Joys and pitfalls of numerical computing

Alex H. Barnett[1]

10/14/21

FWAM *Episode III — Revenge of the Si~~th~~ngular Value Decomposition*

[1]Center for Computational Mathematics, Flatiron Institute, Simons Foundation

Goals/outline

Crucial practical advice & good habits, examples, further reading

- how does accuracy improve with effort? *rate of convergence*
- finite-precision ("rounding error") considerations
- what accuracy is *reasonable* to demand? conditioning of a *problem*
- did you mess up getting such accuracy? stability of an *algorithm*

Goals/outline

Crucial practical advice & good habits, examples, further reading

- how does accuracy improve with effort? *rate of convergence*
- finite-precision ("rounding error") considerations
- what accuracy is *reasonable* to demand? conditioning of a *problem*
- did you mess up getting such accuracy? stability of an *algorithm*

Please ask questions*

$*$ with finite time-frequency product ☺

PS I will ask YOU questions ☺

Accuracy: how much to you need? have?

Usually care about *relative* error: $\varepsilon := \frac{\text{size of error of thing}}{\text{size of thing}} = \frac{|y_{\text{computed}} - y_{\text{true}}|}{|y_{\text{true}}|}$

eg 0.00123 ± 0.00001 is not "correct to 5 digits", rather, 2 digits, rel. err. 10^{-2}, ie 1% err.

Accuracy: how much to you need? have?

Usually care about *relative* error: $\varepsilon := \frac{\text{size of error of thing}}{\text{size of thing}} = \frac{|y_{\text{computed}} - y_{\text{true}}|}{|y_{\text{true}}|}$

eg 0.00123 ± 0.00001 is not "correct to 5 digits", rather, 2 digits, rel. err. 10^{-2}, ie 1% err.

Interesting things take a while to compute → is $\varepsilon = 10^{-1}$ ok, or need 10^{-10} ?

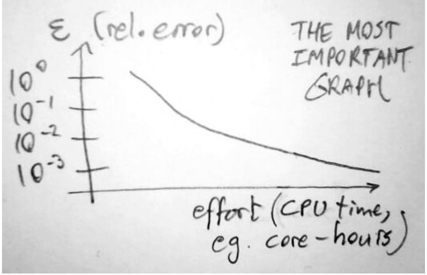
Accuracy: how much to you need? have?

Usually care about *relative* error: $\varepsilon := \frac{\text{size of error of thing}}{\text{size of thing}} = \frac{|y_{\text{computed}} - y_{\text{true}}|}{|y_{\text{true}}|}$

eg 0.00123 ± 0.00001 is not "correct to 5 digits", rather, 2 digits, rel. err. 10^{-2}, ie 1% err.

Interesting things take a while to compute → is $\varepsilon = 10^{-1}$ ok, or need 10^{-10} ?

In our line of work there is really only one graph that matters:



- useful to measure and/or understand this even for simple tasks
- is crucial for larger tasks! methods differ in graph shapes (rates)

FLATIRON INSTITUTE
Center for Computational Mathematics

Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, ... and convergence parameters of any sub-functions called inside your beast

Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, ...and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

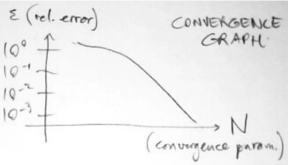
Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, ... and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

Defn. convergence of a method is $\varepsilon(N)$: how rel. err. ε drops as N grows



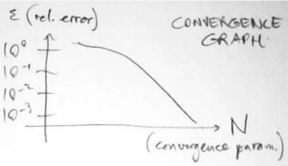
Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

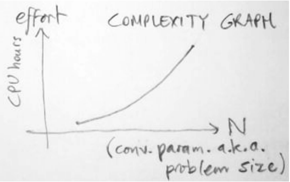
eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, ...and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

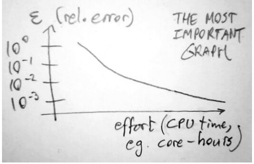
Defn. convergence of a method is $\varepsilon(N)$: how rel. err. ε drops as N grows



$+$



$=$



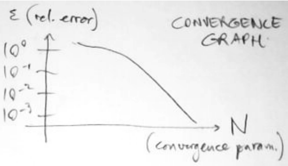
Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

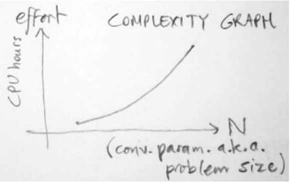
eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, ...and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

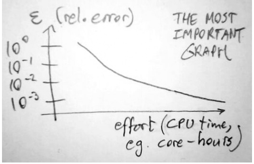
Defn. convergence of a method is $\varepsilon(N)$: how rel. err. ε drops as N grows



$+$



$=$



Eg. say $\varepsilon(N) = cN^{-2}$ ("2nd-order"), but complexity $\mathcal{O}(N^3)$. Qu: cost for 1 *extra* digit?

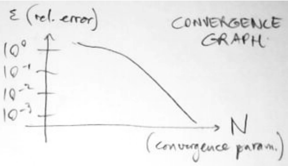
Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

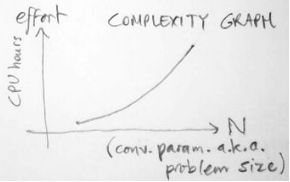
eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, . . . and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

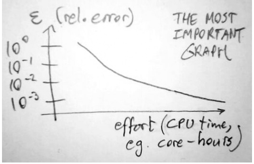
Defn. convergence of a method is $\varepsilon(N)$: how rel. err. ε drops as N grows



+



=



Eg. say $\varepsilon(N) = cN^{-2}$ ("2nd-order"), but complexity $\mathcal{O}(N^3)$. Qu: cost for 1 *extra* digit?

Ans: $\varepsilon \to \varepsilon/10$ needs $N \to \sqrt{10}N$,

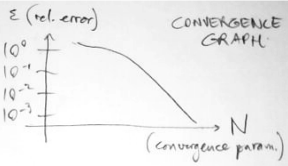
Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

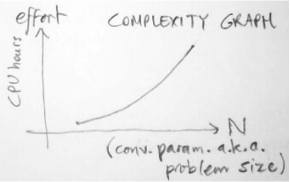
eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, . . . and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

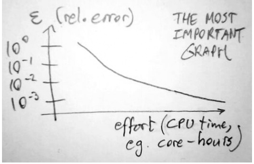
Defn. convergence of a method is $\varepsilon(N)$: how rel. err. ε drops as N grows



$+$



$=$



Eg. say $\varepsilon(N) = cN^{-2}$ ("2nd-order"), but complexity $\mathcal{O}(N^3)$. Qu: cost for 1 *extra* digit?

Ans: $\varepsilon \to \varepsilon/10$ needs $N \to \sqrt{10}N$, which needs effort mult. by $10^{3/2} \approx 32$ times longer run

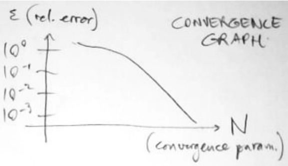
Convergence of a computational routine/method

Often a routine has one (usually many) *convergence parameters*: "dials"

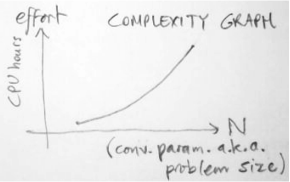
eg how many iterations you run an iterative method, resolution $h = 1/N$ in discretization, number of terms in summing a series, depth/width of a neural net, # of input data, # independent samples you average, size of box (or # particles) in a random simulation, . . . and convergence parameters of any sub-functions called inside your beast

Let's simplify: 1 such param, call it N, with $\lim_{N\to\infty}$ giving true answer

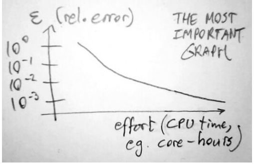
Defn. convergence of a method is $\varepsilon(N)$: how rel. err. ε drops as N grows



$+$



$=$



Eg. say $\varepsilon(N) = cN^{-2}$ ("2nd-order"), but complexity $\mathcal{O}(N^3)$. Qu: cost for 1 *extra* digit?

Ans: $\varepsilon \to \varepsilon/10$ needs $N \to \sqrt{10}N$, which needs effort mult. by $10^{3/2} \approx 32$ times longer run

- some useful methods do *not* converge, eg asymptotic methods

$(\sqrt{\pi}/2)\,\mathrm{erfc}(x) := \int_x^\infty e^{-t^2}dt = e^{-x^2}(1/2x - 1/4x^3 + \dots)$ please don't use $N\to\infty$ terms!

Convergence $\varepsilon(N)$: EXAMPLE I (series)

Toy example: goal compute $y := 1 + \frac{1}{4} + \frac{1}{9} + \cdots = \sum_{k=1}^{\infty} k^{-2}$

```
function y = truncsum(N)
y = 0;
for k=1:N
  y = y + 1/k^2;
end
```

Expected accuracy $\varepsilon(N)$?

Convergence $\varepsilon(N)$: EXAMPLE I (series)

Toy example: goal compute $y := 1 + \frac{1}{4} + \frac{1}{9} + \cdots = \sum_{k=1}^{\infty} k^{-2}$

```
function y = truncsum(N)
y = 0;
for k=1:N
  y = y + 1/k^2;
end
```

N	y_N
10^2	1.63498390018489
10^3	1.64393456668156
10^4	1.64483407184807
10^5	1.64492406689824
10^6	1.64493306684877
10^7	1.64493396684726
10^8	1.64493405783458

Expected accuracy $\varepsilon(N)$?

Quick to experiment with your func:

- "self-convergence" to unknown y_{true} digits "freeze"

Convergence $\varepsilon(N)$: EXAMPLE I (series)

Toy example: goal compute $y := 1 + \frac{1}{4} + \frac{1}{9} + \cdots = \sum_{k=1}^{\infty} k^{-2}$

```
function y = truncsum(N)
y = 0;
for k=1:N
  y = y + 1/k^2;
end
```

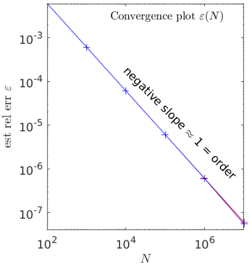
N	y_N
10^2	1.63498390018489
10^3	1.64393456668156
10^4	1.64483407184807
10^5	1.64492406689824
10^6	1.64493306684877
10^7	1.64493396684726
10^8	1.64493405783458

Expected accuracy $\varepsilon(N)$?

Quick to experiment with your func:

- "self-convergence" to unknown y_{true} digits "freeze"
- Rate? Use your best y as y_{true}, plot errors relative to it.

see $\varepsilon(N) \sim cN^{-1}$ 1st-order, algebraic $\rightarrow$ use loglog plot:



math: rigorous tail bnds $\varepsilon(N) \leq \int_N^{\infty} k^{-2} dk = N^{-1}$

rigor unusual; but think, read, measure the rate, compare!

- slow! accelerate? Richardson (etc) extrapolation

Convergence: EXAMPLE II (toy big PCA)

Given $M \times N$ dense matrix A big, eg $M = 40000$ genes, $N = 20000$ samples, 7 GB

Seek $\sigma_1(A) = \sqrt{\lambda_{\max}(A^T A)}$, and assoc. singular vec. $\mathbf{v}_1$ 1st cmpnt, PCA

Convergence: EXAMPLE II (toy big PCA)

Given $M \times N$ dense matrix A big, eg $M = 40000$ genes, $N = 20000$ samples, 7 GB

Seek $\sigma_1(A) = \sqrt{\lambda_{\max}(A^T A)}$, and assoc. singular vec. $\mathbf{v}_1$ 1st cmpnt, PCA

Simple method: power iteration on $A^T A$ takes 14 s; `svd(A)` would be ~ 1 hr

```
v = randn(N,1); v = v/norm(v);
for k=1:30
  v = A'*(A*v);
  vnrm = norm(v); v = v/vnrm;
  sig1est(k) = sqrt(vnrm);
end
```

Convergence: EXAMPLE II (toy big PCA)

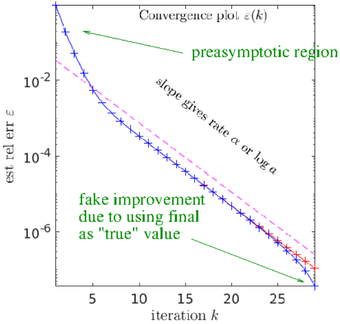
Given $M \times N$ dense matrix A big, eg $M = 40000$ genes, $N = 20000$ samples, 7 GB

Seek $\sigma_1(A) = \sqrt{\lambda_{\max}(A^T A)}$, and assoc. singular vec. $\mathbf{v}_1$ 1st cmpnt, PCA

Simple method: power iteration on $A^T A$ takes 14 s; `svd(A)` would be $\sim$ 1 hr

```
v = randn(N,1); v = v/norm(v);
for k=1:30
  v = A'*(A*v);
  vnrm = norm(v); v = v/vnrm;
  sig1est(k) = sqrt(vnrm);
end
```

plot `abs(sig1est/sig1est(end)-1)` vs param. `k`:



- See $\varepsilon \sim ca^k = ce^{-\alpha k}$ → use log-lin. plot. Called geometric/exponential conv.
- fast (beats any algebraic order) unless $a \approx 1$ ☹. Plenty of theory; we skip

FLATIRON INSTITUTE
Center for Computational Mathematics

Convergence: EXAMPLE II (toy big PCA)

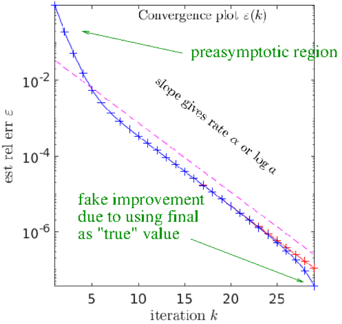
Given $M \times N$ dense matrix A big, eg $M = 40000$ genes, $N = 20000$ samples, 7 GB

Seek $\sigma_1(A) = \sqrt{\lambda_{\max}(A^T A)}$, and assoc. singular vec. $\mathbf{v}_1$ 1st cmpnt, PCA

Simple method: power iteration on $A^T A$ takes 14 s; `svd(A)` would be $\sim$ 1 hr

```
v = randn(N,1); v = v/norm(v);
for k=1:30
  v = A'*(A*v);
  vnrm = norm(v); v = v/vnrm;
  sig1est(k) = sqrt(vnrm);
end
```

plot `abs(sig1est/sig1est(end)-1)` vs param. k:



- See $\varepsilon \sim ca^k = ce^{-\alpha k}$ $\rightarrow$ use log-lin. plot. Called geometric/exponential conv.
- fast (beats any algebraic order) unless $a \approx 1$ ☹. Plenty of theory; we skip

But much better methods exist: *Randomized SVD*, Lanczos ($A^T A$)

$\rightarrow$ lesson is not "code your own methods", rather "test convergence"!

Convergence: EXAMPLE III (stochastic)

Monte Carlo: iid samples y_j drawn from a pdf p
simple task: estimate $\mu := \int yp(y)dy$?

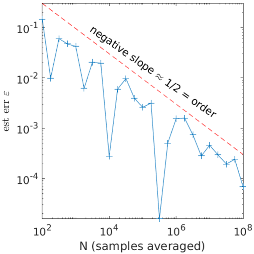
usual estimator $\hat{\mu} = \frac{1}{N}\sum_{j=1}^{N} y_j$ sample mean

Convergence: EXAMPLE III (stochastic)

Monte Carlo: iid samples y_j drawn from a pdf p
simple task: estimate $\mu := \int y p(y) dy$?

usual estimator $\hat{\mu} = \frac{1}{N} \sum_{j=1}^{N} y_j$ sample mean

- convergence $\frac{1}{2}$-order (theory: CLT) $\rightarrow$ v. slow!
- error ε stochastic $\rightarrow$ now conv. accel. not poss.



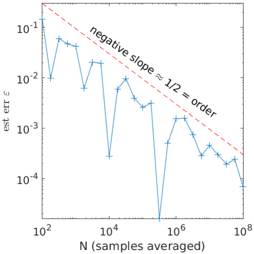
FLATIRON INSTITUTE
Center for Computational Mathematics

Convergence: EXAMPLE III (stochastic)

Monte Carlo: iid samples y_j drawn from a pdf p
simple task: estimate $\mu := \int yp(y)dy$?

usual estimator $\hat{\mu} = \frac{1}{N}\sum_{j=1}^{N} y_j$ sample mean

- convergence $\frac{1}{2}$-order (theory: CLT) $\rightarrow$ v. slow!
- error ε stochastic $\rightarrow$ now conv. accel. not poss.



OTHER CONVERGENCE EXAMPLES

- Taylor series, poly interpolants: exponential $\varepsilon \sim e^{-\alpha N}$ if func analytic

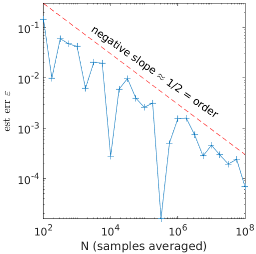
  once you have them, integrate/differentiate *analytically*: spectral methods (Dan, Fri 11:30am)
- Newton methods (root-find in $\mathbb{R}$, or min in $\mathbb{R}^d$): $\varepsilon \sim e^{-cN^2}$ "quadratic"

Convergence: EXAMPLE III (stochastic)

Monte Carlo: iid samples y_j drawn from a pdf p
simple task: estimate $\mu := \int y p(y) dy$?

usual estimator $\hat{\mu} = \frac{1}{N} \sum_{j=1}^{N} y_j$ sample mean

- convergence $\frac{1}{2}$-order (theory: CLT) $\rightarrow$ v. slow!
- error ε stochastic $\rightarrow$ now conv. accel. not poss.



OTHER CONVERGENCE EXAMPLES

- Taylor series, poly interpolants: exponential $\varepsilon \sim e^{-\alpha N}$ if func analytic

  once you have them, integrate/differentiate *analytically*: spectral methods (Dan, Fri 11:30am)
- Newton methods (root-find in $\mathbb{R}$, or min in $\mathbb{R}^d$): $\varepsilon \sim e^{-cN^2}$ "quadratic"

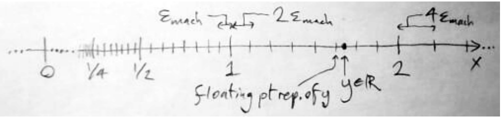
Point isn't to memorize rates of methods: rather *measure* them (type & prefactor) by habit in any routine you use/write

Then you can pick a good N to get acceptable ε, trust results

FLATIRON INSTITUTE Center for Computational Mathematics

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



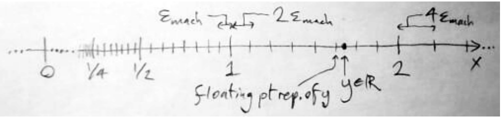
$\varepsilon_{\mathrm{mach}} \approx$ 1.1e−16 double (64bit)

$\varepsilon_{\mathrm{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{\mathrm{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ 1.1e−16 double (64bit)

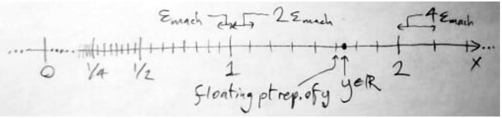
$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}}$

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ `1.1e-16` double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

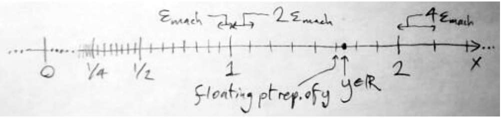
$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}}$

eg, in double: `(1 + 1e-16) - 1 = ?`

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ `1.1e-16` double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

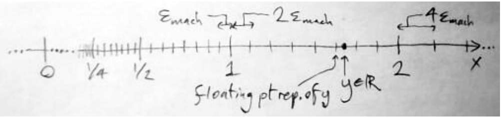
$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

Represents any real to rel. err. $\varepsilon \le \varepsilon_{\text{mach}}$; all arith. done to rel. err. $\varepsilon \le \varepsilon_{\text{mach}}$

eg, in double: `(1 + 1e-16) - 1 = ? 0`   And: `(1 - 1e-16) - 1 = ?`

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ 1.1e−16 double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

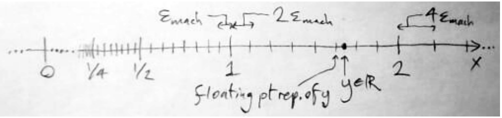
$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}}$

eg, in double: `(1 + 1e-16) - 1 =? 0` And: `(1 - 1e-16) - 1 =? -1.11022302462516e-16`

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ `1.1e-16` double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}}$

eg, in double: `(1 + 1e-16) - 1 = ? 0` And: `(1 - 1e-16) - 1 = ? -1.11022302462516e-16`

A) Most common way $\varepsilon_{\text{mach}}$ amplified is subtraction "catastrophic cancellation"

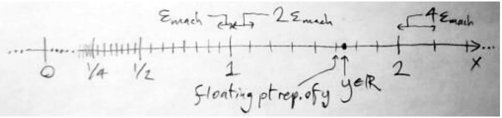
eg, by querying *values* of $f(x)$, estim. $f'(x)$?

let's use simplest formula $\frac{f(x+h)-f(x)}{h}$:

h	err. in f'	dominant cause?
10^{-4}	10^{-4}	1st-order conv.
10^{-8}	10^{-8}	(balanced causes)
10^{-12}	10^{-4}	$2\varepsilon_{\text{mach}}/h$ "CC" ☹

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ `1.1e-16` double (64bit)

$\varepsilon_{\text{mach}} \approx$ `6e-8` single (32bit), GPU/TPU

$\varepsilon_{\text{mach}} \approx$ `5e-4` "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}}$

eg, in double: `(1 + 1e-16) - 1 = ? 0`   And: `(1 - 1e-16) - 1 = ? -1.11022302462516e-16`

A) Most common way $\varepsilon_{\text{mach}}$ amplified is subtraction "catastrophic cancellation"

eg, by querying *values* of $f(x)$, estim. $f'(x)$?

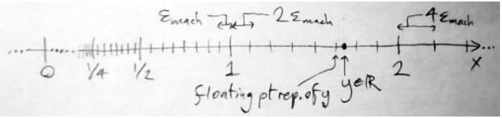
let's use simplest formula $\frac{f(x+h)-f(x)}{h}$:

Better: use several $p > 2$ values to get pth order!

h	err. in f'	dominant cause?
10^{-4}	10^{-4}	1st-order conv.
10^{-8}	10^{-8}	(balanced causes)
10^{-12}	10^{-4}	$2\varepsilon_{\text{mach}}/h$ "CC" ☹

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{mach} \approx$ 1.1e−16 double (64bit)

$\varepsilon_{mach} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{mach} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{mach}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{mach}}$

eg, in double: `(1 + 1e-16) - 1 =? 0` And: `(1 - 1e-16) - 1 =? -1.1102230246251 6e-16`

A) Most common way ε_{mach} amplified is subtraction "catastrophic cancellation"

eg, by querying *values* of $f(x)$, estim. $f'(x)$?

let's use simplest formula $\frac{f(x+h)-f(x)}{h}$:

Better: use several $p > 2$ values to get pth order!

h	err. in f'	dominant cause?
10^{-4}	10^{-4}	1st-order conv.
10^{-8}	10^{-8}	(balanced causes)
10^{-12}	10^{-4}	$2\varepsilon_{mach}/h$ "CC" ☹

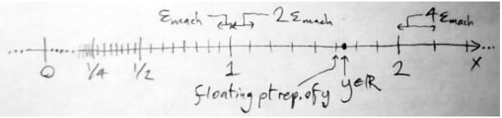
B) Even without subtraction (or equiv), err. can accumulate:

eg recall $\sum_{k=1}^{N} k^{-2}$:

N	y_N
10^8	1.6449340578345 8
10^9	?

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ 1.1e−16 double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \le \varepsilon_{\text{mach}}}$

eg, in double: `(1 + 1e-16) - 1 =? 0` And: `(1 - 1e-16) - 1 =? -1.1102230246251 6e-16`

A) Most common way $\varepsilon_{\text{mach}}$ amplified is subtraction "catastrophic cancellation"

eg, by querying *values* of $f(x)$, estim. $f'(x)$?

let's use simplest formula $\frac{f(x+h)-f(x)}{h}$:

Better: use several $p > 2$ values to get pth order!

h	err. in f'	dominant cause?
10^{-4}	10^{-4}	1st-order conv.
10^{-8}	10^{-8}	(balanced causes)
10^{-12}	10^{-4}	$2\varepsilon_{\text{mach}}/h$ "CC" ☹

B) Even without subtraction (or equiv), err. can accumulate:

eg recall $\sum_{k=1}^{N} k^{-2}$:

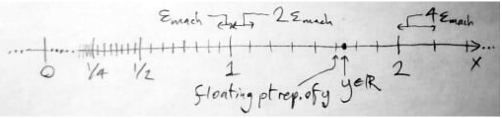
N	y_N
10^8	1.6449340578345 8
10^9	1.6449340578345 8

Here $\varepsilon \approx \sqrt{\varepsilon_{\text{mach}}}$, bad! ☹

fix?

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ 1.1e−16 double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

$\boxed{\text{Represents any real to rel. err. } \varepsilon \leq \varepsilon_{\text{mach}}\text{; all arith. done to rel. err. } \varepsilon \leq \varepsilon_{\text{mach}}}$

eg, in double: `(1 + 1e-16) - 1 = ? 0` And: `(1 - 1e-16) - 1 = ? -1.1102230246251 6e-16`

A) Most common way $\varepsilon_{\text{mach}}$ amplified is subtraction "catastrophic cancellation"

eg, by querying *values* of $f(x)$, estim. $f'(x)$?

let's use simplest formula $\frac{f(x+h)-f(x)}{h}$:

Better: use several $p > 2$ values to get pth order!

h	err. in f'	dominant cause?
10^{-4}	10^{-4}	1st-order conv.
10^{-8}	10^{-8}	(balanced causes)
10^{-12}	10^{-4}	$2\varepsilon_{\text{mach}}/h$ "CC" ☹

B) Even without subtraction (or equiv), err. can accumulate:

eg recall $\sum_{k=1}^{N} k^{-2}$:

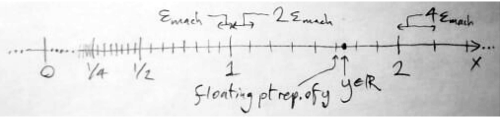
N	y_N
10^8	1.6449340578345 8
10^9	1.6449340578345 8

Here $\varepsilon \approx \sqrt{\varepsilon_{\text{mach}}}$, bad! ☹

fix? sum small to large, most stable

Floating-point representation, rounding error

So far rounding error basically irrelevant. Now let's face its consequences:



$\varepsilon_{\text{mach}} \approx$ 1.1e−16 double (64bit)

$\varepsilon_{\text{mach}} \approx$ 6e−8 single (32bit), GPU/TPU

$\varepsilon_{\text{mach}} \approx$ 5e−4 "half" (16bit), GPU/TPU

Represents any real to rel. err. $\varepsilon \leq \varepsilon_{\text{mach}}$; all arith. done to rel. err. $\varepsilon \leq \varepsilon_{\text{mach}}$

eg, in double: `(1 + 1e-16) - 1 =? 0` And: `(1 - 1e-16) - 1 =? -1.1102230246251 6e-16`

A) Most common way $\varepsilon_{\text{mach}}$ amplified is subtraction "catastrophic cancellation"

eg, by querying *values* of $f(x)$, estim. $f'(x)$?

let's use simplest formula $\frac{f(x+h)-f(x)}{h}$:

Better: use several $p > 2$ values to get pth order!

h	err. in f'	dominant cause?
10^{-4}	10^{-4}	1st-order conv.
10^{-8}	10^{-8}	(balanced causes)
10^{-12}	10^{-4}	$2\varepsilon_{\text{mach}}/h$ "CC" ☹

B) Even without subtraction (or equiv), err. can accumulate:

eg recall $\sum_{k=1}^{N} k^{-2}$:

N	y_N
10^8	1.6449340578345 8
10^9	1.6449340578345 8

Here $\varepsilon \approx \sqrt{\varepsilon_{\text{mach}}}$, bad! ☹

fix? sum small to large, most stable

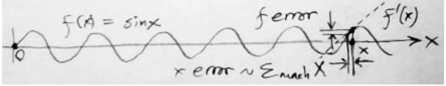
Usually stoch. $\varepsilon \sim \sqrt{\#\ \text{flops}}\,\varepsilon_{\text{mach}}$

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



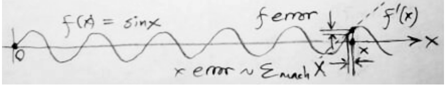
Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

→ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

→ result garbage, just via *input variation*

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\rightarrow$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\rightarrow$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

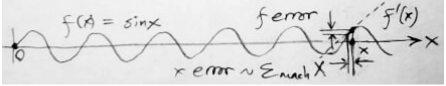
$$\kappa(x) := \left| \frac{x f'(x)}{f(x)} \right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\rightarrow$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\rightarrow$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

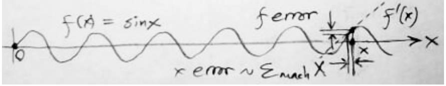
gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

FLATIRON INSTITUTE
Center for Computational Mathematics

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\rightarrow$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\rightarrow$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

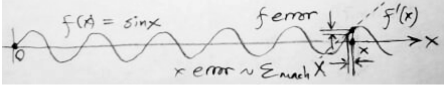
why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

Eg $f(x) = \sin(x)$, $\kappa(x) = |x \cot x|$ $x = 10^{16} \Rightarrow \kappa$ typ. $\geq 10^{16}$

the problem is ill-conditioned: meaningless to demand *any* digits in double-prec!

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\rightarrow$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\rightarrow$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left| \frac{x f'(x)}{f(x)} \right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

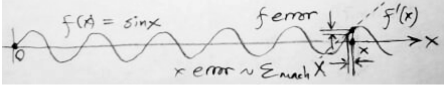
Eg $f(x) = \sin(x)$, $\kappa(x) = |x \cot x|$ $\qquad x = 10^{16} \Rightarrow \kappa$ typ. $\geq 10^{16}$

the problem is ill-conditioned: meaningless to demand *any* digits in double-prec!

eg $x = 10^5 \Rightarrow \kappa$ typ. $\geq 10^5$ $\quad \Rightarrow$ expect $\varepsilon \sim$?

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\to$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\to$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left| \frac{xf'(x)}{f(x)} \right|$$

$\longleftarrow$ sensitivity to rel. change in x

$\longleftarrow$ converts abs. to rel. error

gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

Eg $f(x) = \sin(x)$, $\kappa(x) = |x \cot x|$ $\quad x = 10^{16} \Rightarrow \kappa$ typ. $\geq 10^{16}$

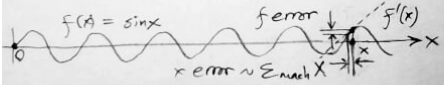
the problem is ill-conditioned: meaningless to demand *any* digits in double-prec!

eg $x = 10^5 \Rightarrow \kappa$ typ. $\geq 10^5$ $\quad \Rightarrow$ expect $\varepsilon \sim$? 10^{-11}

FLATIRON INSTITUTE Center for Computational Mathematics

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\rightarrow$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\rightarrow$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left| \frac{xf'(x)}{f(x)} \right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

Eg $f(x) = \sin(x)$, $\kappa(x) = |x \cot x|$ $x = 10^{16} \Rightarrow \kappa$ typ. $\geq 10^{16}$

the problem is ill-conditioned: meaningless to demand *any* digits in double-prec!

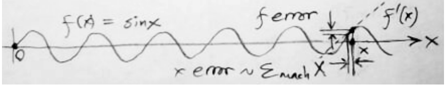
eg $x = 10^5 \Rightarrow \kappa$ typ. $\geq 10^5$ $\Rightarrow$ expect $\varepsilon \sim ?\ 10^{-11}$

eg $x = 1 \Rightarrow \kappa(x) = 0.64$ $\Rightarrow$ good method should get $\varepsilon \approx ?$

FLATIRON INSTITUTE
Center for Computational Mathematics

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\to$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\to$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

Eg $f(x) = \sin(x)$, $\kappa(x) = |x \cot x|$ $x = 10^{16} \Rightarrow \kappa$ typ. $\geq 10^{16}$

the problem is ill-conditioned: meaningless to demand *any* digits in double-prec!

eg $x = 10^5 \Rightarrow \kappa$ typ. $\geq 10^5$ $\Rightarrow$ expect $\varepsilon \sim$? 10^{-11}

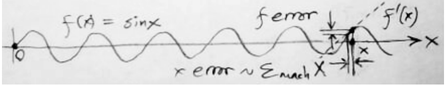
eg $x = 1 \Rightarrow \kappa(x) = 0.64$ $\Rightarrow$ good method should get $\varepsilon \approx$? $\varepsilon_{\text{mach}}$

eg $x = \pi$?

FLATIRON INSTITUTE
Center for Computational Mathematics

For which tasks is it reasonable to demand accuracy?

Qu: is `sin(1e16)` reasonable to compute accurately (in double prec.)?



Ans: no! $x = 10^{16}$, floating rel. err. $\varepsilon_{\text{mach}}$

$\rightarrow$ abs. err. $10^{16}\varepsilon_{\text{mach}} \approx 1.1 = \mathcal{O}(1)$ wiggle

$\rightarrow$ result garbage, just via *input variation*

Defn. (relative) condition number of *task* "eval. $f(x)$" is

$$\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$$

$\leftarrow$ sensitivity to rel. change in x

$\leftarrow$ converts abs. to rel. error

gives **Rule:** for this task, can only demand rel. err. at best $\varepsilon \approx \kappa\varepsilon_{\text{mach}}$

why? look at picture: ε must exceed change in f due to $\varepsilon_{\text{mach}}$ rel. err. in input x

Eg $f(x) = \sin(x)$, $\kappa(x) = |x \cot x|$ $\quad x = 10^{16} \Rightarrow \kappa$ typ. $\geq 10^{16}$

the problem is ill-conditioned: meaningless to demand *any* digits in double-prec!

eg $x = 10^5 \Rightarrow \kappa$ typ. $\geq 10^5 \quad \Rightarrow$ expect $\varepsilon \sim$? 10^{-11}

eg $x = 1 \Rightarrow \kappa(x) = 0.64 \quad \Rightarrow$ good method should get $\varepsilon \approx$? $\varepsilon_{\text{mach}}$

eg $x = \pi$? $\Rightarrow \kappa(x) = \infty$, can't demand *relative* acc. (merely abs. accuracy)

Stability of an algorithm (method) for some task

Recap: task "eval. $f(x)$" has cond. # $\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$ indep. of any method

Stability of an algorithm (method) for some task

Recap: task "eval. $f(x)$" has cond. # $\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$ indep. of any method

Defn. A *method* for this task called backward stable if returns an exact answer $f(\tilde{x})$ for *some* perturbed data $\tilde{x}$ with $|\tilde{x} - x|/|x| = \mathcal{O}(\varepsilon_{\text{mach}})$

- modern notion of stability here $\mathcal{O}$ implies some "small" const, eg $\lesssim 10^2$

Thus: backward stable $\Rightarrow$ rel. err. $\varepsilon = \mathcal{O}(\kappa \varepsilon_{\text{mach}})$ by rule: can't demand more!

Stability of an algorithm (method) for some task

Recap: task "eval. $f(x)$" has cond. # $\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$ indep. of any method

Defn. A *method* for this task called backward stable if returns an exact answer $f(\tilde{x})$ for *some* perturbed data $\tilde{x}$ with $|\tilde{x} - x|/|x| = \mathcal{O}(\varepsilon_{\text{mach}})$

- modern notion of stability here $\mathcal{O}$ implies some "small" const, eg $\lesssim 10^2$

Thus: backward stable $\Rightarrow$ rel. err. $\varepsilon = \mathcal{O}(\kappa\varepsilon_{\text{mach}})$ by rule: can't demand more!

1) Consequences for physical simulations (nonlinear ODEs, PDEs...)

Eg, task: solve ODE

$$\begin{cases} u' = F(t,u) & \text{for } 0 \le t \le T \\ u(0) = x & \text{initial condition} \end{cases}$$

Output "$f(x)$" is final state $u(T)$

Stability of an algorithm (method) for some task

Recap: task "eval. $f(x)$" has cond. # $\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$ indep. of any method

Defn. A *method* for this task called backward stable if returns an exact answer $f(\tilde{x})$ for *some* perturbed data $\tilde{x}$ with $|\tilde{x} - x|/|x| = \mathcal{O}(\varepsilon_{\text{mach}})$

- modern notion of stability

here $\mathcal{O}$ implies some "small" const, eg $\lesssim 10^2$

Thus: backward stable $\Rightarrow$ rel. err. $\varepsilon = \mathcal{O}(\kappa\varepsilon_{\text{mach}})$ by rule: can't demand more!

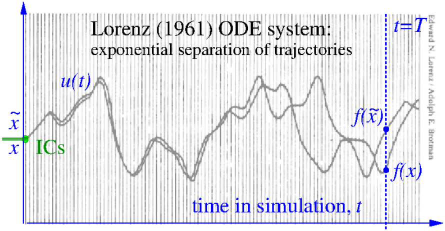
1) Consequences for physical simulations (nonlinear ODEs, PDEs. . .)

Eg, task: solve ODE

$$\begin{cases} u' = F(t,u) & \text{for } 0 \le t \le T \\ u(0) = x & \text{initial condition} \end{cases}$$

Output "$f(x)$" is final state $u(T)$

κ = sensitivity to IC



Stability of an algorithm (method) for some task

Recap: task "eval. $f(x)$" has cond. # $\kappa(x) := \left|\frac{xf'(x)}{f(x)}\right|$ indep. of any method

Defn. A *method* for this task called backward stable if returns an exact answer $f(\tilde{x})$ for *some* perturbed data $\tilde{x}$ with $|\tilde{x} - x|/|x| = \mathcal{O}(\varepsilon_{\text{mach}})$

- modern notion of stability here $\mathcal{O}$ implies some "small" const, eg $\lesssim 10^2$

Thus: backward stable $\Rightarrow$ rel. err. $\varepsilon = \mathcal{O}(\kappa\varepsilon_{\text{mach}})$ by rule: can't demand more!

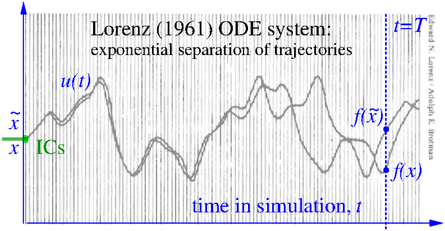
1) Consequences for physical simulations (nonlinear ODEs, PDEs...)

Eg, task: solve ODE

$$\begin{cases} u' = F(t,u) & \text{for } 0 \le t \le T \\ u(0) = x & \text{initial condition} \end{cases}$$

Output "$f(x)$" is final state $u(T)$

κ = sensitivity to IC



- common that $\kappa \sim e^{\lambda T}$ (Lyapunov exponent $\lambda > 0$, chaos, eg n-body sims.)
- then even stable solver must soon lose all accurate digits see: shadowing
- meaning of long-T numerics is only *statistical* (correlations, manifold, etc)

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms . . . don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned?

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms ... don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ?

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms . . . don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ? no: catastrophic cancellation!

. . . w/o clarity on conditioning vs stability, may conclude ill-conditioned problem. Not so!

Suggest stable methods?

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms . . . don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ? no: catastrophic cancellation!

. . . w/o clarity on conditioning vs stability, may conclude ill-conditioned problem. Not so!

Suggest stable methods? i) `2*sin(x/2)^2`

FLATIRON INSTITUTE
Center for Computational Mathematics

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms . . . don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ? no: catastrophic cancellation!

. . . w/o clarity on conditioning vs stability, may conclude ill-conditioned problem. Not so!

Suggest stable methods? i) `2*sin(x/2)^2` ii) Taylor series (how many terms? conv. . .)

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms . . . don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ? no: catastrophic cancellation!

. . . w/o clarity on conditioning vs stability, may conclude ill-conditioned problem. Not so!

Suggest stable methods? i) `2*sin(x/2)^2` ii) Taylor series (how many terms? conv. . .)

3) Linear systems: solve $A\mathbf{c} = \mathbf{b}$, square $N \times N$ needs whole lecture

Task is $\mathbf{f}(\mathbf{b}) =$ "$\mathbf{c}$ solving $A\mathbf{c} = \mathbf{b}$" brain hurts because $\mathbf{b}$ is input, $\mathbf{c}$ is output!

FLATIRON INSTITUTE
Center for Computational Mathematics

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms ... don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ? no: catastrophic cancellation!

... w/o clarity on conditioning vs stability, may conclude ill-conditioned problem. Not so!

Suggest stable methods? i) `2*sin(x/2)^2` ii) Taylor series (how many terms? conv...)

3) Linear systems: solve $A\mathbf{c} = \mathbf{b}$, square $N \times N$ needs whole lecture

Task is $\mathbf{f}(\mathbf{b}) =$ "$\mathbf{c}$ solving $A\mathbf{c} = \mathbf{b}$" brain hurts because $\mathbf{b}$ is input, $\mathbf{c}$ is output!

Stable alg: gives $\tilde{\mathbf{c}}$ solving $A\tilde{\mathbf{c}} = \tilde{\mathbf{b}}$ exactly, where $\frac{\|\tilde{\mathbf{b}}-\mathbf{b}\|}{\|\mathbf{b}\|} = \mathcal{O}(\varepsilon_{\text{mach}})$

Defn. relative residual of $\tilde{\mathbf{c}}$ is $\frac{\|A\tilde{\mathbf{c}}-\mathbf{b}\|}{\|\mathbf{b}\|}$:

FLATIRON INSTITUTE
Center for Computational Mathematics

Stability of algorithms: more examples

Recap: (backward) stable if "exact answer to nearly the right question"

2) There are unstable algorithms . . . don't use them!

Eg eval. $f(x) = 1 - \cos(x)$, for $|x| \ll 1$ we all know $f(x) = x^2/2 + \mathcal{O}(x^4)$

ALWAYS FIRST ASK: Is *task* (problem) well-conditioned? yes, $\kappa \approx 2$

Now, methods: naive code `1-cos(x)` stable ? no: catastrophic cancellation!

. . . w/o clarity on conditioning vs stability, may conclude ill-conditioned problem. Not so!

Suggest stable methods? i) `2*sin(x/2)^2` ii) Taylor series (how many terms? conv. . .)

3) Linear systems: solve $A\mathbf{c} = \mathbf{b}$, square $N \times N$ needs whole lecture

Task is $\mathbf{f}(\mathbf{b}) =$ "$\mathbf{c}$ solving $A\mathbf{c} = \mathbf{b}$" brain hurts because $\mathbf{b}$ is input, $\mathbf{c}$ is output!

Stable alg: gives $\tilde{\mathbf{c}}$ solving $A\tilde{\mathbf{c}} = \tilde{\mathbf{b}}$ exactly, where $\frac{\|\tilde{\mathbf{b}}-\mathbf{b}\|}{\|\mathbf{b}\|} = \mathcal{O}(\varepsilon_{\text{mach}})$

Defn. relative residual of $\tilde{\mathbf{c}}$ is $\frac{\|A\tilde{\mathbf{c}}-\mathbf{b}\|}{\|\mathbf{b}\|}$: $\boxed{\text{Stable alg} \Leftrightarrow \text{Rel. resid. } \mathcal{O}(\varepsilon_{\text{mach}})}$

- even a stable alg doesn't mean $\tilde{\mathbf{c}}$ is close to $\mathbf{c}$ . . .

Let's demo a classic *unstable algorithm* . . .

FLATIRON INSTITUTE Center for Computational Mathematics

MATLAB demo: unstable vs stable linear solve

```
>> c = [1;2;3];                              % "true" solution column vector
>> A = ones(3,3) + 1e-14*rand(3,3)           % system matrix (precisely: ill-cond.)
A =       1.00000000000001         1.00000000000001                        1
          1.00000000000001         1.00000000000001         1.00000000000001
                         1                        1         1.00000000000001
>> b = A*c;                                  % make data (input to solver)
```

MATLAB demo: unstable vs stable linear solve

```
>> c = [1;2;3];                          % "true" solution column vector
>> A = ones(3,3) + 1e-14*rand(3,3)       % system matrix (precisely: ill-cond.)
A =       1.00000000000001      1.00000000000001                     1
          1.00000000000001      1.00000000000001      1.00000000000001
                         1                     1      1.00000000000001
>> b = A*c;                              % make data (input to solver)
```

Now let's do some solving. . .

```
>> ct = inv(A)*b;                     % classic pitfall, may be unstable
>> norm(A*ct-b) / norm(b)             % rel resid terrible, proving it's unstable!
                  0.046875
```

MATLAB demo: unstable vs stable linear solve

```
>> c = [1;2;3];                            % "true" solution column vector
>> A = ones(3,3) + 1e-14*rand(3,3)         % system matrix (precisely: ill-cond.)
A =        1.00000000000001          1.00000000000001                        1
           1.00000000000001          1.00000000000001         1.00000000000001
                          1                         1         1.00000000000001
>> b = A*c;                                % make data (input to solver)
```

Now let's do some solving...

```
>> ct = inv(A)*b;                        % classic pitfall, may be unstable
>> norm(A*ct-b) / norm(b)                % rel resid terrible, proving it's unstable!
                  0.046875

>> ct = linsolve(A,b);                   % use (backward) stable solver
>> norm(A*ct-b) / norm(b)                % rel resid O(e_mach): must be if stable
      8.54650082837135e-17
```

MATLAB demo: unstable vs stable linear solve

```
>> c = [1;2;3];                              % "true" solution column vector
>> A = ones(3,3) + 1e-14*rand(3,3)           % system matrix (precisely: ill-cond.)
A =        1.00000000000001          1.00000000000001                         1
           1.00000000000001          1.00000000000001          1.00000000000001
                          1                         1          1.00000000000001
>> b = A*c;                                  % make data (input to solver)
```

Now let's do some solving. . .

```
>> ct = inv(A)*b;                        % classic pitfall, may be unstable
>> norm(A*ct-b) / norm(b)                % rel resid terrible, proving it's unstable!
                  0.046875

>> ct = linsolve(A,b);                   % use (backward) stable solver
>> norm(A*ct-b) / norm(b)                % rel resid O(e_mach): must be if stable
      8.54650082837135e-17

>> norm(ct-c) / norm(c)                  % rel err in soln? huge, but that's ok...
        0.0426438890711514
```

FLATIRON INSTITUTE
Center for Computational Mathematics

MATLAB demo: unstable vs stable linear solve

```
>> c = [1;2;3];                          % "true" solution column vector
>> A = ones(3,3) + 1e-14*rand(3,3)       % system matrix (precisely: ill-cond.)
A =        1.00000000000001        1.00000000000001                       1
           1.00000000000001        1.00000000000001        1.00000000000001
                          1                       1        1.00000000000001
>> b = A*c;                              % make data (input to solver)
```

Now let's do some solving...

```
>> ct = inv(A)*b;                    % classic pitfall, may be unstable
>> norm(A*ct-b) / norm(b)            % rel resid terrible, proving it's unstable!
                 0.046875

>> ct = linsolve(A,b);               % use (backward) stable solver
>> norm(A*ct-b) / norm(b)            % rel resid O(e_mach): must be if stable
      8.54650082837135e-17

>> norm(ct-c) / norm(c)              % rel err in soln? huge, but that's ok...
        0.0426438890711514
```

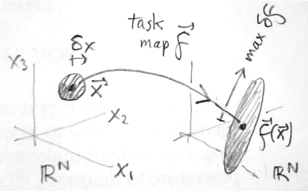
If time: here's one stable way to store a soln operator...

```
[U,S,V] = svd(A); W = diag(1./diag(S))*U';    % inv(A)=VW, need two factors
ct = V*(W*b);                                 % apply them to any RHS
norm(A*ct-b) / norm(b)                        % rel resid again O(e_mach)
      2.83455365181694e-16
```

If time: conditioning of linear systems

For vector map $\mathbf{f}(\mathbf{x})$, condition number is

$$\kappa(\mathbf{x}) := \lim_{\delta x \to 0} \sup_{\|\delta \mathbf{x}\| \le \delta \mathbf{x}} \frac{\|\delta \mathbf{f}\| / \|\mathbf{f}\|}{\|\delta \mathbf{x}\| / \|\mathbf{x}\|}$$



- Lin. solve task: can show $\kappa(\mathbf{b}) \le \kappa(A) := \|A\|\,\|A^{-1}\| = \frac{\sigma_1(A)}{\sigma_N(A)}$ or ∞

Consequence for how accurate solution $\tilde{\mathbf{c}}$ is? Let $\varepsilon = \frac{\|\tilde{\mathbf{c}} - \mathbf{c}\|}{\|\mathbf{c}\|}$ rel. soln. err.

Now recall: stable solver (best you can demand) has $\varepsilon = \mathcal{O}(\kappa \varepsilon_{\text{mach}})$

if A ill-cond, natural that $\mathbf{c}$ floppy in certain directions, since residual small

- Idea useful in inverse problems: replace $\varepsilon_{\text{mach}}$ by meas. err; reverse above pic!

Idea to sample *all* $\mathbf{c}$ consistent w/ small residual $\to$ Bayes Inv. Prob. (Bob, Fri 9:10am)

Recap

- Convergence rates (type & prefactor) key to *measure* and understand
- Finite-precision $\varepsilon_{\text{mach}}$ can be amplified by catastrophic cancellation
- Before methods, first understand condition # of your *problem*

  condition number of problem combines with $\varepsilon_{\text{mach}}$ to limit accuracy of *any* method
- Stable methods: solve exactly some $\varepsilon_{\text{mach}}$-perturbation of problem

  "(un)stable" vs "ill-conditioned" have precise definitions: learn and use!

  check for unstable method and avoid
- For linear systems: "stable" $\Leftrightarrow$ finds relative residual $\mathcal{O}(\varepsilon_{\text{mach}})$

References for today material

- *Numerical Methods*. Anne Greenbaum & Tim Chartier. book (2012)
- *Numerical Linear Algebra*. Trefethen & Bau. book (1997)

Convergence acceleration and all-round fun:

- *The SIAM 100-Digit Challenge*. book (2004)

Randomized SVD, PCA, and big matrix factorizations:

- Halko, Martinsson & Tropp. *SIAM Rev.* **53**(2) 217–288 (2011)
- Martinsson's slides at `http://users.oden.utexas.edu/~pgm`

I will host slides at `https://users.flatironinstitute.org/~ahb`
(also see: 2019 FWAM on interpolation & quadrature; Burns on PDE)

Starting new Sci. Comput. Seminar & Concepts, 9:45am Tues, 3rd fl.

(fortnightly from 10/26, see Indico)

THANK-YOU!

FLATIRON INSTITUTE
Center for Computational Mathematics