Numerical Analysis

Yutian LI

CUHKSZ

2018/19 Term 1

Reference Books

BF R. L. Burden and J. D. Faires, Numerical Analysis, 9th edition, Thomsom Brooks/Cole, 2010.

At K. E. Atkinson, An Introduction to Numerical Analysis, 2nd edition, John Wiley, 1989.

DMN D. Kahaner, C. Moler and S. Nash, Numerical Methods and Software, Prentice-Hall, 1989.

St G. W. Stewart, Afternotes on Numerical Analysis, SIAM, 1996.

Software

- Matlab
- Python
- Julia

Contents

Chapter 1: Introduction

1. Overview & Motivation
2. Computational Errors
3. Computer Arithmetic
4. Convergence and Stability
5. Some basic algorithms and Matlab codes

Overview & Motivation

1. Solve Nonlinear Equations (Chapter 2)

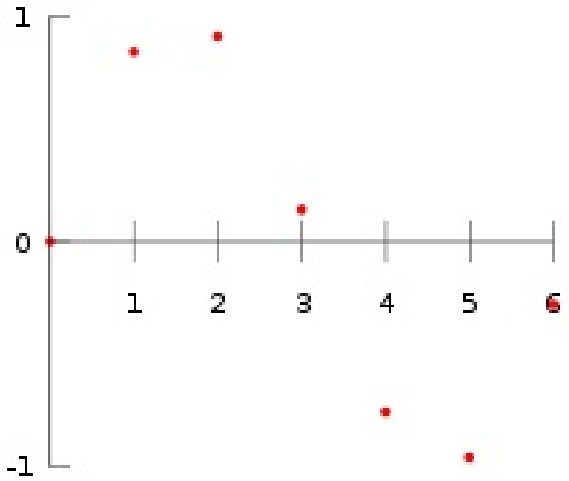
The LambertW function is the inverse function of $f(w) = we^w$: that is

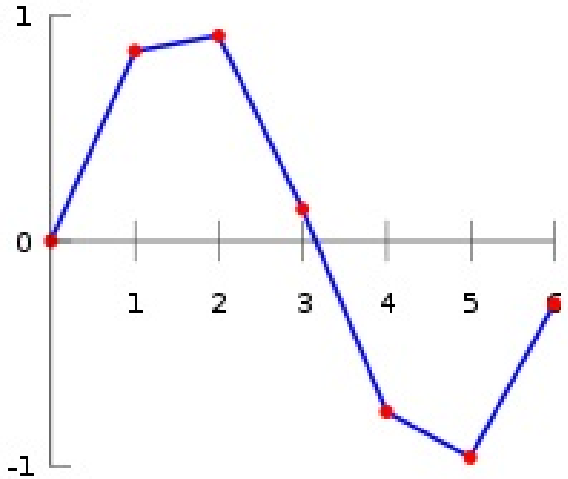
$$z := w(z)e^{w(z)}.$$

One needs to solve a nonlinear equation to find the value of $w(z)$. e.g., to fnid $w(1)$, we need solve

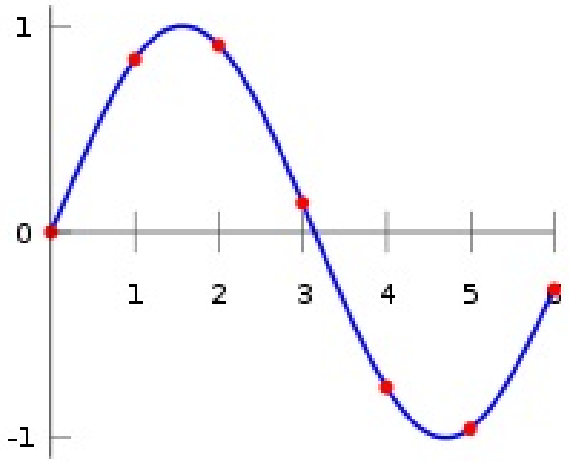
$$we^w = 1.$$

2. Interpolation (Chapter 3)





(a) piecewise linear function



(b) polynomials

3. Numerical Integration (Chapter 4)

The c.d.f. (cumulative distribution function) of the standard normal distribution is given by

$$N(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} dt = \frac{1}{2} + \frac{1}{\sqrt{2\pi}} \int_{0}^{x} e^{-\frac{t^2}{2}} dt$$

To find the (approximate) values of $N(x)$, we need perform a numerical method to evaluate the above integral. This is the quadrature rules.

4. Solve linear system (Chapter 5)

In solving the application problems, the last step is often to find the solution of a linear system of equations

$$AX = b.$$

(e.g., when we use finite difference or finite element methods to solve a differential equation)
When A is large, it is necessary to use some numerical method and to solve it on a computer.

5. Least squares

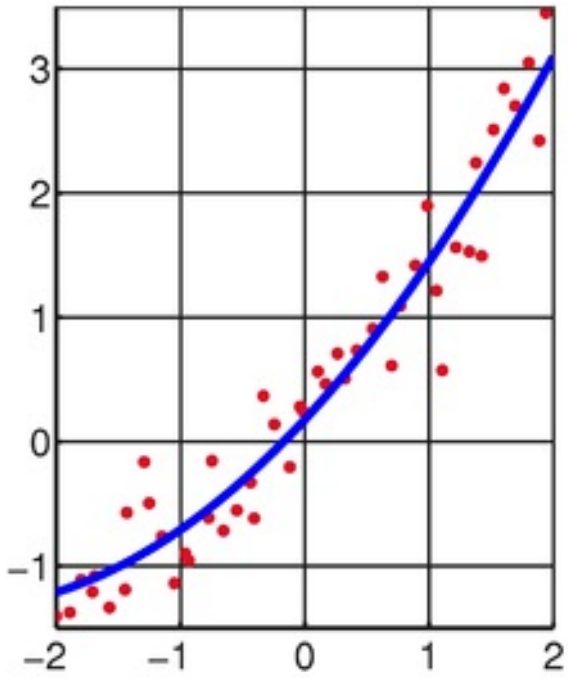


Figure: The result of fitting a set of data points with a quadratic function

Computational Errors

- Truncation error —— the error made by numerical algorithms that arises from taking finite number of steps in computation.
- Round-off error —— the error produced when a computer is used to perform real number calculations.

Truncation error:

Consider Taylor's theorem

$$f(x) = P_n(x) + R_n(x),$$

where

$$\begin{aligned} P_n(x) =& f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots + \frac{f^{(n)}(x_0)}{n!}(x - x_0)^n, \\ R_n(x) =& \frac{f^{(n+1)}(\xi)}{(n+1)!}(x - x_0)^{n+1}. \end{aligned}$$

If we use $P_n(x)$ to approximate $f(x)$, the error $R_n(x)$ is called the truncation error.

Round-off error:

- In the computational world, each representable number has only a fixed and finite number of digits.
- Only a small subset of the real number can be represented in a typical computer

So for example, $1/3$, π, $\sqrt{2}$, these numbers cannot be represented exactly in the computation by a computer.

Round-Off Error Can Be Fatal.

On February 25, 1991 an Iraqi Scud missle hit an American Army barracks and killed 28 soldiers. The Patriot Missle System that was supposed to intercept and destroy the Scud failed. The General Accounting Office report, GAO/IMTEC-92-26, Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia gave the reason as round-off error.

See `www.math.psu.edu/dna/455.f97/notes.html`.

When we do computations using computers, the round-off errors are inevitable. But we can try to minimize the negative impact of these errors by rewriting the formula you wish to compute. This must be done on a case-by-case basis.

Computer Arithmetic

– Binary Floating Point Number.

In 1985, the IEEE (Institute for Electrical and Electronic Engineers) published a report called Binary Floating Point Arithmetic Standard 754–1985. An updated version was published in 2008 as IEEE 754-2008. A 64-bit (binary digit) representation is used for a real number. The first bit is a sign indicator, denoted s. This is followed by an 11-bit exponent, c, called the *characteristic*, and a 52-bit binary fraction, f, called the *mantissa*. The base for the exponent is 2.

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

$e.g.,$ 0 10000000011 1011100100010000000000000000000000000000000000000000

$$\begin{aligned} s =& 0, \\ c =& 2^{10} + 0 \cdot 2^9 + 0 \cdot 2^8 + \cdots + 2^1 + 2^0 = 1027, \\ f =& 1 \cdot 2^{-1} + 2^{-3} + 2^{-4} + 2^{-5} + 2^{-8} + 2^{-12} \end{aligned}$$

and the machine number precisely represents the decimal number

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

Numbers occurring in calculations that have a magnitude less than

$$2^{-1022} \cdot (1+0) \approx 0.22251 \times 10^{-307}$$

result in **underflow** and are generally set to zero. Numbers greater than

$$2^{1023} \cdot (2 - 2^{-52}) \approx 0.17977 \times 10^{309}$$

result in **overflow** and typically cause the computations to stop.

– Decimal floating-point number

$$\pm 0.d_1 d_2 \cdots d_k \times 10^n, \qquad 1 \leqslant d_1 \leqslant 9, \quad 0 \leqslant d_2, \, d_3 \cdots \leqslant 9.$$

Any positive real number within the numerical range of the machine can be normalized to the form

$$y = 0.d_1 d_2 \cdots d_k d_{k+1} d_{k+2} \cdots \times 10^n.$$

The floating-point form of y, denoted $fl(y)$, is obtained by terminating the mantissa of y at k decimal digits.

There are two common ways of performing this termination

Chopping: simply chops off the digits $d_{k+1}d_{k+2}\cdots$ This produces the floating-point form

$$fl(y) = 0.d_1d_2\cdots d_k \times 10^n.$$

Rounding: adds $5 \times 10^{n-(k+1)}$ to y and then chops the result to obtain a number of the form

$$fl(y) = 0.\delta_1\delta_2\cdots\delta_k \times 10^n.$$

For rounding, when $d_{k+1} \geqslant 5$, we add 1 to d_k to obtain $fl(y)$; that is, we *round up*. When $d_{k+1} < 5$, we simply chop off all but the first k digits; that is, *round down*.
The error that results from replacing a number with its floating-point form is called **round-off error** regardless of whether the rounding or the chopping method is used.

Example

Determine the five-digit (a) chopping and (b) rounding values of π.

The number π has an infinite decimal expansion of the form $\pi = 3.14159265\cdots$. Written in normalized decimal form, we have

$$\pi = 0.314159265\cdots \times 10^1.$$

The floating-point form of π using five-digit chopping is

$$fl(\pi) = 0.31415 \times 10^1 = 3.1415.$$

Since the sixth digit of the decimal expansion of π is a 9, the floating-point of π using five-digit rounding is

$$fl(\pi) = (0.31415 + 0.00001) \times 10^1 = 3.1416.$$

Definition

Suppose that p^* is an approximation to p. The **actual error** is $p - p^*$, the **absolute error** is $|p - p^*|$, and the **relative error** is $\frac{|p-p^*|}{|p|}$, provided that $p \neq 0$.

Definition

The number p^* is said to approximate p to t **significant digits** (or figures) if t is the largest nonnegative integer for which

$$\frac{|p - p^*|}{|p|} \leqslant 5 \times 10^{-t}.$$

Example

Determine the actual, absolute, and relative errors when approximating p by p^* when

(a) $p = 0.3000 \times 10^1$ and $p^* = 0.3100 \times 10^1$;

(b) $p = 0.3000 \times 10^{-3}$ and $p^* = 0.3100 \times 10^{-3}$;

(c) $p = 0.3000 \times 10^4$ and $p^* = 0.3100 \times 10^4$.

As a measure of accuracy, the absolute error can be misleading and the relative error more meaningful because the relative error takes into consideration the size of the value.

- Inaccurate representation of numbers.

We see that the floating-point representation $fl(y)$ for the number y has the relative error

$$\left|\frac{y - fl(y)}{y}\right|.$$

If k decimal digits and chopping are used for the machine representation of

$$y = 0.d_1 d_2 \cdots d_k d_{k+1} \cdots \times 10^n,$$

then

$$\left|\frac{y - fl(y)}{y}\right| = \left|\frac{0.d_{k+1} d_{k+2} \cdots \times 10^{n-k}}{0.d_1 d_2 \cdots \times 10^n}\right| \leqslant \frac{1}{0.1} \times 10^{-k} = 10^{-k+1}.$$

In a similar manner, a bound for the relative error when using k-digit rounding arithmetic is $0.5 \times 10^{-k+1}$.

- Inaccurate calculation.

Assume that the floating-point representations $fl(x)$ and $fl(y)$ are given for the real numbers x and y and that the symbols $\oplus$, $\ominus$, $\otimes$, $\oslash$ represent machine addition, substraction, multiplication and division operations, respectively. We will assume a finite-digit arithmetic given by

$$x \oplus y = fl(fl(x) + fl(y)), \qquad x \ominus y = fl(fl(x) - fl(y)),$$
$$x \otimes y = fl(fl(x) \times fl(y)), \qquad x \oslash y = fl(fl(x) \div fl(y)).$$

Example

Suppose $x = \frac{5}{7}$ and $y = \frac{1}{3}$ we want to do arithmetic calculations involving x and y.

$$x = 0.714285714\cdots \times 10^0, \qquad y = 0.333333333\cdots \times 10^0.$$

- Five-digit chopping

$$\begin{aligned} x \oplus y =& fl(fl(x) + fl(y)) = fl(0.71428 \times 10^0 + 0.33333 \times 10^0) \\ =& fl(1.04761 \times 10^0) = 0.14076 \times 10^1 \end{aligned}$$

- Five-digit rounding

$$\begin{aligned} x \oplus y =& fl(fl(x) + fl(y)) = fl(0.71429 \times 10^0 + 0.33333 \times 10^0) \\ =& fl(1.04762 \times 10^0) = 0.14076 \times 10^1 \end{aligned}$$

$$\text{Absolute Error} = \left| \frac{22}{21} - 0.10476 \times 10^{1} \right| = 0.190 \times 10^{-4},$$

$$\text{Relative Error} = \left| \frac{0.190 \times 10^{-4}}{\frac{22}{21}} \right| = 0.182 \times 10^{-4},$$

- How to avoid loss of accuracy due to round-off error?

– Reformulate the problem.

One of the most common error producing calculations involves the cancelation of significant digits due to subtraction of nearly equal numbers. So we should reformulate the problem to avoid subtracting nearly equal numbers.

Example

The two roots of $ax^2 + bx + c = 0$ are given by

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a},$$

Consider the equation $x^2 + 62.10x + 1 = 0$. The two roots are

$$x_1 = -0.01610723, \qquad x_2 = -62.08390.$$

In this equation, b^2 is much larger than $4ac$, so the numerator of x_1 is a subtraction of two nearly equal numbers. Use 4-digit rounding we get

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = -0.02000.$$

which is a poor approximation to x_1 with the large relative error

$$\left|\frac{-0.01611 + 0.02000}{-0.01611}\right| = 0.24 \times 10^0.$$

To improve the accuracy of the calculation, we change the formula by rationalizing the numerator,

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \times \frac{-b - \sqrt{b^2 - 4ac}}{-b - \sqrt{b^2 - 4ac}} = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.$$

Using this we get

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = -0.01610,$$

which has the small relative error 6.2×10^{-4}.

– Rewrite polynomials into nested form.

Example

Evaluate $f(x) = x^3 - 6.1x^2 + 3.2x + 1.5$ for $x = 4.71$.

Using 3-digit arithmetic we have,

	x	x^2	x^3	$6.1x^2$	$3.2x$	Result	Relative error
Exact	4.71	22.1841	104.487111	135.32301	15.072	-14.263899	–
Chopping	4.71	22.1	104.	134.	15.0	-13.5	0.05
Rounding	4.71	22.2	105.	135.	15.1	-13.4	0.06

In both cases, large relative errors occur. To improve the calculation, we change the polynomial to the following nested form,

$$f(x) = x^3 - 6.1x^2 + 3.2x + 1.5 = ((x - 6.1)x + 3.2)x + 1.5$$

Then using chopping we have

$$f(4.71) = ((4.71 - 6.1)4.71 + 3.2)4.71 + 1.5 = -14.2$$

The relative error is

$$\left| \frac{-14.263899 + 14.2}{-14.263899} \right| \approx 0.0045$$

Using rounding we have

$$f(4.71) = ((4.71 - 6.1)4.71 + 3.2)4.71 + 1.5 = -14.3.$$

The relative error is

$$\left| \frac{-14.263899 + 14.3}{-14.263899} \right| \approx 0.0025$$

Polynomials should always be expressed in nested form before performing an evaluation, because this form minimized the number of arithmetic calculations.

– Avoid large numbers eat small numbers.

We should add those numbers with small magnitude first when we do the summation

$$fl\left(\sum_{i=1}^{n} fl(x_i)\right).$$

Example

$x_1 = 100.5$ and $x_i = 0.01$, $i = 2, \cdots, 101$, using 4-digit arithmetic, we have $fl(x_1) = 100.5$ and $fl(x_i) = 0.0100$, $i = 2, \cdots, 101$. So after doing the summation $s = fl(\sum_{i=1}^{101} fl(x_i))$, we have $s = 100.5$. If we define $x_i = 0.01$, $i = 1, \cdots, 100$ and $x_{101} = 100.5$, then we have $s = 101.5$ which is the true value of the summation.

Convergence and Stability

- **Convergence**

Definition

Suppose $\{\beta_n\}_{n=1}^{\infty}$ is a sequence known to converge to zero, and $\{\alpha_n\}_{n=1}^{\infty}$ converges to a number α. If a positive constant K exists with

$$|\alpha_n - \alpha| \leqslant K|\beta_n| \qquad \text{for large } n,$$

then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with the **rate (or, order) of convergence** $O(\beta_n)$ (big oh of β_n), and written as

$$\alpha_n = \alpha + O(\beta_n).$$

In most cases, $\beta_n = 1/n^p$, $(p > 0)$, so we have

$$\alpha_n = \alpha + O(1/n^p)$$

Similarly, if a function $F(h)$ satisfies

$$|F(h) - L| \leqslant Kh^p \qquad \text{for small positive } h,$$

we write

$$F(h) = L + O(h^p).$$

- **Stability.**

If small changes in the initial data produce correspondingly small changes in the final results, then we say that the algorithm is stable. Otherwise, the algorithm is said unstable.

Example

The solution of the recursive equation

$$p_n = \frac{10}{3} p_{n-1} - p_{n-2}, \qquad n = 2, 3, \cdots$$

is

$$p_n = c_1 \left(\frac{1}{3}\right)^n + c_2 3^n.$$

If $p_0 = 1$ and $p_1 = 1/3$, we have $c_1 = 1$ and $c_2 = 0$, then $p_n = (1/3)^n$. If we use 5 digit rounding to compute p_n, then the computed sequence $\hat{p}_0 = 1.0000$, $\hat{p}_1 = 0.33333$, which corresponding to $\hat{c}_1 = 1.0000$ and $\hat{c}_2 = -0.12500 \times 10^{-5}$, and

$$\hat{p}_n = 1.0000 \left(\frac{1}{3}\right)^n - 0.12500 \times 10^{-5} \times 3^n.$$

The round-off error is

$$p_n - \hat{p}_n = 0.12500 \times 10^{-5}(3^n)$$

This error grows exponentially, so the algorithm is unstable.

Example

The solution of the recursive equation

$$p_n = \frac{3}{4}p_{n-1} - \frac{1}{8}p_{n-2}, \qquad n = 2, 3, \cdots$$

is

$$p_n = c_1 \left(\frac{1}{2}\right)^n + c_2 \left(\frac{1}{4}\right)^n.$$

If the computed values for c_1 and c_2 are $\hat{c}_1$ and $\hat{c}_2$, due to round-off errors, then we have the round-off error in computing p_n is

$$|p_n - \hat{p}_n| = \left|(c_1 - \hat{c}_1)\left(\frac{1}{2}\right)^n + (c_2 - \hat{c}_2)\left(\frac{1}{4}\right)^n\right| \leqslant |c_1 - \hat{c}_1|\frac{1}{4} + |c_2 - \hat{c}_2|\frac{1}{16}$$

Small changes in the initial errors $|c_1 - \hat{c}_1|$ and $|c_2 - \hat{c}_2|$ will result in a small change in $|p_n - \hat{p}_n|$, the algorithm is stable.

- **Relation between convergence and stability.**

The concept of convergence is about the theoretical errors (truncation errors) between the exact value and the true value of an algorithm.

The concept of stability is about the computational errors (round-off error) between the true value of an algorithm and the computed value from the algorithm.

Consider the algorithm

$$p_n = \frac{10}{3} p_{n-1} - p_{n-2}, \qquad n = 2, 3, \cdots$$

When $p_0 = 1$ and $p_1 = 1/3$, we have $p_n = (1/3)^n$. Clearly,

$$\lim_{n \to \infty} p_n = 0.$$

However, for the computed value

$$\lim_{n \to \infty} \hat{p}_n = \infty.$$

Some basic algorithms and Matlab codes

- Sum $\sum_{i=1}^{n} x_i$

```
s = 0;        % initialize
for i = 1 : n
s = s + x(i);      % A common mistake is forget the s here!
end
```

Or we can use Matlab command **sum(x)**.

- Product $\prod_{i=1}^{n} x_i$

```
s = 0;        % initialize
for i = 1 : n
s = s * x(i);      % A common mistake is forget the s here!
end
```

Or we can use Matlab command **prod(x)**.

- Matrix-vector multiplication $y = A * x$

```
for i = 1 : n
y(i) = 0;          % initialize
for j = 1 : n
y(i) = y(i)+A(i, j) x(j);          % Use ';' to compress the outputs.
end
end
```

Or we can use Matlab command **y = A*x**.