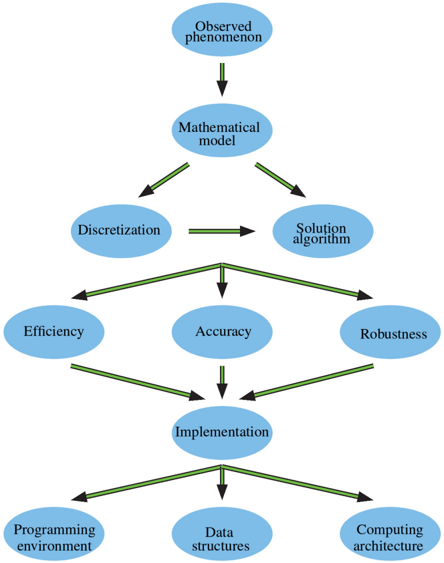
Scientific computing

Department of Mathematics
IIT Guwahati

Texts/References:

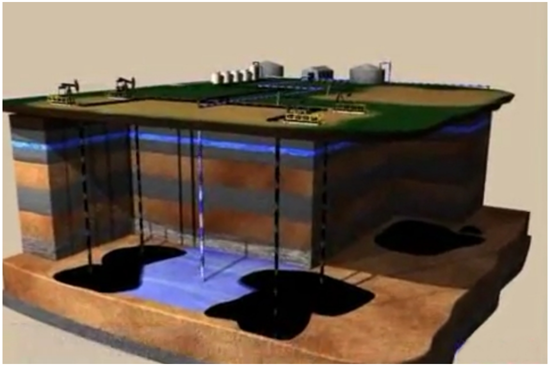
1. D. Kincaid and W. Cheney, Numerical Analysis: Mathematics of Scientific Computing, 3rd Ed., AMS, 2002.
2. G. D. Smith, Numerical Solutions of Partial Differential Equations, 3rd Ed., Calrendorn Press, 1985.
3. K. E. Atkinson, An Introduction to Numerical Analysis, Wiley, 1989.
4. S. D. Conte and C. de Boor, Elementary Numerical Analysis - An Algorithmic Approach, McGraw-Hill, 1981.
5. R. Mitchell and S. D. F. Griffiths, The Finite Difference Methods in Partial Differential Equations, Wiley, 1980.
6. Richard L. Burden, J. Douglas Faires,NUmerical analysis, ninth addition.
7. L. F. Shampine, I. Gladwell, S. Thompson, Solving ODE with Matlab, Cambridge university press 2003.

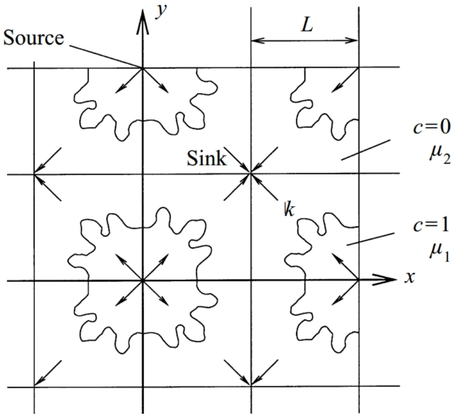
Scientific computing is a discipline concerned with the development and study of **Numerical algorithms** for solving mathematical problems that arise in various disciplines in science and engineering.



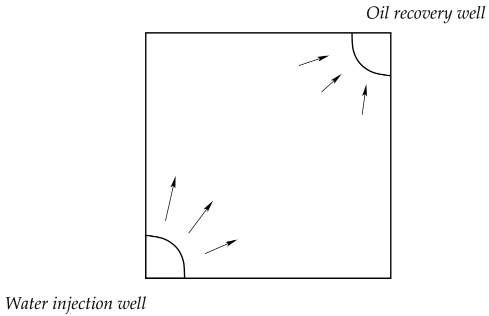
Scientific computing

Oil reservoir simulation



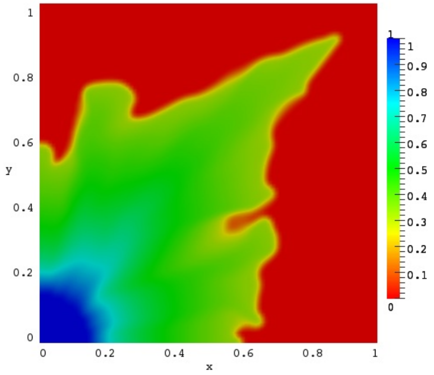


y
L
Source
$c=0$
μ_2
Sink
k
$c=1$
μ_1
x


Oil recovery well
Water injection well

Model equations:

$$\begin{aligned}\frac{\partial s}{\partial t} + \nabla \cdot \boldsymbol{F}(s, \boldsymbol{u}, \boldsymbol{x}) &= 0\\ \boldsymbol{K}^{-1}(\boldsymbol{x})\boldsymbol{u} - \nabla \cdot \big(\mu(s)\boldsymbol{\epsilon}(\boldsymbol{u})\big) + \nabla p &= s\boldsymbol{g}\\ \nabla \cdot \boldsymbol{u} &= 0\end{aligned}$$



Ref: Sudarshan K. K.; Praveen, C.; Veerappa Gowda, G. D. A finite volume method for a two-phase multicomponent polymer flooding. J. Comput. Phys. 275 (2014), 667–695.

The course will basically have the following two components:

- Numerical Analysis
- Numerical methods for ODEs and PDEs

Content: Errors; Iterative methods for nonlinear equations; Polynomial interpolation, spline interpolations; Numerical integration based on interpolation, quadrature methods, Gaussian quadrature; Initial value problems for ordinary differential equations - Euler method, Runge-Kutta methods, multi-step methods, predictor-corrector method, stability and convergence analysis; Finite difference schemes for partial differential equations - Explicit and implicit schemes; Consistency, stability and convergence; Stability analysis (matrix method and von Neumann method), Lax equivalence theorem; Finite difference schemes for initial and boundary value problems (FTCS, Backward Euler and Crank-Nicolson schemes, ADI methods, Lax Wendroff method, upwind scheme).

ERROR

The most fundamental feature of numerical computing is the inevitable presence of error. The result of any interesting computation is typically only approximate, and our goal is to ensure that the resulting error is tolerably small.

The most fundamental feature of numerical computing is the inevitable presence of error. The result of any interesting computation is typically only approximate, and our goal is to ensure that the resulting error is tolerably small.

Relative and absolute errors: Given a scalar quantity u and its approximation v :

- The **absolute error** in v is

$$|u - v|$$

- The **relative error** (assuming $u \neq 0$) is

$$\frac{|u - v|}{|u|}$$

A comparison

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

Error types

1. Errors in the problem to be solved.
2. Approximation errors
   - Discretization errors
   - Convergence errors
3. Roundoff error

Discretization error in action

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2} f''(x_0) + \frac{h^3}{6} f'''(x_0) + \frac{h^4}{24} f''''(x_0) + \cdots.$$

Then

$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - \left(\frac{h}{2} f''(x_0) + \frac{h^2}{6} f'''(x_0) + \frac{h^3}{24} f''''(x_0) + \cdots \right).$$

Our *algorithm* for approximating $f'(x_0)$ is to calculate

$$\frac{f(x_0+h) - f(x_0)}{h}.$$

The obtained approximation has the *discretization error*

$$\left| f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h} \right| = \left| \frac{h}{2} f''(x_0) + \frac{h^2}{6} f'''(x_0) + \frac{h^3}{24} f''''(x_0) + \cdots \right|.$$

Discretization error in action

$$f(x_0+h) = f(x_0) + hf'(x_0) + \frac{h^2}{2}f''(x_0) + \frac{h^3}{6}f'''(x_0) + \frac{h^4}{24}f''''(x_0) + \cdots.$$

Then

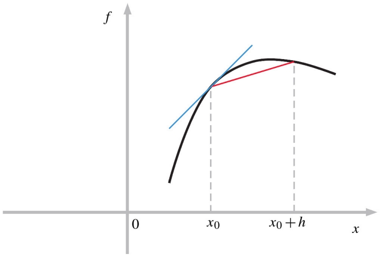
$$f'(x_0) = \frac{f(x_0+h) - f(x_0)}{h} - \left(\frac{h}{2}f''(x_0) + \frac{h^2}{6}f'''(x_0) + \frac{h^3}{24}f''''(x_0) + \cdots\right).$$

Our *algorithm* for approximating $f'(x_0)$ is to calculate

$$\frac{f(x_0+h) - f(x_0)}{h}.$$

The obtained approximation has the *discretization error*

$$\left|f'(x_0) - \frac{f(x_0+h) - f(x_0)}{h}\right| = \left|\frac{h}{2}f''(x_0) + \frac{h^2}{6}f'''(x_0) + \frac{h^3}{24}f''''(x_0) + \cdots\right|.$$



If we know $f''(x_0)$, and it is nonzero, then for h small we can estimate the discretization error by

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

If we know $f''(x_0)$, and it is nonzero, then for h small we can estimate the discretization error by

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

Damaging effect of roundoff errors!!!! The numbers in the above slide might suggest that an arbitrary accuracy can be achieved by the algorithm, provided only that we take h small enough. Indeed, suppose we want

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

Can't we just set $h \le 10^{-10}/0.5$ in our algorithm?

Not quite! Let us record results for very small, positive values of h:

h	Absolute error
1.e-8	4.361050e-10
1.e-9	5.594726e-8
1.e-10	1.669696e-7
1.e-11	7.938531e-6
1.e-13	4.250484e-4
1.e-15	8.173146e-2
1.e-16	3.623578e-1

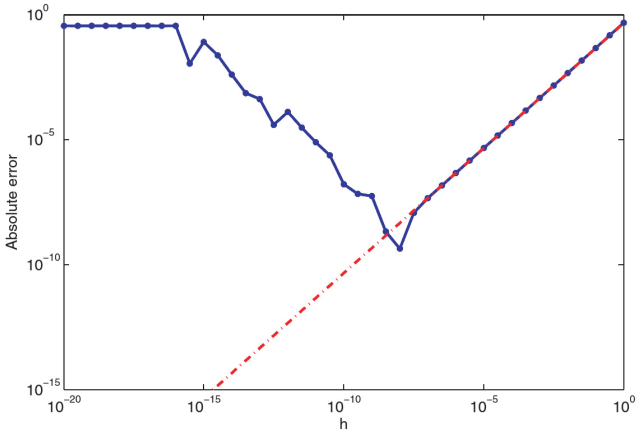


Figure *The combined effect of discretization and roundoff errors. The solid curve interpolates the computed values of* $|f'(x_0) - \frac{f(x_0+h)-f(x_0)}{h}|$ *for* $f(x) = \sin(x)$, $x_0 = 1.2$. *Also shown in dash-dot style is a straight line depicting the discretization error without roundoff error.*

Computer representation of real numbers

- Any real number $x \in \mathbb{R}$ can be represented in **decimal system** as

$$x = \pm q \times 10^e = \pm(d_0.d_1 d_2 d_3 \cdots d_{t-1} d_t d_{t+1} \cdots) \times 10^e,$$

where e is an exponent. $d_0 \neq 0$, but d_0 can take the values $1, 2, \cdots, 9$, and $d_1, d_2, d_3 \cdots$ are integers $0, 1, 2, \cdots, 9$.

Computer representation of real numbers

- Any real number $x \in \mathbb{R}$ can be represented in **decimal system** as

$$x = \pm q \times 10^e = \pm(d_0.d_1d_2d_3\cdots d_{t-1}d_td_{t+1}\cdots) \times 10^e,$$

where e is an exponent. $d_0 \neq 0$, but d_0 can take the values $1, 2, \cdots, 9$, and $d_1, d_2, d_3 \cdots$ are integers $0, 1, 2, \cdots, 9$.
Remark: The condition $d_0 \neq 0$ is to ensure a unique representation of the specified form. For example 21.3 cannot be represented as 0.213×10^{-2}.

Computer representation of real numbers

- Any real number $x \in \mathbb{R}$ can be represented in **decimal system** as

  $$x = \pm q \times 10^e = \pm(d_0.d_1d_2d_3\cdots d_{t-1}d_td_{t+1}\cdots) \times 10^e,$$

  where e is an exponent. $d_0 \neq 0$, but d_0 can take the values $1, 2, \cdots, 9$, and $d_1, d_2, d_3\cdots$ are integers $0, 1, 2, \cdots, 9$.
  Remark: The condition $d_0 \neq 0$ is to ensure a unique representation of the specified form. For example 21.3 cannot be represented as 0.213×10^{-2}.
- Any real number $x \in \mathbb{R}$ can be represented in **binary system** as

  $$x = \pm q \times 2^e = \pm(1.d_1d_2d_3\cdots d_{t-1}d_td_{t+1}\cdots) \times 2^e,$$

  where e is an exponent and $d_1, d_2, d_3\cdots$ are integers 0 and 1.

Floating point representation of real numbers

We associate a floating point representation $fl(x)$ of a form similar to that of x but with only t digits, so

$$fl(x) = sgn(x) \times (d_0.\tilde{d}_1\tilde{d}_2\tilde{d}_3 \cdots \tilde{d}_{t-1}) \times 2^e.$$

Floating point representation of real numbers

We associate a floating point representation $fl(x)$ of a form similar to that of x but with only t digits, so

$$fl(x) = sgn(x) \times (d_0.\tilde{d}_1\tilde{d}_2\tilde{d}_3\cdots\tilde{d}_{t-1}) \times 2^e.$$

Observation: Some real numbers will be by default in the floating point form. But, not all. eg. $1/3 = 3.33333333333333\cdots \times 10^{-1}$. So we need to do some approximation for this sort of real numbers.

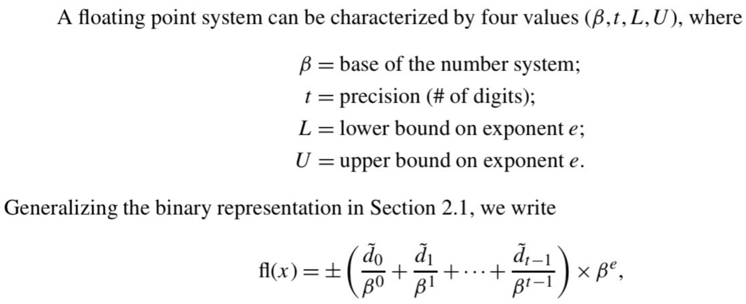
Floating point representation of real numbers

We associate a floating point representation $fl(x)$ of a form similar to that of x but with only t digits, so

$$fl(x) = sgn(x) \times (d_0.\tilde{d}_1\tilde{d}_2\tilde{d}_3\cdots\tilde{d}_{t-1}) \times 2^e.$$

Observation: Some real numbers will be by default in the floating point form. But, not all. eg. $1/3 = 3.33333333333333\cdots \times 10^{-1}$. So we need to do some approximation for this sort of real numbers.

In general form......


A floating point system can be characterized by four values (β, t, L, U), where

$$\begin{aligned} \beta &= \text{base of the number system;} \\ t &= \text{precision (\# of digits);} \\ L &= \text{lower bound on exponent } e; \\ U &= \text{upper bound on exponent } e. \end{aligned}$$

Generalizing the binary representation in Section 2.1, we write

$$\mathrm{fl}(x) = \pm\left(\frac{\tilde{d}_0}{\beta^0} + \frac{\tilde{d}_1}{\beta^1} + \cdots + \frac{\tilde{d}_{t-1}}{\beta^{t-1}}\right) \times \beta^e,$$

Remark: There can be infinitely many digits d_i s in the binary system corresponding to a finite decimal form.
Eg. $1.0 \times 10^{-1} = 1.1001100110011 \cdots \times 2^{-1}$

Definition

(Standard single-precision floating point representation) A machine number in standard single-precision floating point representation is,

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

The number is represented by $32 - bit$. Here f is the fractional part of the real number x.

- The left most bit is assigned to s for the sign. $s = 0/1$ corresponds to $+/-$ respectively.
- The next 8 bits are assigned to b. The value of b in representing a floating point number is restricted by $0 < b < (11111111)_2 = 255$.

- The values 0 and 255 are reserved for special case of 0 and $\pm\infty$. The range of representable machine numbers is mainly determined by the number of bits assigned to b. Therefore, it can be concluded that the largest positive single precision number is

$$2^{254-127} \times (1.f)_2 \approx 2^{128} \approx 3.4 \times 10^{38};$$

- The smallest positive single precision number is

$$2^{1-127} \times (1.0)_2 = 2^{-126} \approx 1.2 \times 10^{-38}.$$

- the last 23 bits are assigned to f. The value of $(1.f)_2$ is restricted by

$$1 \leq (1.f)_2 \leq (1.11\cdots 1)_2 = 2 - 2^{-23}.$$

- The significant decimal digits of accuracy is mainly determined by the number of bits assigned to f. The number of significant decimal digits of a single precision number is approximately 6, as

$$2^{-23} \approx 0.12 \times 10^{-6}.$$

Definition

(Standard double-precision floating point representation) A machine number in standard single-precision floating point representation is,

$$x = (-1)^s \times (1.f)_2 \times 2^{b-1023}.$$

The number is represented by $64 - bit$. Here f is the fractional part of the real number x.

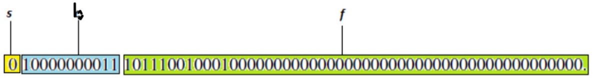
- The left most bit is assigned to s for the sign. $s = 0/1$ corresponds to $+/-$ respectively.
- The next 11 bits are assigned to b. The value of b in representing a floating point number is restricted by $0 < b < (11111111111)_2 = 2047.$

- The values 0 and 2047 are reserved for special case of 0 and $\pm\infty$.
- the last 52 bits are assigned to f. The value of $(1.f)_2$ is restricted by

$$1 \leq (1.f)_2 \leq (1.11\cdots1)_2 = 2 - 2^{-52}.$$

Consider the machine number 0 10000000011 1011100100010000000000000000000000000000000000000000.

$$(-1)^s 2^{c-1023}(1+f) = 27.56640625.$$



- For 0, set $b = 0, f = 0$, with s arbitrary; i.e., the minimal positive value representable in the system is considered 0.
- For $\pm\infty$, set $b = 1 \cdots 1, f = 0$.
- The pattern $b = 1 \cdots 1,\ f \neq 0$ is by convention `NaN`.

Home work: Give an estimate on the largest and smallest double precision number. Give an estimate on the number of significant decimal digits of a double precision number.

Example Let us plot the decimal representation of the numbers in the system specified by $(\beta, t, L, U) = (2, 3, -2, 3)$. The smallest possible number in the mantissa $d_0.d_1d_2$ is 1.00 and the largest possible number is 1.11 in binary, which is equal to $1 + 1/2 + 1/4 = 1.75$.

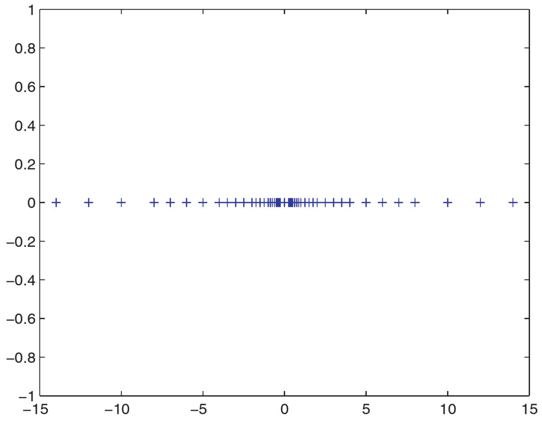


Figure *Picture of the floating point system described in Example.*