Basic concepts of numerical analysis

Mariano Marcano
Department of Computer Science
University of Puerto Rico
Río Piedras Campus
Outline

- Problem well-posedness and stability.
- Algorithm stability.
- Condition number.
- Algorithm convergence.
- Source of error and accuracy.
- The floating-point number system and rounding.
Consider the following problem: find $x$ such that 

$$f(p, x) = 0,$$

where $p$ is the set of data or parameters, which the solution depends on, and $f$ is the functional relation between $x$ and $p$. Depending on the kind of problem that is represented, $x$ and $p$ may be real numbers, vectors, or functions.
Basic concepts of numerical analysis

- **direct problem**: $f$ and $p$ are given and $x$ is unknown.
- **inverse problem**: $f$ and $x$ are known and $p$ is unknown.
- A problem is **well posed** if it admits a unique solution $x$ which depends continuously on the data.
- For the moment **stable** and well posed will have the same meaning.
- If a problem is not well posed, then it is called **ill posed** or **unstable**.
Basic concepts of numerical analysis

- Let $P$ be the set of admissible parameters, i.e., the set of values of $p$ for which the problem has a unique solution.
- Continuous dependence on the parameters means that small perturbation on the parameters $p \in P$ yields “small” changes in the solution $x$. 
Stability. For $p \in P$, let’s denote by $\delta p$ a perturbation admissible in the sense that $p + \delta p \in P$ and by $\delta x$ the corresponding change in the solution, in such a way that

$$f(p + \delta p, x + \delta x) = 0.$$ 

Then, we require that there are $\eta_0 = \eta_0(p) > 0$ and $\kappa_0 = \kappa_0(p)$ such that

$$\text{if } \|\delta p\| \leq \eta_0 \text{ then } \|\delta x\| \leq \kappa_0\|\delta p\|.$$
The *relative condition number* for the problem is defined as

$$\text{cond}(p) = \sup \left\{ \frac{\| \delta x \| / \| x \|}{\| \delta p \| / \| p \|} : \delta p \neq 0, \ p + \delta p \in P \right\}.$$ 

In case $p = 0$ or $x = 0$, we introduce the *absolute condition number* for the problem

$$\text{cond}_{\text{abs}}(p) = \sup \left\{ \frac{\| \delta x \|}{\| \delta p \|} : \delta p \neq 0, \ p + \delta p \in P \right\}.$$
The problem is called *ill conditioned* if \( \text{cond}(p) \) is “big” for any admissible parameter \( p \).

The meaning of “small” or “big” depends on the considered problem.

The property of a problem being well conditioned is independent of the numerical method used to solve the problem.
If the problem has a unique solution, then there is a mapping $G$ (resolvent) between the set of parameters and of the solutions such that

$$x = G(p) \quad \text{that is} \quad f(p, G(p)) = 0.$$ 

Using this in the definition of stability, one gets

$$x + \delta x = G(p + \delta p).$$
Assuming that $G$ is differentiable in $p$ and denoting by $G'$ its derivative with respect to $p$, a Taylor’s expansion of $G$ up to the first derivative term is

$$G(p + \delta p) - G(p) = G'(p)\delta p + o(\|\delta p\|),$$

where $h = o(\|\delta p\|)$ are terms that go to zero faster than $\|\delta p\|$, i.e.,

$$\lim_{\|\delta p\| \to 0} \frac{h}{\|\delta p\|} = 0.$$
By dropping $o(\|\delta p\|)$, one gets

$$\text{cond}(p) \approx \|G'(p)\| \frac{\|p\|}{\|G(p)\|} \quad \text{and} \quad \text{cond}_{\text{abs}} \approx \|G'(p)\|.$$
Example

Let $A$ be a nonsingular real $n \times n$ matrix, $b \in \mathbb{R}^n$ and $x \in \mathbb{R}^n$, for the linear system

$$Ax = b$$

$x$ is the unknown solution and $b$ and $A$ are the parameters.

Considering perturbations in the right-hand side $b$ only, we have that $p = b$, $x = G(b) = A^{-1}b$, and $G'(b) = A^{-1}$, thus the condition number is

$$\text{cond}(b) \approx \frac{\|A^{-1}\| \|b\|}{\|A^{-1}b\|} = \frac{\|Ax\|}{\|x\|} \|A^{-1}\| \leq \|A\| \|A^{-1}\| = \text{cond}(A),$$

where $\text{cond}(A)$ is the condition number of matrix $A$. 
Example

This means that if the matrix $A$ is well conditioned, solving the linear system $Ax = b$ is a stable problem with respect to perturbations of the right-hand side $b$. 
Let us assume that the problem is well posed.

A numerical method for the approximate solution of the problem consists of a sequence of approximate problems

\[ f_n(p_n, x_n) = 0, \quad n \geq 1. \]

We expect that the numerical solution converges to the exact solution, i.e.,

\[ x_n \to x \quad \text{as} \quad n \to \infty. \]

For that, it is necessary that \( p_n \to p \) and that \( f_n \) "approximates" \( f \), as \( n \to \infty \).
Stability of numerical methods

A numerical method is consistent with the problem if the parameter $p$ of the problem is admissible for $f_n$ and

$$f_n(p, x) = f_n(p, x) - f(p, x) \to 0 \quad \text{as} \quad n \to \infty,$$

where $x$ is the solution to the problem corresponding to the parameter $p$. 
Stability. Let $P_n$ be the set of admissible parameters. For $p_n \in P_n$, denote by $\delta p_n$ a perturbation admissible in the sense that $p_n + \delta p_n \in P_n$ and by $\delta x_n$ the corresponding change in the solution, in such a way that

$$f(p_n + \delta p_n, x_n + \delta x_n) = 0.$$ 

Then, we require that there is an $\eta_0 = \eta_0(p_n) > 0$ and $\kappa_0 = \kappa_0(p_n)$ such that

$$\text{if } \|\delta p_n\| \leq \eta_0 \text{ then } \|\delta x_n\| \leq \kappa_0\|\delta p_n\|.$$
Let $x$ be the exact solution of a problem and $x_n$ be the numerical approximation.

- **Absolute error**

  \[ E(x_n) = \| x - x_n \| . \]

- **Relative error**

  \[ E_{\text{rel}}(x_n) = \frac{\| x - x_n \|}{\| x \|}, \quad \text{whenever} \quad x \neq 0. \]
A numerical method is *convergent* if and only if for all $\epsilon > 0$ exist $n_0 = n_0(\epsilon)$ and $\delta = \delta(n_0, \epsilon) > 0$ such that for all $n > n_0(\epsilon)$, and for all $\delta p_n$ for which $\|\delta p_n\| \leq \delta$ implies that

$$\|x(p) - x_n(p + \delta p_n)\| \leq \epsilon,$$

where $p$ is an admissible parameter for the problem, $x(p)$ is the corresponding solution, and $x_n(p + \delta p_n)$ is the numerical solution with parameter $p + \delta p_n$. 
Convergence of numerical methods

*Lax-Richtmyer theorem:* for a consistent numerical method, stability is equivalent to convergence.
Sources of error:

- Method truncation errors;
- Numerical representation errors.
- And rounding errors.
A numerical method must be convergent, accurate, reliable, and efficient. A numerical method is:

- convergent if the computational error can be made arbitrarily small by increasing the computational effort;
- accurate if the truncation and rounding errors are small with respect to a fixed tolerance;
- reliable if the global error can be made smaller than certain tolerance;
- and efficient if the computational complexity that is needed to control the error is as small as possible. The complexity of an algorithm is a measure of its executing time.
The positional system

Let $x \in \mathbb{R}$ with a finite number of digits $x_k$ where $0 \leq x_k \leq \beta$ for $\beta \in \mathbb{N}$ (a base) and $k = -m, \ldots, n$. The positional representation of $x$ with respect to the base $\beta$ is given by

$$x_\beta = (-1)^s(x_nx_{n-1}\ldots x_1x_0.x_{-1}x_{-2}\ldots x_{-m}), \quad x_n \neq 0,$$

where $s = 0, 1$ depends on the sign of $x$. The point between $x_0$ and $x_{-1}$ is called decimal point if the base is 10 and binary point if the base is 2. This expression can be written as

$$x_\beta = (-1)^s \left( \sum_{k=-m}^{n} x_k \beta^k \right).$$
Examples

- The number \( x_{10} = 425.33 \) denotes the number

\[
4 \cdot 10^2 + 2 \cdot 10^1 + 5 \cdot 10^0 + 3 \cdot 10^{-1} + 3 \cdot 10^{-2}.
\]

while \( x_6 = 425.33 \) denotes the real number

\[
4 \cdot 6^2 + 2 \cdot 6^1 + 5 \cdot 6^0 + 3 \cdot 6^{-1} + 3 \cdot 6^{-2}.
\]

- A rational number can have a finite number of digits in a base and an infinite number of digits in another base. For example, \( 1/3 \), in base 10 is \( x_{10} = 0.\overline{3} \), whereas in base 3 it is \( x_3 = 0.1 \).
Any real number can be approximated by numbers having a finite representation.

For a fixed base $\beta$ and for any $\epsilon > 0$ and for any $x_\beta \in \mathbb{R}$, there exists $y_\beta \in \mathbb{R}$ such that

$$|y_\beta - x_\beta| < \epsilon,$$

where $y_\beta$ has finite positional representation.
The positional system

- Note that for a positive number

\[ x_\beta = x_n x_{n-1} \ldots x_1 x_0 \cdot x_{-1} x_{-2} \ldots x_{-m} \ldots, \quad x_n \neq 0, \]

with finite or infinite number of digits, for any \( r \geq 1 \) one can build two numbers with \( r \) digits.

- The number

\[ x_\beta^{(l)} = \sum_{k=0}^{r-1} x_{n-k} \beta^{n-k} \]

is smaller than \( x_\beta \).
The positional system

- Adding $\beta^{n-r+1}$ to $x^{(l)}_\beta$ (i.e., the number with $\beta$ in the digit in the position $r-1$ and zero in the other digits)

$$x^{(u)}_\beta = x^{(l)}_\beta + \beta^{n-r+1},$$

gives a number larger than $x_\beta$.

- This gives the following relation

$$x^{(l)}_\beta < x_\beta < x^{(u)}_\beta.$$
The positional system

- Note that
  \[ x^{(u)}_\beta - x^{(l)}_\beta = \beta^{n-r+1}. \]

- By choosing \( r \) such that \( \beta^{n-r+1} < \epsilon \) and taking \( y_\beta \) equal to \( x^{(l)}_\beta \) or \( x^{(u)}_\beta \), one gets the desired inequality.

- This result legitimates the computer representation of real numbers.
The floating-point number system

The *floating-point* representation for a nonzero real number is as follows:

\[ x_\beta = (-1)^s(1.a_1a_2\ldots a_t)\beta^e, \]

where

- \( t \in \mathbb{N} \) is the number of allowed significant digits \( a_i \) (with \( 0 \leq a_i \leq \beta - 1 \)),
- \( m = a_1a_2\ldots a_t \in \mathbb{N} \) is the mantissa,
- and \( e \in \mathbb{N} \) is the exponent, which we let \( L \leq e \leq U \).
The floating-point number system

In the floating-point representation the $N$ memory positions are distributed among

- the sign (one position),
- the significant digits ($t$ positions),
- and the exponent ($N - t - 1$ positions).
The floating-point number system

Two formats that are available in the computer for the floating-point number representation:

- **single precision**, in the case of binary representation, $N = 32$ bits: 1 bit for the sign; 8 bits for the exponent, and 23 bits for the mantissa.

- **double precision**, $N = 64$ bits: 1 bit for the sign; 11 bits for the exponent, and 52 bits for the mantissa.
The floating-point number system

The floating point representation implemented in the computer: The representation follows the order sign \( (s) \), exponent \( (e) \), and mantissa \( (m) \)

\[
s \, e_1 \, e_2 \cdots \, e_p \, m_1 \, m_2 \cdots \, m_t,
\]

where \( N = p + t + 1 \) is the numbers of bits in the memory assigned to store a number in the decided precision.
The floating-point number system

- In $p$ bits we can store $2^p$ binary digits from positions 0 to $2^p - 1$.
- The position 0 and $2^p - 1$ are reserved to represent NaN (Not a Number) and $\infty$, respectively.
- We will have $2^p - 2$ positions. Because we need to represent negative and positive exponents, we need to add an exponent bias to the exponent, such that we can relocate the exponents from $-2^{p-1} + 2$ to $2^{p-1} - 1$ into the positions from 1 to $2^p - 2$. Then the bias is $2^{p-1} - 1$. 

Mariano Marcano  
Numerical analysis
The floating-point number system

In double precision $N = 64$, $m = 52$, and $p = 11$. Thus the exponent bias is $2^{10} - 1 = 1023$.
Consider for example the representation in the computer of $(12)_{10} = (1100)_2$, the sign is 0, the mantissa is 1000 · · · 0 and the exponent is 3. We must add the bias to 3 before finding its binary representation, $1023 + 3 = 1026$. And the binary representation of the exponent is 10000000010. Thus the floating point of 12 is stored in the computers as follows

$$0 \ 10000000010 \ 1000 \cdots 0$$
The floating-point representation of 1 in double precision is

0 01111111111 0000000000000000... 0

This is because the exponent is 0 and we must add the bias.
The floating-point number system

- The *machine epsilon* $\epsilon_M$ is the distance between the number 1 and the nearest floating-point number.
- In other words, $\epsilon_M$ is the smallest positive floating-point number such that $1 + \epsilon_M > 1$. 
The floating-point number system

- The binary double precision representation of one is
  \[ 1.0\ldots0 \times 2^0, \]
  where the mantissa has 52 zeros.
- The largest and nearest double precision number is
  \[ 1.0\ldots1 \times 2^0, \]
- This number is
  \[ 1 + 2^{-52}. \]
- Thus, the machine epsilon in double precision is
  \[ \epsilon_M = 2^{-52} \approx 2.2204 \times 10^{-16}. \]
A standard for the base, number of significant digits, and range of the exponent was developed in 1985 by the Institute of Electrical and Electronics Engineers (IEEE) and was approved in 1989 by the International Electronical Commission (IEC) as the international standard IEC559. For double precision the standard is

- base = 2;
- mantissa = 52;
- \( N = 64; \)
- \( L = -1022; \)
- \( U = 1023; \)
The smallest positive floating-point number is

$$x_{\text{min}} = (1.00 \cdots 0)_2 2^{-1022} \approx 2.2 \times 10^{-308}$$

and the largest is

$$x_{\text{max}} = (1.11 \cdots 1)_2 \cdot 2^{1023} = (2 - 2^{-52}) \cdot 2^{1023} \approx 1.8 \times 10^{308}.$$
For a real number $x$ the rounded floating point is defined as

$$\text{fl}(x) = (-1)^s(1.a_1 a_2 \cdots \tilde{a}_t)_{\beta} \cdot \beta^e,$$

where $\tilde{a}_t$ is chose following the procedure in the next slide.
For binary numbers, the IEEE rounding is as follows: let $y$ be the bit in the 52nd position to the right of the binary point:

- If the bit in position 53 is 0 then round down (truncate after the 52nd bit).
- If the bit in position 53 is 1 and there is a bit in some position to the right of 1 that is 1 then round up (add 1 to $y$).
- If the bit in position 53 is 1 and all known bits to the right of 1 are 0’s then:
  - If $y = 0$ truncate after the 52nd bit.
  - Otherwise add 1 to $y$. 
Example. Write the floating point representation of 9.4 using the IEEE rounding rule.

Since

\[(9.4)_{10} = (1001.0110)_{2}\]

The floating point representation is

\[+1.001011001100110011001100110011001100110011001101\times2^3,\]

after rounding

\[\text{fl}(9.4) = 1.001011001100110011001100110011001100110011001100110011001101\times2^3\]
Overflow and underflow

If after an arithmetic operation the result is such that

- $e > U$, then we speak about overflow;
- $e < -L$, then there was an underflow.
If \( x \in \mathbb{R} \) is such that \( x_{\text{min}} \leq |x| \leq x_{\text{max}} \), then

\[
fl(x) = x(1 + \delta)
\]

with

\[
|\delta| \leq u = \frac{1}{2} \epsilon_M,
\]

where \( u \) is the so-called roundoff unit or machine precision. As a consequence, the following bound holds for the relative error

\[
E_{\text{rel}} = \frac{|x - fl(x)|}{|x|} \leq u.
\]
Example. Compute the absolute error $|\text{fl}(9.4) - 9.4|$ and the relative error $|\text{fl}(9.4) - 9.4|/9.4$

- We eliminated
  
  $0.1100 \times 2^{-52} \times 2^3 = 0.8 \times 2^{-49}$

  (where $(0.1100)_2 = (12/15)_{10} = 0.8$.)

- and added
  
  $2^{-52} \times 2^3 = 2^{-49}$.

- Then
  
  $\text{fl}(9.4) = 9.4 - 0.8 \times 2^{-49} + 2^{-49}$
Rounding

The absolute error is

$$|\text{fl}(9.4) - 9.4| = (1 - 0.8) \times 2^{-49} = 0.2 \times 2^{-49}$$

The relative error is

$$\frac{|\text{fl}(9.4) - 9.4|}{9.4} = \frac{0.2 \times 2^{-49}}{9.4} = \frac{8}{47} \times 2^{-52} < \frac{1}{2^{\epsilon_M}}.$$
A flop is a single elementary floating-point operation.

An inner product between two vectors of length $n$ will require $2n - 1$ flops.

A matrix multiplication between two matrices of size $m \times r$ and $r \times n$ will require $(2r - 1)mn$ flops.
References

