Chapter 2

Nonlinear Equations

"Roots! Bloody Roots!"

— Sepultura, *Roots bloody roots*

Solving equations is a very common operation. Most of the time, the equation one has to solve has both a left and a right-hand side. It is however customary to move everything to the left-hand side, so that one is left with the following

$$f(x) = 0. \tag{2.1}$$

An x fulfilling the equation is commonly also called a root of the function f. If the function f depends only on one variable, the problem is one-dimensional. Note that we also assume the function is nonlinear, since otherwise the solution should be quite obvious.

Often one faces the problem to solve multiple equations with multiple unknowns

$$\begin{aligned} f_1(x_1, x_2, ..., x_n) &= 0 \\ f_2(x_1, x_2, ..., x_n) &= 0 \\ &\vdots \\ f_m(x_1, x_2, ..., x_n) &= 0 \end{aligned} \tag{2.2}$$

If the number of equations equals the number of unknowns, i.e. $m = n$ one has at least the *hope* to satisfy all the equation simultaneously. Note that the emphasis is on hope, because in general there is no certainty. We write (2.2) usually in vector notation

$$\mathbf{f}(\mathbf{x}) = 0, \tag{2.3}$$

where $\mathbf{f} = [f_1, f_2, ..., f_n]^T$ is a n-dimensional real vector-valued function from some domain D to $\mathbb{R}^n$ and $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ is a n-dimensional real vector. Or in "math" notation: $f : D \to \mathbb{R}^n$ and $x \in D \subset \mathbb{R}^n$.

Before we begin, we have to state a little downer: there is in general neither uniqueness nor existence of a solution. This can be best understood by the graphs in figure 2.1. The

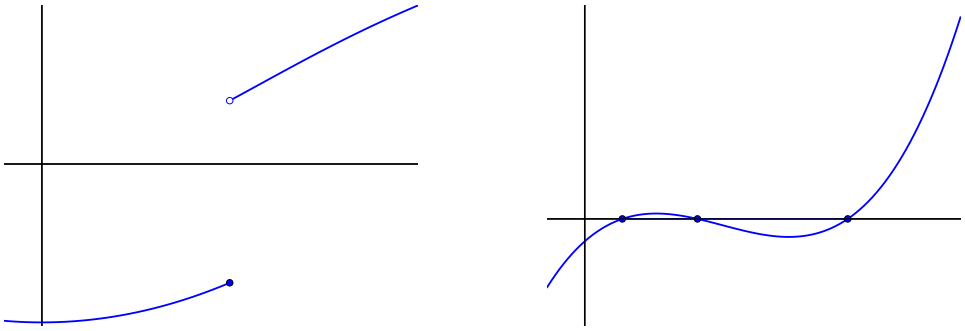

Figure 2.1: Left: Non-continuous function. Right: Multiple solutions.

left panel illustrates the case of a non-continuous function which does not pass through zero, i.e. there is no solution. The right panel illustrates the case of multiple solutions. In the latter case, it is your duty to make sure to choose the adequate solution (which indeed depends on your application).

Finding solutions to general nonlinear equations is an inherently iterative process. It is crucial to give a root-finding algorithm a good starting value or initial guess. The algorithm then improves on the initial guess by iteration. If the iterative process converges, it has to be stopped as soon as you have reached the desired precision or tolerance. How to choose this precision/tolerance usually depends on your application.

Further information can be found in [1, 2] and references therein.

2.1 Single nonlinear equations

So in this section we consider the following problem: find a zero (a.k.a. root) x^* for the (real) scalar function f, that is

$$f(x^*) = 0. \tag{2.4}$$

In the following we assume that the function is at least continuous.

To solve the problem, we will make use of iterative methods. We start from some initial value or initial guess $x^{(0)}$ and construct a sequence

$$x^{(0)} \longrightarrow x^{(1)} \longrightarrow x^{(2)} \longrightarrow ... \longrightarrow x^{(k)} \longrightarrow x^{(k+1)} \longrightarrow ... \xrightarrow{k\to\infty} x^* \tag{2.5}$$

that hopefully converges to a solution x^* of the problem.

We index the sequence by superscripts in parenthesis, which should not to be confused with exponentiation. This notation is commonly used to have the subscript free for labeling components in the multi-dimensional case.

2.1.1 Fixed-point iteration & a few generalities

The first idea is to rewrite the equation into a so-called *fixed-point iteration*

$$x^{(k+1)} = \phi(x^{(k)}) \ , \ k = 0, 1, ..., \tag{2.6}$$

where one calls the equation

$$x = \phi(x) \tag{2.7}$$

the associated *fixed-point equation*. One calls a point x^* fulfilling the fixed-point equation, i.e. $\phi(x^*) = x^*$, a *fixed-point* of ϕ. A fixed-point iteration is said to be *consistent* with the original equation if

$$f(x^*) = 0 \quad \Longleftrightarrow \quad x^* = \phi(x^*). \tag{2.8}$$

After having thrown around a few definitions, it is time for an example. Let's try to find the root of the following equation

$$f(x) = xe^x - 1 \stackrel{!}{=} 0 \tag{2.9}$$

for $x \in [0, 1]$. The graph of the function is shown in figure 2.2.

We can come up[1] with these three fixed-point equations

$$\begin{aligned} x = \phi_1(x) \quad &\text{with} \quad \phi_1(x) = e^{-x}, \\ x = \phi_2(x) \quad &\text{with} \quad \phi_2(x) = \frac{x^2 e^x + 1}{e^x(1+x)}, \\ x = \phi_3(x) \quad &\text{with} \quad \phi_3(x) = x - xe^x + 1. \end{aligned} \tag{2.10}$$

The following table shows the result of the first few iterations:

k	ϕ_1	ϕ_2	ϕ_3
0	0.8000000	0.9000000	0.6000000
1	0.4493290	0.6402998	0.5067287
2	0.6380562	0.5713091	0.6656338
3	0.5283184	0.5671575	0.3704946
4	0.5895956	0.5671433	0.8338514
5	0.5545515	0.5671433	-0.0858149
...	...	...	...

We observe that ϕ_2 seems to have reached a fixed-point after 4 iterations (up to the number of digits in the table). ϕ_1 seems to slowly tend to the same number. However, ϕ_3 leaves the interval $[0, 1]$, which is an indication for divergence.

Fixed-point iterations have a simple graphical interpretation. In figure 2.3 we show the result for the first few iterations. In each of the graph, the fixed-point is where the blue curve (ϕ_i, $i = 1, 2, 3$) cuts the black "diagonal" curve $y = x$. From the figure we see that ϕ_1 and ϕ_2 converge. However, ϕ_2 seems to converge faster. The ϕ_3 fixed-point iteration diverges, that is it spirals away from the fixed-point.

As an exercise, try to show the consistency of the fixed-point iterations and determine if they converge to a fixed-point with MATLAB©.

So far, we note the following then:

(i) The fixed-point equations/iterations are not unique. We will see several ways to construct them in the next subsections.

[1] We will see later how one derives these.

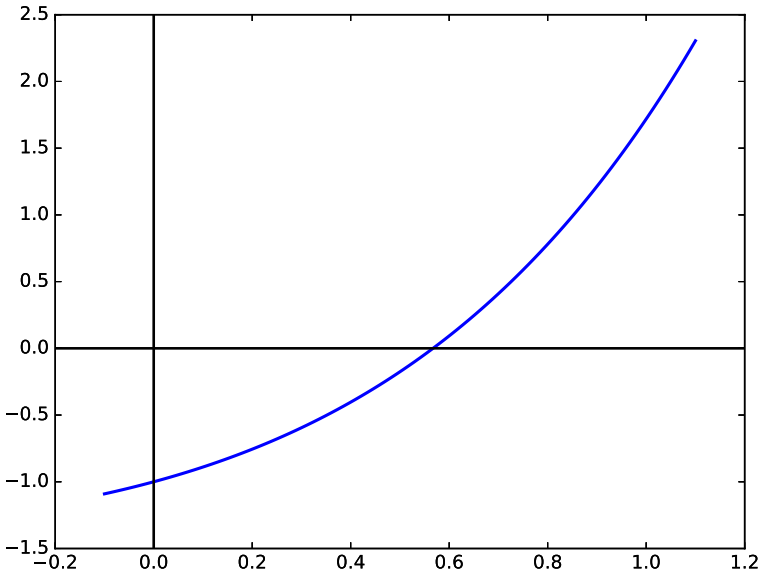

Figure 2.2: Plot of function (2.9).

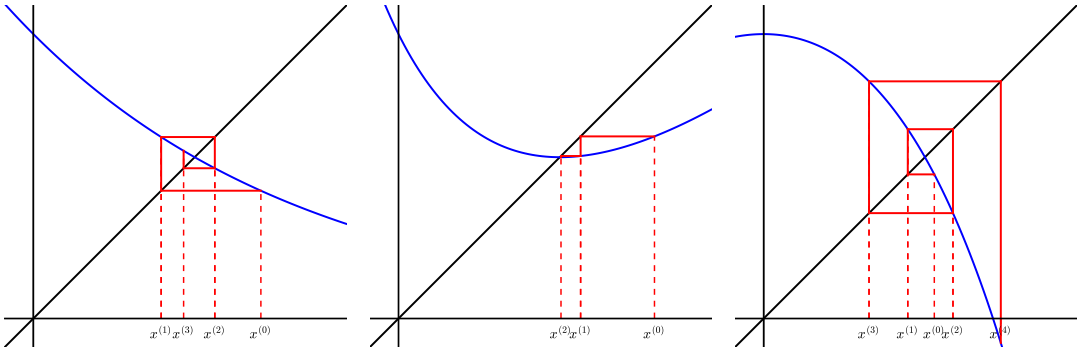

Figure 2.3: Fixed-point iterations. The blue curve is the respective fixed-point function ϕ_1 (left), ϕ_2 (middle) and ϕ_3 (right). The black "diagonal" curve is the equation $y = x$.

(ii) Fixed-point iterations may not converge.

(iii) If the fixed-point iterations converge, they may do that with different speeds.

There is actually some theory on when or when not a fixed-point iteration converges: Banach's fixed-point theorem... However, we shall not discuss the theoretical details in this course and refer to e.g. [1, Chapter 3] and [3, Chapter 5].

Instead we shall refine the notion of convergence speed. We say that a sequence $x^{(k)}$ with limit $\lim_{k\to\infty} x^{(k)} = x^*$ *converges with order* $p \geq 1$ if there exists a constant $C > 0$ such that

$$\left|x^{(k+1)} - x^*\right| \leq C \left|x^{(k)} - x^*\right|^p . \tag{2.11}$$

The constant C is called the *rate of convergence*. For $p = 1$ one additionally requires $0 < C < 1$ (for obvious reasons!). For $p = 1, 2$ we say that we have linear/quadratic convergence, respectively.

Let us try to estimate the convergence rate and order numerically. For that, we define the error at the k-th step as

$$\epsilon^{(k)} = \left| x^{(k)} - x^* \right| \tag{2.12}$$

and rewrite eq. (2.11) as

$$\epsilon^{(k+1)} \leq C \left(\epsilon^{(k)} \right)^p . \tag{2.13}$$

Suppose we know the error at the $(k+1)$,k and $(k-1)$-th iterations, so that we have the following relations among them

$$\begin{aligned} \epsilon^{(k+1)} &= C \left(\epsilon^{(k)} \right)^p \\ \epsilon^{(k)} &= C \left(\epsilon^{(k-1)} \right)^p . \end{aligned} \tag{2.14}$$

By taking the logarithm of the above equations we arrive at

$$\begin{aligned} \log\left(\epsilon^{(k+1)} \right) &= \log(C) + p \log\left(\epsilon^{(k)} \right) \\ \log\left(\epsilon^{(k)} \right) &= \log(C) + p \log\left(\epsilon^{(k-1)} \right) . \end{aligned} \tag{2.15}$$

The latter equations can easily be solved for p and C:

$$p = \frac{\log\left(\epsilon^{(k+1)} \right) - \log\left(\epsilon^{(k)} \right)}{\log\left(\epsilon^{(k)} \right) - \log\left(\epsilon^{(k-1)} \right)} \tag{2.16}$$

$$C = \frac{\epsilon^{(k+1)}}{\left(\epsilon^{(k)} \right)^p} = \frac{\epsilon^{(k)}}{\left(\epsilon^{(k-1)} \right)^p} . \tag{2.17}$$

As an exercise, try to estimate the convergence rate and order of the fixed-point iterations (2.10). Obviously only the converging ϕ_1 and ϕ_2 are interesting. As the exact solution x^*, take the result from ϕ_2 when it has reached machine precision. The following table shows what your result should look like

	ϕ_1		ϕ_2	
k	p	C	p	C
1	0.7451165	0.3489721	1.8914068	0.5859477
2	1.1866067	0.8971140	1.9832614	0.7450448
3	0.9091583	0.4305099	1.9994808	0.8143094
4	1.0560201	0.6937276	-	-
5	0.9697282	0.4999380	-	-
6	1.0176407	0.6165178	-	-
7	0.9901489	0.5382915	-	-
8	1.0056361	0.5862095	-	-
9	0.9968194	0.5556549	-	-
...	...	...	-	-

We observe that ϕ_1 converges linearly and the convergence rate is close to one. The fixed-point iteration ϕ_2 converges quadratically.

When looking for the solution of a nonlinear equation in practice, one usually wants to know the solution up to some tolerance/precision τ which is dictated by your application, that is find the smallest[2] k such that

$$\epsilon^{(k)} = \left| x^{(k)} - x^* \right| \leq \tau. \tag{2.18}$$

However, x^* is unknown to you. Otherwise you wouldn't be looking for it by numerical means! Practically, one estimates the error by computing the difference between two consecutive iterations. The iteration is stopped, when this difference is below a certain tolerance $\tilde{\tau}$

$$\left| x^{(k)} - x^{(k-1)} \right| \leq \tilde{\tau}. \tag{2.19}$$

Is this criterion enough? Let us motivate this stopping criteria for the case of a linearly convergent fixed-point iteration

$$\left| x^{(k)} - x^* \right| \leq C \left| x^{(k-1)} - x^* \right|.$$

Assuming we know the convergence rate $0 < C < 1$, we can make the following elementary manipulations

$$\begin{aligned}
\left| x^{(k)} - x^* \right| &\leq C \left| x^{(k-1)} - x^{(k)} + x^{(k)} - x^* \right| && \text{Adding a zero} \\
&\leq C \left(\left| x^{(k-1)} - x^{(k)} \right| + \left| x^{(k)} - x^* \right| \right) && \text{Triangle inequality} \\
(1 - C) \left| x^{(k)} - x^* \right| &\leq C \left| x^{(k-1)} - x^{(k)} \right| \\
\frac{1 - C}{C} \left| x^{(k)} - x^* \right| &\leq \left| x^{(k-1)} - x^{(k)} \right|.
\end{aligned}$$

In words: there is a relationship between the error at the k-th step $\epsilon^{(k)}$ and the difference between two consecutive iterations. This relationship depends on the convergence rate C and tells you how to choose $\tilde{\tau}$. Hence we conclude, that the stopping criteria (2.19) is reasonable.

2.1.2 Bisection method

The *bisection method* is a very simple and reliable root finding algorithm. Suppose you know an interval $[a, b]$ over which the function changes sign, e.g. $f(a) > 0$ and $f(b) < 0$. Under the assumption that the function is continuous, you then know there must be (at least!) one root. One says that the root is bracketed. Now bisect the interval into two subintervals $[a, c]$ and $[c, b]$, where $c = (a + b)/2$. Choose the subinterval which again brackets the root. Iterate over this process and you have the bisection method. This is illustrated in figure 2.4.

A few things to remember for the bisection method:

[2] Indeed, you don't want to waste resources.

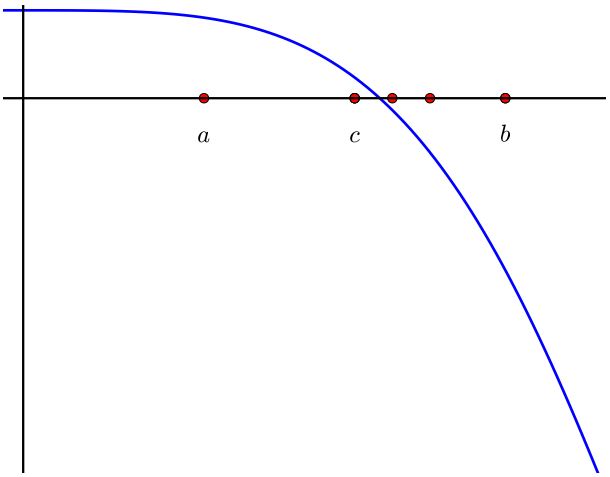

Figure 2.4: Bisection method.

(i) It's a very easy and reliable method.

(ii) We can give an *a priori* error estimate

$$\epsilon^{(k)} = \left| c^{(k)} - x^* \right| \leq \frac{1}{2^{k+1}} |b - a| \quad , \quad k = 0, 1, 2, ..., \tag{2.20}$$

where $c^{(k)}$ is the result of the bisection of the k-th interval. The bisection method converges *linearly*.

(iii) The main disadvantage of the bisection method is its slow convergence speed.

2.1.3 Newton's method

A very popular method for searching the roots of a nonlinear equation is *Newton's method*. Sometimes it's also called the *Newton-Raphson method*. Newton's method is derived from the Taylor expansion of the function $f(x)$ at the iteration's current point $x^{(k)}$

$$f(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{f''(x^{(k)})}{2}(x - x^{(k)})^2 + \cdots . \tag{2.21}$$

The expansion is then truncated to keep only the constant and linear terms

$$f(x) \approx \tilde{f}(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}), \tag{2.22}$$

i.e. we linearize the function around $x^{(k)}$. The next point in the iteration is then defined by

$$\tilde{f}(x^{(k+1)}) = f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) \stackrel{!}{=} 0, \tag{2.23}$$

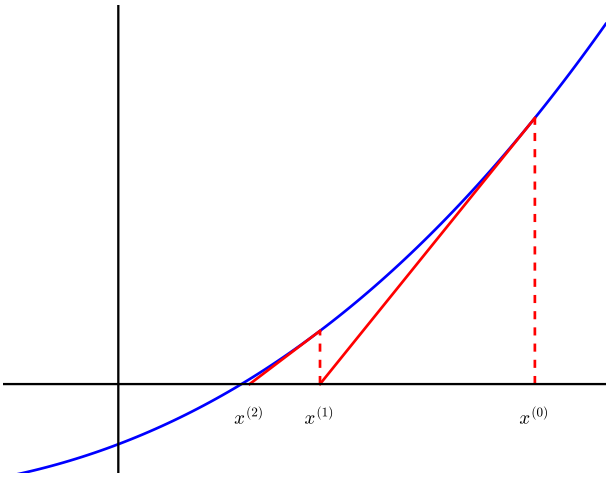

Figure 2.5: Newton finds the next value by going along the tangent of the curve until it cuts the x-axis.

which can easily be solved to give

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}. \tag{2.24}$$

Newton's method has an easy graphical interpretation: extend the tangent to the curve at the current point $x^{(k)}$ and follow it until it cuts the x-axis. The point where the tangent cuts defines your new value $x^{(k+1)}$. This is illustrated in figure 2.5.

A few remarks concerning Newton's method:

(i) It needs the function[3] and it's first derivative $f'(x)$.

(ii) It can be written as a fixed-point iteration

$$\phi(x) = x - \frac{f(x)}{f'(x)}$$

As an exercise, show the consistency!

(iii) When close enough to the root, it converges quadratically, i.e. $p = 2$.

(iv) Newton's method fails when $f'(x^{(k)}) = 0$!

[3] Obviously!

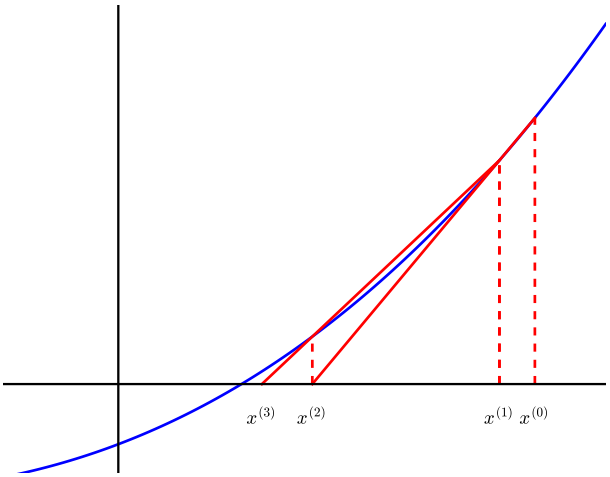

Figure 2.6: Illustration of the secant method.

2.1.4 Secant method

The so-called *secant method* is popular method when the derivative $f'(x)$ is unavailable or when it is to expensive to compute. The idea is to approximate the derivative by simple finite differences

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

Then the iteration is given by

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}. \tag{2.25}$$

The secant method has a straightforward graphical interpretation: The line through two successive iteration values $x^{(k-1)}$ and $x^{(k)}$ is extrapolated until it cuts the x-axis, which then defines the new value $x^{(k+1)}$. The method is illustrated in figure 2.6.

A few remarks concerning the secant method:

(i) It is a two-stage fixed-point iteration:

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

Therefore it needs two starting values!

(ii) One can show that if it converges, then the secant method converges with order $p = (1 + \sqrt{5})/2 \approx 1.618$, i.e. the so-called golden ratio. No need to know the exact number, just keep in mind that it's faster than linear but slower than quadratic (when near the root, of course)!

(iii) It fails when $f(x^{(k-1)}) = f(x^{(k)})$!

Methods such as the secant method that approximate the derivative in Newton's method are also referred to as *quasi-Newton methods*.

2.2 Systems of nonlinear equations

In this section we consider the problem to solve nonlinear systems of equations:

$$\mathbf{f}(\mathbf{x}) = 0, \tag{2.26}$$

where $\mathbf{f} = [f_1, f_2, ..., f_n]^T$ is a n-dimensional real vector-valued continuous function and $\mathbf{x} = [x_1, x_2, ..., x_n]^T$ is a n-dimensional real vector.

Let's make a simple two dimensional example: solve

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 + x_2 - 2 \quad &= 0 \\ f_2(x_1, x_2) &= x_2 e^{x_1} - 2 \quad &= 0 \end{aligned} \tag{2.27}$$

in the domain $D = [0, 2]^2$. In figure 2.7 we plot the contours of f_1 (blue lines) and f_2 (red lines). The sought solution is where the zero contour lines of both functions meet. You can see that this happens twice: at $\mathbf{x} = [x_1, x_2]^2 = [0, 2]^T$ and $\mathbf{x} \approx [1.2, 0.6]^T$.

In the two-dimensional case, one can use the contour plots to get an idea of where the solutions are. Unfortunately, in more than two-dimensions this is not so easy to visualize. Hence, finding roots becomes increasingly difficult. Actually, it seems virtually impossible without any good insight where to look for the root(s). This insight has to come from your application.

Below, we will look only at the "simplest" method for nonlinear systems of equations, namely Newton's method. Although one can also apply the fixed-point iteration idea here, we shall not further discuss that possibility.

2.2.1 Newton's method

As in the scalar case, the idea of Newton's method for systems is to express the function $\mathbf{f}(\mathbf{x})$ by a Taylor expansion at some point $\mathbf{x}^{(k)}$

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(k)}) + \mathrm{D}\mathbf{f}(\mathbf{x}^{(k)}) \left(\mathbf{x} - \mathbf{x}^{(k)}\right) + ..., \tag{2.28}$$

where

$$\mathrm{D}\mathbf{f} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{2.29}$$

is the *Jacobian matrix*. The function is then approximated by keeping only the constant and linear term in the expansion

$$\mathbf{f}(\mathbf{x}) \approx \widetilde{\mathbf{f}}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(k)}) + \mathrm{D}\mathbf{f}(\mathbf{x}^{(k)}) \left(\mathbf{x} - \mathbf{x}^{(k)}\right), \tag{2.30}$$

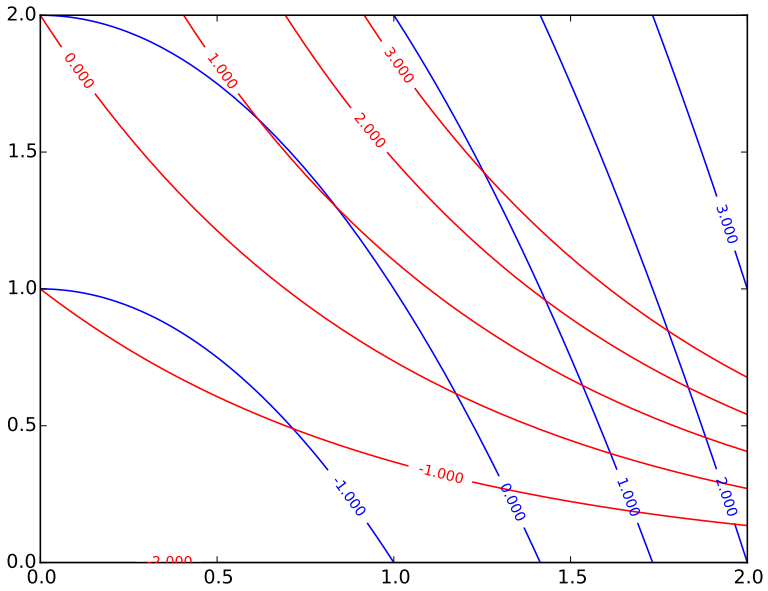

Figure 2.7: Contour lines of f_1 (blue lines) and f_2 (red lines) in (2.27).

i.e. we linearize the function $\mathbf{f}(\mathbf{x})$ at $\mathbf{x}^{(k)}$. We obtain the iteration by the solution of the linear system of equations

$$\widetilde{\mathbf{f}}(\mathbf{x}^{(k+1)}) = \mathbf{f}(\mathbf{x}^{(k)}) + \mathrm{D}\mathbf{f}(\mathbf{x}^{(k)}) \left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\right) = 0, \tag{2.31}$$

which gives Newton's method

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \mathrm{D}\mathbf{f}(\mathbf{x}^{(k)})^{-1}\mathbf{f}(\mathbf{x}^{(k)}). \tag{2.32}$$

Note that $\mathrm{D}\mathbf{f}(\mathbf{x}^{(k)})^{-1}$ is the *inverse of the Jacobian matrix*.

Let's apply Newton's method to the system of two nonlinear equations (2.27). The Jacobian matrix is given by

$$\mathrm{D}\mathbf{f}(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 2x_1 & 1 \\ x_2 e^{x_1} & e^{x_1} \end{pmatrix} \tag{2.33}$$

and it's inverse can easily be computed from the formula

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad , \quad A^{-1} = \frac{1}{ad - cb} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

to give

$$\mathrm{D}\mathbf{f}^{-1}(\mathbf{x}) = \frac{1}{(2x_1 - x_2)e^{x_1}} \begin{pmatrix} e^{x_1} & -1 \\ -x_2 e^{x_1} & 2x_2 \end{pmatrix}. \tag{2.34}$$

Implementing this into MATLAB© and using the starting value $\mathbf{x}^{(0)} = [1.9, 1.5]^T$ one gets the following sequence

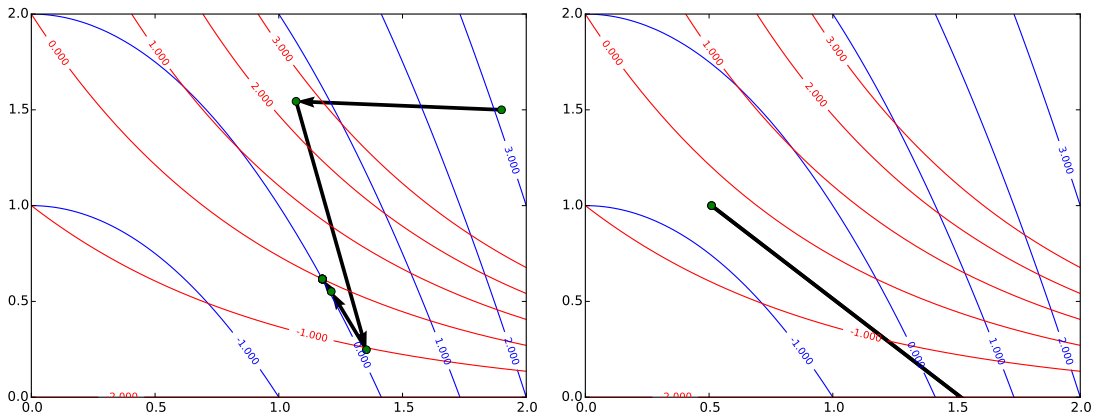

Figure 2.8: First few iterations of Newton's method applied to (2.27). Left panel: good starting value. Right panel: bad starting value.

k	$x_1^{(k)}$	$x_2^{(k)}$	$\|\mathbf{x}^{(k)} - \mathbf{x}^{(k-1)}\|_2$
0	1.9000000	1.5000000	-
1	1.0699403	1.5442267	8.3123707e-01
2	1.3539471	0.2474872	1.3274763e+00
3	1.2118524	0.5516047	3.3567596e-01
4	1.1777319	0.6141119	7.1213459e-02
5	1.1760060	0.6170128	3.3755079e-03
6	1.1760019	0.6170194	7.7670622e-06
7	1.1760019	0.6170194	4.1848107e-11
8	1.1760019	0.6170194	1.1102230e-16

We observe that the Newton method rapidly converges to $\mathbf{x}^* \approx [1.176, 0.617]^T$. This is also illustrated graphically in the left panel of figure 2.8.

In the right panel we show the result of using $\mathbf{x}^{(0)} = [0.51, 1.]^T$ as starting value. With this starting value, the Newton method rapidly wanders out of the domain $D = [0, 2]^2$ where we are looking for a solution! What happens there? Well, the Jacobian matrix is nearly singular, i.e. not invertible, at this starting value.

A few remarks concerning Newton's method for nonlinear systems:

(i) It needs the function and it's Jacobian matrix $\mathrm{D}\mathbf{f}$. In each Newton iteration one has to solve a system of linear equations (2.31).

(ii) When close enough to the root, it converges quadratically, i.e. $p = 2$.

(iii) Newton's method fails when when the Jacobian matrix is (nearly) singular. This is analogous to the scalar case when $f'(x) \approx 0$.

(iv) When implementing the Newton method, never compute the inverse of the Jacobian matrix! Rather solve the linear system

$$\mathrm{D}\mathbf{f}(\mathbf{x}^{(k)}) \left(\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)} \right) = \mathrm{D}\mathbf{f}(\mathbf{x}^{(k)}) \boldsymbol{\Delta}\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)})$$

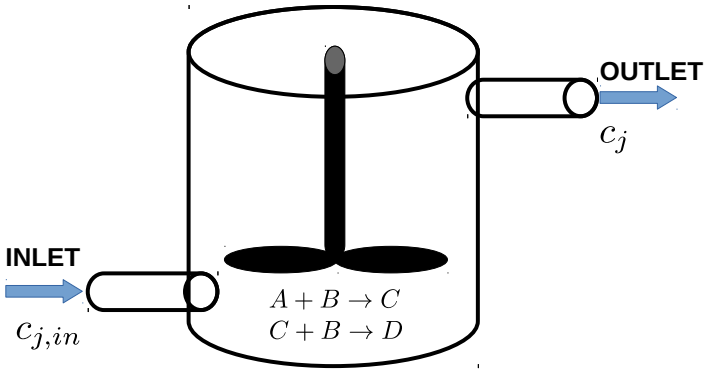

Figure 2.9: CSTR with two chemical reactions.

and iterate by $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{\Delta x}^{(k)}$. The reason for this, is that computing the inverse of matrix is n times (n is the size of the matrix) more expensive than to solve the linear system.

As a last example/exercise let's consider a Continuous Stirred-Tank Reactor (CSTR) (see figure 2.9). The reactor is operated isothermally with negligible volume change due to reactions, in inflow mode with a constant fluid volume V and with two elementary chemical reactions

$$\begin{aligned} A + B &\to C \qquad & r_{R1} &= k_1 c_A c_B \\ C + B &\to D \qquad & r_{R1} &= k_2 c_C c_B, \end{aligned} \tag{2.35}$$

where the c_j, $j = A, B, C, D$, are the respective chemical species concentrations and k_1 and k_2 the respective rate constants. Under the assumption that the reactor is perfectly mixed, the concentration of each species within the reactor is spatially homogeneous.

The concentration of each species is then dictated by the following set of mass balances

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}(V c_A) &= v\,(c_{A,in} - c_A) &&+ V\,(-k_1 c_A c_B) \\ \frac{\mathrm{d}}{\mathrm{d}t}(V c_B) &= v\,(c_{B,in} - c_B) &&+ V\,(-k_1 c_A c_B - k_2 c_C c_B) \\ \frac{\mathrm{d}}{\mathrm{d}t}(V c_C) &= v\,(c_{C,in} - c_C) &&+ V\,(+k_1 c_A c_B - k_2 c_C c_B) \\ \frac{\mathrm{d}}{\mathrm{d}t}(V c_D) &= v\,(c_{D,in} - c_D) &&+ V\,(+k_2 c_C c_B)\,, \end{aligned} \tag{2.36}$$

where v is the volumetric flow rate of the inlet and outlet, V is the reactor volume and the $c_{j,in}$, $j = A, B, C, D$, are the respective concentration of species at the inlet.

The goal is now to compute the steady-state concentrations. This means that the time derivatives on the left-hand side vanish and we get a system of four nonlinear equations.

Let's rewrite the resulting system as follows

$$\begin{aligned} v\left(c_{A,in}-x_1\right)+V\left(-k_1x_1x_2\right) &= 0 \\ v\left(c_{B,in}-x_2\right)+V\left(-k_1x_1x_2-k_2x_3x_2\right) &= 0 \\ v\left(c_{C,in}-x_3\right)+V\left(+k_1x_1x_2-k_2x_3x_2\right) &= 0 \\ v\left(c_{D,in}-x_4\right)+V\left(+k_2x_3x_2\right) &= 0, \end{aligned} \tag{2.37}$$

where we replaced $c_A = x_1$, $c_B = x_2$, $c_C = x_3$ and $c_D = x_4$. The vector function $\mathbf{f}$ is then simply

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} v\left(c_{A,in}-x_1\right)+V\left(-k_1x_1x_2\right) \\ v\left(c_{B,in}-x_2\right)+V\left(-k_1x_1x_2-k_2x_3x_2\right) \\ v\left(c_{C,in}-x_3\right)+V\left(+k_1x_1x_2-k_2x_3x_2\right) \\ v\left(c_{D,in}-x_4\right)+V\left(+k_2x_3x_2\right) \end{pmatrix}, \tag{2.38}$$

where $\mathbf{x} = [x_1, x_2, x_3, x_4]^T$. The Jacobian matrix is simply

$$\mathrm{D}\mathbf{f}(\mathbf{x}) = \begin{pmatrix} -v-Vk_1x_2 & -Vk_1x_1 & 0 & 0 \\ -Vk_1x_2 & -v-Vk_1x_1-Vk_2x_3 & -Vk_2x_2 & 0 \\ Vk_1x_2 & Vk_1x_1-Vk_2x_3 & -v-Vk_2x_2 & 0 \\ 0 & Vk_2x_3 & Vk_2x_2 & -v \end{pmatrix}. \tag{2.39}$$

We have now (nearly) everything at hand to apply Newton's method: see algorithm 1.

Given an initial guess $\mathbf{x}^{(0)}$, tolerance τ and maximum number of iterations $N_{\max}$
for $k = 0$ **to** $N_{\max}$ **do**
  % solve linear system for $\boldsymbol{\Delta}\mathbf{x}^{(k)}$
  $\mathrm{D}\mathbf{f}(\mathbf{x}^{(k)})\boldsymbol{\Delta}\mathbf{x}^{(k)} = -\mathbf{f}(\mathbf{x}^{(k)})$
  % Newton step
  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \boldsymbol{\Delta}\mathbf{x}^{(k)}$
  % Check stopping criteria
  if $\|\boldsymbol{\Delta}\mathbf{x}^{(k)}\| \leq \tau$ **then**
    Return $\mathbf{x}^{(k+1)}$
  end
end

Algorithm 1: Newton's method.

The only missing thing is a good initial guess... Use your practical insight!

2.3 Review questions

Here a few review questions[4] for the present chapter:

(a) Why does one need iterative methods?

[4] FAQs at exams...

(b) What can be said about existence and uniqueness for solutions of nonlinear equations?

(c) What methods have we seen for single nonlinear equations? Explain them and state their strengths and their requirements.

(d) What method have we seen for systems of equations? Explain it.

2.4 Bibliography

[1] Uri M. Ascher and Chen Greif. *A First Course in Numerical Methods*. Society for Industrial & Applied Mathematics (SIAM), jun 2011. doi: 10.1137/9780898719987. URL http://dx.doi.org/10.1137/9780898719987.

[2] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical recipes in FORTRAN. The art of scientific computing*. 1992.

[3] Hans Rudolf Schwarz and Norbert Köckler. Numerische mathematik. 2011. doi: 10.1007/978-3-8348-8166-3. URL http://dx.doi.org/10.1007/978-3-8348-8166-3.