ERROR ANALYSIS FOR NUMERICAL COMPUTER METHODS

David F. Gleich

January 28, 2021

These notes are based on Chapter 1 in Gautschi's Numerical Analysis textbook.

THE SETUP

You have a problem P that you wish to solve

$$\underbrace{\begin{array}{ccccc} \mathbf{x} & & & & \mathbf{y} \\ \text{Input } \textit{or} & \rightarrow & \boxed{\mathbf{f}} & \rightarrow & \text{Output } \textit{or} \\ \text{Data} & & & & \text{Result} \end{array}}_{\text{the problem } P}$$

Examples of P include:

$$f(x,y) = x - y \qquad f(x) = \log(x) \qquad \mathbf{f}(\mathbf{x}) = \boldsymbol{A}^{-1}\mathbf{x}$$

$$f(\mathbf{x}) = \text{ roots of polynomials with coefficients } \mathbf{x}.$$

But the question of numerical computer *error* regards what happens because Your computer both

cannot represent exactly your input

and

cannot solve exactly your problem.

The goal in error analysis to is understand how well what you do on the computer approximates what you set out to do.

A VISUAL SETUP

Here is a diagram to help understand what is going on.

$$\begin{array}{rccc} & \mathbf{x} \in \mathbb{R}^m & \mathbf{f} & \mathbf{y} \in \mathbb{R}^n \\ & \bullet & \longrightarrow & \bullet \\ \text{Put } \mathbf{x} \text{ on machine} & \downarrow & & \\ & \bullet & \longrightarrow & \bullet \\ & \mathbf{x}^* \in \mathbb{M}^m & \mathbf{f} & \mathbf{y}^* \in \mathbb{R}^n \end{array}$$

This models what happens in terms of how the computer cannot represent our input. But the problem is even worse, because we don't actually compute $\mathbf{f}$ exactly.

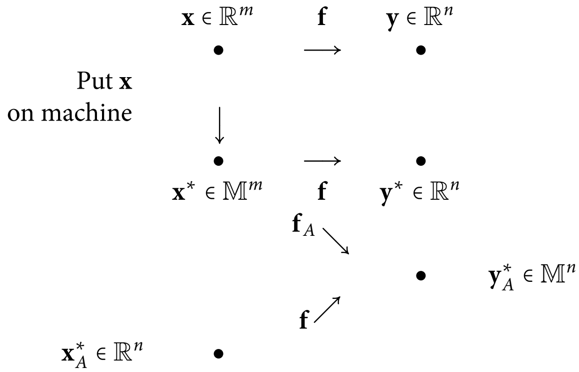
$$\begin{array}{rcccc} & \mathbf{x} \in \mathbb{R}^m & \mathbf{f} & \mathbf{y} \in \mathbb{R}^n & \\ & \bullet & \longrightarrow & \bullet & \\ \text{Put } \mathbf{x} \text{ on machine} & \downarrow & & & \\ & \bullet & \longrightarrow & \bullet & \\ & \mathbf{x}^* \in \mathbb{M}^m & \mathbf{f} & \mathbf{y}^* \in \mathbb{R}^n & \\ & & \searrow & & \\ & & \mathbf{f}_A & \bullet & \mathbf{y}_A^* \in \mathbb{M}^n \end{array}$$

This is problematic as $\mathbf{f}_A$ encodes the *algorithm* on the computer, in light of all numerical truncation, floating point approximation, etc. In order to make the analysis mathematical, we insist that our algorithms are *backwards stable*.

BACKWARDS STABILITY

If an algorithm is backwards stable, then we know that we solve the right problem for a perturbed input.

In terms of the diagram, this means that for a backwards stable algorithm, there exists $\mathbf{x}_A^*$ such that $\mathbf{y}_A^* = f(\mathbf{x}_A^*)$. On in the diagram



ERROR ANALYSIS IS CONNECTING THE DOTS.

We need to understand how $\mathbf{y}_A^*$ relates to $\mathbf{y}^*$ and how $\mathbf{y}^*$ and $\mathbf{y}$.

1. $\mathbf{y}_A^*$ vs $\mathbf{y}^*$ – this is an algorithm question that turns into a math question (hard!, next class)

2. $\mathbf{y}^*$ vs $\mathbf{y}$ – this is a math question – just calculus! (this class!)

To quantify these, we need norms.

NORMS

Norms are multivariate magnitudes or generalizations of the absolute value.[1] Let $\mathbf{x} \in \mathbb{R}^m$.[2] Then the p-norms are the functions:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{m} |x_i|^p \right)^{1/p}$$

[1] You can actually show that any norm applied to a scalar *will* be a scaled absolute value!

[2] In class, I wrote these with underlines $\underline{\mathbf{x}}$ because I can't write bold!