Notes on Scientific Computing

Nilay Kumar

Last updated: February 4, 2013

0 Administrativia

A standard reference on numerics is *Numerical Recipies.* For a standard programming book on C, Kernigham and Ritchie is an excellent place start to start.

There will be 6 to 8 homework sets and a final project. There will be a review session every Friday at 3pm.

0.1 Tentative topics

- Introduction to basic numerics and numerical stability
- Simple differential equation solvers
- Statistical mechanics of liquid argon; simulate $10^3 - 10^4$ interacting argon atoms; compute equation of state $f(P, V, T) = 0$
- Monte Carlo techniques for a canonical ensemble; MC used to simulate thermal fluctuations; random number generation
- General discussion of topics in statistics; applications to the above two topics
- Linear algebra algorithms; large systems (sparse matrices of order $n = 10^9$)
- Linear algebra on parallel computers
- **Time permitting:** General relativity; quantum Monte Carlo

1 Back recurrence

The Bessel functions are solutions to a partial differential equation in cylindrical coordinates. They satisfy the following recurrence relation:

$$J_{n+1}(x) = \frac{2n}{x} J_n(x) - J_{n-1}(x)$$

where n is an integer and x is real. Does knowing $J_0(x)$ and $J_1(x)$ allow you to know $J_n(x)$ for $n \geq 2$?

Let us write this relation as a 2 component vector

$$\begin{pmatrix} J_n \\ J_{n+1} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & \frac{2n}{x} \end{pmatrix} \begin{pmatrix} J_{n-1} \\ J_n \end{pmatrix}$$

What are the eigenvalues of this matrix?

$$\begin{vmatrix} -\lambda & 1 \\ -1 & \frac{2n}{x} - \lambda \end{vmatrix} = 0$$

which leads to

$$\lambda = \frac{n}{x} \pm \sqrt{\frac{n^2}{x^2} - 1}.$$

If $\frac{n}{x} > 1$, then $\lambda_+ > 1$ and $\lambda_- < 1$. Note that if we apply this recurrence relation m times, we will have λ_+ exponentially growing and λ_- exponentially decaying. Note also that the Bessel and Neumann functions are known to exponentially decay and grow respectively. So if we start with a $\tilde{J}_0(x)$ and $\tilde{J}_1(x)$ to some machine precision, we know that these can be written as

$$\tilde{J}_0(x) = \alpha J_0(x) + \beta N_0(x)$$
$$\tilde{J}_1(x) = \alpha J_1(x) + \beta N_1(x)$$

From this, we can determine α and β. As we iterate, however, the Neumann functions will dominate over the Bessel functions - i.e. we have numerical instability, as the Bessel function will get lost in the noise.

If we iterate backwards, however, with some random choice of the precision numbers, the resulting quantity will essentially be $\alpha J_0(x)$, as the growing term will be supressed by the back-recurrence. We can make an algorithm out of this using

$$1 = J_0(x) + J_2(x) + J_4(x) + \cdots$$

which determines the constant α. We are putting complete garbage in... and getting good results out!

2 High school trigonometry

Recall, last class we discussed the IEEE 754 standards for floating point numbers. First, there are the single precision numbers ($10^{\pm 38}$, 7 bits precision) and double precision numbers ($10^{\pm 308}$, 14 bits precision). We saw that the recurrence relation for Bessel functions is unstable to forward recurrence (i.e. fixing x and calculating higher order Bessel functions for that x), but is stable to back recurrence! This was because of the presence of both an exponentially growing and decaying solution to the recurrence formula.

Another example that concerns finite precision arithmetic is the following. Consider $\cos\left((m+1)x\right) = 2\cos(x)\cos(mx) - \cos\left((m-1)x\right)$. From $\cos(x)$, then, we might try to compute $\cos(Nx)$:

$$\begin{aligned}
\cos(2x) &= \cos(x) - 1 \\
\cos(3x) &= 2\cos(x)\cos(2x) - \cos(x) \\
\cos(4x) &= 2\cos(x)\cos(3x) - \cos(2x)
\end{aligned}$$

Consequently, we can write

$$\begin{pmatrix} \cos(mx) \\ \cos\left((m+1)x\right) \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -1 & 2\cos(x) \end{pmatrix} \begin{pmatrix} \cos\left((m-1)x\right) \\ \cos(mx) \end{pmatrix}$$

The eigenvalues of this matrix are found via the secular equation to be

$$\lambda = \cos(x) \pm i|\sin(x)|$$

Here, the magnitude of the eigenvalues is one, which is what one might expect for trigonometric functions. Thus, this recurrence relation looks much more stable than that of the Bessel functions; there are no exponentially growing/decaying solutions in the exact arithmetic. The question remains: is this an accurate way of computing $\cos(Nx)$ from $\cos(x)$? In other words, how accurately is $|\lambda| = 1$ represented in finite precision arithmetic?

$$\begin{aligned}
\cos(Nx) &= \cos(N\arccos(c)) \\
c &\equiv \cos(x)
\end{aligned}$$

How much would a small error in $\cos(x)$ affect $\cos(Nx)$? Well, take the floating point approximation to $\cos(x)$,

$$\tilde{c} = (1+\varepsilon)\cos(x).$$

Since

$$\frac{d\cos(Nx)}{dc} = \sin(Nx)\left(N\frac{dx}{dc}\right) = \frac{N\sin(Nx)}{\sin(x)},$$

we can write

$$\Delta\cos(Nx) \sim \frac{d\cos(Nx)}{dc}\Delta c = \frac{N\sin(Nx)}{\sin(x)}\varepsilon,$$

which suggests the size of the error given the initial uncertainty in c. Notice that this error is not uniform in the particular x that we are examining. Indeed, if one were trying to develop a good algorithm to compute $\cos(Nx)$, it would have to return $\cos(Nx)$ with essentially constant accuracy. In this case, it seems that if one starts with a small x, and iterates many times, the error would grow quite large.

Example 1. Using double precision values for $\cos(x)$ and double precision iteration, with $x = 10^{-6}$ and $N = 10^5$, one obtains using the above method,

$$\begin{aligned}\text{recurrence: } & 0.995003\mathbf{3695}\\ \text{MATLAB: } & 0.995004\mathbf{4165}\end{aligned}$$

Recall that via double precision, the accuracy of input was to the order of 13 digits or so, but the output is accurate only to about 6 digits. Again, this is completely due to the finite-precision errors in $|\lambda| \approx 1$.

3 Ordinary differential equations

3.1 Euler method

Consider a very general case:

$$\frac{dy^{(i)}}{dt} = g^{(i)}\left(y^{(j)}(t)\right)$$

Note that a lot of physics is second-order – this can be dealt with by choosing $y^{(1)} = x$ and $y^{(2)} = \frac{dx}{dt}$, which yields $2N$ linear differential equations. When dealing with such a system of differential equations, one first discretizes the equation in time,

$$\frac{dy^{(i)}}{dt} = \frac{y^{(i)}_{n+1} - y^{(i)}_n}{t_{n+1} - t_n} = g^{(i)}\left(y^{(i)}_n, t_n\right).$$

This discretization right away suggests the simple (but fairly poor) **Euler method**:

$$y_{n+1}^{(i)} = y_n^{(i)} + \tau g^{(i)} \left(y_n^{(j)} t_n \right)$$

This integrator is, of course, accurate to $O(\tau^2)^2$ per iteration, and thus, after N iterations, of order $O(N\tau^2)$. Since, quite generically, one wants to use $N \sim 1/\tau$, the Euler method has an $O(\tau)$ error, even in exact arithmetic.

Example 2 (Harmonic oscillator)**.** Let us take $m = 1$:

$$\frac{d^2x}{dt^2} = -\omega_0^2 x,$$

which, after defining

$$y^{(1)} = x$$
$$y^{(2)} = \frac{dx}{dt},$$

becomes

$$\frac{dy^{(1)}}{dt} = \frac{dx}{dt} = y^{(2)}$$
$$\frac{dy^{(2)}}{dt} = -\omega_0^2 y^{(1)}.$$

In matrix notation,

$$\frac{d}{dt} \begin{pmatrix} y^{(1)} \\ y^{(2)} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & 0 \end{pmatrix} \begin{pmatrix} y^{(1)} \\ y^{(2)} \end{pmatrix}$$

We can thus write

$$y_{n+1}^{(i)} = y_n^{(i)} + \tau M^{ij} y_n^{(i)} = \begin{pmatrix} 1 & \tau \\ -\tau\omega_0^2 & 1 \end{pmatrix} \begin{pmatrix} y^{(1)} \\ y^{(2)} \end{pmatrix}$$

Again, we have a linear recurrence relation, and so we ask for the eigenvalues of M. The secular equation yields

$$\lambda_\pm = 1 \pm \sqrt{-\omega_0^2 \tau^2} = 1 \pm i\omega_0\tau.$$

Note that the magnitude of both eigenvalues is more than one, so the Euler method is exponentially unstable for simple harmonic oscillation. Indeed, one tends to get oscillatory solutions bounded by a *growing* envelope. Of course, the smaller we choose our time step, τ, the slower this envelope will grow, but it will grow nonetheless. Another possible improvement would be to Taylor expand to a higher order.

3.2 Implicit method

$$\frac{y_{n+1}^{(i)} - y_n^{(i)}}{t_{n+1} - t_n} = g^{(i)}\left(y_{n+1}, t_{n+1}\right)$$
$$y_{n+1}^{(i)} = y_n^{(i)} + \tau g^{(i)}\left(y_{n+1}, t_{n+1}\right)$$

For simple harmonic motion, $g_{n+1}^{(i)} = M^{ij} y_{n+1}^{(i)}$ and $y_{n+1} = \left(\frac{1}{1-\tau M}\right) y_n$. The eigenvalues of this matrix look like

$$\lambda_{\pm} = \frac{1 \pm i\omega_0\tau}{1 + \omega_0^2\tau^2} < 1$$

which will give a *decreasing* envelope! Both the Euler method and the implicit method, unfortunately seem to destroy all our basic ideas of conservation of energy, etc.