

Accuracy and reliability in scientific computing

Ph. Caussignac
LCVMM-IMB-SB-EPFL

February 19, 2010

Abstract

In this note, we give a short review of some documents on qualitative computing together with some personal ideas. We shall present different kinds of errors occurring in scientific computing and some possible ways to prevent them. Since the results of numerical simulations are obtained by implementing algorithms on a computer, rounding errors are always present. Some errors like cancellation, recursion or integer overflow can be avoided by careful programming. Other errors remain, because the set of floating point numbers on a computer is only a subset of the real numbers. The effect of such errors can be estimated with perturbation techniques, either in a theoretical way or on the computer.

1 Errors in numerical simulation

1.1 Main sources of errors

A sketch of what is numerical simulation is given in Figure 1 (borrowed from Ref. [3]).

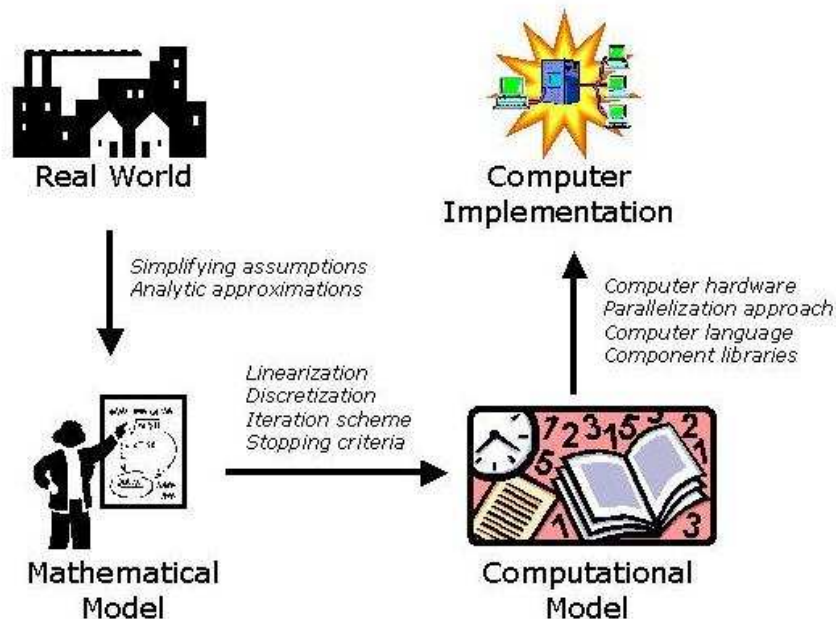


Figure 1: Numerical simulation

Errors can occur in each step of this process. Let us review, for each step, some possible sources of errors.

1. **The mathematical model** is wrong: it does not reproduce the aspect of the real world it has been designed for. The mathematical model is correct, but corresponds to an ill-posed mathematical problem, for example missing or erroneous boundary conditions.
2. Even if the mathematical model is well-posed, the resulting problem can be numerically ill-posed; for example the numerical solution can not be bounded by the data with a small constant¹. The chosen algorithm for obtaining **the computational model** is not stable or inefficient (for instance: no convergence of a linearization algorithm).
3. Most of the errors occur during the implementation process. If **syntax errors** are easy to fix, **integer overflow** or **memory overflow** can be hard to detect. Mistyping or simply wrong translation of the algorithm into the programming language result in a wrong code. Parallel implementation of the method can make a code running correctly on a one processor computer produce wrong results on a parallel machine. Floating number "errors" are always present; but, when maintained to a small order, results are correct within the machine precision. However, there are some particular cases for which they produce so large errors that the code become useless.

1.2 Computer arithmetics and related errors

In the computer, the set of real numbers \mathbb{R} is represented by the true subset \mathbb{F} of floating point numbers. Without entering into the details, a floating point number is defined by:

$$f = \pm m \cdot b^{\pm e},$$

where b is the basis (usually 2), $\pm m$ is the significant or mantissa and $\pm e$ is the exponent. Since m and e contain a fixed number of integers, we see that not all real numbers can be written in this way, in particular irrational numbers, or numbers of very large modulus. The set \mathbb{F} does not have the same algebraic structure than \mathbb{R} with respect to the arithmetic operations $+$, $-$, $*$, $/$.

For example², assume that the computer works with $b = 10$, with a 7 digits mantissa and a 2 digits exponent. The two real numbers $r_1 = 1$, $r_2 = 10^{-7}$ will be represented by the following **normalized** floating numbers:

$$\begin{aligned} f_1 &= 0.1000000 \cdot 10^{+01}, \\ f_2 &= 0.1000000 \cdot 10^{-06}. \end{aligned}$$

For the purpose of addition, these numbers must be converted to floating numbers g_1, g_2 which have the same exponent, say $e = +01$; then we have:

$$\begin{aligned} g_1 &= 0.1000000 \cdot 10^{+01}, \\ g_2 &= 0.0000000 \cdot 10^{+01}. \end{aligned}$$

Since the mantissa has seven digits, f_2 has been rounded to give g_2 and we lost the last digit. The result of the addition is

$$g_1 \hat{+} g_2 = 0.1000000 \cdot 10^{+01};$$

hence, for the computer $f_1 \hat{+} f_2 = f_1$ and the standard axiom of addition

$$r_1 + r_2 = r_1 \Leftrightarrow r_2 = 0$$

is not satisfied.

¹A well-known example is the ODE problem

$$y'(t) = 10y(t) - 10t, t > 0, y(0) = y_0,$$

whose solution

$$y(t) = t + \frac{1}{10} + (y_0 - \frac{1}{10})e^{10t}$$

can have a large error near $y_0 = 1/10$.

²This is a pure academic example because most of the computers works in the basis 2 and some bits of a "real word" are used for special purposes

This example allows us to define the number ϵ , the so-called *machine epsilon*, related to the precision of the computer, as the largest positive floating point number such that

$$\widehat{1+\epsilon}/b = \widehat{1};$$

the numbers and operations with hats relate to the machine. For most computers now, ϵ is of the order of 10^{-16} in double precision.

Let us consider the three main consequences of the computer arithmetics on the following examples.

1. Cancellation

We want to solve the quadratic equation

$$x^2 - bx + c = 0, \quad b = 742.0, c = 2.0,$$

in seven digits precision, for example C float on a standard PC. Using the usual formula, we get:

$$\begin{aligned} \Delta = \sqrt{b^2 - 4c} &= 741.9946, \quad x_1 = \frac{b + \Delta}{2} = 741.9973, \quad x_2 = \frac{b - \Delta}{2} = 0.002686, \\ \implies x_1 + x_2 &= 742.0000, \quad x_1 x_2 = 1.9927. \end{aligned}$$

In fact, one shall have $x_1 x_2 = c = 2$. Let us compute x_2 from this relation:

$$\begin{aligned} x_2 &= \frac{c}{x_1} = 0.002695, \\ \implies x_1 + x_2 &= 742.0000, \quad x_1 x_2 = 2.0000. \end{aligned}$$

Hence the result is much better. In the first way of computing x_2 , **we have subtracted two large positive real numbers which are approximately equal!**

2. Recursion

It is a very common error; we write in C float the following code:

```
n=10000; x=0.0; h=1.0/n;
while(x<=1.0) x=x+h;
```

The result is $x = 1.000054$; this means that, if we use a similar code for partitioning the interval $[0, 1]$, the point $x = 1$ is never reached and the number of points is wrong!

3. Integer overflow

On most computers, integer overflow is not signaled. Consider the following C code for computing $n!$:

```
int i, n=20, nfact=1;
for(i=1; i<=n; i++) nfact*=i;
```

The result is $20! = -2102132736$; $n!$ is correct for $n \leq 12$.

Spectacular effects of wrong software can be found at the URL:

<http://www5.in.tum.de/~huckle/bugse.html> .

1.3 Main pitfalls in numerical software coding

Once again, we shall emphasize that, since most of the numerical softwares deal with real numbers, the usual cause of errors stems from the floating number arithmetic representation in the computer. We quote just below some usual coding mistakes.

- Mathematical or physical constants are initialized to a number with too few decimals (π , ...).
- Test using equality between real numbers.
- Precision inconsistency, conversion (float \leftrightarrow double)

- Wrong stopping criterion

Some other mistakes can be very hard to detect, for example:

- The code is wrong but produces results which compare likely to the expected behavior of the model
- The numerical problem is ill-conditioned for the specific values of the parameters
- The algorithm is in fact unstable or the code is runned in a domain of instability

1.4 Validation and verification

A good software shall undergo many tests for different ranges of the parameters and the results of some specific runs serve for validation, that is comparing the output to the real world problem. Figure 2 (again from Ref. [3]) depicts the verification and validation process.

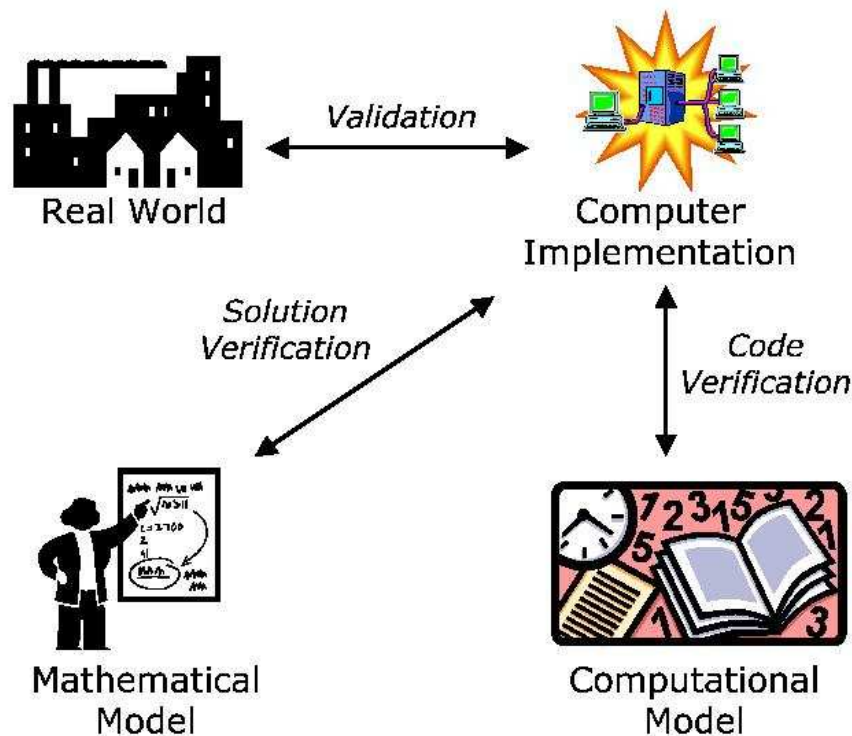


Figure 2: Validation and verification

The **code verification** is familiar to every programmer. Assuming we are coding in a high-level compiled language, the compiler can help finding syntax errors, misprints, dead code, infinite loops and bad type of variables. This kind of errors are usually searched while the code is not running: this is **the static analysis**. A careful inspection of the source code by the author, or a reading of the program by somebody else, often allow to find mistakes. **The dynamic analysis** consists in looking at the behaviour of the code while it is running: value of the variables, memory size, time spent in some particular function and so on. This process can be facilitated by an interactive debugger and a profiler. Most of the time, the author uses the method of printing some specific variables in place of the debugger. The design of **tests** is highly recommended. For an inhomogeneous problem, one can easily generate exact solutions. Take the example of the Poisson problem of finding a solution u of:

$$-\Delta u = f \text{ in } \Omega, \quad u = g \text{ on } \partial\Omega;$$

by choosing an explicit function u , one generates the data f and g just by computing $-\Delta u$ and $u|_{\partial\Omega}$. A very basic test consists in running the corresponding code with $f = 0$ and $g = 0$: one shall obtain a solution which is approximately zero. Of course, this kind of tests have in general nothing to do with the real world problem to solve and many people in applied sciences do not like them. If the programming is modular (and it should always be the case), testing each module separately is a great help for finding bugs. Also, running the code with extreme values of the parameters is very useful; for example the solution of a Navier-Stokes problem at very high viscosity will be very close to the solution of the corresponding Stokes problem which, in some specific cases, is known exactly. For numerical linear algebra problems, there are test matrices repositories.

The **solution verification** consists in checking some theoretical properties of the numerical approximation when compared to the mathematical model. For example, if the solution of the mathematical problem is in the approximation space, the numerical solution shall be equal to the exact one and this to the machine precision! But, in general, solutions of the mathematical model are not known, and this is precisely the reason for computing a numerical approximation. Nethertheless, some simple tests can be done. For an elliptic problem, one shall observe the convergence of the numerical approximation when the meshsize h is going to zero. One can plot the numerical solution at a given point for several values of h and see if there is really convergence. By considering the solution on the finest mesh as being exact, one can obtain an estimate of the approximation error which can be compared to the theoretical a priori error estimate. For a parabolic problem, we can observe if, for large time, the solution tends to a steady state. For a nonlinear problem, choosing a stricter stopping criterion often helps detecting instabilities.

The **validation of the computational model and it's computer implementation** is usually made by comparing the computed solution to some experimental data. This will not be discussed further here.

2 Arithmetic error analysis, reliability and accuracy

Even if the computational model and it's implementation are correct, the computed solution comes out with errors due essentially to the floating point representation of real numbers. There are cases for which this errors do not matter and cases where the results are wrong. The goal of making a reliability and accuracy analysis is to assert that a specific code, with the underlying numerical method and implementation, will produce results accurate to the machine precision, in a specific domain of the data or parameters.

At the time being, at least to the author knowledge, the two principal methods for ensuring reliability of a numerical computer software are interval computations ([3],[5]) and perturbation methods ([3],[4]). For other specialized methods, see Ref. [7].

Interval computations use interval arithmetics for coding algorithms. Instead of working with numbers, one works with closed intervals and the basic arithmetic operations are defined on intervals. For example, the addition and subtraction of the two intervals $\mathbf{x} = [\underline{x}, \bar{x}]$ and $\mathbf{y} = [\underline{y}, \bar{y}]$ are defined by

$$\mathbf{x} + \mathbf{y} = [\underline{x} + \underline{y}, \bar{x} + \bar{y}], \quad \mathbf{x} - \mathbf{y} = [\underline{x} - \bar{y}, \bar{x} - \underline{y}];$$

notice that for $\hat{x} \in \mathbf{x}$, $\hat{y} \in \mathbf{y}$, we have $\hat{x} + \hat{y} \in \mathbf{x} + \mathbf{y}$, $\hat{x} - \hat{y} \in \mathbf{x} - \mathbf{y}$. Thus, the exact result of these operations is in the interval obtained by the interval operations. By combining such operations, we obtained bounds for the final result. The goal of interval computations is to predict an interval where the exact result of a mathematical problem is located. But, in order of this interval to have a small size, **one has to use a specific coding** of the numerical method. Otherwise, we would get an answer like "the real number solution of this particular problem is in $(-\infty, \infty)$ ", which is always a true but useless result. With interval computations, one can prove for example the convergence of a Newton method, assert that a matrix is nearly singular by showing it has an eigenvalue in a very narrow interval including zero. The specific coding method allows of course to obtain bounds for the solution of a problem, but does not have the ability of checking the reliability of a standard numerical code.

On the contrary, **perturbation methods** yield theoretical and practical estimates of the accuracy of a numerical algorithm. Theoretical bounds for conditioning, for effects of the floating point errors by approximating the data or for the stability of the algorithm, can be proven. Existing software allow the same kind of analysis by numerical perturbations of the computer code.

2.1 Basic notions

Not every problem can be solved on a computer: it has to be numerically well-posed.

Definition 1 *A problem is numerically well-posed if*

1. *It has one and only one solution*
2. *This solution is Lipschitz-continuous with respect to the data, with a "small" constant*

The "numerical analysis" notion of conditioning characterizes the stability of the solution of a given problem with respect to the data. **The condition number** depends only on the continuous problem. Let us look at the following simple example.

Example 2 (Function evaluation) *Let us assume that we want to evaluate a given function $f \in C^2$ at a point y of its domain:*

$$x = f(y).$$

Here, the data is y and the solution is x . Since we can not represent y exactly on the computer, we have for the computed value :

$$\hat{x} = f(y + \epsilon) = f(y) + \epsilon f'(y) + O(\epsilon^2);$$

hence, we obtain, for $x \neq 0$:

$$\frac{x - \hat{x}}{x} = \frac{f'(y)}{f(y)}\epsilon + O(\epsilon^2).$$

The modulus of the relative error is asymptotically given by:

$$\left| \frac{x - \hat{x}}{x} \right| \approx \left| \frac{f'(y)}{f(y)} \right| \epsilon = \varkappa(f, y)\epsilon;$$

the quantity \varkappa is the condition number of f for its evaluation at y . Notice that, since $|x - \hat{x}| = |f'(y)|\epsilon + O(\epsilon^2)$, the solution is Lipschitz continuous with respect to the data.

In the next example, we introduce the usual condition number of a matrix for linear system solving.

Example 3 (Linear system) *For a regular matrix A of order n , a given vector $b \in \mathbb{R}^n$, we look for $x \in \mathbb{R}^n$ such that $Ax = b$ and hence $x = A^{-1}b$. The computed solution is given by*

$$\hat{A}\hat{x} = \hat{b},$$

because of errors on the representation of the data A and b in the computer and also because of arithmetic errors in the algorithm for linear system solving. Let us assume that the computed solution can be represented as the exact solution of a perturbed system³; in fact we assume the existence of a matrix E and a vector e , the norm of which are small in the following sense:

$$\|E\| \leq \epsilon \|A\|, \|e\| \leq \epsilon \|b\|, 0 < \epsilon \ll 1,$$

and such that

$$(A + E)\hat{x} = b + e.$$

We compute \hat{x} to the first order in ϵ :

$$\begin{aligned} \hat{x} &= (I + A^{-1}E)^{-1}A^{-1}(b + e) = (I - A^{-1}E)A^{-1}(b + e) + O(\epsilon^2) \\ &= A^{-1}(b + e) - A^{-1}EA^{-1}b + O(\epsilon^2) = x + A^{-1}(e - A^{-1}Ex) + O(\epsilon^2); \end{aligned}$$

thus, we get, up to order ϵ :

$$\begin{aligned} \|x - \hat{x}\| &\leq \|A^{-1}\| (\|e\| + \|E\| \|x\|) \leq \|A^{-1}\| \left(\|b\| \frac{\|e\|}{\|b\|} + \|A\| \frac{\|E\|}{\|A\|} \|x\| \right) \\ &\leq \|A^{-1}\| \|A\| \left(\frac{\|e\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right) \|x\|. \end{aligned}$$

³This assumption is known as Wilkinson's principle (see below)

Consequently, with the usual matrix condition number

$$\kappa(A) = \|A^{-1}\| \|A\|,$$

we obtain the following relative error estimate:

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \kappa(A) \left(\frac{\|e\|}{\|b\|} + \frac{\|E\|}{\|A\|} \right) \leq 2\kappa(A)\epsilon.$$

The usual relationship is obtained by setting $E = 0$.

Formally, we could define a condition number as a Lipschitz constant of the relative error. If the notion of stability is used for obtaining a condition number, this notion is primarily related to the continuous problem. In fact we could try to solve a well-conditioned problem with an unstable algorithm. In fact, the numerical method enters into the condition number definition through the Wilkinson principle, which will fail if the algorithm is unstable.

The usual notion of stability measures the sensitivity of the numerical method with respect to the data. An interesting example follows.

Example 4 We want to compute the quantity

$$y_n = \frac{1}{e} \int_0^1 x^n e^x dx$$

with a recurrence, since:

$$y_n = 1 - ny_{n-1}.$$

If we start from $n = 0$, we get:

$$y_0 = 1 - 1/e, \quad \hat{y}_0 = y_0 + \epsilon,$$

$$\hat{y}_1 = 1 - \hat{y}_0 = y_1 - \epsilon,$$

$$\hat{y}_2 = 1 - 2\hat{y}_1 = y_2 + 2\epsilon,$$

$$\hat{y}_3 = 1 - 3\hat{y}_2 = y_3 - 6\epsilon,$$

...

$$\hat{y}_n = 1 - \hat{y}_{n-1} = y_n + (-1)^n n! \epsilon.$$

Thus we see that for n large, the algorithm is unstable; there is no hope to obtain by this method y_n by a small perturbation of an exact problem. On the contrary, the recurrence starting by $n=N$:

$$y_{n-1} = \frac{1 - y_n}{n}, n = N, N-1, \dots, 1,$$

is stable!

2.2 Formal analysis of rounding errors

The following theory is presented in Ref. [2]. We consider three types of problems. The first one is the continuous problem:

$$(P) F(x) = y, \quad F : X \rightarrow Y, \quad (1)$$

where X and Y are some real vector spaces. If we assume that the maps F and F^{-1} are continuous, this problem will have the unique solution

$$x = F^{-1}(y). \quad (2)$$

For solving Problem (P), we set up a numerical approximation, which consists in discretization, linearization with a Newton method and some stopping criterion and so on. Doing so, we get the numerical problem:

$$(P_h) F_h(x_h) = y_h, \quad F_h : X_h \subset X \rightarrow Y_h \subset Y, \quad (3)$$

the solution of which, taking account of the numerical method, can be written as:

$$x_h = G_h(y_h). \quad (4)$$

Of course, the subspaces X_h and Y_h are of finite dimension, and thus isomorphic to \mathbb{R}^m and \mathbb{R}^n ; notice that there are cases where their inclusion in X and Y is not true (for instance non-conforming finite elements). At this point, the traditional numerical analysis tries to show the convergence of the algorithm, in particular using the following **Lax principle**.

Proposition 5 *Assume that the numerical method is **consistent**, that is :*

$$F_h(z) \rightarrow F(z), \|z - x\| < \delta, \quad (5)$$

$$y_h \rightarrow y, h \rightarrow 0, \quad (6)$$

and also **uniformly stable**, that is the family G_h is equicontinuous for $0 \leq h < 1$, then it is convergent:

$$x_h \rightarrow x, h \rightarrow 0. \quad (7)$$

Proof. On has

$$x - x_h = G(y) - G_h(y_h) = G(y) - G_h(y) + G_h(y) - G_h(y_h);$$

the continuity of G_h together with (6) implies that $G_h(y) - G_h(y_h) \rightarrow 0$. Since $G = F^{-1}$ and $G_h = F_h^{-1}$, one can write

$$G - G_h = G_h \circ F_h \circ F^{-1} - G_h \circ F \circ F^{-1},$$

which means that

$$G(y) - G_h(y) = G_h(F_h(x)) - G_h(F(x)),$$

and now (5) together with the equicontinuity of G_h implies that $G(y) - G_h(y) \rightarrow 0$. ■

One should notice that, the equations (5-7) make sense only if we have $X_h \subset X$ and $Y_h \subset Y$.

The following example is well-known.

Example 6 (Conforming finite elements) *Given a domain $\Omega \subset \mathbb{R}^2$ with smooth boundary, a "function" $f \in Y = L^2(\Omega)$, we look for a solution u of the following problem:*

$$\begin{aligned} -\Delta u &= f \text{ in } \Omega, \\ u &= 0 \text{ on } \partial\Omega. \end{aligned}$$

The weak formulation makes use of the space $X=H_0^1(\Omega)$, of the usual dot product (\cdot, \cdot) of $L^2(\Omega)$, and of the continuous coercive bilinear form:

$$a(u, v) = \int_{\Omega} \nabla u \cdot \nabla v \, d\mathbf{x},$$

and reads

$$u \in X : a(u, v) = (f, v) \quad \forall v \in X. \quad (8)$$

Now, the operator $F : u \in X \mapsto F(u) \in Y$ is defined by:

$$a(u, v) = (F(u), v) \quad \forall v \in X; \quad (9)$$

taking test-functions in $\mathcal{D}(\Omega)$, the space of C^∞ function with compact support, we can integrate (9) by parts and obtain:

$$(F(u), v) = (u, -\Delta v), \quad \forall v \in \mathcal{D}(\Omega),$$

which shows that F is nothing else but the Laplace operator in the sense of distributions, that is, the equation $-\Delta u = f$ holds in $X' = H^{-1}(\Omega)$. The inverse of F is simply given by $u = F^{-1}(f)$, u being the unique solution of (9):

$$a(F^{-1}(f), v) = (f, v) \quad \forall v \in X.$$

Let us come to a discretization of this problem. With a usual finite element triangulation of the domain:

$$\bar{\Omega} = \cup_{k=1}^n T_k,$$

we define the space for the lowest possible continuous interpolation, that is with polynomials of degree one:

$$X_h = \{v_h \in C_0^0(\bar{\Omega}); v_h|_{T_k} \in P^1, k = 1, 2, \dots, N\};$$

since $X \subset Y$, we will choose $Y_h = X_h$ and furthermore we define the projector $P_h : f \in L^2(\Omega) \mapsto f_h \in X_h$ by

$$(f, v_h) = (f_h, v_h) \quad \forall v_h \in X_h.$$

Of course, the unique solution u_h of the problem discretized with these particular finite elements is obtained from:

$$u_h \in X_h : a(u_h, v_h) = (P_h f, v_h) \quad \forall v_h \in X_h. \quad (10)$$

The discrete operator $F_h : u_h \in X_h \mapsto F_h(u_h) \in X_h$ is defined by:

$$a(u_h, v_h) = (F_h(u_h), v_h) \quad \forall v_h \in X_h; \quad (11)$$

it is obviously linear and consequently $F_h \in \mathcal{L}(X_h)$ can be identified with a matrix by choosing a particular basis of X_h . Denoting by $\mathcal{S} = \{S_i\}_{i=1}^N$ the set of all internal vertices of the triangulation, with coordinates $\mathbf{x}_i, 1 \leq i \leq n$, the Lagrange basis is defined by:

$$\varphi_i \in X_h, \varphi_i(\mathbf{x}_i) = 1, \varphi_i(\mathbf{x}_j) = 0, i \neq j, i, j = 1, 2, \dots, N.$$

One has:

$$v_h(\mathbf{x}) = \sum_{i=1}^N v_i \varphi_i(\mathbf{x}), v_i = v(\mathbf{x}_i), \quad \forall v_h \in X_h,$$

which means that in the Lagrange basis, the function v_h is represented by the vector $\vec{v} = (v_1, v_2, \dots, v_N)^t$. Thus we have

$$\sum_{i=1}^N v_i (F_h(u_h), \varphi_i) = \sum_{i=1}^N v_i (P_h f, \varphi_i) \quad \forall v_i \in \mathbb{R},$$

and hence the matrix is given by

$$K_{ij} = F_h(\varphi_j)_i = a(\varphi_j, \varphi_i),$$

and the right-hand-side by

$$b_i = (P_h f, \varphi_i).$$

The third problem that we introduce is the realization of the numerical algorithm on a computer with precision ϵ :

$$\left(\widehat{P}_{h\epsilon}\right) \widehat{F}_{h\epsilon}(\widehat{x}_{h\epsilon}) = \widehat{y}_{h\epsilon}, \quad \widehat{F}_{h\epsilon} : \mathbb{F}^m \rightarrow \mathbb{F}^n, \quad (12)$$

and the solution of this problem is given by:

$$\widehat{x}_{h\epsilon} = \widehat{G}_{h\epsilon}(\widehat{y}_{h\epsilon}). \quad (13)$$

We see that, starting from the continuous problem (1), we set up a numerical or discretized version (3) of it and finally, the implementation yields the problem (12).

Example 7 For the evaluation of the cosinus, we have only the two problems:

$$(P) x = \cos(y), \quad (P_\epsilon) \widehat{x}_\epsilon = \cos(\widehat{y}_\epsilon);$$

in fact there is a numerical algorithm, but it is hard coded in the computer, thus not accessible to the user.

Example 8 For linear system solving, there is no continuous problem, but only:

$$(P_h) Ax_h = y_h \quad \left(\widehat{P}_{h\epsilon}\right) \widehat{A}\widehat{x}_{h\epsilon} = \widehat{y}_{h\epsilon}.$$

The goal of rounding error analysis is to compare the solution of the implemented problem (12) to the solution of the numerical problem (3). If this task can be performed in some way on the computer, a theoretical analysis can be made using the following trick. We replace the problem (12), which involves floating numbers by a problem on real numbers:

$$(P_{h\epsilon}) \quad F_{h\epsilon}(x_{h\epsilon}) = y_{h\epsilon}, \quad F_{h\epsilon} : \mathbb{R}^m \rightarrow \mathbb{R}^n, \quad (14)$$

whose solution is given by

$$x_{h\epsilon} = G_{h\epsilon}(y_{h\epsilon}). \quad (15)$$

For general problems, the function $F_{h\epsilon}$ is not known explicitly; however, we can write it for linear algebra problems, in particular for linear systems (see below). Once we know $F_{h\epsilon}$, we can obtain estimates on $x_{h\epsilon} - x_h$ in two ways. The first way consists in comparing $G_{h\epsilon}$ to G_h ; this is the so-called **forward error analysis**. In the **backward error analysis**, we get estimates by comparing $F_{h\epsilon}$ to F_h with the help of the Lax principle applied for ϵ instead of h .

2.3 Computability, reliability and stability

Using the notions of the preceding paragraph, we introduce some definitions useful in qualitative computing. We have in mind solving the continuous problem (P), with the implementation ($P_{h\epsilon}$) of the numerical algorithm (P_h), and, in the sequel, "problem" means one of them or all the three together.

Definition 9 *A solution of the problem is **computable** (or **calculable on a computer**) if there exists \hat{x} such that $x_{h\epsilon}$ tends to \hat{x} when h and ϵ are going to zero.*

By the Lax principle, there are two sufficient conditions for computability (that is the convergence of a solution of ($P_{h\epsilon}$)): the uniform stability of this problem, that is the uniform equicontinuity of $G_{h\epsilon}$, and its consistency $F_{h\epsilon}(z)$ goes to $F(z)$ for z in a neighborhood of x .

Remark 10 *Notice that, depending on the precision, a solution can be computable on a specific computer and not on another one.*

To simplify, let us now assume that h is fixed and address only the problem of convergence of the computer implementation to a solution of (P_h); in the sequel, we drop the subscript h , having in mind that (P) stand for (P_h), (P_ϵ) for ($P_{h\epsilon}$) and so on.

Definition 11 *The **direct error of consistency** at the point z close to x is the quantity*

$$EC(G, z) = \|G_\epsilon(z) - G(z)\|. \quad (16)$$

Definition 12 *The **residual error of consistency** at the point z close to x is the quantity*

$$EC(F, z) = \|F_\epsilon(z) - F(z)\|. \quad (17)$$

With this definitions, we get a more precise definition of consistency by requiring that the error of consistency goes to zero with ϵ . Assuming that the map F is C^2 , we can hope that

$$F_\epsilon(x_\epsilon) = F(x + \Delta x), \|\Delta x\| \leq \epsilon \|x\|, \quad (18)$$

notice that because of the representation of real numbers by floating numbers, the bound for $\|\Delta x\|$ can not be enhanced. In this case, one has:

$$EC(F, x_\epsilon) = \|F'(x)\Delta x\| + O(\epsilon^2) \leq C(F, x)\epsilon. \quad (19)$$

Of course, we have the same result for the direct consistency error if G is C^2 :

$$EC(G, y_\epsilon) = \|G'(y)\Delta y\| + O(\epsilon^2) \leq C(G, y)\epsilon. \quad (20)$$

The consistency error allows us to introduce the concept of reliability.

Definition 13 *An algorithm satisfying*

$$EC(G, y_\epsilon) \leq C(G, y)\epsilon, \quad (21)$$

*is called **arithmetically reliable**.*

Consider the direct error; it can be decomposed in two parts:

$$x - x_\epsilon = [G(y) - G(y_\epsilon)] + [G(y_\epsilon) - G_\epsilon(y_\epsilon)],$$

the first one depending on the stability of the "exact" problem and the second one on the direct consistency error. Assuming that G is C^2 , one has:

$$G(y) - G(y_\epsilon) = DG(y)(y - y_\epsilon) + O(\epsilon^2)$$

and, if the algorithm is arithmetically reliable, we get the computability $x_\epsilon \rightarrow x$ when $\epsilon \rightarrow 0$.

3 Perturbation theory for linear systems

In this section, we present error estimates for linear systems, based on Wilkinson's principle of backward error analysis.

Given a regular matrix $A \in \mathcal{L}(\mathbb{R}^n)$ and a vector $b \in \mathbb{R}^n$, we want to solve the problem:

$$(P) \ x \in \mathbb{R}^n \text{ such that } Ax = b. \quad (22)$$

On the computer, this problem is replaced by

$$(\hat{P}) \ \hat{x} \in \mathbb{F}^n \text{ such that } \hat{A}\hat{x} = \hat{b}, \quad (23)$$

where \hat{A} and \hat{b} are approximations of A and b , respectively in $\mathbb{F}^{n \times n}$ and \mathbb{F}^n , $\mathbb{F} \subset \mathbb{R}$ being the set of floating point numbers.

The **Wilkinson principle** consists in replacing (\hat{P}) by an exact problem whose solution is \hat{x} :

$$(W) \ \hat{x} \in \mathbb{R} \text{ such that } (A + E)\hat{x} = b + e, \quad (24)$$

where E and e are perturbations of the original matrix and right-hand-side. The basic question concerns the existence of such perturbations, and possibly the smallest ones in a given norm, or such that the absolute value of their components are small.

Defining the residual:

$$r = b - A\hat{x}, \quad (25)$$

one notices that (24) is equivalent to:

$$E\hat{x} - e = r. \quad (26)$$

3.1 Normwise perturbation

Here, we want to bound the norm of the perturbations by the norms of a given matrix and a given vector; we follow the presentation of Ref. [4]. Given a matrix ΔA and a vector Δb , we want to find a real number $\eta_{\Delta A, \Delta b}$ such that

$$\eta_{\Delta A, \Delta b} = \min \{c : (A + E)\hat{x} = b + e, \|E\| \leq c\|\Delta A\|, \|e\| \leq c\|\Delta b\|\}. \quad (27)$$

Theorem 14 *Given ΔA and Δb , the minimal normwise backward error is given by:*

$$\eta_{\Delta A, \Delta b} = \frac{\|r\|}{\|\Delta A\| \|\hat{x}\| + \|\Delta b\|}. \quad (28)$$

Proof. Since

$$\|r\| = \|E\hat{x} - e\| \leq c(\|\Delta A\| \|\hat{x}\| + \|\Delta b\|),$$

we have

$$\eta_{\Delta A, \Delta b} \geq \frac{\|r\|}{\|\Delta A\| \|\hat{x}\| + \|\Delta b\|}.$$

We set

$$\underline{e} = -\frac{\|\Delta b\|}{\|\Delta A\| \|\hat{x}\| + \|\Delta b\|} r, \underline{E} = \frac{\|\Delta A\| \|\hat{x}\|}{\|\Delta A\| \|\hat{x}\| + \|\Delta b\|} r z^t, z = \frac{\hat{x}}{\|\hat{x}\|^2};$$

then one has

$$\begin{aligned} \underline{E}\hat{x} - \underline{e} &= r, \\ \|\underline{E}\| &= \frac{\|r\|}{\|\Delta A\| \|\hat{x}\| + \|\Delta b\|} \|\Delta A\|, \|\underline{e}\| = \frac{\|r\|}{\|\Delta A\| \|\hat{x}\| + \|\Delta b\|} \|\Delta b\|. \end{aligned}$$

■

The following theorem gives a bound for the forward error.

Theorem 15 *Given A regular, b , ΔA , Δb , E and b , such that $\|E\| \leq c\|\Delta A\|$ and $\|e\| \leq c\|\Delta b\|$, assuming that x is the solution of (P) and that \hat{x} is the solution of (W), one has*

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \frac{c}{1 - c\|A^{-1}\| \|\Delta A\|} \left(\frac{\|A^{-1}\| \|\Delta b\|}{\|x\|} + \|A^{-1}\| \|\Delta A\| \right), \quad (29)$$

provided $c\|A^{-1}\| \|\Delta A\| < 1$. Furthermore, this bound is attainable to first order in c .

Proof. One has

$$A(\hat{x} - x) = e - Ex + E(x - \hat{x}),$$

and then

$$\|\hat{x} - x\| \leq c\|A^{-1}\| \|\Delta b\| + c\|A^{-1}\| \|\Delta A\| \|x\| + c\|A^{-1}\| \|\Delta A\| \|x - \hat{x}\|,$$

which implies (29). If we set

$$\underline{e} = c\|\Delta b\| w, \underline{E} = -c\|\Delta A\| \|x\| w \frac{\hat{x}^t}{\|\hat{x}\|^2}, \|w\| = 1,$$

we get from

$$\hat{x} - x = A^{-1}(\underline{e} - \underline{E}\hat{x}),$$

the equation:

$$\hat{x} - x = A^{-1}(c\|\Delta b\| w + c\|\Delta A\| \|x\| w).$$

Now, since $\|w\| = 1$, one has $\|A^{-1}w\| = \|A^{-1}\|$, which yields

$$\frac{\|x - \hat{x}\|}{\|x\|} = c \left(\frac{\|A^{-1}\| \|\Delta b\|}{\|x\|} + \|A^{-1}\| \|\Delta A\| \right).$$

■

In (29), the quantity

$$\varkappa_{\Delta A, \Delta b}(A, x) = \frac{\|A^{-1}\| \|\Delta b\|}{\|x\|} + \|A^{-1}\| \|\Delta A\| \quad (30)$$

is the **norm condition number**.

If we consider the particular case where

$$\Delta A = A, \Delta b = b, \quad (31)$$

one sees that, since $b=Ax$:

$$\varkappa_{\Delta A, \Delta b}(A, x) \leq 2\|A^{-1}\| \|A\| \equiv 2\varkappa(A). \quad (32)$$

The fundamental result for the forward error using the specific perturbation (31) is given in the following corollary.

Corollary 16 Assume that the perturbation is given by (31) and that $\eta_{A,b}\varkappa(A) < 1$; then one has:

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \frac{2\eta_{A,b}\varkappa(A)}{1 - \eta_{A,b}\varkappa(A)} = 2\varkappa(A) \frac{\|r\|}{\|A\| \|\hat{x}\| + \|b\|} + O((\eta_{A,b}\varkappa(A))^2). \quad (33)$$

Remark 17 In Lapack ([1]), the norm is the infinite one and the estimate for the forward error is given by

$$\frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} \leq 2\varkappa_\infty(A) \frac{\|r\|_\infty}{\|A\|_\infty \|\hat{x}\|_\infty + \|b\|_\infty}, \quad (34)$$

with

$$\varkappa_\infty(A) = \|A^{-1}\|_\infty \|A\|_\infty. \quad (35)$$

In many practical cases, the matrix has a particular (sparse) structure and the normwise computation of the error can provide an overestimate. It is then better to look for a componentwise error estimate. The perturbation is done with a matrix ΔA and a vector Δb which **we assume to have non-negative entries**. We denote by $|x|$ and $|A|$ the vector and the matrix whose components are the absolute value of the original vector and matrix. The problem consists now in finding:

$$\omega_{\Delta A, \Delta b} = \min \{c : (A + E)\hat{x} = b + e, |E| \leq c\Delta A, |e| \leq c\Delta b\}. \quad (36)$$

Theorem 18 Given ΔA and Δb , the minimal componentwise backward error is given by:

$$\omega_{\Delta A, \Delta b} = \max_i \frac{|r_i|}{(\Delta A |\hat{x}| + \Delta b)_i}, \quad (37)$$

where $\xi/0$ is interpreted as 0 if $\xi = 0$ and ∞ otherwise.

Proof. Since

$$|r_i| = |(E\hat{x} - e)_i| \leq c(\Delta A |\hat{x}|)_i + c\Delta b_i,$$

one has

$$\omega_{\Delta A, \Delta b} \geq \max_i \frac{|r_i|}{(\Delta A |\hat{x}| + \Delta b)_i}.$$

We set

$$\underline{E} = D_1 \Delta A D_2, \underline{e} = -D_1 \Delta b,$$

with

$$D_1 = \text{diag} \left(\frac{r_i}{(\Delta A |\hat{x}| + \Delta b)_i} \right), D_2 = \text{diag} (\text{sgn}(\hat{x}_i)).$$

Then, we get:

$$(\underline{E}\hat{x} - \underline{e})_i = (D_1 (\Delta A D_2 + \Delta b))_i = \frac{r_i}{(\Delta A |\hat{x}| + \Delta b)_i} (\Delta A |\hat{x}| + \Delta b)_i;$$

this implies that $\underline{E}\hat{x} - \underline{e} = r$ and furthermore we have

$$|\underline{E}|_i = |\underline{e}|_i = \frac{|r_i|}{(\Delta A |\hat{x}| + \Delta b)_i}.$$

■

For the forward error, we have the following result.

Theorem 19 Given A regular, b , $\Delta A > 0$, $\Delta b > 0$, E and b , such that $|E| \leq c\Delta A$ and $|e| \leq c\Delta b$, assuming that x is the solution of (P), that \hat{x} is the solution of (W) and that $\|\cdot\|$ is an absolute norm⁴, one has

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq \frac{c}{1 - c \| |A^{-1}| \Delta A \|} \left(\frac{\| |A^{-1}| \Delta A |x| + |A^{-1}| \Delta b \|}{\|x\|} \right), \quad (38)$$

provided $c \| |A^{-1}| \Delta A \| < 1$. Furthermore, this bound is attainable in the ∞ -norm to first order in c .

⁴A norm is absolute iff $\| |x| \| = \|x\|$

Proof. Using

$$\widehat{x} - x = A^{-1}e - A^{-1}Ex + A^{-1}E(x - \widehat{x}),$$

we get

$$\|\widehat{x} - x\| \leq c \left\| |A^{-1}| \Delta b \right\| + c \left\| |A^{-1}| \Delta A |x| \right\| + c \left\| |A^{-1}| \Delta A \right\| \|x - \widehat{x}\|$$

and hence (38). By setting:

$$\underline{E} = -cD_1 \Delta A D_2 x \frac{\widehat{x}^t}{\|\widehat{x}\|^2}, \underline{e} = cD_1 \Delta b,$$

with

$$D_2 = \text{diag}(\text{sign}(\widehat{x}_i)), D_1 = \text{diag}(\xi_i), \xi_i = \text{sign}((A^{-1})_{ki}),$$

where k is such that

$$\left\| |A^{-1}| \Delta A |x| + |A^{-1}| \Delta b \right\|_\infty = \left(|A^{-1}| \Delta A |x| + |A^{-1}| \Delta b \right)_k,$$

we obtain

$$\begin{aligned} \|\widehat{x} - x\|_\infty &= \|A^{-1}(\underline{e} - \underline{E}\widehat{x})\|_\infty = \|A^{-1}(cD_1(\Delta b + \Delta A D_2 x))\|_\infty \\ &= c \left\| |A^{-1}| \Delta b + |A^{-1}| \Delta A |x| \right\|_\infty. \end{aligned}$$

■

In (38), the quantity

$$\text{cond}_{\Delta A, \Delta b}(A, x) = \frac{\left\| |A^{-1}| \Delta A |x| + |A^{-1}| \Delta b \right\|}{\|x\|} \quad (39)$$

is the **componentwise condition number**.

If we consider the particular case where

$$\Delta A = |A|, \Delta b = |b|, \quad (40)$$

one sees that:

$$\text{cond}_{|A|, |b|}(A, x) \leq 2 \frac{\left\| |A^{-1}| |A| |x| \right\|}{\|x\|} \equiv 2 \text{cond}(A, x). \quad (41)$$

Usually, one uses the infinite norm; then one has

$$\text{cond}_\infty(A, x) = \frac{\left\| |A^{-1}| |A| |x| \right\|_\infty}{\|x\|_\infty}. \quad (42)$$

Lemma 20 *The condition number for the matrix A alone can be defined by*

$$\text{cond}_\infty(A) = \max_{x \neq 0} \frac{\left\| |A^{-1}| |A| |x| \right\|_\infty}{\|x\|_\infty} = \left\| |A^{-1}| |A| \right\|_\infty. \quad (43)$$

Proof. By writing $x = \|x\| (x / \|x\|)$, one sees that

$$\text{cond}_\infty(A, x) \leq \max_{\|x\|_\infty=1} \left\| |A^{-1}| |A| |x| \right\|_\infty = \left\| |A^{-1}| |A| \right\|_\infty.$$

■

On the other hand, since

$$\left(|A^{-1}| |A| \right)_{ij} = \sum_k |a_{ik}^{-1}| |a_{kj}|,$$

it is easy to see that

$$\text{cond}_\infty(A) \leq \varkappa_\infty(A). \quad (44)$$

Finally, from (38) and (37) and using all the bounds for the condition number, we get for the forward componentwise error the following result.

Corollary 21 Assume that the perturbation is given by (40) and that $\omega_{A,b} \text{cond}(A, x) < 1$; then one has:

$$\frac{\|x - \hat{x}\|}{\|x\|} \leq 2 \text{cond}(A, x) \max_i \frac{|r_i|}{(|A| |\hat{x}| + |b|)_i} + O((\omega_{A,b} \text{cond}(A, x))^2). \quad (45)$$

In the infinite norm, one has

$$\begin{aligned} \frac{\|x - \hat{x}\|_\infty}{\|x\|_\infty} &\leq 2 \text{cond}_\infty(A) \max_i \frac{|r_i|}{(|A| |\hat{x}| + |b|)_i} + O((\omega_{A,b} \text{cond}_\infty(A))^2) \\ &\leq 2 \varkappa_\infty(A) \max_i \frac{|r_i|}{(|A| |\hat{x}| + |b|)_i} + O((\omega_{A,b} \varkappa_\infty(A))^2). \end{aligned} \quad (46)$$

It is worthwhile noticing that the estimate containing $\text{cond}(A, x)$ is not useful, since x is not known. Hence, we will look for the forward error $\|x - \hat{x}\| / \|\hat{x}\|$; we proceed like in the proof of (38). For perturbations of the form (40), since

$$\hat{x} - x = A^{-1}e - A^{-1}E\hat{x},$$

we get

$$\|\hat{x} - x\|_\infty \leq c \left(\|A^{-1}\| |b| + |A^{-1}| |A| \|\hat{x}\| \right)$$

and the forward error estimate is then given by:

$$\frac{\|x - \hat{x}\|}{\|\hat{x}\|} \leq \text{cond}_{|A|,|b|}(A, \hat{x}) \max_i \frac{|r_i|}{(\Delta A |\hat{x}| + \Delta b)_i}. \quad (47)$$

4 Perturbations on the computer

4.1 General methodology

The basic process for obtaining perturbation results on the computer requires three steps. First, one has to set up **the perturbation model**. Which of the input data do we want to perturb and how? The size of the perturbations are measured with some metric (e.g. a norm) relatively to the data size, which gives a perturbation parameter, say t . The range of this parameter has also to be determined. For each fixed value of the perturbation parameter, one does several random perturbations, yielding a random sample. Second, we have to choose what are the relevant output data for a specific behaviour, for instance conditioning, stability and so on. These are the so-called **indicators**, which are described in more details below. The last step consists in **analysing the indicators** with statistical tools (mean, standard deviation) and also to plot the results in function of t .

In general, the compute code is designed and then extended to include practical perturbation analysis. For realizing the three steps describe above we need to code the following tasks:

- Implementing the perturbation model: code in function of t the kind of perturbation and the resulting indicator(s). Then, usually in a double loop, produce several random perturbations for some discrete values of t in the perturbation interval.
- Analysis of the samples obtained after running the code: compute the indicators from the output and put them into some form suited for further processing. This can be done during execution or later on by saving the output to disk.
- Compute the statistical functions and visualize them.

The very last step is the user interpretation of the results, followed in general by another perturbation run allowing a deeper insight in the software and/or algorithm behavior.

Let us explain some indicators within the general framework of Section 2.2. We want to solve the exact problem $F(x) = y$, with solution x ; programming some algorithm on a computer yields an approximate solution $x_\theta = G_\theta(y)$, where the parameter θ includes the effect of the algorithm (e.g. discretization, stopping criterion) and of the implementation as well (rounding effects, ...). We denote by F_θ the implementation (numerical and computer) of F ; thus the residual will be computed with the

help of $F_\theta(x_\theta)$ and yield a backward error estimate. The scheme can be illustrated by the following diagram:

$$y \xrightarrow{G_\theta} x_\theta \xrightarrow{F_\theta} \zeta_\theta = F_\theta(x_\theta).$$

Now, we want to perturb some data, say z by an amount Δz , the size of which is measured in some specific norm (the data metric) by $t = \|\Delta z\|$; then we have

$$z + \Delta z \xrightarrow{G_\theta} x_\theta + \Delta x_\theta \xrightarrow{F_\theta} \zeta_\theta + \Delta \zeta_\theta = y + \Delta w_\theta.$$

From this diagram, we can define some indicators, like in Ref. [6].

Indicator \mathcal{I} : Reliability and Numerical Stability This indicator measures how G_θ is a good approximation of F^{-1} ; thus we want to estimate the relative error Δw_θ in function of the size of the perturbation. This is directly related to the numerical stability. We define

$$\mathcal{I}_{\theta t} = \sup_{\|\Delta z\|=t} \frac{\|\Delta w_\theta\|}{t}. \quad (48)$$

Notice that the values of Δw_θ depend of y , which can be troubleshooting. But, if we know a backward error estimate (i.e. $\|F - F_\theta\|$), we can use it for computing Δw_θ and avoid this problem. If the indicator $\mathcal{I}_{\theta t}$ has a bad behaviour, there is no hope to get a reliable solution of the problem. The subsequent indicators are used for backward stable problems.

Indicator \mathcal{K} : Stability and Conditioning This indicator is a measure of the sensitivity of F^{-1} at ζ_θ and is defined by

$$\mathcal{K}_{\theta t} = \sup_{\|\Delta z\|=t} \frac{\|\Delta x_\theta\|}{\|\Delta \zeta_\theta\|}. \quad (49)$$

In fact, this is the sensitivity of F_θ^{-1} ; because one cannot compute F exactly; accordingly, this should be computed at y , but $x = F^{-1}(y)$ is not known. This indicator gives an estimate of the condition number, which would yield $\Delta x_\theta \leq \varkappa \Delta w_\theta \approx \varkappa \Delta \zeta_\theta$.

Indicator \mathcal{L} : Algorithm sensitivity This is the sensitivity of the algorithm G_θ with respect to the input data; thus, it is defined by

$$\mathcal{L}_{\theta t} = \sup_{\|\Delta z\|=t} \frac{\|\Delta x_\theta\|}{t}. \quad (50)$$

This indicator is related to the preceding one, because if $\mathcal{K}_{\theta t}$ is very sensitive, so will be $\mathcal{L}_{\theta t}$ (presence of Δx_θ).

The indicator being defined, we want to describe in more details their use for linear system solving. This is done in the next section, using the notions and notations of the software PRECISE.

4.2 Using PRECISE for linear systems

4.2.1 Some notions from statistics

First of all, we have to set up the statistics context. Let X be a real continuous random variable, for instance all the possible perturbations of size t of a matrix. A random variable has a distribution function F_X , which in general is not known. The value $F_X(x)$ represents the probability that X is smaller or equal to x . One has:

$$F_X(x) = \int_{-\infty}^x f_X(t) dt, \quad (51)$$

where f_X is the density function, which gives the probability that X takes the value x . The **mean** or **expectation value** of the random variable is defined by

$$m(X) = E(X) = \int_{-\infty}^{\infty} x f_X(x) dx. \quad (52)$$

In the case of a discrete random variable, we would have

$$E(X) = \sum_k k p_X(k), \quad (53)$$

with p_X the so-called probability function. For example, for an uniform distribution of n discrete values, we would have $p_X(k) = 1/n$. Once the mean is defined, we can obtain **the variance**:

$$\sigma^2(X) = V(X) = E((X - E(X))^2), \quad (54)$$

and the standard deviation by:

$$\sigma(X) = D(X) = \sqrt{E(X)}. \quad (55)$$

In order to get information about some parameters of the distribution, e.g. the mean and the standard deviation, we make some measurements of the values of X , and obtain a data set $\mathbf{x} = (x_1, x_2, \dots, x_n)$. Then, we assume that these values correspond to observations on independent random variables $\mathbf{X} = (X_1, X_2, \dots, X_n)$, all with the same distribution F and say that we have a random sample from F . In particular, the mean m and standard deviation σ of each random variable is the same.

Now, we want to find estimators of the mean and variance of X from the random sample. We define

$$m_* = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i, \quad (56)$$

and consider m_* as a function of \mathbf{X} :

$$m_*(\mathbf{X}) = \bar{X} = \frac{1}{n} \sum_{i=1}^n X_i. \quad (57)$$

We quote without proof the following well-known theorem.

Theorem 22 *Given n random variables X_1, X_2, \dots, X_n , and real numbers c_1, c_2, \dots, c_n , one has*

$$E\left(\sum_{i=1}^n c_i X_i\right) = \sum_{i=1}^n c_i E(X_i);$$

if these variables are independent, then:

$$V\left(\sum_{i=1}^n c_i X_i\right) = \sum_{i=1}^n c_i^2 V(X_i).$$

It is then straightforward, with the help of this theorem to obtain:

$$E(m_*(\mathbf{X})) = m, \quad V(m_*(\mathbf{X})) = \frac{\sigma^2}{n}. \quad (58)$$

The first equation of (58) says that m_* is an unbiased estimator of the mean, since its expectation yields the correct value; the second equation means that for large n the variance is small.

The most natural idea would be to estimate the variance with

$$\sigma_*^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2;$$

unfortunately, this estimator is biased since $E(\sigma_*) = \frac{n-1}{n} \sigma^2$ (see below). The commonly used variance estimator is given by

$$s_n^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2. \quad (59)$$

Theorem 23 *The sample variance s_n^2 is an unbiased estimator of σ^2 .*

Proof. One computes:

$$\begin{aligned} \sum_{i=1}^n (X_i - \bar{X})^2 &= \sum_{i=1}^n (X_i - m - (\bar{X} - m))^2 \\ &= \sum_{i=1}^n (X_i - m)^2 - 2(\bar{X} - m) \sum_{i=1}^n (X_i - m) + n(\bar{X} - m)^2 \\ &= \sum_{i=1}^n (X_i - m)^2 - n(\bar{X} - m)^2. \end{aligned}$$

Hence, we have, with X any of the X_i :

$$\begin{aligned} E\left(\sum_{i=1}^n (X_i - \bar{X})^2\right) &= E\left(\sum_{i=1}^n (X_i - m)^2\right) - nE((\bar{X} - m)^2) \\ &= nE((X - m)^2) - nE((\bar{X} - m)^2) \\ &= nV(X) - nV(\bar{X}) = n\sigma^2 - \sigma^2. \end{aligned}$$

■

Concerning the practical computation of the sample variance (or standard deviation s_n) using the formula (59), it is worth noticing that we have to pass twice through the sample data: the first pass for computing the mean and the second one for the variance. Some statistics textbooks recommend to use the following one-pass formula:

$$s_n^2 = \frac{1}{n-1} \left(\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 \right).$$

But, for large values of n , this formula can suffer from cancellation and even yield a negative value of the variance!

4.2.2 Linear systems

We want to use practical perturbation for investigating methods for solving a linear system $Ax = b$, yielding an approximate solution $x_\theta = G_\theta(b)$; of course, here we have $F(x) = Ax$ and $\zeta_\theta = F_\theta(x_\theta) = A_\theta x_\theta$, with A_θ the representation of A in the computer. The perturbation z_t consists in perturbation of the matrix and/or of the right-hand-side. For each value of the perturbation size t , we generate a random sample of such perturbations. Thus, we have the following scheme:

$$\begin{array}{ccc} z + \Delta z = \begin{cases} A + E \\ b + e \end{cases} & \xrightarrow{G_\theta} x_\theta + \Delta x_\theta & \xrightarrow{F_\theta} \zeta_\theta + \Delta \zeta_\theta = y + \Delta w_\theta. \\ \downarrow & \downarrow & \downarrow \\ Z_t & X_t & Y_t \end{array}$$

We denote the mean and standard deviation for a sample, for instance X_t , by $m(X_t)$ and $\sigma(X_t)$. The reliability of an indicator will be measured by the standard deviation of the sample corresponding to the variables occurring in its definition. Thus, **for a normwise perturbation**, the indicator \mathcal{L} (50) will be computed as

$$\mathcal{L}_{\theta t} = \frac{\|\sigma(X_t)\|}{t \|x_\theta\|}, \quad (60)$$

with $\|\cdot\|$ some discrete norm. This term stems from a scaled norm, that is an estimate to the computed solution. **For a componentwise perturbation**, we set:

$$\mathcal{L}_{\theta t} = \frac{\|\sigma(X_t)\|_\infty}{t \|x_\theta\|_\infty}. \quad (61)$$

The indicator \mathcal{K} requires the quantities Δx_θ and $\Delta \zeta_\theta$; for Δx_θ , we proceed as before. For characterizing $\Delta \zeta_\theta$, we will make use of the theoretical backward estimates (28) or (37), since $\Delta \zeta_\theta$ is associated to the backward error at ζ_θ . Thus, **for normwise perturbations**, we set:

$$\mathcal{K}_{\theta t} = \frac{\|\sigma(X_t)\| (\|\Delta A\| \|x_\theta\| + \|\Delta b\|)}{\|x_\theta\| \|\sigma(Y_t)\|}, \quad (62)$$

and **for componentwise perturbations**:

$$\mathcal{K}_{\theta t} = \frac{\|\sigma(X_t)\|_\infty}{\|x_\theta\|_\infty} \frac{1}{\max_i \frac{|\sigma_i(Y_t)|}{(\Delta A |\hat{x}| + \Delta b)_i}}. \quad (63)$$

For the indicator \mathcal{I} , we need some estimate of Δw_θ , related to the backward error at y ; hence, instead of $\sigma(Y_t)$ as before, we have to use the biased standard deviation:

$$\hat{\sigma}_i(Y_t) = \sqrt{\sigma_i(Y_t)^2 + (m_i(Y_t) - b_i)^2}. \quad (64)$$

Therefore, the formula **for normwise perturbations** reads:

$$\mathcal{I}_{\theta t} = \sup_{\|\Delta z\|=t} \frac{\|\Delta w_\theta\|}{t} = \frac{\|\hat{\sigma}(Y_t)\|}{t (\|\Delta A\| \|x_\theta\| + \|\Delta b\|)},$$

and the one **for componentwise perturbations** is:

$$\mathcal{I}_{\theta t} = \frac{1}{t} \max_i \frac{|\hat{\sigma}_i(Y_t)|}{(\Delta A |\hat{x}| + \Delta b)_i}.$$

Once we have defined the indicators and implemented the perturbation scheme and the indicators on the computer (using PRECISE or only a part of it), we make runs to get results.

First, the interval for the perturbation size shall not only be close to the machine precision ϵ , but for beginning we can choose for instance $t \in [\epsilon, 1]$. Then, for specific perturbation model and metrics, one has to collect the indicator statistics and plot these indicators as a function of the perturbation size t . A typical behavior for the indicators \mathcal{I} and \mathcal{L} , for a stable algorithm, is depicted in Figure 3 (borrowed to Ref. [6]).

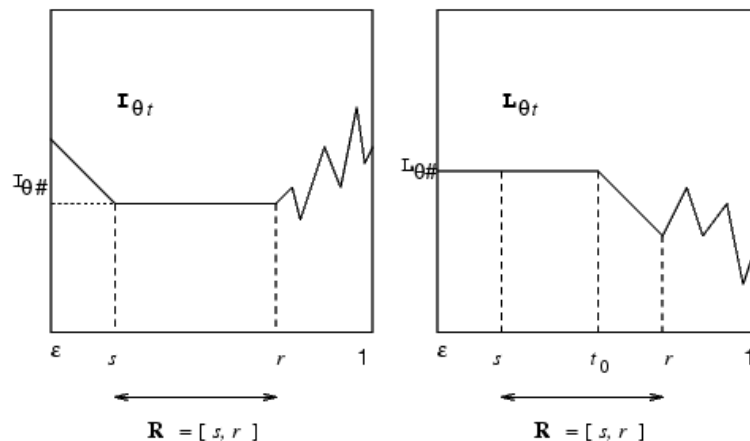


Figure 3: Typical behavior of indicators

The key feature is the existence of an interval $[s, r]$ on which the map $t \mapsto I_{\theta t}$ is **constant**. If this does not occur, this means that the method is unstable and unreliable; in this case further investigations using other indicators is meaningless. The value of s is the lower threshold of reliability for the estimation given by PRECISE for the three indicators $\mathcal{L}_{\theta t}$, $\mathcal{K}_{\theta t}$ and $\mathcal{I}_{\theta t}$.

The existence of this interval implies that the other indicators are valid on this interval. Furthermore, the value of s is of the order of the backward error at the computed solution x_θ for the particular perturbation model. This is because this value is the lowest value of the size t of the perturbations such that the approximate method G_θ , considered as a perturbation of F^{-1} , behaves in a way which is indistinguishable from the of F^{-1} at the level of uncertainty on the data defined by t . Of course, the condition of backward stability within finite machine precision ϵ is $s \leq \epsilon$.

We do not go further into the interpretation of the indicators and refer the reader to the more detailed description given in Refs. [6], [7].

References

- [1] E. Anderson et al., **LAPACK User's Guide**, SIAM Philadelphia (1992).
- [2] F. Chaitin-Chatelin, *Le calcul qualitatif*, CERFACS REPORT TR/PA/95/10 (1995).
- [3] Bo Einarsson (Ed.), **Accuracy and Reliability in Scientific Computing Handbook of Mathematical Fluid Dynamics**, SIAM Philadelphia (2005).
- [4] N. J. Higham, **Accuracy and Reliability of Numerical Algorithms**, SIAM Philadelphia (2002).
- [5] R. B. Kearfott, interval Computations: Introduction, Uses and Resources, *Euromath Bulletin* **2** (1996), 95-112.
- [6] R. A. McCoy and V. Toumazou, *PRECISE User's Guide Version 1.0*, CERFACS REPORT TR/PA/97/38 (1997).
- [7] F. Chaitin-Chatelin and V. Frayssé, **Lectures on Finite Precision Computations**, SIAM Philadelphia (1996).