Stability of Algorithms

The problem is to evaluate $f(x)$. For now, let's assume that this is a scalar-valued function of a scalar argument x. We write a program and run it on a computer with IEEE arithmetic and it returns an answer denoted as $\tilde{f}(x)$. We know that in most cases we cannot expect to get the exact answer to our problem – if the answer is, say, π or even $\frac{1}{10}$, that number cannot be represented exactly on the machine. In what sense can we expect $\tilde{f}(x)$ to approximate $f(x)$?

Ideally, we might hope to get **almost the right answer to exactly the problem we are trying to solve.** Unfortunately, after a little thought, we realize that this may not be possible if the problem is *ill-conditioned*. In that case, replacing the input value x by round(x), that is, by the closest number to x that can be stored in the machine word, may change the answer drastically: $f(\text{round}(x))$ is not close to $f(x)$. Therefore, the best that we can hope for is probably

$$\frac{|\tilde{f}(x) - f(x)|}{|f(x)|} \leq C\kappa_f(x)\epsilon_{machine}, \tag{1}$$

where $\kappa_f(x)$ is the (relative) condition number of the problem of evaluating $f(x)$ and C is a moderate size "constant" that might depend on the number of operations performed in the algorithm, and hence on the size of the input if x is a vector or other assemblage of numbers, but is independent of the particular value(s) of x and independent of $\epsilon_{machine}$, assuming that $\epsilon_{machine}$ is below some reasonable threshold.

One way to show that (1) holds is to show that the algorithm is *backward stable*, meaning that it gives **exactly the right answer to almost the problem we are trying to solve**:

$$\tilde{f}(x) = f(\hat{x}), \quad \text{for some } \hat{x} \text{ with } \frac{|\hat{x} - x|}{|x|} \leq c\epsilon_{machine}, \tag{2}$$

where c is another moderate size constant independent of the particular value(s) of x. As an example, consider the problem of computing the inner product of two real vectors of length 2:

$$u^T v = u_1 v_1 + u_2 v_2.$$

If we do this using the most straightforward algorithm, we will actually compute

$$\text{round}(u_1) \otimes \text{round}(v_1) \oplus \text{round}(u_2) \otimes \text{round}(v_2) =$$

$$[u_1(1+\epsilon_1)v_1(1+\epsilon_2)(1+\epsilon_3) + u_2(1+\epsilon_4)v_2(1+\epsilon_5)(1+\epsilon_6)]\,(1+\epsilon_7),$$

where each $|\epsilon_j| \leq \epsilon_{machine}$. The computed result is the exact inner product of the vectors $\hat{u}$ and $\hat{v}$, where

$$\hat{u}_1 = u_1(1+\epsilon_1)(1+\epsilon_3), \quad \hat{v}_1 = v_1(1+\epsilon_2)(1+\epsilon_7),$$

$$\hat{u}_2 = u_2(1+\epsilon_4)(1+\epsilon_6), \quad \hat{v}_2 = v_2(1+\epsilon_5)(1+\epsilon_7).$$

[We could have divided the ϵ_j's between the entries of $\hat{u}$ and $\hat{v}$ differently, but this is one way to define $\hat{u}$ and $\hat{v}$ so that each entry satisfies $|\hat{u}_i - u_i|/|u_i| \leq c\epsilon_{machine}$, $|\hat{v}_i - v_i|/|v_i| \leq c\epsilon_{machine}$, $i = 1, 2$, for a moderate size constant c; e.g., $c = 3$.]

Sometimes inequality (1) holds even if the algorithm is not backward stable. Trying to show in a more direct way that (1) holds is sometimes called *forward error analysis*: We just directly compare the computed result and the exact answer. Consider the problem of computing the outer product of two real vectors of length 2:

$$uv^T = \begin{bmatrix} u_1v_1 & u_1v_2 \\ u_2v_1 & u_2v_2 \end{bmatrix}.$$

Again we do this using the most straightforward algorithm and we compute

$$\begin{bmatrix} \text{round}(u_1) \otimes \text{round}(v_1) & \text{round}(u_1) \otimes \text{round}(v_2) \\ \text{round}(u_2) \otimes \text{round}(v_1) & \text{round}(u_2) \otimes \text{round}(v_2) \end{bmatrix} =$$

$$\begin{bmatrix} u_1(1+\epsilon_1)v_1(1+\epsilon_3)(1+\epsilon_5) & u_1(1+\epsilon_1)v_2(1+\epsilon_4)(1+\epsilon_6) \\ u_2(1+\epsilon_2)v_1(1+\epsilon_3)(1+\epsilon_7) & u_2(1+\epsilon_2)v_2(1+\epsilon_4)(1+\epsilon_8) \end{bmatrix}.$$

Unless the rounding errors just happen to make this a rank one matrix, there is no way that this could be the *exact* outer product of some nearby vectors $\hat{u}$ and $\hat{v}$, because $\hat{u}\hat{v}^T$ is always a rank one matrix. Thus the algorithm is *not* backward stable. But it still satisfies the desired inequality (1) because each entry of the computed matrix is close to the corresponding entry of the exact matrix; the absolute value of the difference between the computed (i, j)-entry and the true value u_iv_j is less than $4\epsilon_{machine}$ times $|u_iv_j|$.

In the text, an even weaker form of stability is defined. An algorithm is said to be *stable* if it finds **almost the right answer to almost the problem we are trying to solve**:

$$\frac{|\tilde{f}(x) - f(\hat{x})|}{|f(\hat{x})|} \leq C\epsilon_{machine}, \quad \text{for some } \hat{x} \text{ with } \frac{|\hat{x} - x|}{|x|} \leq c\epsilon_{machine}.$$

This is a less useful definition, since it is usually difficult to verify.

An algorithm that does not satisfy any of the above definitions of stability, or, one that does not satisfy inequality (1) is said to be *unstable*. To show that an algorithm is unstable, one usually finds a well-conditioned problem for which (1) is not satisfied. An example in the text is computing the eigenvalues of a Hermitian matrix by forming the characteristic polynomial and then computing the roots of that polynomial. It can be shown that a small change in the matrix makes a small change in the eigenvalues, so the Hermitian eigenvalue problem is well-conditioned. However, if small errors are made in computing the coefficients of the characteristic polynomial, these can change the roots of that polynomial drastically. Thus, this algorithm should not be used for computing eigenvalues.

In summary, inequality (1) is usually what we want to hold for a good algorithm. If f is a vector or matrix-valued function of a vector or matrix of arguments x, we usually hope that (1) holds with absolute value signs replaced by some appropriate norm. Sometimes one asks for even greater accuracy in the computation of a vector or matrix-valued function; one asks that the error in *each entry* of the output be small compared to the size of that entry of the exact solution. For example, while many algorithms for computing the eigenvalues of a Hermitian matrix A produce approximate eigenvalues $\tilde{\lambda}_j$ that satisfy

$$\frac{|\tilde{\lambda}_j - \lambda_j|}{\|A\|} = O(\epsilon_{machine}),$$

there are some algorithms specially designed to satisfy the stricter requirement

$$\frac{|\tilde{\lambda}_j - \lambda_j|}{|\lambda_j|} = O(\epsilon_{machine}).$$

If the eigenvalues of A range over many orders of magnitude, say, the largest eigenvalue is 1 and the smallest is 10^{-32}, then this latter criterion ensures that even the smallest eigenvalues are computed accurately to many decimal

places. See, for example, chapter 5 of *Applied Numerical Linear Algebra* by James Demmel.