FLOATING POINT ARITHMETHIC - ERROR ANALYSIS

- **Brief review of floating point arithmetic**
- **Model of floating point arithmetic**
- **Notation, backward and forward errors**
- **Read: Section 2.7 of text.**

Roundoff errors and floating-point arithmetic

➤ The basic problem: The set A of all possible representable numbers on a given machine is finite - but we would like to use this set to perform standard arithmetic operations (+,*,-,/) on an infinite set. The usual algebra rules are no longer satisfied since results of operations are rounded.

➤ Basic algebra breaks down in floating point arithmetic.

Example: In floating point arithmetic.

$$a + (b + c) \; != \; (a + b) + c$$

✍ Matlab experiment: For 10,000 random numbers find number of instances when the above is true. Same thing for the multiplication..

Machine precision - machine epsilon

➤ When a number x is very small, there is a point when $1 + x == 1$ in a machine sense. The computer no longer makes a difference between 1 and $1 + x$.

Definition: the machine epsilon is the smallest number ϵ such that

$$fl(1 + \epsilon) \neq 1$$

This number is denoted by $\underline{\mathrm{u}}$ – sometimes by eps.

✍ Matlab experiment: find the machine epsilon on your computer.

➤ Many discussions on what conditions/ rules should be satisfied by floating point arithmetic. The IEEE standard is a set of standards adopted by many CPU manufacturers.

Rule 1.

$$fl(x) = x(1+\epsilon), \quad \textbf{where} \quad |\epsilon| \le \underline{\mathbf{u}}$$

Rule 2. For all operations $\odot$ (one of $+, -, *, /$)

$$fl(x \odot y) = (x \odot y)(1+\epsilon_\odot), \quad \textbf{where} \quad |\epsilon_\odot| \le \underline{\mathbf{u}}$$

Rule 3. For $+, *$ operations

$$fl(a \odot b) = fl(b \odot a)$$

✍ Matlab experiment: Verify experimentally Rule 3 with 10,000 randomly generated numbers a_i, b_i.

Example: **Consider the sum of 3 numbers: $y = a+b+c$.**

- **Done as $fl(fl(a+b)+c)$**

$$\begin{aligned}
\eta &= fl(a+b) = (a+b)(1+\epsilon_1) \\
y_1 &= fl(\eta+c) = (\eta+c)(1+\epsilon_2) \\
&= \left[(a+b)(1+\epsilon_1)+c\right](1+\epsilon_2) \\
&= \left[(a+b+c)+(a+b)\epsilon_1)\right](1+\epsilon_2) \\
&= (a+b+c)\left[1+\frac{a+b}{a+b+c}\epsilon_1(1+\epsilon_2)+\epsilon_2\right]
\end{aligned}$$

So disregarding the high order term $\epsilon_1\epsilon_2$

$$\begin{aligned}
fl(fl(a+b)+c) &= (a+b+c)(1+\epsilon_3) \\
\epsilon_3 &\approx \frac{a+b}{a+b+c}\epsilon_1+\epsilon_2
\end{aligned}$$

➤ If we redid the computation as $y_2 = fl(a + fl(b + c))$ we would find

$$\begin{aligned} fl(a + fl(b + c)) &= (a + b + c)(1 + \epsilon_4) \\ \epsilon_4 &\approx \frac{b + c}{a + b + c}\epsilon_1 + \epsilon_2 \end{aligned}$$

➤ The first error is amplified by the factor $(a + b)/y$ in the first case and $(b + c)/y$ in the second case.

➤ In order to sum n numbers more accurately, it is better to start with the small numbers first. [However, sorting before adding is not worth the cost!]

➤ But watch out if the numbers have mixed signs!

The absolute value notation

- For a given vector x, $|x|$ is the vector with components $|x_i|$, i.e., $|x|$ is the component-wise absolute value of x.
- Similarly for matrices:

$$|A| = \{|a_{ij}|\}_{i=1,\ldots,m;\ j=1,\ldots,n}$$

- Obvious result. The basic inequality

$$|fl(a_{ij}) - a_{ij}| \le \underline{\mathbf{u}}\ |a_{ij}|$$

translates into

$$fl(A) = A + E \qquad \text{with} \qquad |E| \le \underline{\mathbf{u}}\ |A|$$

- $A \le B$ means $a_{ij} \le b_{ij}$ for all $1 \le i \le m;\ 1 \le j \le n$

Error Analysis: Inner product

- Inner products are in the innermost parts of many calculations. Their analysis is important.

Lemma: If $|\delta_i| \leq \underline{\mathbf{u}}$ and $n\underline{\mathbf{u}} < 1$ then

$$\Pi_{i=1}^{n}(1+\delta_i) = 1 + \theta_n \quad \text{where} \quad |\theta_n| \leq \frac{n\underline{\mathbf{u}}}{1 - n\underline{\mathbf{u}}}$$

- Common notation $\gamma_n \equiv \frac{n\underline{\mathbf{u}}}{1-n\underline{\mathbf{u}}}$

Main result on inner products:

$$|fl(x^T y) - x^T y| \leq \gamma_n \, |x|^T \, |y|$$

- Absolute value notation used

- When $\gamma_n \leq 1.01 n \underline{\mathbf{u}}$ then

$$|fl(x^T y) - x^T y| \leq 1.01 \; n \; \underline{\mathbf{u}} \; |x|^T \; |y|$$

- $\gamma_n \leq 1.01 n \underline{\mathbf{u}}$ means $[1/(1 - n\underline{\mathbf{u}})] \leq 1.01$

- For $\underline{\mathbf{u}} = 2.0 \times 10^{-16}$, assumption $\gamma_n \leq 1.01 n \underline{\mathbf{u}}$ holds for $n \leq 4.46 \times 10^{13}$.

- Consequence of lemma:

$$|fl(A * B) - A * B| \leq \gamma_n \; |A| * |B|$$

- Another way to write the result (less precise) is

$$|fl(x^T y) - x^T y| \leq \; n \; \underline{\mathbf{u}} \; |x|^T \; |y| + O(\underline{\mathbf{u}}^2)$$

✍ Prove the lemma [Hint: use induction]

✍ Assume you use single precision for which you have $\underline{\mathbf{u}} = 2. \times 10^{-6}$. What is the largest n for which $\gamma_n \le 1.01 n \underline{\mathbf{u}}$ holds? Any conclusions for the use of single precision arithmetic?

✍ What does the main result on inner products imply for the case when $y = x$? [Contrast the relative accuracy you get in this case vs. the general case when $y \neq x$]

Backward and forward errors

➤ Assume the approximation $\hat{y}$ to $y = f(x)$ is computed with arithmetic precision ϵ. Possible analysis: find an upper bound for the Forward error

$$|\Delta y| = |y - \hat{y}|$$

➤ This is not always easy.

Alternative question: find the equivalent perturbation on the initial data (x) which produces the result $\hat{y}$. In other words, for what Δx do we have:

$$f(x + \Delta x) = \hat{y}$$

➤ The value of $|\Delta x|$ is called the backward error. An analysis to find an upper bound for $|\Delta x|$ is called Backward error analysis.

Example:

$$A = \begin{pmatrix} a & b \\ 0 & c \end{pmatrix} \quad B = \begin{pmatrix} d & e \\ 0 & f \end{pmatrix}$$

Consider the product: $fl(A.B) =$

$$\begin{bmatrix} (ad)(1+\epsilon_1) & [ae(1+\epsilon_2) + bf(1+\epsilon_3)]\,(1+\epsilon_4) \\ 0 & cf(1+\epsilon_5) \end{bmatrix}$$

with $\epsilon_i \leq \underline{\mathbf{u}}$, for $i = 1, ..., 5$. Result can be written as:

$$\begin{bmatrix} a & b(1+\epsilon_3)(1+\epsilon_4) \\ 0 & c(1+\epsilon_5) \end{bmatrix} \begin{bmatrix} d(1+\epsilon_1) & e(1+\epsilon_2)(1+\epsilon_4) \\ 0 & f \end{bmatrix}$$

- So $fl(A.B) = (A + E_A)(B + E_B)$.
- Backward errors E_A, E_B satisfy:

$$|E_A| \leq 2\underline{\mathbf{u}}\,|A| + O(\underline{\mathbf{u}}^{\,2})\ ; \qquad |E_B| \leq 2\underline{\mathbf{u}}\,|B| + O(\underline{\mathbf{u}}^{\,2})$$

➤ When solving $Ax = b$ by Gaussian Elimination, we will see that a bound on $\|e_x\|$ such that this holds exactly

$$A(x_{\text{computed}} + e_x) = b$$

is much harder to find than bounds on $\|E_A\|$, $\|e_b\|$ such that this holds exactly

$$(A + E_A)x_{\text{computed}} = (b + e_b).$$

Note: In many instances backward errors are more meaningful than forward errors: if initial data is accurate only to 4 digits for example, then my algorithm for computing x need not have 10 digits of accuracy. A backward error of order 10^{-4} is acceptable.

✍ Show for any x, y, there exist $\Delta x, \Delta y$ such that

$$fl(x^T y) = (x + \Delta x)^T y, \quad \text{with} \quad |\Delta x| \le \gamma_n |x|$$
$$fl(x^T y) = x^T (y + \Delta y), \quad \text{with} \quad |\Delta y| \le \gamma_n |y|$$

✍ (Continuation) Let A an $m \times n$ matrix, x an n-vector, and $y = Ax$. Show that there exist a matrix ΔA such

$$fl(y) = (A + \Delta A)x, \quad \text{with} \quad |\Delta A| \le \gamma_n |A|$$

✍ (Continuation) From the above derive a result about a column of the product of two matrices A and B. Does a similar result hold for the product AB as a whole?

Supplemental notes: Floating Point Arithmetic

In most computing systems, real numbers are represented in two parts: A mantissa and an exponent. If the representation is in the base β then:

$$\boldsymbol{x = \pm(.d_1 d_2 \cdots d_m)_\beta \beta^e}$$

- $.d_1 d_2 \cdots d_m$ is a fraction in the base-β representation
- e is an integer - can be negative, positive or zero.
- Generally the form is normalized in that $d_1 \neq 0$.

Example: In base 10 (for illustration)

1. 1000.12345 can be written as

$$0.100012345_{10} \times 10^4$$

2. 0.000812345 can be written as

$$0.812345_{10} \times 10^{-3}$$

➤ Problem with floating point arithmetic: we have to live with limited precision.

Example: Assume that we have only 5 digits of accuray in the mantissa and 2 digits for the exponent (excluding sign).

$.d_1$	d_2	d_3	d_4	d_5	e_1	e_2

Let us try to add 1000.2 and 1.07

$1000.2 = \boxed{.1}\boxed{0}\boxed{0}\boxed{0}\boxed{2}\boxed{0}\boxed{4}$; $\quad 1.07 = \boxed{.1}\boxed{0}\boxed{7}\boxed{0}\boxed{0}\boxed{0}\boxed{1}$

First task: align decimal points. The one with smallest exponent will be (internally) rewritten so its exponent matches the largest one:

$$1.07 = 0.000107 \times 10^4$$

Second task: add mantissas:

$$\begin{array}{rl} & 0.\ 1\ 0\ 0\ 0\ 2 \\ + & 0.\ 0\ 0\ 0\ 1\ 0\ 7 \\ \hline = & 0.\ 1\ 0\ 0\ 1\ 2\ 7 \end{array}$$

Third task: round result. Result has 6 digits - can use only 5 so we can

- Chop result: | .1 | 0 | 0 | 1 | 2 | ;
- Round result: | .1 | 0 | 0 | 1 | 3 | ;

Fourth task: Normalize result if needed (not needed here)

result with rounding: | .1 | 0 | 0 | 1 | 3 | 0 | 4 | ;

✍ Redo the same thing with $1000.2 + 3000.8$

The IEEE standard

32 bit (Single precision) :

$\pm$	$\leftarrow$ 23 bits $\rightarrow$	8bits
	mantissa	exponent

➤ In binary: The leading one in mantissa does not need to be represented. One bit gained. ➤ Hidden bit.

➤ Largest exponent: $2^7 - 1 = 127$; Smallest: $= -126$. ['bias' of 127]

$\boxed{\text{64 bit}}$ (Double precision) :

$$\underbrace{\pm \mid \leftarrow \quad \text{52 bits} \quad \rightarrow}_{\text{mantissa}} \mid \underbrace{\text{11 bits}}_{\text{exponent}}$$

- Bias of 1023 so if c is stored exponend actual exponent is 2^{c-1023}
- $e + bias = 2047$ (all ones) = special use
- Largest exponent: 1023; Smallest = -1022.
- With hidden bit: mantissa has 53 bits represented.

✍ **Take the number 1.0 and see what will happen if you add $1/2, 1/4, ..., 2^{-i}$. Do not forget the hidden bit!**

Hidden bit

$\downarrow$ $\leftarrow$ **52 bits** $\rightarrow$

1	1	0	0	0	0	0	0	0	0	0	0	e	e
1	0	1	0	0	0	0	0	0	0	0	0	e	e
1	0	0	1	0	0	0	0	0	0	0	0	e	e

....

1	0	0	0	0	0	0	0	0	0	0	1	e	e
1	0	0	0	0	0	0	0	0	0	0	0	e	e

➤ **Conclusion**

$fl(1 + 2^{-52}) \neq 1$ **but:** $fl(1 + 2^{-53}) == 1$ **!!**