TMA4215 Numerical Mathematics: Lecture notes.

Anne Kværnø

November 22, 2016

Contents

1 How to develop, analyse and test a numerical algorithm

The aim of this section is give one example of how to develop an algorithm, how to analyse it and how to perform and present the result of an numerical experiment. Numerical differentiation is chosen as an example here. Roughly the steps are:

- Given a problem, or rather a class of problems.
- Suggest an algorithm for finding a numerical approximation to the solution of the problem.
- Do some analysis of the algorithm, usually an error/convergence analysis.
- Choose carefully some test problems, and use those to confirm that the algorithm behaves as expected.
- Present the result.

So let us see how this applies to the example of numerical differentiation.

Problem: Given an $f \in C^1(a,b)$ and some arbitrary $x_0 \in (a,b)$, find a numerical approximation to $f'(x_0)$.

Algorithm: In this case, we can simply use the definition of the derivative:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h)-f(x)}{h}.$$

So our algorithm is: Given $h > 0$ (and we assume h to be small) the approximation is

$$f'(x_0) \approx \frac{f(x_0+h)-f(x_0)}{h}. \tag{1}$$

This particular approximation is called *the forward difference approximation.*

Error analysis: The error is

$$e(h; f, x_0) = f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}.$$

So the error depends on the method parameter h, which we can control, and of the problem, in this case the function f and the point x_0. In the following, we will for short use $e(h)$ unless we want to emphasise the dependence on f and/or x_0. The sign of the error does usually not matter.

To get an expression for the error, make a taylor expansion of $f(x_0+h)$:

$$e(h) = f'(x_0) + \frac{f(x_0)+hf'(x_0)+\frac{1}{2}h^2f''(\xi)}{h} = \frac{1}{2}hf''(\xi), \qquad \xi \in (x_0, x_0+h). \tag{2}$$

This is in fact an exact expression for the error, but ξ is unknown, and in practice we do not know $f''(x)$ (after all, we want an approximation for $f'(x)$. We may however add the assumption that $f''(x)$ exists and is bounded on the open interval (a,b), that is there exist a constant $M > 0$ such that

$$|f''(x)| \le M, \qquad \text{for all} \qquad x \in (a,b). \tag{3}$$

which again leads to the *error bound*

$$|e(h)| \leq \frac{1}{2}Mh \tag{4}$$

Now to complete the argument: Since (a, b) is an open interval, $x_0 \in (a, b)$ then also $x_0 + h \in (a, b)$ for sufficiently small h. From (4) we can then conclude that the approximation (1) converges to $f'(x_0)$, in the sense that

$$|e(h)| \xrightarrow[h\to 0]{} 0.$$

Numerical verification: The next step is now to verify the theory by some numerical experiments. Choose a test problem that

- satisfies all conditions of the theory,
- for which an exact solution is known (if possible),
- is not too trivial[1].

In our case, let us choose the test problem

$$f(x) = \sin(x), \qquad x_0 = 0.5. \tag{5}$$

with exact solution $f'(x) = \cos(x)$. Since $f \in C^\infty(\mathbb{R})$ and $|f^{(p)}| \leq 1$, the conditions of the theory is satisfied with $M = 1$. Let us now justify the error expression (2). We will pretend that f'' is unknown. If f'' is continuous around x_0, then for small values of h, we have $f''(\xi) \approx f''(x_0)$ so that $|e(h)| \approx Ch$, and $|e(h/2)| \approx |e(h)|/2$. And this is something that we can measure. The results the of experiment is given in Table 1.

h	$\lvert e(h)\rvert$	$\lvert e(h)/e(h/2)\rvert$
0.1	$2.54 \cdot 10^{-2}$	2.06
0.05	$1.23 \cdot 10^{-2}$	2.03
0.025	$6.08 \cdot 10^{-3}$	2.02
0.0125	$3.02 \cdot 10^{-3}$	

So, the result of this experiment complies with the error expression (2).

Convergence plots In this course, we will several time be in the situation that the error of our methods is given by

$$|e(h)| \approx Ch^p \tag{6}$$

where C is some unknown constant, and h is some method dependent parameter. In this case the method is said to be *of order p*. Assume that we have measured $e(h)$ for different values of h. The results can of course be presented in a table, as above, but it may be even more illustrative using a graph, in particular if several methods are compared. Now, taking the logarithm on both sides of (6) , we get

$$\log|e(h)| \approx p \log h + \log C,$$

[1]Well: in the process of implementing the algorithm, it is often a good idea to use examples with trivial solutions, just to rule out coding bugs.

which is a straight line with slope p. For this reason, the correct way to display the error as a function of h is to use a loglog plot. An example of this is given in Figure 1. In this plot, also a reference line ($y(h) = h^1$) is included, making it easy to see that this method really is of order 1.

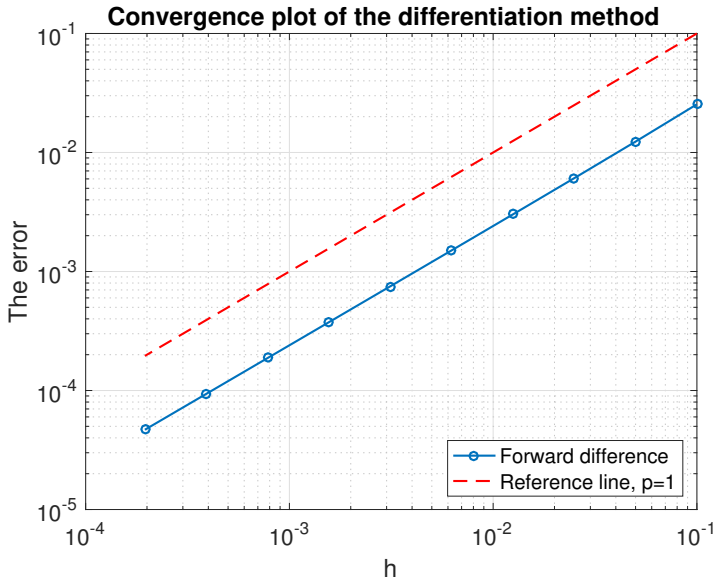


Figure 1: Errors of the forward difference approximation to $f'(x_0)$ with $f(x) = \sin x$ and $x_0 = 0.5$.

2 Error propagation

When solving problems (mathematical models) on computers, there are at least three sources of errors:

- *Data errors*: For example, input data (constants and parameters) are given from physical measurements, and are therefore subject to measurement errors.
- *Rounding errors*: Only a finite set of numbers can be represented in a computer, and each step in a sequence of operations produces a rounding error.
- *Approximation errors*: An approximation to the solution of the problem is usually found by some numerical method.

If x is the exact value, the approximated value is given by

$$\tilde{x} = x + \Delta x = x(1 + \delta x)$$

where Δx is the absolute error and $\delta x = \Delta x/x$ the relative error.

We will now see how errors in input data, represented by $x = (x_1, x_2, \ldots, x_m) \in D \in \mathbb{R}^m$ propagates in the model given by

$$y = \varphi(x), \qquad \varphi : D \to \mathbb{R},$$

where y is the output of the model. We will assume φ to be sufficiently differentiable. Given errors in the input data x_i, we get

$$y + \Delta y = \varphi(x_1 + \Delta x_1, \ldots, x_m + \Delta x_m) = \varphi(x) + \sum_{i=1}^{m} \frac{\partial \varphi}{\partial x_i} \Delta x_i$$

Here quadratic terms are ignored. Thus, the absolute error in y is given by

$$\Delta y = \sum_{i=1}^{m} \frac{\partial \varphi}{\partial x_i} \Delta x_i$$

and the relative error

$$\delta y = \frac{\Delta y}{y} = \sum_{i=1}^{m} \frac{\partial \varphi}{\partial x_i} \frac{x_i}{\varphi} \, \delta x_i.$$

From the triangle inequality, we get

$$|\delta y| \leq \sum_{i=1}^{m} \left| \frac{\partial \varphi}{\partial x_i} \frac{x_i}{\varphi} \right| \, |\delta x_i|.$$

The expression $\left| \frac{\partial \varphi}{\partial x_i} \frac{x_i}{\varphi} \right|$ is thus the factor of which the error in x_i is amplified or damped, and is often referred to as the *condition numbers* of the problem. If the condition number is $\gg 1$ the problem is ill-conditioned, otherwise it is well-conditioned. If the number is > 1, the error is amplified, so if φ is a part of a process and will be repeated several times, the complete process will be unstable.

From this, the error propagation of some common operations can be derived:

φ	Absolute error	Relative error
$y = x_1 + x_2,$	$\Delta y = \Delta x_1 + \Delta x_2,$	$\delta y = \dfrac{x_1}{x_1+x_2}\delta x_1 + \dfrac{x_2}{x_1+x_2}\delta x_2$
$y = x_1 x_2,$	$\Delta y = x_2\Delta x_1 + x_1\Delta x_2,$	$\delta y = \delta x_1 + \delta x_2,$
$y = \dfrac{x_1}{x_2},$	$\Delta y = \dfrac{1}{x_2}\Delta x_1 + \dfrac{x_1}{x_2^2}\Delta x_2,$	$\delta y = \delta x_1 - \delta x_2,$
$y = \sqrt{x},$	$\Delta y = \dfrac{1}{2\sqrt{x}}\Delta x$	$\delta y = \frac{1}{2}\delta x.$

With respect to the relative errors in y, we observe that addition is ill-conditioned if $x_1 \approx -x_2$, thus subtraction of two almost equal numbers should be avoided. All other operations in this table are well-conditioned. With respect to the absolute error, division with x_2 small compared to x_1 will amplify the errors, so will taking the square root of small numbers.

Example 2.1. *Given the quadratic equation*

$$x^2 + px + q = 0, \qquad p > 0$$

with solutions $x = (-p \pm \sqrt{p^2 - 4q})/2$. *Let us consider the computation of one of the roots,*

$$x_1 = \varphi(a, b) = \frac{-p + \sqrt{p^2 - 4q}}{2}.$$

The relative error in x_1 *is*

$$\delta x_1 = -\frac{p}{\sqrt{p^2 - 4q}}\,\delta q + \frac{p + \sqrt{p^2 - 4q}}{2\sqrt{p^2 - 4q}}\,\delta q.$$

If $p > 0$ *and* $q < 0$ *then the condition numbers satisfy*

$$\left| -\frac{p}{\sqrt{p^2 - 4q}} \right| < 1 \qquad \textit{and} \qquad \left| \frac{p + \sqrt{p^2 - 4q}}{2\sqrt{p^2 - 4q}} \right| < 1.$$

In this case, the problem is really well-conditioned. But if $p^2 \approx 4q$ *the condition numbers can be large and the problem ill-conditioned. These conclusions also hold for the second root (check it yourself).*

What about the practical computations. Let us assume the well-conditioned case, $p > 0$ *and* $q < 0$. *In the computer, the following computations will be performed to compute the two roots:*

$$\begin{aligned} r &= p^2 \\ s &= r - 4q \\ t &= \sqrt{s} \\ u_1 &= -p + t \qquad & u_2 &= -p - t \\ x_1 &= u_1/2 & x_2 &= u_2/2 \end{aligned}$$

According to the discussion above, all operations are harmless, except for possibly the computation of u_1. If $p^2 \gg -4q$ then $p \approx t$ and we have subtraction of two almost equal numbers. We can illustrate this numerically by an example: Let $p = 1.2$ and $q = -1.4 \cdot 10^{-8}$. The roots $\tilde{x}_i$ are found by the straightforward operation in matlab `x1 = (-p+sqrt(p^2-4*q))/2` *and similar for x_2. The result, together with the exact values of the roots are*

$$\begin{aligned} x_1 &= 1.166666655324074\ldots \cdot 10^{-8}, & \tilde{x}_1 &= 1.166666652174797 \cdot 10^{-8}, & |\delta x_1| &= 2.7 \cdot 10^{-9}, \\ x_2 &= -1.200000011666666\ldots, & \tilde{x}_2 &= -1.200000011666666, & |\delta x_2| &\sim \varepsilon, \end{aligned}$$

where $\varepsilon = 2.2 \cdot 10^{-16}$ is the machine precision. The error in x_1 may still seem small, but it has in fact been amplified by a factor of approximately 10^7. In this case, there is a simple remedy. Noticing that $x_1 x_2 = q$ makes it possible to compute $x_1 = q/x_2$, which is a well conditioned operation. In fact, we get

$$\tilde{x}_1 = 1.166666655324074 \cdot 10^{-8}, \qquad |\delta x_1| \sim \varepsilon.$$

To sum up:

- Condition numbers tell how much an error in input data can be amplified by the model.
- Rounding errors may cause mayhem even in well behaved-problems. Sometimes, but not always, the problem can be solved by rearranging the computations.
- Avoid subtraction of two almost equal numbers.

3 Norms and inner products on $\mathbb{R}^m$

Let $\mathbb{R}^m$ denote the set of all m-dimensional column vectors $x = [x_1, x_2, \ldots, x_m]^T$ with real-number coefficients.

Definition 3.1. *A vector norm on $\mathbb{R}^m$ is a function $\|\cdot\| : \mathbb{R}^m \to \mathbb{R}$ satisfying*

1. $\|x\| \geq 0$ *and* $\|x\| = 0 \Leftrightarrow x = 0$,
2. $\|\alpha x\| = |\alpha|\|x\|$,
3. $\|x + y\| \leq \|x\| + \|y\|$

for all $x, y \in \mathbb{R}^m$ and for all $\alpha \in \mathbb{R}$.

Some common examples of vector norms are:

$$\|x\|_1 = \sum_{i=1}^{m} |x_i| \qquad \text{The } l_1\text{-norm}$$

$$\|x\|_2 = \sqrt{\sum_{i=1}^{m} x_i^2} \qquad \text{The } l_2\text{-norm (or Euclidean norm)}$$

$$\|x\|_\infty = \max_{1 \leq i \leq m} |x_i| \qquad \text{The } l_\infty\text{-norm (or the max-norm)}$$

Example 3.2. *If $x = [1.3, -3.5, 2.4]$ then $\|x\|_1 = 7.2$, $\|x\|_2 = 4.4385$ and $\|x\|_\infty = 3.5$.*

Two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ on $\mathbb{R}^m$ are *equivalent*, that means there exist two real constants c_1 and c_2 such that for all $x \in \mathbb{R}^m$ one has that

$$c_1\|x\|_a \leq \|x\|_b \leq c_2\|x\|_a.$$

For the norms mentioned above, the following can be proved (do it!):

$$\|x\|_\infty \leq \|x\|_2 \leq \|x\|_1 \leq \sqrt{m}\|x\|_2 \leq m\|x\|_\infty.$$

Definition 3.3. *An inner product on $\mathbb{R}^m$ is a function $\langle \cdot, \cdot \rangle : \mathbb{R}^m \times \mathbb{R}^m \to \mathbb{R}$ satisfying*

1. $\langle x, y \rangle = \langle y, x \rangle$,
2. $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$,
3. $\langle x + z, y \rangle = \langle x, y \rangle + \langle z, x \rangle$,
4. $\langle x, x \rangle \geq 0$ *and* $\langle x, x \rangle = 0 \Leftrightarrow x = 0$

for all $x, y, z \in \mathbb{R}^m$ and $\alpha \in \mathbb{R}$.

The best known inner product on $\mathbb{R}^m$ is

$$\langle x, y \rangle = x^T y = \sum_{i=1}^{m} x_i y_i,$$

but there are others, as we will see later in the course.

Given an inner product, we can always define a norm by $\|x\|^2 = \langle x, x \rangle$. Such norms satisfies the *Cauchy-Schwarz inequality*:

$$|\langle x, y \rangle| \leq \|x\|\|y\|.$$

Two vectors x and y are *orthogonal* if $\langle x, y \rangle = 0$.

4 Extrapolation methods

The idea of the extrapolation methods is as follows: If it is possible to express the error of some numerical approximation as a power series of some parameter (typically a stepsize h), this information can be used to systematically cancel the lowest order error terms, and thereby obtain a higher order approximation.

Let us start with an example:

Example 4.1. *Consider the central difference formula for approximating the derivative of a function f at some point x_0,*

$$\frac{f(x_0+h)-f(x_0-h)}{2h} \approx f'(x_0).$$

By Taylor expansion of $f(x_0+h)$ and $f(x_0-h)$ around x_0 the the error can be expressed as a power series in h:

$$\frac{f(x_0+h)-f(x_0-h)}{2h} = \frac{1}{2h}\sum_{p=0}^{\infty}\left(1-(-1)^p\right)\frac{h^p}{p!}f^{(p)}(x_0) = f'(x_0)+\sum_{k=1}^{\infty}\frac{h^{(2k)}}{(2k+1)!}f^{(2k+1)}(x_0).$$

Assume we want to compute some quantity Q with some algorithm $F(h)$ where h is a method dependent parameter. Further, assume that we have an error expansion given by

$$F(h) = Q + C_1h^2 + C_2h^4 + C_3h^6 + \cdots = Q + \sum_{k=1}^{\infty} C_kh^{2k}. \tag{7}$$

in which the constants C_k depends on the problem, *but not on h*. In the example above

$$F(h) = \frac{f(x_0+h)-f(x_0-h)}{2h}, \qquad Q = f'(x_0) \qquad \text{and} \qquad C_k = \frac{f^{(2k+1)}(x_0)}{(2k+1)!}. \tag{8}$$

The idea is to compute $F(h)$ for different values of h, and use this information to systematic eliminate the error terms, and thereby obtain a higher order methods. We can use half the stepsize to get

$$F\left(\tfrac{h}{2}\right) = Q + C_1\left(\tfrac{h}{2}\right)^2 + C_2\left(\tfrac{h}{2}\right)^4 + C_3\left(\tfrac{h}{2}\right)^6 + \cdots \tag{9}$$

By subtracting (7) from 4 times this equation, and divide the whole thing by 3 we get

$$\frac{4F\left(\frac{h}{2}\right)-F(h)}{3} = Q + C_2^1h^4 + C_3^1h^6 + \cdots$$

where $C_k^1 = (4^{-k+1}-1)/3\cdot C_k$. This can be done more general: Assume

$$\begin{aligned} T_{j-1,k} &= Q + D_kh^{2k} + D_{k+1}h^{2(k+1)} + D_{k+2}h^{2(k+2)} + \cdots \\ T_{j,k} &= Q + D_k\left(\tfrac{h}{2}\right)^{2k} + D_{k+1}\left(\tfrac{h}{2}\right)^{2(k+1)} + D_{k+2}\left(\tfrac{h}{2}\right)^{2(k+2)} + \cdots \end{aligned}$$

for some constants D_k, independent of h. And by the same procedure as above, the h^{2k}-term can be elminated such that

$$T_{j,k+1} = \frac{4^kT_{j,k}-T_{j-1,k}}{4^k-1} = Q + D_{k+1}^1h^{2(k+1)} + D_{k+2}^1h^{2(k+2)} + \cdots. \tag{10}$$

This gives a systematic way of constructing higher order schemes from a lower order one,

$$T_{j,1} = F\left(\frac{h}{2^{j-1}}\right), \qquad j = 1, 2, \ldots \tag{11a}$$

$$T_{j,k+1} = \frac{4^k T_{j,k} - T_{j-1,k}}{4^k - 1}, \qquad k = 1, 2, \ldots, j-1. \tag{11b}$$

resulting in a table

$$\begin{array}{lllll} F(h) = T_{1,1} & & & & \\ F(h/2) = T_{2,1} & T_{2,2} & & & \\ F(h/4) = T_{3,1} & T_{3,2} & T_{3,3} & & \\ F(h/8) = T_{4,1} & T_{4,2} & T_{4,3} & T_{4,4} & \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Example 4.2. *Let $f(x) = \sin(x)$ and use (7) and extrapolation to find an approximation to $f'(x_0)$ for $x_0 = 0.5$. Starting with $h = 0.1$, the first four rows in the table will be*

0.876120655431924			
0.877216948194290	0.877582379115078		
0.877491149896850	0.877582550464370	0.877582561887655	
0.877559708356366	0.877582561176204	0.877582561890327	0.877582561890369

with errors $E_{i,i} = Q - T_{i,i}$:

$$E_{1,1} = 1.46 \cdot 10^{-3}, \qquad E_{2,2} = 1.83 \cdot 10^{-7}, \qquad E_{3,3} = 2.72 \cdot 10^{-12}, \qquad E_{4,4} = 3.55 \cdot 10^{-15}.$$

This has been computed by the matlab code `extrapolation.m`.

Other examples satisfying (7) and for which the algorithm (11) are applicable are:

- The trapezoidal rule for integrals $\int_a^b f(x)dx$

  $$T(h) = h\left(\frac{1}{2}f(x_0) + \sum_{j=1}^{N-1} f(x_i) + \frac{1}{2}f(x_N)\right)$$

  where $h = (b-a)/n$ for some n, and $x_i = a + ih$, $i = 0, \cdots, n$. This algorithm is better known as *Romberg integration*.

- The solution of an ODE by the implicit midpoint rule from t_0 to t_{end}

  $$k_1 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \qquad y_{n+1} = y_n + hk_1.$$

  where $F(h) = y_N$, the numerical solution at t_{end} using the stepsize $h = (t_{end} - t_0)/N$, and $Q = y(t_{end})$. For the extrapolation strategy to work in this case, the nonlinear equations has to be solved sufficiently accurate.

The strategy proposed here can be modified to other situation where the error can be expressed as power series of h, e.g. if

$$F(h) = Q + C_1 h + C_2 h^2 + C_3 h^3 + C_4 h^4 + \dots$$

It is also possible to use other sequences of stepsizes h.

But in all cases, the constants C_k depends on some derivatives of the underlying problem, so it will only work if this problem is sufficiently differentiable.

5 Solution of systems of nonlinear equations

Given a system of nonlinear equations

$$\mathbf{f}(\mathbf{x}) = 0, \qquad \mathbf{f} : \mathbb{R}^m \to \mathbb{R}^m \tag{12}$$

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

$$\mathbf{x} = \mathbf{g}(\mathbf{x}), \qquad \mathbf{g} : \mathbb{R}^m \to \mathbb{R}^m. \tag{13}$$

The solution ξ of (12) should satisfy $\xi = \mathbf{g}(\xi)$, and is thus called a *fixed point* of $\mathbf{g}$. The iteration schemes becomes: given an initial guess $\mathbf{x}^{(0)}$, the *fixed point iterations* becomes

$$\mathbf{x}^{(k+1)} = \mathbf{g}(\mathbf{x}^{(k)}), \qquad k = 1, 2, \ldots . \tag{14}$$

The following questions arise:

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

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

Theorem 5.1. *Let $D \subseteq \mathbb{R}^m$ be a and closed set. If*

$$\mathbf{g}(D) \subseteq D \tag{15a}$$

and

$$\|\mathbf{g}(\mathbf{y}) - \mathbf{g}(\mathbf{v})\| \leq L\|\mathbf{y} - \mathbf{v}\|, \qquad \text{with } L < 1 \text{ for all } \mathbf{y}, \mathbf{v} \in D, \tag{15b}$$

then G has a unique fixed point in D and the fixed point iterations (14) *converges for all $\mathbf{x}^{(0)} \in D$. Further,*

$$\|\mathbf{x}^{(k)} - \xi\| \leq \frac{L^k}{1 - L}\|\mathbf{x}^{(1)} - \mathbf{x}^{(0)}\|. \tag{15c}$$

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

$$\|\mathbf{x}^{(l)} - \mathbf{x}^{(k)}\| < \varepsilon \qquad \text{for all} \quad l, k > N. \tag{16}$$

Assumption (15a) ensures $\mathbf{x}^{(k)} \in D$ as long as $\mathbf{x}^{(0)} \in D$. From (14) and (15b) we get:

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

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

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

since $L < 1$. For the same reason, the sequence satisfy (16), so the sequence converges to some $\xi \in D$. Since the inequality is true for all $p > 0$ it is also true for ξ, proving (15c).
To prove that the fixed point is unique, let ξ and η be two different fixed points in D. Then

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

which is impossible. □

For a given problem, it is not necessarily straightforward to justify the two assumptions of the theorem. But it is sufficient to find some L satisfying the condition $L < 1$ in some norm to prove convergence.
Let $\mathbf{x} = [x_1, \ldots, x_m]^T$ and $\mathbf{g}(\mathbf{x}) = [g_1(\mathbf{x}), \ldots, g_m(\mathbf{x})]^T$. Let $\mathbf{y}, \mathbf{v} \in D$, assume D to be convex,[2] and let $\mathbf{x}(\theta) = \theta\mathbf{y} + (1-\theta)\mathbf{v}$ be the straight line between $\mathbf{y}$ and $\mathbf{v}$. According to the mean value theorem for functions, for each g_i there exist at $\tilde{\theta}_i$ such that

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

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

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

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

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

then

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

We can then conclude that (15b) is satisfied if

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

Newton's method

Newton's method is a fixed point iterations for which

$$\mathbf{g}(\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} - J_f(\mathbf{x}^{(k)})^{-1}\mathbf{f}(\mathbf{x}^{(k)}), \tag{17}$$

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

where the *Jacobian* is the matrix function

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

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

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

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

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

which is

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

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

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

which finally results in (17).

It is possible to prove, e.g. [1, Sec. 7.1] that if *i)* (12) has a solution ξ, *ii)* $J_f(\mathbf{x})$ is nonsingular in some open neighbourhood around ξ and *iii)* the initial guess $\mathbf{x}^{(0)}$ is sufficiently close to ξ, the Newton iterations will converge to ξ and

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

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

Steepest descent

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

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

The direction of steepest descent is $-\nabla\psi(\mathbf{x})$, where the gradient $\nabla\psi$ is given by

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

And the steepest descent algorithm reads

---

function STEEPEST DESCENT(ψ, $\mathbf{x}^{(0)}$)
  for k=0,1,2,.... **do**
    $\mathbf{p} = -\nabla\psi(\mathbf{x}^{(k)})/\|\nabla\psi(\mathbf{x}^{(k)})\|$ ▷ The steepest descent direction.
    Minimize $\psi(\mathbf{x}^{(k)} + \alpha\mathbf{p})$, giving $\alpha = \alpha^\star$.
    $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^\star\mathbf{p}$
  end for
end function

This algorithm will always converge to some point ξ in which $\nabla\psi(\xi) = 0$, 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 (12) by defining

$$\psi(\mathbf{x}) = \mathbf{f}(\mathbf{x})^T\mathbf{f}(\mathbf{x}) = \|\mathbf{f}(\mathbf{x})\|_2^2.$$

Thus, ξ is a minimum of $\psi(\mathbf{x})$ if and only if ξ is a solution of $\mathbf{f}(\mathbf{x}) = 0$. In this case, we can show that

$$\nabla\psi(\mathbf{x}) = 2J_f(\mathbf{x})^T\mathbf{f}(\mathbf{x}).$$

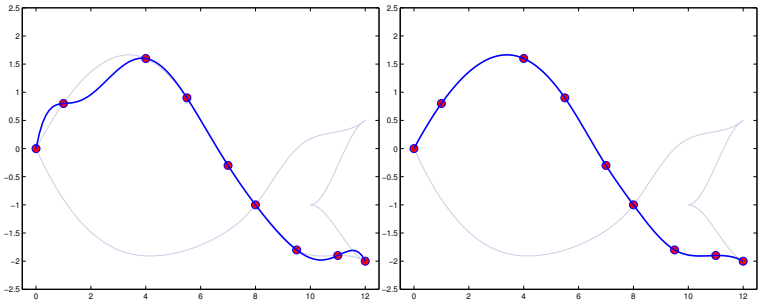
6 Cubic Splines

6.1 Introduction

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

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

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



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

Definition 6.1. *On some interval $[a, b]$, suppose that $n+1$ points $a = t_0 < t_1 < \cdots < t_n = b$ has been specified. A spline of degree k is a function S satisfying*

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

We will write the spline by

$$S(x) \quad = \quad \begin{cases} S_0(x) & x \in [t_0, t_1) \\ S_1(x) & x \in [t_1, t_2) \\ \vdots & \\ S_{n-1}(x) & x \in [t_{n-1}, t_n] \end{cases} \tag{18}$$

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

Example 6.2. *The linear spline interpolating the the points $\{t_i, y_i\}_{i=0}^n$ is given by*

$$S_i(x) = y_i \frac{x - t_{i+1}}{t_i - t_{i+1}} + y_{i+1} \frac{x - t_i}{t_{i+1} - t_i}, \qquad x \in [t_i, t_{i+1}) \qquad i = 0, 1, \ldots, n-1, \tag{19}$$

the straight lines between the points.

6.2 Cubic splines

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

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

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

$$S_i(t_i) = y_i, \qquad S_i(t_{i+1}) = y_{i+1}, \qquad i = 0, \cdots, n-1 \tag{20}$$
$$S'_{i-1}(t_i) = S'_i(t_i), \qquad i = 1, 2, \ldots, n-1 \tag{21}$$
$$S''_{i-1}(t_i) = S''_i(t_i), \qquad i = 1, 2, \ldots, n-1 \tag{22}$$

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

- Natural cubic splines: $S''(t_0) = S''(t_n) = 0$.
- Clamped cubic splines: $S'(t_0)$ and $S'(t_n)$ are specified.
- Not-a-knot condition: $S'''_0(t_1) = S'''_1(t_1)$ and $S'''_{n-2}(t_{n-1}) = S'''_{n-1}(t_n)$.
- Periodic conditions: $S'_0(t_0) = S'_{n-1}(t_n)$ and $S''_0(t_0) = S''_{n-1}(t_n)$.

We will now construct an efficient algorithm for solving finding the splines. The idea is as follows: Since S is a cubic spline, S'' is a linear spline. Let $z_i = S''(t_i)$, $i = 0, 1, \ldots, n$ (to be found). Further, let $h_i = t_{i+1} - t_i$. Then, from 19 we have that

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

So, by this, (22) is satisfied. Integrating twice gives

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

The integration constants C_i and D_i can be determined by (20), the result becomes

$$S_i(x) = \frac{z_i}{6h_i}(t_{i+1}-x)^3 + \frac{z_{i+1}}{6h_i}(x-t_i)^3 + \left(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}h_i}{6}\right)(x-t_i) + \left(\frac{y_i}{h_i} - \frac{z_i h_i}{6}\right)(t_{i+1}-x), \tag{23}$$

We now activate the second condition (21). Notice that

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

and

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

so these conditions will simply become

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

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

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

with

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

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

Input: n, $(t_i, y_i)_{i=0}^n$
  for $i = 0, 1, \ldots, n-1$ **do** ▷ Set up the linear system
    $h_i \leftarrow t_{i+1} - t_i$
    $b_i \leftarrow 6(y_{i+1} - y_i)$
  end for

  $u_1 \leftarrow 2(h_0 + h_1)$ ▷ The LU-factorization
  $v_1 \leftarrow b_1 - b_0$
  for $i = 2, 3, \ldots, n-1$ **do**
    $u_i \leftarrow 2(h_i + h_{i-1}) - h_{i-1}^2/u_{i-1}$
    $v_i \leftarrow b_i - b_{i-1} - h_{i-1}v_{i-1}/u_{i-1}$
  end for

  $z_n \leftarrow 0$ ▷ Back substitution
  for $i = n-1, n-2, \ldots, 1$ **do**
    $z_i = (v_i - h_i z_{i+1})/u_i$
  end for
  $z_0 \leftarrow 0$.

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

Theorem 6.3. *Let $f \in C^2[a,b]$. If S is the natural cubic spline interpolating f in the knots $a = t_0 < t_1 < \cdots < t_n = b$ then*

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

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

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

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

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

□

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

References

[1] Alfio Quarteroni, Riccardo Sacco, and Fausto Saleri. *Numerical mathematics*, volume 37 of *Texts in Applied Mathematics*. Springer-Verlag, Berlin, second edition, 2007.