Ordinary Differential EquationsODEs

Part 3

Massimo Ricotti

ricotti@astro.umd.edu

University of Maryland

Stiff ODEs

- A system of more than one ODE is stiff if solutions vary on two or more widely disparate lengthscales. E.g.,

$$y'' = 100y.$$

  - General solution: $y = Ae^{-10x} + Be^{+10x}$.
  - Suppose BCs are $y(0) = 1$, $y'(0) = -10$. Then $B = 0$, i.e., pure decaying solution.
  - Numerical technique would *begin* giving correct solution.
  - But once x becomes large, numerical solution will diverge exponentially, i.e., will contain growing exponential solution.

- Why? RE introduces admixture of growing solution, i.e.,

$$y_{\text{numerical}} = e^{-10x} + \varepsilon e^{+10x} \ (\varepsilon \ll 1).$$

  No matter how small ε is, it will dominate for $x \gg 1$.

- Common problem in hydrodynamics.

Explicit differencing

- Consider the system

$$y' = -cy \ (c > 0).$$

  - The solution is $y = e^{-ct}$, i.e., a decaying function.
  - Explicit or forward Euler differencing with stepsize h gives

$$y_{n+1} = y_n + hy'_n = (1 - hc)y_n.$$

    This is "explicit" because y_{n+1} is an explicit function of y_n.

- Method is unstable if $h > 2/c$ since this would mean $|y_n| \to \infty$ as $n \to \infty$.
  - Why? Suppose initial value is y_0. Integrate from 0 to k:

$$y_k = (1 - hc)^k y_0.$$

    This $\to \pm\infty$ (oscillating) if $1 - hc < -1$, i.e., if $h > 2/c$.
- In general, explicit differencing of stiff equations requires very small steps, i.e., long integration times.

Implicit differencing

- Simplest cure is to use implicit differencing.
  - Evaluate RHS at new y location.
  - Get backward Euler scheme:

$$y_{n+1} = y_n + hy'_{n+1} = y_n - hcy_{n+1},$$

or

$$y_{n+1} = \frac{y_n}{1+hc} = \left(\frac{1}{1+hc}\right)^n y_0.$$

(Get explicit formula only because our model equation is linear.)

  - Absolutely stable: $y_n \to 0$ as $h \to \infty$, which is correct asymptotic solution. (Note that $c < 0$ corresponds to $y = e^{ct}$, which not only is not stable, it is not supposed to be.)

- Give up *accuracy* (if we stick to a first-order method) for *stability* at large stepsizes.
- Nice analogy:

  > Imagine you are returning from a hike in the mountains. You are in a narrow canyon with steep walls on either side. An explicit algorithm would sample the local gradient to find the descent direction. But following the gradient on either side of the trail will send you bouncing back and forth from wall to wall. You will eventually get home, but it will be long after dark before you arrive. An implicit algorithm would have you keep your eyes on the trail and anticipate where each step is taking you. It is well worth the extra concentration.

Linear sets

- Can generalize to sets of linear equations with constant coefficients:

$$\mathbf{y}' = -\mathbf{C}\mathbf{y},$$

  where $\mathbf{C}$ is a positive definite matrix (i.e., symmetric with positive eigenvalues).

- Explicit differencing yields

$$\mathbf{y}_{n+1} = (\mathbf{1} - h\mathbf{C})\mathbf{y}_n.$$

  - Now $\mathbf{A}^n \to 0$ as $n \to \infty$ iff largest eigenvalue of $\mathbf{A}$ has magnitude less than 1. Thus for our system, require largest eigenvalue of $\mathbf{1} - h\mathbf{C}$ to be less than 1, or $h < 2/\lambda_{\max}$, where $\lambda_{\max}$ is largest eigenvalue of $\mathbf{C}$.

- Implicit differencing yields

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\mathbf{y}'_{n+1}, \\ &= (\mathbf{1} + h\mathbf{C})^{-1}\mathbf{y}_n. \end{aligned}$$

  - If eigenvalues of $\mathbf{C}$ are λ, then eigenvalues of $(\mathbf{1} + h\mathbf{C})^{-1}$ are $(1 + h\lambda)^{-1} < 1$ for all h (because $\lambda > 0$).
  - Price we pay for stability is that we must invert matrix.
- For nonlinear systems, must use iterative technique (like N-R with Jacobian).

Semi-implicit methods

For a generic non-linear system

$$\begin{aligned} \mathbf{y}' &= \mathbf{f}(\mathbf{y}) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + h\mathbf{f}(\mathbf{y_{n+1}}), \end{aligned}$$

In general this system has to be solved iteratively at each step. Or, we can try to linearizing the equations (like Newton's method):

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\left[\mathbf{f}(\mathbf{y}_n) + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_{y_n} (\mathbf{y_{n+1}} - \mathbf{y_n})\right], \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + h\left[1 - h\left.\frac{\partial \mathbf{f}}{\partial \mathbf{y}}\right|_{y_n}\right]^{-1} \mathbf{f}(\mathbf{y}_n), \end{aligned}$$

Higher-order methods

- Our implicit methods are all only 1^{st}-order accurate (Euler schemes).
- Easily get 2^{nd}-order method by averaging explicit and implicit steps:

$$y_{n+1} = y_n + h(y'_n + y'_{n+1})/2.$$

- Often called “Crank-Nicholson” differencing.
- Then, for our linear system, will get:

$$y_{n+1} = \left(\frac{1 - hc/2}{1 + hc/2}\right) y_n = \left(\frac{1 - hc/2}{1 + hc/2}\right)^n y_0.$$

- This is unconditionally stable, although in this case the behaviour as $h \to \infty$ is to oscillate (in bounded fashion) about $y = 0$.

- Second order because
  $(h/2)\left[y'(t) + y'(t+h)\right] = hy'(t) + (h^2/2)y''(t) + \mathcal{O}(h^3)$.
- *NRic* §16.6 provides generalizations of RK and BS for stiff systems.
- Actual routines are a bit more complex as they are written to achieve a desired accuracy

```
void simpr(float y[], float dydx[], float dfdx[], float **dfdy, int n,
float xs, float htot, int nstep, float yout[],
void (*derivs)(float, float [], float []))

void lubksb(float **a, int n, int *indx, float b[]);
void ludcmp(float **a, int n, int *indx, float *d);
int i,j,nn,*indx;
float d,h,x,**a,*del,*ytemp;

indx=ivector(1,n);
a=matrix(1,n,1,n);
del=vector(1,n);
ytemp=vector(1,n);
h=htot/nstep;
for (i=1;i<=n;i++)
for (j=1;j<=n;j++) a[i][j] = -h*dfdy[i][j];
++a[i][i];

ludcmp(a,n,indx,&d);
for (i=1;i<=n;i++)
yout[i]=h*(dydx[i]+h*dfdx[i]);
lubksb(a,n,indx,yout);
for (i=1;i<=n;i++)
ytemp[i]=y[i]+(del[i]=yout[i]);
x=xs+h;
(*derivs)(x,ytemp,yout);
for (nn=2;nn<=nstep;nn++)
for (i=1;i<=n;i++)
yout[i]=h*yout[i]-del[i];
lubksb(a,n,indx,yout);
for (i=1;i<=n;i++)
ytemp[i] += (del[i] += 2.0*yout[i]);
x += h;
(*derivs)(x,ytemp,yout);

for (i=1;i<=n;i++)
yout[i]=h*yout[i]-del[i];
lubksb(a,n,indx,yout);
for (i=1;i<=n;i++)
yout[i] += ytemp[i];
free_vector(ytemp,1,n);
free_vector(del,1,n);
free_matrix(a,1,n,1,n);
free_ivector(indx,1,n);
```