Error concepts:

$$\mathbf{d}_{n+1} = \mathbf{y}(t_n + h\,;\, t_n, \mathbf{y}(t_n)) - \Big(\mathbf{y}(t_n) + h\boldsymbol{\Phi}(t_n, \mathbf{y}(t_n); h)\Big), \quad \text{the local truncation error}$$

$$\mathbf{l}_{n+1} = \mathbf{y}(t_n + h\,;\, t_n, \mathbf{y}_n) - \Big(\mathbf{y}_n + h\boldsymbol{\Phi}(t_n, \mathbf{y}_n; h)\Big), \quad \text{the local error}$$

$$\mathbf{le}_{n+1} = h\Big(\hat{\boldsymbol{\Phi}}(t_n, \mathbf{y}_n; h) - \boldsymbol{\Phi}(t_n, \mathbf{y}_n; h)\Big), \quad \text{the local error estimate,} \quad \mathbf{le}_{n+1} \approx \mathbf{l}_{n+1}$$

$$\mathbf{e}_n = \mathbf{y}(t_n) - \mathbf{y}_n \quad \text{the global error}$$

1.6 Stiff ODEs

Let us start this section by an example:

Example 1.6.1. Given the following system of two ODEs

$$\begin{aligned} y_1' &= -2y_1 + y_2 + 2\sin(t), & y_1(0) &= 2, \\ y_2' &= (a-1)y_1 - ay_2 + a\,\big(\cos(t) - \sin(t)\big), & y_2(0) &= 3, \end{aligned}$$

where a is some positive parameter. The exact solution, which is independent of the parameter, is

$$y_1(t) = 2e^{-t} + \sin(t), \qquad y_2(t) = 2e^{-t} + \cos(t).$$

Solve this problem now with some adaptive ODE solver, for instance the Heun-Euler scheme.

Now try the tolerances Tol $= 10^{-2}$, 10^{-4}, 10^{-6}, and perform the experiment with two different values of the parameters, $a = 2$ and $a = 999$. For $a = 2$ the expected behaviour is observed, for higher accuracy, more steps are required. But the example $a = 999$ requires much more steps, and the step size seems almost independent of the tolerance, at least for Tol $= 10^{-2}$, 10^{-4}.

The example above with $a = 999$ is a typically example of a *stiff ODE*. What defines these types of ODEs is that there are (at least) two different time scales at play at the same time: a slow time scale that dominates the time evolution of the solution of the ODE, and a fast time scale at which small perturbations of the solution may occur. In physical systems, this might be due to very strong damping of selected components of the system.

Let us study example 1.6.1 a bit closer: The general solution of the ODE can be shown to be

$$\mathbf{y}(t) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-t} + c_2 \begin{pmatrix} -1 \\ a-1 \end{pmatrix} e^{-(a+1)t} + \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}$$

for some constants c_1, c_2. The terms e^{-t}, $\sin(t)$, and $\cos(t)$ evolve at a time scale of order 1. In contrast, the term $e^{-(a+1)t}$ reverts back to being essentially equal to zero at a time scale of order $1/(a+1)$.

When a stiff ODE is solved by some explicit adaptive method like the Heun-Euler scheme, an unreasonably large number of steps is required, and this number seems independent of the tolerance. The problem is that, for explicit methods, the local error is dominated by

what is happening at the fast time scale, and the step length will be adapted to that time scale as well. Even worse, any larger step size will lead to instabilities and exponentially increasing oscillations in the numerical solution.

In the remaining part of this note we will explain why this happens, and how we can overcome the problem. For simplicity, the discussion is restricted to linear problems, but also nonlinear ODEs can be stiff, and often will be.

Exercise 1.6.1. Repeat the experiment on the Van der Pol equation

$$\begin{aligned} y_1' &= y_2, & y_1(0) &= 2, \\ y_2' &= \mu(1 - y_1^2)y_2 - y_1, & y_2(0) &= 0. \end{aligned}$$

Use $\mu = 2$, $\mu = 5$ and $\mu = 50$.

1.6.1 Linear stability analysis

The phenomenon observed above can be analyzed by studying the very simple linear test equation

$$y' = \lambda y, \qquad y(0) = y_0,$$

where the parameter $\lambda \in \mathbb{C}$ satisfies

$$\Re\lambda < 0.$$

and represent the fastest decaying component in a system. A more extensive explanation will be given below. Here, and in the following, $\Re\lambda$ will denote the real part of λ, and $\Im\lambda$ will denote the imaginary part of λ.

The analytic solution of this problem is

$$y(x) = y_0\, e^{\lambda x} = y_0\, e^{\Re\lambda\, x}\big(\cos(\Im\lambda\, x) + i\sin(\Im\lambda\, x)\big).$$

Since $\Re\lambda < 0$, the solution $y(x)$ tends to zero as $x \to \infty$. We want a similar behaviour for the numerical solution, that is $|y_n| \to 0$ when $n \to \infty$.

One step of some Runge–Kutta method applied to the linear test equation can always be written as

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

The function $R(z)\colon \mathbb{C} \to \mathbb{C}$ is called the *stability function* of the method. $R(z)$ is either a polynomial or a rational function of z.

Example 1.6.2. The application of Euler's method for the linear test equation results in the sequence

$$y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n = (1 + z)y_n \qquad \text{with } z = h\lambda.$$

The stability function of Euler's method is therefore the function

$$R(z) = 1 + z.$$

For a comparison, Heun's method for this test equation is

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

which can be rewritten as

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

As a consequence, the stability function for Heun's method is

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

One step with the Trapezoidal rule is

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

and the corresponding stability function is

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

We now return back to the analysis of the behaviour of an arbitrary Runge-Kutta method with stability function R. Taking the absolute value on each side of the expression

$$y_{n+1} = R(z)y_n,$$

we see that there are three possible outcomes:

$$\begin{array}{lllll} |R(z)| < 1 & \Rightarrow & |y_{n+1}| < |y_n| & \Rightarrow & y_n \to 0 \qquad \text{(stable)} \\ |R(z)| = 1 & \Rightarrow & |y_{n+1}| = |y_n| & & \\ |R(z)| > 1 & \Rightarrow & |y_{n+1}| > |y_n| & \Rightarrow & |y_n| \to \infty \qquad \text{(unstable)} \end{array}$$

The *stability region* of a method is defined by

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

To get a stable numerical solution, we have to choose the step size h such that $z = \lambda h \in \mathcal{S}$.

Example 1.6.3. In the case of Euler's method, we have obtained the stability function

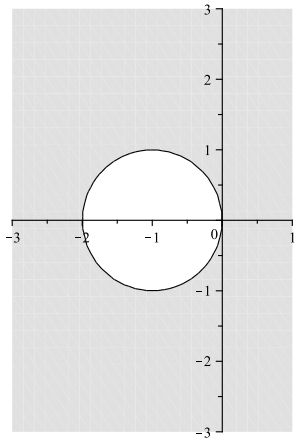
$$R(z) = 1 + z.$$

As a consequence, the stability region for Euler's method is

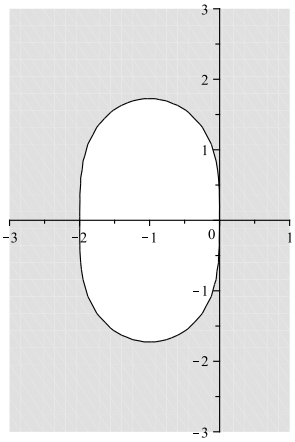
$$\mathcal{S} = \{z \in \mathbb{C} \ : \ |1 + z| \leq 1\}.$$

This is a ball in the complex plane, which is centered at -1 and has a radius of 1.

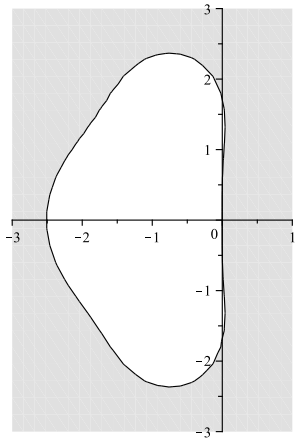
In Fig. 1.2 the stability regions for some common methods are presented.



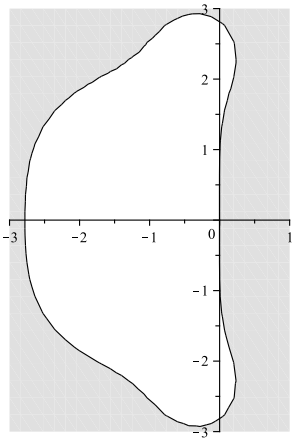
$p = s = 1$



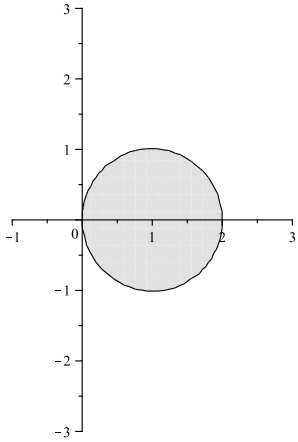
$p = s = 2$



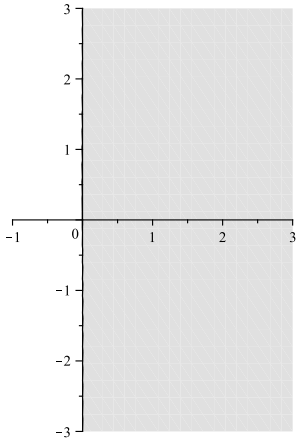
$p = s = 3$



$p = s = 4$



Backward Euler



Trapezoidal rule

Figure 1.2: 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.

1.6.2 Linear systems of ODEs.

We are given a system of m differential equation of the form

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(x). \qquad (*)$$

Such systems have been discussed in Mathematics 3, and the technique for finding the exact solution will shortly be repeated here:

Solve the homogenous system $\mathbf{y}' = A\mathbf{y}$, that is, find the eigenvalues λ_i and the corresponding eigenvectors $\mathbf{v}_i$ satisfying

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \qquad i = 1, 2, \ldots, m. \qquad (**)$$

Assume that A has a full set of linearly independent (complex) eigenvectors $\mathbf{v}_i$ with corresponding (complex) eigenvalues λ_i. Let $V = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$, and $\Lambda = \mathrm{diag}\{\lambda_1, \ldots, \lambda_m\}$. Then V is invertible and

$$AV = V\Lambda \qquad \text{and therefore} \qquad V^{-1}AV = \Lambda.$$

The ODE (*) can thus be rewritten as

$$V^{-1}\mathbf{y}' = V^{-1}AVV^{-1}\mathbf{y} + V^{-1}\mathbf{g}(t).$$

Let $\mathbf{z} = V^{-1}\mathbf{y}$ and $\mathbf{q}(t) = V^{-1}\mathbf{g}(t)$ such that the equation can be decoupled into a set of independent scalar differential equations

$$\mathbf{z}' = \Lambda\mathbf{z} + \mathbf{q}(t) \qquad \text{or, equivalently} \qquad z_i' = \lambda_i z_i + q_i(t), \quad i = 1, \ldots, m.$$

The solution of such equations has been discussed in Mathematics 1. When these solutions are found, the exact solution is given by

$$\mathbf{y}(t) = V\mathbf{z}(t),$$

and possible integration constants are given by the initial values.

As it turns out, the eigenvalues $\lambda_i \in \mathbb{C}$ are the key to understanding the behaviour of the ODE integrators, and it motivates the study of the stability properties of the very simplified, though complex, linear test equation

$$y' = \lambda y, \qquad \lambda \in \mathbb{C}^-.$$

The discussion below is also relevant for nonlinear ODEs $\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}(t))$, in which case we have to consider the eigenvalues of the Jacobian $\mathbf{f}_\mathbf{y}$ of $\mathbf{f}$ with respect to $\mathbf{y}$.

We now return to the introductory example. There, the ODE can be written as

$$\mathbf{y}' = A\mathbf{y} + \mathbf{g}(t),$$

with

$$A = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix}, \qquad \mathbf{g}(t) = \begin{pmatrix} \sin(t) \\ a(\cos(t) - \sin(t)) \end{pmatrix}.$$

The eigenvalues of the matrix A are $\lambda_1 = -1$ and $\lambda_2 = -(a+1)$. The general solution is given by

$$\mathbf{y}(t) = c_1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} e^{-t} + c_2 \begin{pmatrix} -1 \\ a-1 \end{pmatrix} e^{-(a+1)t} + \begin{pmatrix} \sin(t) \\ \cos(t) \end{pmatrix}.$$

In the introductory example, the initial values were chosen such that $c_1 = 2$ and $c_2 = 0$. However, for large values of a, the term $e^{-(a+1)t}$ will still go to 0 almost immediately, even if $c_2 \neq 0$. It is this term that creates problems for the numerical solution.

Example 1.6.4. We have already discussed the stability function and stability region for Euler's method in the example above. We now solve the introductory problem

$$\mathbf{y}' = \begin{pmatrix} -2 & 1 \\ a-1 & -a \end{pmatrix} \mathbf{y} + \begin{pmatrix} \sin(t) \\ a(\cos(t) - \sin(t)) \end{pmatrix}, \qquad \mathbf{y}(0) = \begin{pmatrix} 2 \\ 3 \end{pmatrix}, \qquad a > 0.$$

by Euler's method. We know that the eigenvalues of the matrix A are $\lambda_1 = -1$ and $\lambda_2 = -(1+a)$.

For the numerical solution to be stable for both eigenvalues, we have to require that the step length h satisfies

$$|1 + \lambda_1 h| \leq 1 \qquad \text{and} \qquad |1 + \lambda_2 h| \leq 1.$$

Since both eigenvalues in this case are real and negative, we see after a short computation that this results in the requirement that

$$h \leq \frac{2}{1+a}.$$

Try $a = 9$ and $a = 999$. Choose step sizes a little bit over and under the stability boundary, and you can experience that the result is sharp. If h is just a tiny bit above, you may have to increase the interval of integration to see the unstable solution.

1.7 A-stable methods.

In an ideal world, we would prefer the stability region to satisfy

$$\mathcal{S} \supset \mathbb{C}^- := \{z \in \mathbb{C} \ : \ \Re z \leq 0\},$$

such that the method is stable for all $\lambda \in \mathbb{C}$ with $\Re\lambda \leq 0$ and for all h. Such methods are called A-stable. For all explicit methods, like Euler's and Heun's, the stability function will be a polynomial, and $|R(z)| \to \infty$ as $\Re z \to -\infty$. Explicit methods can never be A-stable, and we therefore have to search among implicit methods. The simplest of those is the implicit, or backward, Euler's method, given by

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

Applied to the linear test equation $y' = \lambda y$, this results in the update

$$y_{n+1} = y_n + h\lambda y_{n+1} \qquad \text{or} \qquad y_{n+1} = \frac{1}{1 - h\lambda} y_n.$$

We therefore see that we have the stability function

$$R(z) = \frac{1}{1-z}.$$

The stability region for the implicit Euler function is thus

$$\mathcal{S} = \left\{ z \in \mathbb{C} \ : \ \left| \frac{1}{1-z} \right| \leq 1 \right\} = \{ z \in \mathbb{C} \ : \ |1-z| \geq 1 \}.$$

This is the whole complex plan apart from an open ball with center $+1$ and radius 1. Thus the method is A-stable, as every complex number z with $\Re z \leq 0$ is contained in $\mathcal{S}$.

1.7.1 Adaptive methods.

Implicit Euler is a method of order 1, and the trapezoidal rule of order 2. Thus, these can be used for error estimation: Perform one step with each of the methods, use the difference between the solutions as an error estimate, and use the solution from the trapezoidal rule to advance the solution.

Example 1.7.1. Repeat the experiment from the introduction, using a combination of backward Euler and the trapezoidal rule.

We observe that there are no longer any step size restriction because of stability. The algorithm behaves as expected.

Comment: Implicit methods can of course also be applied for nonlinear ODEs. Implicit Euler's method will be

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x, \mathbf{y}_{n+1}),$$

which is a nonlinear system which has to be solved for each step. Similar for the trapezoidal rule. Usually these equations are solved by Newton's method or some simplification of it.

For an implementation example, see the programming exercise of the exam May 2023.

1.8 Newton's method for nonlinear equations

We start with the scalar equation

$$f(x) = 0.$$

(**NB!** Everything that follows is formulated for an equation of the form $f(x) = 0$; if the equation you have is given in any other way, you first have to rewrite it in the form $f(x) = 0$.)

Our idea is to set up an iterative method for the solution of this equation. That is, we assume that we have already found some x_k that is reasonably close to the solution. We want to find an update Δ_k such that $x_{k+1} := x_k + \Delta_k$ satisfies $f(x_{k+1}) \approx 0$. Using a first order Taylor series expansion, we now obtain that

$$0 \approx f(x_k + \Delta_k) = f(x_k) + \Delta_k f'(x_k) + O(\Delta_k^2) \approx f(x_k) + \Delta_k f'(x_k).$$

Solving this (approximate) equation for the update Δ_k, we see that we should choose it as !bt

$$\Delta_k = -\frac{f(x_k)}{f'(x_k)}.$$

Or, using that $x_{k+1} = x_k + \Delta_k$, we get the well known Newton's method

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \qquad k = 0, 1, \ldots .$$

As a stopping criterion, use either $|\Delta_k| < \text{tol}$ or $|f(x_k)| \leq \text{tol}$ or a combination of those.

Example 1.8.1. Solve $f(x) = x^3 + x^2 - 3x - 3 = 0$ by Newton's method. Choose $x_0 = 1.5$ and stop the iterations when $|\Delta_k| < 1.e - 10$.

1.8.1 Newton's method for systems

In this section we will discuss how to solve systems of non-linear equations, given by

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

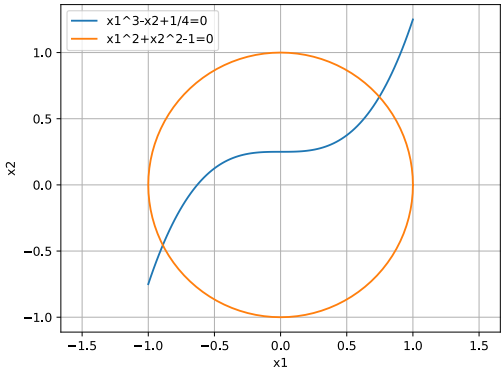
or short by

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}.$$

where $\mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m$.

Example 1.8.2. Given the two equations

$$x_1^3 - x_2 + \frac{1}{4} = 0$$
$$x_1^2 + x_2^2 - 1 = 0$$



The solutions are in the intersections of the two graphs. Thus, there are two solutions, one in the first and one in the third quadrant.

So how to construct an iteration scheme similar to Newton's method for a scalar equation to a system? To avoid completely getting lost in indices, only discuss systems of two equations and two unknowns are discussed here, that is:

$$f(x, y) = 0$$
$$g(x, y) = 0$$

Let $\mathbf{x}^* = [x^*, y^*]^\top$ be an exact solution of these equations and some $\mathbf{x}_k = [x_k, y_k]^\top$ a known approximation to $\mathbf{x}^*$, starting with $\mathbf{x}_0$. We search for a better approximation $\mathbf{x}_{k+1}$.

Let $\delta\mathbf{x}_k = \mathbf{x}^* - \mathbf{x}_k$ be the displacement error of iteraton k. Then, by a multidimensional Taylor exapansion around the known value $\mathbf{x}_k$ we get

$$0 = f(x^*, y^*) = f(x_k + \delta x_k, y_k + \delta y_k) = f(x_k, y_k) + \frac{\partial f}{\partial x}(x_k, y_k)\delta x_k + \frac{\partial f}{\partial y}(x_k, y_k)\delta y_k + \mathcal{O}(\delta\mathbf{x}_k^2),$$
$$0 = g(x^*, y^*) = g(x_k + \delta x_k, y_k + \delta y_k) = g(x_k, y_k) + \frac{\partial g}{\partial x}(x_k, y_k)\delta x_k + \frac{\partial g}{\partial y}(x_k, y_k)\delta y_k + \mathcal{O}(\delta\mathbf{x}_k^2).$$

where $\mathcal{O}(\delta\mathbf{x}_k^2)$ represents higher order terms, which are assumed to be small compared to the linear terms. By ignoring these terms we get a *linear approximation* to $\mathbf{f}(\mathbf{x})$, and by replacing the exact error $\delta\mathbf{x}_k = \mathbf{x}^* - \mathbf{x}_k$ with its approximation $\Delta\mathbf{x}$, the whole thing can be written as a linear system of equation

$$0 = f(x_k, y_k) + \frac{\partial f}{\partial x}(x_k, y_k)\Delta x_k + \frac{\partial f}{\partial y}(x_k, y_k)\Delta y_k$$
$$0 = g(x_k, y_k) + \frac{\partial g}{\partial x}(x_k, y_k)\Delta x_k + \frac{\partial g}{\partial y}(x_k, y_k)\Delta y_k.$$

or more compact

$$\mathbf{f}(\mathbf{x}_k) + J(\mathbf{x}_k)\Delta\mathbf{x}_k = 0,$$

where the Jacobian $J(\mathbf{x})$ is given by

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x}(x,y) & \frac{\partial f}{\partial y}(x,y) \\ \frac{\partial g}{\partial x}(x,y) & \frac{\partial g}{\partial y}(x,y) \end{pmatrix}$$

It is reasonable to assume that $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$ is a a better approximation to $\mathbf{x}^*$ than $\mathbf{x}_k$. Newton's method for systems of equations can then be written as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + J^{-1}(\mathbf{x}_k)\mathbf{f}(\mathbf{x}_k), \qquad k = 0, 1, \ldots$$

and the implementable algorithm,

- Given $\mathbf{f}(\mathbf{x})$, the Jacobi matrix $J(\mathbf{x})$ and a starting value $\mathbf{x}_0$.
- For $k = 0, 1, 2, 3, \ldots$
  - Compute $\mathbf{f}(\mathbf{x}_k$ og $J(\mathbf{x}_k$.
  - Solve the linear system $J(\mathbf{x}_k\Delta\mathbf{x}_k = -\mathbf{f}(\mathbf{x}_k)$ with respect to $\Delta\mathbf{x}_k$.
  - Update: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$.

Stop the iterations when $\|\mathbf{f}(\mathbf{x}_k)\| \leq \text{Tol}$ or $\|\Delta\mathbf{x}_k\| \leq \text{Tol}$, or a combination of this.

The same algorithm is used for solving a system of m equations, in which case the Jacobian matrix becomes

$$J(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\mathbf{x}) & \frac{\partial f_1}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_1}{\partial x_n}(\mathbf{x}) \\ \frac{\partial f_2}{\partial x_1}(\mathbf{x}) & \frac{\partial f_2}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_2}{\partial x_n}(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1}(\mathbf{x}) & \frac{\partial f_m}{\partial x_2}(\mathbf{x}) & \cdots & \frac{\partial f_m}{\partial x_m}(\mathbf{x}) \end{pmatrix}.$$

A complete convergence analysis is rather hard, and will not be given here. But under certain assumptions it is possible to state

- If the problem as a solution, then Newton's method converges given that the starting values are close enough to the solution.
- But it is often hard to find sufficiently good starting values.

Example 1.8.3. Apply the Newton's method to the problem from Example 1.8.2

$$x_1^3 - x_2 + \frac{1}{4} = 0$$
$$x_1^2 + x_2^2 - 1 = 0$$

In this case, the Jabobian is

$$J(\mathbf{x}) = \begin{bmatrix} 3x_1^2 & -1 \\ 2x_1 & 2x_2 \end{bmatrix}$$

Starting with e.g. $\mathbf{x}_0 = [0.6, 0.5]^\top$, the first iteration takes the values

$$\mathbf{f}(\mathbf{x}_0) = \begin{bmatrix} -0.034 \\ -0.390 \end{bmatrix}, \quad J(\mathbf{x}_0) = \begin{bmatrix} 1.08 & -1.0 \\ 1.2 & 1.0 \end{bmatrix}, \quad \Delta\mathbf{x}_0 = \begin{bmatrix} 0.18596491 \\ 0.16684211 \end{bmatrix}, \quad \mathbf{x}_1 = \begin{bmatrix} 0.78596491 \\ 0.66684211 \end{bmatrix}$$

The first iterations are

k	x_1	x_2	$\|\Delta\mathbf{x}_k\|$
0	0.60000000	0.50000000	$0.25 \cdot 10^{-1}$
1	0.78596491	0.66684211	$3.81 \cdot 10^{-2}$
2	0.74787521	0.66493393	$1.74 \cdot 10^{-3}$
3	0.74628412	0.66562980	$2.99 \cdot 10^{-6}$
4	0.74628128	0.66563072	$9.47 \cdot 10^{-12}$
5	0.74628128	0.66563072	$1.28 \cdot 10^{-16}$