Numerical Linear Algebra

We are going to consider several standard problems:

1) Solve $A\vec{x}=\vec{b}$, where A is $n\times n$, $\vec{x}$ is $n\times 1$ and $\vec{b}$ is $n\times 1$.

(for us, n ranges from 10s to millions)

2) The least-squares problem. ~~So~~ Given A, B

$$\min_{\vec{x}} \|\underset{m\times n}{A}\ \underset{n\times 1}{\vec{x}} - \underset{m\times 1}{\vec{b}}\|_2, \text{ where } \|\vec{y}\|_2=\sqrt{\textstyle\sum y_i^2}.$$

Generally, these problems are <u>over</u>determined ($m>n$) so that the minimum is not zero.

However, <u>under</u>determined ($m<n$) problems are also important (and hard, b/c they have ∞ly many solns).

3) The eigenvalue problem.

Given $\underset{n\times n}{A}$, find $\underset{n\times 1}{\vec{x}}$ and λ so $A\vec{x}=\lambda\vec{x}$.

We will try to find algorithms which are <u>fast</u> and <u>stable</u>, and which use anything we are given about the matrix A.

Idea 1. Matrix factorizations.

Given A, we can write A as the product of matrices XY with special structure.

Example Solve $A\vec{x}=\vec{b}$ if A is lower triangular

$$\begin{bmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ \vdots & & & \\ a_{n1} & \cdots & & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \\ \\ b_n \end{bmatrix}$$

This isn't hard to do:

$$x_1 = b_1 / a_{11}$$

$$x_2 = (b_2 - a_{21} x_1) / a_{22}$$

$$x_3 = (b_3 - a_{31} x_1 - a_{32} x_2) / a_{33}$$

$$\vdots$$

$$x_k = \left(b_k - \sum_{i=1}^{k-1} a_{ki} x_i\right) / a_{kk},$$

which is called <u>forward</u> <u>substitution</u>.

Exercise: Write down the analogous <u>back</u> <u>substitution</u> procedure if A is upper triangular.

The method you know (Gaussian elimination) for solving linear systems can be rewritten

Theorem (LU decomposition)
If the $n \times n$ matrix A is nonsingular, $\exists$ a permutation matrix P, a nonsingular lower triangular matrix L, and a nonsingular upper triangular matrix U so that $A = PLU$.

Here, a permutation matrix P is an orthogonal matrix given by permuting the rows of I. We can compute P^{-1} using the fact that $P^T = P^{-1}$.

We can then solve the system $A\vec{x} = \vec{b}$ by the following procedure.

Step 1. Write $A\vec{x}=\vec{b}$ as $PLU\vec{x}=\vec{b}$. Multiply by P^{-1} (permute entries of $\vec{b}$) to get

$$LU\vec{x} = P^{-1}b$$

Step 2. Solve for Ux by forward substitution. Call $U\vec{x}=\vec{c}$.

Step 3. Solve $U\vec{x}=\vec{c}$ by back substitution. Get $\vec{x}$.

Notice that if we have the decomposition $A=PLU$ then it is easy to solve $A\vec{x}=\vec{b}$ for various $\vec{b}$ (we don't have to recompute PLU to do so).

~~*~~ How accurate is this procedure? We now define two ways to think about error in algorithms.

Definition. If $f : \mathbb{R} \to \mathbb{R}$ is differentiable, we call $|f'(x)|$ the <u>absolute</u> <u>condition number</u> of f at x.

Observe that for small δx,

$$|f(x+\delta x) - f(x)| \approx |\delta x|\,|f'(x)|$$

so that a small input error δx can result in a large output error if $|f'(x)|$ is very large.

$|f'(x)|$ large $\iff$ f is ill-conditioned at x.

We usually use the <u>relative</u> condition number, which we can get by playing with the error bound to get <u>relative</u> error in input and output:

$$\frac{|f(x+\delta x) - f(x)|}{|f(x)|} \approx |\delta(x)| \frac{|f'(x)|}{|f(x)|}$$

$$\approx \frac{|\delta(x)|}{|x|} \cdot \left(\frac{|x||f'(x)|}{|f(x)|} \right)$$

so we let

$$\text{relative condition \# of } f \text{ at } x = \frac{|x||f'(x)|}{|f(x)|}.$$

(To generalize this to multivariable functions, we'll need to generalize $|\ |$, which we do ~~in a~~ shortly.)

The condition number is a measure of how <u>intrinsically</u> <u>difficult</u> it is to evaluate f. But we could also simply screw up the evaluation by choosing a sequence of floating point operations which compound roundoff error or something similar.

So here's a measure of algorithm quality:

Definition. If $alg(x)$ is result of a procedure for evaluating $f(x)$ in floating point, we say $alg(x)$ is <u>backward</u> <u>stable</u> if for all x there is some small δx so that

$$alg(x) = f(x + \delta x).$$

We call δx the <u>backwards error</u>.

In this case,

$$\text{error} = |\text{alg}(x) - f(x)| = |f(x+\delta x) - f(x)|$$
$$\approx |f'(x)|\,|\delta x|$$

so a backwards stable algorithm does well unless the condition # of f is large, (in which case you're always in trouble).