Lecture 13
Ordinary differential equations

- Embedded Runge-Kutta
- Modified midpoint method.
- Bulirsch-Stoer method.
- Predictor-corrector methods.

Embedded Runge-Kutta:

The Runge-Kutta formulas with M>4 require more than M evaluations of the function. This in part explains why physicists settled with M=4, it kind of gives the most bang for the buck.

If one is using error control however, there is an additional variable to minimize: how much does it cost to compute the error. Fehlberg noted that one could write a fifth order RK routine that required six evaluations of the function and one could also write a fourth order formula requiring the same six evaluations.

For a while people considered these formulas risky (!) because it was “less pure” to use the same points for both evaluations. Of course there is no basis for such fears, and experience has shown that the Fehlberg construction is useful.

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + a_2 h, y_n + b_{21} k_1)$$

$$\cdots$$

$$k_6 = hf(x_n + a_6 h, y_n + b_{61} k_1 + \cdots + b_{65} k_5)$$

$$y_{n+1} = y_n + c_1 k_1 + c_2 k_2 + c_3 k_3 + c_4 k_4 + c_5 k_5 + c_6 k_6 + O(h^6)$$

Fifth order RK

Fourth order evaluation:

$$y^*_{n+1} = y_n + c^*_1 k_1 + c^*_2 k_2 + c^*_3 k_3 + c^*_4 k_4 + c^*_5 k_5 + c^*_6 k_6 + O(h^5)$$

Cash-Karp Parameters for Embedded Runga-Kutta Method

i	a_i	b_{ij}					c_i	c^*_i
1							$\frac{37}{378}$	$\frac{2825}{27648}$
2	$\frac{1}{5}$	$\frac{1}{5}$					0	0
3	$\frac{3}{10}$	$\frac{3}{40}$	$\frac{9}{40}$				$\frac{250}{621}$	$\frac{18575}{48384}$
4	$\frac{3}{5}$	$\frac{3}{10}$	$-\frac{9}{10}$	$\frac{6}{5}$			$\frac{125}{594}$	$\frac{13525}{55296}$
5	1	$-\frac{11}{54}$	$\frac{5}{2}$	$-\frac{70}{27}$	$\frac{35}{27}$		0	$\frac{277}{14336}$
6	$\frac{7}{8}$	$\frac{1631}{55296}$	$\frac{175}{512}$	$\frac{575}{13824}$	$\frac{44275}{110592}$	$\frac{253}{4096}$	$\frac{512}{1771}$	$\frac{1}{4}$
$j =$		1	2	3	4	5		

Error estimate:

$$\Delta \equiv y_{n+1} - y^*_{n+1} = \sum_{i=1}^{6} (c_i - c^*_i) k_i$$

```
      SUBROUTINE rkqs(y,dydx,n,x,htry,eps,yscal,hdid,hnext,derivs)
            INTEGER n,NMAX, i
            REAL eps,hdid,hnext,htry,x,dydx(n),y(n),yscal(n)
            EXTERNAL derivs
            PARAMETER (NMAX=50)
            REAL errmax,h,xnew,yerr(NMAX),ytemp(NMAX),SAFETY,PGROW,PSHRNK,
           *ERRCON
            PARAMETER (SAFETY=0.9,PGROW=-.2,PSHRNK=-.25,ERRCON=1.89e-4)
            h=htry
      1     call rkck(y,dydx,n,x,h,ytemp,yerr,derivs)
            errmax=0.
            do 11 i=1,n
              errmax=max(errmax,abs(yerr(i)/yscal(i)))
      11    continue
            errmax=errmax/eps
            if(errmax.gt.1.)then
              h=SAFETY*h*(errmax**PSHRNK)
              if(h.lt.0.1*h)then
                h=.1*h
              endif
              xnew=x+h
              if(xnew.eq.x)pause 'stepsize underflow in rkqs'
              goto 1
              else
              if(errmax.gt.ERRCON)then
                hnext=SAFETY*h*(errmax**PGROW)
              else
                hnext=5.*h
              endif
              hdid=h
              x=x+h
              do 12 i=1,n
                y(i)=ytemp(i)
      12        continue
              return
            endif
            END
```

Call RK with Cash-Karp

Compare with scale, choose max

Shrink/grow stepsize, not more than 10 or 5-fold

If shrink, go back

All checks done, load final values

```
 SUBROUTINE rkck(y,dydx,n,x,h,yout,yerr,derivs)
      INTEGER n,NMAX
      REAL h,x,dydx(n),y(n),yerr(n),yout(n)
      EXTERNAL derivs
      PARAMETER (NMAX=50)
CU    USES derivs
      INTEGER i
      REAL ak2(NMAX),ak3(NMAX),ak4(NMAX),ak5(NMAX),ak6(NMAX),
     *ytemp(NMAX),A2,A3,A4,A5,A6,B21,B31,B32,B41,B42,B43,B51,B52,B53,
     *B54,B61,B62,B63,B64,B65,C1,C3,C4,C6,DC1,DC3,DC4,DC5,DC6
      PARAMETER (A2=.2,A3=.3,A4=.6,A5=1.,A6=.875,B21=.2,B31=3./40.,
     *B32=9./40.,B41=.3,B42=-.9,B43=1.2,B51=-11./54.,B52=2.5,
     *B53=-70./27.,B54=35./27.,B61=1631./55296.,B62=175./512.,
     *B63=575./13824.,B64=44275./110592.,B65=253./4096.,C1=37./378.,
     *C3=250./621.,C4=125./594.,C6=512./1771.,DC1=C1-2825./27648.,
     *DC3=C3-18575./48384.,DC4=C4-13525./55296.,DC5=-277./14336.,
     *DC6=C6-.25)
      do 11 i=1,n
        ytemp(i)=y(i)+B21*h*dydx(i)
11    continue
      call derivs(x+A2*h,ytemp,ak2)
```

Cash-Karp parameters

```
 do 12 i=1,n
        ytemp(i)=y(i)+h*(B31*dydx(i)+B32*ak2(i))
12    continue
      call derivs(x+A3*h,ytemp,ak3)
      do 13 i=1,n
        ytemp(i)=y(i)+h*(B41*dydx(i)+B42*ak2(i)+B43*ak3(i))
13    continue
      call derivs(x+A4*h,ytemp,ak4)
      do 14 i=1,n
        ytemp(i)=y(i)+h*(B51*dydx(i)+B52*ak2(i)+B53*ak3(i)+B54*ak4(i))
14    continue
      call derivs(x+A5*h,ytemp,ak5)
      do 15 i=1,n
        ytemp(i)=y(i)+h*(B61*dydx(i)+B62*ak2(i)+B63*ak3(i)+B64*ak4(i)+
     *B65*ak5(i))
15    continue
      call derivs(x+A6*h,ytemp,ak6)
      do 16 i=1,n
        yout(i)=y(i)+h*(C1*dydx(i)+C3*ak3(i)+C4*ak4(i)+C6*ak6(i))
16    continue
      do 17 i=1,n
        yerr(i)=h*(DC1*dydx(i)+DC3*ak3(i)+DC4*ak4(i)+DC5*ak5(i)+DC6*
     *ak6(i))
17    continue
      return
      END
```

Perform the six evaluations of 5th order RK

Cash-Karp fourth order estimate

Modified midpoint method:

This method advances a set of ODE's from a point x to a point x+H through n steps of size H/n. The method is just Euler's, corrected to have a midpoint derivative, with the exception of the endpoints,

$$z_0 \equiv y(x)$$

$$z_1 = z_0 + hf(x, z_0)$$

$$z_{m+1} = z_{m-1} + 2hf(x + mh, z_m)$$

$$y(x + H) \approx y_n \equiv \frac{1}{2}[z_n + z_{n-1} + hf(x + H, z_n)]$$

This method requires only n+1 evaluations of the RHS, and it is second-order. So it compares favorably with the 2n evaluations that second order RK requires. The method is easily coded, but in practice the error-controlled RK we discussed last class will outperform it.

The usefulness of this method comes as an intermediate step in the more powerful Bulirsch-Stoer method.

The value of the modified-midpoint method lies in a long-to-prove result due to Gragg, that the error in the method contains only even powers of h,

$$y_n - y(x + H) = \sum_{i=1}^{\infty} \alpha_i h^{2i}$$

You should remember a similar result we already encountered in this course: the proof that the trapezoidal rule of integration had a similar property. At that point, we capitalized on this extra piece of knowledge: by knowing how the error depends on h, one can cancel the leading term in the error.

This procedure had a generic name, it was called “Richardson extrapolation” and was based on the idea that whenever one uses a finite difference h, one is really interested in the limit h->0.

That this procedure has advantages should be clear from the following naïve implementation. Suppose I take an (even) number of steps n, and compute,

$$y(x+H) \approx \frac{4y_n - y_{n/2}}{3}$$

The resulting approximation is fourth order accurate, yet it requires only 1.5 times the evaluation of derivatives per step h against RK's four evaluations.

If the 4/3 factor sounds familiar, remember the transition from trapezoid to Simpson's rule in integration.

```
 subroutine mmid(y,dydx,nvar,xs,htot,nstep,yout,derivs)
      parameter (nmax=10)
      dimension y(nvar),dydx(nvar),yout(nvar),ym(nmax),yn(nmax)
      h=htot/nstep
      do 11 i=1,nvar
        ym(i)=y(i)
        yn(i)=y(i)+h*dydx(i)
11    continue
      x=xs+h
      call derivs(x,yn,yout)
      h2=2.*h
      do 13 n=2,nstep
        do 12 i=1,nvar
          swap=ym(i)+h2*yout(i)
          ym(i)=yn(i)
          yn(i)=swap
12      continue
        x=x+h
        call derivs(x,yn,yout)
13    continue
      do 14 i=1,nvar
        yout(i)=0.5*(ym(i)+yn(i)+h*yout(i))
14    continue
      return
      end
```

The Bulirsch-Stoer method:

We already have the key idea for the method: to use modified midpoint and Richardson extrapolate the result. As we did with the Romberg method of integration, we must caution here about accuracy vs. order. Richardson extrapolation only works well with smooth functions. If your functions are not smooth (for instance if the derivatives come from a table of numbers specified by some "black box") then adaptive RK is the way to go. Similar considerations apply to the case of functions with singularities. For smooth functions however, Bulirsch-Stoer is the method of choice.

Bulirsch and Stoer also propose using a *rational function* approach to perform the Richardson extrapolation. This allows them to consider frighteningly large values of h and still get meaningful results.

The method therefore consists in taking a leap from point x to x+H, where H is not at all small, via a succession n of modified-midpoint steps. One repeats the procedure with increasing values of n, for instance, n=2,4,6,8,10,… and each time one chooses an n one extrapolates the result to zero stepsize. The extrapolation returns an error estimate and if it is not satisfactory, the value of n is increased.

We said that H could be big. It could be the whole interval of integration. That in general will not be good. In practice one limits the number of subdivisions (say, to less than 16) and if convergence is not reached, then one simply reduces the value of H.

Error control is a bit trickier than in Runge-Kutta. The routines that Numerical Recipes include have quite sophisticated methods of controlling errors. They exceed the scope of this course.
I will show you an older version from an earlier edition of the book that is more understandable.

```
      subroutine bsstep(y,dydx,nv,x,htry,eps,yscal,hdid,hnext,derivs)
            parameter (nmax=10,imax=11,nuse=7,one=1.e0,shrink=.95e0,grow=1.2e0
           *)
            dimension y(nv),dydx(nv),yscal(nv),yerr(nmax),
           *    ysav(nmax),dysav(nmax),yseq(nmax),nseq(imax)
            data nseq /2,4,6,8,12,16,24,32,48,64,96/
            h=htry
            xsav=x
            do 11 i=1,nv
              ysav(i)=y(i)
              dysav(i)=dydx(i)
      11    continue
      1     do 10 i=1,imax
              call mmid(ysav,dysav,nv,xsav,h,nseq(i),yseq,derivs)
              xest=(h/nseq(i))**2
              call rzextr(i,xest,yseq,y,yerr,nv,nuse)
              errmax=0.
              do 12 j=1,nv
                errmax=max(errmax,abs(yerr(j)/yscal(j)))
      12      continue
              errmax=errmax/eps
                      if(errmax.lt.one) then
                        x=x+h
                        hdid=h
                        if(i.eq.nuse)then
                          hnext=h*shrink
                        else if(i.eq.nuse-1)then
                          hnext=h*grow
                        else
                          hnext=(h*nseq(nuse-1))/nseq(i)
                        endif
                        return
                      endif
              10    continue
                    h=0.25*h/2**((imax-nuse)/2)
                    if(x+h.eq.x)pause 'step size underflow.'
                    goto 1
```

Call modified midpoint

Extrapolate

Compute errors, decide if to shrink or grow stepsize

Predictor-corrector methods:

There seems to be growing consensus that these methods have had their day. For high precision applications where right hand sides are expensive to exaluate, Bulirsch-Stoer dominates. For low precision, complicated to evaluate RHSs, Runge-Kutta is the method of choice. Predictor-corrector methods are “squeezed in the middle”, perhaps only useful if one needs high-precision, smooth solutions with expensive to compute RHSs.

To describe the method, let us consider the ODE step as an integral,

$$y(x) = y_n + \int_{x_n}^{x} f(x', y)\, dx'$$

And approximate the integral using several points x_i,

$$y_{n+1} = y_n + h(\beta_0 y'_{n+1} + \beta_1 y'_n + \beta_2 y'_{n-1} + \beta_3 y'_{n-2} + \cdots)$$

Where $y'_n = f(x_n, y_n)$. One is then left with an implicit equation for y_{n+1}

How does one solve for y_{n+1}? One could use Newton-Raphson or an iterative technique, starting with a guess for y_{n+1} and applying the formula repeated times.

Where to get the initial guess? One possibility is to rewrite the formula using an approximation to the integral that does not involve y_{n+1}. This first step is an extrapolation of the value and is called the "predictor" step. The next step, in which one solves the implicit equation is an interpolation and is called the "corrector" step.

Why not run the corrector step again? One should bear in mind that the order of the extrapolation is fixed and remains unchanged if one repeats the step. At best one is changing the coefficient in front of the error term. If one wants more accuracy, one is better off just reducing the stepsize.

There are as many variants of the predictor-corrector method as interpolation and extrapolation formulas we could write for the integral. A popular one is the Adams-Bashforth-Moulton choice,

$$\text{predictor:} \qquad y_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}) + O(h^4)$$

$$\text{corrector:} \qquad y_{n+1} = y_n + \frac{h}{12}(5y'_{n+1} + 8y'_n - y'_{n-1}) + O(h^4)$$

A predictor-corrector method with one step is an explicit method (simply substitute the first formula in the second one). Explicit methods are not good for stiff problems (we'll discuss these soon), so in such case one would like to use some implicit scheme using Newton-Raphson to solve for y_{n+1}.

All these methods are awkward to get going since one needs many points to take the first step. Some get "primed" with RK steps to get going. Changing stepsize is also complicated.

Summary

- Embedded RK uses clever trick to compute the error.
- Bulirsch-Stoer is a Richardson extrapolation of the modified midpoint method.
- Predictor-corrector methods used to be popular, now relegated to specific applications.