OPEN SOURCE ENGINEERING

openeering

A SCILAB PROFESSIONAL PARTNER

powered by ENGINSOFT

NUMERICAL ANALYSIS USING SCILAB: ERROR ANALYSIS AND PROPAGATION

This tutorial provides a collection of numerical examples and advises on error analysis and its propagation. All examples are developed in Scilab.

Level 

www.openeering.com

Step 1: The purpose of this tutorial

The purpose of this Scilab tutorial is to provide a collection of numerical examples that are typically part of numerical analysis courses.

This tutorial is intended to help readers familiarize with numerical methods and their implementations, keeping under control the error propagation.

Here we provide some classical examples on error analysis and propagation.

An introduction to

NUMERICAL ANALYSIS USING SCILAB

(Error study in Numerical Analysis)

Step 2: Roadmap

In this tutorial, after a brief introduction to the concept of numerical errors, we analyze some examples in which numerical errors occur. These examples are useful to understand how important is to pay attention to error propagation when implementing numerical methods.

Step 3: Numerical error

What is allocated in a computer is only a subset of real number (using IEEE 754 standard) and hence, a real number x is represented in the computer as $fl(x)$, a fixed strings of digits. This limited amount of storage introduces a ***rounding error*** (or round-off error). It is possible to prove the following relation

$$fl(x) = x(1 + c)$$

with

$$|c| \leq Eps$$

where Eps is a constant that depends on the floating point system used to represent the number (for double precision arithmetic Eps is equal to $Eps = 2^{-52} = 2.22 \times 10^{-16}$).

```
// Scilab command to obtain eps
// ----------------------------

%eps
```

Step 4: Truncation error

The next step is to study what happens when the basic arithmetic operations are used.

We denote the computer arithmetic as $\oplus$, $\ominus$, $\otimes$, $\oslash$ to distinguish them from real arithmetic $+$, $-$, $\times$ and $\div$ respectively.

A part from the numerical error due to limited length in bits of registers, the computer arithmetic introduces a further truncation error when results are transfer from CPU to the memory. This error is caused by the fact that the length in bits of the CPU register is greater than the one used to store numbers.

As an example, if we sum two numbers x and y (to compute $x + y$) the corresponding machine operation $x \oplus y$ can be written as

$$x \oplus y = fl\big(fl(x) + fl(y)\big) = (fl(x) + fl(y))(1 + Eps_1)$$

with $|Eps_1| \leq Eps$. The sum of two numbers involves the sum of the two floating point operands. Since this operation is done in a register of the computer, it is necessary to consider a further floating point representation when the number is transferred to be stored into the memory.

Step 5: Cancellation error

It is interesting to analyze the arithmetic operations when we consider relative errors. For example, we want to analyze the sum operation:

$$\varepsilon_r^{\oplus}(x,y) = \frac{|(x+y)-(x \oplus y)|}{|x+y|}$$

in terms of the relative error of the input operands x and y. The relative error for x is defined as

$$\varepsilon_r(x) = \frac{|x - fl(x)|}{|x|}$$

A computer operation $\odot$ is considered ***numerically stable*** when

$$\varepsilon_r^{\odot}(x,y) \leq c(\varepsilon_r(x) + \varepsilon_r(y))$$

where $c > 0$ and x and y are two input operands.

It is possible to prove that the computer operations $\otimes$ and $\oslash$ are the only stable operations with respect to the rounding error. Conversely, for addition we have:

$$\varepsilon_r^{\oplus}(x,y) \leq Eps + (1+Eps)\left(\frac{|x|}{|x+y|}\varepsilon_r(x) + \frac{|y|}{|x+y|}\varepsilon_r(y)\right)$$

A similar inequality holds for subtraction.

When $y \simeq -x$, the right hand side of the inequality may be very high and hence may generate a high relative error (***cancellation error***).

Cancellation error should be avoided whenever possible!

Example of cancellation error for subtraction operation

In this example we consider a decimal arithmetic with floating point where a number a is expressed in the following way:

$$a = \pm(0.a_1a_2 \ldots a_m) \times 10^E$$

with $a_1 \neq 0$. E is the exponent (or characteristic) of the representation while $a_1a_2 \ldots a_m$ is the mantissa of the representation. The mantissa has always a fixed number of digits, in this case *m*.

For example, we can compute $(1 + \mathrm{x}) - 1$ using a system with a mantissa of 4 digits for the number, while we use a mantissa of 8 digits for the internal representations in the CPU register.

The example is done using $\mathrm{x} = 3.14 \times 10^{-2}$.

From memory to registers:

Memory (mantissa = 4)		**Register (mantissa = 8)**
$\mathrm{fl}(1) = 0.1000 \times 10^{+1}$	→	$0.1000\ 0000 \times 10^{+1}$
$\mathrm{fl}(\mathrm{x}) = 0.3140 \times 10^{-1}$	→	$0.3140\ 0000 \times 10^{-1}$

Arithmetic operation with alignment of the exponent:

Register (mantissa = 8)		**Register with aligned exp.**
$0.1000\ 0000 \times 10^{+1}$	→	$0.1000\ 0000 \times 10^{+1}$
$0.3140\ 0000 \times 10^{-1}$	→	$0.0031\ 4000 \times 10^{+1}$
	+	$1.0314\ 0000 \times 10^{0}$
$0.1031\ 4000 \times 10^{-1}$	←	$1.0314\ 0000 \times 10^{0}$

From register to memory:

Register (mantissa = 8)		**Memory (mantissa = 4)**
$0.1031\ 4000 \times 10^{-1}$	→	$\mathrm{fl}(1+\mathrm{x}) = 0.1031 \times 10^{+1}$

It is easy to compute the final step in which we subtract again 1 to evaluate $fl((1+\mathrm{x})-1)$. As a result we obtain 0.310×10^{-1} that is not equal to x.

Step 6: Two equivalent expressions

In this example we compare the following two equivalent expressions:

$$f_1(x) = \frac{(1+x)-1}{x}$$

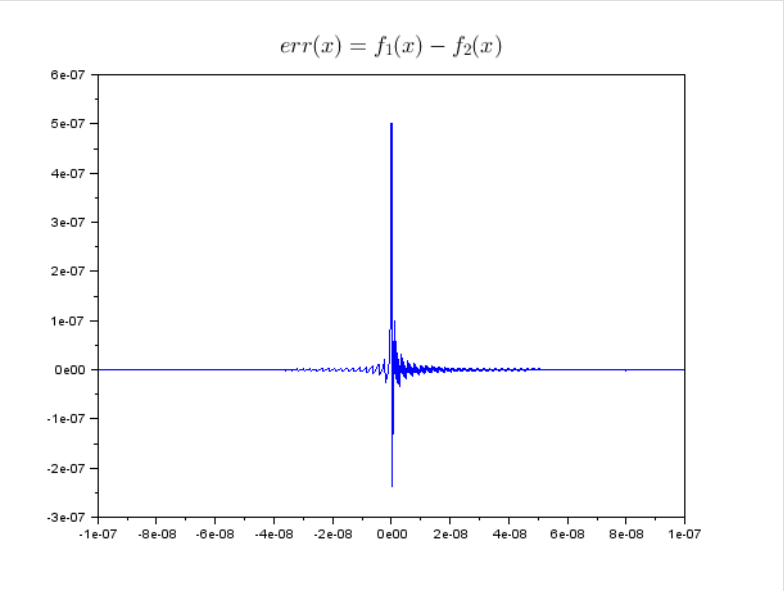
$$f_2(x) = 1$$

These two expressions are equivalent from an algebraic point of view; the second expression is simply obtained from the first one through simplification.

The first function suffers of the cancellation error when x goes to zero. This phenomenon has been explained in the previous step.

In the reported figure we have plotted the error $err = f_1(x) - f_2(x)$. We may see that the cancellation error is huge when x is close to zero. This phenomenon produces a loss of significant digits.

The Scilab script for the figure on the right is available in ***ex01.sce***. The code can be downloaded from the Openeering website.



(Error err in the range [-1e-07;1e-07])

Step 7: Two equivalent polynomials

In this example, we compare the two following polynomial expressions:

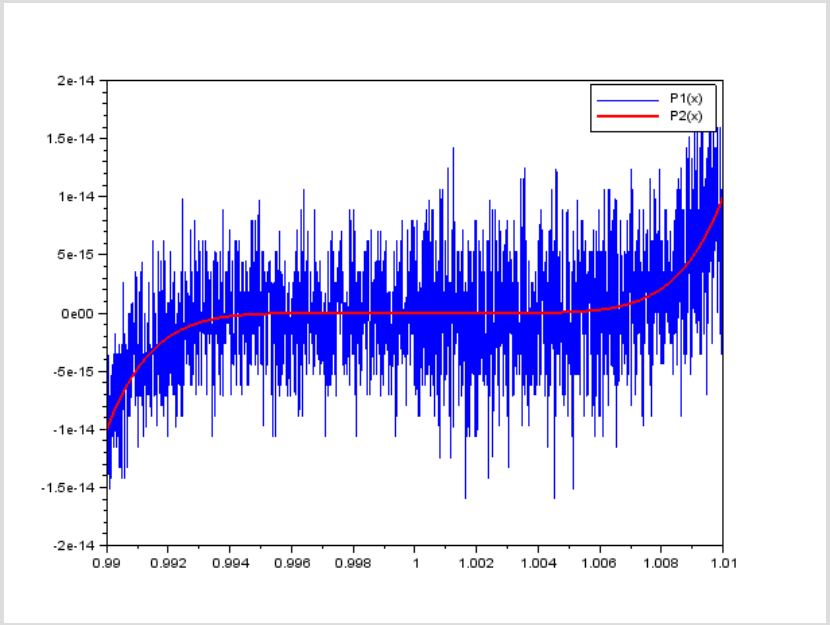
$$P_1(x) = x^7 - 7x^6 + 21x^5 - 35x^4 + 35x^3 - 21x^2 + 7x - 1$$

$$P_2(x) = (x-1)^7$$

where the first expression is obtained from the second by expansion.

As depicted on the right, the second expression is numerically stable while the first one is more unstable.

The Scilab script is reported in the function ***ex02.sce***.



(Comparison between two algebraic equivalent polynomials)

Step 8: Derivative example

In this step, we study the computation of a derivative from a numerical point of view. To approximate the first derivate, we use here the difference quotient (a.k.a. Newton's quotient):

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

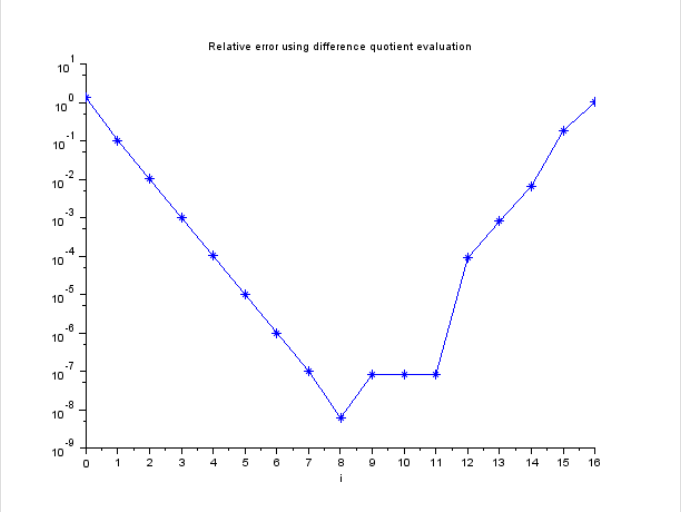
For example, we may analyze the formula for the function $f(x) = x^3 + 1$ at point $x = 1$ varying h in the following way:

$$h = 10^0, 10^{-1}, 10^{-2}, \ldots, 10^{-16}$$

The plot on the right visualizes the results.

From a mathematical point of view, when h goes to zero, the formula should assume the value of the first derivative. Intuitively, we may expect the smaller the h, the better the approximation. Unfortunately, because of cancellation errors, it happens that the most reliable value for the derivate is reached in $h = 10^{-8}$ (for $h = 10^{-16}$, the value of the derivate is the same as $h = 10$).

The Scilab script for plot and table is reported in the function ***ex03.sce***.



(Value of the derivative changing the value of h)

i	h	Der. exact	Der. approx	Abs. error.	Rel. error.
0	1.0e+000	3.000000e+000	7.0000000000e+000	4.00000e+000	1.33333e+000
1	1.0e-001	3.000000e+000	3.3100000000e+000	3.10000e-001	1.03333e-001
2	1.0e-002	3.000000e+000	3.0301000000e+000	3.01000e-002	1.00333e-002
3	1.0e-003	3.000000e+000	3.0030010000e+000	3.00100e-003	1.00033e-003
4	1.0e-004	3.000000e+000	3.0003000100e+000	3.00010e-004	1.00003e-004
5	1.0e-005	3.000000e+000	3.0000300001e+000	3.00001e-005	1.00000e-005
6	1.0e-006	3.000000e+000	3.0000029998e+000	2.99980e-006	9.99933e-007
7	1.0e-007	3.000000e+000	3.0000003015e+000	3.01512e-007	1.00504e-007
8	1.0e-008	3.000000e+000	2.9999999818e+000	1.82324e-008	6.07747e-009
9	1.0e-009	3.000000e+000	3.0000002482e+000	2.48221e-007	8.27404e-008
10	1.0e-010	3.000000e+000	3.0000002482e+000	2.48221e-007	8.27404e-008
11	1.0e-011	3.000000e+000	3.0000002482e+000	2.48221e-007	8.27404e-008
12	1.0e-012	3.000000e+000	3.0002667017e+000	2.66702e-004	8.89006e-005
13	1.0e-013	3.000000e+000	2.9976021665e+000	2.39783e-003	7.99278e-004
14	1.0e-014	3.000000e+000	3.0198066270e+000	1.98066e-002	6.60221e-003
15	1.0e-015	3.000000e+000	3.5527136788e+000	5.52714e-001	1.84238e-001
16	1.0e-016	3.000000e+000	0.0000000000e+000	3.00000e+000	1.00000e+000

(Derivative example – table of results)

Step 9: Error analysis and optimal step

The Newton's quotient to approximate the derivative $f'(x)$ contains two sources of error:

- Round-off error;
- Truncation error.

This follows from the Taylor series expansion of $f(x+h)$:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\zeta)$$

where $\zeta \in [x, x+h]$ and hence:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{\mathrm{h}}{2}f''(\zeta)$$

The first term contributes to round-off error, while the second term gives rise to a truncation error.

As reported on the right the total error can be expressed as

$$\begin{aligned} error_{total} &= error_{round-off} + error_{trunc} \\ &= \frac{2\,\bar{\varepsilon}}{h}M_1 + \frac{h}{2}M_2 \end{aligned}$$

From the above formula it is possible to compute the optimal step size h. Taking the derivative of the total error with respect to h and setting this quantity equal to zero, we have:

$$h^* = 2\sqrt{\frac{\bar{\varepsilon}\,M_1}{M_2}}$$

This h^* represents the optimal value for h to minimize the total error.

Evaluation of the round-off error:

The round-off error for the Newton's quotient can be written as:

$$\begin{aligned} error_{round-off} &= \left|\frac{f(x+h) - f(x)}{h} - \frac{\mathrm{fl}\big(f(x+h)\big) - \mathrm{fl}\big(f(x)\big)}{h}\right| \\ &= \left|-\frac{f(x+h)\,\varepsilon_1 - f(x)\,\varepsilon_2}{h}\right| \end{aligned}$$

where $|\varepsilon_i| \leq Eps$ (remember that $fl(x) = x(1+c)$ where $|c| \leq Eps$). Moreover we ignore error from division by h.

In order to evaluate this error, with h small, we replace $f(x+h)$ by $f(x)$.

We do not know the signs of ε_1 and ε_2 , so we replace the difference $\varepsilon_1 - \varepsilon_2$ with $2\bar{\varepsilon}$. This leads to:

$$error_{round-off} = \frac{2\,\bar{\varepsilon}}{h}|f(x)| = \frac{2\,\bar{\varepsilon}}{h}M_1$$

where $M_1 = \max_{x\in[x,x+h]}|f(x)|$.

Evaluation of the truncation error:

The truncation error is given by:

$$error_{trunc} = \frac{h}{2}|f''(x)| = \frac{h}{2}M_2$$

where for small values of h we replace ζ by h and $M_2 = \max_{x\in[x,x+h]}|f''(x)|$.

Step 10: Exercise

It is possible to compute the second derivatives of a function using the following approximation

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Apply the above formula to compute the second derivative of the function $f(x) = \cos(x)$ at a given point $x = \frac{1}{2}$.

Moreover, prove the following estimate for the total error:

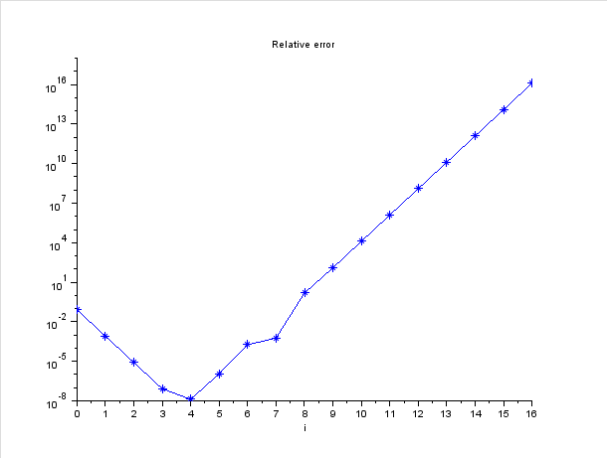
$$\left| f''(x) - \frac{fl\big(f(x+h)\big) - fl\big(2f(x)\big) + fl\big(f(x-h)\big)}{h^2} \right| \leq \frac{3\,\bar{\varepsilon}}{h^2} M_1 + \frac{h^2}{12} M_2$$

where $M_1 = \max_{x\in[x-h,x+h]} |f(x)|$ and $M_2 = \max_{x\in[x-h,x+h]} \left| f^{(iv)}(x) \right|$.

The upper bound error is minimized when h has the value

$$h^* = \sqrt[4]{\frac{36\,\bar{\varepsilon}\,M_1}{M_2}}$$

Hints: Using Taylor expansion for the function $f(x+h)$ and $f(x-h)$ to the fourth order paying attention to the signs.



(Relative error for second derivative approximation)

i	h	Der. exact	Der. approx	Abs. error.	Rel. error.
0	1.0e+000	-8.775826e-001	-8.0684536022e-001	7.07372e-002	8.06046e-002
1	1.0e-001	-8.775826e-001	-8.7685148682e-001	7.31075e-004	8.33056e-004
2	1.0e-002	-8.775826e-001	-8.7757524873e-001	7.31316e-006	8.33330e-006
3	1.0e-003	-8.775826e-001	-8.7758248890e-001	7.29897e-008	8.31713e-008
4	1.0e-004	-8.775826e-001	-8.7758257328e-001	1.13873e-008	1.29757e-008
5	1.0e-005	-8.775826e-001	-8.7758356138e-001	9.99486e-007	1.13891e-006
6	1.0e-006	-8.775826e-001	-8.7774232327e-001	1.59761e-004	1.82047e-004
7	1.0e-007	-8.775826e-001	-8.7707618945e-001	5.06372e-004	5.77008e-004
8	1.0e-008	-8.775826e-001	-2.2204460493e+000	1.34286e+000	1.53018e+000
9	1.0e-009	-8.775826e-001	-1.1102230246e+002	1.10145e+002	1.25509e+002
10	1.0e-010	-8.775826e-001	-1.1102230246e+004	1.11014e+004	1.26499e+004
11	1.0e-011	-8.775826e-001	-1.1102230246e+006	1.11022e+006	1.26509e+006
12	1.0e-012	-8.775826e-001	-1.1102230246e+008	1.11022e+008	1.26509e+008
13	1.0e-013	-8.775826e-001	-1.1102230246e+010	1.11022e+010	1.26509e+010
14	1.0e-014	-8.775826e-001	-1.1102230246e+012	1.11022e+012	1.26509e+012
15	1.0e-015	-8.775826e-001	-1.1102230246e+014	1.11022e+014	1.26509e+014
16	1.0e-016	-8.775826e-001	-1.1102230246e+016	1.11022e+016	1.26509e+016

(Second derivative approximation, table of results)

Step 11: Concluding remarks and References

In this tutorial we have collected a series of numerical examples written in Scilab for the study of numerical errors.

1. Scilab Web Page: Available: www.scilab.org.
2. Openeering: www.openeering.com.
3. J. Higham, accuracy and Stability of Numerical Algorithms, SIAM
4. Atkinson, An Introduction to Numerical Analysis, Wiley

Step 12: Software content

To report a bug or suggest improvements please contact Openeering team at the web site www.openeering.com.

Thank you for your attention,

Silvia Poles and Manolo Venturin

```
--------------
Main directory
--------------
ex01.sce            : Expression example
ex02.sce            : Polynomial example
ex03.sce            : Derivative example
ex04.sce            : Second derivative exercise
license.txt         : The license file
```