Week 2: Friday, Aug 31

Logistics

1. Monday, Sep 3 is a university holiday. Please don't come to class – I won't be there!

Dot products in floating point

The types of floating point error analysis done in numerical linear algebra typically follow a common strategy:

1. Write an expression in exact arithmetic. This expression is often in the form of a recurrence.
2. Write a corresponding expression in the $1 + \delta$ model.
3. Take the difference to get an expression for the error.
4. Find a simplified bound on the error expression.
5. If possible, express that simplified bound in terms of a perturbation to the input data.

Let us work through this strategy in detail in the case of a dot product of two real vectors, as computed by the following code:

```
s = 0;
for i = 1:n
  s = s + x(i)*y(i);
end
```

Let s_i denote the partial sum computed at step i in exact arithmetic, $\hat{s}_i = \mathrm{fl}(s_i)$ be the partial sum in floating point, and $e_i = \hat{s}_i - s_i$ be the difference. Using the $1 + \delta$ model of roundoff error, we have the following recurrences

$$\begin{aligned}
s_1 &= x_1 y_1; & s_{k+1} &= s_k + x_{k+1} y_{k+1} \\
\hat{s}_1 &= x_1 y_1 (1 + \gamma_1); & \hat{s}_{k+1} &= (\hat{s}_k + x_{k+1} y_{k+1} (1 + \gamma_{k+1}))(1 + \delta_k) \\
e_1 &= x_1 y_1 \gamma_1; & e_{k+1} &= s_k \delta_k + (e_k + x_{k+1} y_{k+1} \gamma_{k+1})(1 + \delta_k) \\
& & &= e_k + s_k \delta_k + x_{k+1} y_{k+1} \gamma_{k+1} + O(\epsilon_{\mathrm{mach}}^2)
\end{aligned}$$

Running the recurrence for the e_k forward, we have

$$e_n = \sum_{i=1}^{n} x_i y_i \left(\gamma_i + \sum_{j=i+1}^{n} \delta_j \right) + O(n\epsilon_{\mathrm{mach}}^2).$$

Each of the sums of δ_j is bounded by $n\epsilon_{\mathrm{mach}}$, so we can bound the error expression by

$$|e_n| \leq n\epsilon_{\mathrm{mach}} \sum_{i=1}^{n} |x_i||y_i| + O(n\epsilon_{\mathrm{mach}}^2).$$

As is typical in rounding error analysis, we will drop the terms involving $O(\epsilon_{\mathrm{mach}}^2)$ and write

$$|e_n| \lesssim n\epsilon_{\mathrm{mach}} |x|^T |y|$$

The absolute error bound $|e_n|$ is a bound on the difference between the true and computed output, sometimes called a *forward* error bound. However, there is also another way to interpret our analysis. Note that the computed dot product is

$$\mathrm{fl}(x^T y) = \sum_{i=1}^{n} x_i y_i (1 + \eta_i) + O(n\epsilon_{\mathrm{mach}}^2),$$

where $|\eta_i| \leq n\epsilon_{\mathrm{mach}}$. If we let $\tilde{x}_i = x_i(1 + \eta_i)$, we have

$$\mathrm{fl}(x^T y) = \tilde{x}^T y + O(n\epsilon_{\mathrm{mach}}^2),$$

That is, we can explain the error in the computed dot product as the same error we would have gotten had we perturbed the input just slightly. This perturbation in the input that corresponds to the result is known as a *backward* error.

We note that a similar analysis to the analysis for dot products gives

$$\mathrm{fl}(Ax) = \tilde{A}x,$$

where the elements of $\tilde{A}$ are $\tilde{a}_{ij} = a_{ij}(1 + \eta_{ij})$ with $|\eta_{ij}| \leq n\epsilon_{\mathrm{mach}}$. This is true even under blocked rearrangements of the algorithm (though this error bound does not necessarily hold for Strassen's algorithm).

Algorithms whose computed results in floating point correspond to a small relative backward error, such as the standard dot-product and matrix-vector multiplication algorithm, are said to be *backward stable*.

Forward and backward error analysis

The analysis of forward and backward error for dot products and matrix-vector multiplies is part of a bigger picture. Suppose that we have a space of problem data $\mathcal{P}$ and a space of possible solutions $\mathcal{S}$. We would *like* to evaluate some exact function

$$f : \mathcal{P} \to \mathcal{S},$$

but because of errors in intermediate steps of our calculation, we really evaluate

$$\hat{f} : \mathcal{P} \to \mathcal{S}.$$

The forward error way of thinking says that we are given some problem p and the error is the difference $f(p) - \hat{f}(p)$ between the true and the computed solution. The backward error way of thinking says that the $\hat{f}(p)$ we computed is the same as $f(\hat{p})$, an evaluation of the *true* function for some slightly erroneous data. The difference $\hat{p} - p$ is the backward error.

If we know something of the sensitivity of f (e.g., if we know a bound on $f'(p)$), then we can estimate the forward error from this backward error. As a concrete example, consider computing $y = Ax$, where $A \in \mathbb{R}^{n \times n}$ and $x \in \mathbb{R}^n$. From the analysis we just did, we know that

$$\operatorname{fl}(y) = (A + E)x, \quad |E| \leq n\epsilon_{\mathrm{mach}}|A|;$$

that is, the effects of rounding in matrix multiplication can be explained via a small normwise relative perturbation of $\|E\|/\|A\| < n\epsilon_{\mathrm{mach}}$. But we also know from the analysis in Monday's lecture that therefore

$$\frac{\|\operatorname{fl}(y) - y\|}{\|y\|} \leq \kappa(A) n \epsilon_{\mathrm{mach}}.$$

Thus, backward stability implies that the forward normwise relative error is essentially proportional to the condition number times ϵ_{mach} (times some modest constant depending on n).

Most of the standard algorithms in numerical linear algebra are backward stable. If we have backward stability (a property of the algorithm), we are generally freed from the obligation to work through tedious floating point error analysis, and can instead concentrate on understanding the condition number (a property of the problem).