Math 541 - Numerical Analysis

Lecture Notes – Computer Arithmetic and Finite Precision

Joseph M. Mahaffy,
⟨jmahaffy@mail.sdsu.edu⟩

Department of Mathematics and Statistics
Dynamical Systems Group
Computational Sciences Research Center
San Diego State University
San Diego, CA 92182-7720

http://jmahaffy.sdsu.edu

Spring 2018

Outline

Finite Precision — A single char

Computers use a finite number of bits (0's and 1's) to represent numbers.

For instance, an 8-bit unsigned integer (a.k.a a "char") is stored:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	1	0	0	1	1	0	1

Here, $2^6 + 2^3 + 2^2 + 2^0 = 64 + 8 + 4 + 1 = 77$, which represents the upper-case character "M" (US-ASCII).

Finite Precision — A 64-bit real number, `double`

The ***Binary Floating Point Arithmetic Standard* 754-1985** (IEEE — The Institute for Electrical and Electronics Engineers) standard specified the following layout for a 64-bit real number:

$$\mathbf{s\, c_{10}\, c_9\, \ldots\, c_1\, c_0\, m_{51}\, m_{50}\, \ldots\, m_1\, m_0}$$

where

Symbol	Bits	Description
s	1	The sign bit — 0=positive, 1=negative
c	11	The characteristic (exponent)
m	52	The mantissa

$$r = (-1)^s\, 2^{c-1023}\,(1+m), \quad c = \sum_{k=0}^{10} c_k 2^k, \quad m = \sum_{k=0}^{51} \frac{m_k}{2^{52-k}}$$

Examples: Finite Precision

$$r = (-1)^s \, 2^{c-1023} \, (1+f), \quad c = \sum_{k=0}^{10} c_k 2^k, \quad m = \sum_{k=0}^{51} \frac{m_k}{2^{52-k}}$$

Example 1: The number **3.0**

0 10000000000 1000000000000000000000000000000000000000000000000000

$$r_1 = (-1)^0 \cdot 2^{2^{10}-1023} \cdot \left(1 + \frac{1}{2}\right) = 1 \cdot 2^1 \cdot \frac{3}{2} = 3.0$$

Example 2: The **Smallest Positive Real Number**

0 00000000000 0000000000000000000000000000000000000000000000000001

$$r_2 = (-1)^0 \cdot 2^{0-1023} \cdot (1 + 2^{-52}) = (1 + 2^{-52}) \cdot 2^{-1023} \cdot 1 \approx 10^{-308}$$

Examples: Finite Precision

$$r = (-1)^s \, 2^{c-1023} \, (1+f), \quad c = \sum_{k=0}^{10} c_k 2^k, \quad m = \sum_{k=0}^{51} \frac{m_k}{2^{52-k}}$$

Example 3: The **Largest Positive Real Number**

0 11111111110 1111111111111111111111111111111111111111111111111111

$$\begin{aligned} r_3 &= (-1)^0 \cdot 2^{1023} \cdot \left(1 + \frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^{51}} + \frac{1}{2^{52}}\right) \\ &= 2^{1024} \cdot \left(2 - \frac{1}{2^{52}}\right) \approx 10^{308} \end{aligned}$$

Special Numbers

Note that the IEEE standard does NOT allow **zero!**

- There are some special signals in IEEE-754-1985:
- All zeros for c and m produce **zero**
- c having 11 bits all 1 gives either NaN (Not a Number) or $\pm\infty$
- Further reference at http://www.freesoft.org/CIE/RFC/1832/32.htm

Something is Missing — Gaps in the Representation

There are gaps in the floating-point representation!

Given the representation

0 00000000000 00000000000000000000000000000000000000000000000000**01**

for the value $\frac{2^{-1023}}{2^{52}}$.

The next larger floating-point value is

0 00000000000 00000000000000000000000000000000000000000000000000**10**

i.e. the value $\frac{2^{-1023}}{2^{51}}$.

The difference between these two values is $\frac{2^{-1023}}{2^{52}} = 2^{-1075}$,
so any number in the interval $\left(\frac{2^{-1023}}{2^{52}}, \frac{2^{-1023}}{2^{51}}\right)$ **is not representable!**

Something is Missing — Gaps in the Representation

A gap of 2^{-1075} doesn't seem too bad...

However, the size of the gap depends on the value itself...

Consider $r = 3.0$

0 10000000000 1000000000000000000000000000000000000000000000000000

and the next value

0 10000000000 1000000000000000000000000000000000000000000000000001

The difference is $\dfrac{2}{2^{52}} \approx 4.4 \cdot 10^{-16}$

Something is Missing — Gaps in the Representation

At the other extreme, the difference between

0 11111111110 1111111111111111111111111111111111111111111111111111

and the previous value

0 11111111110 1111111111111111111111111111111111111111111111111110

is $\dfrac{2^{1023}}{2^{52}} = 2^{971} \approx 1.99 \cdot 10^{292}$.

That's a "fairly significant" gap!!!

The number of atoms in the observable universe can be estimated to be no more than $\sim 10^{80}$***.***

The Relative Gap

It makes more sense to factor the exponent out of the discussion and talk about the relative gap:

Exponent	Gap	Relative Gap (Gap/Exponent)
2^{-1023}	2^{-1075}	2^{-52}
2^{1}	2^{-51}	2^{-52}
2^{1023}	2^{971}	2^{-52}

Any difference between numbers smaller than the local gap is not representable, *e.g.* any number in the interval

$$\left[3.0,\ 3.0 + \tfrac{1}{2^{51}}\right)$$

is represented by the value 3.0.

The **MatLab** command `eps` (for epsilon tolerance) gives ***double precision***, which is

$$2^{-52} \approx 2.2204 \times 10^{-16}$$

The Floating Point "Numbers"

Floating point "numbers" represent intervals!

Since (most) humans find it hard to think in binary representation, from now on we will **for simplicity** and **without loss of generality** assume that floating point numbers are represented in the normalized floating point form as...

k-digit decimal machine numbers

$$\pm 0.d_1 d_2 \cdots d_{k-1} d_k \cdot 10^n$$

where

$$1 \leq d_1 \leq 9, \quad 0 \leq d_i \leq 9, \; i \geq 2, \quad n \in \mathbb{Z}$$

k-Digit Decimal Machine Numbers

Any real number can be written in the form

$$r = \pm 0.d_1 d_2 \cdots d_\infty \cdot 10^n$$

given infinite patience and storage space.

We can obtain the floating-point representation `fl(r)` in two ways:

1. Truncating (chopping) — just keep the first k digits (In **MatLab** use `floor(r)`)
2. Rounding — if $d_{k+1} \geq 5$ then add 1 to d_k. Truncate. (Standard for most languages)

Examples

$$\texttt{fl}_{t,5}(\pi) = 0.31415 \cdot 10^1, \quad \texttt{fl}_{r,5}(\pi) = 0.31416 \cdot 10^1$$

In both cases, the error introduced is called the **roundoff error**.

Quantifying the Error

Let p^* be and approximation to p, then...

Definition (The Absolute Error)

$$|p - p^*|$$

Definition (The Relative Error)

$$\frac{|p - p^*|}{|p|}, \quad p \neq 0$$

Definition (Significant Digits)

The number of **significant digits** is the largest value of t for which

$$\frac{|p - p^*|}{|p|} < 5 \cdot 10^{-t}$$

Sources of Numerical Error *Important!!!*

Some Sources of Numerical Error

1. **Representation — Roundoff**
2. **Cancellation**

Consider:

$$\begin{array}{rl} & 0.12345678012345 \cdot 10^{1} \\ - & 0.12345678012344 \cdot 10^{1} \\ \hline = & 0.10000000000000 \cdot 10^{-13} \end{array}$$

this value has (at most) **1** significant digit!!!

If you assume a "cancelled value" has more significant bits (the computer will happily give you some numbers) — Any use of these random digits is **GARBAGE**!!!

Examples: 5-digit Arithmetic

Rounding 5-digit arithmetic

$$\begin{gathered}(0.96384 \cdot 10^5 + 0.26678 \cdot 10^2) - 0.96410 \cdot 10^5 = \\ (0.96384 \cdot 10^5 + 0.00027 \cdot 10^5) - 0.96410 \cdot 10^5 = \\ 0.96411 \cdot 10^5 - 0.96410 \cdot 10^5 = 0.10000 \cdot 10^1\end{gathered}$$

Truncating 5-digit arithmetic

$$\begin{gathered}(0.96384 \cdot 10^5 + 0.26678 \cdot 10^2) - 0.96410 \cdot 10^5 = \\ (0.96384 \cdot 10^5 + 0.00026 \cdot 10^5) - 0.96410 \cdot 10^5 = \\ 0.96410 \cdot 10^5 - 0.96410 \cdot 10^5 = 0.0000 \cdot 10^0\end{gathered}$$

Rearrangement changes the result:

$$\begin{gathered}(0.96384 \cdot 10^5 - 0.96410 \cdot 10^5) + 0.26678 \cdot 10^2 = \\ -0.26000 \cdot 10^2 + 0.26678 \cdot 10^2 = 0.67800 \cdot 10^0\end{gathered}$$

Numerically, order of computation matters! (This is a **HARD** problem)

Example: Loss of Significant Digits due to Subtractive Cancellation

Consider the recursive relation

$$x_{n+1} = 1 - (n+1)x_n \quad \text{with} \quad x_0 = 1 - \frac{1}{e}.$$

This sequence can be shown to converge to **0**

Subtractive cancellation produces an error, which is approximately equal to the machine precision times $n!$.

Example: Proof of Convergence to 0

The ***recursive relation*** is

$$x_{n+1} = 1 - (n+1)x_n$$

with

$$x_0 = 1 - \frac{1}{e} = 1 - \frac{1}{2!} + \frac{1}{3!} - \frac{1}{4!} + ...$$

From the recursive relation

$$\begin{aligned}
x_1 &= 1 - x_0 = \frac{1}{2!} - \frac{1}{3!} + \frac{1}{4!} - ... \\
x_2 &= 1 - 2x_1 = \frac{1}{3} - \frac{2}{4!} + \frac{2}{5!} - ... \\
x_3 &= 1 - 3x_2 = \frac{3!}{4!} - \frac{3!}{5!} + \frac{3!}{6!} - ... \\
&\vdots \\
x_n &= 1 - nx_{n-1} = \frac{n!}{(n+1)!} - \frac{n!}{(n+2)!} + \frac{n!}{(n+3)!} - ...
\end{aligned}$$

This shows that

$$x_n = \frac{1}{n+1} - \frac{1}{(n+1)(n+2)} + ... \to 0 \quad \text{as} \quad n \to \infty.$$

Subtraction Error

The ***recursive relation*** $x_{n+1} = 1 - (n+1)x_n$ with $x_0 = 1 - \frac{1}{e}$

```
clear
x(1) = 1-1/exp(1);
s(1) = 1;
f(1) = 1;
for i = 2:21
x(i) = 1-(i-1)*x(i-1);
s(i) = 1/i;
f(i) = (i-1)*f(i-1);
end
n = 0:20;
z = [n; x; s; f];
fprintf(1, '\n\n n x(n) 1/(n+1) n!\n\n')
fprintf(1, '%2.0f %13.8f %10.8f %10.3g\n',z)
```

Subtractive Cancellation Example: Output

n	x_n	$n!$	n	x_n	$n!$
0	0.63212056	1	11	0.07735223	3.99e+007
1	0.36787944	1	12	0.07177325	4.79e+008
2	0.26424112	2	13	0.06694778	6.23e+009
3	0.20727665	6	14	0.06273108	8.72e+010
4	0.17089341	24	15	0.05903379	1.31e+012
5	0.14553294	120	16	0.05545930	2.09e+013
6	0.12680236	720	17	0.05719187	3.56e+014
7	0.11238350	5.04e+003	18	−0.02945367	6.4e+015
8	0.10093197	4.03e+004	19	1.55961974	1.22e+017
9	0.09161229	3.63e+005	20	−30.19239489	2.43e+018
10	0.08387707	3.63e+006			

Subtraction Error

Consider the **MatLab** computation near $x = 1$ of

$$y = x^7 - 7\,x^6 + 21\,x^5 - 35\,x^4 + 35\,x^3 - 21\,x^2 + 7\,x - 1$$

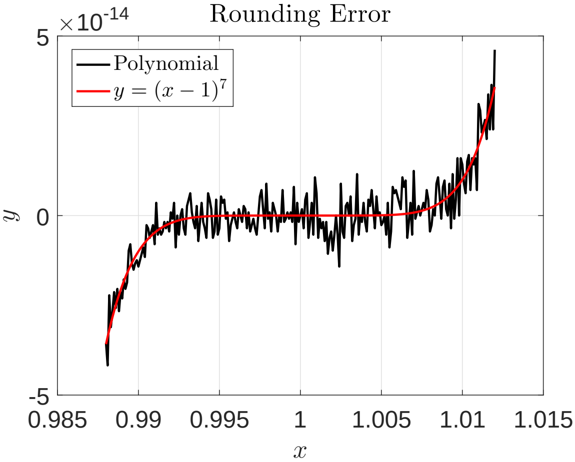
$$\text{compared to} \qquad y = (x-1)^7$$

```
% Rounding Error Graph

x = 0.988:0.0001:1.012;
y = x.^7 - 7*x.^6 + 21*x.^5 - 35*x.^4 + ...
    35*x.^3 - 21*x.^2 + 7*x - 1;
yy = (x - 1).^7;

plot(x,y,'k-','linewidth',1.5);
hold on
plot(x,yy,'r-','linewidth',1.5);
grid
```

Subtraction Error

The program graphs $x \in [0.988, 1.012]$ with the two forms of function:

$$y = x^7 - 7\,x^6 + 21\,x^5 - 35\,x^4 + 35\,x^3 - 21\,x^2 + 7\,x - 1 = (x-1)^7$$



Algorithms

Definition (Algorithm)

An **algorithm** is a procedure that describes, in an ***unambiguous manner***, a finite sequence of steps to be performed in a specific order.

In this class, the objective of an algorithm is to implement a procedure to solve a problem or approximate a solution to a problem.

There are many collection of algorithms “out there” called ***Numerical Recipes***

Key Concepts for Numerical Algorithms Stability

Definition (Stability)

An algorithm is said to be stable if small changes in the input, generates small changes in the output.

- At some point we need to quantify what "small" means!
- If an algorithm is stable for a certain ***range*** of initial data, then is it said to be ***conditionally stable***.
- Stability issues are discussed in great detail in ***Math 543*** and our **Dynamical Systems** classes.

Key Concepts for Numerical Algorithms — Error Growth

Suppose $E_0 > 0$ denotes the initial error, and E_n represents the error after n operations.

- If $E_n \approx \mathcal{C}E_0 \cdot n$ (for a constant $\mathcal{C}$, which is independent of n), then the growth is ***linear***.
- If $E_n \approx \mathcal{C}^n E_0$, $\mathcal{C} > 1$, then the growth is ***exponential*** — in this case the error will dominate very fast(undesirable scenario).
- **Linear error growth** is usually unavoidable, and in the case where $\mathcal{C}$ and E_0 are small the results are generally acceptable. — **Stable algorithm**.
- **Exponential error growth** is unacceptable. Regardless of the size of E_0 the error grows rapidly. — **Unstable algorithm**.
- One property of **chaos** in a dynamical system is the ***exponential growth*** of any error in initial conditions – leading to **unpredictable behavior**

Example

The ***recursive equation***

$$p_n = \frac{10}{3}p_{n-1} - p_{n-2}, \quad n = 2, 3, \dots, \infty$$

has the **exact solution**

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

for any constants c_1 and c_2. (Determined by starting values.)

In particular, if $p_0 = 1$ and $p_1 = \frac{1}{3}$, we get $c_1 = 1$ and $c_2 = 0$, so $p_n = \left(\frac{1}{3}\right)^n$ for all n.

What happens with some rounding error, as we don't know $\frac{1}{3}$ exactly?

Example

Consider what happens in ***5-digit rounding arithmetic***, where the initial starting conditions are rounded.

$$p_0^* = 1.0000, \quad p_1^* = 0.33333$$

which modifies the constants (by solving the general solution for c_1 and c_2 with the p_0^* and p_1^*)

$$c_1^* = 1.0000, \quad c_2^* = -0.12500 \cdot 10^{-5}$$

The generated sequence is

$$p_n^* = 1.0000\,(0.33333)^n - \underbrace{0.12500 \cdot 10^{-5}(3.0000)^n}_{\textbf{Exponential Growth}}$$

p_n^* quickly becomes a very poor approximation to p_n due to the *exponential growth* of the initial roundoff error.

Reducing the Effects of Roundoff Error

- The effects of roundoff error can be reduced by using higher-order-digit arithmetic such as the **double** or **multiple-precision arithmetic** available on most computers.
- Disadvantages in using **double precision arithmetic** are that it takes more computation time, and ***the growth of the roundoff error is not eliminated but only postponed.***
- Sometimes, but not always, it is possible to reduce the growth of the roundoff error by restructuring the calculations.

Key Concepts

Rate of Convergence

Definition (Rate of Convergence)

Suppose the sequence $\underline{\beta} = \{\beta_n\}_{n=1}^{\infty}$ converges to zero, and $\underline{\alpha} = \{\alpha_n\}_{n=1}^{\infty}$ converges to a number α.

If there exists $K > 0$: $|\alpha_n - \alpha| < K\beta_n$, for n large enough, then we say that $\{\alpha_n\}_{n=1}^{\infty}$ converges to α with a **Rate of Convergence** $\mathcal{O}(\beta_n)$ ("Big Oh of β_n").

We write

$$\alpha_n = \alpha + \mathcal{O}(\beta_n)$$

Note: The sequence $\underline{\beta} = \{\beta_n\}_{n=1}^{\infty}$ is usually chosen to be

$$\beta_n = \frac{1}{n^p}$$

for some positive value of p.

Examples: Rate of Convergence

Example 1:
If

$$\alpha_n = \alpha + \frac{1}{\sqrt{n}}$$

then for any $\varepsilon > 0$

$$|\alpha_n - \alpha| = \frac{1}{\sqrt{n}} \leq \underbrace{(1+\varepsilon)}_{K} \underbrace{\frac{1}{\sqrt{n}}}_{\beta_n}$$

Hence,

$$\alpha_n = \alpha + \mathcal{O}\left(\frac{1}{\sqrt{n}}\right)$$

Examples: Rate of Convergence

Example 2: Consider the sequence (as $n \to \infty$)

$$\alpha_n = \sin\left(\frac{1}{n}\right) - \frac{1}{n}$$

The **Maclaurin series** expansion for $\sin(x)$ is:

$$\sin\left(\frac{1}{n}\right) \sim \frac{1}{n} - \frac{1}{6n^3} + \mathcal{O}\left(\frac{1}{n^5}\right)$$

Hence

$$|\alpha_n| = \left|\frac{1}{6n^3} + \mathcal{O}\left(\frac{1}{n^5}\right)\right|$$

It follows that

$$\alpha_n = \mathbf{0} + \mathcal{O}\left(\frac{1}{n^3}\right)$$

Note:

$$\mathcal{O}\left(\frac{1}{n^3}\right) + \mathcal{O}\left(\frac{1}{n^5}\right) = \mathcal{O}\left(\frac{1}{n^3}\right), \quad \text{since} \quad \frac{1}{n^5} \ll \frac{1}{n^3}, \quad \text{as} \quad n \to \infty$$

Generalizing to Continuous Limits

Definition (Rate of Convergence)

Suppose

$$\lim_{h\to 0} G(h) = 0, \quad \text{and} \quad \lim_{h\to 0} F(h) = L$$

If there exists $K > 0$:

$$|F(h) - L| \leq K\,|G(h)|$$

for all $h < H$ (for some $H > 0$), then

$$F(h) = L + \mathcal{O}(G(h))$$

we say that $F(h)$ converges to L with a **Rate of Convergence** $\mathcal{O}(G(h))$.

Usually $G(h) = h^p$, $p > 0$.

Examples: Rate of Convergence

Example 2-b: Consider the function $\alpha(h)$ (as $h \to 0$)

$$\alpha(h) = \sin(h) - h$$

The **Maclaurin series** expansion for $\sin(x)$ is:

$$\sin(h) \sim h - \frac{h^3}{6} + \mathcal{O}\left(h^5\right)$$

Hence

$$|\alpha(h)| = \left|\frac{h^3}{6} + \mathcal{O}\left(h^5\right)\right|$$

It follows that

$$\lim_{h\to 0} \alpha(h) = \mathbf{0} + \mathcal{O}\left(h^3\right)$$

Note:

$$\mathcal{O}\left(h^3\right) + \mathcal{O}\left(h^5\right) = \mathcal{O}\left(h^3\right), \quad \text{since} \quad h^5 \ll h^3, \quad \text{as} \quad h \to 0$$