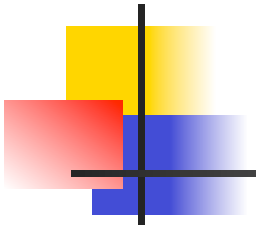
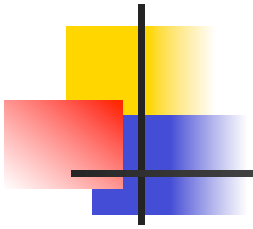


Computational Methods

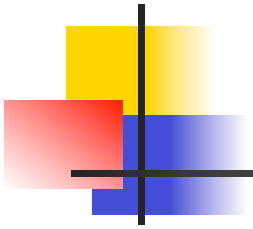
Sources of Errors



Numerical Analysis / Scientific Computing

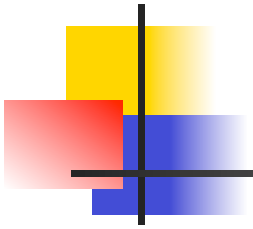
- Many problems in Science and Engineering can not be solved analytically on a computer
  - Numeric solutions are often required
    - Numeric solutions provide only approximate solutions
    - Numeric solutions are not unique
      - Different numeric algorithms might yield different approximations
- Numerical Analysis deals with the design and analysis of numeric algorithms
  - Deals with continuous quantities
  - Considers / analyzes the effects of approximations

Computational Problems and Numerical Algorithms

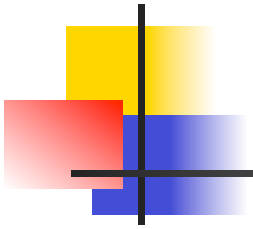


- Solving of computational problems usually involves the following steps:
  - Mathematical modeling
    - Develop a mathematical description for the problem
  - Algorithm design
    - Build an algorithm to solve the mathematical problem formulation
    - Analyze the algorithm for its performance
  - Implementation and Evaluation
    - Implement the algorithm
    - Evaluate its performance with real data

Computational Problems and Numerical Algorithms

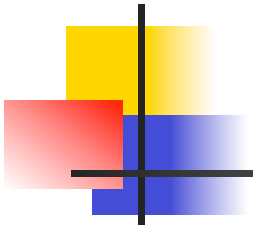


- A problem is *well-posed* if
  - A solution exists and is unique
  - The solution depends continuously on the data
- *Ill-posed* problems are often sensitive to the data and solution algorithms are not stable
  - Some *ill-posed* problems can be approximated by *well-posed* similar problems
- Even solutions to *well-posed* problems can be sensitive to data
  - Computational algorithm should not increase sensitivity



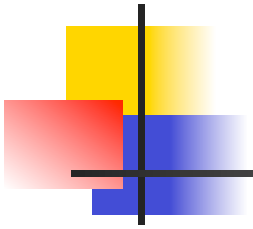
Problem Solution Strategies

- Finding a solution (and subsequently an algorithm) for a computational problem often involves replacing a difficult problem with a simpler one with identical or closely related solution
  - Replace infinite with finite formulations
  - Replace differential equations with algebraic equations
  - Replace non-linear formulations with linear ones
  - Replace complicated functions with simpler ones
  - Replace higher-order systems with lower-order ones
- Solutions may only approximate the original ones



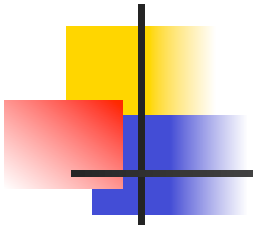
Sources of Approximation

- Problem formulation and input data
  - Simplifications in the original model of the problem
  - Errors in measurements used as input data
  - Approximations resulting from pre-computations
- Algorithm and implementation
  - Truncation and discretization as part of the algorithm design (usually resulting from simplifications of the original mathematical model)
  - Rounding as a result of the use of a finite resolution digital computer



Approximation and Error

- The accuracy of the solution produced by a numerical algorithm depends on errors introduced by the modeling and pre-computation and by the computation in the algorithm
  - The first can usually not be addressed but have to be considered when analyzing an algorithm
- The problem and the solution algorithm can have major effects on the accuracy of the approximation
  - The problem can amplify input error (sensitivity)
  - The algorithm can amplify computation errors (stability)



Approximation and Error

- The total error is generally a result of errors in the data and errors arising through the computation

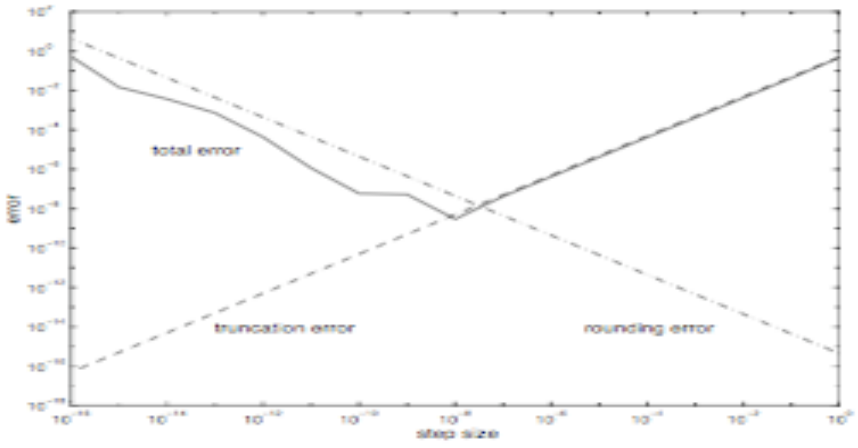
$$\hat{f}(\tilde{x}) - f(x) = \hat{f}(\tilde{x}) - f(\tilde{x}) \; + \; f(\tilde{x}) - f(x)$$

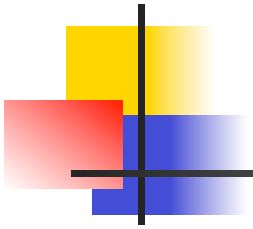
  - Computational error : $\hat{f}(\tilde{x}) - f(\tilde{x})$
  - Propagated data error: $f(\tilde{x}) - f(x)$
- Computational errors arise from simplifications in the algorithm and from numeric limitations
  - Truncation error: Caused by the algorithm
  - Rounding error: Caused by limited numeric precision
  - Truncation and Rounding often trade off

Example: Finite Difference

- Compute the derivative using finite difference approximation: $$f'(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}$$
  - Truncation error is bounded by $M\frac{\Delta x}{2}$
  - Rounding error is bounded by $2\frac{\varepsilon}{\Delta x}$
- Optimal step size: $$\Delta x \approx 2\sqrt{\varepsilon / M}$$

Sin(x) :





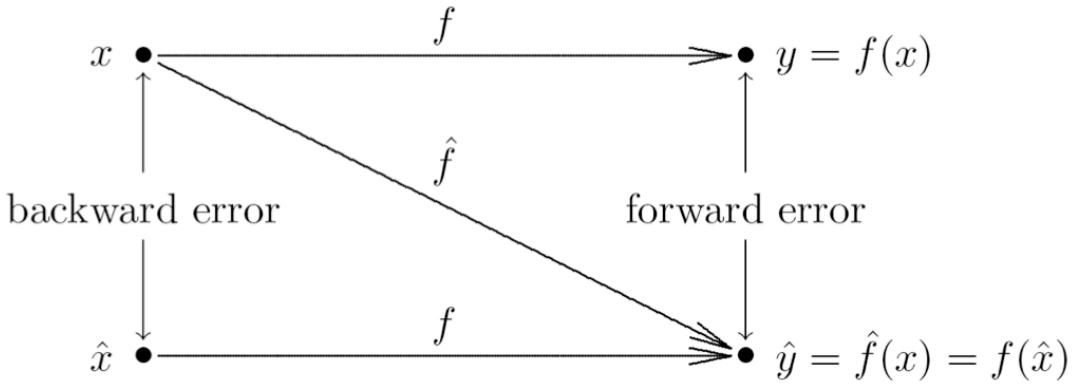
Absolute and Relative Error

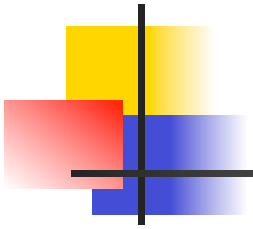
- Absolute error: $\hat{y} - y$
- Relative error: $\dfrac{\hat{y} - y}{y}$
- The true value, *y*, is generally unknown
  - Relative error is often computed relative to the approximate value
  - Error has to be approximated or calculated as a bound

Forward and Backward Error

- Error can be analyzed in the output space or in the input space of the algorithm
  - Forward error : $\Delta y = \hat{y} - y = \hat{f}(x) - f(x)$
    - error in the output of the algorithm for the same input
  - Backward error: $\Delta x = \hat{x} - x \;\; ; \;\; \hat{y} = f(\hat{x})$
    - error in the correct input corresponding to the output

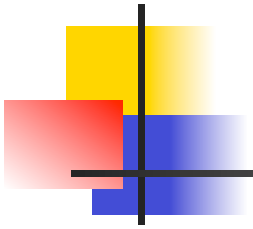
Backward error: $\Delta x = \hat{x} - x,$ where $f(\hat{x}) = \hat{y}$





Backward Error

- Backward error can be a useful analysis tool
  - Backward error captures sensitivity
    - Measures how much the original problem has to change to result in exactly the approximate solution
    - How much data error would explain the total error
  - Approximate solution is good if it has a small backward error
  - Backward error is often easier to estimate than forward error

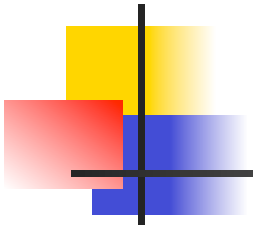


Sensitivity and Conditioning

- A problem is *sensitive* (*ill-conditioned*) if a change in the input data can cause a much larger change in the output data
- Condition number captures sensitivity:

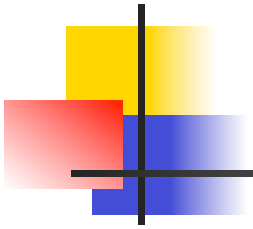
$$cond = \frac{|relative\ \ output\ \ error|}{|relative\ \ data\ \ error|} = \frac{|(f(\hat{x}) - f(x))/f(x)|}{|(\hat{x} - x)/x|} = \frac{|\Delta y / y|}{|\Delta x / x|}$$

  - A problem is sensitive if *cond >> 1*
  - Condition number represents an amplification factor between relative backward and relative forward error



Stability

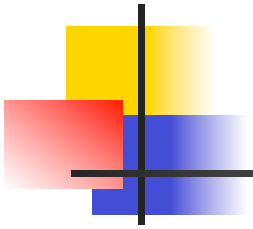
- An algorithm is *stable* if the result is relatively insensitive to perturbations caused by the computation
  - Similar to conditioning for problems
  - An algorithm is stable if it results in a small backward error (i.e. if its result is the exact solution to a similar problem)
  - If an algorithm is stable, the computational error is no worse than a small input error



Accuracy

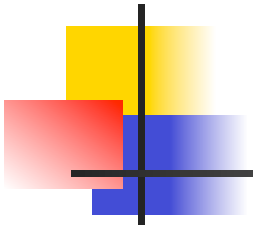
- *Accuracy* measures the similarity of the true and the computed solution
  - Accuracy depends on conditioning of the problem and stability of the algorithm
  - Stability or well-conditioning alone do not guarantee accuracy
    - A stable algorithm applied to an ill-conditioned problem can yield inaccuracy
    - An unstable algorithm applied to a well-conditioned problem can yield inaccurate results
  - To achieve accurate solutions a stable algorithm has to be applied to a well-conditioned problem

Number Representation and Rounding Errors



- Floating point numbers are used to represent continuous number
  - Real numbers can not be represented accurately
  - Operations on floating point numbers are not accurate
- Floating point numbers:
  - Base β
  - Precision p
  - Exponent range $[L,U]$

$$x = \pm\left(d_0 + \frac{d_1}{\beta} + \ldots + \frac{d_{p-1}}{\beta^{p-1}}\right)\beta^E$$



Floating Point Numbers

- Multiple floating point standards exist

Parameters for typical floating-point systems

system	β	p	L	U
IEEE SP	2	24	-126	127
IEEE DP	2	53	-1022	1023
Cray	2	48	-16383	16384
HP calculator	10	12	-499	499
IBM mainframe	16	6	-64	63

- Most floating point systems are normalized so that the first bit of the mantissa is 1
  - No digits wasted on leading zeros (saves 1 bit)

Floating Point Numbers

- Underflow: smallest positive normalized number

$$UFL = \beta^{L}$$

- Overflow: largest floating point number

$$OFL = (1 - \beta^{-p})\beta^{U+1}$$

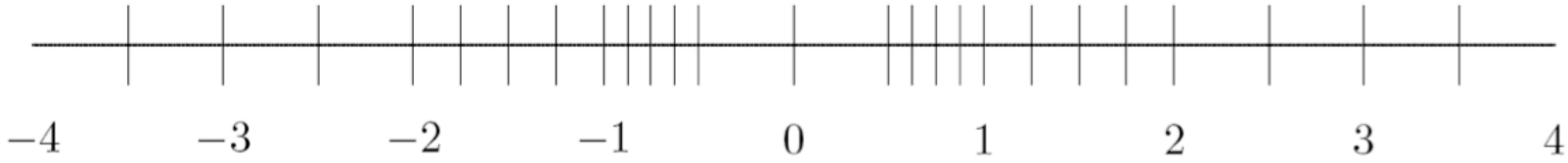
- Machine precision: smallest number larger than 1 minus 1

$$\varepsilon_{mach} = \beta^{1-p}$$

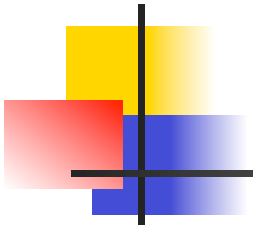
  - Machine precision bounds the rounding error:
    - With rounding to nearest: $\left| \frac{fl(x) - x}{x} \right| \leq \frac{1}{2}\varepsilon_{mach}$

Floating Point Number Example

- Representable numbers for a binary number with 3 bit mantissa and 2 bit exponent

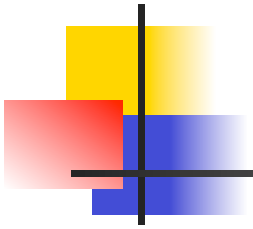


  - OFL = 3.5
  - UFL = 0.5
  - $\varepsilon_{mach} = 0.125$



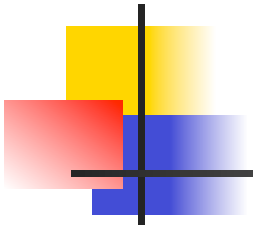
Floating Point Arithmetic

- Floating point operations introduce rounding errors
  - Addition and subtraction
    - For addition and subtraction the mantissa has to be shifted until the exponents of the numbers are equal
      - Potential loss of significant bits in the smaller number
  - Multiplication
    - Mantissas have to be multiplied, yielding theoretically a new mantissa with *2p* digits which has to be rounded
  - Division
    - Quotient of mantissas can theoretically have an infinite number of digits which have to be rounded



Floating Point Arithmetic

- Besides rounding errors, floating point operations can result in unrepresentable numbers
  - Overflow
    - Results of an overflow (a number too large to be represented) possess no good approximation and can be catastrophic.
      - On most computer systems overflow produces an error message
  - Underflow
    - Results of an underflow are usually approximated as 0.
      - On many computer systems underflow is handled silently



Floating Point Arithmetic

- Many general arithmetic laws do not strictly hold in floating point arithmetic
  - Addition and multiplication are commutative but not associative
- Underflow, overflow, and rounding can lead to incorrect results.

  Infinite sum $\sum_{n=1}^{\infty} \frac{1}{n}$

  - While this sum diverges in reality (and thus has no result) numeric calculation of it yields a finite sum
  - Partial sum no longer changes once 1/n is too small compared to the value of the partial sum

Relative Error and Loss of Significance

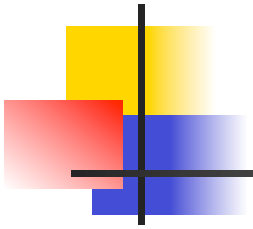
- Errors produced by well implemented arithmetic floating point operations can be modeled by

$$fl(x \ \ op \ \ y) \ = \ (x \ \ op \ \ y)(1+\delta) \ ; \ |\delta| \le \varepsilon_{mach}$$

  - Relative error bound

$$\frac{|fl(x \ \ op \ \ y) - (x \ \ op \ \ y)|}{|(x \ \ op \ \ y)|} \le \varepsilon_{mach}$$

- Error propagation can still lead to high relative error through loss of significance (or cancellation)
  - When during subtraction leading digits cancel out the result uses fewer than *p* digits and thus looses precision



Loss of Significance

- Rounding results in a loss of the least significant digits while cancellation leads to a loss of the most significant digits
  - It is generally a bad idea to compute a small number by subtracting large numbers
    - The propagated rounding error through loss of significance might ultimately dominate the actual result

Loss of Significance

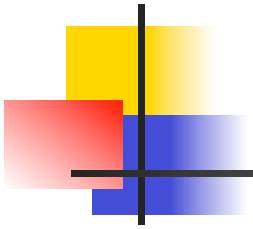
- To avoid cancellation errors, problems can sometimes be reformulated in a way that avoids the problem
  - Multiplication with conjugate expressions:

$$\sqrt{y}-x=\frac{(\sqrt{y}-x)(\sqrt{y}+x)}{\sqrt{y}+x}=\frac{y-x^2}{\sqrt{y}+x}$$

  - Application of identities to restructure the expression:

$$\frac{1-\cos x}{\sin^2 x}=\frac{1}{1+\cos x}$$

$$x=\frac{-b\pm\sqrt{b^2-4ac}}{2a}=\frac{2c}{-b\mp\sqrt{b^2-4ac}}$$



Representation and Error

- Computations on digital computers produce approximations that yield errors
  - Approximation should be taken into account when designing and analyzing algorithms
  - Approximations break into multiple types
    - Data errors
    - Computation errors
      - Truncation error due to algorithm
      - Rounding error due to representation limitations
- Errors can propagate and be amplified
  - Algorithm design should take errors into account