiplying (28) by 16, and subtract (27) from it gives

$$15\cdot I(a,b) - 16S_2 + S_1 \approx 0 \qquad \Rightarrow \qquad I(a,b) \approx S_2 + \frac{S_2 - S_1}{15}. \tag{29}$$

So, we can of course use this as a better approximation to the integral $I(a,b)$, but we can also use it for error estimations of $S_1$ and $S_2$, since clearly

$$I(a,b) - S_1 \approx E_1 = \frac{16}{15}(S_2 - S_1) \tag{30}$$

$$I(a,b) - S_2 \approx E_2 = \frac{1}{15}(S_2 - S_1) \tag{31}$$

In a code, we will use $S_2$, since this is the best of the two approximations and we do have an error estimate for it. But since this error estimate is known, we can add it to $S_2$ and get an even better approximation to I(a,b) for free, that is, we use the approximation (29). In that case the error is usually overestimated, we get a better solution than we asked for. Which is hardly any reason to complain.

**Example 3.1.** *We want to approximate*

$$I(0,\pi/2) = \int_0^{\pi/2} \sin(x)dx = 1.$$

*We get*

$$S_1 = S(0, \pi/2) = 1.00228088$$
$$S_2 = S(0, \pi/4) + S(\pi/4, \pi/2) = 1.00013458.$$

*The error esimates are*

$$E_1 = -2.28 \cdot 10^{-3}, \qquad E_2 = -1.43 \cdot 10^{-4}$$

*The improved approximation is given by:*

$$Q = S2 + E2 = 0.9999916.$$

*So, even in this simple example where $f^{(4)}$ is far from constant over the interval, the error estimates are quite accurate.*

### Splitting of the interval

The algorithm is quite trivial: First, calculate $S_1$, $S_2$ and $E_2$ over the whole $[a, b]$. If $|E_2| \geq \epsilon$ then divide the interval in two equal parts, and repeat the procedure on each subinterval with half of the tolerance on each subinterval. Continue until the error is below the tolerance for the given subinterval. The strategy may be best illustrated by an example:

**Example 3.2.** *We want to approximate* $\int_0^{\pi/2} \sin x dx$ *with an error tolerance* $\epsilon = 10^{-6}$. *The results are given in the following table:*

| *level* | $a = 0,\ b = \pi/2,\ \epsilon = 10^{-5}$ | | |
|---|---|---|---|
| 1 | $S_1 = 1.00228088, S_2 = 1.00013458, \lvert E_2\rvert = 1.43 \cdot 10^{-4}$ | | |
| 2 | $a = 0,\ b = \pi/4,\ \epsilon = 5 \cdot 10^{-6}$ | $a = \pi/4,\ b = \pi/2,\ \epsilon = 5 \cdot 10^{-6}$ | |
| | $S_1 = 0.29293264,\ S_2 = 0.29289564$ $\lvert E_2\rvert = 2.47 \cdot 10^{-6}, Q(0, \pi/4) = 0.29289318$ | $S_1 = 0.70720195,\ S_2 = 0.70711265$ $\lvert E_2\rvert = 5.96 \cdot 10^{-6}$ | |
| 3 | | $a = \pi/4,\ b = 3\pi/8$ $\epsilon = 2.5 \cdot 10^{-6}$ | $a = 3\pi/8,\ b = \pi/2$ $\epsilon = 2.5 \cdot 10^{-6}$ |
| | | $S_1 = 0.32442604$ $S_2 = 0.32442352$ $\lvert E_2\rvert = 1.68 \cdot 10^{-7}$ $Q(\pi/2, 3\pi/4) = 0.32442335$ | $S_1 = 0.38268661$ $S_2 = 0.38268363$ $\lvert E_2\rvert = 1.98 \cdot 10^{-7}$ $Q(3\pi/4, \pi/2) = 0.38268343$ |

*and the numerical approximation is*

$$Q(0, \pi/4) + Q(\pi/4, 3\pi/8) + Q(3\pi/4, \pi/2) = 0.9999996.$$

*The error is about* $4 \cdot 10^{-8}$ *which is well below the tolerance.*

The implementation of the code is given in Figure 1.

## 4 Gauss quadrature and orthogonal polynomials.

The aim of this section is to construct "optimal" quadrature formulas. To be more specific, given the integral

$$I_w(f) = \int_a^b w(x) f(x) dx \tag{32}$$

in which $w(x)$ is a fixLed, positive function. We want to approximate this using a quadrature formula on the form

$$Q_w(f) = \sum_{i=0}^{n} W_i f(x_i).$$

Such a formula can be constructed as follows: Choose $n + 1$ distinct nodes, $x_0, x_1, \cdots, x_n$ in the interval $[a, b]$. Construct the interpolation polynomial

$$p_n(x) = \sum_{i=0}^{n} f(x_i) L_i(x), \qquad L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}.$$

```
function simpson_result = simpson(f,a,b,tol,level,level_max)
%
% simpson_result = simpson(f,a,b,tol,0,level_max)
%
% Compute the integral of a function f from a to be within a tolerance tol,
% using an adaptiv Simpson method.
%
% level_max=10 is a suitable value.
%
level = level+1;
h = b-a;
c = (a+b)/2;
S1 = h*(f(a)+4*f(c)+f(b))/6;
d = (a+c)/2;
e = (c+b)/2;
S2 = h*(f(a)+4*f(d)+2*feval(f,c)+4*f(e) + f(b))/12;
if level>=level_max
    simpson_result = S2;
    warning('Maximum␣level␣reached');
else
    err = (S2-S1)/15;
    if abs(err)<tol
        simpson_result = S2+err;
    else
        left_simpson = simpson(f,a,c,tol/2,level,level_max);
        right_simpson = simpson(f,c,b,tol/2,level,level_max);
        simpson_result = left_simpson+right_simpson;
    end
end
```

Figure 1: Matlab code for adaptive Simpson

An approximation to the integral is then given by

$$Q_w(f) = \int_a^b w(x)p_{n-1}(x)dx = \sum_{i=0}^{m} W_i f(x_i), \qquad W_i = \int_a^b w(x)L_i(x)dx. \tag{33}$$

The quadrature formula is of precision $m$ if

$$I_w(p) = Q_w(p), \qquad \text{for all } p \in \mathbb{P}_m.$$

From the construction, these quadrature formulas is of precision at least $n$. The question is how too choose the nodes $x_i$, $i = 0, \ldots, n$ giving $m$ as large as possible. The key concept here is *orthogonal polynomials.*

## Orthogonal polynomials.

Given two functions $f, g \in C[a, b]$. We define an inner product of these two functions by

$$< f, g >_w = \int_a^b w(x)f(x)g(x)dx, \qquad w(x) > 0. \tag{34}$$

Thus the definition of the inner product depends on the integration interval $[a,b]$ and a given *weight function* $w(x)$. If $f,g,h \in C[a,b]$ and $\alpha \in \mathbb{R}$ then

$$\begin{aligned} \langle f,g\rangle_w &= \langle g,f\rangle_w \\ \langle f+g,h\rangle_w &= \langle f,h\rangle_w + \langle g,h\rangle_w \\ \langle \alpha f,g\rangle_w &= \alpha\langle f,g\rangle_w \\ \langle f,f\rangle_w &\geq 0, \qquad \text{and} \qquad \langle f,f\rangle_w = 0 \Leftrightarrow f \equiv 0. \end{aligned}$$

From an inner product, we can also define a norm on $C[a,b]$ by

$$\|f\|_w^2 = \langle f,f\rangle_w.$$

For the inner product (34) we also have

$$\langle xf,g\rangle_w = \int_a^b w(x)xf(x)g(x)dx = \langle f,xg\rangle_w. \tag{35}$$

Our aim is now to create an orthogonal basis for $\mathbb{P}$, that is, create a sequence of polynomials $\phi_k(x)$ of degree $k$ (no more, no less) for $k = 0,1,2,3,\dots$ such that

$$\langle \phi_i,\phi_j\rangle_w = 0 \quad \text{for all} \quad i \neq j.$$

If we can make such a sequence, then

$$\mathbb{P}_{n-1} = \text{span}\{\phi_0,\phi_1,\cdots,\phi_{n-1}\} \qquad \text{and} \qquad \langle \phi_n,p\rangle_w = 0 \quad \text{for all} \qquad p \in \mathbb{P}_{n-1}.$$

Let us now find the sequence of orthogonal polynomials. This is done by a Gram-Schmidt process:
Let $\phi_0 = 1$. Let $\phi_1 = x - B_1$ where $B_1$ is given by the orthogonality condition:

$$0 = \langle \phi_1,\phi_0\rangle_w = \langle x,1\rangle_w - B_1\langle 1,1\rangle_w \qquad \Rightarrow \qquad B_1 = \frac{\langle x,1\rangle_w}{\|1\|_w^2}.$$

Let us now assume that we have found $\phi_j$, $j = 0,1,\dots,k-1$. Then, let

$$\phi_k = x\phi_{k-1} - \sum_{j=0}^{k-1}\alpha_j\phi_j.$$

Clearly, $\phi_k$ is a polynomial of degree $k$, and $\alpha_j$ can be chosen so that $\langle \phi_k,\phi_i\rangle_w = 0$, $i = 0,1,\dots,k-1$, or

$$\langle \phi_k,\phi_i\rangle_w = \langle x\phi_{k-1},\phi_i\rangle_w - \sum_{j=0}^{k-1}\alpha_j\langle \phi_i,\phi_j\rangle_w = \langle x\phi_{k-1},\phi_i\rangle_w - \alpha_i\langle \phi_i,\phi_i\rangle_w = 0, \qquad i = 0,1,\cdots,k-1.$$

So $\alpha_i = \langle x\phi_{k-1},\phi_i\rangle_w / \langle \phi_i,\phi_i\rangle_w$. But we can do even better. Since $\phi_{k-1}$ is orthogonal to all polynomials of degree $k-2$ or less, we get

$$\langle x\phi_{k-1},\phi_i\rangle_w = \langle \phi_{k-1},x\phi_i\rangle_w = 0 \quad \text{for} \quad i+1 < k-1.$$

So, we are left only with $\alpha_{k-1}$ and $\alpha_{k-2}$. The following theorem concludes the argument:

**Theorem 4.1.** *The sequence of orthogonal polynomials can be defined as follows:*

$$\begin{aligned}&\phi_0(x) = 1, \qquad \phi_1(x) = x - B_1\\ &\phi_k(x) = (x - B_k)\phi_{k-1}(x) - C_k\phi_{k-2}(x), \qquad k \geq 2\end{aligned}$$

*with*

$$B_k = \frac{\langle x\phi_{k-1}, \phi_{k-1}\rangle_w}{\|\phi_{k-1}\|_w^2}, \qquad C_k = \frac{\langle x\phi_{k-1}, \phi_{k-2}\rangle_w}{\|\phi_{k-2}\|_w^2} = \frac{\|\phi_{k-1}\|_w^2}{\|\phi_{k-2}\|_w^2}$$

The last simplification of $C_k$ is given by:

$$\begin{aligned}\langle x\phi_{k-1}, \phi_{k-2}\rangle_w &= \langle \phi_{k-1}, x\phi_{k-2}\rangle_w\\ \phi_{k-1} &= x\phi_{k-2} - B_{k-1}\phi_{k-2} - C_{k-1}\phi_{k-3}.\end{aligned}$$

Solve the second with respect to $x\phi_{k-2}$, replace it into the right hand side of the first expression, and use the orthogonality conditions.

**Example 4.2.** *For the inner product*

$$\langle f, g\rangle = \int_{-1}^{1} f(x)g(x)dx$$

*we get*

$$\begin{array}{lllll}\phi_0 = 1, & \langle x\phi_0, \phi_0\rangle = 0, & \langle \phi_0, \phi_0\rangle = 2, & B_1 = 0, & \\ \phi_1 = x, & \langle x\phi_1, \phi_1\rangle = 0, & \langle \phi_1, \phi_1\rangle = \frac{2}{3}, & B_2 = 0, & C_2 = \frac{1}{3}\\ \phi_2 = x^2 - \frac{1}{3} & \langle x\phi_2, \phi_2\rangle = 0, & \langle \phi_2, \phi_2\rangle = \frac{8}{45}, & B_3 = 0, & C_3 = \frac{4}{15}\\ \phi_3 = x^3 - \frac{3}{5}x, & \textit{etc.} & & & \end{array}$$

*These are the well known Legendre polynomials.*

**Example 4.3.** *Let $w(x) = 1/\sqrt{1-x^2}$, and $[a, b] = [-1, 1]$. We then get the sequence of polynomials:*

$$\begin{array}{lllll}\phi_0 = 1, & \langle x\phi_0, \phi_0\rangle_w = 0, & \langle \phi_0, \phi_0\rangle_w = \pi, & B_1 = 0, & \\ \phi_1 = x, & \langle x\phi_1, \phi_1\rangle_w = 0, & \langle \phi_1, \phi_1\rangle_w = \frac{\pi}{2}, & B_2 = 0, & C_2 = \frac{1}{2}\\ \phi_2 = x^2 - \frac{1}{2} & \langle x\phi_2, \phi_2\rangle_w = 0, & \langle \phi_2, \phi_2\rangle_w = \frac{\pi}{2}, & B_3 = 0, & C_3 = \frac{1}{4}\\ \phi_3 = x^3 - \frac{3}{4}x, & \textit{etc.} & & & \end{array}$$

*These are nothing but the monic Chebyshev polynomials $\tilde{T}_k$.*

The following theorem will become useful:

**Theorem 4.4.** *Let $f \in C[a, b]$, $f \not\equiv 0$ satisfying $\langle f, p\rangle_w = 0$ for all $p \in \mathbb{P}_{k-1}$. Then $f$ changes signs at least $k$ times on $(a, b)$.*

*Proof.* By contradiction. Suppose that $f$ changes sign only $r < k$ times, at the points $t_1 < t_2 < \cdots < t_r$. Then $f$ will not change sign on each of the subintervals:

$$(a, t_1), (t_1, t_2), \cdots, (t_{r-1}, t_r), (t_r, b).$$

Let $p(x) = \prod_{i=1}^{r}(x - t_i) \in \mathbb{P}_r \subseteq \mathbb{P}_{k-1}$. Then $p(x)$ has the same sign properties as $f(x)$, and $f(x)p(x)$ does not change sign on the interval. Since $w > 0$ we get

$$\int_a^b w(x)f(x)p(x) \neq 0$$

which contradicts the assumption of the theorem. □

**Corollary 4.5.** *The orthogonal polynomial $\phi_k$ has exactly $k$ distinct zeros in $(a, b)$.*

## Gauss quadrature

We can now state the main result of this note:

**Theorem 4.6.** *Let $\phi_{n+1} \in \mathbb{P}_{n+1}$ be orthogonal to $\mathbb{P}_n$, that is*

$$\int_a^b w(x)\phi_{n+1}(x)p(x)dx = 0, \qquad \text{for all } p \in \mathbb{P}_n.$$

*Chose the nodes $x_i$, $i = 0, 1, \ldots, n$ to be the zeros of $\phi_{n+1}$, and let $Q(f)$ be the quadrature formula constructed by (33). Then $Q(f)$ is of precision $2n + 1$, that is*

$$\int_a^b w(x)p(x)dx = Q(p), \qquad \text{for all } p \in \mathbb{P}_{2n+1}. \tag{36}$$

*Proof.* Let $p \in \mathbb{P}_{2n+1}$. Divide $p$ by $\phi_{n+1}$ and let $r$ be the remainer term, thus

$$p = q\phi_{n+1} + r, \qquad q, r \in \mathbb{P}_n.$$

Since $\phi_{n+1}(x_i) = 0$, we get that $p(x_i) = r(x_i)$ for $i = 0, 1 \cdots, n$. But

$$\int_a^b w(x)p(x)dx = \overbrace{\int_a^b w(x)\phi_{n+1}(x)q(x)dx}^{=0} + \int_a^b w(x)r(x)dx = \sum_{i=0}^{n} W_i r(x_i)$$

and

$$\sum_{i=0}^{n} W_i p(x_i) = \sum_{i=0}^{n} W_i q(x_i)\overbrace{\phi_{n+1}(x_i)}^{=0} + \sum_{i=0}^{n} W_i r(x_i) = \sum_{i=0}^{n} W_i r(x_i)$$

which proves (36). □

Quadrature rules constructed by using the zeros of orthogonal polynomials as nodes are called Gauss quadratures.

For the error of the Gauss-quadratures, we have the following theorem:

**Theorem 4.7.** *The error of a Gauss quadrature is*

$$E(f) = \int_a^b w(x)f(x)dx - \sum_{i=0}^{n} W_i f(x_i) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\int_a^b w(x)\omega_{n+1}^2(x)dx$$

*where* $\xi \in (a,b)$ *and* $\omega_{n+1}(x) = \prod_{i=0}^{n}(x-x_i)$.

*Proof.* Recall two results from the earlier discussion on Hermite interpolation (see notes on divided differences, Theorem 3.9, to be found on It's learning.)
Given $n+1$ distinct nodes $x_i$, $i = 0,1,\dots,n$ there exist one and only one $p_{2n+1} \in \mathbb{P}_{2n+1}$ such that

$$f(x_i) = p_{2n+1}(x_i), \qquad f'(x_i) = p'_{2n+1}(x_i), \qquad i = 0,1,\dots,n.$$

Further, the error of this Hermite interpolation polynomial is

$$f(x) - p_{2n+1}(x) = \frac{f^{(2n+2)}(\eta(x))}{(2n+2)!}\omega_{n+1}^2(x) \tag{37}$$

It follows that

$$\int_a^b w(x)f(x)dx - \int_a^b w(x)\,p_{2n+1}(x)dx = \frac{1}{(2n+2)!}\int_a^b f^{(2n+2)}(\eta(x))\,w(x)\omega_{n+1}^2(x)dx \tag{38}$$

Since $p_{2n+1} \in \mathbb{P}_{2n+1}$ and the quadrature rule is of precision $2n+1$ (Theorem 4.6), we have that

$$\int_a^b w(x)\,p_{2n+1}(x)dx = \sum_{i=0}^{n} W_i\,p_{2n+1}(x_i) = \sum_{i=0}^{n} W_i f(x_i)$$

Finally, since $w(x) > 0$ and $\omega_{n+1}^2(x) \geq 0$ the mean value theorem for integrals can be applied to the right hand side of (38). $\square$

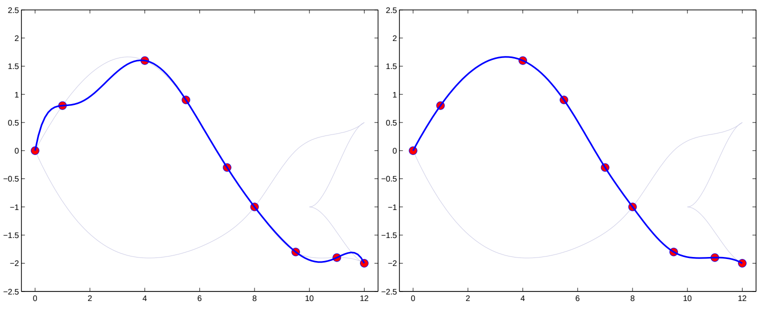
# 5 Note on Splines.

## 5.1 Introduction

Assume that we have a set of $n+1$ points $\{x_i, y_i\}_{i=0}^{n}$ and we want to find a curve interpolationg these points. One possibility is of course to use polynomial interplation, that is, find a polynomial $p_n \in \mathbb{P}_n$ so that

$$p_n(x_i) = y_i, \qquad i = 0,1,\dots,n.$$

This may be quite unsatisfactory, as the following picture demonstrate:



In the picture to the left, polynomial interpolation have been used, to the right, cubic splines.

The idea of splines is to split the interval $[a, b]$ by $a = t_0 < t_1 < \cdots < t_n = b$, and let interpolating curve be a polynomial on each subinterval $[t_{i-1}, t_i)$. The points $t_i$, $i = 0, 1, \ldots, n$ are called *knots* (skjøter på norsk), and they may or may not correspond to the interpolation nodes $x_i$. The piecewise polynomials are then glued together by some smoothness conditions. More formally, the definition is:

**Definition 5.1.** *On some interval $[a, b]$ , suppose that $n+1$ points $a = t_0 < t_1 < \cdots < t_n = b$ has been specified. A spline of degree $k$ is a function $S$ satisfying*

1. On each interval $[t_{i-1}, t_i)$, $S$ is a polynomial of degree $k$.
2. $S \in C^{(k-1)}[a, b]$.

We will write the spline by

$$S(x) \quad = \quad \begin{cases} S_0(x) & x \in [t_0, t_1) \\ S_1(x) & x \in [t_1, t_2) \\ \vdots & \\ S_{n-1}(x) & x \in [t_{n-1}, t_n] \end{cases} \tag{39}$$

where $S_i \in \mathbb{P}_k$.

**Example 5.2.** *The linear spline interpolating the the points $\{t_i, y_i\}_{i=0}^n$ is given by*

$$S_i(x) = y_i \frac{x - t_{i+1}}{t_i - t_{i+1}} + y_{i+1} \frac{x - t_i}{t_{i+1} - t_i}, \qquad x \in [t_i, t_{i+1}) \qquad i = 0, 1, \ldots, n-1, \tag{40}$$

*the straight lines between the points.*

## 5.2 Cubic splines

We will now construct an algorithm for finding the cubic splines, interpolating the points $\{t_i, y_i\}_{i=0}^n$. It means that

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \qquad x \in [t_i, t_{i+1}) \qquad i = 0, 1, \ldots, n-1$$

which gives a total of $4n$ parameters to be determined. A cubic spline is two times continuous differentiable, thus it has has to satisfy

$$S_i(t_i) = y_i, \qquad S_i(t_{i+1}) = y_{i+1}, \qquad i = 0, \cdots, n-1 \tag{41}$$

$$S'_{i-1}(t_i) = S'_i(t_i), \qquad i = 1, 2, \ldots, n-1 \tag{42}$$

$$S''_{i-1}(t_i) = S''_i(t_i), \qquad i = 1, 2, \ldots, n-1 \tag{43}$$

a total of $4n - 2$ conditions, leaving two free parameters. Some common choices for those are

- Natural cubic splines: $S''(t_0) = S''(t_n) = 0$.
- Clamped cubic splines: $S'(t_0)$ and $S'(t_n)$ are specified.

- Not-a-knot condition: $S_0'''(t_1) = S_1'''(t_1)$ and $S_{n-2}'''(t_{n-1}) = S_{n-1}'''(t_n)$.
- Periodic conditions: $S_0'(t_0) = S_{n-1}'(t_n)$ and $S_0''(t_0) = S_{n-1}''(t_n)$.

We will now construct an efficient algorithm for solving finding the splines. The idea is as follows: Since $S$ is a cubic spline, $S''$ is a linear spline. Let $z_i = S''(t_i)$, $i = 0, 1, \dots, n$ (to be found). Further, let $h_i = t_{i+1} - t_i$. Then, from 40 we have that

$$S_i''(x) = \frac{z_i}{h_i}(t_{i+1} - x) + \frac{z_{i+1}}{h_i}(x - t_i).$$

So, by this, (43) is satisfied. Integrating twice gives

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1} - x)^3 + \frac{z_{i+1}}{6h_i}(x - t_i)^3 + C_i x + D_i.$$

The integration constants $C_i$ and $D_i$ can be determined by (41), the result becomes

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1}-x)^3+\frac{z_{i+1}}{6h_i}(x-t_i)^3+\left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x-t_i)+\left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(t_{i+1}-x), \quad (44)$$

We now activate the second condition (42). Notice that

$$S_i'(t_i) = -\frac{h_i}{3}z_i - \frac{h_i}{6}z_{i+1} - \frac{y_i}{h_i} + \frac{y_{i+1}}{h_i}$$

and

$$S_{i-1}''(t_i) = \frac{h_i}{6}z_{i-1} + \frac{h_{i-1}}{3}z_i - \frac{y_{i-1}}{h_{i-1}} + \frac{y_i}{h_{i-1}}$$

so these conditions will simply become

$$h_{i-1}z_{i-1} + 2(h_i + h_{i-1})z_i + h_i z_{i+1} = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}), \qquad i = 1, \cdots, n-1.$$

Let us now assume $z_0 = z_n = 0$, the natural spline condition. The whole system now becomes a tridiagonal system of equations:

$$\begin{pmatrix} u_1 & h_1 & & & & \\ h_1 & u_2 & h_2 & & & \\ & h_2 & u_3 & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & h_{n-3} & u_{n-2} & h_{n-2} \\ & & & & h_{n-2} & u_{n-1} \end{pmatrix} \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_{n-2} \\ v_{n-1} \end{pmatrix}$$

with

$$h_i = t_{i+1} - t_i, \qquad u_i = 2(h_i + h_{i-1}), \qquad b_i = \frac{6}{h_i}(y_{i+1} - y_i), \qquad v_i = b_i - b_{i-1}.$$

Notice that the matrix is diagonal dominant, so the system can be solved by some direct methods for tridiagonal systems. The complete algortihm becomes:

For natural cubic splines, we do have the following result:

```
Input: $n,\ (t_i, y_i)_{i=0}^n$
  for $i = 0, 1, \ldots, n-1$ do
    $h_i \leftarrow t_{i+1} - t_i$
    $b_i \leftarrow 6(y_{i+1} - y_i)$
  end for
  $u_1 \leftarrow 2(h_0 + h_1)$
  $v_1 \leftarrow b_1 - b_0$
  for $i = 2, 3, \ldots, n-1$ do
    $u_i \leftarrow 2(h_i + h_{i-1}) - h_{i-1}^2/u_{i-1}$
    $v_i \leftarrow b_i - b_{i-1} - h_{i-1}v_{i-1}/u_{i-1}$
  end for
  $z_n \leftarrow 0$
  for $i = n-1, n-2, \ldots, 1$ do
    $z_i = (v_i - h_i z_{i+1})/u_i$
  end for
  $z_0 \leftarrow 0.$
```

**Theorem 5.3.** *Let $f \in C^2[a,b]$ . If $S$ is the natural cubic spline interpolating $f$ in the knots $a = t_0 < t_1 < \cdots < t_n = b$ then*

$$\int_a^b \big(S''(x)\big)^2 \, dx \leq \int_a^b \big(f''(x)\big)^2 \, dx.$$

*Proof.* Let $g = f - S$. Then

$$\int_a^b \big(f''(x)\big)^2 \, dx = \int_a^b \big(S''(x)\big)^2 \, dx + \int_a^b \big(g''(x)\big)^2 \, dx + 2\int_a^b g''(x)S''(x)dx.$$

The statement of the theorem is clearly true if we can prove that the last term is positive. Notice that $S_i'''$ is constant on each each interval $[t_i, t_{i+1})$, and let us call this constant $a_i$. By partial integration we get

$$\int_a^b S''g''dx = \sum_{i=0}^{n-1} \int_{t_i}^{t_{i+1}} S''g''dx = \sum_{i=0}^{n-1} \left\{ (S''(t_{i+1})g'(t_{i+1}) - S''(t_i)g'(t_i) - \int_{t_i}^{t_{i+1}} S'''g'dx \right\}$$
$$= S''(t_n)g'(t_n) - S''(t_0)g'(t_0) - \sum_{i=0}^{n-1} c_i \int_{t_i}^{t_{i+1}} g'dx = \sum_{i=0}^{n-1} c_i(g(t_{i+1}) - g(t_i)) = 0.$$

□

The *curvature* of a function $f$ is defined as $|f''|/\left(\sqrt{1+(f')^2}\right)^3$. If we assume that $|f'| \ll 1$ we are left with $f''$ as a approximate measure for the curvature. In this sense, the natural cubic spline is the smoothest possible function interpolating the given data.

# 6 Numerical solution of ordinary differential equations

## 6.1 Eulers method.

Let us start this introduction to the numerical solution of *ordinary differential equations* (ODEs) by something familiar. Given a scalar (one equation only) ODE

$$y' = f(t,y), \qquad t_0 \le t \le t_{end}, \qquad y(t_0) = y_0, \tag{45}$$

in which the function $f$, the integration interval $[t_0,t_{end}]$ and the initial value $y_0$ is assumed to be given. The *solution* of this initial value problem (IVP) is a function $y(t)$ on the interval $[t_0, t_{end}]$.

**Example 6.1.** *The ODE/IVP*

$$y' = -2ty, \qquad 0 \le t \le 1, \qquad y(0) = 1.0$$

*has as solution the function*

$$y(t) = e^{-t^2}.$$

But in many practical situations, it is not possible to express the solution $y(t)$ in closed form, even if a solution exist. In these cases, a numerical algorithm can give an *approximation* to the exact solution. Let us start with Eulers method, which should be known from some calculus classes. Divide the interval $[t_0, t_{end}]$ into $Nstep$ equal subintervals, each of size $h = (t_{end} - t_0)/Nstep$, and let $t_n = t_0 + nh$. Euler's method can be derived by several means. One possibility is to use the first few terms of the Taylor expansion of the exact solution, which is given by

$$y(t_0+h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2y''(t_0) + \cdots + \frac{1}{p!}h^py^{(p)}(t_0) + \frac{1}{(p+1)!}h^{p+1}y^{(p+1)}(\xi), \tag{46}$$

where $\xi$ is somewhere between $t_0$ and $t_{end}$. The integer $p \ge 1$ is a number of our own choice, but we have to require $y$ to be sufficiently differentiable, in this case that $y^{(p+1)}$ exist and is continuous. If $h$ is small, we may assume that the solution will be completely dominated by the first two terms, thus

$$y(t_0+h) \approx y(t_0) + hy'(t_0) = y_0 + hf(t_0,y_0),$$

and we call this approximate solution $y_1$. Starting from the point $t_1 = t_0 + h$ and $y_1$ we can repeat the process. We have now developed Euler's method, given by

$$y_{n+1} = y_n + hf(t_n,y_n), \qquad n = 0, 1, \cdots, Nsteps - 1,$$

resulting in approximations $y_n \approx y(t_n)$.

**Example 6.2.** *Eulers method with* $h = 0.1$ *applied to the ODE of Example 6.1 gives*

| $t_n$ | $y_n$ |
|---|---|
| 0.0 | 1.0000 |
| 0.1 | 1.0000 |
| 0.2 | 0.9800 |
| 0.3 | 0.9408 |
| 0.4 | 0.8844 |
| 0.5 | 0.8136 |
| 0.6 | 0.7322 |
| 0.7 | 0.6444 |
| 0.8 | 0.5542 |
| 0.9 | 0.4655 |
| 1.0 | 0.3817 |

.

*In this case we know the exact solution,* $y(1.0) = e^{-1.0^2} = 0.3679$ *and the error at the endpoint is* $e_{10} = y(1.0) - y_{10} = -1.38 \cdot 10^{-2}$*. If we repeat this experiment (write a MATLAB program to do so) with different stepsizes, and measure the error at the end of the interval, we get*

| $h$ | $e_{Nstep} = y(1.0) - y_{Nstep}$ |
|---|---|
| 0.1 | $-1.38 \cdot 10^{-2}$ |
| 0.05 | $-6.50 \cdot 10^{-3}$ |
| 0.025 | $-3.16 \cdot 10^{-3}$ |
| 0.0125 | $-1.56 \cdot 10^{-3}$ |

.

From this example, it might look like the error at the endpoint $e_{Nstep} \sim h$, where $h = (tend - t_0)/Nstep$. But is this true for all problems, and if yes, can we prove it? To do so, we need to see what kind of errors we have and how they behave. This is illustrated in Figure 2. For each step an error is made, and these errors are then propagated til the next steps and accumulate at the endpoint.

**Definition 6.3.** *The* local truncation error $d_{n+1}$ *is the error done in one step when starting at the exact solution* $y(t_n)$*. The* global error *is the difference between the exact and the numerical solution at point* $t_n$*, thus* $e_n = y(t_n) - y_n$*.*

The local truncation error of Euler's method is

$$d_{n+1} = y(t_n + h) - y(t_n) - hf(t_n, y(t_n)) = \frac{1}{2}h^2 y''(\xi),$$

where $\xi \in (t_n, t_{n+1})$. This is given from the Taylor-expansion of $y(t_n + h)$ around $t_n$ with $p = 1$. To see how the global error propagates from one step to the next, the trick is: We have

$$\begin{aligned} y(t_n + h) &= y(t_n) + hf(t_n, y(t_n)) + d_{n+1}, \\ y_{n+1} &= y_n + hf(t_n, y_n). \end{aligned}$$

Take the difference of these two, and get

$$e_{n+1} = e_n + h\left(f(t_n, y(t_n)) - f(t_n, y_n)\right) + d_{n+1} = (1 + hf_y(t_n, v))e_n + d_{n+1}, \tag{47}$$

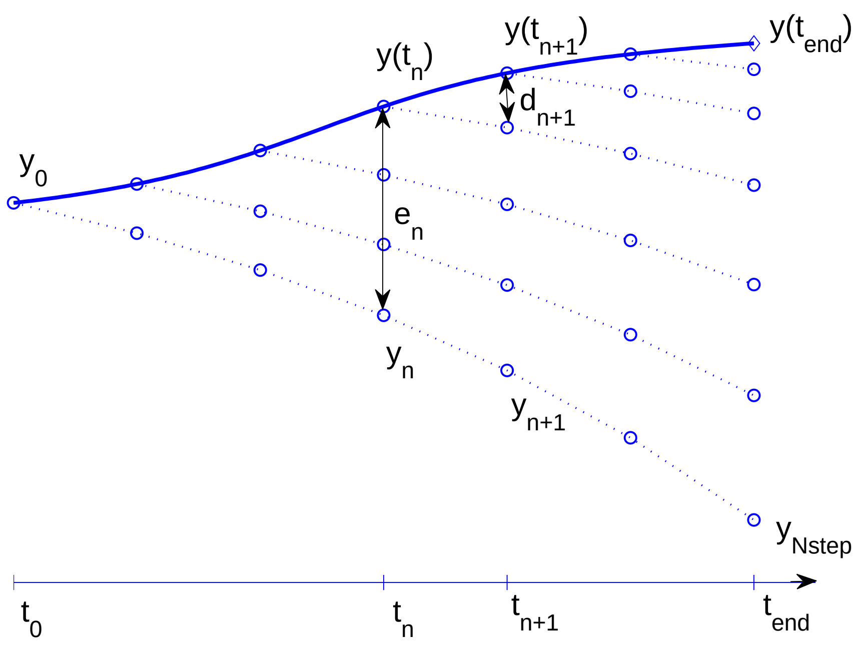


Figure 2: Lady Windermere's Fan

where $v$ is somewhere between $y(t_n)$ and $y_n$. We have here used the mean value theorem (Theorem 1.8 in Burden and Faires) for $f(t_n, y(t_n)) - f(t_n, y_n)$. This is about as far as we get with exact calculations, since $\xi$ in $d_{n+1}$ as well as $v$ in $f_y$ are unknown, and will also change from one step to the next. So we will look for an *upper bound* of the global error. We will first assume upper bounds for our unknown, that is, we assume there exist positive constants $D$ and $L$ so that

$$\frac{1}{2}\left|y''\right| \leq D \text{ for all } t \in (t_0, t_{end}) \qquad \text{and} \qquad |f_y| \leq L \text{ for all } t \in [t_0, t_{end}] \text{ and for all } y.$$

Taken the absolute value of both sides of (47) and using the triangle inequality gives

$$|e_{n+1}| \leq (1 + hL)\,|e_n| + Dh^2.$$

Since $e_0 = 0$ (there is no error at the initial point) we can use this formula recursively to get an upper bound for the error at the endpoint:

$$\begin{aligned}
|e_1| &\leq Dh^2, \\
|e_2| &\leq (1 + hL)Dh^2 + Dh^2 \\
&\vdots \\
|e_{Nstep}| &\leq \sum_{i=0}^{Nstep-1} (1 + hL)^i Dh^2 = \frac{(1 + hL)^{Nstep} - 1}{1 + hL - 1} Dh^2.
\end{aligned}$$

Using the fact that $1 + hL \leq e^{hL}$ (why?) and $h \cdot Nstep = t_{end} - t_0$ we finally reach the conclusion

$$|e_{Nstep}| \leq \frac{\left(e^{hL}\right)^{Nstep} - 1}{L} Dh = \frac{e^{L(t_{end}-t_0)} - 1}{L} D \cdot h = C \cdot h.$$

The constant $C = \left(e^{hL} - 1\right) D/L$ depends only on the problem, and we have proved convergence

$$|y(t_{end}) - y_{Nstep}| \to 0 \text{ when } h \to 0 \text{ (or } Nstep \to \infty).$$

**Summary:** In this section we have

1. Formulated the problem.
2. Developed an algorithm.
3. Implemented and tested it.
4. Proved convergence.

This is fairly much what this course in Numerical Mathematics is about.

## 6.2 Some background on ODEs.

In this section some useful notation on ordinary differential equations will be presented. We will also give existence and uniqueness results, but without proofs.
A system of $m$ first order ordinary differential equation is given by

$$y' = f(t, y) \tag{48}$$

or, written out, as

$$\begin{aligned} y_1' &= f_1(t, y_1, \cdots, y_m), \\ y_2' &= f_2(t, y_1, \cdots, y_m), \\ &\vdots \\ y_m' &= f_m(t, y_1, \cdots, y_m). \end{aligned}$$

This is an *initial value problem* (IVP) if the solution is given at some point $t_0$, thus

$$y_1(t_0) = y_{1,0},\ y(t_0) = y_{2,0},\ \cdots\ y_m(t_0) = y_{m,0}.$$

**Example 6.4.** *The following equation is an example of the Lotka-Volterra equation:*

$$\begin{aligned} y_1' &= y_1 - y_1 y_2, \\ y_2' &= y_1 y_2 - 2y_2. \end{aligned}$$

An ODE is called *autonomous* if $f$ is not a function of $t$, but only of $y$. The Lotka-Volterra equation is an example of an autonomous ODE. A nonautonomous system can be made autonomous by a simple trick, just add the equation

$$y'_{m+1} = 1, \qquad y_{m+1}(t_0) = t_0,$$

and replace $t$ with $y_{m+1}$. Also higher order ODE/IVPs

$$u^{(m)} = f(t, u, u', \cdots, u^{(m-1)}), \qquad u(t_0) = u_0,\ u'(t_0) = u'_0,\ \cdots, u^{(m-1)}(t_0) = u_0^{(m-1)},$$

where $u^{(m)} = d^m u/dt^m$, can be written as a system of first order equations, again by a simple trick: Let

$$y_1 = u,\ y_2 = u',\ \cdots y_m = u^{(m-1)},$$

and we get the system

$$\begin{aligned} y'_1 &= y_2, & y_1(t_0) &= u_0, \\ y'_2 &= y_3, & y_2(t_0) &= u'_0, \\ &\vdots & &\vdots \\ y'_{m-1} &= y_m, & y_{m-1}(t_0) &= u_0^{(m-2)}, \\ y'_m &= f(t, y_1, y_2, \cdots, y_m), & y_m(t_0) &= u_0^{(m-1)}. \end{aligned}$$

**Example 6.5.** *Van der Pol's equation is given by*

$$u'' + \mu(u^2 - 1)u' + u = 0.$$

*Using $y_1 = u$ and $y_2 = u'$ this equation can be rewritten as*

$$\begin{aligned} y'_1 &= y_2, \\ y'_2 &= \mu(1 - y_1^2)y_2 - y_1. \end{aligned}$$

*This problem was first introduced by Van der Pol in 1926 in the study of an electronic oscillator.*

Before concluding this section, we present some existence and uniqueness results for solution of ODEs.

**Definition 6.6.** *A function $f : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$ satisfies the* Lipschitz condition *with respect to $y$ on a domain $(a, b) \times D$ where $D \subset \mathbb{R}^m$ if there exist a constant $L$ so that*

$$\|f(t, y) - f(t, \tilde{y})\| \leq L\|y - \tilde{y}\|, \qquad \textit{for all} \quad t \in (a, b),\ y, \tilde{y} \in D.$$

*The constant $L$ is called* the Lipschitz constant.

It is not hard to show that the function $f$ satisfies the Lipschitz condition if $\partial f_i/\partial y_j,\ i, j = 1, \cdots, m$ are continuous and bounded on the domain.

**Theorem 6.7.** *Consider the initial value problem*

$$y' = f(t, y), \qquad y(t_0) = y_0. \tag{49}$$

*If*

1. *$f(t, y)$ is continuous in $(a, b) \times D$,*
2. *$f(t, y)$ satisfies the Lipschitz condition with respect to $y$ in $(a, b) \times D$.*

*with given initial values $t_0 \in (a, b)$ and $y_0 \in D$, then (49) has one and only one solution in $(a, b) \times D$.*

## 6.3 Numerical solution of ODEs.

In this section we develop some simple methods for the solution of initial value problems. In both cases, let us assume that we somehow have found solutions $y_l \approx y(t_l)$, for $l = 0, 1, \cdots, n$, and we want to find an approximation $y_{n+1} \approx y(t_{n+1})$ where $t_{n+1} = t_n + h$, where $h$ is the stepsize. Basically, there are two different classes of methods in practical use.

1. *One-step methods.* Only $y_n$ is used to find the approximation $y_{n+1}$. One-step methods usually require more than one function evaluation pr. step. They can all be put in a general abstract form
$$y_{n+1} = y_n + h\Phi(t_n, y_n; h).$$
2. *Linear multistep methods:* $y_{n+1}$ is approximated from $y_{n-k+1}, \cdots, y_n$.

### 6.3.1 Some examples of one-step methods.

Assume that $t_n, y_n$ is known. The exact solution $y(t_{n+1})$ with $t_{n+1} = t_n + h$ of (48) passing through this point is given by

$$y(t_n + h) = y_n + \int_{t_n}^{t_{n+1}} y'(\tau)d\tau = y_n + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau))d\tau. \tag{50}$$

The idea is to find approximations to the last integral. The simplest idea is to use $f(\tau, y(\tau)) \approx f(t_n, y_n)$, in which case we get the Euler method again:

$$y_{n+1} = y_n + hf(t_n, y_n).$$

The integral can also be approximated by the trapezoidal rule

$$\int_{t_n}^{t_{n+1}} f(\tau, y(\tau)) = \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y(t_{n+1})).$$

By replacing the unknown solution $y(t_{n+1})$ by $y_{n+1}$ we get the *trapezoidal method*

$$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right).$$

Here $y_{n+1}$ is available by solving a (usually) nonlinear system of equations. Such methods are called implicit. To avoid this extra difficulty, we could replace $y_{n+1}$ on the right hand side by the approximation from Eulers method, thus

$$\begin{aligned}\tilde{y}_{n+1} &= y_n + hf(t_n, y_n);\\ y_{n+1} &= y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, \tilde{y}_{n+1})\right).\end{aligned}$$

This method is called the *improved Euler method.* Similarly, we could have used the midpoint rule for the integral,

$$\int_{t_n}^{t_{n+1}} f(\tau, y(\tau)) = \left(f(t_n + \frac{h}{2}, y(t_n + \frac{h}{2})\right),$$

and replaced $y(t_n + \frac{h}{2})$ by one half Euler step. The result is the *modified Euler method*:

$$\begin{aligned}
\tilde{y}_{n+\frac{1}{2}} &= y_n + \frac{h}{2} f(t_n, y_n), \\
y_{n+1} &= y_n + h f(t_n + \frac{h}{2}, \tilde{y}_{n+\frac{1}{2}}).
\end{aligned}$$

Do we gain anything by constructing these methods? Let us solve the problem from Example 6.1 using improved/modified Euler with $h = 0.1$. For each step, also the global error $e_n = y(t_n) - y_n$ is computed. For comparison, also the result for the Euler method is included.

|  | Euler |  | improved Euler |  | modified Euler |  |
|---|---|---|---|---|---|---|
| $t_n$ | $y_n$ | $e_n$ | $y_n$ | $e_n$ | $y_n$ | $e_n$ |
| 0.0 | 1.000000 | 0 | 1.000000 | 0 | 1.000000 | 0 |
| 0.1 | 1.000000 | $-9.95 \cdot 10^{-3}$ | 0.990000 | $4.98 \cdot 10^{-5}$ | 0.990000 | $4.98 \cdot 10^{-5}$ |
| 0.2 | 0.980000 | $-1.92 \cdot 10^{-2}$ | 0.960696 | $9.34 \cdot 10^{-5}$ | 0.960597 | $1.92 \cdot 10^{-4}$ |
| 0.3 | 0.940800 | $-2.69 \cdot 10^{-2}$ | 0.913814 | $1.17 \cdot 10^{-4}$ | 0.913528 | $4.03 \cdot 10^{-4}$ |
| 0.4 | 0.884352 | $-3.22 \cdot 10^{-2}$ | 0.852040 | $1.04 \cdot 10^{-4}$ | 0.851499 | $6.45 \cdot 10^{-4}$ |
| 0.5 | 0.813604 | $-3.48 \cdot 10^{-2}$ | 0.778765 | $3.60 \cdot 10^{-5}$ | 0.777930 | $8.71 \cdot 10^{-4}$ |
| 0.6 | 0.732243 | $-3.46 \cdot 10^{-2}$ | 0.697773 | $-9.69 \cdot 10^{-5}$ | 0.696636 | $1.04 \cdot 10^{-3}$ |
| 0.7 | 0.644374 | $-3.17 \cdot 10^{-2}$ | 0.612924 | $-2.98 \cdot 10^{-4}$ | 0.611507 | $1.12 \cdot 10^{-3}$ |
| 0.8 | 0.554162 | $-2.69 \cdot 10^{-2}$ | 0.527850 | $-5.58 \cdot 10^{-4}$ | 0.526202 | $1.09 \cdot 10^{-3}$ |
| 0.9 | 0.465496 | $-2.06 \cdot 10^{-2}$ | 0.445717 | $-8.59 \cdot 10^{-4}$ | 0.443904 | $9.54 \cdot 10^{-4}$ |
| 1.0 | 0.381707 | $-1.38 \cdot 10^{-2}$ | 0.369053 | $-1.17 \cdot 10^{-3}$ | 0.367153 | $7.27 \cdot 10^{-4}$ |

As we can see, there is a significant improvement in accuracy, compared with the Euler method.

## 6.4 Runge-Kutta methods

The Euler method, as well as the improved and modified Euler methods are all examples on *explicit Runge-Kutta methods* (ERK). Such schemes are given by

$$\begin{aligned}
k_1 &= f(t_n, y_n), \qquad (51) \\
k_2 &= f(t_n + c_2 h, y_n + h a_{21} k_1), \\
k_3 &= f\big(t_n + c_3 h, y_n + h(a_{31} k_1 + a_{32} k_2)\big), \\
&\vdots \\
k_s &= f\big(t_n + c_s h, y_n + h \sum_{j=1}^{s-1} a_{sj} k_j\big), \\
y_{n+1} &= y_n + h \sum_{i=1}^{s} b_i k_i,
\end{aligned}$$

where $c_i$, $a_{ij}$ and $b_i$ are coefficients defining the method. We always require $c_i = \sum_{j=1}^{s} a_{ij}$. Here, $s$ is the number of *stages*, or the number of function evaluations needed for each step. The vectors $k_i$ are called stage derivatives. The improved Euler method is then a two-stage

RK-method, written as

$$\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f(t_n + h, y_n + hk_1),\\
y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2).
\end{aligned}$$

Also implicit methods, like the trapezoidal rule,

$$y_{n+1} = y_n + \frac{h}{2}\big(f(t_n, y_n) + f(t_n + h, y_{n+1})\big)$$

can be written in a similar form,

$$\begin{aligned}
k_1 &= f(t_n, y_n),\\
k_2 &= f\big(t_n + h, y_n + \frac{h}{2}(k_1 + k_2)\big),\\
y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2).
\end{aligned}$$

But, contrary to what is the case for explicit methods, a nonlinear system of equations has to be solved to find $k_2$.

**Definition 6.8.** *An $s$-stage Runge-Kutta method is given by*

$$\begin{aligned}
k_i &= f\big(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij}k_j\big), \qquad i = 1, 2, \cdots, s,\\
y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i.
\end{aligned}$$

*The method is defined by its coefficients, which is given in a* Butcher tableau

$$\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s}\\
c_2 & a_{21} & a_{22} & \cdots & a_{2s}\\
\vdots & \vdots & & & \vdots\\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss}\\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}, \qquad \textit{where } c_i = \sum_{j=1}^{s} a_{ij}, \quad i = 1, \cdots, s.$$

*The method is* explicit *if $a_{ij} = 0$ whenever $j \geq i$, otherwise* implicit.

**Example 6.9.** *The Butcher-tableaux for the methods presented so far are*

$$\begin{array}{c|c}
0 & 0\\
\hline
 & 1
\end{array}
\qquad
\begin{array}{c|cc}
0 & 0 & 0\\
1 & 1 & 0\\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}
\qquad
\begin{array}{c|cc}
0 & 0 & 0\\
\frac{1}{2} & \frac{1}{2} & 0\\
\hline
 & 0 & 1
\end{array}
\qquad
\begin{array}{c|cc}
0 & 0 & 0\\
1 & \frac{1}{2} & \frac{1}{2}\\
\hline
 & \frac{1}{2} & \frac{1}{2}
\end{array}$$

*Euler* *improved Euler* *modified Euler* *trapezoidal rule*

When the method is explicit, the zeros on and above the diagonal is usually ignored. We conclude this section by presenting the maybe most popular among the RK-methods over times, *The* 4th order Runge-Kutta method (Kutta – 1901):

$$
\begin{aligned}
k_1 &= f(t_n, y_n) \\
k_2 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_1) \\
k_3 &= f(t_n + \tfrac{h}{2}, y_n + \tfrac{h}{2}k_2) \\
k_4 &= f(t_n + h, y_n + hk_3) \\
y_{n+1} &= y_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
\qquad \text{or} \qquad
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
\; . \tag{52}
$$

## 6.5 Order conditions for Runge-Kutta methods.

The following theorem is proved in the exercises.

**Theorem 6.10.** *Let*

$$y' = f(t, y), \qquad y(t_0) = y_0, \qquad t_0 \le t \le t_{end}$$

*be solved by a one-step method*

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h), \tag{53}$$

*with stepsize $h = (t_{end} - t_0)/Nstep$. If*

1. *the increment function $\Phi$ is Lipschitz in $y$, and*
2. *the local truncation error $d_{n+1} = \mathcal{O}(h^{p+1})$ ,*

*then the method is of order $p$, that is, the global error at $t_{end}$ satisfies*

$$e_{Nstep} = y(t_{end}) - y_{Nstep} = \mathcal{O}(h^p).$$

A RK method is a one-step method with increment function $\Phi(t_n, y_n; h) = \sum_{i=1}^{s} b_i k_i$. It is possible to show that $\Phi$ is Lipschitz in $y$ whenever $f$ is Lipschitz and $h \le h_{max}$, where $h_{max}$ is some predefined maximal stepsize. What remains is the order of the local truncation error. To find it, we take the Taylor-expansions of the exact and the numerical solutions and compare. The local truncation error is $\mathcal{O}(h^{p+1})$ if the two series matches for all terms corresponding to $h^q$ with $q \le p$. In principle, this is trivial. In practise, it becomes extremely tedious (give it a try). Fortunately, it is possible to express the two series very elegant by the use of *B-series* and *rooted trees*. Here, we present how this is done, but not why it works. A complete description can be found in the note *the B-series tutorial.*

### B-series and rooted trees

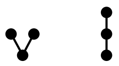
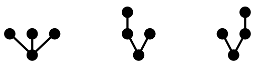
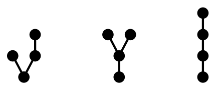
We assume that the equation is rewritten in autonomous form

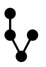
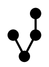
$$y(t)' = f(y(t)), \qquad y(t_0) = y_0. \tag{54}$$

The Taylor expansion of the exact solution of (54) is given by

$$y(t_0+h) = y(t_0) + hy'(t_0) + \frac{h}{2}y''(t_0) + \cdots + \frac{h^p}{p!}y^{(p)}(t_0) + \cdots . \tag{55}$$

From the ODE (54) and repeated use of the chain rule, we get $y' = f$, $y'' = f_y f$, $y''' = f_{yy}ff + f_y f_y f$ , etc. Each higher derivative of $y$ is split into several terms, denoted as *elementary differentials.* These can be represented by rooted trees. A node $\bullet$ represents $f$. A branch out from a bullet represent the derivative of $f$ with respect to $y$. As the chain rule apply, this will always mean that we multiply by $y' = f$, represented by a new node on the end of the branch. We get the following table:

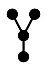
| Elementary differentials | corresponding trees |
|---|---|
| $y' = f$ | |
| $y'' = f_y f$ | |
| $y''' = f_{yy}ff + f_y f_y f$ |  |
| $y^{iv} = f_{yyy}fff + f_{yy}f_y ff + f_{yy}ff_y f$ |  |
| $+f_{yy}ff_y f + f_y f_{yy}ff + f_y f_y f_y f$ |  |

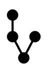
The elementary differentials corresponding to the trees  and  are equal, thus

$$y^{iv} = f_{yyy}fff + 3f_{yy}f_y ff + f_y f_{yy}ff + f_y f_y f_y f.$$

And we can go on like that. For each tree $\tau$ with $p$ nodes we construct a set of total $p$ new trees with $p+1$ nodes by adding one new node to an existing node in $\tau$. This procedure might produce the same tree several times, and the total number of ways to construct a distinct tree is denoted by $\alpha(\tau)$. Let $T$ be the set of all possible, distinct, rooted trees constructed this way, and let $\tau \in T$. A tree with $p$ nodes corresponds to one of the terms in $y^{(p)}$, thus we call this *the order* of the tree and denote it $|\tau|$. The elementary differentials corresponding to a tree is denoted $F(\tau)(y)$.

**Example 6.11.**

*For* $\tau =$  *we have* $|\tau| = 4,$ $F(\tau)(y) = f_y f_{yy}ff,$ $\alpha(\tau) = 1.$

*For* $\tau =$  *we have* $|\tau| = 4,$ $F(\tau)(y) = f_{yy}f_y ff,$ $\alpha(\tau) = 3.$

*Here, $f$ and its differentials are evaluated in $y$.*

Putting this together: If $y(t)$ is the solution of (54), then

$$y^{(p)}(t_n) = \sum_{\substack{\tau \in T \\ |\tau| = p}} \alpha(\tau)F(\tau)(y(t_n)).$$

Insert this into (55), and we can write the exact solution as a B-series:

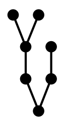
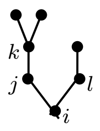
$$y(t_n + h) = y(t_n) + \sum_{\tau \in T} \frac{h^{|\tau|}}{|\tau|!}\, \alpha(\tau)\ F(\tau)(y(t_n)). \tag{56}$$

The numerical solution after one step can also be written as a B-series, but with some different coefficients

$$y_{n+1} = y_n + \sum_{\tau \in T} \frac{h^{|\tau|}}{|\tau|!}\, \gamma(\tau)\varphi(\tau)\ \alpha(\tau)\ F(\tau)(y_n). \tag{57}$$

where $\gamma(\tau)$ is an integer, and $\varphi(\tau)$ depends on the method coefficients, given in the Butcher tableau in Definition 6.8. Both can be found quite easily by the following procedure: Take a tree $\tau$. Label the root with $i$, and all other non-terminal nodes by $j, k, l, \cdots$ . The root correspond to $b_i$. A branch between a lower node $j$ and an upper node $k$ correspond to $a_{jk}$. A terminal node, connected to a node with label $k$ corresponds to $c_k$. $\phi(\tau)$ is found by multiplying all these coefficients, and then take the sum over all the indices from 1 to $s$.

**Example 6.12.**

*The tree* $\tau =$  *can be labelled*  *so that* $\varphi(\tau) = \sum_{i,j,k,l=1}^{s} b_i a_{ij} a_{jk} c_k^2 a_{il} c_l.$

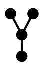
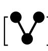
A tree $\tau$ can also be described by its subtrees. Let $\tau = [\tau_1, \tau_2, \cdots, \tau_l]$ be the tree composed by joining the root of the subtrees $\tau_1,\ \tau_2,\ \cdots,\ \tau_l$ to a joint new root. The term $\gamma(\tau)$ is defined recursively by

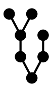
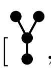
- $\gamma(\bullet) = 1$.
- $\gamma(\tau) = |\tau| \cdot \gamma(\tau_1) \cdots \gamma(\tau_l)$ for $\tau = [\tau_1, \tau_2, \cdots, \tau_l]$.

**Example 6.13.**

$\tau =$ $= [\bullet]$, $\gamma(\tau) = 2 \cdot 1 = 2$

$\tau =$ $= [\bullet, \bullet]$, $\gamma(\tau) = 3 \cdot 1 \cdot 1 = 3$

$\tau =$  $= [$$]$, $\gamma(\tau) = 4 \cdot 3 = 12$

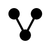
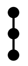
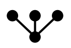
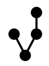
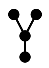
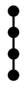
$\tau =$  $= [$$,$ $]$, $\gamma(\tau) = 7 \cdot 12 \cdot 2 = 168$

By comparing the two series (56) and (57) with $y(t_n) = y_n$ we can state the following theorem:

**Theorem 6.14.** *A Runge-Kutta method is of order p if and only if*

$$\varphi(\tau) = \frac{1}{\gamma(\tau)} \qquad \forall \tau \in T, \quad |\tau| \le p.$$

The order conditions up to order 4 are:

| $\tau$ | $\lvert\tau\rvert$ | $\varphi(\tau) = 1/\gamma(\tau)$ |
|---|---|---|
| ● | 1 | $\sum b_i = 1$ |
| | 2 | $\sum b_i c_i = 1/2$ |
|  | 3 | $\sum b_i c_i^2 = 1/3$ |
|  | | $\sum b_i a_{ij} c_j = 1/6$ |
|  | 4 | $\sum b_i c_i^3 = 1/4$ |
|  | | $\sum b_i c_i a_{ij} c_j = 1/8$ |
|  | | $\sum b_i a_{ij} c_j^2 = 1/12$ |
|  | | $\sum b_i a_{ij} a_{jk} c_k = 1/24$ |

### 6.5.1 Error control and stepsize selection.

A user of some numerical black box software will usually require one thing: The accuracy of the numerical solution should be within some user specified tolerance. To accomplish this we have to measure the error, and if the error is too large, it has to be reduced. For ordinary differential equations, this means to reduce the stepsize. On the other hand, we would like our algorithm to be as efficient as possible, that is, to use large stepsizes. This leaves us with two problems: How to measure the error, and how to get the right balance between accuracy and efficiency.

**Local error estimate.** As demonstrated in Figure 2, the global error $y(t_n) - y_n$ comes from two sources: the local truncation error and the propagation of errors produced in preceding steps. This makes it difficult (but not impossible) to measure the global error. Fortunately it is surprisingly easy to measure the *local error*, $l_{n+1}$, the error produced in one step when starting at $(t_n, y_n)$, see Figure 3. Let $y(t; t_n, y_n)$ be the exact solution of the ODE through the point $t_n, y_n$. For a method of order $p$ we get

$$l_{n+1} = y(t_n + h; t_n, y_n) - y_{n+1} = \Psi(t_n, y_n)h^{p+1} + \mathcal{O}(h^{p+2}),$$

where $\mathcal{O}(h^{p+1})$ refer to higher order terms [2] The term $\Psi(t_n, y_n)h^{p+1}$ is called *the principal error term*, and we assume that this term is the dominating part of the error. This assumption is true if the stepsize $h$ is sufficiently small. Taking a step from the same point $t_n, y_n$ with a method of order $\hat{p} = p + 1$ gives a solution $\hat{y}_{n+1}$ with a local error satisfying

$$y(t_n + h; t_n, y_n) - \hat{y}_{n+1} = \mathcal{O}(h^{p+2}).$$

[2] Strictly speaking, the Landau-symbol $\mathcal{O}$ is defined by

$$f(x) = \mathcal{O}(g(x)) \quad \text{for } x \to x_0 \qquad \text{if} \qquad \lim_{x\to x_0} \frac{\|f(x)\|}{\|g(x)\|} < K < \infty$$

for some unspecified constant K. Thus $f(h) = \mathcal{O}(h^q)$ means that $\|f(h)\| \le Kh^q$ when $h \to 0$, and refer to the remainder terms of a truncated series.

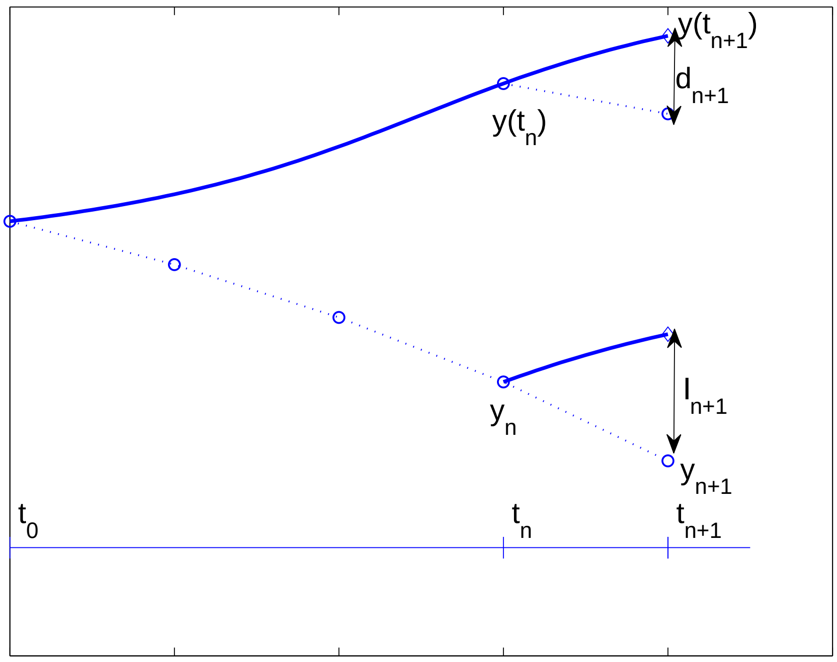


Figure 3: Lady Windermere's Fan

The *local error estimate* is given by

$$le_{n+1} = \hat{y}_{n+1} - y_{n+1} = \Psi(t_n, y_n)h^{p+1} + \mathcal{O}h^{p+2} \approx l_{n+1}.$$

**Embedded Runge-Kutta pair** Given a Runge-Kutta method of order $p$. To be able to measure the local error, we need a method of order $p+1$ (or higher). But we do not want to spend more work (in terms of $f$-evaluations) than necessary. The solution is *embedded Runge-Kutta pairs*, which, for explicit methods are given by

$$\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & & \ddots & & \\
c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\
\hline
 & b_1 & b_2 & \cdots & b_{s-1} & \\
\hline
 & \hat{b}_1 & \hat{b}_2 & \cdots & \hat{b}_{s-1} & \hat{b}_s
\end{array}$$

The method given by the $b_i$'s is of order $p$, the error estimating method given by the $\hat{b}_i$'s is of order $p+1$. (Sometimes it is the other way round. The important thing is to have two methods of different order.) The local error estimate of $y_{n+1}$ is then given by

$$le_{n+1} = \hat{y}_{n+1} - y_{n+1} = h\sum_{i=1}^{s}(\hat{b}_i - b_i)k_i.$$

**Example 6.15.** *A combination of the Euler method and improved Euler will result in the following pair*

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

*so that*

$$k_1 = f(t_n, y_n), \quad k_2 = f(t_n + h, y_n + hk_1), \quad y_{n+1} = y_n + hk_1, \quad l_{n+1} \approx le_{n+1} = \frac{h}{2}(-k_1 + k_2).$$

**Example 6.16.** *Assume that you have decided to use improved Euler, which is of order 2, as your advancing method, and you would like to find an error estimating method of order 3. There are no 2-stage order 3 ERKs, so you have to add one stage to your method. This gives a method like*

$$\begin{array}{c|ccc} 0 & & & \\ 1 & 1 & & \\ c_3 & a_{31} & a_{32} & \\ \hline & \frac{1}{2} & \frac{1}{2} & \\ \hline & \hat{b}_1 & \hat{b}_2 & \hat{b}_3 \end{array}$$

*where we require $c_3 = a_{31} + a_{32}$, which give us five free parameters. These have to satisfy all four order condition for an order 3 method. Using $c_3$ as a free parameter, we get the following class of 3rd order methods:*

$$b_1 = \frac{3c_3 - 1}{6c_3}, \quad b_2 = \frac{2 - 3c_3}{6(1 - c_3)}, \quad b_3 = \frac{1}{6c_3(1 - c_3)}, \quad a_{31} = c_3^2, \quad a_{31} = c_3 - c_3^2.$$

It is also possible to use the highest order method to advance the solution. In this case, we still measure the local error estimate of the lowest order order solution, but we get a more accurate numerical solution for free. This idea is called *local extrapolation.*

MATLAB has two integrators based on explicit Runge-Kutta schemes, `ODE23` which is based on an order 3/2 pair by Bogacki and Shampine, (a 3th order advancing and a 2nd order error estimating method), and `ODE45` based on an order 5/4 pair by Dormand and Prince. Both use local extrapolation.

**Stepsize control** Let the user specify a tolerance $Tol$, and a norm $\|\cdot\|$ in which the error is measured. Let us start with $t_n, y_n$, and do one step forward in time with a stepsize $h_n$, giving $y_{n+1}$ and $le_{n+1}$. If $\|le_{n+1}\| \le Tol$ the step is accepted, and we proceed till the next step, maybe with an increased stepsize. If $\|le_{n+1}\| > Tol$ the step is rejected and we try again with a smaller stepsize. In both cases, we would like to find a stepsize $h_{new}$ which gives a local error estimate smaller than $Tol$, but at the same time as close to $Tol$ as possible. To find the right stepsize, we make one assumption: The function $\Psi(t_n, y_n)$ of the principle error term do not change much from one step to the next, thus $\|\Psi(t_n, y_n)\| \approx \|\Psi(t_{n+1}, y_{n+1})\| \approx C$. Then

$$\begin{aligned} \text{we have:} \qquad & \|le_{n+1}\| &\approx C \cdot h_n^{p+1} \\ \text{we want:} \qquad & Tol &\approx C \cdot h_{new}^{p+1} \end{aligned}$$

We get rid of the unknown $C$ by dividing the two equations with each other, and $h_{new}$ can be solved from

$$\frac{\|le_{n+1}\|}{Tol} \approx \left(\frac{h_n}{h_{new}}\right)^{p+1}.$$

Rejected steps are wasted work, and it should be avoided. Thus we choose the new stepsize somewhat conservative. The new stepsize is computed by

$$h_{new} = P \cdot \left(\frac{Tol}{\|le_{n+1}\|}\right)^{\frac{1}{p+1}} h_n. \tag{58}$$

where $P$ is a *pessimist factor*, usually chosen somewhere in the interval [0.5,0.95]. In the discussion so far we have used the requirement $\|le_{n+1}\| \leq Tol$, that is *error pr. step* (EPS). This do not take into account the fact that the smaller the step is, the more steps you take, and the local errors from each step adds up. From this point of view, it would make sense to rather use the requirement $le\|_{n+1} \leq Tol \cdot h_n$, that is *error pr. unit step* (EPUS). The stepsize selection is then given by

$$h_{new} = P \cdot \left(\frac{Tol}{\|le_{n+1}\|}\right)^{\frac{1}{p}} h_n. \tag{59}$$

Careful analysis has proved that the local extrapolation together with EPS gives proportionality between the global error and the tolerance. The same is true for the use of the lower order method to advance the solution in combination with EPUS.

## 6.6 Stiff equations and linear stability

**Example 6.17.** *Given the ODE*

$$y' = -1000y, \qquad y(0) = 1.$$

*with exact solution*

$$y(t) = e^{-1000t}.$$

*Thus $y(t) \to 0$ as $t \to \infty$. The Euler method applied to this problem yields*

$$y_{n+1} = y_n - 1000hy_n = (1 - 1000h)y_n.$$

*so that $y_n = (1 - 1000h)^n$. This gives us two situations:*

$$\begin{aligned} &\textit{If} \quad |1 - 1000h| < 1 \quad \textit{then} \quad y_n \to 0 \ \textit{as} \ n \to \infty. \\ &\textit{If} \quad |1 - 1000h| > 1 \quad \textit{then} \quad |y_n| \to \infty \ \textit{as} \ n \to \infty \end{aligned}$$

*Clearly, the second situation does not make sense at all, as the numerical solution is unstable even if the exact solution is stable. We have to choose a stepsize $h < 0.002$ to get a stable numerical solution for this problem.*

To be more general: Consider a linear ODE

$$y' = My, \qquad y(0) = y_0, \tag{60}$$

where $M$ is a constant, $m \times m$ matrix. We assume that $M$ is diagonalizable, that is

$$V^{-1}MV = \Lambda$$

where

$$\Lambda = diag\{\lambda_1, \lambda_2, \cdots, \lambda_m\}, \qquad V = [v_1, v_2, \cdots, v_m],$$

where $\lambda_i$, $i = 1, \cdots, m$ are the eigenvalues of $M$ and $v_i$ are the corresponding eigenvectors. By premultiplying (60) with $V^{-1}$, we get

$$V^{-1}y' = V^{-1}MVV^{-1}y, \qquad V^{-1}y(t_0) = V^{-1}y_0$$

or, using $u = V^{-1}y$,

$$u' = \Lambda u, \qquad u(t_0) = V^{-1}y_0 = u_0.$$

The system is now decoupled, and can be written componentwise as

$$u_i' = \lambda_i u_i, \qquad u_i(0) = u_{i,0}, \qquad \lambda_i \in \mathbb{C}, \qquad i = 1, \cdots, m. \tag{61}$$

We have to accept the possibility of complex eigenvalues, however, as $M$ is a real matrix, then complex eigenvalues appears in complex conjugate pairs. In the following, we will consider the situation when

$$\text{Re}(\lambda_i) < 0 \ \text{ for } \ i = 1, \cdots, m, \qquad \text{thus} \qquad y(t) \to 0 \text{ as } t \to \infty. \tag{62}$$

Apply the Euler method to (60):

$$y_{n+1} = y_n + hMy_n.$$

We can do exactly the same linear transformations as above, so the system can be rewritten as

$$u_{i,n+1} = (1 + h\lambda_i)u_{i,n}, \qquad i = 1, \cdots, m.$$

For the numerical solution to be stable, we have to require

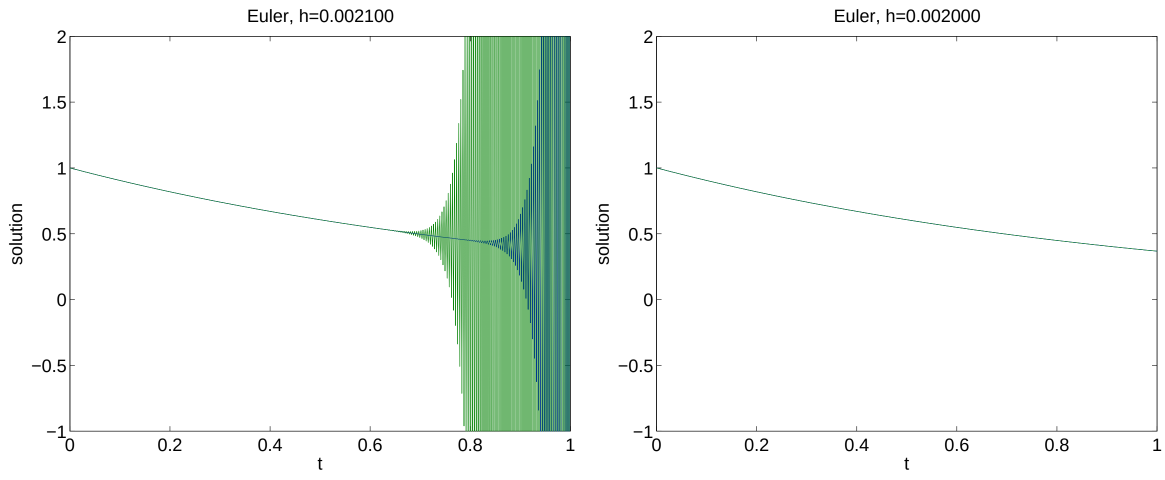
$$|1 + h\lambda_i| \leq 1, \qquad \text{for all the eigenvalues } \lambda_i. \tag{63}$$

(The case $|1 + h\lambda_i h| = 1$ is included, as this is sufficient to prevent the solution from growing.)

**Example 6.18.** *Given*

$$y' = \begin{bmatrix} -2 & 1 \\ 998 & -999 \end{bmatrix} y, \qquad y(0) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

*with exact solution $y_1(t) = y_2(t) = e^{-t}$. The matrix has eigenvalues $-1$ and $-1000$. The initial values are chosen so that the fast decaying mode is missing in the exact solution. This problem is solved by Eulers method, with two almost equal stepsizes, $h = 0.0021$ and $h = 0.002$. The difference is striking, but completely in correspondence with (63) and the result of Example 5.1.*



The MATLAB-file `linsys` used to produce these plots are given on the web-page. Use it to do your own experiments.

Example 6.18 is a typical example of a stiff equation. The stepsize is restricted by a fast decaying component.

**Example 6.19.** *Let*

$$M = \begin{bmatrix} -2 & -2 \\ 1 & 0 \end{bmatrix} \qquad \textit{with eigenvalues} \qquad \lambda_{1,2} = -1 \pm i.$$

*The requirement* (63) *becomes*

$$|1 + h(-1 \pm i)| \leq 1 \quad \textit{or} \quad (1-h)^2 + h^2 \leq 1 \quad \textit{which is satisfied if and only if} \quad 0 \leq h \leq 1.$$

Stiffness occurs in situations with fast decaying solutions (transients) in combination with slow solutions. If you solve an ODE by an adaptive explicit scheme, and the stepsize becomes unreasonable small, stiffness is the most likely explanation. If the stepsizes in additions seems to be independent of your choice of tolerances, then you can be quite sure. The stepsize is restricted by stability related to the transients, and not by accuracy. The effect is demonstrated in the MATLAB-file `teststiffness` available on the web page. The *backward Euler* method is one way to overcome this problem:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \tag{64}$$

or, applied to the problem of (61)

$$u_{i,n+1} = u_{i,n} + h\lambda u_{i,n+1}, \qquad \Rightarrow \qquad u_{i,n+1} = \frac{1}{1-h\lambda_i} u_{i,n}.$$

Since $|1/(1-h\lambda_i)| \leq 1$ whenever $\mathrm{Re}(\lambda_i) \leq 0$ there is no stepsize restriction caused by stability issues. In fact, $u_{i,n+1} \to 0$ as $\mathrm{Re}(h\lambda_i) \to -\infty$, so fast transients decay quickly, as they are supposed to do. But this nice behaviour is not for free: for a nonlinear ODE a nonlinear system of equations has to be solved for each step. We will return to this topic later.

## Linear stability theory for Runge-Kutta methods.

Given the linear test equation

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}. \tag{65}$$

Thus $\lambda = \alpha + i\beta$. The solution can be expressed by

$$y(t_n + h) = e^{\alpha h} e^{i\beta h} y(t_n).$$

Clearly, the solution is stable if $\alpha \leq 0$, that is $\lambda \in \mathbb{C}^-$. For the numerical solution we then require the stepsize $h$ to be chosen so that

$$|y_{n+1}| \leq |y_n| \quad \text{whenever} \lambda \in \mathbb{C}^- \tag{66}$$

When a RK method is applied (65), we simply get

$$y_{n+1} = R(z) y_n, \qquad z = h\lambda$$

where $R$ is a polynomial or a rational function. $R$ is called *the stability function* of the RK method. The numerical solution is stable if $|R(z)| \leq 1$, otherwise it is unstable. This motivates the following definition of *the region of absolute stability* as

$$\mathcal{D} = \{z \in \mathbb{C} \; : \; |R(z)| \leq 1\}.$$

The condition (66) is satisfied for all $h > 0$ if

$$\mathbb{C}^- \subseteq \mathcal{D},$$

Methods satisfying this condition are called *A-stable*. The Backward Euler method (64) is an example of an $A$-stable method.

**Example 6.20.** *A 2-stage ERK applied to* (65) *is given by:*

$$k_1 = \lambda y_n, \qquad k_2 = \lambda(y_n + h a_{21} \lambda y_n), \qquad y_{n+1} = y_n + h\lambda(b_1 + b_2) y_n + (h\lambda)^2 b_2 a_{21} y_n$$

*If this method is of order 2, then $b_1 + b_2 = 1$ and $b_2 a_{21} = 1/2$, so that*

$$R(z) = 1 + z + \frac{1}{2} z^2.$$

The stability function of an $s$-stage ERKs is a polynomial of degree $s$. As a consequence, no ERKs can be A-stable! If the order of the method is $s$, then

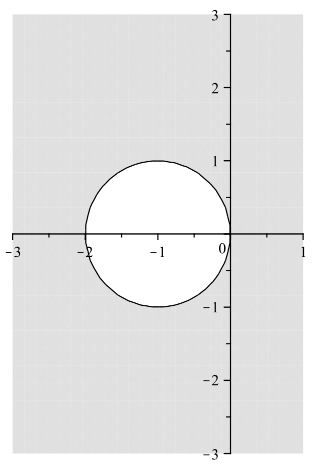
$$R(z) = \sum_{i=0}^{s} \frac{z^i}{i!}.$$

See Figure 4 for plots of the stability regions. But it has been proved that ERK with $p = s$ only exist for $s \leq 4$. To get an order 5 ERK, 6 stages are needed.

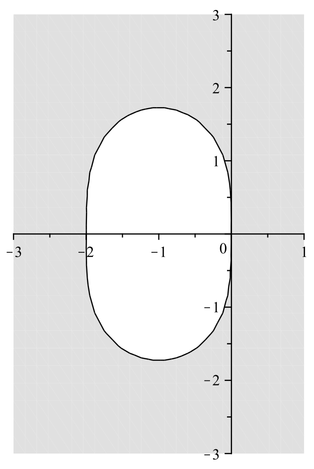
**Example 6.21.** *The trapezoidal rule (see section 3.1) applied to* (65) *gives*

$$y_{n+1} = y_n + \frac{h}{2}(\lambda y_n + \lambda y_{n+1}) \qquad \Rightarrow \qquad R(z) = \frac{1+z}{1-z}.$$

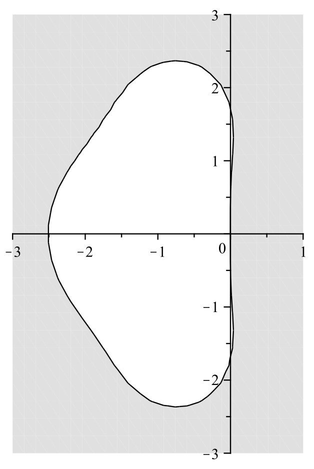
*In this case $\mathcal{D} = \mathbb{C}^-$, which is perfect.*



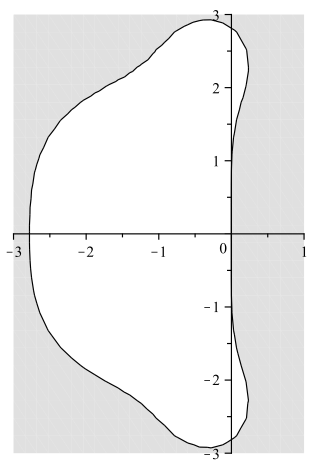
$p = s = 1$



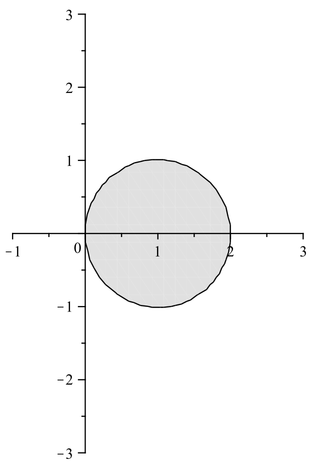
$p = s = 2$



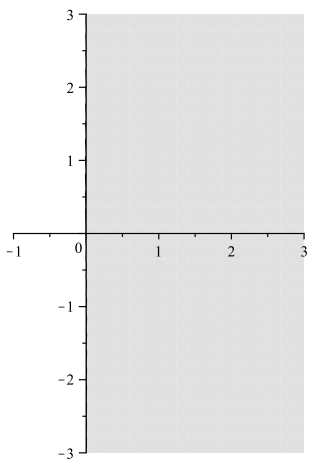
$p = s = 3$



$p = s = 4$



Backward Euler



Trapezoidal rule

Figure 4: Stability regions in $\mathbb{C}^-$: The first four are the stability regions for explicit RK methods of order $p = s$. The white regions are stable, the grey unstable.

To summarise:

- For a given $\lambda \in \mathbb{C}^-$, choose a stepsize $h$ so that $h\lambda \in \mathcal{D}$.
- If your problem is stiff, use an $A$-stable method.
- There are no $A$-stable explicit methods.

## 6.7 Linear multistep methods

A $k$-step linear multistep method (LMM) applied to the ODE

$$y' = f(t,y), \quad y(t_0) = y_0, \quad t_0 \le t \le t_{end}.$$

is given by

$$\sum_{l=0}^{k} \alpha_l y_{n+l} = h \sum_{l=0}^{k} \beta_l f_{n+l}, \tag{67}$$

where $\alpha_l, \beta_l$ are the method coefficients, $f_j = f(t_j, y_j)$ and $t_j = t_0 + jh$, $h = (t_{end} - t_0)/Nstep$. Usually we require

$$\alpha_k = 1 \quad \text{and} \quad |\alpha_0| + |\beta_0| \ne 0.$$

To get started with a $k$-step method, we also need starting values $y_l \approx y(t_l)$, $l = 0, 1, \cdots, k-1$. A method is explicit if $\beta_k = 0$, otherwise implicit. The *leapfrog method*

$$y_{n+2} - y_n = 2hf(t_{n+1}, y_{n+1}) \tag{68}$$

and the method given by

$$y_{n+2} - y_{n+1} = h\left(\frac{3}{2}f_{n+1} - \frac{1}{2}f_n\right) \tag{69}$$

are both examples of explicit 2-step methods.

**Example 6.22.** *Given the problem*

$$y' = -2ty, \qquad y(0) = 1$$

*with exact solution $y(t) = e^{-t^2}$. Let $h = 0.1$, and $y_1 = e^{-h^2}$. This problem is solved by* (69), *and the numerical solution and the error is given by*

| $t_n$ | $y_n$ | $\lvert e_n\rvert$ |
|---|---|---|
| 0.0 | 1.000000 | 0.00 |
| 0.1 | 0.990050 | 0.00 |
| 0.2 | 0.960348 | $4.41 \cdot 10^{-4}$ |
| 0.3 | 0.912628 | $1.30 \cdot 10^{-3}$ |
| 0.4 | 0.849698 | $2.45 \cdot 10^{-3}$ |
| 0.5 | 0.775113 | $3.69 \cdot 10^{-3}$ |
| 0.6 | 0.692834 | $4.84 \cdot 10^{-3}$ |
| 0.7 | 0.606880 | $5.75 \cdot 10^{-3}$ |
| 0.8 | 0.521005 | $6.29 \cdot 10^{-3}$ |
| 0.9 | 0.438445 | $6.41 \cdot 10^{-3}$ |
| 1.0 | 0.361746 | $6.13 \cdot 10^{-3}$ |

.

*The corresponding MATLAB code is given in* `lmm.m`.

### 6.7.1 Consistency and order.

We define the *local discretization error* $\tau_{n+k}(h)$ by

$$h\tau_{n+k}(h) = \sum_{l=0}^{k} \left( \alpha_l y(t_{n+l}) - h\beta_l y'(t_{n+l}) \right). \tag{70}$$

You can think about the $h\tau_{n+k}$ as the defect obtained when plugging the exact solution into the difference equation (67). A method is *consistent* if $\tau_{n+k}(h) \underset{h\to 0}{\to} 0$. The term $h\tau_{n+k}(h)$ can be written as a power series in $h$

$$h\tau_{n+k}(h) = C_0 y(t_n) + C_1 h y'(t_n) + C_2 h^2 y''(t_n) + \cdots + C_q h^q y^{(q)}(t_n) + \cdots,$$

by expanding $y(t_n + lh)$ and $y'(t_n + lh)$ into their Taylor series around $t_n$,

$$\begin{aligned} y(t_n + lh) &= y(t_n) + (lh)y'(t_n) + \frac{1}{2}(lh)^2 y''(t_n) + \cdots + \frac{(lh)^q}{q!} y^{(q)}(t_n) + \cdots \\ y'(t_n + lh) &= y'(t_n) + (lh)y''(t_n) + \frac{1}{2}(lh)^2 y'''(t_n) + \cdots + \frac{(lh)^{q-1}}{q-1!} y^{(q)}(t_n) + \cdots \end{aligned}$$

for sufficiently differentiable solutions $y(t)$. Insert this into (70), get the following expressions for $C_q$:

$$C_0 = \sum_{l=0}^{k} \alpha_l, \qquad C_q = \frac{1}{q!} \sum_{l=0}^{k} \left( l^q \alpha_l - q l^{q-1} \beta_l \right), \quad q = 1, 2, \cdots. \tag{71}$$

The method is consistent if $C_0 = C_1 = 0$. It is *of order* $p$ if

$$C_0 = C_1 = \cdots = C_p = 0, \qquad C_{p+1} \neq 0.$$

The constant $C_{p+1}$ is called the *error constant.*

**Example 6.23.** *The LMM* (69) *is defined by*

$$\alpha_0 = 0,\ \alpha_1 = -1,\ \alpha_2 = 1,\ \beta_0 = -\frac{1}{2},\ \beta_1 = \frac{3}{2},\ \beta_2 = 0,$$

*thus*

$$\begin{aligned} C_0 &= \alpha_0 + \alpha_1 + \alpha_2 = 0. \\ C_1 &= \alpha_1 + 2\alpha_2 - (\beta_0 + \beta_1 + \beta_2) = 0 \\ C_2 &= \frac{1}{2!}\big(\alpha_1 + 2^2\alpha_2 - 2(\beta_1 + 2\beta_2)\big) = 0 \\ C_3 &= \frac{1}{3!}\big(\alpha_1 + 2^3\alpha_2 - 3(\beta_1 + 2^2\beta_2)\big) = \frac{5}{12}. \end{aligned}$$

*The method is consistent and of order 2.*

**Example 6.24.** *Is it possible to construct an explicit 2-step method of order 3? There are 4 free coefficients $\alpha_0, \alpha_1, \beta_0, \beta_1$, and 4 order conditions to be solved ($C_0 = C_1 = C_2 = C_3 = 0$). The solution is*

$$\alpha_0 = -5,\ \alpha_1 = 4,\ \beta_0 = 2,\ \beta_1 = 4.$$

*Test this method on the ODE of Example 2.1. (Replace the method coefficients in* `lmm.m`*.) The result is nothing but disastrous. Taking smaller steps only increase the problem.*

To see why, you have to know a bit about how to solve difference equations.

### 6.7.2 Linear difference equations

A linear difference equation with constant coefficients is given by

$$\sum_{l=0}^{k} \alpha_l y_{n+l} = \varphi_n, \qquad n = 0, 1, 2, \cdots . \tag{72}$$

The solution of this equation is a sequence $\{y_n\}$ of numbers (or vectors). Let $\{\tilde{y}_n\}$ be the general solution of the homogeneous problem

$$\sum_{l=0}^{k} \alpha_l y_{n+l} = 0. \tag{73}$$

Let $\psi_n$ be one particular solution of (72). The general solution of (72) is then $\{y_n\}$ where $y_n = \tilde{y}_n + \psi_n$. To find a unique solution, we will need the starting values $y_0, y_1, \cdots, y_{k-1}$. Let us try $\tilde{y}_n = r^n$ as a solution of the homogeneous equation (73). This is true if

$$\sum_{l=0}^{k} \alpha_l r^{n+l} = r^n \sum_{l=0}^{k} \alpha_l r^l = 0.$$

The polynomial $\rho(r) = \sum_{l=0}^{k} \alpha_l r^l$ is called *the characteristic polynomial*, and $\{r^n\}$ is a solution of (73) if $r$ is a root of $\rho(r)$. The $k$th degree polynomial $\rho(r)$ has $k$ roots altogether, $r_1, r_2, \cdots, r_k$, they can be distinct and real, they can be distinct and complex, in which case they appear in complex conjugate pairs, or they can be multiple. In the latter case, say $r_1 = r_2 = \cdots = r_\mu$ we get a set of linear independent solutions $\{r_1^n\}$, $\{nr_1^n\}$, $\cdots$, $\{n^{\mu-1}r_1^n\}$. Altogether we have found $k$ linear independent solutions $\{\tilde{y}_{n,l}\}$ of the homogeneous equation, and the general solution is given by

$$y_n = \sum_{l=1}^{k} \kappa_l \tilde{y}_{n,l} + \psi_n.$$

The coefficients $\kappa_l$ can be determined from the starting values.

**Example 6.25.** *Given*

$$y_{n+4} - 6y_{n+3} + 14y_{n+2} - 16y_{n+1} + 8y_n = n$$
$$y_0 = 1,\ y_1 = 2,\ y_2 = 3\ y_3 = 4.$$

*The characteristic polynomial is given by*

$$\rho(r) = r^4 - 6r^3 + 14r^2 - 16r + 8$$

*with roots $r_1 = r_2 = 2$, $r_3 = 1+i$, $r_4 = 1-i$. As a particular solution we try $\psi_n = an + b$. Inserted into the difference equation we find this to be a solution if $a = 1$, $b = 2$. The general solution has the form*

$$y_n = \kappa_1 2^n + \kappa_2 n 2^n + \kappa_3 (1+i)^n + \kappa_4 (1-i)^n + n + 2.$$

*From the starting values we find that $\kappa_1 = -1$, $\kappa_2 = \frac{1}{4}$, $\kappa_3 = -i/4$ and $\kappa_4 = i/4$. So, the solution of the problem is*

$$\begin{aligned} y_n &= 2^n\left(\frac{n}{4} - 1\right) - \frac{i(1+i)^n}{4} + \frac{i(1-i)^n}{4} + n + 2 \\ &= 2^n\left(\frac{n}{4} - 1\right) - 2^{\frac{n-2}{2}} \sin\left(\frac{n\pi}{4}\right) + n + 2. \end{aligned}$$

**Example 6.26.** *The homogeneous part of the difference equation of Example 6.24 is*

$$\rho(r) = r^2 + 4r - 5 = (r-1)(r+5).$$

*One root is 5. Thus, one solution component is multiplied by a factor -5 for each step, independent of the stepsize. Which explain why this method fails.*

### 6.7.3 Zero-stability and convergence

Let us start with the definition of convergence. As before, we consider the error at $t_{end}$, using $Nstep$ steps with constant stepsize $h = (t_{end} - t_0)/Nstep$.

**Definition 6.27.**

- *A linear multistep method* (67) *is* convergent *if, for all ODEs satisfying the conditions of Theorem 6.7 we get*

$$y_{Nstep} \underset{h\to 0}{\to} y(t_{end}), \qquad \textit{whenever} \qquad y_l \underset{h\to 0}{\to} y(t_0+lh), \quad l = 0, 1, \cdots, k-1.$$

- *The method is* convergent of order $p$ *if, for all ODEs with $f$ sufficiently differentiable, there exists a positive $h_0$ such that for all $h < h_0$*

$$\|y(t_{end} - y_{Nstep}\| \le Kh^p \qquad \textit{whenever} \qquad \|y(t_0+lh) - y_l\| \le K_0 h^p, \quad l = 0, 1, \cdots, k-1.$$

The first characteristic polynomial of an LMM (67) is

$$\rho(r) = \sum_{l=0}^{k} \alpha_l r^l,$$

with roots $r_1, r_2, \cdots, r_k$. From the section on difference equation, it follows that for the boundedness of the solution $y_n$ we require:

1. $|r_i| \le 1$, for $i = 1, 2, \cdots, k$.
2. $|r_i| < 1$ if $r_i$ is a multiple root.

A method satisfying these two conditions is called *zero-stable.*

We can now state (without proof) the following important result:

**Theorem 6.28.** *(Dahlquist)*

$$\textit{Convergence} \qquad \Leftrightarrow \qquad \textit{Zero-stability + Consistency.}$$

For a consistent method, $C_0 = \sum_{l=0}^{k} \alpha_l = 0$ so the characteristic polynomial $\rho(r)$ will always have one root $r_1 = 1$.

The zero-stability requirement puts a severe restriction on the maximum order of a convergent $k$-step method:

**Theorem 6.29.** *(The first Dahlquist-barrier) The order $p$ of a zero-stable $k$-step method satisfies*

$$\begin{aligned} p &\le k+2 && \textit{if } k \textit{ is even,} \\ p &\le k+1 && \textit{if } k \textit{ is odd,} \\ p &\le k && \textit{if } \beta_k \le 0. \end{aligned}$$

Notice that the last line include all explicit LMMs.

## 6.8 Adams-Bashforth-Moulton methods

The most famous linear multistep methods are constructed by the means of interpolation. For instance by the following strategy:
The solution of the ODE satisfy the integral equation

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t))dt. \tag{74}$$

Assume that we have found $f_i = f(t_i, y_i)$ for $i = n-k+1, \cdots, n$, with $t_i = t_0 + ih$. Construct the polynomial of degree $k-1$, satisfying

$$p_{k-1}(t_i) = f(t_i, y_i), \qquad i = n-k+1, \dots, n.$$

The interpolation points are equidistributed (constant stepsize), so Newton's backward difference formula can be used in this case (see Exercise 2), that is

$$p_{k-1}(t) = p_{k-1}(t_n + sh) = f_n + \sum_{j=1}^{k-1} (-1)^j \binom{-s}{j} \nabla^j f_n$$

where

$$(-1)^j \binom{-s}{j} = \frac{s(s+1)\cdots(s+j-1)}{j!}$$

and

$$\nabla^0 f_n = f_n, \qquad \nabla^j f_n = \nabla^{j-1} f_n - \nabla^{j-1} f_{n-1}.$$

Using $y_{n+1} \approx y(t_{n+1})$. $y_n \approx y(t_n)$ and $p_{k-1}(t) \approx f(t, y(t))$ in (74) gives

$$\begin{aligned} y_{n+1} - y_n \int_{t_n}^{t_{n+1}} p_{k-1}(t)dt &= h \int_0^1 p_{k-1}(t_n + sh)ds \\ &= hf_n + h \sum_{j=1}^{k-1} \left( (-1)^j \int_0^1 \binom{-s}{1} ds \right) \nabla^j f_n. \end{aligned} \tag{75}$$

This gives the *Adams-Bashforth methods*

$$y_{n+1} - y_n = h \sum_{j=0}^{k-1} \gamma_j \nabla^j f_n, \qquad \gamma_0 = 1, \quad \gamma_j = (-1)^j \int_0^1 \binom{-s}{j} ds.$$

**Example 6.30.** *We get*

$$\gamma_0 = 1, \quad \gamma_1 = \int_0^1 sds = \frac{1}{2}, \quad \gamma_2 = \int_0^1 \frac{s(s+1)}{2} ds = \frac{5}{12}$$

*and the first few methods becomes:*

$$\begin{aligned} y_{n+1} - y_n &= hf_n \\ y_{n+1} - y_n &= h\left(\frac{3}{2} f_n - \frac{1}{2} f_{n-1}\right) \\ y_{n+1} - y_n &= h\left(\frac{23}{12} f_n - \frac{4}{3} f_{n-1} + \frac{5}{12} f_{n-2}\right) \end{aligned}$$

A $k$-step Adams-Bashforth method is explicit, has order $k$ (which is the optimal order for explicit methods) and it is zero-stable. In addition, the error constant $C_{p+1} = \gamma_k$. Implicit Adams methods are constructed similarly, but in this case we include the (unknown) point $(t_{n+1}, f_{n+1})$ into the set of interpolation points. So the polynomial

$$p_k^*(t) = p_k^*(t_n + sh) = f_{n+1} + \sum_{j=1}^{k} (-1)^j \binom{-s+1}{j} \nabla^j f_{n+1}$$

interpolates the points $(t_i, f_i)$, $i = n-k+1, \ldots, n+1$. Using this, we get the *Adams-Moulton* methods

$$y_{n+1} - y_n = h \sum_{j=0}^{k} \gamma_j^* \nabla^j f_{n+1}, \qquad \gamma_0^* = 1, \quad \gamma_j^* = (-1)^j \int_0^1 \binom{-s+1}{j} ds.$$

**Example 6.31.** *We get*

$$\gamma_0^* = 1, \quad \gamma_1^* = \int_0^1 (s-1) ds = -\frac{1}{2}, \quad \gamma_2^* = \int_0^1 \frac{(s-1)s}{2} ds = -\frac{1}{12}$$

*and the first methods becomes*

$$\begin{aligned} y_{n+1} - y_n &= h f_{n+1} && \textit{(Backward Euler)} \\ y_{n+1} - y_n &= h \left( \frac{1}{2} f_{n+1} + \frac{1}{2} f_n \right) && \textit{(Trapezoidal method)} \\ y_{n+1} - y_n &= h \left( \frac{5}{12} f_{n+1} + \frac{2}{3} f_n - \frac{1}{12} f_{n-1} \right). \end{aligned}$$

A $k$-step Adams-Moulton method is implicit, of order $k+1$ and is zero-stable. The error constant $C_{p+1} = \gamma_{k+1}^*$. Despite the fact that the Adams-Moulton methods are implicit, they have some advantages compared to their explicit counterparts: They are of one order higher, the error constants are much smaller, and the linear stability properties (when the methods are applied to the linear test problem $y' = \lambda y$) are much better.

| $k$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $\gamma_k$ | 1 | $\frac{1}{2}$ | $\frac{5}{12}$ | $\frac{3}{8}$ | $\frac{251}{720}$ | $\frac{95}{288}$ | $\frac{19087}{60480}$ |
| $\gamma_k^*$ | 1 | $-\frac{1}{2}$ | $-\frac{1}{12}$ | $-\frac{1}{24}$ | $-\frac{19}{720}$ | $-\frac{3}{160}$ | $-\frac{863}{60480}$ |

Table 1: The $\gamma$'s for the Adams methods.

## 6.9 Predictor-corrector methods

A predictor-corrector (PC) pair is a pair of one explicit (predictor) and one implicit (corrector) methods. The nonlinear equations from the application of the implicit method are solved by a fixed number of fixed point iterations, using the solution by the explicit method as starting values for the iterations.

**Example 6.32.** *We may construct a PC method from a second order Adams-Bashforth scheme and the trapezoidal rule as follows:*

```
$y^{[0]}_{n+1} = y_n + \frac{h}{2}(3f_n - f_{n-1})$                    (P : Predictor)
for $l = 0, 1, \ldots, m$
    $f^{[l]}_{n+1} = f(t_{n+1}, y^{[l]}_{n+1})$                      (E : Evaluation)
    $y^{[l+1]}_{n+1} = y_n + \frac{h}{2}(f^{[l]}_{n+1} + f_n)$       (C : Corrector)
end
$y_{n+1} = y^{[m]}_{n+1}$
$f_{n+1} = f(t_{n+1}, y_{n+1}).$                                   (E : Evaluation)
```

*Such schemes are commonly referred as P(EC)$^m$E schemes.*

The predictor and the corrector is often by the same order, in which case only one or two iterations are needed.

**Error estimation in predictor-corrector methods.**

The local discretization error of some LMM is given by

$$h\tau_{n+1} = \sum_{l=0}^{k}(\alpha_l y(t_{n-k+1+l} - h\beta_l y'(t_{n-k+1+l})) = h^{p+1}C_{p+1}y^{(p+1)}(t_{n-k+1}) + \mathcal{O}(h^{p+2}).$$

But we can do the Taylor expansions of $y$ and $y'$ around $t_n$ rather than $t_{n-k+1}$. This will not alter the principal error term, but the terms hidden in the expression $\mathcal{O}(h^{p+2})$ will change. As a consequence, we get

$$h\tau_{n+1} = h^{p+1}C_{p+1}y^{(p+1)}(t_n) + \mathcal{O}(h^{p+2}).$$

Assume that $y_i = y(t_i)$ for $i = n-k+1, \ldots, n$, and $\alpha_k = 1$. Then

$$h\tau_{n+1} = y(t_{n+1}) - y_{n+1} + \mathcal{O}(h^{p+2}) = h^{p+1}C_{p+1}y^{(p+1)}(t_n) + \mathcal{O}(h^{p+2}).$$

Assume that we have chosen a predictor-corrector pair, using methods of the same order $p$. Then

$$\begin{aligned} &(P) && y(t_{n+1}) - y^{[0]}_{n+1} \approx h^{p+1}C^{[0]}_{p+1}y^{(p+1)}(t_n), \\ &(C) && y(t_{n+1}) - y_{n+1} \approx h^{p+1}C_{p+1}y^{(p+1)}(t_n), \end{aligned}$$

and

$$y_{n+1} - y^{[0]}_{n+1} \approx h^{p+1}(C^{[0]}_{p+1} - C_{p+1})y^{(p+1)}(t_n).$$

From this we get the following local error estimate for the corrector, called *Milne's device*:

$$y(t_{n+1}) - y_{n+1} \approx \frac{C_{p+1}}{C^{[0]}_{p+1]} - C_{p+1}}(y_{n+1} - y^{[0]}_{n+1}).$$

**Example 6.33.** *Consider the PC-scheme of Example 6.32. In this case*

$$C^{[0]}_{p+1} = \frac{5}{12}, \qquad C_{p+1} = -\frac{1}{12}, \qquad \textit{so} \qquad \frac{C_{p+1}}{C^{[0]}_{p+1]} - C_{p+1}} = -\frac{1}{6}.$$

*Apply the scheme to the linear test problem*

$$y' = -y, \qquad y(0) = 1,$$

*using* $y_0 = 1$*,* $y_1 = e^{-h}$ *and* $h = 0.1$*. One step of the PC-method gives*

| $l$ | $y_2^{[l]}$ | $\lvert y_2 - y_2^{[l]}\rvert$ | $\lvert y(0.2) - y_2^{[l]}\rvert$ | $\frac{1}{6}\lvert y_2^{[l]} - y_2^{[0]}\rvert$ |
|---|---|---|---|---|
| 0 | 0.819112 | $4.49 \cdot 10^{-4}$ | $3.81 \cdot 10^{-4}$ | |
| 1 | 0.818640 | $2.25 \cdot 10^{-5}$ | $9.08 \cdot 10^{-5}$ | $7.86 \cdot 10^{-5}$ |
| 2 | 0.818664 | $1.12 \cdot 10^{-6}$ | $6.72 \cdot 10^{-5}$ | $7.47 \cdot 10^{-5}$ |
| 3 | 0.818662 | $5.62 \cdot 10^{-8}$ | $6.84 \cdot 10^{-5}$ | $7.49 \cdot 10^{-5}$ |

*After 1-2 iterations, the iteration error is much smaller than the local error, and we also observe that Milne's device gives a reasonable approximation to the error.*

**Remark** Predictor-corrector methods are not suited for stiff problems. You can see this by e.g. using the trapezoidal rule on $y' = \lambda y$. The trapezoidal rule has excellent stability properties. But the iteration scheme

$$y^{[l+1]}_{n+1} = y_n + \frac{h}{2}\lambda(y^{[l]}_{n+1} + y_n)$$

will only converge if $|h\lambda/2| < 1$.