LECTURE NOTES

ECO 613/614

FALL 2007

KAREN A. KOPECKY

Computing Basics

1 Sources of Error

Numerical solutions to problems differ from their analytical counterparts. Why? The reason for the difference is that most numerical computations involve error. There are two main types of error that can arise. The first, called *roundoff error*, arises from the fact that computers do not store real numbers instead they store a subset of real numbers: floating-point numbers. Thus the solution of computer arithmetic is approximated with the nearest machine-representable number. The second, called *truncation error*, arises from truncating infinite series, substituting for continuous functions with discrete approximations, and approximating limits of infinite processes.

There are two statistics used to measure error: *absolute error* and *relative error*. Suppose a real number x is represented in the computer by the floating-point number $\tilde{x}$, then the absolute roundoff error is $||x-\tilde{x}||$ and the relative roundoff error is $\frac{||x-\tilde{x}||}{||x||}$. Similarly, suppose we approximate a function f by $\tilde{f}$, then the absolute truncation error from evaluating f at some point x is $||f(x)-\tilde{f}(x)||$ and the relative truncation error is $\frac{||f(x)-\tilde{f}(x)||}{||f(x)||}$.

Most computations will involve a combination of truncation and roundoff error. For example suppose we want to evaluate some function, $f(\cdot)$ at some real number x numerically. In order to do this we approximate $f(\cdot)$ with $\tilde{f}(\cdot)$ and the computer stores our real number x as the floating-point number $\tilde{x}$. We can bound the total error, either absolute or relative, by the sum of the roundoff error and the truncation error:

$$\text{total error} = ||\tilde{f}(\tilde{x}) - f(x)|| \leq \underbrace{||\tilde{f}(\tilde{x}) - f(\tilde{x})||}_{\text{truncation error}} + \underbrace{||f(\tilde{x}) - f(x)||}_{\text{roundoff error}}. \tag{1}$$

Good computer programs keep total error, or the upper bound on total error that is determined by the sum of roundoff error and truncation error, as small as possible given constraints on computation time and memory space. Roundoff error is a characteristic of computer hardware. For this type of error the most important thing is to keep it from propagating. Truncation error, on the other hand, is a characteristic of the algorithm or numerical procedure being employed. Often times truncation error can be reduced by increasing the accuracy of approximations. But this improvement comes at a cost of either memory space or time. This is the tradeoff between *accuracy* and *efficiency*.

To see how numerical error can grow we must first understand how the computer stores its approximations of real numbers. Floating-point numbers are stored on a computer in binary (base 2) form. Basically, a floating point number is a collection of *bits* (0's and 1's). Most computation today is done in double precision which means that each floating point number consists of 2 *words*. A word is a collection of bits. On standard computers today 1 word = 4 bytes or 32 bits (1 byte = 8 bits). This means that floating point numbers are stored using 64 bits each, set to either 0 or 1. These bits are split up between what's called the number's *mantissa* and the number's *exponent*. In addition, a few bits are set aside to store the sign of the number and possibly other characteristic flags (such as whether it is zero or "undefined"). Thus the value of the floating point number can be expressed as

$$\underbrace{\sum_{i=0}^{N} d_i 2^{-i}}_{\text{mantissa}} \times 2^E, \tag{2}$$

where E is the exponent, $d_i \in \{0, 1\}$, and N is the maximum number of 0's and 1's in the number. Another way to write this is

$$[d_1 2^{-1} + d_2 2^{-2} + \cdots + d_N 2^{-N}] \times 2^E = [.d_1 d_2 \ldots d_N] \times 2^E$$

This may be easier to see if we consider the equivalent representation in base 10:

$$\underbrace{\sum_{i=0}^{N} s_i 10^{-i}}_{\text{mantissa}} \times 10^E, \tag{3}$$

where $s_i \in \{0, 1, \ldots, 9\}$. Or

$$[s_1 10^{-1} + s_2 10^{-2} + \cdots + s_N 10^{-N}] \times 10^E = [.s_1 s_2 \ldots s_N] \times 10^E$$

Notice that allocating more bits to storing E will increase the range of numbers the computer can handle, while increasing N will increase the degree of precision of calculations.

Notice that the smallest number such that when added to 1 the computer can tell is no longer 1 is 2^{-N}. This is also the smallest relative spacing between two numbers that the computer can distinguish. It is called *machine epsilon*. For double-precision N is usually 52 or 53 thus machine-epsilon is 2^{-52} or 2^{-53} which is approximately 10^{-16}. This value is also sometimes referred to as *unit roundoff* and here we will denote it by u. Thus, for double-precision, any real number can be approximated to a relative accuracy of 10^{-16}, or, in other words, the real number and its computer floating-point approximation will agree up to at least the fifteenth digit.

Every time an arithmetic computation is performed on a computer the result is approximated by a floating-point number. In some cases this can lead to large roundoff errors. One example is the subtraction of a number from another of similar value. The error is worse the larger are the two numbers. Suppose for instance that x and y are two large real numbers with very similar values. It must be that

$$|x - \tilde{x}| \leq u|x|$$

and

$$|y - \tilde{y}| \leq u|y|$$

where u is the unit roundoff. Also

$$|(x - y) - \widetilde{(x - y)}| \leq u|x - y|$$

We can bound the absolute error as follows

$$\begin{aligned}
|(x - y) - \widetilde{(\tilde{x} - \tilde{y})}| = & & (4)\\
|(x - y) - (\tilde{x} - \tilde{y}) + (\tilde{x} - \tilde{y}) - \widetilde{(\tilde{x} - \tilde{y})}| &\leq |(x - y) - (\tilde{x} - \tilde{y})| + |(\tilde{x} - \tilde{y}) - \widetilde{(\tilde{x} - \tilde{y})}| & (5)\\
&\leq |x - \tilde{x}| + |y - \tilde{y}| + |(\tilde{x} - \tilde{y}) - \widetilde{(\tilde{x} - \tilde{y})}| & (6)\\
&\leq u|x| + u|y| + u|\tilde{x} - \tilde{y}| & (7)\\
&\leq u|x| + u|y| + u|x - y| \approx 2u|x| + 2u|y| & (8)
\end{aligned}$$

So the relative error is bound by

$$\frac{2u|x| + 2u|y|}{|x - y|} \approx \frac{4|x|u}{|x - y|}$$

which could be quite large since $|x| >> |x-y|$. Another way to see this is to consider that, x and y, are accurate to n digits and have the first $\tilde{n}$ digits in common. Then their difference will have approximately $n - \tilde{n}$ significant digits. The number of significant digits in the difference is smaller the more digits that x and y have in common.

Another example of how arithmetic computations can result in big roundoff error is when very large numbers are multiplied by very small numbers, magnifying the approximation error of the small number. Two general rules of thumb are (1) avoid taken the difference of two very similar numbers when possible and (2) avoid multiplying very large numbers by very small numbers.

Roundoff error can also become significant when the results of computations are out of the range of floating-point numbers the computer can store. While these numbers vary somewhat across processors and programs, for double-precision computing, the smallest positive number that can be represented on the computer, termed *machine zero*, is approximately 10^{-308}, while the largest number, termed *machine infinity*, is approximately 10^{308}. This is because there are usually about 11 bits reserved for the exponents. Which means 2^{11} possible values for exponent or 2048 but half are for positive and half for negative so the largest number is approximately $2^{1024} \approx 10^{308}$. It is possible to get output of computer computations done on machine-representable numbers that exceed machine infinity in value. This event is called *overflow*. Similarly, when output of computer computations results in a positive, non-zero, but less than machine epsilon value, it is called *underflow*.

2 Conditioning and Stability

At various steps in a computational algorithm roundoff error and truncation error are introduced. The impact that this error has on the difference between the true solution to the initial problem and the solution resulting from the computation will depend on both the sensitivity of the problem and the sensitivity of the algorithm to the introduction of error.

Conditioning is a property of the problem. A problem in which a given relative change in the value of the inputs results in a similar relative change in the output value is said to be *insensitive* or *well-conditioned*. Problems that are well-conditioned do not cause errors to propagate since the relative error of the output will be on the same scale as the relative error of inputs. Problems in which small relative changes in inputs leads to large relative changes in the output are said to be *ill-conditioned* or *sensitive*. An example of an ill-conditioned problem is

$$\begin{bmatrix} 1.00001 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2.00001 \\ 2 \end{bmatrix}. \tag{9}$$

Its exact solution is $x = (1,1)'$ but if we change the first element on the right-hand-side from 2.00001 to 2 the solution changes drastically to $x = (0,2)'$.

One measure of ill conditioning in a linear equation $Ax = b$ is the "elasticity" of the solution vector x with respect to the data vector b:

$$\epsilon = \sup_{||\Delta b||>0} \frac{||\Delta x||/||x||}{||\Delta b||/||b||}.$$

The elasticity gives the maximum percentage change in the size of the solution vector x induced by a 1 percent change in the size of the data vector b. If the elasticity is large, then small errors in b can result in large errors in x.

The elasticity is expensive to compute and is instead estimated using the *condition number* of the matrix A. For invertible A this is $\kappa \equiv ||A|| \cdot ||A^{-1}||$. The condition number of A is the least upper bound of the elasticity. Rule of thumb is that for each power of 10 in the condition number, one significant digit is lost in the computed solution vector x.

Stability is a property of the algorithm. An algorithm is said to be stable if its outcome is relatively insensitive to roundoff error. Thus the accuracy of numerical procedures depends on both the conditioning of the problem and the stability of the algorithm. In other words, when numerical solutions to problems differ greatly from their analytical counterpart the reason could be that the problem is ill-conditioned or that the numerical procedure is unstable or both.

Side note: In practice, when numerical solutions to problems are not as expected the majority of the time the culprit is human error, i.e., there is either a mistake in the code or there is a mistake in the setup of the problem. Thus this is always the first possibility you should explore.

References

- **Judd, Kenneth L.** 1998. *Numerical Methods in Economics.* Cambridge, MA: MIT Press.
- **Nocedal, Jorge and Stephen J. wright.** 1999. *Numerical Optimization.* Springer-Verlag New York, Inc.