

Lecture Notes in TMA4215 Numerical Mathematics

Anne Kværnø

November 20, 2009

Abstract

These lecture notes are supplementary to the text book used in TMA4215 Numerical Mathematics. The notes focus on numerical solution of ordinary differential equations, and will replace the corresponding chapters in the textbook. In addition, I have included some sections that are not completely covered by the textbook. The notes are to some extent a revision and translation of the note “Forelesningsnotater i Numerisk Matematikk” by Brynjulf Owren, but there are some significant differences.

I am grateful for the feedback I have got from some of you, and hope you will continue to report errors and misprints.

Contents

1	Eulers method.	3
2	Some background on ODEs.	6
3	Numerical solution of ODEs.	8
3.1	Some examples of one-step methods.	8
4	Runge-Kutta methods	10
4.1	Order conditions for Runge-Kutta methods.	11
4.2	Error control and stepsize selection.	14
5	Stiff equations and linear stability	18
6	Collocation.	22
7	Linear multistep methods	24
7.1	Consistency and order.	25
7.2	Linear difference equations	26
7.3	Zero-stability and convergence	27
7.4	Adams-Bashforth-Moulton methods	28
7.5	Predictor-corrector methods	30
8	Bernoulli polynomials and the Euler-Maclaurin formula.	32
9	Orthogonal polynomials.	33
10	Solution of systems of nonlinear equations	37

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

$$y' = f(t, y), \quad t_0 \leq t \leq t_{end}, \quad y(t_0) = y_0, \quad (1)$$

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 1.1. *The ODE/IVP*

$$y' = -2ty, \quad 0 \leq t \leq 1, \quad y(0) = 1.0$$

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

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

where ξ is somewhere between t_0 and t_{end} . The integer $p \geq 1$ 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

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

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

Example 1.2. *Eulers method with $h = 0.1$ applied to the ODE of Example 1.1 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 $h = (t_{end} - t_0)/Nstep$. 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 1. For each step an error is made, and these errors are then propagated til the next steps and accumulate at the endpoint.

Definition 1.3. 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

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

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

$$e_{n+1} = e_n + h(f(t_n, y(t_n)) - f(t_n, y_n)) + d_{n+1} = (1 + hf_y(t_n, v))e_n + d_{n+1}, \quad (3)$$

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

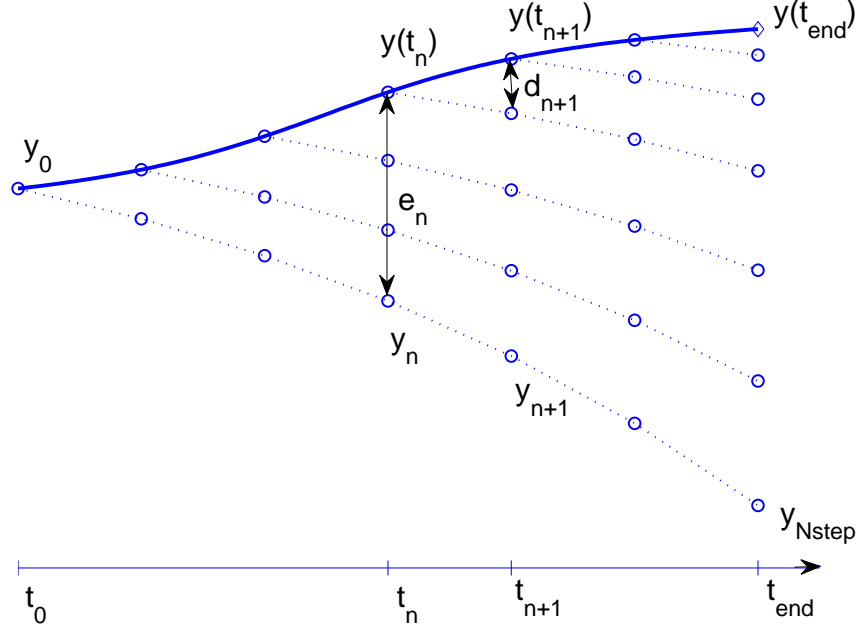


Figure 1: Lady Windermere's Fan

with exact calculations, since ξ 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

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

Taken the absolute value of both sides of (3) 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

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

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

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

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. To be more precise, for each class of problems (ODEs, integrals, linear and nonlinear equations, ...) we will roughly do the following

1. Formulate the problem. What do we know about it? What about existence and uniqueness results?
2. Develop an algorithm to find a solution. Can it be generalised?
3. Implement and test the algorithm. Test it on problems with known solutions. Do we get a reasonable result?
4. Prove convergence. How accurate is the algorithm?
5. Verify the theoretical results by numerical experiments.
6. Try the algorithm on harder problems. Do we get unexpected results? If yes, can they be explained? Can we get around them?
7. Can we make our algorithm to automatically adjust itself to fulfil requirements given by the user (adaptivity)?

The last two points do not apply to all the problems we are going to discuss. We will also derive mathematical results that are useful for either the theoretical analysis, or for development of methods.

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

$$y' = f(t, y) \tag{4}$$

or, written out, as

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

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

$$y_1(t_0) = y_{1,0}, y_2(t_0) = y_{2,0}, \dots, y_m(t_0) = y_{m,0}.$$

Example 2.1. *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

$$y_{m+1}' = 1, \quad y_{m+1}(t_0) = t_0,$$

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

$$u^{(m)} = f(t, u, u', \dots, u^{(m-1)}), \quad u(t_0) = u_0, u'(t_0) = u_0', \dots, u^{(m-1)}(t_0) = u_0^{(m-1)},$$

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

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

and we get the system

$$\begin{array}{ll}
y_1' = y_2, & y_1(t_0) = u_0, \\
y_2' = y_3, & y_2(t_0) = u_0', \\
\vdots & \vdots \\
y_{m-1}' = y_m, & y_{m-1}(t_0) = u_0^{(m-2)}, \\
y_m' = f(t, y_1, y_2, \dots, y_m), & y_m(t_0) = u_0^{(m-1)}.
\end{array}$$

Example 2.2. *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 2.3. A function $f : \mathbb{R} \times \mathbb{R}^m \rightarrow \mathbb{R}^m$ 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

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

The constant L is called the Lipschitz constant.

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

Theorem 2.4. Consider the initial value problem

$$y' = f(t, y), \quad y(t_0) = y_0. \quad (5)$$

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 (5) has one and only one solution in $(a, b) \times D$.

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 $l = 0, 1, \dots, n$, 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 per 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}, \dots, y_n .

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 (4) passing through this point is given by

$$y(t_n + h) = y_n + \int_{t_n}^{t_{n+1}} y'(\tau) d\tau = y_n + \int_{t_n}^{t_{n+1}} f(\tau, y(\tau)) d\tau. \quad (6)$$

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*

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

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

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

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

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

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

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

Do we gain anything by constructing these methods? Let us solve the problem from Example 1.1 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.

t_n	Euler		improved Euler		modified Euler	
	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.

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

$$\begin{aligned}
 k_1 &= f(t_n, y_n), \\
 k_2 &= f(t_n + c_2h, y_n + ha_{21}k_1), \\
 k_3 &= f(t_n + c_3h, y_n + h(a_{31}k_1 + a_{32}k_2)), \\
 &\vdots \\
 k_s &= f(t_n + c_sh, y_n + h \sum_{j=1}^{s-1} a_{sj}k_j), \\
 y_{n+1} &= y_n + h \sum_{i=1}^s b_i k_i,
 \end{aligned} \tag{7}$$

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,

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

can be written in a similar form,

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

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

Definition 4.1. *An s -stage Runge-Kutta method is given by*

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

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

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

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

Example 4.2. The Butcher-tableaux for the methods presented so far are

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

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):

$$\begin{array}{l}
 k_1 = f(t_n, y_n) \\
 k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\
 k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\
 k_4 = f(t_n + h, y_n + hk_3) \\
 y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
 \end{array}
 \quad \text{or} \quad
 \begin{array}{c|cccc}
 0 & & & & \\
 \frac{1}{2} & \frac{1}{2} & & & \\
 \frac{1}{2} & 0 & \frac{1}{2} & & \\
 1 & 0 & 0 & 1 & \\
 \hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
 \end{array}
 . \quad (8)$$

4.1 Order conditions for Runge-Kutta methods.

The following theorem were proved in Exercise 2, task 2:

Theorem 4.3. *Let*

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

be solved by a one-step method

$$y_{n+1} = y_n + h\Phi(t_n, y_n; h), \quad (9)$$

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

1. *the increment function Φ 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 Φ is Lipschitz in y whenever f is Lipschitz and $h \leq h_{max}$, 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 $q \leq p$. 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

$$y(t)' = f(y(t)), \quad y(t_0) = y_0. \quad (10)$$

The Taylor expansion of the exact solution of (10) is given by

$$y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{h^2}{2}y''(t_0) + \dots + \frac{h^p}{p!}y^{(p)}(t_0) + \dots. \quad (11)$$

From the ODE (10) and repeated use of the chain rule, we get $y' = f$, $y'' = f_y f$, $y''' = f_{yy} f f + f_y f_y f$, 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
$y' = f$	
$y'' = f_y f$	
$y''' = f_{yy} f f + f_y f_y f$	
$y^{iv} = f_{yyy} f f f + f_{yy} f_y f f + f_y f_{yy} f f$	
$+ f_{yy} f f_y f + f_y f_{yy} f f + f_y f_y f_y f$	

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

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

And we can go on like that. For each tree τ 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 τ . 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 4.4.

For $\tau = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \end{array}$ we have $|\tau| = 4$, $F(\tau)(y) = f_y f_{yy} f f$, $\alpha(\tau) = 1$.

For $\tau = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \end{array}$ we have $|\tau| = 4$, $F(\tau)(y) = f_{yy} f_y f f$, $\alpha(\tau) = 3$.

Here, f and its differentials are evaluated in y .

Putting this together: If $y(t)$ is the solution of (10), then

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

Insert this into (11), and we can write the exact solution as a B-series:

$$y(t_n + h) = y(t_n) + \sum_{\tau \in T} \frac{h^{|\tau|}}{|\tau|!} \alpha(\tau) F(\tau)(y(t_n)). \quad (12)$$

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

$$y_{n+1} = y_n + \sum_{\tau \in T} \frac{h^{|\tau|}}{|\tau|!} \gamma(\tau) \varphi(\tau) \alpha(\tau) F(\tau)(y_n). \quad (13)$$

where $\gamma(\tau)$ is an integer, and $\varphi(\tau)$ depends on the method coefficients, given in the Butcher tableau in Definition 4.1. Both can be found quite easily by the following procedure: Take a tree τ . Label the root with i , and all other non-terminal nodes by j, k, l, \dots . 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 4.5.

The tree $\tau = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \end{array}$ can be labelled $\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \\ \diagup \quad \diagdown \\ \bullet \quad \bullet \end{array}$ 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 τ can also be described by its subtrees. Let $\tau = [\tau_1, \tau_2, \dots, \tau_l]$ be the tree composed by joining the root of the subtrees $\tau_1, \tau_2, \dots, \tau_l$ 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 $\tau = [\tau_1, \tau_2, \dots, \tau_l]$.

Example 4.6.

$$\begin{aligned}
 \tau = \bullet &= [\bullet], & \gamma(\tau) &= 2 \cdot 1 = 2 \\
 \tau = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array} &= [\bullet, \bullet], & \gamma(\tau) &= 3 \cdot 1 \cdot 1 = 3 \\
 \tau = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array} &= [\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}], & \gamma(\tau) &= 4 \cdot 3 = 12 \\
 \tau = \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array} &= [\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}, \begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}], & \gamma(\tau) &= 7 \cdot 12 \cdot 2 = 168
 \end{aligned}$$

By comparing the two series (12) and (13) with $y(t_n) = y_n$ we can state the following theorem:

Theorem 4.7. *A Runge-Kutta method is of order p if and only if*

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

The order conditions up to order 4 are:

τ	$ \tau $	$\varphi(\tau) = 1/\gamma(\tau)$
\bullet	1	$\sum b_i = 1$
$\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}$	2	$\sum b_i c_i = 1/2$
$\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}$	3	$\sum b_i c_i^2 = 1/3$ $\sum b_i a_{ij} c_j = 1/6$
$\begin{array}{c} \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \diagup \quad \diagdown \\ \bullet \\ \diagup \quad \diagdown \\ \bullet \end{array}$	4	$\sum b_i c_i^3 = 1/4$ $\sum b_i c_i a_{ij} c_j = 1/8$ $\sum b_i a_{ij} c_j^2 = 1/12$ $\sum b_i a_{ij} a_{jk} c_k = 1/24$

4.2 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 1, 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

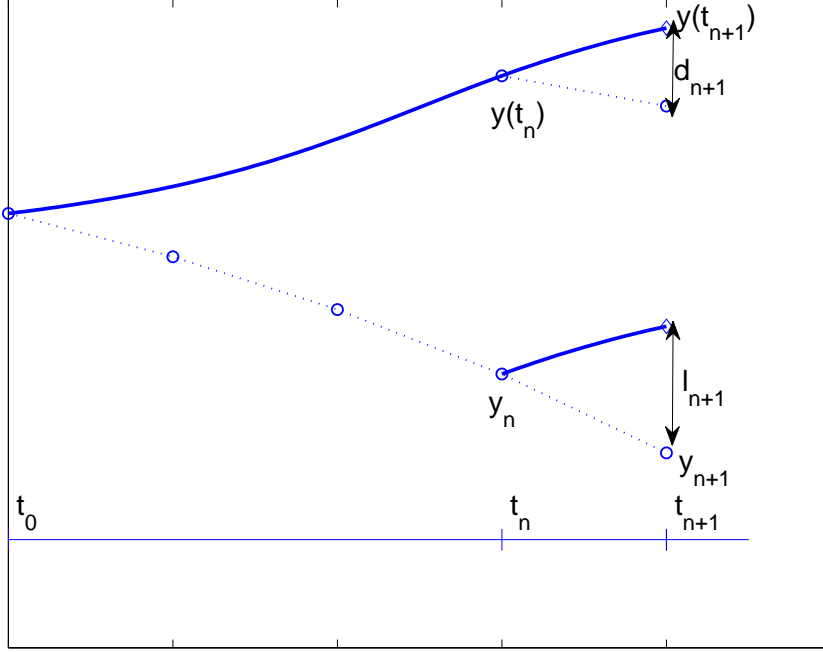


Figure 2: Lady Windermere's Fan

starting at (t_n, y_n) , see Figure 2. 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 ¹ 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

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

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

¹Strictly speaking, the Landau-symbol \mathcal{O} is defined by

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

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

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

0					
c_2	a_{21}				
c_3	a_{31}	a_{32}			
\vdots	\vdots	\ddots			
c_s	a_{s1}	a_{s2}	\cdots	$a_{s,s-1}$	
	b_1	b_2	\cdots	b_{s-1}	
	\hat{b}_1	\hat{b}_2	\cdots	\hat{b}_{s-1}	\hat{b}_s

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 4.8. A combination of the Euler method and improved Euler will result in the following pair

0		
1	1	
	1	
	$\frac{1}{2}$	$\frac{1}{2}$

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 4.9. 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

0			
1	1		
c_3	a_{31}	a_{32}	
	$\frac{1}{2}$	$\frac{1}{2}$	
	\hat{b}_1	\hat{b}_2	\hat{b}_3

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 3th order methods:

$$b_1 = \frac{3c_3 - 1}{6c_3}, \quad b_2 = \frac{2 - 3c_3}{6(1 - c_3)}, \quad b_3 = \frac{1}{6c_3(1 - c_3)}, \quad a_{31} = c_3^2, \quad a_{32} = c_3 - c_3^2.$$

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 $\|le_{n+1}\| \leq Tol$ 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

$$\text{we have:} \quad \|le_{n+1}\| \approx C \cdot h_n^{p+1}$$

$$\text{we want:} \quad Tol \approx C \cdot h_{new}^{p+1}$$

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

$$h_{new} = P \cdot \left(\frac{Tol}{\|le_{n+1}\|} \right)^{\frac{1}{p+1}} h_n. \quad (14)$$

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

$$h_{new} = P \cdot \left(\frac{Tol}{\|le_{n+1}\|} \right)^{\frac{1}{p}} h_n. \quad (15)$$

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.

5 Stiff equations and linear stability

Example 5.1. *Given the ODE*

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

with exact solution

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

Thus $y(t) \rightarrow 0$ as $t \rightarrow \infty$. 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:

If $|1 - 1000h| < 1$ then $y_n \rightarrow 0$ as $n \rightarrow \infty$.

If $|1 - 1000h| > 1$ then $|y_n| \rightarrow \infty$ as $n \rightarrow \infty$

Clearly, the second situation do 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

$$y' = My, \quad y(0) = y_0, \tag{16}$$

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

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

where

$$\Lambda = \text{diag}\{\lambda_1, \lambda_2, \dots, \lambda_m\}, \quad V = [v_1, v_2, \dots, v_m],$$

where λ_i , $i = 1, \dots, m$ are the eigenvalues of M and v_i are the corresponding eigenvectors. By premultiplying (16) with V^{-1} , we get

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

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

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

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

$$u'_i = \lambda_i u_i, \quad u_i(0) = u_{i,0}, \quad \lambda_i \in \mathbb{C}, \quad i = 1, \dots, m. \tag{17}$$

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

$$\text{Re}(\lambda_i) < 0 \text{ for } i = 1, \dots, m, \quad \text{thus} \quad y(t) \rightarrow 0 \text{ as } t \rightarrow \infty. \tag{18}$$

Apply the Euler method to (16):

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

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

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

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

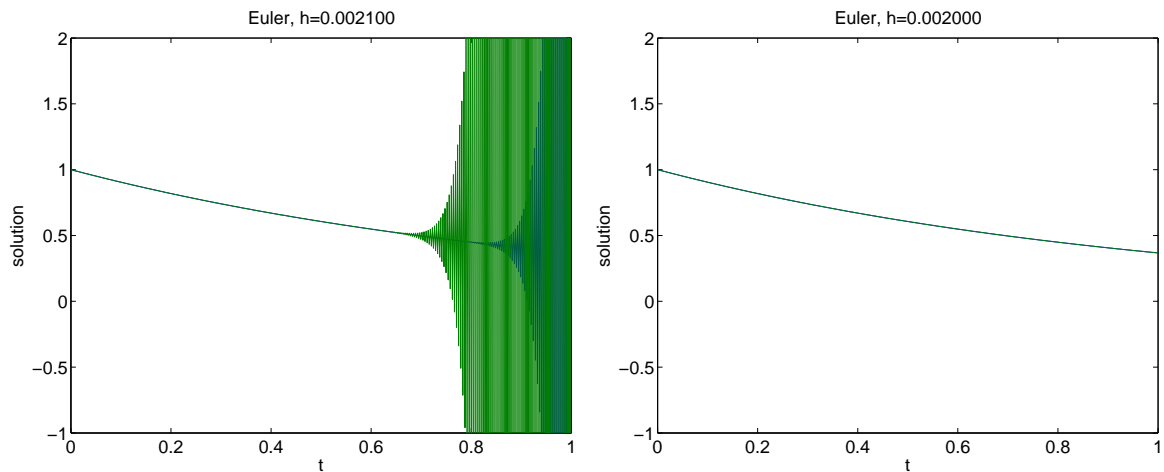
$$|1 + h\lambda_i| \leq 1, \quad \text{for all the eigenvalues } \lambda_i. \quad (19)$$

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

Example 5.2. *Given*

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

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 (19) 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 5.2 is a typical example of a stiff equation. The stepsize is restricted by a fast decaying component.

Example 5.3. *Let*

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

The requirement (19) becomes

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

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:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \quad (20)$$

or, applied to the problem of (17)

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

Since $|1/(1 - h\lambda_i)| \leq 1$ whenever $\text{Re}(\lambda_i) \leq 0$ there is no stepsize restriction caused by stability issues. In fact, $u_{i,n+1} \rightarrow 0$ as $\text{Re}(h\lambda_i) \rightarrow -\infty$, 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

$$y' = \lambda y, \quad \lambda \in \mathbb{C}. \quad (21)$$

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

$$|y_{n+1}| \leq |y_n| \quad \text{whenever } \lambda \in \mathbb{C}^- \quad (22)$$

When a RK method is applied (21), we simply get

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

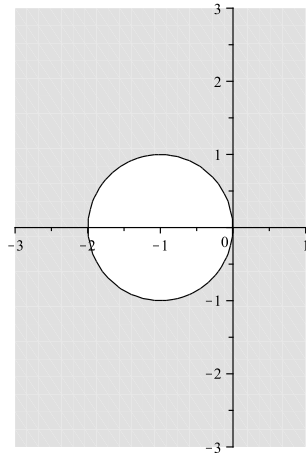
where R is a polynomial or a rational function. R is called *the stability function* of the RK method. The numerical solution is stable if $|R(z)| \leq 1$, otherwise it is unstable. This motivates the following definition of *the region of absolute stability* as

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

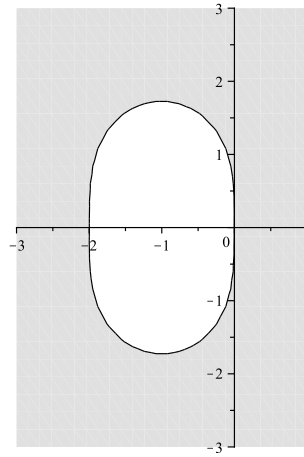
The condition (22) is satisfied for all $h > 0$ if

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

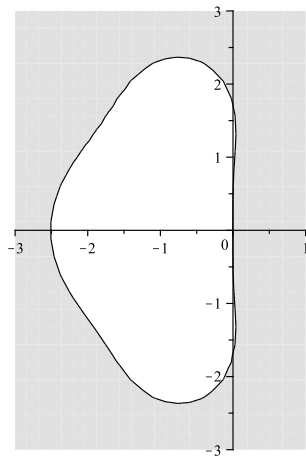
Methods satisfying this condition are called *A-stable*. The Backward Euler method (20) is an example of an *A-stable* method.



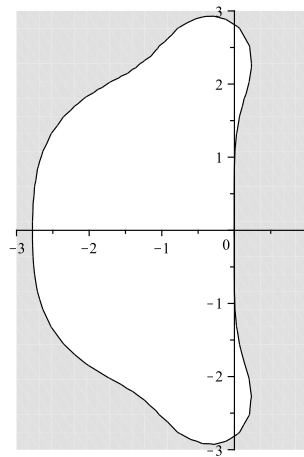
$$p = s = 1$$



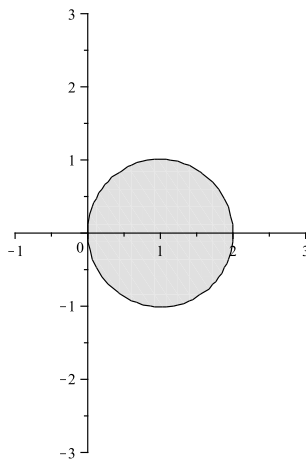
$$p = s = 2$$



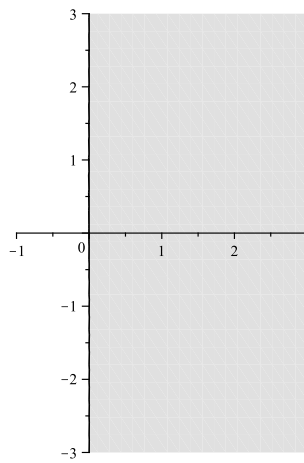
$$p = s = 3$$



$$p = s = 4$$



Backward Euler



Trapezoidal rule

Figure 3: 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.

Example 5.4. A 2-stage ERK applied to (21) is given by:

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

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 3 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 5.5. The trapezoidal rule (see section 3.1) applied to (21) gives

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

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

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 Collocation.

We will now see how the idea of orthogonal polynomials can be used to construct high order Runge-Kutta methods.

Given an ordinary differential equation

$$y'(t) = f(t, y(t)), \quad y(t_n) = y_n, \quad t \in [t_n, t_n + h].$$

Recall also the definition of a Runge-Kutta method applied to the ODE:

$$\begin{aligned} k_i &= f(t_n + c_i h, y_n + h \sum_{j=1}^s a_{ij} k_j) \\ y_{n+1} &= y_n + h \sum_{i=1}^s b_i k_i. \end{aligned} \tag{23}$$

The idea of collocation methods for ODEs are as follows: Assume that s distinct points $c_1, c_2, \dots, c_s \in [0, 1]$ are given. we will then search for a polynomial $u \in \mathbb{P}_s$ satisfying the ODE in the points $t_n + c_i h$, $i = 1, \dots, s$. These are the *collocation conditions*, expressed as

$$u'(t_n + c_i h) = f(t_n + c_i h, u(t_n + c_i h)), \quad i = 1, 2, \dots, s, \tag{24}$$

The initial condition is $u(t_n) = y_n$. When finally $u(t)$ has been found, we will set $y_{n+1} = u(t_n + h)$.

Let k_i be some (so far unknown) approximation to $y'(t_n + c_i h)$, and let $u' = p \in \mathbb{P}_{s-1}$ be the interpolation polynomial satisfying $p(t_n + c_i h) = k_i$. For simplicity, introduce the change of variables $t = t_n + \tau h$, $\tau \in [0, 1]$. Then

$$p(t) = p(t_n + h\tau) = \sum_{j=1}^s k_j \ell_j(\tau), \quad \ell_j(\tau) = \prod_{\substack{k=1 \\ k \neq j}}^s \frac{\tau - c_k}{c_j - c_k}.$$

The polynomial $u(t)$ is given by

$$u(\tilde{t}) = u(t_n) + \int_{t_n}^{\tilde{t}} p(t) dt = y_n + h \sum_{j=1}^s k_j \int_0^{\tilde{\tau}} \ell_j(\tau) d\tau, \quad \tilde{t} \in [t_n, t_n + h] \quad (\text{thus } \tilde{\tau} \in [0, 1])$$

The collocation condition becomes

$$u'(t_n + c_i h) = f \left(t_n + c_i h, y_n + h \sum_{j=1}^s k_j \int_0^{c_i} \ell_j(\tau) d\tau \right)$$

and in addition we get

$$u(t_n + h) = y_n + h \sum_{j=1}^s k_j \int_0^1 \ell_j(\tau) d\tau.$$

But $u'(t_n + c_i h) = k_i$, and $y_{n+1} = u(t_n + h)$ so this is exactly the Runge–Kutta method (23) with coefficients

$$a_{ij} = \int_0^{c_i} \ell_j(\tau) d\tau, \quad i, j = 1, \dots, s, \quad b_i = \int_0^1 \ell_j(\tau) d\tau, \quad i = 1, \dots, s. \quad (25)$$

Runge–Kutta methods constructed this way are called *collocation methods* and they are of order at least s . But a clever choice of the c_i 's will give better results.

Example 6.1. Let $P_2(x) = x^2 - 1/3$ be the second order Legendre polynomial. Let c_1 and c_2 be the zeros of $P_2(2x - 1)$ (using a change of variable to transfer the interval $[-1, 1]$ to $[0, 1]$), that is

$$c_1 = \frac{1}{2} - \frac{\sqrt{3}}{6}, \quad c_2 = \frac{1}{2} + \frac{\sqrt{3}}{6}.$$

The corresponding cardinal functions becomes

$$\ell_1(\tau) = -\sqrt{3} \left(\tau - \frac{1}{2} - \frac{\sqrt{3}}{6} \right), \quad \ell_2(\tau) = \sqrt{3} \left(\tau - \frac{1}{2} + \frac{\sqrt{3}}{6} \right).$$

From (25) we get a Runge–Kutta method with the following Butcher tableau:

$$\begin{array}{c|cc} \frac{1}{2} - \frac{\sqrt{3}}{6} & \frac{1}{4} & \frac{1}{4} - \frac{\sqrt{3}}{6} \\ \frac{1}{2} + \frac{\sqrt{3}}{6} & \frac{1}{4} + \frac{\sqrt{3}}{6} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}.$$

This method has order 4 (check it yourself).

Collocation methods using the zeros of the Legendre-polynomials $P_s(2x - 1)$ for the c_i 's are called Gauss-Legendre methods, also known as Kuntzmann-Butcher methods. They are of order $2s$, they are also A -stable.

7 Linear multistep methods

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

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

is given by

$$\sum_{l=0}^k \alpha_l y_{n+l} = h \sum_{l=0}^k \beta_l f_{n+l}, \quad (26)$$

where α_l, β_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

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

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

$$y_{n+2} - y_n = 2hf(t_{n+1}, y_{n+1}) \quad (27)$$

and the method given by

$$y_{n+2} - y_{n+1} = h \left(\frac{3}{2} f_{n+1} - \frac{1}{2} f_n \right) \quad (28)$$

are both examples of explicit 2-step methods.

Example 7.1. *Given the problem*

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

with exact solution $y(t) = e^{-t^2}$. Let $h = 0.1$, and $y_1 = e^{-h^2}$. This problem is solved by (28), and the numerical solution and the error is given by

t_n	y_n	$ e_n $
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`.

7.1 Consistency and order.

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

$$h\tau_{n+k}(h) = \sum_{l=0}^k (\alpha_l y(t_{n+l}) - h\beta_l y'(t_{n+l})). \quad (29)$$

You can think about the $h\tau_{n+k}$ as the defect obtained when plugging the exact solution into the difference equation (26). A method is *consistent* if $\tau_{n+k}(h) \xrightarrow{h \rightarrow 0} 0$. 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 (29), get the following expressions for C_q :

$$C_0 = \sum_{l=0}^k \alpha_l, \quad C_q = \frac{1}{q!} \sum_{l=0}^k (l^q \alpha_l - q l^{q-1} \beta_l), \quad q = 1, 2, \dots. \quad (30)$$

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

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

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

Example 7.2. The LMM (28) is defined by

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

thus

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

The method is consistent and of order 2.

Example 7.3. 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

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

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.

7.2 Linear difference equations

A linear difference equation with constant coefficients is given by

$$\sum_{l=0}^k \alpha_l y_{n+l} = \varphi_n, \quad n = 0, 1, 2, \dots \quad (31)$$

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

$$\sum_{l=0}^k \alpha_l y_{n+l} = 0. \quad (32)$$

Let ψ_n be one particular solution of (31). The general solution of (31) 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, \dots, y_{k-1} .

Let us try $\tilde{y}_n = r^n$ as a solution of the homogeneous equation (32). 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 (32) if r is a root of $\rho(r)$. The k th degree polynomial $\rho(r)$ has k roots altogether, r_1, r_2, \dots, 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 $r_1 = r_2 = \dots = r_\mu$ we get a set of linear independent solutions $\{r_1^n\}, \{nr_1^n\}, \dots, \{n^{\mu-1}r_1^n\}$. 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 κ_l can be determined from the starting values.

Example 7.4. *Given*

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

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 7.5. The homogeneous part of the difference equation of Example 5.2 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.

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 7.6.

- A linear multistep method (26) is convergent if, for all ODEs satisfying the conditions of Theorem 2.4 we get

$$y_{Nstep} \xrightarrow{h \rightarrow 0} y(t_{end}), \quad \text{whenever} \quad y_l \xrightarrow{h \rightarrow 0} y(t_0 + lh), \quad l = 0, 1, \dots, k-1.$$

- 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 $h < h_0$

$$\|y(t_{end}) - y_{Nstep}\| \leq Kh^p \quad \text{whenever} \quad \|y(t_0 + lh) - y_l\| \leq K_0h^p, \quad l = 0, 1, \dots, k-1.$$

The first characteristic polynomial of an LMM (26) is

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

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

1. $|r_i| \leq 1$, for $i = 1, 2, \dots, k$.
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 7.7. (Dahlquist)

$$\text{Convergence} \quad \Leftrightarrow \quad \text{Zero-stability} + \text{Consistency}.$$

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

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

Notice that the last line include all explicit LMMs.

7.4 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

$$y(t_{n+1}) - y(t_n) = \int_{t_n}^{t_{n+1}} f(t, y(t)) dt. \quad (33)$$

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

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

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

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

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 (33) gives

$$\begin{aligned} y_{n+1} - y_n &\int_{t_n}^{t_{n+1}} p_{k-1}(t) dt = h \int_0^1 p_{k-1}(t_n + sh) ds \\ &= h f_n + h \sum_{j=1}^{k-1} \left((-1)^j \int_0^1 \binom{-s}{j} ds \right) \nabla^j f_n. \end{aligned} \quad (34)$$

This gives the *Adams-Bashforth methods*

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

Example 7.9. We get

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

and the first few methods becomes:

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

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) , $i = n - k + 1, \dots, n + 1$. Using this, we get the *Adams-Moulton* methods

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

Example 7.10. 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

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

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
γ_k	1	$\frac{1}{2}$	$\frac{5}{12}$	$\frac{3}{8}$	$\frac{251}{720}$	$\frac{95}{288}$	$\frac{19087}{60480}$
γ_k^*	1	$-\frac{1}{2}$	$-\frac{1}{12}$	$-\frac{1}{24}$	$-\frac{19}{720}$	$-\frac{3}{160}$	$-\frac{863}{60480}$

Table 1: The γ 's for the Adams methods.

7.5 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 7.11. *We may construct a PC method from a second order Adams-Bashforth scheme and the trapezoidal rule as follows:*

$$y_{n+1}^{[0]} = y_n + \frac{h}{2}(3f_n - f_{n-1}) \quad (P : \text{Predictor})$$

for $l = 0, 1, \dots, m$

$$f_{n+1}^{[l]} = f(t_{n+1}, y_{n+1}^{[l]}) \quad (E : \text{Evaluation})$$

$$y_{n+1}^{[l+1]} = y_n + \frac{h}{2}(f_{n+1}^{[l]} + f_n) \quad (C : \text{Corrector})$$

end

$$y_{n+1} = y_{n+1}^{[m]}$$

$$f_{n+1} = f(t_{n+1}, y_{n+1}). \quad (E : \text{Evaluation})$$

Such schemes are commonly referred as $P(EC)^mE$ 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

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

Assume that $y_i = y(t_i)$ for $i = n - k + 1, \dots, n$, 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

$$(P) \quad y(t_{n+1}) - y_{n+1}^{[0]} \approx h^{p+1} C_{p+1}^{[0]} y^{(p+1)}(t_n),$$

$$(C) \quad y(t_{n+1}) - y_{n+1} \approx h^{p+1} C_{p+1} y^{(p+1)}(t_n),$$

and

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

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

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

Example 7.12. Consider the PC-scheme of Example 7.11. In this case

$$C_{p+1}^{[0]} = \frac{5}{12}, \quad C_{p+1} = -\frac{1}{12}, \quad \text{so} \quad \frac{C_{p+1}}{C_{p+1}^{[0]} - C_{p+1}} = -\frac{1}{6}.$$

Apply the scheme to the linear test problem

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

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

l	$y_2^{[l]}$	$ y_2 - y_2^{[l]} $	$ y(0.2) - y_2^{[l]} $	$\frac{1}{6} y_2^{[l]} - y_2^{[0]} $
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

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

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

8 Bernoulli polynomials and the Euler-Maclaurin formula.

Let the integral

$$I(f) = \int_a^b f(x)dx$$

be approximated by the Trapezoidal rule

$$T(h) = h \left(\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n) \right)$$

where $x_i = a + ih$, $i = 0, \dots, n$ with $h = (b - a)/n$. The aim of this note is to prove that the error can be written as

$$I(f) - T(h) = \sum_{k=1}^{m-1} \frac{b_{2k}}{(2k)!} h^{2k} (f^{(2k-1)}(a) - f^{(2k-1)}(b)) - \frac{b_{2m}}{(2m)!} h^{2m} (b - a) f^{(2m)}(\eta) \quad (35)$$

where $\eta \in (a, b)$, and b_k are the Bernoulli numbers. These will be defined later. Obviously, the formula requires $f \in C^{2m}[a, b]$. The formula proves that the error can be written as an even power series of h , which is the theoretical fundament for the development of the Romberg integration algorithm.

We will first define the well known Bernoulli polynomials which will then be used to prove Euler-Maclaurin's formula. This again will be used to prove (35).

Bernoulli polynomials.

For our purpose, it is convenient to define the Bernoulli polynomials by the following recurrence relation:

$$\begin{aligned} B_0(t) &= 1, \\ B'_k(t) &= kB_{k-1}(t) \quad \text{and} \quad \int_0^1 B_k(t) dt = 0, \quad k \geq 1. \end{aligned}$$

The first few polynomials becomes

$$\begin{aligned} B_1(t) &= t - \frac{1}{2} \\ B_2(t) &= t^2 - t + 1/6 \\ B_3(t) &= t^3 - \frac{3}{2}t^2 + \frac{1}{2}t. \end{aligned}$$

We will need the following properties of these polynomials:

$$B_k(0) = B_k(1), \quad k \geq 2, \quad (36a)$$

$$B_k(1 - t) = (-1)^k B_k(t), \quad (36b)$$

$$B_k(t) - B_k(0) \quad \text{has no zeros in } (0,1) \text{ if } k \text{ is even.} \quad (36c)$$

The *Bernoulli numbers* is given by $b_k = B_k(0)$. As a consequence of (36a) and (36b) we get $b_k = 0$ for k odd and $k \geq 3$. The Bernoulli numbers can also be found by the generating function

$$\frac{t}{e^t - 1} = \sum_{k=0}^{\infty} \frac{b_k}{k!} t^k.$$

Euler-Maclaurins formula

By repeated use of integration by parts and the Bernoulli polynomials, we get

$$\begin{aligned}
\int_0^1 f(t)dt &= \int_0^1 f(t)B_0(t)dt \\
&= f(t)B_1(t)\Big|_0^1 - \int_0^1 B_1(t)f'(t)dt \\
&= \frac{1}{2}[f(1) + f(0)] - \frac{1}{2}B_2(t)f'(t)\Big|_0^1 + \frac{1}{2}\int_0^1 B_2(t)f''(t)dt \\
&= \frac{1}{2}[f(1) + f(0)] - \sum_{j=2}^{\bar{m}} \frac{(-1)^j}{j!} B_j(t)f^{(j-1)}(t)\Big|_0^1 + (-1)^{\bar{m}} \frac{1}{\bar{m}!} \int_0^1 B_{\bar{m}}(t)f^{(\bar{m})}(t)dt.
\end{aligned}$$

Now, since $B_{2k+1}(0) = B_{2k+1}(1) = 0$ and $B_{2k}(0) = B_{2k}(1) = b_{2k}$ this can be written as

$$\begin{aligned}
\int_0^1 f(t)dt &= \frac{1}{2}[f(1) + f(0)] - \sum_{k=1}^{m-1} \frac{b_{2k}}{(2k)!} [f^{(2k-1)}(1) - f^{(2k-1)}(0)] \\
&\quad - \frac{b_{2m}}{(2m)!} [f^{(2m-1)}(1) - f^{(2m-1)}(0)] + \frac{1}{(2m)!} \int_0^1 B_{2m}(t)f^{(2m)}(t)dt.
\end{aligned}$$

Using property (36c) and the mean value theorem for integrals, the last two terms becomes

$$-\frac{1}{(2m)!} \int_0^1 [b_{2m} - B_{2m}(t)] f^{(2m)}(t)dt = -\frac{b_{2m}}{(2m)!} f^{(2m)}(\eta)$$

As a result, we get the *Euler-Maclaurin formula*:

$$\int_0^1 f(t)dt = \frac{1}{2}[f(1) + f(0)] - \sum_{k=1}^{m-1} \frac{b_{2k}}{(2k)!} [f^{(2k-1)}(1) - f^{(2k-1)}(0)] - \frac{b_{2m}}{(2m)!} f^{(2m)}(\eta).$$

Error of the Trapezoidal rule.

Applying the Euler-Maclaurin formula to a function $g(t) = f(x_i + ht)$, using the change of variable $t = (x - x_i)/h$ gives

$$\begin{aligned}
\int_{x_i}^{x_{i+1}} f(x)dx &= \frac{h}{2}[f(x_i) + f(x_{i+1})] - \sum_{k=1}^{m-1} \frac{b_{2k}}{(2k)!} h^{2k} [f^{(2k-1)}(x_{i+1}) - f^{(2k-1)}(x_i)] \\
&\quad - \frac{b_{2m}}{(2m)!} h^{2m+1} f^{(2m)}(\eta_i)
\end{aligned}$$

The expression (35) is finally obtained by summing over all the intervals $[x_i, x_{i+1}]$.

9 Orthogonal polynomials.

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

$$I_w(f) = \int_a^b w(x)f(x)dx \tag{37}$$

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

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

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

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

An approximation to the integral is then given by

$$Q_w(f) = \int_a^b w(x) p_{n-1}(x) dx = \sum_{i=0}^m A_i f(x_i), \quad A_i = \int_a^b w(x) \ell_i(x) dx. \quad (38)$$

The quadrature formula is of precision m if

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

From the construction, these quadrature formulas is of precision at least n . The question is how to choose the nodes x_i , $i = 0, \dots, n$ 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

$$\langle f, g \rangle = \int_a^b w(x) f(x) g(x) dx, \quad w(x) > 0. \quad (39)$$

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

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

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 (39) we also have

$$\langle xf, g \rangle_w = \int_a^b w(x) x f(x) g(x) dx = \langle f, xg \rangle_w. \quad (40)$$

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

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

If we can make such a sequence, then

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

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:

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

Let us now assume that we have found ϕ_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, ϕ_k is a polynomial of degree k , and α_j can be chosen so that $\langle \phi_k, \phi_i \rangle_w = 0$, $i = 0, 1, \dots, k-1$, or

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

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 ϕ_{k-1} is orthogonal to all polynomials of degree $k-2$ or less, we get

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

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

Theorem 9.1. *The sequence of orthogonal polynomials can be defined as follows:*

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

with

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

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 9.2. For the inner product

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

we get

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

These are the well known Legendre polynomials.

Example 9.3. Let $w(x) = 1/\sqrt{1-x^2}$, and $[a, b] = [-1, 1]$. We then get the sequence of polynomials:

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

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

The following theorem will become useful:

Theorem 9.4. 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 $t_1 < t_2 < \dots < t_r$. Then f will not change sign on each of the subintervals:

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

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 9.5. The orthogonal polynomial ϕ_k has exactly k distinct zeros in (a, b) .

10 Solution of systems of nonlinear equations

Given a system of nonlinear equations

$$F(x) = 0, \quad F : \mathbb{R}^m \rightarrow \mathbb{R}^m \quad (41)$$

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

$$x = G(x), \quad G : \mathbb{R}^m \rightarrow \mathbb{R}^m. \quad (42)$$

The solution x^* of (41) should satisfy $x^* = G(x^*)$, and is thus called a *fixed point* of G . The iteration schemes becomes: given an initial guess $x^{(0)}$, the *fixed point iterations* becomes

$$x^{(k+1)} = G(x^{(k)}), \quad k = 1, 2, \dots \quad (43)$$

The following questions arise:

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

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

Theorem 10.1. *Let $D \subseteq \mathbb{R}^m$ be a convex² and closed set. If*

$$G(D) \subseteq D \quad (44a)$$

and

$$\|G(y) - G(v)\| \leq L\|y - v\|, \quad \text{with } L < 1 \text{ for all } y, v \in D, \quad (44b)$$

then G has a unique fixed point in D and the fixed point iterations (43) converges for all $x^{(0)} \in D$. Further,

$$\|x^{(k)} - x^*\| \leq \frac{L^k}{1-L} \|x^{(1)} - x^{(0)}\|. \quad (44c)$$

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

$$\|x^{(l)} - x^{(k)}\| < \varepsilon \quad \text{for all } l, k > N. \quad (45)$$

Assumption (44a) ensures $x^{(k)} \in D$ as long as $x^{(0)} \in D$. From (43) and (44b) we get:

$$\|x^{(k+1)} - x^{(k)}\| = \|G(x^{(k)}) - G(x^{(k-1)})\| \leq L\|x^{(k)} - x^{(k-1)}\| \leq L^k \|x^{(1)} - x^{(0)}\|.$$

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

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

² 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 (45), so the sequence converges to some $x^* \in D$. Since the inequality is true for all $p > 0$ it is also true for x^* , proving (44c).

To prove that the fixed point is unique, let x^* and y^* be two different fixed points in D . Then

$$\|x^* - y^*\| = \|G(x^*) - G(y^*)\| < \|x^* - y^*\|$$

which is impossible. \square

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 $x = [x_1, \dots, x_m]^T$ and $G(x) = [g_1(x), \dots, g_m(x)]^T$. Let $y, v \in D$, and let $x(\theta) = \theta y + (1 - \theta)v$ be the straight line between y and v . The mean value theorem for functions gives

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

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

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

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

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

then

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

We can then conclude that (44b) 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

$$G(x^{(k)}) = x^{(k)} - J(x^{(k)})^{-1}F(x^{(k)}),$$

where the *Jacobian* is the matrix function

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

It is possible to prove that if *i*) (41) has a solution x^* , *ii*) $J(x)$ is nonsingular in some open neighbourhood around x^* and *iii*) the initial guess $x^{(0)}$ is sufficiently close to x^* , the Newton iterations will converge to x^* and

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

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

Steepest descent

Steepest descent is an algorithm that search for a (local) minimum of a given function $g : \mathbb{R}^m \rightarrow \mathbb{R}$. The idea is as follows.

- Given some point $x \in \mathbb{R}^m$.
- Find the direction of steepest decline of g from x (steepest descent direction)
- Walk steady in this direction till g starts to increase again.
- Repeat from a).

The direction of steepest descent is $-\nabla g(x)$, where the gradient ∇g is given by

$$\nabla g(x) = \left[\frac{\partial g}{\partial x_1}(x), \dots, \frac{\partial g}{\partial x_m}(x) \right]^T.$$

And the steepest descent algorithm reads

function STEEPEST DESCENT($g, x^{(0)}$)

for $k=0,1,2,\dots$ **do**

$$z = -\nabla g(x^{(k)}) / \|\nabla g(x^{(k)})\|$$

\triangleright The steepest descent direction.

 Minimize $g(x^{(k)} + \alpha z)$, giving $\alpha = \alpha^*$.

$$x^{(k+1)} = x^{(k)} + \alpha^* z$$

end for

end function

This algorithm will always converge to some point x^* in which $\nabla g(x^*) = 0$, usually a local minimum, if one exist. But the convergence can be very slow.

This can be used to find solution of the nonlinear system of equations (41) by defining

$$g(x) = F(x)^T F(x) = \|F(x)\|_2^2.$$

Thus, x^* is a minimum of $g(x)$ if and only if x^* is a solution of $F(x) = 0$. In this case, we can show that

$$\nabla g(x) = 2J(x)^T F(x).$$