TANA09 Lecture 2

- Number representation in computers. Floating point systems.
- Computer arithmetic and computational errors.
- Analysis of computational errors. Catastrophic cancellation. Accumulated errors.
- Non-linear equations. Fixed point iteration.

Number representation in computers

Real numbers can be written in *exponent form* or as a *floating point number*.

As an example

$$763.45 = 7.6345 \cdot 10^2.$$

A *floating point number* is *normalized* if there is exactly one digit before the decimal point. The above number has the *integer part* 7 and the *fractional part* 0.6345.

This means

$$(7 \cdot 10^0 + 6 \cdot 10^{-1} + 3 \cdot 10^{-2} + 4 \cdot 10^{-3} + 5 \cdot 10^{-4})10^2.$$

We have a *position system* with the base 10.

What does a computers floating point system look like?

Definition A *floating point system* is characterized by by the parameters (β, t, L, U), where β is the *base*, t is the number of digits in the fractional part, and L and U are the smallest and largest exponents respectively.

Exemple The number system $(10, 3, -9, 9)$ contains the numbers

$$4.562, \quad 123.7, \quad \text{och } 0.006532.$$

The number 0 can not be written as a normalized floating point number.

IEEE 754 Single Precision $(2, 23, -126, 127)$

In the computer a number is stored as one *word* (32 bits). The bits are assigned as follows

s (1 bit)	e (8 bits)	f (23 bits)

In the *normal case*, $1 \leq e \leq 254$, the bit pattern is interpreted as

$$x = (-1)^s (1.f)_2 \cdot 2^{e-127}.$$

The cases $e = 0$ or $e = 255$ lets us define $x = 0$, $x = \pm\infty$, and $x = \text{NaN}$.

Exemple How is the number 13.25 stored in the computer?

Observation If we store $x = 0.1$ in the floating point system $(2, 23, -126, 127)$ we obtain

$$x = (0.1)_{10} = (0.0001100110011\ldots)_2 = (1.1001100110011\ldots)\cdot 2^{-4}$$

With 23 bits in the fractional part $x = 0.1$ is not stored exactly on the computer. We get a round off error $|x - xr| \leq 2^{-27} = 7.45 \cdot 10^{-9}$. Is this important?

> **Remark** A number that can be stored exactly in the 10-base number system can not nessecarily be stored exactly in the binary number system.

The errors are small but computers can do *many* computations *fast*.

Example Let $h = 0.1$ and increment a variable x until it reaches 1. In Matlab

```
x=0;h=1/10;
while x<1, x=x+h;, end
```

The code stops with $x = 1.1000$. To make sure the code stops use

```
while abs(x-1)<h/2, x=x+h; , end
```

or a `for`-loop.

> **Remark** Do not test equality between real numbers!

Exempel Patriot missiles had an internal clock that measured the time in tenths of seconds stored as an integer. Before the variable was used in computations it was multiplied by 0.1 and made into a single precision floating point number.

The longer the system was in use the larger the error became. A too large error in the internal clock means the system can't solve the equations needed to hit the target.

During testing the system was turned on and managed to hit incomming missiles but the first time it was to be used the system was on for several days before the attack and failed with horrid results.

This, and other, exemples are presented in the book *Accuracy and Reliability in Scientific Computing*, SIAM, 2005. By Bo Einarsson.

Computer arithmetic and computational errors

Exemple If we store a real number in a floating point system we make a *round-off error*. Store $x = 573.672$ in the system $(10, 3, -9, 9)$. What is the resulting error?

Exemple Suppose we want to add $x = 34.23$ and $y = 85.28$. What is the best possible error bound if we do the calculations in the floating point system $(10, 3, -9, 9)$?

Round-off errors in floating point systems

Theorem If a number x is stored in the floating point system (β, t, L, U) the *relative error* is bounded by

$$\frac{|x - x_r|}{|x|} \leq \frac{1}{2}\beta^{-t},$$

where x_r is the closest number to x in the number system.

Definition The constant $\mu = \frac{1}{2}\beta^{-t}$ is called the *unit round-off* or *machine precision*.

Remark In Matlab the function `eps` returns 2μ.

Arithmetic operations in floating point systems

Suppose we do computations in the number system (β, t, L, U). Then

Theorem If an arithmetic operation $x \odot y$ is carried out then

$$\frac{|x \odot y - \text{fl}[x \odot y]|}{|x \odot y|} \leq \mu$$

where $\text{fl}[x \odot y]$ is the result computed within the number system, μ is the unit round-off, and $\odot$ is either $+$, $-$, $*$, or $/$.

Interpretation First do the computation *exactly* and round the result to fit into the floating point system.

If is possible to implement standard functions $\exp(x)$, $\log(x)$, $\sqrt{x}$,..., so that the they can be computed with a relative error at most μ.

Exemple Compute $a + b + c$ in the number system $(10, 3, -9, 9)$ when $a = 9.876 \cdot 10^4$, $b = -9.880 \cdot 10^4$, and $c = 3.456 \cdot 10^1$.

We have two alternatives

$$\begin{aligned} \text{fl}[\text{fl}[a+b]+c] &= \text{fl}[\text{fl}[-0.004 \cdot 10^4] + 3.456 \cdot 10^0] \\ &= \text{fl}[-4.000 \cdot 10^1 + 3.456 \cdot 10^1] = -5.440 \cdot 10^0. \end{aligned}$$

or

$$\text{fl}[a + \text{fl}[b+c]] = \text{fl}[9.876 \cdot 10^4 - 9.877 \cdot 10^4] = -1.000 \cdot 10^1.$$

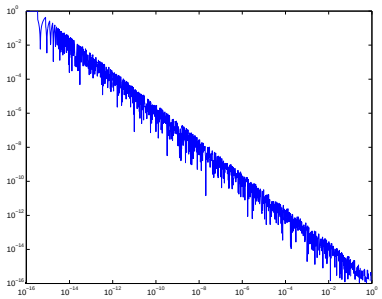
Perform an error analysis and determine the best option

Assumption All computations are carried out with a relative error equal to the unit round-off $\mu = 0.5 \cdot 10^{-3}$.

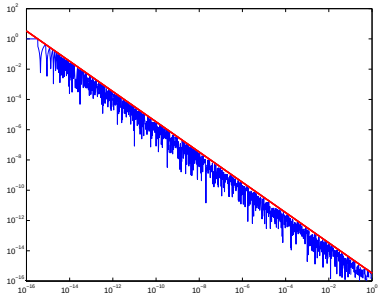
Analysis of computational errors

Compute $f(x) = \sqrt{1+x} - 1$ for small x. In Matlab

```
>> x=10.^-(0:0.01:16); f=sqrt(1+x)-1;
>> loglog(x,abs(f-f_ex)./f_ex);
```



Perform an error analysis that explains the result.



The relative error is

$$\frac{|\mathrm{fl}[f(x)] - f(x)|}{|f(x)|}, \qquad f(x) = \sqrt{1+x} - 1.$$

and the error bound $$\frac{|\Delta f|}{|f|} \leq \frac{3\mu}{x}.$$

Cancellation cause the relative error to *grow* as x decreases! How can we fix the problem?

Example We want to compute the infinite sum

$$S = \sum_{k=1}^{\infty} \frac{1}{k \log(1+k)^6}.$$

In Matlab

```
k=1;S1=1;S0=0;
while S1>S0,
    S0=S1;k=k+1;S1=S1+1/k/log(1+k)^6;
end
```

The result is $S = 1.3614$. The loop stops at $k = 1.88 \cdot 10^8$ with $t_k = (k \log(1+k)^6)^{-1} \approx 1.1 \cdot 10^{-16}$

Question Can we trust the result? Convergence of infinite sums cannot be tested numerically. We need to know in advance that the series is supposed to converge.

Accumulated errors

Example Let $x = (x_1, x_2, \ldots, x_n)^T$ be a vector and compute the sum

$$S = \sum_{i=1}^{n} x_i.$$

Lemma Let μ be the unit round-off. The error in the computed sum is bounded by

$$|S - \bar{S}| \lesssim \mu \sum_{k=1}^{n} (n-k+1)|x_k|.$$

Remark A more detailed analysis in the book gives $\leq$ with an extra constant 1.06 added.

Remark Compute the sum with the smallest terms first!

Floating point arithmetic and error analysis

Important things to remember are

- Can assume all computations are carried out with a relative error of at most μ.
- The order of computation is important. Thus

$$\mathrm{fl}[a + (b + c)] \neq \mathrm{fl}[(a + b) + c].$$

- Matematically equivalent expressions can give very different results. Rewrite

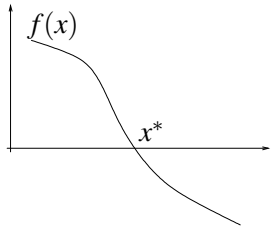
$$\sqrt{1+x} - 1 = \frac{x}{\sqrt{1+x}+1},$$

in order to avoid the cancellation.

- If no misstakes are made then usually errors due to the floating point system are negligible.

Non-linear equations

Problem Solve a non-linear equation $f(x) = 0$.

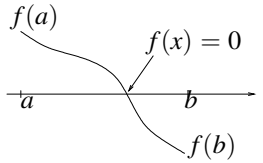


Questions Existance and Uniqueness? Good numerical methods? Error estimate?

We assume we can compute $f(x)$ and possibly $f'(x)$.

Existance

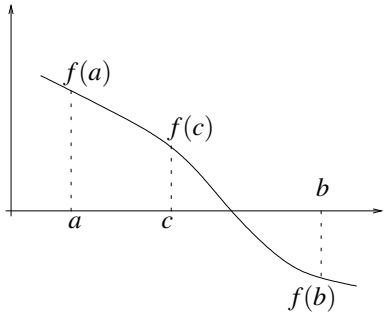
Theorem If $f(x)$ is continuous on $[a,b]$ and $c \in [f(a), f(b)]$ then there exists an $x \in [a,b]$ such that $f(x) = c$.



Lemma Suppose $f(x)$ is continuous. If $f(a)f(b) < 0$ then there is a root x^* to the equation $f(x) = 0$ in (a,b).

This is a good criteria for *existance* of a root to $f(x) = 0$.

The bisection method



There is a *root* in (a,b) if $f(a)f(b) < 0$.

In each step we compute $c = \frac{1}{2}(a+b)$ and $f(c)$. Continue with either (a,c) or (c,b).

Exemple Solve the equation $f(x) = x - \mathrm{e}^{-x} = 0$ using the bisection method. We know that $0.55 < x^* < 0.6$.

k	a_k	b_k	c_k	$f(c_k)$
0	0.55	0.60	0.575	$1.23 \cdot 10^{-2}$
1	0.55	0.575	0.5625	$-7.28 \cdot 10^{-3}$
2	0.5625	0.575	0.56875	$2.5 \cdot 10^{-3}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
20	0.567143250	0.56714344	0.567143345	$8.5 \cdot 10^{-8}$

Approximate $x^* = c_{20} \pm \frac{a_{20}+b_{20}}{2} = 0.567143345 \pm 9.5 \cdot 10^{-8}$.

The method always converges! The convergence is slow!

The number of iterations does not depend on the function $f(x)$. Only on the desired accuracy.

Equation solving and iterative methods

Definition An *iteratative method* constructs a sequence $\{x_k\}_{k=1}^{\infty}$, where x_0 is the initial guess.

Definition An iteratative method is *convergent* if

$$x_k \to x^*, \quad \text{as } k \to \infty,$$

where x^* is a *root* to the equation $f(x) = 0$.

Questions How to construct good iterative methods? How to prove convergence?

Fixed point iteration

Definition A *fixed point iteration* can be written on the form

$$x_{k+1} = \varphi(x_k), \qquad k = 0, 1, 2, \ldots$$

A *fixed point* x^* to the function $\varphi(x)$ satisfies $x^* = \varphi(x^*)$.

Preferably a fixed point x^* should be a root to the equation $f(x) = 0$.

Exemple We want to solve $f(x) = x - \mathrm{e}^{-x} = 0$ and test the methods

(i) $x_{k+1} = \mathrm{e}^{-x_k} = \varphi_1(x_k)$ $\quad$ (ii) $x_{k+1} = -\log(x_k) = \varphi_2(x_k)$.

What happens?

(i) Method: $x_{k+1} = \mathrm{e}^{-x_k}$

k	x_k
1	0.57694981
2	0.56160877
3	0.57029086
4	0.56536097
5	0.56815502
10	0.56708395
20	0.56714309
30	0.56714329

We obtain $x^* \approx 0.56714329$.

(ii) Method: $x_{k+1} = -\log(x_k)$

k	x_k
1	0.59783700
2	0.51443714
3	0.66468192
4	0.40844668
5	0.89539391
6	0.11049153
7	2.20281638
8	-0.78973671

The sequence is divergent!

How to investigate if a method is convergent? The convergence speed?

Theorem Suppose $\varphi(x)$ has a en real fix point x^* and that $|\varphi'(x)| \leq m < 1$ in a neighbourhood of x^*. Then, if x_0 is sufficiently close to x^*, we have

$$\lim_{k \to \infty} x_k = x^*.$$

Lemma For a convergent fixed point iteration $x_{k+1} = \varphi(x_k)$ we have that

$$|x_{k+1} - x^*| \leq m|x_k - x^*|$$

This is called *linear convergence*. Better with a small m.

Exemple Solve a system of equations

$$f(x) = \begin{pmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ (x_1 - 0.5)^2 + x_2^2 - 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

Try to use the fixed point iteration

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}), \quad \text{with } x_0 = (0.4, 1.2)^T.$$

k	$x_1^{(k)}$	$x_2^{(k)}$	$\|x^{(k)} - x^*\|$
0	0.4000	1.2000	$2.76 \cdot 10^{-1}$
1	-0.2000	0.7500	$5.00 \cdot 10^{-1}$
5	0.2661	0.8528	$1.17 \cdot 10^{-1}$
10	0.2034	0.9652	$4.67 \cdot 10^{-2}$
15	0.2581	0.9686	$8.10 \cdot 10^{-3}$
20	0.2486	0.9682	$1.43 \cdot 10^{-3}$

Often easy to find a fixed point iteration that converges.