Computational Physics 210: O.D.E. Notes
Ordinary Differential Equations

January 11, 2010

1. The Simple Form

Problems in physics and numerous other science and engineering fields commonly involve differential equations. Integrating differential equations is one of the most common computational tasks. There are many types of differential equations, the simplest of which is the *ordinary differential equation*. In ODEs, there is differentiation with respect to only one variable, which we can denote x. It can be a space coordinate, but it can also be time or even temperature or other continuous 'variables.'

A very general form of ODE is the *first order* form (containing only first derivatives)

$$\frac{d\vec{Y}}{dx} = \vec{f}(x, \vec{Y}), \tag{1}$$

which is a vector equation for several (say, M) functions (components of $\vec{Y}$) that are functions of x and in general are coupled to one another. A simple formal solution is

$$Y(x) = \int_{x_\circ}^{x} f(x', Y(x'))dx' + C \tag{2}$$

for some (interrelated) constants $x_\circ$ and C determined by an initial condition.

The reason that this form is so general is that higher derivatives can be handled by introducing auxiliary functions as components of $\vec{Y}$. Newton's force law provides an illustration (using the variable t instead of x):

$$m\frac{d^2z}{dt^2} = F(z). \tag{3}$$

Define the momentum of the particle with coordinate z as

$$p(t) = m\frac{dz}{dt}; \frac{dz}{dt} = \frac{p}{m} \tag{4}$$

so that

$$\frac{dp}{dt} = F(z), \tag{5}$$

and the equation becomes

$$\frac{d}{dt}\begin{bmatrix} z(t) \\ p(t) \end{bmatrix} = \begin{bmatrix} p(t)/m \\ F(t) \end{bmatrix}, \quad i.e. \quad \frac{d}{dt}\vec{Y}(t) = \vec{f}[x(t), \vec{Y}(t)], \tag{6}$$

where $\vec{Y} = (z, p)$ and $\vec{f} = (p/m, F)$. To generalize, a second order ODE of a vector function of M components becomes a first order ODE of 2M components. And there is nothing special about this vector equation; exactly the same type of thing is done on each component (except that errors may build differently in coupled equations). Thus it is sufficient to consider initially the single first order ODE

$$\frac{dy}{dx} = f(x, y), \tag{7}$$

which is what we concentrate on solving. Here f is some given function that is readily evaluated numerically.

2. Simple Approximation for the Derivative

To simplify things, we consider the domain to be finite and scale the length to unity: $0 \le x \le 1$. Further, this interval is divided into N equal steps of length $h = 1/N$. Having already studied numerical approximations to a first derivative, we know the simplest approximations to the derivative at x_n are the right and left two-point formulas, the right one being

$$\frac{dy_n}{dx} = \frac{y_{n+1} - y_n}{h} + \mathcal{O}(h) = f(x_n, y_n), \tag{8}$$

so presuming we have the values at x_n, solving for y_{n+1} gives

$$\begin{aligned} y_{n+1} &= y_n + hf(x_n, y_n) + \mathcal{O}(h^2), \\ y_{n+2} &= y_{n+1} + hf(x_{n+1}, y_{n+1}) + \mathcal{O}(h^2). \end{aligned} \tag{9}$$

Given this formula and the boundary condition $y(x = 0) = y_o$, one can obtain recursively the values for $y_1, y_2, ..., y_N$. It is noted in this formula that the error is of order h^2; this is the error *at this step*. Since there are N steps, the total error that has accumulated by the time the process is iterated out to n=N is N$\mathcal{O}(h^2) = \mathcal{O}(h)$. Thus, taking only half as large a step size, hence taking twice as much time, reduces the likely error only by a factor of two. This is unacceptably slow "convergence" to the solution for most problems.

So – what is a reasonable improvement to try?

3. An example

Let us first try a modest example, but not too modest. The differential equation is

$$\frac{dy}{dx} = -3x^2 y. \tag{10}$$

Dividing through by y, integrating both sides to get $ln(y)$ on the left, and exponentiating, one finds the analytic answer is

$$y = e^{-x^3}, \tag{11}$$

where we have built in a boundary condition $y(0) = 1$. We now look at integrating our ODE through the range $-5 \leq x \leq 5$. [We could scale the problem onto the interval $-1 \leq x \leq 1$, would involve dealin with the function $y = exp(-125x^3)$.]

The code for integrating this equation, is given explicitly below, along with outputs for the cases of N = 10^3, 10^4, 10^5, and 10^6 points.

Before coding, however, one should visualize the task that is being undertaken. The function varies from $exp(+125)$ to $exp(-125)$ over the range, that is, by $(250)/log_e 10 \approx 110$ *orders of magnitude*! Perhaps it is silly to even try such a thing, not to mention that the algorithm is the most simplistic one imaginable. Still, it may be instructive to try. Here are the results. Note that the code uses double precision ("implicit real8").

```
      Program EulerODE
      implicit real*8 (a-h,o-z)   %% <== N.B. double precision
c  Euler's method to solve dy/dx = -x^2 y, y(0)=1.
c     We use the interval [-Xmax,Xmax] with N intervals
c
c   Forward difference: y(n)   = y(n-1) * (1-h*x(n)**2)
c  Backward difference: y(n-1) = y(n)   * (1+j*x(n)**2)
c   (obtained by h --> -h, hence indices decreasing)
c
c     parameter (N=100, Xmax=5.)
c     dimension y(-N:N),diff(-N:N)
      dimension y(-1000000:1000000),diff(-1000000:1000000)
c
c  A Statement Function, defined before any executable statement
c
      yexact(x) = exp(-x**3)
c
c          N=100000
      read *, N
          Nprt=N/10
            nn=10
          Xmax=5.
            write(91,*) N,' integration points'
        y(0)=1.
        diff(0)=0.
c  Make sure N is an even number
                        N = 2*(N/2)
        h = 2.*Xmax/real(N)
c
c  Begin marching right & left, integrating the ODE
c
      write(91,2)
      do j=1,N
       xn=h*(j-1)
       x =h*j
        y( j) = y( j-1) * (1. - h * 3.d0 * xn * xn)
        y(-j) = y(-j+1) * (1. + h * 3.d0 * xn * xn)
        diff( j) = y( j) - yexact( x)
        diff(-j) = y(-j) - yexact(-x)
      end do
c
      write(91,1) (j*h, y(j), diff(j), diff(j)/y(j),diff(j)/y(j),
     *                j=-N/2,N/2,Nprt)
c
      stop
    1 format(6x,f8.4,3e15.7,f14.5)
    2 format(10x,'x',8x,'y(calc)',10x,'error',2(7x,'rel.error'))
      end
```

```
1000 integration points
         x        y(calc)          error       rel.error       rel.error
      -5.0000  0.1051086E+46 -0.1935576E+55 -0.1841500E+10**************
      -4.0000  0.2669011E+25 -0.6232480E+28 -0.2335127E+04   -2335.12692
      -3.0000  0.6761480E+11 -0.4644334E+12 -0.6868813E+01      -6.86881
      -2.0000  0.2144826E+04 -0.8361324E+03 -0.3898370E+00      -0.38984
      -1.0000  0.2654870E+01 -0.6341231E-01 -0.2388528E-01      -0.02389
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3701108E+00  0.2231377E-02  0.6028943E-02       0.00603
       2.0000  0.2633820E-03 -0.7208060E-04 -0.2736732E+00      -0.27367
       3.0000  0.1765773E-12 -0.1702952E-11 -0.9644227E+01      -9.64423
       4.0000  0.1122580E-32 -0.1603800E-27 -0.1428673E+06 -142867.29909
       5.0000  0.2536642E-74 -0.5166421E-54 -0.2036716E+21**************
10000 integration points
      -5.0000  0.1234405E+54 -0.1812136E+55 -0.1468024E+02     -14.68024
      -4.0000  0.2473554E+28 -0.3761595E+28 -0.1520725E+01      -1.52073
      -3.0000  0.4230469E+12 -0.1090014E+12 -0.2576579E+00      -0.25766
      -2.0000  0.2879581E+04 -0.1013768E+03 -0.3520541E-01      -0.03521
      -1.0000  0.2711777E+01 -0.6505132E-02 -0.2398845E-02      -0.00240
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3681004E+00  0.2209651E-03  0.6002848E-03       0.00060
       2.0000  0.3278578E-03 -0.7604787E-05 -0.2319538E-01      -0.02320
       3.0000  0.1526761E-11 -0.3527682E-12 -0.2310567E+00      -0.23106
       4.0000  0.6399927E-28 -0.9638182E-28 -0.1505983E+01      -1.50598
       5.0000  0.2904248E-55 -0.4875996E-54 -0.1678919E+02     -16.78919
100000 integration points
      -5.0000  0.1457058E+55 -0.4785184E+54 -0.3284142E+00      -0.32841
      -4.0000  0.5673796E+28 -0.5613535E+27 -0.9893791E-01      -0.09894
      -3.0000  0.5198520E+12 -0.1219628E+11 -0.2346107E-01      -0.02346
      -2.0000  0.2970608E+04 -0.1034972E+02 -0.3484041E-02      -0.00348
      -1.0000  0.2717630E+01 -0.6521997E-03 -0.2399884E-03      -0.00024
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3679015E+00  0.2207514E-04  0.6000284E-04       0.00006
       2.0000  0.3346982E-03 -0.7644175E-06 -0.2283901E-02      -0.00228
       3.0000  0.1841305E-11 -0.3822344E-13 -0.2075888E-01      -0.02076
       4.0000  0.1465823E-27 -0.1379875E-28 -0.9413652E-01      -0.09414
       5.0000  0.3910579E-54 -0.1255842E-54 -0.3211397E+00      -0.32114
1000000 integration points
      -5.0000  0.1881210E+55 -0.5436609E+53 -0.2889954E-01      -0.02890
      -4.0000  0.6176481E+28 -0.5866832E+26 -0.9498665E-02      -0.00950
      -3.0000  0.5308144E+12 -0.1233824E+10 -0.2324398E-02      -0.00232
      -2.0000  0.2979921E+04 -0.1037133E+01 -0.3480404E-03      -0.00035
      -1.0000  0.2718217E+01 -0.6523688E-04 -0.2399988E-04      -0.00002
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3678816E+00  0.2207300E-05  0.6000028E-05       0.00001
       2.0000  0.3353861E-03 -0.7648111E-07 -0.2280390E-03      -0.00023
       3.0000  0.1875675E-11 -0.3853332E-14 -0.2054370E-02      -0.00205
       4.0000  0.1589476E-27 -0.1433465E-29 -0.9018473E-02      -0.00902
       5.0000  0.5024974E-54 -0.1414470E-55 -0.2814881E-01      -0.02815
```

The message is that this method, using the simple Euler formula for the derivative, will work if it is really pushed. When "only" 10^4 points were used, the results are total nonsense near the endpoints of the range. For 10^5 points, the error at the endpoints is still bad but at least the result is beginning to make sense. For a million points the error is about 3% at the endpoints, about a factor of ten smaller than for 10 times fewer points. This is in fact exactly what we should have guessed, since the error is $\mathcal{O}(1/\mathrm{N})$. So — yes, the methods works, but only if one is willing to use a ten million points or more to get what still amounts to mediocrity.

Of course, if we know the analytic form of $f(x, y)$ in this case, we might have the sense to make the change of variable $x^3 \rightarrow \tau$, with solution $exp(-\tau)$ which is managable numerically over a much larger domain. However, often $f(x, y)$ is only known numerically, and substitution of variables is much less obvious.

4. The Next Level Approximation

One can do better by invoking the Taylor series to approximate the derivative. This leads to a class of improvements. One writes

$$y_{n+1} = y(x_n + h) = y_n + hy'_n + \frac{1}{2}y''_n + \mathcal{O}(h^3). \tag{12}$$

Since our ODE is

$$y'_n = f(x_n, y_n), \tag{13}$$

we substitute into Eq. (11) to obtain

$$y_n'' = \frac{df}{dx}(x_n, y_n) = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}\frac{dy}{dx} = \frac{\partial f}{\partial x} + f\frac{\partial f}{\partial y}, \tag{14}$$

substituting these in leads to

$$y_{n+1} = y_n + hf_n + \frac{1}{2}h^2[\frac{\partial f}{\partial x} + f\frac{\partial f}{\partial y}]_n + \mathcal{O}(h^3). \tag{15}$$

Here f and its partial derivatives are evaluated at x_n. (This is a known function; if it is only "known" numerically, then its derivatives have to be evaluated numerically.) This equation can be compared with Eq. 9. Evidently the term in Eq. 15 is precisely the 2nd order correction that was needed to make the error one order higher in h. The application to the same ODE follows.

```
      Program TaylorODE
      implicit real*8 (a-h,o-z)
c  Euler's method to solve dy/dx = -x^2 y, y(0)=1.
c     We use the interval [-Xmax,Xmax] with N intervals
c
c   Forward difference: y(n)   = y(n-1) * (1-h*x(n)**2)
c  Backward difference: y(n-1) = y(n)   * (1+j*x(n)**2)
c   (obtained by h --> -h, hence indices decreasing)
c
c     parameter (N=100, Xmax=5.)
c     dimension y(-N:N),diff(-N:N)
      dimension y(-1000000:1000000),diff(-1000000:1000000)
c
c  A Statement Function, defined before any executable statement
c
c    Some exact information is coded in explicitly
c
      yexact(x) = exp(-x**3)
      f(x,y)    = - 3.d0 * x * x * y
      dfdx(x,y) = - 6.d0 *     x * y
      dfdy(x,y) = - 3.d0 * x * x

c
c          N=100000
      print *, ' input N'
      read *, N
          Nprt=N/10
            nn=10
          Xmax=5.
            write(91,*) N,' integration points'
        y(0)=1.
        diff(0)=0.
c  Make sure N is an even number
                        N = 2*(N/2)
        h0 = 2.*Xmax/real(N)
c
c  Begin marching right & left, integrating the ODE
c
      write(91,2)
      do j=0,N-1
       h  = h0
       xj1=h*(j+1)
       xj =h*j
       yj=y(j)
        y(j+1) = y(j) + h*f(xj,yj) + 0.5d0*h*h*
     *                ( dfdx(xj,yj) + f(xn,yn)*dfdy(xj,yj) )
       diff( j+1) = y( j+1) - yexact( xj1)
       h  = -h
       xj =-xj
```

```
        xj1=-xj1
        yj =y(-j)
         y(-j-1) = y(-j) + h*f(xj,yj) + 0.5d0*h*h*
     *                ( dfdx(xj,yj) + f(xn,yn)*dfdy(xj,yj) )
         diff(-j-1) = y(-j-1) - yexact(xj1)
       end do
c
       write(91,1) (j*h0, y(j), diff(j), diff(j)/y(j),diff(j)/y(j),
     *                j=-N/2,N/2,Nprt)
c
       stop
     1 format(6x,f8.4,3e15.7,f14.5)
     2 format(10x,'x',8x,'y(calc)',10x,'error',2(7x,'rel.error'))
       end
```

```
1000 integration points     using 2nd order Taylor integration
         x        y(calc)          error       rel.error       rel.error
      -5.0000  0.6414876E+53 -0.1871427E+55 -0.2917324E+02     -29.17324
      -4.0000  0.2677583E+28 -0.3557566E+28 -0.1328649E+01      -1.32865
      -3.0000  0.4644528E+12 -0.6759547E+11 -0.1455379E+00      -0.14554
      -2.0000  0.2947825E+04 -0.3313332E+02 -0.1123992E-01      -0.01124
      -1.0000  0.2717247E+01 -0.1034814E-02 -0.3808319E-03      -0.00038
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3678570E+00 -0.2247412E-04 -0.6109473E-04      -0.00006
       2.0000  0.3371952E-03  0.1732524E-05  0.5138045E-02       0.00514
       3.0000  0.2168918E-11  0.2893893E-12  0.1334257E+00       0.13343
       4.0000  0.5956664E-27  0.4352853E-27  0.7307535E+00       0.73075
       5.0000  0.8460667E-51  0.8455501E-51  0.9993894E+00       0.99939
10000 integration points
      -5.0000  0.1842241E+55 -0.9333530E+53 -0.5066401E-01      -0.05066 <--
      -4.0000  0.6168147E+28 -0.6700196E+26 -0.1086257E-01      -0.01086
      -3.0000  0.5312158E+12 -0.8324842E+09 -0.1567130E-02      -0.00157
      -2.0000  0.2980602E+04 -0.3559093E+00 -0.1194085E-03      -0.00012
      -1.0000  0.2718271E+01 -0.1055844E-04 -0.3884249E-05       0.00000
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3678792E+00 -0.2234934E-06 -0.6075184E-06       0.00000
       2.0000  0.3354789E-03  0.1630348E-07  0.4859761E-04       0.00005
       3.0000  0.1881872E-11  0.2342949E-14  0.1245010E-02       0.00125
       4.0000  0.1620321E-27  0.1650990E-29  0.1018928E-01       0.01019
       5.0000  0.5436749E-54  0.2703285E-55  0.4972245E-01       0.04972
100000 integration points
      -5.0000  0.1934581E+55 -0.9947896E+51 -0.5142144E-03      -0.00051 <--
      -4.0000  0.6234458E+28 -0.6908953E+24 -0.1108188E-03      -0.00011
      -3.0000  0.5320398E+12 -0.8452213E+07 -0.1588643E-04      -0.00002
      -2.0000  0.2980954E+04 -0.3583018E-02 -0.1201970E-05       0.00000
      -1.0000  0.2718282E+01 -0.1057954E-06 -0.3891995E-07       0.00000
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3678794E+00 -0.2233689E-08 -0.6071796E-08       0.00000
       2.0000  0.3354628E-03  0.1620853E-09  0.4831691E-06       0.00000
       3.0000  0.1879552E-11  0.2309156E-16  0.1228567E-04       0.00001
       4.0000  0.1603971E-27  0.1602011E-31  0.9987783E-04       0.00010
       5.0000  0.5168955E-54  0.2533987E-57  0.4902320E-03       0.00049
1000000 integration points
      -5.0000  0.1935566E+55 -0.9989829E+49 -0.5161192E-05      -0.00001 <--
      -4.0000  0.6235142E+28 -0.6926895E+22 -0.1110944E-05       0.00000
      -3.0000  0.5320482E+12 -0.8464481E+05 -0.1590924E-06       0.00000
      -2.0000  0.2980958E+04 -0.3585403E-04 -0.1202769E-07       0.00000
      -1.0000  0.2718282E+01 -0.1058239E-08 -0.3893045E-09       0.00000
       0.0000  0.1000000E+01  0.0000000E+00  0.0000000E+00       0.00000
       1.0000  0.3678794E+00 -0.2233741E-10 -0.6071937E-10       0.00000
       2.0000  0.3354626E-03  0.1619897E-11  0.4828846E-08       0.00000
       3.0000  0.1879529E-11  0.2305934E-18  0.1226868E-06       0.00000
       4.0000  0.1603812E-27  0.1597932E-33  0.9963333E-06       0.00000
       5.0000  0.5166446E-54  0.2523453E-59  0.4884312E-05       0.00000
```

Now the error decreases *much* more rapidly with mesh size, as should be expected for a 2nd order approximation: once the solution begins to be reasonable, decreasing the mesh size by a factor of ten reduces the error by a factor of 100. However, the improvement has been achieved only by using information that might not be available in an actual application ($df/dx, df/dy$). These can be evaluated numerically, although doing so might cost important time, and the precision might be uncertain.

1 Higher Order Methods

1.1 Multistep Methods

In studying numerical approximations to derivatives, it was found that better approximations (higher order in the mesh size h) could be obtained by using information from not only mesh point $n \pm 1$ but also $n \pm 2$, or even further points. Similarly, in integrating differential equations, information from two or more steps back can be built into the algorithm. The derivation of such expressions often relies on the exact expression involving an integral over one step:

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f[x, y(x)]dx. \tag{16}$$

The problem is that $f(x, y)$ is not known over all of the integral (or at least the integral would not be done numerically). But we know y_n and y_{n-1} from the previous steps, so linear interpolation can be applied. Using the *Lagrange interpolation formula* we have

$$f(x, y) = \frac{x - x_{n-1}}{h} f_n - \frac{x - x_n}{h} f_{n-1} + \mathcal{O}(h^2), \tag{17}$$

where as before $f_n = f(x_n, y_n)$. Doing the (x_n, x_{n+1}) integral using this extrapolation gives

$$y_{n+1} = y_n + h\frac{3f_n - f_{n-1}}{2} + \mathcal{O}(h^3) = y_n + hf_n + \frac{1}{2}h^2\frac{f_n - f_{n-1}}{h} + \mathcal{O}(h^3), \tag{18}$$

where the last expression emphasizes the relation of this form to a Taylor series expansion. Clearly one type of improvement would be to use a better approximation to the 2nd derivative of $y(x)$, and then of course one could try adding in an h^3 term with an approximate 3rd derivative. Note the progression in these approximations. The increment Δy involves: f_n (Euler's), $(df/dx)_n$ (Taylor's), and then a numerical approximation to df/dx (multistep).

1.2 Implicit Methods

There are approaches yclept "implicit" methods, which are different that the explicit ones covered before because the approximation for y_{n+1} involves y_{n+1} itself, requiring

that an equation be solved. One approach is to consider the derivative

$$(\frac{dy}{dx})_{x_n+\frac{1}{2}} = f(x_{n+\frac{1}{2}}, y_{n+\frac{1}{2}}). \tag{19}$$

Using the symmetric approximation (with respect to $x_{n+1/2}$) $(f_{n+1} - f_n)/h$ for the derivative and half the sum for the value at the midpoint, one obtains

$$\frac{y_{n+1} - y_n}{h} + \mathcal{O}(h^2) = \frac{f_{n+1} + f_n}{2}, \tag{20}$$

giving the expression to be evaluated recurrently as

$$y_{n+1} = y_n + \frac{1}{2}h[f(x_n, y_n) + f(x_{n+1}, y_{n+1}) + \mathcal{O}(h^3). \tag{21}$$

Evidently this equation is implicit for y_{n+1}. This equation is amenable to numerical solution, say by Newton-Raphson iteration, but we will not get into those details. For a wide class of problems the right-hand-side of the ODE is linear in y:

$$f(x, y) = g(x)y. \tag{22}$$

The implicit equation can be solved analytically now, with result

$$y_{n+1} = \frac{1 + \frac{1}{2}hg(x_n)}{1 - \frac{1}{2}hg(x_{n+1})}y_n. \tag{23}$$

The denominator can be expanded to some finite order in h if desired. Numerical test show that this expression gives $\mathcal{O}(h^2)$ errors.

1.2.1 A particular implicit method

The methods presented earlier can be combined. In conjunction with the multistep approach Eq.16, one can use the symmetric Lagrange interpolation formula

$$\begin{aligned} f = \frac{(x - x_n)(x - x_{n-1})}{h^2}f_{n+1} - \frac{(x - x_{n+1})(x - x_{n-1})}{h^2}f_n \\ + \frac{(x - x_{n+1})(x - x_n)}{h^2}f_{n-1} + \mathcal{O}(h^3) \end{aligned} \tag{24}$$

to perform interpolation over the necessary interval. This expression is implicit; if the r.h.s. is linear in y, then it can be solved for y_{n+1} and used recursively as before.