Numerical Methods for Computational Science and Engineering

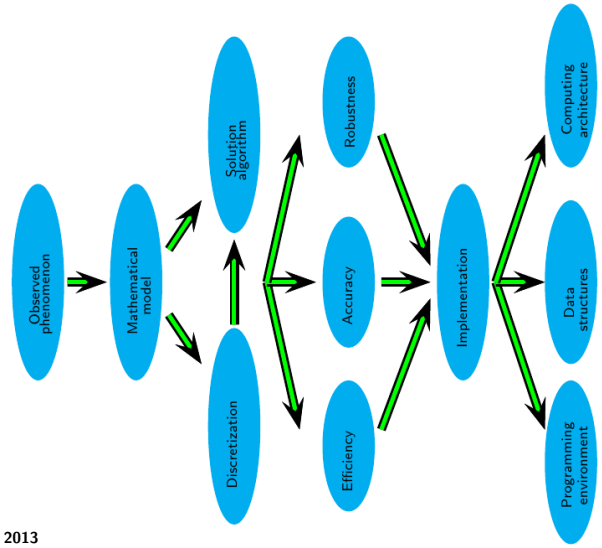
Lecture 1, Sept 19, 2013: Introduction

Peter Arbenz
Computer Science Department, ETH Zürich
E-mail: arbenz@inf.ethz.ch

Outline of today's lecture

- What is numerical methods for CSE
- Survey of the lecture
- Organization of the lecture (exercises/examination)
- References
- Start of the lecture

Scientific Computing



Survey on lecture

1. Introduction
2. Roundoff errors
3. Nonlinear equations in one variable (2 lectures)
4. Linear algebra review
5. Direct methods for linear system (2)
6. Linear least squares problems (2)
7. Iterative methods for linear system (2)
8. Eigenvalues and singular values (2)
9. Nonlinear systems and optimization (3)
10. (Piecewise) polynomial interpolation (3)
11. Best approximation

Survey on lecture (cont.)

12. Filtering algorithms, Fourier transform
13. Numerical differentiation
14. Numerical integration (2)
15. Ordinary differential equations, initial value problems (3)

About this course

Focus

- on algorithms (principles, scope, and limitations),
- on (efficient, stable) implementations in MATLAB,
- on numerical experiments (design and interpretation).

No emphasis on

- theory and proofs (unless essential for understanding of algorithms)
- hardware-related issues (e.g. parallelization, vectorization, memory access)

  (These aspects will be covered in the course "High Performance Computing for Science and Engineering" offered by D-INFK)

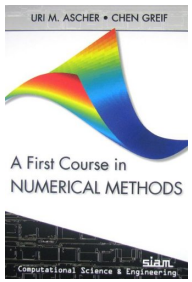
Goals

- Knowledge of the fundamental algorithms in numerical mathematics
- Knowledge of the essential terms in numerical mathematics and the techniques used for the analysis of numerical algorithms
- Ability to choose the appropriate numerical method for concrete problems
- Ability to interpret numerical results
- Ability to implement numerical algorithms efficiently in MATLAB

Indispensable: Learning by doing (➔ exercises)

Literature

Uri Ascher & Chen Greif: A First Course in Numerical Methods. SIAM, 2011.
http://www.siam.org/books/cs07/



- Excellent reference.
- Main reference for large parts of this course.
- Target audience: undergraduate students in computer science.
- I will follow this book quite closely.

Literature (cont.)

- W. Dahmen & A. Reusken: Numerik für Ingenieure und Naturwissenschaftler, Springer, 2006.

  Good reference for large parts of this course; a lot of simple examples and lucid explanations, but also rigorous mathematical treatment; Target audience: undergraduate students in science and engineering. Available through Nebis.

- H.-R. Schwarz & N. Köckler: Numerische Mathematik. Teubner, 2006. 6. Auflage.

  Main reference for large parts of this course; Target audience: undergraduate students in science and engineering. Available through Nebis.

- C. Moler: Numerical Computing with MATLAB. SIAM 2004.

  Good reference for some parts of this course; Target audience: MATLAB users and programmers. See `http://www.mathworks.ch/moler/`.

Prerequisites

Essential prerequisite for this course is a solid knowledge in linear algebra and calculus. Familiarity with the topics covered in the first semester courses is taken for granted, see

- K. NIPP AND D. STOFFER, *Lineare Algebra*, vdf Hochschulverlag, Zürich, 5 ed., 2002.
- M. GUTKNECHT, *Lineare algebra*, lecture notes, SAM, ETH Zürich, 2009. `http://www.sam.math.ethz.ch/~mhg/unt/LA/HS07/`.
- M. STRUWE, *Analysis für Informatiker*. Lecture notes, ETH Zürich, 2009.

Organization

Lecturer:

Prof. Peter Arbenz	arbenz@inf.ethz.ch

Assistants:

Stefan Pauli	stefan.pauli@inf.ethz.ch
Daniel Hupp	huppd@inf.ethz.ch
Christian Schüller	schuellc@inf.ethz.ch
Robert Carnecky	crobi@inf.ethz.ch
Laura Scarabosio	laura.scarabosio@sam.math.ethz.ch
Jonas Šukys	jonas.sukys@sam.math.ethz.ch
Cecilia Pagliantini	cecilia.pagliantini@sam.math.ethz.ch
Alexander Lobbe	alobbe@student.ethz.ch
Sebastian Grandis	grandiss@student.ethz.ch
Sharan Jagathrakashakan	sharanj@student.ethz.ch

Venue

Classes:	Mon 10.15-12.00 (CAB G11); Thu 10.15-12.00 (HG G5)
Tutorials:	Mon 13.15-15.00
	Thu 8.15-10.00

Please register (on course website) for tutorial groups until September 23th:
`http://www.math.ethz.ch/education/bachelor/lectures/hs2013/math/nummath_cse`

Consulting hours: if needed, see the course website.

Assignments

- The assignment sheets will be uploaded on the course webpage on Monday every week the latest.
- The exercise should be solved until the following tutorial class. (Hand them in to the assistant or grade yourself.)

Examination

- Three-hour written examination involving coding problems to be done at the computer on
  TBA
- Dry-run for computer based examination:
  Does not exist anymore.
  Try out a computer in the student labs in HG.
- Pre-exam question session:
  TBA

Examination (cont.)

- Topics of examination:
  - All topics, that have been addressed in class or in a homework assignment.
  - One exam question will be one of the homework assignment.
- Lecture slides will be available as (a single) PDF file during the examination.
- The Ascher-Greif book will be made available, too.
- The exam questions will be asked both in German and in English. You can chose among the two.

Problem solving environment: MATLAB

- We use MATLAB for the exercises.
- Although most of the algorithm we are dealing with have been implemented in MATLAB, it is useful when you program them yourselves.
- These (little) programs will be building blocks when you will solve more complex problems in your future.
- MATLAB help
  - MATLAB commands `help`/`doc`
  - MATLAB online documentation, e.g., `http://www.mathworks.nl/help/pdf_doc/allpdf.html`
- Numerous introductory textbooks / user guides / primers
- My own MATLAB introduction: `http://people.inf.ethz.ch/arbenz/MatlabKurs/matlabintro.pdf`

Numerical algorithms and errors

- The most fundamental feature of numerical computing is the inevitable presence of errors.
- The result of any interesting computation (and of many uninteresting ones) is typically only approximate, and our goal is to ensure that the resulting error is tolerably small.
- Example: How many loop iterations are there in this little MATLAB program?

```
x = 0; h = 1/10;
while x<1,
   x=x+h;
   % do something depending on x
end
```


How to measure errors

- Can measure errors as absolute or relative, or a combination of both.
- The absolute error in v approximating given scalar quantity u is $|u - v|$.
- The relative error (assuming $u \neq 0$) is $\frac{|u-v|}{|u|}$.

u	v	Absolute Error	Relative Error
1	0.99	0.01	0.01
1	1.01	0.01	0.01
−1.5	−1.2	0.3	0.2
100	99.99	0.01	0.0001
100	99	1	0.01

Approximation example

The Stirling approximation

$$v = S_n = \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$$

is used to approximate $u = n! = 1 \cdot 2 \cdots n$ for large n.

The formula involves the constant $e = \exp(1) = 2.7182818\ldots$.

We use the MATLAB program `Example1_1.m` from `http://www.siam.org/books/cs07/programs.zip` to compute the values of u and v for $n = 1, \ldots, 10$.

Notice that the quality of v approximating u increases although the absolute error increases rapidly!

Types of errors

- Errors in the formulation of the problem to be solved.
  - Errors in the mathematical model. Simplifications.
  - Error in input data. Measurements.
- Approximation errors
  - Discretization error.
  - Convergence error in iterative methods.
  - Discretization/convergence errors may be assessed by an analysis of the method used.
- Roundoff errors
  - Roundoff errors arise everywhere in numerical computation because of the finite precision arithmetic.
  - Roundoff errors behave quite erratic.

Discretization errors in action

Problem: want to approximate the derivative $f'(x_0)$ of a given smooth function $f(x)$ at the point $x = x_0$.

Example: Let $f(x) = \sin(x)$, $-\infty < x < \infty$, and set $x_0 = 1.2$. Thus, $f(x_0) = \sin(1.2) \approx 0.932\ldots$.

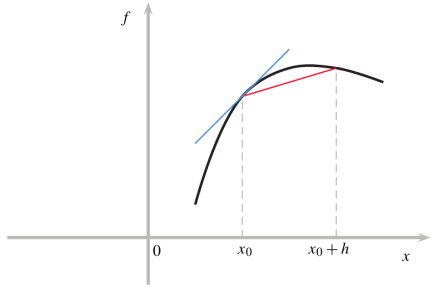
Discretization: Function values $f(x)$ are available only at a discrete number of points, e.g. at grid points $x_j = x_0 + jh$, $j \in \mathbb{Z}$.

Want to approximate $f'(x_0)$ by values $f(x_j)$.

Discretization errors in action (cont.)

Taylor's series gives us an algorithm to approximate $f'(x_0)$:

$$f'(x_0) \approx D_{x_0,h}(f) = \frac{f(x_0+h) - f(x_0)}{h}$$



Discretization errors in action (cont.)

Expanding $f(x)$ by a Taylor series around $x = x_0$ gives

$$\frac{f(x_0+h) - f(x_0)}{h} = f'(x_0) - \frac{h}{2} f''(\xi), \quad x_0 < \xi < x_0 + h.$$

So, we expect the error to decrease linearly with h.

$$\left| f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h} \right| = \frac{h}{2} \left| f''(\xi) \right| \approx \frac{h}{2} \left| f''(x_0) \right|$$

Or, using the big-O notation:

$$\left| f'(x_0) - D_{x_0,h}(f) \right| = \mathcal{O}(h).$$

Results

Try for $f(x) = \sin(x)$ at $x_0 = 1.2$.

(So, we are approximating $\cos(1.2) = 0.362357754476674\ldots$)

h	Absolute error
0.1	$4.71667 \cdot 10^{-2}$
0.01	$4.666196 \cdot 10^{-3}$
0.001	$4.660799 \cdot 10^{-4}$
10^{-4}	$4.660256 \cdot 10^{-5}$
10^{-7}	$4.619326 \cdot 10^{-8}$

These results reflect the discretization error as expected.

Note that $f''(x_0)/2 = -\sin(1.2)/2 \approx -0.466$.

Results for smaller h

The above results indicate that we can compute the derivative as accurate as we like, provided that we take h small enough.

If we wanted

$$\left|\cos(1.2) - \frac{\sin(1.2+h) - \sin(1.2)}{h}\right| < 10^{-10}.$$

We have to set $h \leq 10^{-10}/0.466$.

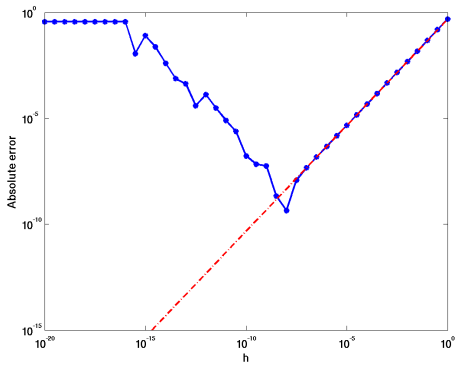
The following numbers and plot are generated by
`.../Greif/programs/chap01/Example1_3Figure1_3.m`

Results for smaller h

h	Absolute error
10^{-8}	$4.36105 \cdot 10^{-10}$
10^{-9}	$5.594726 \cdot 10^{-8}$
10^{-10}	$1.669696 \cdot 10^{-7}$
10^{-11}	$7.938531 \cdot 10^{-6}$
10^{-13}	$6.851746 \cdot 10^{-4}$
10^{-15}	$8.173146 \cdot 10^{-2}$
10^{-16}	$3.623578 \cdot 10^{-1}$

These results reflect both discretization and roundoff errors.

Results for all h



The solid curve interpolates the computed values of $|f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}|$ for $f(x) = \sin(x)$ at $x_0 = 1.2$.

The dash-dotted straight line depicts the discretization error without roundoff error.

Algorithm properties

Performance features that may be expected from a good numerical algorithm.

- Accuracy
  Relates to errors. How accurate is the result going to be when a numerical algorithm is run with some particular input data.
- Efficiency
  - How fast can we solve a certain problem?
    Rate of convergence. Floating point operations (flops).
  - How much memory space do we need?
  - These issues may affect each other.
- Robustness
  (Numerical) software should run under all circumstances. Should yield correct results to within an acceptable error or should fail gracefully if not successful.

Complexity I

Complexity/computational cost of an algorithm $:\Longleftrightarrow$ number of elementary operators
Asymptotic complexity $\hat{=}$ "leading order term" of complexity w.r.t. *large* problem size parameters
The usual choice of problem size parameters in numerical linear algebra is the number of independent real variables needed to describe the input data (vector length, matrix sizes).

operation	description	#mul/div	#add/sub	
inner product	$(\mathbf{x} \in \mathbb{R}^n, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}^H\mathbf{y}$	n	$n-1$	$\mathcal{O}(n)$
outer product tensor product	$(\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n) \mapsto \mathbf{x}\mathbf{y}^H$	nm	0	$\mathcal{O}(mn)$
matrix product	$(A \in \mathbb{R}^{m\times n}, B \in \mathbb{R}^{n\times k}) \mapsto AB$	mnk	$mk(n-1)$	$\mathcal{O}(mnk)$

Big-O and Θ notation

For an error e depending on h we denote

$$e = \mathcal{O}(h^q)$$

if there are two positive constants q and C such that

$$|e| \leq C\, h^q \qquad \forall h > 0 \text{ small enough.}$$

Similarly, for $w = w(n)$ the expression

$$w = \mathcal{O}(n \log n)$$

means that there is a constant $C > 0$ such that

$$|w| \leq Cn \log n \qquad \text{as } n \to \infty.$$

Big-O and Θ notation

More abstract:
Class $\mathcal{O}(f)$ of functions is defined as

$$\mathcal{O}(f) = \{g \mid \exists c_1, c_2 > 0 : \forall N \in \mathbb{Z}^+ : g(N) \leq c_1 f(N) + c_2\}$$

The Θ notation signifies a stronger relation than the $\mathcal{O}$ notation: a function $\phi(h)$ for small h (resp., $\phi(n)$ for large n) is $\Theta(\psi(h))$ (resp., $\Theta(\psi(n))$) if ϕ is asymptotically bounded both above and below by ψ.

Example:
$\mathcal{O}(h^2)$ means at least "quadratic convergence" (see later). $\Theta(h^2)$ is exact quadratic convergence.

Complexity II

To a certain extent, the asymptotic complexity allows to predict the dependence of the runtime *of a particular implementation* of an algorithm on the problem size (for large problems). For instance, an algorithm with asymptotic complexity $\mathcal{O}(n^2)$ is likely to take 4× as much time when the problem size is doubled.

One may argue that the memory accesses are more decisive for run times than floating point operations. In general there is a linear dependence among the two. So, there is no difference in the O notation.

Scaling

Scaling ≡ multiplication with diagonal matrices (with non-zero diagonal entries) from left and/or right.

It is important to know the different effects of multiplying with a diagonal matrix from left or right:

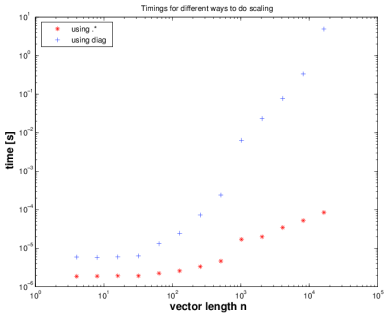
$$DA \quad \text{vs.} \quad AD \quad \text{with } \mathbb{R}^{n\times n} \ni A, D = \text{diag}(d_1, \dots, d_n)$$

Scaling with $D = \text{diag}(d_1, \dots, d_n)$ in MATLAB:

```
y = diag(d)*x;
```

or

```
y = d.*x;
```



Elementary matrices

Matrices of the form $A = I + \alpha \mathbf{u}\mathbf{v}^T$ are called elementary. Again we can apply A to a vector $\mathbf{x}$ in a straightforward and a more clever way:

$$A\mathbf{x} = (I + \alpha \mathbf{u}\mathbf{v}^T)\mathbf{x}$$

or

$$A\mathbf{x} = \mathbf{x} + \alpha \mathbf{u}(\mathbf{v}^T\mathbf{x})$$

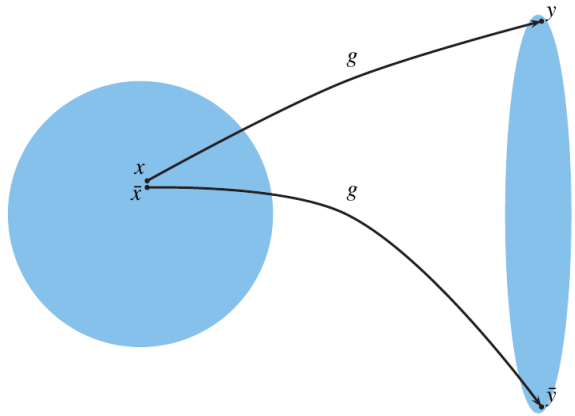
Cf. exercises.

Problem conditioning and algorithm stability

Qualitatively speaking:

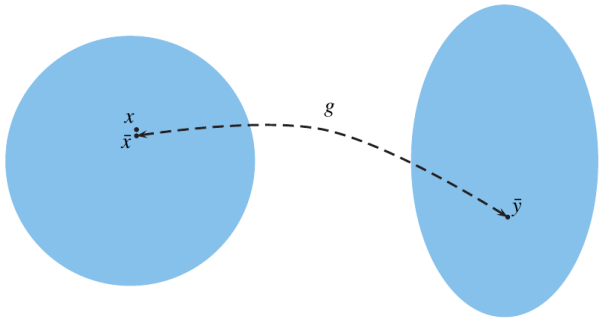
- The problem is ill-conditioned if a small perturbation in the data may produce a large difference in the result.
  The problem is well-conditioned otherwise.
- The algorithm is stable if its output is the exact result of a slightly perturbed input.

An unstable algorithm



Ill-conditioned problem of computing output values y from input values x by $y = g(x)$: when x is slightly perturbed to $\bar{x}$, the result $\bar{y} = g(\bar{x})$ is far from y.

A stable algorithm



An instance of a stable algorithm for computing $y = g(x)$: the output $\bar{y}$ is the exact result, $\bar{y} = g(\bar{x})$, for a slightly perturbed input, i.e., $\bar{x}$ which is close to the input x. Thus, if the algorithm is stable and the problem is well-conditioned, then the computed result $\bar{y}$ is close to the exact y.

Unstable algorithm

Problem statement: evaluate the integrals

$$y_n = \int_0^1 \frac{x^n}{x+10}\,dx, \quad \text{for } n = 0, 1, 2, \ldots, 30.$$

Algorithm development: observe that analytically, for $n > 0$,

$$y_n + 10y_{n-1} = \int_0^1 \frac{x^n + 10x^{n-1}}{x+10}\,dx = \int_0^1 x^{n-1}\,dx = \frac{1}{n}.$$

Also,

$$y_0 = \int_0^1 \frac{1}{x+10}\,dx = \log(11) - \log(10).$$

Algorithm:

- Evaluate $y_0 = \log(11) - \log(10)$.
- For $n = 1, 2, \ldots, 30$, evaluate $y_n = \frac{1}{n} - 10y_{n-1}$.

Unstable algorithm (cont.)

Run the MATLAB program `Example1_6.m` by Ascher and Greif to see the catastrophic amplification of roundoff errors.
This code is available from
`http://www.siam.org/books/cs07/programs.zip`.

Unstable algorithm (cont.)

Roundoff error accumulation

- In general, if E_n is error after n elementary operations, cannot avoid linear roundoff error accumulation

$$E_n \simeq c_0 n E_0.$$

- Will not tolerate an exponential error growth such as

$$E_n \simeq c_1^n E_0, \qquad \text{for some constant } c_1 > 1.$$

This is an **unstable algorithm**.