SCI 4010 COMPUTATIONAL METHODS

MSc Course on Scientific Computing
Hong Kong Baptist University

Contents

Chapter 1

Introduction

1.1 Scientific Computing

One of the earliest — and still one of the largest — uses of computers was to solve problems in science and engineering and, more specifically, to obtain solutions of mathematical models that represent some physical situation. The techniques used to obtain such solutions are part of the general area called scientific computing, and the use of the techniques to elicit insight into scientific or engineering problems is called computational science (or computational engineering).

There is now hardly an area od science or engineering that does not use computers for modeling. Trajectories for earth satellites and for planetary missions are routinely computed. Engineers use computers to simulate the flow of air about an aircraft or other aerospace vehicle as it passes through the atmosphere, and to verify the structural integrity of aircraft. Such studies are of crucial importance to the aerospace industry in the design of safe and economical aircraft and space craft. Modeling new designs on a computer can save many millions of dollars compared to building a series of phototypes. Similar considerations apply to the design of automobiles and many other products, including new computers.

The mathematical models of all of the practical problems are systems of differential equations, either ordinary or partial. Differential equations come in all ”sizes and shapes”, and even with the pargest computers we are nowhere near being able to solve many of the problems posed by science and engineers. But there is more to scientific computing, and the scope of the field is changing rapidly. There are many other mathematical models, each with its own challenges. In operations research and economics, large linear and nonlinear optimization problems need to be solved. Data reduction – the condensation of a large number of measurements into usable statistics – has always been an important part of scientific computing.

Traditionally, science and technology are regarded as being composed of two parts, theory and experiment. However, the incredible pace at which science and technology have developed means that such a view must now be modified. In 1982, a group of fifteen distinguished scientists and engineers presented a report to the U.S. government in which they argued that “science and technology is no longer divided into just two parts – theoretical and experimental; there is now a third, equal part – computational.”

Advantages of Scientific Computing

1. In principal, all mathematical equations can be solved numerically (compared with theoretical approach)

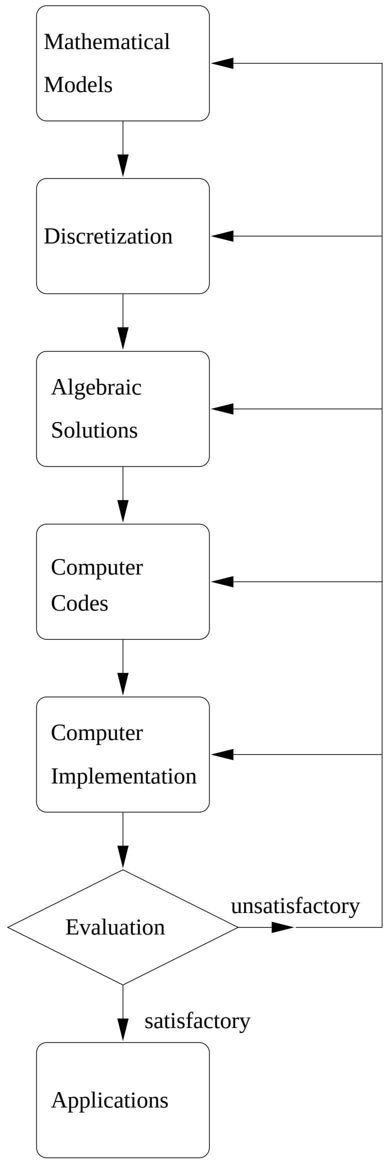
2. Low cost, high efficiency and no danger (compared with experimental approach)

Reference Books:

1. **G.H. Golub & J.M. Ortega:** *Scientific Computing – An Introduction to Parallel Computing*, Academic Press, 1993.

2. **K.E. Atkinson:** *Elementary Numerical Analysis*, John Wiley & Sons, Inc., 1993

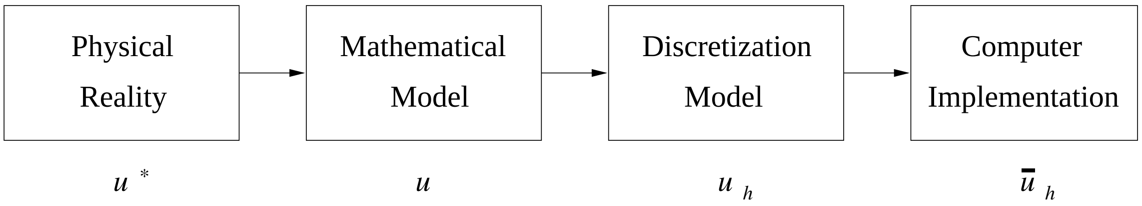
Process of Scientific Computing



Scientific computing draws on mathematics and computer science to develop the best ways to use computer systems to solve problems from science and engineering. This relationship is depicted schematically in the above figure.

1.2 Error

1.2.1 Sources of Error



$$
\begin{aligned}
u^* - \bar{u}_h &= (u^* - u) + (u - u_h) + (u_h - \bar{u}_h) \\
\|u^* - \bar{u}_h\| &\le \underset{\substack{\uparrow \\ \text{Mathematical} \\ \text{Analysis}}}{\|u^* - u\|} + \underbrace{\|u - u_h\| + \|u_h - \bar{u}_h\|}_{\substack{\uparrow \\ \text{Numerical} \\ \text{Analysis}}}
\end{aligned}
$$

The errors on the right hand side of the last equations are come from

- $\|u^* - u\|$: due to simplification of mathematical model and error in the input data
- $\|u - u_h\|$: discretization error, due to approximation for infinite dimension problem by finite dimension one
- $\|u_h - \bar{u}_h\|$: due to rounding off, limit of an infinite sequence terminated in finite steps.

1.2.2 Computer Representation of Numbers

Definition 1.1 Number System: *A number system in* **base** β*,*

$$x = \sigma \cdot (.a_1 a_2 \cdots a_t)_\beta \cdot \beta^e$$

where $\sigma = +1/-1,\ a_1 \neq 0,\ 0 \le a_j \le \beta - 1,\ e$ *integer, it means that*

$$x = \pm\left(\frac{a_1}{\beta} + \frac{a_2}{\beta^2} + \cdots + \frac{a_t}{\beta^t}\right) \cdot \beta^e .$$

In the above, σ is the `sign`, e an `exponent`, β an `base`, $(.a_1 \cdots a_t)_\beta$ an `mantissa`. Most digital computers use the base 2, i.e., Binary number system or some variant of it such as base 8 or base 16. Decimal number system is formed when $\beta = 10$. For example,

$$136.31 = +(.13631) \cdot 10^3 = +\left(\frac{1}{10} + \frac{3}{10^2} + \frac{6}{10^3} + \frac{3}{10^4} + \frac{1}{10^5}\right) \cdot 10^3 .$$

In a typical computer, only a relatively small subset of the real number system is used for the representation of all the real numbers. This subset contains only rational numbers, both positive and negative, and stores a fractional part, called the `mantissa`, together with an exponential part, called the `characteristic` or `exponent`. For example, a single-precision `floating point` number used in the IBM 3000 and 4300 series consists of a 1-binary-digit (`bit`) sign indicator, a 7-bit exponent with a base of 16, and a 24-bit mantissa.

Since 24 binary digits correspond to between 6 and 7 decimal digits, we can assume that this number has at least 6 decimal digits of `precision` for the floating-point number system. The exponent of 7 binary digits gives a range of 0 to 127. However, using only positive intergers for the characteristic/exponent does not permit an adequate representation of numbers with small magnitude. To ensure that numbers with small magnitude are equally representable, 64 is subtracted from the characteristic, so the range of the exponential part is acturaly from -64 to +53.

Floating-point representation of a binary number in computer

σ	e	mantissa

Consider, for example, the machine number

0	1000010	101100110000010000000000

The leftmost bit is a zero, which indicates that the number is positive. The next seven bits, 1000010, are equivalent to the decimal number

$$1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 66$$

and are used to describe the characteristic, 16^{66-64}. The final 24 bits indicate that the mantissa is

$$1 \cdot (1/2)^1 + 1 \cdot (1/2)^3 + 1 \cdot (1/2)^4 + 1 \cdot (1/2)^7 + 1 \cdot (1/2)^8 + 1 \cdot (1/2)^{14} .$$

As a consequence, this machine number precisely represents the decimal number

$$+[1 \cdot (1/2)^1 + 1 \cdot (1/2)^3 + 1 \cdot (1/2)^4 + 1 \cdot (1/2)^7 + 1 \cdot (1/2)^8 + 1 \cdot (1/2)^{14}]16^{66-64} = 179.015625.$$

Parameters for a computer to specify its arithmetic characteristic are β, t, L, U, where

- β — the base;

- t — the number of digits of mantissa;
- L — the lower bound;
- U — the upper bound.

To ensure `uniqueness` of representation and obtain all the available precision of the system, a `normalization` is imposed that at least one of the four leftmost bits of the mantissa of a machine number is a 1. The smallest normalized number, positive machine number that can be represented is

0	0000000	000100000000000000000000

which equals to $16^{-65} \approx 10^{-78}$, while the largest is

0	1111111	111111111111111111111111

which is approximately $16^{63} \approx 10^{76}$.

1.2.3 Chopping and Rounding

Consider a real number x of the form $x = \sigma \cdot (.a_1 \cdots a_t a_{t+1} \cdots)_\beta \cdot \beta^e$.

Definition 1.2 Chopped *machine representation for the number x is*

$$fl(x) = \sigma \cdot (.a_1 \cdots a_t)_\beta \cdot \beta^e \,.$$

Rounded *machine representation*

$$fl(x) = \begin{cases} \sigma \cdot (.a_1 \cdots a_t)_\beta \cdot \beta^e, & 0 \le a_{t+1} < \frac{\beta}{2} \\ \sigma \left[(.a_1 \cdots a_t)_\beta + (.\underbrace{0 \cdots 0}_{t-1} 1)_\beta \right] \cdot \beta^e, & \frac{\beta}{2} \le a_{t+1} < \beta \end{cases}$$

Error: Denote $fl(x) = x(1+\epsilon)$. Then $-\beta^{-t+1} \le \epsilon \le 0$ for chopping, and $-\frac{1}{2}\beta^{-t+1} \le \epsilon \le \frac{1}{2}\beta^{-t+1}$ for rounding.

Definition 1.3 Underflow and Overflow: *Let x_U be the largest number can be represented in computer and x_L be the smallest one. All machine number x must satisfy*

$$x_L \le |x| \le x_U \,.$$

If $|x| < x_L$, then $fl(x)$ will be set to zero and computation will continue. This is called Underflow. *If $|x| > x_U$, then there will be fatal error and the program will terminate.* This is called Overflow.

Impact of Rounding/Chopping. The rounding/chopping errors are due to the finite length of the floating-point mantissa. In general, the term rounding error includes chopping where applicable. Because of rounding error, the basic laws of arithmetic such as associative law and distributive law do not hold in computer. For example,

$$fl[(1+2^{-t-1})+2^{-t-1}] = 1 \qquad \text{by rounding}$$
$$fl[1+(2^{-t-1}+2^{-t-1})] = 1+2^{-t} \qquad \text{by rounding}$$

1.2.4 Loss of Significant Digits

Consider the evaluation of

$$f(x) = x[\sqrt{x+1}-\sqrt{x}]$$

Using a six-digit decimal calculator, the results of $f(100)$ is

$$\sqrt{100} = 10.0000, \quad \sqrt{101} = 10.0499, \quad \sqrt{101}-\sqrt{100} = 0.0499000$$

which leads to $f(100) = 4.99000$. The following is a simple way to reformulate so as to avoid the loss of significance error:

$$f(x) = x[\sqrt{x+1}-\sqrt{x}] \cdot \frac{\sqrt{x+1}+\sqrt{x}}{\sqrt{x+1}+\sqrt{x}} = \frac{x}{\sqrt{x+1}+\sqrt{x}}.$$

Then Using the above formula we can show that $f(100) = 4.98756$, which agrees with the true solution to within 6 digits.

Homework 1.1 1. Let $a_j = 2^{-t-1}, 1 \le j \le N$, where $N = 2^{t+1}$. Show that

$$fl[1+a_0+a_1+\cdots+a_N] = 1,$$
$$fl[a_0+a_1+\cdots+a_N+1] = 2.$$

2. Give an example to demonstrate

$$fl[a\cdot(b+c)] \ne fl[a\cdot b + a\cdot c].$$

3. Using the IBM mainframe format, find the decimal equivalent of the following floating-point machine numbers:

a.	0	1000011	101010010011000000000000
b.	1	0111111	010001111000000000000001.

4. Reformulate the following functions to avoid the loss of significance error when $x \approx 0$

$$(\textbf{a}.) \quad (1-\cos x)/x^2, \qquad (\textbf{b}.) \quad \sqrt[3]{1+x}-1.$$

Evaluate the above functions at $x = 0.02$ with a six-digit calculator by using the direct method and the modified formula.

1.2.5 Propagation of Errors

Consider the computation of the sum

$$S = \sum_{j=1}^{m} x_j$$

with $x_1, \cdots x_m$ floating-point numbers. Define

$$\begin{aligned} S_2 &= fl(x_1 + x_2) \\ S_{\gamma+1} &= fl(S_\gamma + x_{\gamma+1}), \quad \gamma = 2, \cdots, m-1 \end{aligned}$$

Due to the rounding error,

$$\begin{aligned} S_2 &= (x_1 + x_2)(1 + \epsilon_2) \\ S_{\gamma+1} &= (S_\gamma + x_{\gamma+1})(1 + \epsilon_{\gamma+1}) \end{aligned}$$

where ϵ_j is random number in interval $[-\frac{1}{2}\beta^{-t+1}, \frac{1}{2}\beta^{-t+1}]$. If $\beta = 2$, then $[-2^{-t}, 2^{-t}]$. Therefore, we have

$$\Rightarrow \begin{cases} \begin{aligned} S_2 - (x_1 + x_2) &= (x_1 + x_2)\epsilon_2 \\ S_3 - (x_1 + x_2 + x_3) &= (x_1 + x_2)\epsilon_2 + (x_1 + x_2)(1 + \epsilon_2)\epsilon_3 \\ &\quad + x_3\epsilon_3 \\ &\approx (x_1 + x_2)\epsilon_2 + (x_1 + x_2 + x_3)\epsilon_3 \\ S_4 - (x_1 + x_2 + x_3 + x_4) &\approx (x_1 + x_2)\epsilon_2 + (x_1 + x_2 + x_3)\epsilon_3 \\ &\quad + (x_1 + x_2 + x_3 + x_4)\epsilon_4 \\ S_m - \sum_{j=1}^{m} x_j &\approx (x_1 + x_2)\epsilon_2 + \cdots + (x_1 + \cdots + x_m)\epsilon_m \\ &= x_1(\epsilon_2 + \cdots \epsilon_m) + x_2(\epsilon_2 + \cdots + \epsilon_m) \\ &\quad + x_3(\epsilon_3 + \cdots + \epsilon_m) + x_4(\epsilon_4 + \cdots + \epsilon_m) \\ &\quad + \cdots + x_m\epsilon_m \end{aligned} \end{cases}$$

From the above formulas, we deduce that the best strategy for addition is to add from the smallest to the largest.

Inner Products: Consider to compute

$$S = \sum_{j=1}^{m} x_j y_j$$

Let

$$\begin{aligned} S_1 &= fl(x_1 y_1) \\ S_{\gamma+1} &= fl(S_\gamma + fl(x_{\gamma+1} y_{\gamma+1})), \quad \gamma = 1, 2, \cdots m \end{aligned}$$

Then

$$\begin{aligned} S_1 &= x_1 y_1 (1+\epsilon_1), \\ S_2 &= [S_1 + x_2 y_2 (1+\epsilon_2)](1+\eta_2), \\ \vdots \quad & \vdots \quad \vdots \\ S_m &= [S_{m-1} + x_m y_m (1+\epsilon_m)](1+\eta_m), \end{aligned}$$

where ϵ_j, η_j are random numbers uniformly distributed in $[-\frac{1}{2}\beta^{-t+1}, \frac{1}{2}\beta^{-t+1}]$. This gives

$$\Rightarrow S_m = \sum_{j=1}^{m} x_j y_j (1+\gamma_j)$$

with

$$\begin{aligned} 1+\gamma_j &= (1+\epsilon_j)(1+\eta_j)(1+\eta_{j+1})\cdots(1+\eta_m) \\ &\approx 1+\epsilon_j+\eta_j+\eta_{j+1}+\cdots+\eta_m . \end{aligned}$$

Let $\delta = \frac{1}{2}\beta^{-t+1}$. We have $|\gamma_j| \le (m+1-j+1)\delta \le m\delta, \quad \forall j$. Therefore,

$$|S - S_m| \le \sum_{j=1}^{m} |x_j y_j \gamma_j| \le m\delta \cdot \sum_{j=1}^{m} |x_j y_j| \le m\delta \, \|x\|_2 \, \|y\|_2$$

where $\|x\|_2$ is the standard l^2-norm, defined by

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_m^2} .$$

1.3 Stability

For a mathematical problem, if a small change in the input data leads to a small change in its solution, we say that the problem is stable or well-posed. Otherwise it is unstable or ill-posed.

For example, consider the solution of linear system

$$Ax = b$$

where A is nonsingular square matrix. We further consider the perturbed system

$$A\tilde{x} = b + \delta b$$

If $\|\tilde{x} - x\|/\|x\|$ is small whenever $\|\delta b\|/\|b\|$ is small, then we say the problem $Ax = b$ is stable. Otherwise it is unstable. *Note there here no rounding is taken into account.*

We further consider the Hilbert matrix defined by

$$A_n = \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \cdots & \frac{1}{n+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n+1} & \frac{1}{n+2} & \cdots & \frac{1}{2n-1} \end{bmatrix}$$

Linear systems with Hilbert matrix

$$A_n x = b$$

are unstable or ill-posed.

To reduce the effects of round-off error, we can use high-order digit arithmetic such as the double- or multiple-precision option available on most digital computers. A disadvantage in using double-precision arithmetic is it takes more computer time, and the growth of round-off error is not eliminated but only postponed until subsequent computations are performed.

1.3.1 Stability of Numerical Algorithm

Definition 1.4 *A* numerical algorithm *for solving a mathematical problem is considered to be* **stable***, if its* numerical solution*, which has a small perturbation in the input data, is close to the exact solution.*

Example : *We solve the following system*

$$10^{-11}(0.3)x_1 + x_2 = 0.7 \tag{1.1}$$

$$x_1 + x_2 = 0.9 \tag{1.2}$$

by Gauss elimination with ten decimal digits.

Algorithm 1

Step 1: eliminate x_1 from equation (1.2),
$-10^{12}(0.3333333333)x_2 = -10^{12}(0.2333333333)$
$\Rightarrow x_2 = 0.7000000000$

Step 2: substitute the above result into (1.1), and get
$x_1 = 0.0000000000$

Algorithm 2

Step 1: eliminate x_1 from equation (1.1),
$x_2 = 0.7000000000$
Step 2: substitute the above result into (1.2), and get
$x_1 = 0.2000000000$

The Algorithm 2 above is stable while the Algorithm 1 is unstable. Their numerical results are extremely different.

Example : *Recursion formula for computing*

$$y_n = \int_0^1 \frac{x^n}{x+5}\,dx, \qquad n = 0, 1, \cdots 8.$$

Reformulating from the above equations yields

$$\begin{aligned} y_n + 5y_{n-1} &= \int_0^1 \frac{x^n + 5x^{n-1}}{x+5}\,dx \\ &= \int_0^1 x^{n-1}\,dx = \frac{1}{n}\,. \end{aligned}$$

This gives

$$\Rightarrow y_n = \frac{1}{n} - 5y_{n-1}, \quad n = 1, \cdots 8\,.$$

Assume only 3 decimals is used:

$$\begin{aligned} y_0 &= \int_0^1 \frac{dx}{x+5} = \ln(x+5)|_0^1 = \ln 6 - \ln 5 \approx 0.182 \\ y_1 &= 1 - 5y_0 \approx 0.090 \\ y_2 &= \frac{1}{2} - 5y_1 \approx 0.050 \\ y_3 &= \frac{1}{3} - 5y_2 \approx 0.083 \qquad \text{strange that } y_3 > y_2 \\ y_4 &= \frac{1}{4} - 5y_3 \approx -0.165 \qquad \text{obviously absurd that } y_4 < 0. \end{aligned}$$

The reason for the absurd results is that in each step the propagation error is enlarged by 5 times

A stable algorithm: Use the recursion formula in the other direction

$$y_{n-1} = \frac{1}{5n} - \frac{y_n}{5}$$

Now the error will divided by 5 in each step.

Since $y_{10} \approx y_9$, $y_9 + 5y_9 \approx \dfrac{1}{10}$, $\Rightarrow \quad y_9 \approx \dfrac{1}{60} \approx 0.017$, and also

$$\begin{array}{lll} y_8 = \dfrac{1}{45} - \dfrac{y_9}{5} \approx 0.019 & y_7 = \dfrac{1}{40} - \dfrac{y_8}{5} \approx 0.021 & \\ y_6 \approx 0.025 & y_5 \approx 0.028 & y_4 \approx 0.034 \\ y_3 \approx 0.043 & y_2 \approx 0.058 & y_1 \approx 0.088 \\ y_0 \approx 0.182 & \text{correct!} & \end{array}$$

Homework 1.2 *:*

1. The sequence $p_n = (1/3)^n$, $n > 0$, can be generated recursively by the following algorithms:

$$\begin{array}{lll} \textsf{Algorithm a}: & p_0 = 1, & p_n = \dfrac{1}{3}p_{n-1} \quad \text{for} \quad n > 1; \\ \textsf{Algorithm b}: & p_0 = 1,\ p_1 = \dfrac{1}{3} & p_n = \dfrac{10}{3}p_{n-1} - p_{n-2} \quad \text{for} \quad n > 2. \end{array}$$

   Verify that

   - *(1).* In theory both formulas are correct;
   - *(2).* Using six-digit rounding arithmetic to demonstrate that Algorithm a is stable;
   - *(3).* Using six-digit rounding arithmetic to demonstrate that Algorithm b is unstable.