Numerical Stability

Raymond J. Spiteri

Lecture Notes for CMPT 898:
Numerical Software

University of Saskatchewan

January 11, 2013

Objectives

- Problem conditioning and numerical stability
-
-

Problem Conditioning

In a very abstract sense, solving a problem is like evaluating a function

$$y = f(x).$$

Here, x represents the input to the problem (the data), f represents the "problem" itself, and y represents its solution.

We are interested in studying the effect on y when a given x is perturbed slightly.

If small changes in x lead to small changes in y, we say the problem is well-conditioned.

If small changes in x lead to large changes in y, we say the problem is ill-conditioned.

Problem Conditioning

Of course what constitutes "large" or "small" may depend on the problem.

Although we are sometimes forced to do otherwise, it only makes mathematical sense to solve well-conditioned problems.

Because floating-point arithmetic used by computers introduces relative errors not absolute errors, we define conditioning in terms of a relative condition number.

Relative Condition Number

Let δx denote a small perturbation of x and let

$$\delta f = f(x + \delta x) - f(x)$$

be the corresponding perturbation in f.
Then, the *relative condition number* $\kappa = \kappa(x)$ is defined to be

$$\kappa(x) = \lim_{\delta \to 0} \max_{\|\delta x\| \leq \delta} \left(\frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right).$$

Or, if you just assume δx and δf are infinitesimal

$$\kappa(x) = \max_{\delta x} \left(\frac{\|\delta f\|}{\|f(x)\|} \Big/ \frac{\|\delta x\|}{\|x\|} \right).$$

Thus $\kappa(x)$ is the maximum value of the ratio "relative change in f" to "relative change in x".

Relative Condition Number

If f has a derivative , we can write

$$\frac{\delta f}{\delta x} = \mathbf{J}(x),$$

where $\mathbf{J} = \frac{\partial f_i}{\partial x_j}$ is known as the Jacobian of f at x.

For example, suppose

$$f(x_1, x_2, x_3) = \begin{pmatrix} x_1 x_2 + \sin(x_3) + x_1^2 \\ 7 + e^{x_2} \end{pmatrix}.$$

Then,

$$\begin{aligned} \mathbf{J} &= \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix} \\ &= \begin{bmatrix} x_2 + 2x_1 & x_1 & \cos(x_3) \\ 0 & e^{x_2} & 0 \end{bmatrix}. \end{aligned}$$

Relative Condition Number

Note 1. $\delta f \approx \mathbf{J}(x)\delta x$ *with* $\delta f = \mathbf{J}(x)\delta x$ *in the limit* $\|\delta x\| \to 0$.

In terms of $\mathbf{J}$,

$$\kappa(x) = \frac{\|\mathbf{J}(x)\|}{\|f(x)\|/\|x\|}.$$

We say a problem is *well-conditioned* if κ is small (e.g., $\approx 1, 10, 10^2$), and *ill-conditioned* if it is large (e.g., $\approx 10^6, 10^{14}$).

Note 2. *What constitutes "large" depends on the precision you are working in!*

A general rule of thumb is that if $\kappa = 10^p$, then you cannot really trust the last p digits of the floating-point representation of your answer.

Relative Condition Number

In single precision, $\epsilon_{\text{machine}} \approx 10^{-8}$; so $\kappa = 10^6$ is pretty ill-conditioned: only the first 2 digits of the answer are reliable (this is OK for some applications!).

But, in double precision, where $\epsilon_{\text{machine}} \approx 10^{-16}$, $\kappa = 10^6$ is not such a big deal.

Example 1: DIVISION BY 2
Consider the (trivial) problem of dividing a number by 2. This can be described by the function

$$f : x \to \frac{x}{2}.$$

So,

$$\mathbf{J} = \left[\frac{\partial f}{\partial x}\right] = \frac{1}{2},$$

and

$$\kappa = \frac{\|\mathbf{J}\|}{\|f(x)\|/\|x\|} = \frac{\frac{1}{2}}{\frac{1}{2}|x|/|x|} = 1.$$

So this is an optimally well-conditioned problem!

Relative Condition Number

Example 2: SUBTRACTION
Consider the problem of subtracting two numbers. This can be described by the function

$$f(x) : (x_1, x_2) \to x_1 - x_2.$$

For simplicity, let $\|\cdot\| = \|\cdot\|_\infty$. Then,

$$\mathbf{J} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 1 & -1 \end{bmatrix} \implies \|\mathbf{J}\|_\infty = 2,$$

and

$$\kappa = \frac{\|\mathbf{J}\|}{\|f(x)\|/\|x\|} = \frac{2}{|x_1 - x_2|/\max\{|x_1|, |x_2|\}}.$$

So we see κ is large if $|x_1 - x_2|$ is small; i.e., $x_1 \approx x_2$. This leads us to the well-known result that subtraction of nearly equal quantities leads to large (cancellation) errors in the result.

Relative Condition Number

Example 3: COMPUTING EIGENVALUES OF A NON-SYMMETRIC MATRIX

This problem is often ill-conditioned.

For example, consider

$$\mathbf{A} = \begin{bmatrix} 1 & 1000 \\ 0 & 1 \end{bmatrix}$$

and

$$\tilde{\mathbf{A}} = \begin{bmatrix} 1 & 1000 \\ 0.001 & 1 \end{bmatrix}.$$

The eigenvalues of $\mathbf{A}$ are $\{1, 1\}$, whereas those of $\tilde{\mathbf{A}}$ are $\{0, 2\}$. *(verify!)*

$\rightarrow$ a large change in the output (eigenvalues) for a small change ($\sim 10^{-3}$) of the input ($\mathbf{A} \rightarrow \tilde{\mathbf{A}}$).

Note 3. *On the other hand, if* $\mathbf{A}$ *is symmetric (or more generally, if it is* normal[1]*) then its eigenvalues are well-conditioned.*

For such matrices, it can be shown that if λ and $\lambda+\delta\lambda$ are the eigenvalues of $\mathbf{A}$ and $\mathbf{A}+\delta\mathbf{A}$ respectively, then

$$|\delta\lambda| \leq \|\delta\mathbf{A}\|_2.$$

$\rightarrow$ using the 2-norm, we can take

$$\|\mathbf{J}\| = \max \left\| \frac{\delta f}{\delta x} \right\| = \max \frac{|\delta\lambda|}{\|\delta\mathbf{A}\|} = 1,$$

and thus

$$\kappa = \frac{1}{|\lambda|/\|\mathbf{A}\|_2} = \|\mathbf{A}\|_2/|\lambda|.$$

[1] A real matrix $\mathbf{A}$ is *normal* if $\mathbf{A}^T\mathbf{A} = \mathbf{A}\mathbf{A}^T$.

Numerical stability

It would be nice if we could get exact solutions to numerical problems.

But, the reality is that because the problems we study are continuous whereas computer arithmetic is discrete, this is not generally possible.

Stability tells us what is possible (or what we can expect) when solving a continuous problem with discrete arithmetic.

In other words, it tells us what it means to get the "right answer" even if this is not the exact answer.

Numerical stability

Recall that an abstract way to think of solving a problem is like evaluating a function

$$y = f(x),$$

where x represents the input to the problem (the data), f represents the "problem" itself, and y represents its solution.

An *algorithm* can be viewed as a different function $\tilde{f}$ that usually takes the same data (actually the rounded data) and maps it to a different solution $\tilde{f}(x)$.

For example, the computer code used to implement the algorithm is viewed as the $\tilde{f}$.

So, even if two different implementations are meant to produce the same result, these are generally two different functions $\tilde{f}_1$ and $\tilde{f}_2$.

Numerical stability

A good algorithm should have an $\tilde{f}$ that closely approximates the underlying problem f.

If nothing else, $\tilde{f}$ will be affected by rounding error during its execution.

If $\tilde{f}$ is a good algorithm, we might expect the relative error to be small, e.g., some small multiple of unit round-off.

We say that $\tilde{f}$ is an accurate algorithm for f if for all (relevant) input data x

$$\frac{\|\tilde{f}(x) - f(x)\|}{\|f(x)\|} = \mathcal{O}(u_e), \tag{1}$$

where u_e is unit round-off.

We will define the meaning of $\mathcal{O}(u_e)$ shortly.

Numerical stability

If $f(x)$ is ill-conditioned, the goal of achieving (1) is in fact unreasonable.

Rounding of input is inevitable, so even if the algorithm could somehow do everything exactly on the (rounded) input, $\|\tilde{f}(x) - f(x)\|$ may still be large!

So instead of always aiming for accuracy, the most we can (always) reasonably aim for is *stability*:

We say that an algorithm $\tilde{f}$ for a problem f is *stable* if for all (relevant) input data x

$$\frac{\|\tilde{f}(\tilde{x}) - f(x)\|}{\|f(x)\|} = \mathcal{O}(u_e),$$

for some $\tilde{x}$ satisfying

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(u_e).$$

Numerical stability

To express this in words:

The best we can hope for in practice is a stable algorithm, i.e., one that gives nearly the right answer to nearly the right question.

Note 4. *$\mathcal{O}(u_e)$ is too strict for problems such as solution of differential equations, where there are many "layers" of approximations made by the algorithms.*

The situation is better in numerical linear algebra, where a concept known as backward stability holds for many of the fundamental algorithms.

Backward stability

A backward stable algorithm satisfies the condition

$$\tilde{f}(x) = f(\tilde{x})$$

for some $\tilde{x}$ satisfying

$$\frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(u_e);$$

i.e., the algorithm gives exactly the right answer to nearly the right problem.

Of course, this is stronger than (just) stability.

$$\mathcal{O}(u_e)$$

We use the concept of

$$\| \text{ computed quantity } \| = \mathcal{O}(u_e)$$

in a sense that has a few assumptions built into it:

- $\| \text{ computed quantity } \|$ means the norm of some number(s) computed by some algorithm $\tilde{f}$ for a problem f, depending on both in the input data x for f and on u_e, e.g., the relative error.

Note 5. *Provided the input and output are finite-dimensional (which is always the case for this course), the norm used is not relevant.*

Theorem 1. *For finite-dimensional inputs and outputs, the properties of accuracy, stability, and backward stability all hold or fail to hold independent of the choice of norm.*

Note 6. *The only effect from one choice of norm to another is the constant buried in the $\mathcal{O}(u_e)$ notation.*

- There is an implicit process of $u_e \to 0$.

Of course, this is nonsense within a given floating-point number system.

One should imagine instead a series of computations done in higher and higher precision (perhaps on different computers), e.g., single precision, double precision, quadruple precision, etc.

Then $\| \text{ computed quantity } \| \to 0$ as the precision is increased.

- $\mathcal{O}(\cdot)$ applies *uniformly* to all data x.

i.e., the constant buried in the $\mathcal{O}(u_e)$ notation can be specified *independently of* x; i.e., it does not depend on the input x.

Stability of Floating-Point Arithmetic

The four simplest computational problems (functions) are $+, -, \times, /$.

We do not go into algorithmic details!

We now analyze the stability of their floating-point analogues: $\oplus, \ominus, \otimes, \oslash$.

It turns out that the axioms

$$\begin{aligned} \mathrm{fl}(x) &= x(1+\epsilon), & |\epsilon| \leq \epsilon_{\mathrm{machine}}, \\ x \circledast y &= (x * y)(1+\epsilon), \end{aligned}$$

imply that these most basic arithmetic operations are in fact backward stable.

Stability of Floating-Point Arithmetic

Let us show this for $\ominus$ since one might suspect this has the greatest risk of instability.

input: $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^2$

output: $x_1 - x_2 \in \mathbb{R}$

In functional form,

$$f(x_1, x_2) = x_1 - x_2.$$

So, our algorithm is

$$\tilde{f}(x_1, x_2) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2);$$

i.e., first round x_1, x_2 to their nearest floating-point numbers, then apply floating-point subtraction.

Now,

$$\begin{aligned} \mathrm{fl}(x_1) &= x_1(1+\epsilon_1), \\ \mathrm{fl}(x_2) &= x_2(1+\epsilon_2), \end{aligned}$$

for some ϵ_1, ϵ_2 satisfying

$$|\epsilon_1|, |\epsilon_2| \leq u_e.$$

Also,

$$\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = [\mathrm{fl}(x_1) - \mathrm{fl}(x_2)]\,(1+\epsilon_3),$$

for some ϵ_3 satisfying

$$|\epsilon_3| \leq u_e.$$

Thus,

$$\begin{aligned}
\mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) &= [x_1(1+\epsilon_1) - x_2(1+\epsilon_2)]\,(1+\epsilon_3) \\
&= x_1(1+\epsilon_1)(1+\epsilon_3) - \\
&\qquad\qquad x_2(1+\epsilon_2)(1+\epsilon_3) \\
&= x_1(1+\epsilon_4) - x_2(1+\epsilon_5),
\end{aligned}$$

for some ϵ_4, ϵ_5 satisfying

$$|\epsilon_4|, |\epsilon_5| \leq 2u_e + \mathcal{O}(u_e^2) \qquad \text{(verify!)}$$

i.e.,

$$f(x) = \mathrm{fl}(x_1) \ominus \mathrm{fl}(x_2) = \tilde{x}_1 - \tilde{x}_2,$$

where

$$\frac{|\tilde{x}_1 - x_1|}{|x_1|}, \ \frac{|\tilde{x}_2 - x_2|}{|x_2|} = \mathcal{O}(u_e).$$

In this case, any $C > 2$ will suffice for the constant implicit in $\mathcal{O}(\cdot)$.

Any norm on $\mathbb{R}^2$ now implies $\tilde{f}(x) = f(\tilde{x})$; i.e., floating-point subtraction is backward stable.

Stability of Floating-Point Arithmetic

Example 1: INNER PRODUCT
It can be shown that the inner product of two vectors is backward stable (exercise).

Example 2: OUTER PRODUCT
Given vectors $\mathbf{x} \in \mathbb{R}^m, \mathbf{y} \in \mathbb{R}^n$, compute the rank-one outer product $\mathbf{A} = \mathbf{x}\mathbf{y}^T$.

Obvious algorithm:
Set

$$\tilde{a}_{ij} = \mathrm{fl}(x_i) \otimes \mathrm{fl}(y_j),\ i = 1, 2, \ldots, m,\ j = 1, 2, \ldots, n.$$

This algorithm is stable, but not backwards stable (this is okay, by the way).

The reason it is not backward stable is because $\tilde{\mathbf{A}}$ will likely <u>not</u> have rank 1 ($\mathrm{rank}(\tilde{\mathbf{A}}) > 1$).

So, it cannot be written as

$$\tilde{\mathbf{x}}\tilde{\mathbf{y}}^T = (\mathbf{x} + \boldsymbol{\delta}\mathbf{x})(\mathbf{y} + \boldsymbol{\delta}\mathbf{y})^T.$$

In general when the dimension of the output exceeds that of the input (in this case, when $mn > m + n$), algorithms are rarely backward stable.

Example 3: ADDING 1
Let $x \in \mathbb{R}$, and suppose $f(x) = x + 1$.

Then, $\tilde{f}(x) = \text{fl}(x) \oplus 1$.

Again, this algorithm is stable, but not backward stable. It fails backward stability for $x \approx 0$.

However, the problem to compute $x + y$ for data x, y *is* backward stable.

In general, backward stability is a very special property; it is a reasonable goal for some problems but not others.

A lack of backward stability is not a deal-breaker, but a lack of (general) stability is.

Summary

- Conditioning pertains to the sensitivity of a *mathematical problem*.

- Stability pertains to the sensitivity of an *algorithm* used to solve a mathematical problem.

- "Nearly the right answer to nearly the right problem."