FLOATING POINT ARITHMETHIC - ERROR ANALYSIS

- Brief review of floating point arithmetic

- Model of floating point arithmetic

- Notation, backward and forward errors

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.

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

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

Floating point representation:

Real numbers are represented in two parts: A mantissa (significand) and an exponent. If the representation is in the base β then:

$$x = \pm(.d_1 d_2 \cdots d_t)\beta^e$$

- $.d_1 d_2 \cdots d_t$ is a fraction in the base-β representation (Generally the form is normalized in that $d_1 \neq 0$), and e is an integer
- Often, more convenient to rewrite the above as:

$$x = \pm(m/\beta^t) \times \beta^e \equiv \pm m \times \beta^{e-t}$$

- Mantissa m is an integer with $0 \leq m \leq \beta^t - 1$.

Machine precision - machine epsilon

- Notation : $fl(x)$ = closest floating point representation of real number x ('rounding')

- 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 $1 + x$.

Machine epsilon: The smallest number ϵ such that $1+\epsilon$ is a float that is different from one, is called machine epsilon. Denoted by $\mathbf{macheps}$ or $\mathbf{eps}$, it represents the distance from 1 to the next larger floating point number.

- With previous representation, $\mathbf{eps}$ is equal to $\beta^{-(t-1)}$.

Example: In IEEE standard double precision, $\beta = 2$, and $t = 53$ (includes 'hidden bit'). Therefore $\mathbf{eps} = 2^{-52}$.

Unit Round-off A real number x can be approximated by a floating number $fl(x)$ with relative error no larger than $\underline{\mathbf{u}} = \frac{1}{2}\beta^{-(t-1)}$.

- $\underline{\mathbf{u}}$ is called Unit Round-off.
- In fact can easily show:

$$fl(x) = x(1 + \delta) \text{ with } |\delta| < \underline{\mathbf{u}}$$

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

- What conditions/ rules should be satisfied by floating point arithmetic? The IEEE standard is a set of standards adopted by many CPU manufacturers.

Among IEEE rules:

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

Rule 2. $$fl(x \odot y) = (x \odot y)(1+\epsilon_\odot), \text{ where } |\epsilon_\odot| \le \underline{\mathbf{u}}$$ for $\odot = +, -, *, /$

Rule 3. For $+, *$ operations: $$fl(a \odot b) = fl(b \odot a)$$

✍3 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 $fl(a + b + c) = fl(fl(a + b) + c)$

$$\begin{aligned} fl(a+b) &= (a+b)(1+\epsilon_1) \\ fl(a+b+c) &= \left[(a+b)(1+\epsilon_1)+c\right](1+\epsilon_2) \\ &= a(1+\epsilon_1)(1+\epsilon_2) + b(1+\epsilon_1)(1+\epsilon_2) + c(1+\epsilon_2) \\ &= a(1+\theta_1) + b(1+\theta_2) + c(1+\theta_3) \end{aligned}$$

with $1 + \theta_1 = 1 + \theta_2 = (1 + \epsilon_1)(1 + \epsilon_2)$ and $1 + \theta_3 = (1 + \epsilon_2)$

➤ For a longer sum we would have something like:

$$1 + \theta_j = (1 + \epsilon_1)(1 + \epsilon_2)(\cdots)(1 + \epsilon_{n-j})$$

We will study such products shortly

➤ Remark on order of the sum. If $y_1 = fl(fl(a+b)+c)$:

$$\begin{aligned} y1 &= [(a+b+c)+(a+b)\epsilon_1)]\,(1+\epsilon_2) \\ &= (a+b+c)\left[1+\frac{a+b}{a+b+c}\epsilon_1(1+\epsilon_2)+\epsilon_2\right] \end{aligned}$$

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 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 accurately, it is better to start with small numbers first. [However, sorting before adding is not worth it.]

➤ 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: $$\boxed{|A| = \{|a_{ij}|\}_{i=1,...,m;\ j=1,...,n}}$$

➤ An obvious result: The basic inequality

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

translates into $$\boxed{|fl(A) - A| \leq \underline{\mathbf{u}}\ |A|}$$

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

Backward and forward errors

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

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

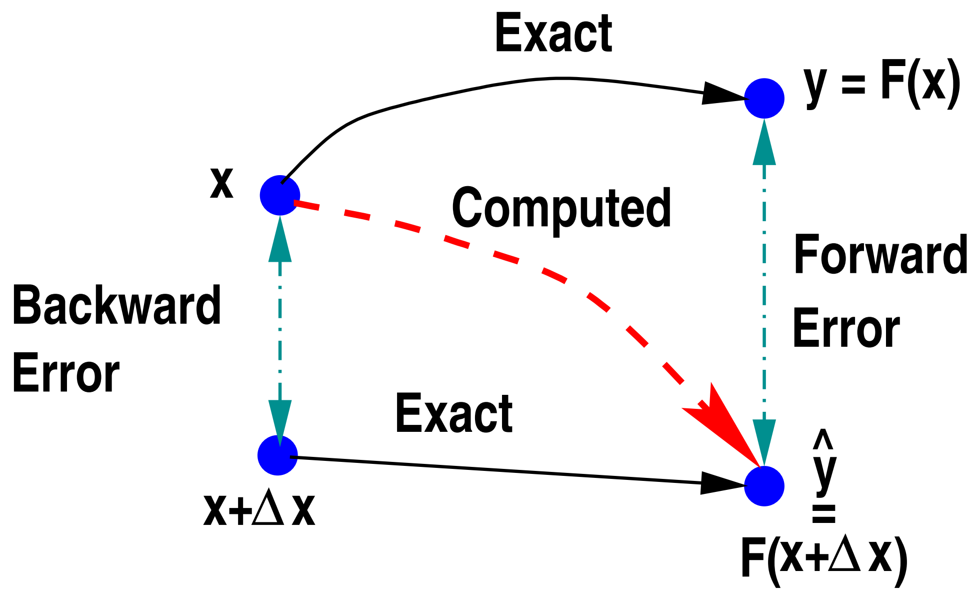
➤ Called Forward error analysis. This is not always easy.

Alternative question:

Find smallest equivalent perturbation on initial data (x) that produces (exactly) the result $\hat{y}$:

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

➤ The smallest value of $\|\Delta x\|$ s.t. above is satisfied is called the backward error. An analysis to find this eror is called Backward error analysis.



Formal definition $\boxed{\eta(\hat{y}) = \min\{\epsilon | \hat{y} = F(x + \Delta x) \qquad \|\Delta x\| \leq \epsilon\}}$

Note: In practice backward errors may be more meaningful than forward errors: if initial data is accurate only to 4 digits say, then my algorithm for computing x need not be required to produce a backward error of less then 10^{-10} for example. A backward error of order 10^{-4} is sufficient.

Error Analysis: Inner product

➤ The following lemma helps with analysis of inner products.

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

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

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

✍4 Prove the lemma [Hint: use induction]

Example: Previous sum of numbers can be written

$$\begin{aligned} fl(a+b+c) &= fl(fl(a+b)+c) \\ &= [(a+b)(1+\epsilon_1)+c]\,(1+\epsilon_2) \\ &= a(1+\epsilon_1)(1+\epsilon_2)+b(1+\epsilon_1)(1+\epsilon_2)+c(1+\epsilon_2) \\ &= a(1+\theta_1)+b(1+\theta_2)+c(1+\theta_3) \\ &= \text{exact sum of slightly perturbed inputs,} \end{aligned}$$

where all θ_i's satisfy $|\theta_i| \le \gamma_n$ (here $n = 2$)

- Backward error result (output is exact sum of perturbed input)
- Alternatively, can write 'forward' bound:
  $|fl(a+b+c)-(a+b+c)| \le |a\theta_1|+|b\theta_2|+|c\theta_3|$.

(bound on | output - exact sum |)

Analysis of inner products (cont.)

Consider $$s_n = fl(x_1 * y_1 + x_2 * y_2 + \cdots + x_n * y_n)$$

- In what follows η_i's come from $*$, ϵ_i's come from $+$
- They satisfy: $|\eta_i| \leq \underline{\mathbf{u}}$ and $|\epsilon_i| \leq \underline{\mathbf{u}}$.
- The inner product s_n is computed as:

1. $s_1 = fl(x_1y_1) = (x_1y_1)(1 + \eta_1)$
2. $s_2 = fl(s_1 + fl(x_2y_2)) = fl(s_1 + x_2y_2(1 + \eta_2))$
   $= (x_1y_1(1 + \eta_1) + x_2y_2(1 + \eta_2))\,(1 + \epsilon_2)$
   $= x_1y_1(1 + \eta_1)(1 + \epsilon_2) + x_2y_2(1 + \eta_2)(1 + \epsilon_2)$
3. $s_3 = fl(s_2 + fl(x_3y_3)) = fl(s_2 + x_3y_3(1 + \eta_3))$
   $= (s_2 + x_3y_3(1 + \eta_3))(1 + \epsilon_3)$

Expand:
$$
\begin{aligned}
s_3 = \; & x_1 y_1 (1+\eta_1)(1+\epsilon_2)(1+\epsilon_3) \\
& + x_2 y_2 (1+\eta_2)(1+\epsilon_2)(1+\epsilon_3) \\
& + x_3 y_3 (1+\eta_3)(1+\epsilon_3)
\end{aligned}
$$

➤ Induction would show that [with convention that $\epsilon_1 \equiv 0$]

$$
s_n = \sum_{i=1}^{n} x_i y_i (1+\eta_i) \prod_{j=i}^{n} (1+\epsilon_j)
$$

Q: How many terms in the coefficient of $x_i y_i$ do we have?

A:
- When $i > 1 : 1 + (n - i + 1) = n - i + 2$
- When $i = 1 : n$ (since $\epsilon_1 = 0$ does not count)

➤ Bottom line: always $\leq n$.

- For each of these products

$$(1+\eta_i)\ \prod_{j=i}^{n}(1+\epsilon_j) = 1+\theta_i, \quad \text{with} \quad |\theta_i| \le \gamma_n \quad \text{so:}$$

$$s_n = \sum_{i=1}^{n} x_i y_i (1+\theta_i) \quad \text{with} \quad |\theta_i| \le \gamma_n \quad \text{or:}$$

$$\boxed{fl\left(\sum_{i=1}^{n} x_i y_i\right) = \sum_{i=1}^{n} x_i y_i + \sum_{i=1}^{n} x_i y_i \theta_i \quad \text{with} \quad |\theta_i| \le \gamma_n}$$

- This leads to the final result (forward form)

$$\left| fl\left(\sum_{i=1}^{n} x_i y_i\right) - \sum_{i=1}^{n} x_i y_i \right| \le \gamma_n \sum_{i=1}^{n} |x_i||y_i|$$

- or (backward form)

$$fl\left(\sum_{i=1}^{n} x_i y_i\right) = \sum_{i=1}^{n} x_i y_i (1+\theta_i) \quad \text{with} \quad |\theta_i| \le \gamma_n$$

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

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

✍6 Let $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $y = Ax$. Show that there exist a matrix ΔA s.t.

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

✍7 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?

✍8 Assume you use single precision for which you have $\underline{\mathbf{u}} = 2. \times 10^{-6}$. What is the largest n for which we have $\gamma_n \le 0.01$? Any conclusions for the use of single precision arithmetic?

✍9 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 $y \ne x$]

Recap: Main results on inner products:

- Forward error expression:

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

- Consequence for matrix products: $(A \in \mathbb{R}^{m\times n}, \; B \in \mathbb{R}^{n\times p})$

$$|fl(AB) - AB| \le \gamma_n \, |A||B|$$

- Backward error expression:

$$fl(x^Ty) = [x \,.\!* \,(1 + d_x)]^T [y \,.\!* \,(1 + d_y)]$$

where $\|d_\square\|_\infty \le \gamma_n, \; \square = x, y$. Equality valid even if one of the d_x, d_y absent

Error Analysis for linear systems: Triangular systems

➤ Recall:

ALGORITHM : 1. *Back-Substitution algorithm*

$$\begin{array}{l}
\textit{For } i = n : -1 : 1 \textit{ do:} \\
\quad t := b_i \\
\quad \left.\begin{array}{l}\textit{For } j = i+1 : n \textit{ do} \\ \quad t := t - a_{ij} x_j \\ \textit{End}\end{array}\right\} \begin{array}{l} t := t - (a_{i,i+1:n}, x_{i+1:n}) \\ \quad = t - \textit{an inner product}\end{array} \\
\quad x_i = t / a_{ii} \\
\textit{End}
\end{array}$$

➤ Requirement: each a_{ii} must be $\neq 0$.

➤ Round-off error (use previous results for $(\cdot, \cdot)$)?

➤ Backward error analysis: $\hat{x}$ = computed x solves a slightly perturbed system

The computed solution $\hat{x}$ of the triangular system $Ux = b$ computed by the back-substitution algorithm satisfies:

$$(U + E)\hat{x} = b$$

with

$$|E| \leq n\, \underline{\mathbf{u}}\; |U| + O(\underline{\mathbf{u}}^{\,2})$$

➤ Remarkable result: Backward error $|E|$ is small relative to $|U|$ - unless n is huge

➤ It is said that triangular solve is "backward stable".

Error Analysis for Gaussian Elimination

If no zero pivots are encountered during Gaussian elimination (no pivoting) then the computed factors $\hat{L}$ and $\hat{U}$ satisfy

$$\hat{L}\hat{U} = A + H$$

with
$$\boxed{|H| \leq 3(n-1) \ \times \ \underline{\mathbf{u}} \left(|A| + |\hat{L}| \ |\hat{U}|\right) + O(\underline{\mathbf{u}}^{\,2})}$$

➤ Solution $\hat{x}$ computed via $\hat{L}\hat{y} = b$ and $\hat{U}\hat{x} = \hat{y}$ is s. t.

$$(A+E)\hat{x} = b \quad \text{with} |E| \leq n\underline{\mathbf{u}} \left(3|A| \ + 5 \ |\hat{L}| \ |\hat{U}|\right) + O(\underline{\mathbf{u}}^{\,2})$$

- “Backward” error estimate.
- $|\hat{L}|$ and $|\hat{U}|$ are not known in advance – they can be large.
- What if partial pivoting is used?
- Equivalent to standard LU on matrix PA. Permutations introduce no errors
- $|\hat{L}|$ is small since $|l_{ij}| \le 1$. Therefore, only U is “uncertain”
- In practice partial pivoting is “stable” – i.e., highly unlikely to have a very large U.

Supplemental notes: Floating Point Arithmetic

[For information only – Will *not* be covered in class]

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

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

- $.d_1 d_2 \cdots d_m$ 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 only - no base 10 computers)

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

➤ Try to add 1000.2 = .10002e+03 and 1.07 = .10700e+01:

$1000.2 =$

.1	0	0	0	2	0	4

;

$1.07 =$

.1	0	7	0	0	0	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: $\boxed{.1}\boxed{0}\boxed{0}\boxed{1}\boxed{2}$; or Round result: $\boxed{.1}\boxed{0}\boxed{0}\boxed{1}\boxed{3}$;

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

Result with rounding: $\boxed{.1}\boxed{0}\boxed{0}\boxed{1}\boxed{3}\boxed{0}\boxed{4}$;

✍10 Redo the same thing with 7000.2 + 4000.3 or 6999.2 + 4000.3.

The IEEE standard

32 bit (Single precision) :

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

➤ Number is scaled so it is in the form $1.d_1d_2...d_{23} \times 2^e$ - but leading one is not represented.

➤ e is between -126 and 127.

➤ [Here is why: Internally, exponent e is represented in "biased" form: what is stored is actually $c = e + 127$ – so the value c of exponent field is between 1 and 254. The values $c = 0$ and $c = 255$ are for special cases (0 and ∞)]

64 bit (Double precision) :

±	11 bits	← 52 bits →
sign	exponent	mantissa

➤ Bias of 1023 so if e is the actual exponent the content of the exponent field is $c = e + 1023$

➤ Largest exponent: 1023; Smallest = -1022.

➤ $c = 0$ and $c = 2047$ (all ones) are again for 0 and ∞

➤ Including the hidden bit, mantissa has total of 53 bits (52 bits represented, one hidden).

➤ In single precision, mantissa has total of 24 bits (23 bits represented, one hidden).

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

Hidden bit (Not represented)

Expon. $\downarrow$ $\leftarrow$ 52 bits $\rightarrow$

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

....

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

(Note: The 'e' part has 12 bits and includes the sign)

➤ Conclusion

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

Special Values

- Exponent field = 00000000000 (smallest possible value)
  No hidden bit. All bits == 0 means exactly zero.
- Allow for unnormalized numbers,
  leading to gradual underflow.
- Exponent field = 11111111111 (largest possible value)
  Number represented is ”Inf” ”-Inf” or ”NaN”.

Recent trend: GPUs

➤ Graphics Processor Units: Very fast boards attached to CPUs for heavy-duty computing

➤ e.g., NVIDIA V100 can deliver 112 Teraflops (1 Teraflops = $\mathbf{10^{12}}$ operations per second) for certain types of computations.

➤ Single precision much faster than double ...

➤ ... and there is also “half-precision” which is $\approx \mathbf{16}$ times faster than standard 64bit arithmetic

➤ Used primarily for Deep-learning