Set 3: Finite Arithmetic and Numerical Stability

Kyle A. Gallivan

Department of Mathematics

Florida State University

Foundations of Computational Math 1

Fall 2017

Sources

Additional sources for this set:

- *Matrix Algorithms, Volume 1: Basic Decompositions*, G. W. Stewart, SIAM, 1998.
- *Accuracy and Stability of Numerical Algorithms*, N. J. Higham, SIAM, Second Edition, 2002.

Overview

So far:

- Floating point representation
- Perturbation cause by $\mathbb{R} \rightarrow \mathcal{F}$ mapping of data
- Conditioning analysis of effect of any perturbation to data.

Next:

- Errors introduced by applying a particular algorithm in finite precision
- FP implementation = algorithm for basic arithmetic operations
- Algorithm = particular series of FP operations to solve a problem
- numerical stability of an algorithm

IEEE SP Floating Point Representation Summary

Sign Field (σ)	Exponent Field (ϵ)	Mantissa Field (μ)	Represents
$\{0,1\}$	$1 \leq \epsilon \leq 254$	μ	$(-1)^{\sigma} \quad 1.\ \mu \times 2^{\epsilon-127}$
$\{0,1\}$	$\epsilon = 0$	$\mu = 0$	± 0
$\{0,1\}$	$\epsilon = 255$	$\mu = 0$	$\pm\infty$
$\{0,1\}$	$\epsilon = 255$	$\mu \neq 0$	NaN
$\{0,1\}$	$\epsilon = 0$	$\mu \neq 0$	$(-1)^{\sigma} \quad 0.\ \mu \times 2^{-126}$

- "hidden" bit is 0 for subnormal and 1 for normalized numbers based on the exponent field contents.

IEEE Floating Point Rounding Summary

The following options are allowed by the standard:

- Round to nearest FP number – to even on ties (default)
- Round to nearest FP number – away from 0 on ties
- Round toward 0 (chopping or truncation)
- Round toward $+\infty$
- Round toward $-\infty$

Floating Point Arithmetic

- Since $\mathcal{F} \subset \mathbb{R}$ real operations $\times, \quad /, \quad +, \quad -$ are all well-defined on floating point numbers $x, y \in \mathcal{F}$.
- real arithmetic maps $\mathcal{F} \to \mathbb{R}$ not $\mathcal{F} \to \mathcal{F}$.
- floating point arithmetic must be defined to be closed, i.e., $\mathcal{F} \to \mathcal{F}$.
- a particular implementation of FP arithmetic can be viewed as a choice of algorithm for a given machine
- additional error is introduced by the compuations and must be bounded through stability analysis

Floating point addition

Suppose $x = \beta^{e_1} * m_1 \in \mathcal{F}$ and $y = \beta^{e_2} * m_2 \in \mathcal{F}$ then, $x \boxed{+} y$ requires

1. compare exponents (c-stage)
2. shift mantissa (s-stage)
3. add mantissas (a-stage)
4. normalize (n-stage)

Example

Assume 4-digit floating point decimal arithmetic

$$x = 10^4(0.6314) \text{ and } y = 10^1(0.3865)$$

$$10^4(0.6314\underline{000}) + 10^4(0.0003\underline{865})$$

$$= 10^4(0.6317865) \approx 10^4(0.6318) = x \boxed{+} y$$

no normalization needed

Guard digits are underlined. Note $x \boxed{+} y = fl(x \; op \; y)$

Example

Assume 4-digit floating point decimal arithmetic

$$x = 10^4(0.6314) \text{ and } y = 10^4(0.6065)$$

$$10^4(0.6314) - 10^4(0.6065) = 10^4(0.0249) \quad \text{(no shift needed)}$$

$$x \boxed{-} y = 10^4(0.0249) = 10^3(0.2490)$$

Note $fl(x - y) = x \boxed{-} y$.

Simple Floating Point Multiplication

Suppose $x = \beta^{e_1} * m_1 \in \mathcal{F}$ and $y = \beta^{e_2} * m_2 \in \mathcal{F}$ then $x \boxed{*} y$ requires

1. multiply mantissas ($m \leftarrow m_1 * m_2$) – assume double length internal mantissa here
2. add exponents ($e \leftarrow e_1 + e_2$)
3. round internal mantissa m to representation length of mantissa

Note there are many floating point multiplier designs addressing area and time complexity tradeoffs.

Example

Assume 4-digit floating point decimal arithmetic

$$x = .5130 \times 10^4 \text{ and } y = .3120 \times 10^1$$

$$m = m_1 * m_2 = .1600\underline{5600}$$

$$e_1 + e_2 = 5 = e_3$$

$m_3 = .1601$ (round using all 4 extra digits)

$$x \boxed{*} y = .1601 \times 10^5 = fl(x * y)$$

So the exact product 16005.6 is rounded to 16010.0.

Standard Model

- Let $op : \mathbb{R} \to \mathbb{R}$ be any of the real operations $\times, \quad /, \quad +, \quad -$.
- Let $\boxed{op} : \mathcal{F} \to \mathcal{F}$ be the corresponding machine arithmetic operation with floating point operands.

Lemma 3.1. *The floating point arithmetic satisfies the* **standard model** *if* $\forall x, y \in \mathcal{F}$

$$x \boxed{op} y = (x \; op \; y)(1 + \delta), \quad |\delta| \leq u$$

Standard Model

- $x \boxed{op} y$ is as good as $fl(x\ op\ y)$ in terms of rounding error bound, i.e.,

$$x \boxed{op} y = (x\ op\ y)(1+\delta), \quad |\delta| \leq u$$
$$fl(x\ op\ y) = (x\ op\ y)(1+\mu), \quad |\mu| \leq u$$

- In IEEE floating point systems the use of three extra bits (a guard bit, a rounding bit and a sticky bit) guarantees (Higham 2002)

$$x \boxed{op} y = fl(x\ op\ y)$$

- Some texts therefore use the notation $fl(x\ op\ y)$ for $x \boxed{op} y$.
- This does not affect error bounds and we will take the exact answer rounded as $x \boxed{op} y$, typically.

Standard Model Modified Form

Lemma 3.2. *The floating point arithmetic satisfies the* **standard model** *if* $\forall x, y \in \mathcal{F}$

$$fl(x) = \frac{x}{(1+\mu)}, \quad |\mu| \leq u$$

$$x \boxed{op} y = \frac{(x \; op \; y)}{(1+\delta)}, \quad |\delta| \leq u$$

Standard Model Modified Form

The standard model above assumes no underflow but it is easily adjusted to take this possibility into account:

$$x \boxed{op} y = fl(x\ op\ y)(1+\delta)+\eta$$

for $+,\ -,\ *,\ /$ and $|\delta| \leq u$ and $|\eta| \leq 0.5\ \beta^{e_{min}-t}$ when gradual underflow via subnormal numbers is used (see Higham 2002 Accuracy and Stability of Numerical Algorithms, Second Edition).

When the bound is applied in an analysis only one of δ and η are nonzero – the former when no underflow occurs and the later when it does. The bound on η is simply one half the distance between two subnormal floating point numbers. For $+$ and $-$, $\eta = 0$ when gradual underflow is used. This simply reflects the fact that gradual underflow makes cancellation exact for two floating point numbers that are sufficiently close.

Weak and Forward Stability

- Weak stability says that finite precision causes

$$|g(d) - f(d)|$$

to be acceptably small where $g(d)$ is an algorithm applied to input data d in finite arithmetic.

- However the answer is not the exact answer for nearby input data. Acceptably small means that it is within the radius predicted by the condition number relative to machine precision.

$$\frac{|g(d) - f(d)|}{|f(d)|} \leq \kappa p(n) \epsilon_M$$

- Forward stability says that finite precision causes $|g(d) - f(d)|$ to be bounded but not necessarily as above.

Backward Stability

Definition 3.1. An algorithm is **backward stable** if the computed result is the exact result of solving the problem with nearby input data.

Note. We therefore combine the conditioning of the problem and the backward stability of the algorithm to assess the accuracy of numerical algorithms.

Backward Stability

- For a backward stable algorithm, the computed result is the exact result of solving the problem with nearby input data which is close to the exact result for the original data for a well-conditioned problem.
- The relative perturbation from the backward error analysis is bounded by $p(n)\epsilon_M$ where $p(n)$ is a mildly growing function of the size of the problem.
- So the perturbation due to computing is roughly the same order as the perturbation to the data due to the initial representation in finite precision, i.e., in the uncertainty of the data.

Conditioning and Stability Types

$$\mathcal{E} = \frac{|g(d) - f(d)|}{|f(d)|} \text{ error,} \quad \frac{|f(d+e) - f(d)|}{|f(d)|} \leq \kappa \frac{|e|}{|d|} \text{ conditioning}$$

$$\frac{|g(d) - f(d)|}{|f(d)|} \leq C(n) \text{ forward stable}$$

$$\frac{|g(d) - f(d)|}{|f(d)|} \leq \kappa p(n) \epsilon_M \text{ weakly stable}$$

$$\frac{|f(d+e) - f(d)|}{|f(d)|} \leq \kappa p(n) \epsilon_M \text{ backward stable}$$

Conditioning and Stability Types

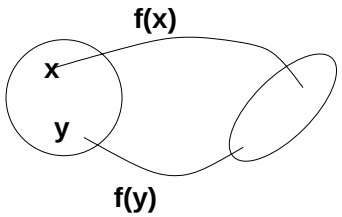
- A proof of the backward stability of an algorithm must show
  1. $g(d) = f(d + e)$, i.e., computed solution is the exact solution of another problem
  2. The problem solved is "nearby", i.e.,

$$\frac{|e|}{|d|} \leq p(n)\epsilon_M$$

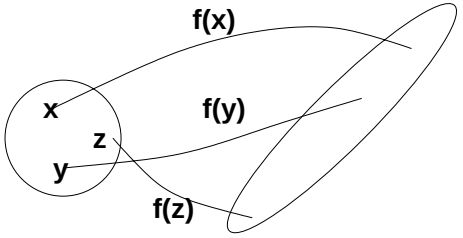
     where $p(n)$ is a slowly growing function of the parameters of the problem and ϵ_M is machine precision.

- putting these bounds into the conditioning bound yields a bound on the error between the computed solution and the exact solution.

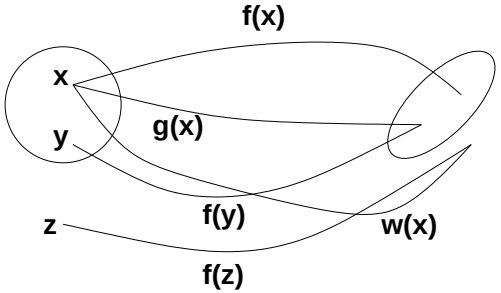
Geometry of Stability (Stewart)



Well conditioned



Ill conditioned



backward stable on well conditioned -- g(x)

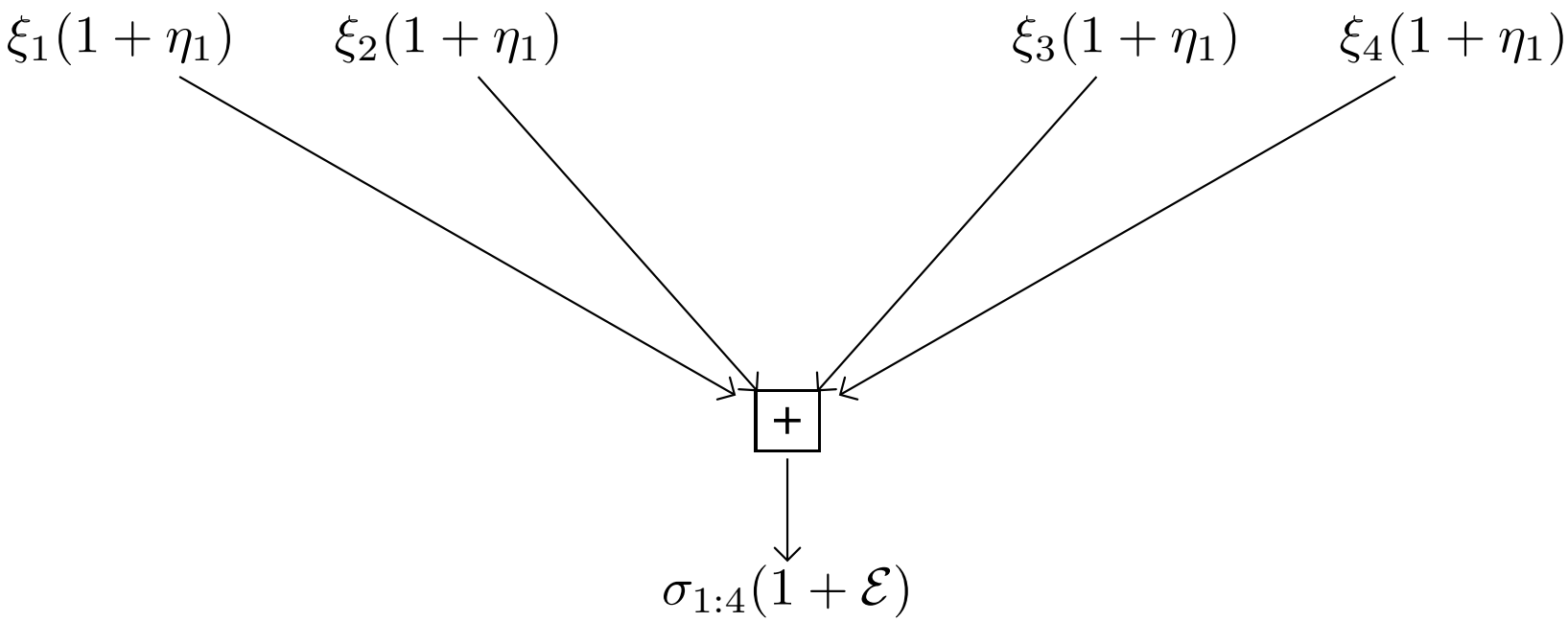
weakly stable on well conditioned -- w(x)

(note it is within the circle defined by the major axis)

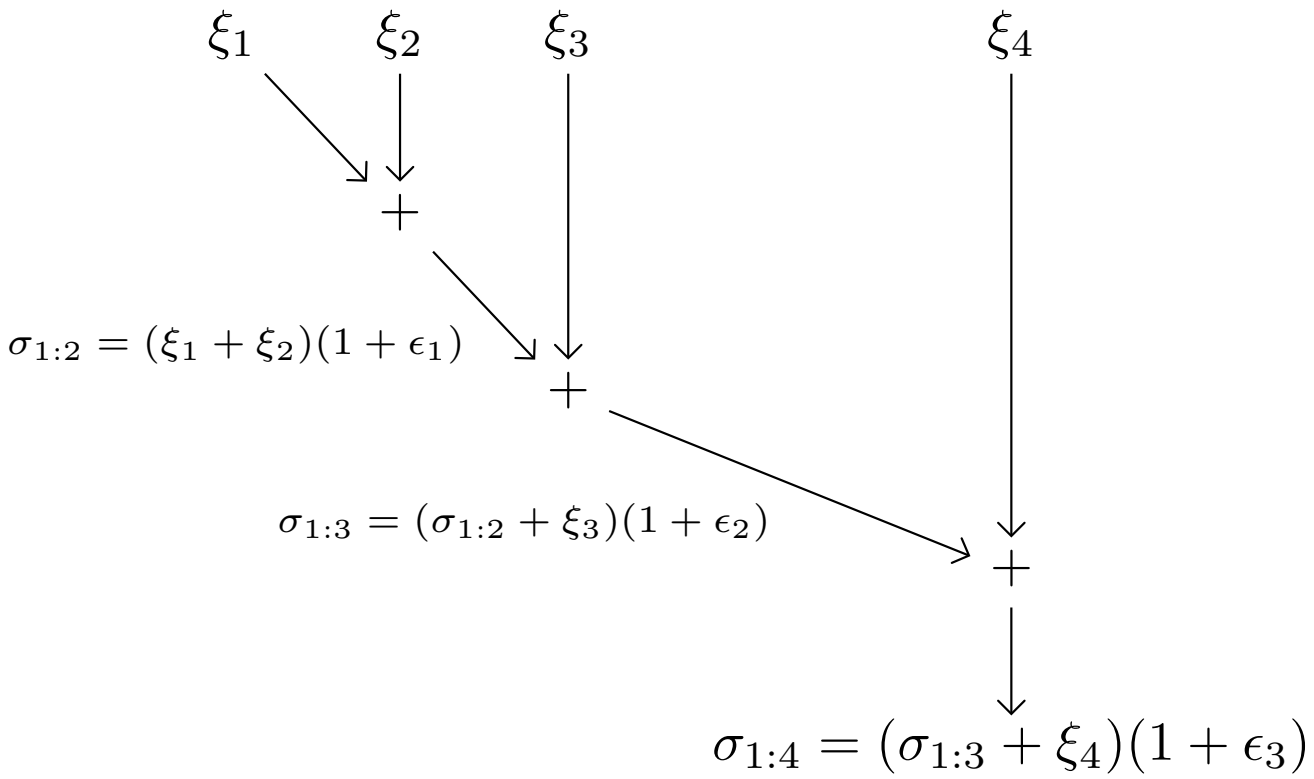
Summary

- A stable algorithm on a well-conditioned problem produces accurate results.
- An unstable algorithm on a well-conditioned problem may produce inaccurate results
- A stable algorithm on an ill-conditioned problem may produce inaccurate results.

Resolvent Form of Summation and Conditioning

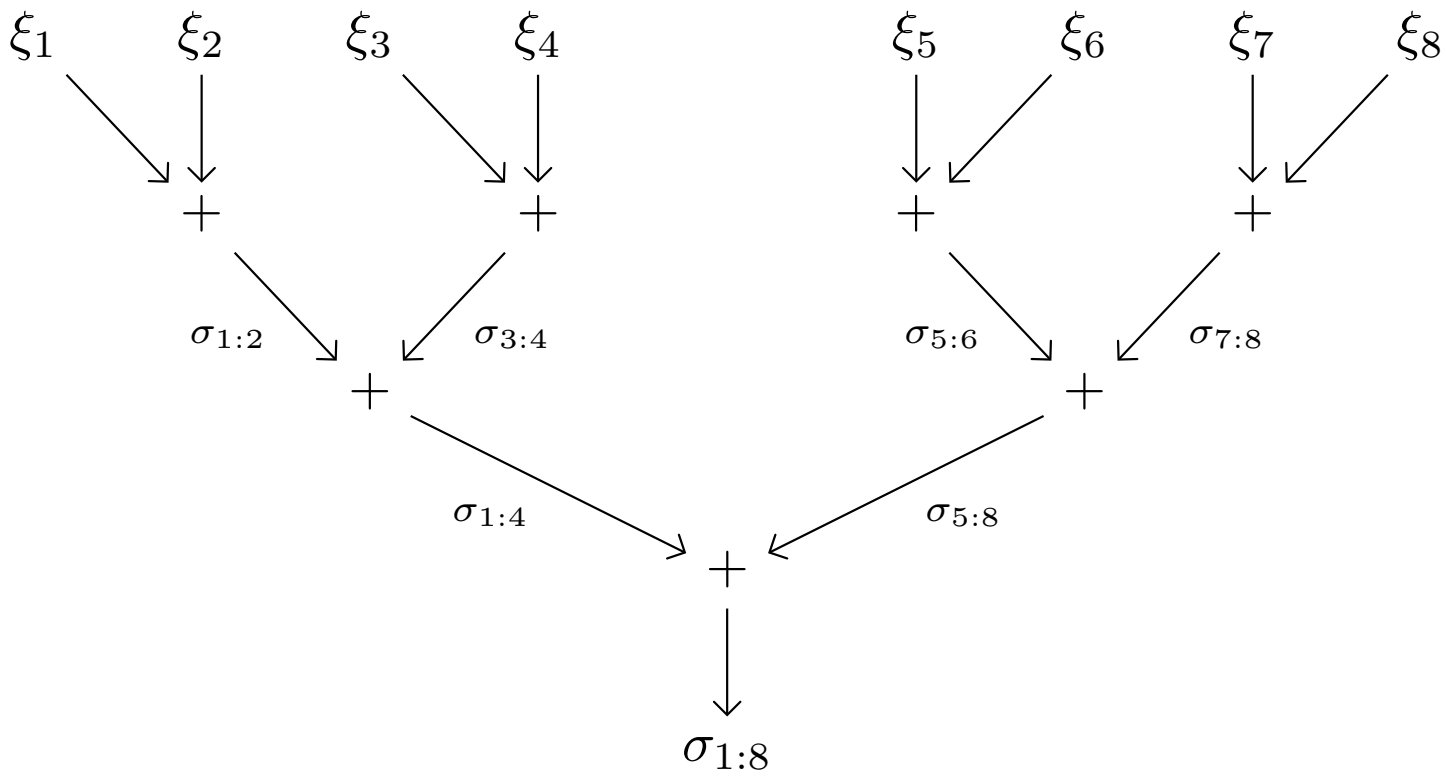


Error and Stability for an Algorithm



$\sigma_{1:i}$ and ξ_i are all floating point numbers

Different Algorithm



Error bound from stability analysis will be different.

An Example of Error Analysis

Consider the sum of n FP numbers,i.e., we are assuming no initial representation error.

$$\sigma_n = \xi_1 + \cdots + \xi_n$$

No assumption is made about ordering of the scalars. We consider the simple recursion:

$$\begin{aligned} \sigma_0 &= 0 \\ \sigma_i &= \sigma_{i-1} + \xi_i \quad 1 \leq i \leq n \end{aligned}$$

Error Analysis

Assume that $n = 4$ and use the standard model of floating point arithmetic of IEEE with the extra bits needed so

$$fl(x\ op\ y) = x\ \boxed{op}\ y.$$

The first iteration yields:

$$\sigma_2 = fl(\xi_1 + \xi_2) = (\xi_1 + \xi_2)(1 + \epsilon_1) = \xi_1(1 + \epsilon_1) + \xi_2(1 + \epsilon_1)$$

where $|\epsilon_1| \leq u$ and u is unit roundoff.

Error Analysis

The second iteration yields:

$$\sigma_3 = fl(\sigma_2 + \xi_3) = (\sigma_2 + \xi_3)(1 + \epsilon_2) = \sigma_2(1 + \epsilon_2) + \xi_3(1 + \epsilon_2)$$
$$= \xi_1(1 + \epsilon_1)(1 + \epsilon_2) + \xi_2(1 + \epsilon_1)(1 + \epsilon_2) + \xi_3(1 + \epsilon_2)$$

where $|\epsilon_2| \leq u$.

Note that ξ_1 and ξ_2 have been "affected" by an additional ϵ_i.

Error Analysis

The third iteration yields:

$$\begin{aligned}\sigma_4 =& fl(\sigma_3 + \xi_4) = (\sigma_3 + \xi_4)(1+\epsilon_3) = \sigma_3(1+\epsilon_3) + \xi_4(1+\epsilon_3) \\ =& \xi_1(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) + \xi_2(1+\epsilon_1)(1+\epsilon_2)(1+\epsilon_3) \\ &+ \xi_3(1+\epsilon_2)(1+\epsilon_3) + \xi_4(1+\epsilon_3)\end{aligned}$$

$$\begin{aligned}\sigma_4 =& \xi_1(1+\eta_1) + \xi_2(1+\eta_2) + \xi_3(1+\eta_3) + \xi_4(1+\eta_4) \qquad (1) \\ =& (\xi_1 + \xi_2 + \xi_3 + \xi_4) + (\xi_1\eta_1 + \xi_2\eta_2 + \xi_3\eta_3 + \xi_4\eta_4) \qquad (2)\end{aligned}$$

Error Analysis

- (1) is backward error analysis
- (2) is forward error analysis
- the floating point sum is the exact sum of real numbers that are modifications of the original floating point numbers.
- errors have been "cast back" to input data
- this is not always possible – it depends on number of errors and structure of input

Error Analysis

$$
\begin{aligned}
1+\eta_1 &= 1+\epsilon_1+\epsilon_2+\epsilon_3+\epsilon_1\epsilon_2+\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_3 \\
&= 1+\epsilon_1+\epsilon_2+\epsilon_3+O(u^2) \\
|\eta_1| &\lesssim 3u \\
1+\eta_2 &= 1+\epsilon_1+\epsilon_2+\epsilon_3+\epsilon_1\epsilon_2+\epsilon_2\epsilon_3+\epsilon_1\epsilon_2\epsilon_3 \\
&= 1+\epsilon_1+\epsilon_2+\epsilon_3+O(u^2) \\
|\eta_2| &\lesssim 3u \\
1+\eta_3 &= 1+\epsilon_2+\epsilon_3+\epsilon_2\epsilon_3 = 1+\epsilon_2+\epsilon_3+O(u^2) \\
|\eta_3| &\lesssim 2u \\
1+\eta_4 &= 1+\epsilon_3 \\
|\eta_4| &\lesssim u
\end{aligned}
$$

Error Analysis

- roundoff errors may cancel — bound may be very loose
- relative perturbation worst case bound $\approx nu$ for ξ_i early in list.
- generically this is a good reason to avoid large magnitudes first
- algorithm is backward stable for reasonable sizes of n
- ill-conditioned sums can still have large relative error
- large n relative to u needs more thought
- error analysis can be adapted to give tighter forward error bounds when more complicated algorithms that include sorting are used.

Error Analysis Simplification Bounds

- There are many simple inequalities that are useful in manipulating error expressions, especially for basic vector and matrix operations.
- See for example, Higham Accuracy and Stability of Numerical Algorithms, Second Edition. In particular, Chapter 3 contains several useful results.
- Two basic lemmas follow.

Error Analysis Simplification Bounds

Lemma 3.3. *If* $|\delta_i| \leq u$ *with* $\rho_i = \pm 1$, $1 \leq i \leq n$, *and* $nu < 1$ *then*

$$\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n$$

$$|\theta_n| \leq \frac{nu}{1-nu}$$

Error Analysis Simplification Bounds

Lemma 3.4. *If* $|\delta_i| \leq u$ *and* $nu \leq 0.01$ *then*

$$\prod_{i=1}^{n}(1+\delta_i) = 1+\eta_n$$

$$|\eta_n| \leq 1.01nu$$

Basic Considerations

Some numerical error considerations when developing a stable algorithm:

- Overflow
- Convergence of infinite sequence
- Convergence of infinite series (summation)
- Cancellation

Overflow

- intermediate results may grow and move outside the range of the floating point representation
- solution requires consideration of relative scale of operands
- rewriting or scaling can solve the problem
- excessive growth of intermediate values compared to magnitude of input data or solution tends to indicate stability problems

Overflow

$$p(x) = ax^2 + bx + c$$

$$p(x) = 10^{20}x^2 - 3 \times 10^{20}x + 2 \times 10^{20} = 0$$

$$x_1 = 1 \text{ and } x_2 = 2$$

$$y_{max} \approx 10^{38} \text{ IEEE single precision}$$

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$ac = 10^{40} \quad b^2 = 10^{40}$$

Solution: rescale problem by $\max(|a|, |b|, |c|)$

Overflow

$p(x) = 10^{-20}x^2 - 3x + 2 \times 10^{20} = 0, \quad x_1 = 10^{20}$ and $x_2 = 2 \times 10^{20}$

- Wide variation in parameters' magnitudes tends to be a problem.
- Note $p(x)$ is close to a constant function due to scaling.
- Applying previous scaling of parameters does not yield a useful polynomial.

$$\tilde{p}(x) = \frac{1}{2}10^{-40}x^2 - \frac{3}{2} \times 10^{-20}x + 1 = 0$$

Overflow

$p(x) = 10^{-20}x^2 - 3x + 2 \times 10^{20} = 0, \quad x_1 = 10^{20}$ and $x_2 = 2 \times 10^{20}$

- rescale variable $x = 10^{20}y \rightarrow p(y) = 10^{20}y^2 - 3 \times 10^{20}y + 2 \times 10^{20}$
- apply previous scaling of parameters to find $y_1 = 1$ and $y_2 = 2$
- no overflow when computing $x_1 = 10^{20}y_1$ and $x_2 = 10^{20}y_2$
- scaling strategies get fairly complicated

Overflow

Consider evaluating $r = \sqrt{x^2 + y^2}$

$$x = y = 10^{20} \rightarrow r = \sqrt{2}\ \ 10^{20} \quad \therefore \text{ within range}$$

$$x^2 = 10^{40} \rightarrow \text{ out of range, i.e., FP operation yields INF}$$

An algorithmic solution to overflow problem:

if $|y| > |x|$

$$r = |y| \times sqrt(1 + (x/y) * *2)$$

else

$$r = |x| \times sqrt(1 + (y/x) * *2)$$

end if

Limit of Sequence Idea

$$e = \lim_{n\to\infty} s_n = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$$

$$\lim_{n\to\infty} |e - s_n| = 0, \text{ but } \lim_{n\to\infty} |e - fl(s_n)| = \gamma > 0$$

- For single precision $u \approx 6.0 \times 10^{-8}$
- $1/n = 10^{-k}$ nonterminating binary expansion
- As $n \to u^{-1}$ digits lost in
$$1 + \frac{1}{n}$$
- $n \geq 8 \to fl(s_n) = fl((1 + \alpha u)^n) = fl(1^n) = 1$
- loss of accuracy even with exact exponentiation
- no further improvement after $k = 8$

Error Results

$$\hat{f} = fl\left(\left(1+\frac{1}{n}\right)^n\right) \text{ and } e = 2.718281828\ldots$$

n	$fl(1+\frac{1}{n})$	$\hat{f}$	$e-\hat{f}$
10^1	1.10000002	2.593743	1.25×10^{-1}
10^2	1.00999999	2.704811	1.35×10^{-2}
10^3	1.00100005	2.717051	1.23×10^{-3}
10^4	1.00010002	2.718597	-3.15×10^{-4}
10^5	1.00001001	2.721962	-3.68×10^{-3}
10^6	1.00000095	2.595227	1.23×10^{-1}
10^7	1.00000012	3.293968	-5.76×10^{-1}
$>10^8$	1.00000000	1.0	1.71×10^{0}

Convergence of Infinite Series

Same phenomenon allows convergence of a floating point version of a summation in $\mathcal{F}$ that does not coverge in $\mathbb{R}$

$$\sum_{i=1}^{\infty} \frac{1}{i} \to \infty$$

$$\sum_{i=1}^{\infty} fl(\frac{1}{i}) \to \gamma < \infty$$

$$s = fl(s + \frac{1}{i}) \quad i = 1, 2, \ldots$$

eventually $s = fl(s + \frac{1}{i}) = s$.

Approximation via Infinite Series

$$\sum_{i=1}^{\infty} \frac{1}{i^2} \rightarrow \frac{\pi^2}{6}$$

- $s = fl(s + 1/i^2)$ is fixed after $i = 4096$ (single precision)
- $s = s + 2^{-24}$ with $s \approx 1.6$
- $\pi^2/6 = 1.644934066848$ vs $s = \underline{1.64472532}$
- all signs $+$, magnitudes are the problem, sum in reverse order
- need to know the number of terms beforehand
- reverse order $k = 10^7 \rightarrow s = \underline{1.64493}39389801$
- reverse order $k = 10^9 \rightarrow s = \underline{1.6449340}581894$

Cancellation

- $x,\ y,\ z \in \mathcal{F}$ and $s = fl(fl(x+y)+z)$
- Simple recursive summation is stable.
- Conditioning (Example of Stewart '98)

$$x = 472635.0000 \quad y = 27.5013 \quad z = -472630.0000$$

$$x + y + z = 32.5013 \quad \kappa = \frac{945292.5013}{32.5013} \approx 3 \times 10^4$$

- Expect to lose 4 digits
- cancellation in exact sum implies ill-conditioning, what about finite precision?
- 6 digit decimal floating point numbers and arithmetic (one guard digit also)

Cancellation

Consider $fl(fl(x+y)+z)$

$$x = 472635.0000 \quad y = 27.5013 \quad z = -472630.0000$$

$$fl(472635.0000 + 27.5013) = fl(472635.0 + 27.5)$$

$$= fl(472662.5) = 472663$$

$$fl(472663 - 472630) = 33$$

$$x + y + z = 32.5013$$

- cancellation in second operation
- result in second operation is exact!
- 2 digits only as expected
- worst case digit loss in final result

Cancellation

Consider $fl(fl(x+z)+y)$ (same algorithm different ordering of data)

$$x = 472635.0000 \quad y = 27.5013 \quad z = -472630.0000$$

$$fl(472635.0 - 472630.0) = 5.00000$$

$$fl(05.00000 + 27.50130) = 32.5013$$

$$x + y + z = 32.5013$$

- cancellation in first operation as bad as before, i.e., 5 digits of agreement in operands of opposite sign.
- first result is exact!
- final result is exact!
- same results if $27.5 \leq y \leq 28.4999$

Cancellation

- For these two orderings, the operation with cancellation **does not** introduce errors – the results are exact.
- In the first ordering, cancellation reveals loss of significant digits from the **first operation**.
- $fl(472635.0000 + 27.5013) = 472663.0000$ destroyed all information about rightmost 4 digits in $27.\underline{5013}$
- The second operation cannot recreate the information.

Cancellation

- cancellation and ordering reveals ill-conditioning of sum
- ill-conditioning does not mean answers **must** be bad only that they cannot be guaranteed to be good
- here the algorithm is stable but an unstable algorithm may be due to cancellation
- various stable summation algorithms available – they differ in the size of their a priori forward error bounds

Approximation via Infinite Series

(Example of Forsythe) Recall $e^{-5.5} = 0.00408677\ldots$ is well conditioned.

Truncate Taylor series to compute approximation:

$$e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{3!} + \ldots$$

$$e^{-5.5} = 1.0000 - 5.5000 + 15.125 - 27.729 + 38.128 + \ldots$$

Approximation via Infinite Series

(Example of Forsythe)

$$e^{-5.5} = 1.0000 - 5.5000 + 15.125 - 27.729 + 38.128 + \ldots$$

- after some index k, terms will not affect the result
- terms in the "middle" of the sequence get large in magnitude compared to the final answer and then go to 0.
- terms alternate in sign to bring the final sum down from large intermediate value
- large intermediate sum values use most of the finite digit representation for information that will be removed by subsequent terms, i.e., significant information for final value is lost in dropped digits

Approximation via Infinite Series

(Example of Forsythe)

- $e^{-5.5} = 0.004086771438\ldots$
- For IEEE single precision, the computed value is

$$s_{28} = 0.0040874378$$

- For $\beta = 10 \;\; t = 5$, the computed value is

$$s_{25} = 0.003945199$$

Example of Forsythe in FP with $\beta = 10$ $t = 5$

k	k-th term	computed s_k
1	-5.500000	-4.500000
2	15.125000	10.625000
3	-27.729200	-17.104240
4	38.127601	21.023399
5	-41.940399	-20.916992
6	38.445301	17.528299
7	-30.207000	-12.678679
8	20.767299	8.088620
9	-12.691200	-4.602586
10	6.980140	2.377549
11	-3.490070	-1.112518
12	1.599609	0.487089

Example of Forsythe in FP with $\beta = 10 \;\; t = 5$

k	k-th term	computed s_k
13	-0.6767600178719	-0.1896701604128
14	0.2658700048923	0.0762000009418
15	-0.0974859967828	-0.0212860815227
16	0.0335110016167	0.0122250001878
17	-0.0108420001343	0.0013830000535
18	0.0033126999624	0.0046957000159
19	-0.0009589500260	0.0037366999313
20	0.0002637100115	0.0040004001930
21	-0.0000690669985	0.0039312997833
22	0.0000172670007	0.0039486000314
23	-0.0000041290000	0.0039444998838

Approximation via Infinite Series

(Example of Forsythe)

$$e^{-x} = 1/(1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots)$$

$$e^{-5.5} = 1/(1.0000 + 5.5000 + 15.125 + 27.729 + 38.128 + \dots)$$

- denominator is increasing until after some index k, terms will not affect the result
- terms are all same sign – all digits contribute

Approximation via Infinite Series

(Example of Forsythe)

$$e^{-x} = 1/(1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \dots)$$

$$e^{-5.5} = 1/(1.0000 + 5.5000 + 15.125 + 27.729 + 38.128 + \dots)$$

- $e^{-5.5} = 0.004086771438\dots$
- For IEEE single precision, the computed value is

$$s_{21} = \underline{0.0040867719}799$$

- For $\beta = 10 \;\; t = 5$, the computed value is

$$s_{23} = \underline{0.0040867584}758$$

Example of reciprocal series in FP with $\beta = 10 \;\; t = 5$

k	k-th term	computed s_k
1	5.50000	0.1538500040770
2	15.12500	0.0461999997497
3	27.72920	0.0203000009060
4	38.12770	0.0114000001922
5	41.94049	0.0076999999583
6	38.44549	0.0060000000522
7	30.20719	0.0049999998882
8	20.76740	0.0046000001021
9	12.69120	0.0043000001460
10	6.9801	0.0041899997741
11	3.4900	0.0041299997829
12	1.5996	0.0041100000963

Example of reciprocal series in FP with $\beta = 10 \;\; t = 5$

k	k-th term	computed s_k
13	0.6767600178719	0.0040899999440
14	0.2658700048923	0.0040899999440
15	0.0974859967828	0.0040879999287
16	0.0335110016167	0.0040870001540
17	0.0108420001343	0.0040870001540
18	0.0033128000796	0.0040867999196
19	0.0009589699912	0.0040867598727
20	0.0002637199941	0.0040867598727
21	0.0000690700035	0.0040867589414
22	0.0000172679993	0.0040867589414
23	0.0000041293001	0.0040867584758
24	0.0000009463000	0.0040867584758

Approximation via Infinite Series

- mathematical analysis of sequence and series convergence is not enough
- be careful of magnitude of terms and precision
  - overflow, underflow
  - unchanging
- be careful if large magnitudes with alternating signs contribute to accuracy of final "small" answer
- another ordering of terms may be better
- another series may be better

Roots of a Quadratic

$$p(x) = ax^2 + bx + c$$

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$b^2 \gg |4ac| \rightarrow |b| \approx \sqrt{b^2 - 4ac}$$

- Two terms are of concern for cancellation.
- First: $-b + \sqrt{b^2 - 4ac}$ or $-b - \sqrt{b^2 - 4ac}$ will have cancellation.
- $fl(\sqrt{b^2 - 4ac})$ will have roundoff errors.
- This cancellation will amplify their significance relative to the true difference.
- This cancellation damaging in this algorithm even for well-conditioned roots.

Roots of a Quadratic

$$x^2 - 6.433x + 0.009474 = 0$$

$$x_1 = 6.431526943899538$$

$$x_2 = 0.001473056100462$$

Use the standard quadratic formula to see effects of cancellation since

$$b^2 \gg |4ac|$$

$$(6.433)^2 - 4 \times 0.009474 = 41.345592\ldots$$

Use $\beta = 10$, $t = 4$ with rounding

$$x^2 - 6.433x + 0.009474 = 0$$

compute b^2: $\quad u = fl[10^1(0.6433)]^2 = 10^2(0.4138)$

compute $4ac$: $\quad v = fl[10^1(0.4000) * 10^{-2}(0.9474)] = 10^{-1}(0.3790)$

compute $b^2 - 4ac$: $\quad w = fl[u - v] = fl[10^2(0.4138) - 10^{-1}(0.3790)]$

$$= fl[10^2(0.4138 - 0.0003790)] = 10^2(0.4134)$$

compute $\sqrt{b^2 - 4ac}$: $\quad y = fl[\sqrt{w}] = 10^1(0.6430)$

Use $\beta = 10$, $t = 4$ with rounding

compute $-b - \sqrt{b^2 - 4ac}$: $\quad z = fl[10^1(0.64330000 - 0.6430000)]$

$$= 10^{-2}(0.3000)$$

compute $(-b - \sqrt{b^2 - 4ac})/2$: $\quad \alpha_- = fl[z/2] = 10^{-2}(0.1500)$

compute $-b + \sqrt{b^2 - 4ac}$: $\quad z = fl[10^1(0.64330000 + 0.6430000)]$

$$= 10^2(0.1286)$$

compute $(-b + \sqrt{b^2 - 4ac})/2$: $\quad \alpha_+ = fl[z/2] = 10^1(0.6430)$

Roots of a Quadratic

$$x^2 - 6.433x + 0.009474 = 0$$

$$x_1 = 6.431526943899538$$

$$x_2 = 0.001473056100462$$

- Using 4 digit arithmetic.
- The computed root $\alpha_+ = 6.430$ agrees with x_1 to three significant digits. (The digit is lost due to growth of an intermediate value to 12.863 before rounding.)
- The computed root $\alpha_- = 0.001500$ agrees with x_2 to one significant digit and is the root rounded to two digits (half of the digits lost).
- The two roots are well-separated so they are well-conditioned.
- Cancellation causes accuracy problems for this algorithm on x_2.

Alternative Algorithm

Find root larger in magnitude, x_1, then recover x_2

$$p(x) = ax^2 + bx + c$$

$$x_1 = \frac{-(b + sign(b)\sqrt{b^2 - 4ac})}{2a}$$

$$x_1 x_2 = \frac{c}{a}$$

Fixes problem with the first cancellation.

Roots of a Quadratic

Use $\beta = 10$, $t = 4$ with rounding

$$x^2 - 6.433x + 0.009474 = 0$$

$$\alpha_1 = fl\Big[\frac{fl[10^1((0.6433) + 10^1(0.6430)]}{2}\Big] = 10^1(0.6430)$$

$$\alpha_2 = fl\Big[\frac{fl[2 * 10^{-2}(0.9474)]}{fl[10^1((0.6433) + 10^1(0.6430)]}\Big] = 10^{-2}(0.1473)$$

- The computed root α_2 agrees with x_2 to four significant digits.
- The computed root α_1 agrees with x_1 to three significant digits.

Roots of a Quadratic

- if $b^2 \approx 4ac$ a second cancellation occurs
- nearly repeated roots $\rightarrow$ ill-conditioning
- not fixed easily, requires tricks and extra precision

Trigonometric Function

- $f(x) = \frac{1-\cos x}{x^2}$
- $0 \leq f(x) < 1/2$ and $\lim_{x \to 0} f(x) = 1/2$
- well-conditioned
- consider $x = 1.2 \times 10^{-5}$ and 10 significant digits
- cancellation results in no accuracy

$$1 - \cos x = 1 - 0.9999999999 = 0.0000000001$$

$$\frac{1 - \cos x}{x^2} = \frac{10^{-10}}{1.44 \times 10^{-10}} = 0.6944\ldots$$

Trigonometric Function

$$\cos x = 1 - 2\sin^2(x/2) \rightarrow f(x) = \frac{1}{2}\left(\frac{\sin(x/2)}{(x/2)}\right)^2$$

$$\text{given } \sin(x/2) \text{ to 10 digits} \rightarrow f(x) = 0.4999999997$$

- rewrite removes cancellation
- new algorithm is stable
- a stable algorithm on a well-conditioned problem is reliable

Taylor Series Derivation of Alternate

For small x,

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!}\cos\xi$$

$$y = \frac{1-\cos x}{x^2} = \frac{1}{2} - \frac{x^2}{4!} + \frac{x^4}{6!} - \frac{x^6}{8!}\cos\xi$$

$$y \approx \frac{1}{2} - \frac{x^2}{4!} + \frac{x^4}{6!} \quad \text{and} \quad |\frac{x^6}{8!}\cos\xi| \leq \frac{10^{-6}}{8!} \approx 2.5 \times 10^{-11}, \quad |x| \leq 0.1$$

- No cancellation due to difference in magnitudes.
- More than single precision accuracy.
- Algorithm depends on $|x|$.

Cancellation

- shows earlier error
- may be fixable
- may be unimportant
- may show ill-conditioning and may not be fixable
- conclusion: watch for it in algorithm design, perhaps monitor particular data, avoid it if possible

Instability Without Cancellation

- single cancellation often enough to cause problems
- accumulation of many roundoff errors possible (but not typical)
- small number noncancellation error can cause significant loss of accuracy
- other examples possible, e.g., factorization to solve a linear system

Stability

- Conditioning is a property of a problem that is independent of the algorithm used to solve the problem.
- A particular algorithm on a particular problem defines a sequence of floating point operations each of which may be affected by finite precision (roundoff error).
- The goal of analyzing the stability of an algorithm is to guarantee that the effects of finite precision will not degrade the accuracy of the solution or to identify aspects of the algorithm that may cause such degradation.
- Stability analysis requires knowledge of the floating point system (representation and arithmetic) as well as the algorithm.

Stability

- Stability is analyzed for a class of problems not just a single specific instance, e.g., solving symmetric positive definite linear systems.
- Instability often results from a small number of key computations that amplify errors.
- Roundoff errors can cancel due to structure in the problem resulting in more accuracy than predicted by bound from a stability analysis.
- Roundoff can be beneficial, i.e., it may emphasize directions of interest.
- Stability analysis also includes consideration of effects of discretization of infinite dimensional problems and nondiscrete domains. This is discussed later.

Roundoff Comments

(Higham '02)

- Try to avoid subtracting quantities contaminated by error (sometimes it is unavoidable, and it is not always a bad thing)
- Try to keep the magnitude of intermediate quantities from growing large compared to magnitude of solution. (looking at the trends in intermediate values is often very informative)
- Consider different expressions that are equivalent mathematically but not numerically. They may each be reliable under different circumstances.

Roundoff Comments

(Higham '02)

- Take precautions to avoid underflow and overflow, e.g., monitor values, scale, ...
- Use $new = old + small\ \ update$ if small update can be computed to several digits.
- Use well-conditioned transformations to your problem if possible and not too inefficient, e.g., orthogonal transformations in linear algebra.