To err is human.
– Alexander Pope

Oh yeah, and what about computers?
– Toews

1 Errors

1.1 How errors arise

In order to begin to talk about errors, we need to agree on what it is that scientific computing does. Although
the question admits distinct viewpoints, few would argue that the thrust and goal of the discipline is to
calculate things. What sort of things? Well, many sorts of things: solutions to sets of algebraic equations,
inverses of matrices, the value of a certain function evaluated at a certain point, the fractal dimension of
an image—the list could go on and on. In every case, the answer is produced via computation, i.e. through
the manipulation of numbers. Problems for which the “answer” is not achieved through the manipulation
of numbers (a theory of the Good, for example) are not problems that fall within the purview of scientific
computing.

Computers only do what they are told to do. In order to compute an answer, the computer needs
instructions on what to do. These instructions are generally in the form of a “program”, i.e. a set of
commands written in some language the computer can understand. Most computers are polyglots, which is
to say that you can request the same thing in any of a bunch of different languages (Matlab, C, Java, etc.)
and the computer will understand what you mean. But the choice of language and the choice of machine may
influence the answer in subtle ways. People who do scientific computing well understand these subtleties.

Our model of scientific computing thus reduces to one of a function for which there are inputs and
outputs. The output is the answer: it is the solution to the algebraic system, the matrix inverse, the
function value—whatever was sought. The input is whatever information is needed to define the problem:
the coefficients of the algebraic equations, a matrix, a function definition. The process of converting the
inputs to the outputs is what concerns scientific computing.

Example 1.1. Consider the task of asking a computer to store the number $x = 0.1$ and then display it (as
it is stored.) The input in this case is the exact base 10 representation of a number. The output need not be
in base 10, but in the spirit of comparing apples to apples and oranges to oranges, it is reasonable to request
that the output also be in base 10. The computer of course will store the number in base 2, first allocating
some bits for storage, then assigning 1’s and 0’s as best it can to represent the “closest possible number.”
Lastly, it will need to convert its base 2 representation into a base 10 representation.

As an exercise, you might show that 0.1 cannot be represented exactly in a 64-bit binary computer. As
such, the output of this simplest possible problem in scientific computing is not exactly equal to the input.
Note that there are no errors in the inputs, since the number 0.1 doesn’t “represent” anything in the outside
world. The errors in this case are generated by the computer in endeavoring to produce the output.

Example 1.2. Suppose you are interested in calculating precisely how long it will take for a ball to hit the
ground if you drop it from a position $x_0 = 0.1$ km high. Using 9.8 as an approximation for the gravitational
constant, the position of the ball at time $t$ is given by

$$x(t) = x_0 - 4.9t^2.$$  

To solve this problem mathematically, you need to set $x(t) = 0$ and solve for $t$, yielding

$$t = \sqrt[2]{\frac{x_0}{4.9}}.$$  

To solve this problem on a computer, you could simply ask the machine to evaluate the right hand side of
the above expression.
The output to this problem is a value for $t$. Note that $t$ represents something “real”: I’m going to drop a ball, and there is an actual amount of time it will take to hit the ground. The inputs to the problem have been chosen in such a way that the output should represent this amount of time fairly accurately. The inputs are of several sorts, however. The obvious one is the initial position $x_0$. Another is the gravitational model, which is based on Newton’s Law $F = ma$. Note that this model doesn’t make an explicit appearance in what I ask the machine to do, but clearly drives my choice of task for the machine.

The errors in this problem come from several sources. Unlike the last problem, the input 0.1 now “represents” something, and as such, it won’t be know with infinite precision. As already noted, this number can’t be stored perfectly on the machine in any event, so this input contains some errors. The gravitational model is also not perfect, so the formula for $t$ would yield the incorrect answer even if $x_0$ were known and stored perfectly. Lastly, the calculations for finding $t$ (i.e. taking a square root and performing multiplication and division) involve approximations. The error in the output will accrue as a product of all these things.

1.2 Vector Spaces

The above discussion illustrates how errors creep into the calculations we perform on our machines, but it doesn’t begin to answer the more critical question of how to assess the seriousness of these errors. After all, we can plug $\sqrt{2}$ in our calculator and feel fairly confident about the answer, even if we know that it necessarily contains some error and we don’t know exactly how big that error might be. When should we start to worry?

In order to talk sensibly about error, we need to have a notion of size, i.e. to be able to say which quantities are big and which ones are small. This task is not as simple as it might appear to be, however, for the simple reason that the set of “things” that might be computed is vast and heterogeneous: it includes numbers, vectors, matrices, and functions, among many other things. While absolute error and relative error might be natural concepts for numbers, how do they extend, e.g., to functions?

As luck would have it, most of the objects we’ll be concerned about in this course can be viewed as living in vector spaces, a setting in which the concept of a norm provides a clean way to think about size. Recall that a vector space is a set whose elements can be added together and multiplied by scalars (to yield new elements that are also in the set.) More formally, a vector space over the real numbers is a set $X$ together with two mappings, one called addition (mapping $X \times X \rightarrow X$) and the other called scalar multiplication (mapping $\mathbb{R} \times X \rightarrow X$.) These mappings must satisfy the following familiar properties for all $x, y, z$ in $X$ and $a$ and $b$ in $\mathbb{R}$:

1. $x + (y + z) = (x + y) + z$
2. $x + y = y + x$
3. \exists an element 0 such that $0 + x = x$
4. \exists an element $-x$ such that $-x + x = 0$
5. $a(x + y) = ax + ay$.
6. $(a + b)x = ax + bx$
7. $a(bx) = (ab)x$
8. $1x = x$

Below are a few important examples of vector spaces.

**Example 1.3. (Vectors)** The set $\mathbb{R}^n$, i.e. the set of objects of the form $(x_1, \cdots, x_n)$, where addition is performed element-wise, i.e.

$$(x_1, \cdots, x_n) + (y_1, \cdots, y_n) = (x_1 + y_1, \cdots, x_n + y_n)$$

and scalar multiplication satisfies

$$c(x_1, \cdots, x_n) = (cx_1, \cdots, cx_n).$$
Example 1.4. (Matrices) The set $\mathbb{R}^{n \times m}$, i.e. the set of $n \times m$ matrices with real elements. As with the previous example, addition is performed element-wise and scalar multiplication distributes the scalar to every element.

Example 1.5. (Functions) The set of real-valued continuous functions on the interval $[0, 1]$, where addition satisfies

$$(f + g)(x) = f(x) + g(x)$$

and multiplication satisfies

$$(cf)(x) = c(f(x)).$$

1.3 Norms

Given a vector space $V$, a norm on $V$ is a mapping $\rho : V \to [0, \infty)$ satisfying the following properties:

1. $\rho(ax) = |a| \rho(x)$
2. $\rho(x + y) \leq \rho(x) + \rho(y)$
3. If $\rho(x) = 0$, then $x = 0$.

Norms provide a notion of size: one we find a norm $\rho$, we can say that element $x$ is bigger than element $y$ if $\rho(x) > \rho(y)$. Similarly, we can talk about how far apart two elements are by considering the norm of their difference: $x$ and $y$ are close if $\rho(x - y)$ is small, far if it is large.

Here are some examples of norms:

Example 1.6. (Scalars:) For $x \in \mathbb{R}$, the most familiar norm is that of the absolute value, defined formally as

$$|x| \equiv \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases}.$$ 

Example 1.7. (Vectors:) Let $x$ be an $n$-dimensional vector, i.e. $x = (x_1, x_2, \cdots, x_n) \in \mathbb{R}^n$. The $p$-norm of $x$ is defined as

$$\|x\|_p \equiv \left(\sum_{i=1}^{n} x_i^p\right)^{1/p}, \quad 1 \leq p < \infty.$$ 

There is a variant for $p = \infty$ as well, called the sup-norm:

$$\|x\|_\infty = \max_{i=1}^{n} |x_i|.$$ 

Note that in the special case that $n = 1$, i.e. in the case where $x$ is a scalar, both the $p$-norms and sup-norm reduce to just the absolute value of $x$.

Example 1.8. (Matrices:) Let $A$ be a matrix with $n$ rows and $m$ columns, i.e.

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{pmatrix} \in \mathbb{R}^{n \times m}.$$ 

Recall from linear algebra that the matrix-vector product $y = Ax$ is defined for any $x \in \mathbb{R}^m$, and that in this case $y \in \mathbb{R}^n$. In other words, $A$ is a mapping which takes elements $x \in \mathbb{R}^m$ to elements $y \in \mathbb{R}^n$. The size of $A$ is generally defined as the maximum of the ratios of the size of $y$ to the size $x$, i.e.

$$\|A\| \equiv \sup_{x \in \mathbb{R}^m} \frac{\|Ax\|}{\|x\|}.$$
Note that the numerator of this expression requires a notion of size in the vector space $\mathbb{R}^n$, while the denominator uses size in $\mathbb{R}^m$. As noted above, there are multiple options for how to define size in these vector spaces. If the sizes of $x$ and $y$ are measured in the same $p$–norm, then the induced matrix norm is generally notated with a subscript $p$: $$\|A\|_p = \sup_{x \in \mathbb{R}^m} \frac{\|Ax\|_p}{\|x\|_p}.$$ You can conceive of mixed matrix norms, too, where the size of the input is measured in a $p$−norm and the size of the output is measured in some other norm, e.g. a $q$−norm, $q \neq p$. In this case, we have $$\|A\|_{p,q} = \sup_{x \in \mathbb{R}^m} \frac{\|Ax\|_q}{\|x\|_p}.$$ These matrix norms are rather exotic and not commonly used in applications.

**Example 1.9. (Functions:)** Let $f(x)$ be a function with domain $\mathbb{R}$. Just as with vectors, there is a family of $p$−norms that can be used to measure the size of $f$: $$\|f\|_p = \left( \int_{-\infty}^{\infty} |f(x)|^p \, dx \right)^{1/p}, \quad 1 \leq p < \infty.$$ Note that not every function $f(x)$ defined on $\mathbb{R}$ will have a finite $p$ norm, and that there are functions that have a finite $p$–norm for certain values of $p$ and not for others. Also note that if both $f$ and $g$ have finite $p$ norm for some particular choice of $p$, then so does the function $h(x) = \alpha f(x) + \beta g(x)$ (exercise for the reader.)

Just as with vectors, the sup-norm for functions is defined slightly differently: $$\|f\|_{\infty} = \sup_{x \in \mathbb{R}} |f(x)|,$$ where sup represents the supremum.

### 1.4 Measuring errors via norms

Now that we have introduced vector spaces and norms, we can proceed to give a formal definition of error, one which extends the familiar concepts of absolute and relative error from numbers to many of the other objects we’ll be computing.

Let $x$ represent the true value of the thing we would like to compute, let $\hat{x}$ represent an approximation of $x$. If $x$ and $\hat{x}$ are not identical, we say that there is an error associated with the approximation $\hat{x}$. Suppose $x$ and $\hat{x}$ lie in a vector space $V$ with norm $\rho$. The **absolute error** of the approximation is given by $$\epsilon_a \equiv \rho(\hat{x} - x),$$ and the **relative error** is $$\epsilon_r \equiv \frac{\rho(\hat{x} - x)}{\rho(x)}.$$

Here are some useful examples of these concepts:

**Example 1.10. (Vectors:)** Suppose the true value of our vector is $x = (1, 1)$ and our approximation is $\hat{x} = (1.1, 0.9)$. The absolute and relative errors are given in the 2-norm as $$\epsilon_a = \sqrt{0.1^2 + 0.1^2} = 0.14, \quad \epsilon_r = \frac{0.14}{\sqrt{1^2 + 1^2}} = 0.1.$$ In the 3-norm, these errors are given by $$\epsilon_a = (0.1^3 + 0.1^3)^{1/3} = 0.13, \quad \epsilon_r = \frac{0.14}{(1^3 + 1^3)^{1/3}} = 0.1.$$
Note that changing norms reduced the absolute error but not the relative error. This will not always be the case.

Example 1.11. (Functions:) Suppose the true population size of the African leaping turtle is \( f(t) = 3e^{-t} \), but that your approximation (based on very limited data) is \( \tilde{f}(x) = 2e^{-x} \). Assuming the domain of this function is the time interval \([0, \infty)\), the absolute error of your approximation in the 2-norm is

\[
\epsilon_a = \rho(f - \tilde{f}) = \left( \int_0^\infty (3e^{-t} - 2e^{-t})^2 \, dt \right)^{1/2} = 1.
\]

A similar calculation shows that the relative error is 1/3.
1.5 Sensitivity Analysis

In the functional model of scientific computing, inputs are turned into outputs via some computational process. If the inputs have errors, the outputs will have errors. **Sensitivity analysis** is the attempt to understand how errors in the inputs propagate into errors in the outputs.

Here are some examples of sensitivity analysis in action:

**Example 1.12.** *(Linear functions: )* Suppose \( y = ax + b \) gives the predicted number of cardiac arrests per day, \( y \), as a function of the ambient temperature in degrees, \( x \). If we predict that the temperature will be \( x_0 \) on a given day, the predicted number of cardiac arrests will be \( y_0 = ax_0 + b \). Temperature predictions are notoriously tricky, however: it is natural to ask how the number of cardiac arrests will change if the actual temperature deviates from \( x_0 \).

Writing the actual temperature as \( x_0 + h \), we can calculate the difference between the actual number of cardiac arrests and the predicted number as

\[
y(x_0 + h) - y(x_0) = (ax_0 + h) - (ax_0 + b) = ah,
\]

which shows that

\[
\text{(error in predicting cardiac arrests)} = a \times \text{(error in predicting temperature)}.
\]

In other words, if our temperature predictions are off by two degrees, our cardiac arrest predictions will be off by \( 2a \) cases of cardiac arrest. A more common way of writing this fact is to express the ratio of the output to input errors as

\[
\frac{(error \ in \ predicting \ cardiac \ arrests)}{(error \ in \ predicting \ temperature)} = \frac{y(x_0 + h) - y(x_0)}{h} = a.
\]

**Example 1.13.** Consider the same example, this time with \( y = x^2 \). Again, if we predict a temperature of \( x_0 \), we anticipate \( y_0 = x_0^2 \) cases of cardiac arrest. If our temperature error is \( h \), then we anticipate a cardiac arrest prediction error of

\[
y(x_0 + h) - y(x_0) = (x_0 + h)^2 - x_0^2 = 2x_0h + h^2.
\]

This answer is not as clean as the last one. Do note, however, that if we look at the ratio of the output to the input error, we get

\[
\frac{y(x_0 + h) - y(x_0)}{h} = 2x_0 + h,
\]

and that for small values of \( h \), both sides of the equation approach the derivative of \( y \) with respect to \( x \), evaluated at \( x_0 \).

Motivated by the last two examples, we can define the **sensitivity** of an output to an input as the derivative of the function that takes the one to the other. Note that the derivative should be interpreted as the ratio of the change in the output to the change in the input.

It is easy to extend this definition to functions that take vector inputs and yield vector outputs. Concretely, let \( f : \mathbb{R}^m \to \mathbb{R}^n \) be a vector valued function, i.e.

\[
f(x_1, \cdots, x_m) = (y_1, \cdots, y_n).
\]

Then we define the **sensitivity** of the \( i \)th output variable with respect to the \( j \)th input variable as

\[
\epsilon_{ij} \equiv \frac{\partial y_i}{\partial x_j}.
\]

Note that the **Jacobian** of \( f \) is just the set of sensitivites arranged into a matrix:

\[
J = \begin{pmatrix}
\epsilon_{11} & \cdots & \epsilon_{1m} \\
\vdots & \ddots & \vdots \\
\epsilon_{n1} & \cdots & \epsilon_{nm}
\end{pmatrix}.
\]
In practice, sensitivity analysis often extends well beyond deterministic, vector valued functions. For example, if noisy data is used to estimate some parameters, one might reasonably inquire how the variance of the noise influences the variance of the solution. Since this is not a deterministic problem (i.e. there is no way to write down a concrete $f$ as above), sensitivity analysis might very well take the form of something like a Monte Carlo simulation. We will encounter this form of sensitivity analysis a great deal in what follows.

### 1.6 Condition

Consider a problem in scientific computing in which inputs $x$ lead to outputs $y$ via some computational process. We say that the problem is **well conditioned** if whenever the distance between two inputs is small, so is the distance between the corresponding outputs. Formally, we can express this idea in the familiar $\epsilon - \delta$ notation by saying that for any $\epsilon > 0$, there exists a $\delta > 0$ such that if $\rho_x(x - \tilde{x}) < \delta$ then $\rho_y(y - \tilde{y}) < \epsilon$, where $\rho_x$ and $\rho_y$ are norms on the input and output spaces, respectively. The larger the ratio $\epsilon/\delta$ must be, the worse the condition of the problem.

**Example 1.14.** Consider the problem of inverting a matrix of the form

$$A = \begin{pmatrix} 1 & 0 \\ 0 & \epsilon \end{pmatrix}$$

Clearly we have

$$A^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1/\epsilon \end{pmatrix}.$$

Note that if $\tilde{A}$ is the same as $A$, with $\epsilon$ replaced by $\tilde{\epsilon}$, then

$$\|A - \tilde{A}\|_p = \begin{pmatrix} 0 & 0 \\ 0 & (\epsilon - \tilde{\epsilon}) \end{pmatrix}^p = |\epsilon - \tilde{\epsilon}|$$

while

$$\|A^{-1} - \tilde{A}^{-1}\| = \begin{pmatrix} 0 & 0 \\ 0 & (1/\epsilon - 1/\tilde{\epsilon}) \end{pmatrix}^p = |1/\epsilon - 1/\tilde{\epsilon}|$$

If $\epsilon$ is very small and $\tilde{\epsilon} = \epsilon/2$, then the input distance $\|A - \tilde{A}\|_p = \epsilon/2$ would be small while the output distance $1/\epsilon$ would be very large. This problem becomes progressively more ill-conditioned as $\epsilon \to 0$.

Note the difference between sensitivity analysis and condition analysis as we have defined them here: the former endeavors to understand how changes to specific input parameters change specific output parameters, while the latter looks at how norm bounds on input errors relate to norm bounds of output errors. That said, the distinction is not totally straightforward, and the two terms are often used interchangeably. For example, it is possible to characterize problems with huge sensitivity as ill-conditioned, and vice-versa.