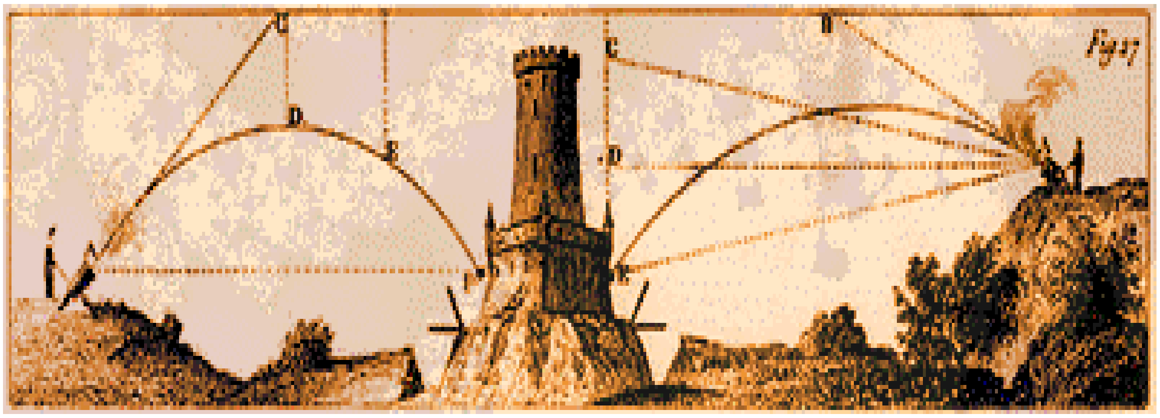
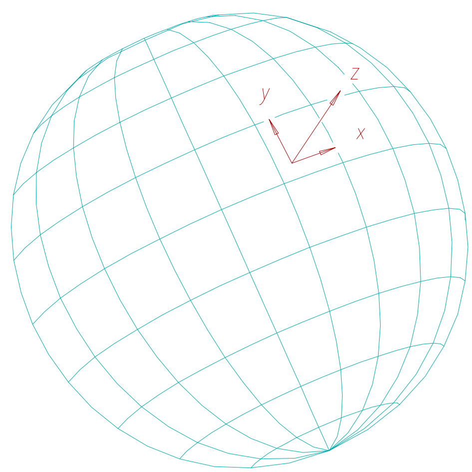
Ordinary differential equations

To motivate the solution of ordinary differential equations, consider the problem of aiming an early siege gun firing a round shot, as shown below:



The cannoneer was forced to estimate the effects of wind, Coriolis force and air resistance. However, we—the heirs of Newton—can *calculate* the angle of inclination and the aiming point, needed to hit a target with any precision. Of course too much precision is useless if we cannot control the amount and quality of the gunpowder charge, or aim with equivalent precision, so we will also be interested in the sensitivity of the answer to the initial conditions of the problem.

We set up local coordinates at a point on the Earth's surface as shown to the right. Assuming one is in the northern hemisphere, the x-direction is due East; the y-direction is due North; and the z-axis points straight out from the Earth's center.



The equations of motion are Newton's Second Law, but we wish to include air resistance and Coriolis force. The latter is a pseudoforce that arises from the Earth's rotation—that is, the cannon is actually being fired in an accelerated frame; the effective equation of motion for the projectile is therefore[1]

1. See, *e.g.*, J.V. Noble, *Lectures on Theoretical Mechanics*.

$$\frac{d\vec{v}}{dt} = -g\,\hat{z} - \Gamma\, v\, \vec{v} + \vec{\Omega} \times \left(\vec{r} \times \vec{\Omega}\right) + 2\,\vec{v} \times \vec{\Omega}\,.$$

We have modeled the air friction as proportional to the square of the velocity, since we expect viscosity to be unimportant. The coefficient Γ is

$$\Gamma = \frac{1}{2m}\,\chi\, A\, \rho(z)$$

where ρ is the air density at altitude z, A is the cross-sectional area of the projectile, χ is the "shape factor" of the projectile (about 0.3 for a sphere) and m is the projectile mass. The Earth's angular velocity, expressed in local cordinates, is

$$\vec{\Omega} = \Omega\left(\hat{y}\cos\theta + \hat{z}\sin\theta\right),$$

where θ is the conventional latitude (as measured from the Equator). As usual, g is the acceleration of gravity at the Earth's surface, about 9.8 m/sec^2 and $v = |\vec{v}|$.

For terrestrial distances and the muzzle velocities appropriate to artillery, we may neglect the term quadratic in the Earth's angular velocity Ω. (Check this yourself, for muzzle speeds comparable with that of sound, and ranges up to 10 km.) Thus the system of equations we must solve is

$$\frac{dv_x}{dt} = -\Gamma v\, v_x + 2\Omega\left(v_y \sin\theta - v_z\cos\theta\right)$$

$$\frac{dv_y}{dt} = -\Gamma v\, v_y - 2\Omega\, v_x \sin\theta$$

$$\frac{dv_z}{dt} = -g - \Gamma v\, v_z + 2\Omega\, v_x\cos\theta\,.$$

This is a coupled system of nonlinear first-order ordinary differential equations. They are *nonlinear* because the velocities appear quadratically; they are *first-order* because only the first derivative of the velocity appears in each equation.

To solve the aiming problem, we shall need to calculate the projectile's trajectory, possibly many times, changing the initial direction and elevation as necessary to discover those precise values that will cause the projectile to intersect the target coordinates within the desired accuracy. This approach is, not surprisingly, called the "shooting" method. To find the trajectory we shall solve for the (vector) velocity, then integrate this to find the coordinates as a function of—say—time.

To solve for the desired direction and elevation we might minimize the distance from the coordinates of the point of aim to those of the end of the trajectory:

$$\Delta(\psi, \chi) = \left|\vec{r}_{aim} - \vec{r}_{impact}(\psi, \chi)\right|\,.$$

This procedure has an advantage over trying to find the root: if the range of the gun is not sufficient to reach the target, the minimizer will reveal this.

Before we can begin to solve our paradigmatic problem, however, we need to be able to solve first-order ordinary differential equations. This is the subject of the next section.

1. First-order equations

We wish to solve the first-order general differential equation

$$\dot{x} \equiv \frac{dx}{dt} = f(x, t) \tag{1}$$

In general we can only solve Eq. 1 approximately, starting from the value of x—call it x_0—at some initial time t_0 , then advancing the time by small increments $dt = h$, using the differential equation itself to give us $x(t + h)$ given $x(t)$.

For example, we could expand in Taylor's series[2]

$$x(t+h) = x(t) + h\dot{x}(t) + \frac{h^2}{2}\ddot{x}(t) + \ldots \tag{2}$$

and keep only the lowest order terms:

$$x(t+h) \approx x(t) + hf(x(t), t) \ . \tag{3}$$

2. Simple-minded methods

We shall define

$$x_n \overset{df}{=} x(t) \ ;$$

in terms of x_n the approximation Eq. 3 becomes

$$x_{n+1} = x_n + h\, f(x_n , t) \ . \tag{4}$$

We now ask whether this approach can yield an incorrect result. The answer is "Certainly!"— Eq. (4) can become unstable if h is too large. Consider the equation

$$\dot{x} = -Ax$$

where A is a constant. We know the solution of this equation is $x(t) = x(0)\, e^{-At}$. But if we turn it into a difference equation,

$$x_{n+1} = x_n \left(1 - Ah\right),$$

the solution of the latter is

$$x_n = x_0 \left(1 - Ah\right)^n .$$

2. See, *e.g.*, ***HMF*** §3.6.1.

In other words, the solution becomes oscillatory (and incorrect!) if $Ah > 1$.

The obvious cure is to keep h small—that is, to integrate with many time steps. Unfortunately, the computer being used for the integration may be too slow to permit an adequately fine subdivision of the interval, so that the numerical solution will not adequately approximate the actual solution. Speed of evaluation can be crucial in the folllowing circumstances:

- the problem may require solving many such equations;
- the problem must be solved in real time, as in weather prediction, process control, on-line data analysis or missile interception;
- the function $f(x, t)$ may be expensive to evaluate.

If the practitioner's aim is to obtain results in a finite time, the simple-minded algorithm must be replaced by one that permits coarser time steps without compromising precision.

3. Quadrature formulas

Suppose we wished to base a solution on a numerical quadrature formula—say Simpson's rule:

$$x(t+2h) = x(t) + \int_t^{t+2h} dt'\, f\big(x(t'), t'\big)$$

$$\approx \frac{h}{3}\Big[f\big(x(t), t\big) + 4 f\big(x(t+h), t+h\big) + f\big(x(t+2h), t+2h\big)\Big];$$

then we would obtain the (implicit) formula

$$x_{n+1} \approx x_{n-1} + \frac{h}{3}\Big[f_{n-1} + 4 f_n + f_{n+1}\Big].$$

Suppose we try this on

$$\dot{x} = -Ax :$$

we have

$$x_{n+1}\left(1 + \frac{hA}{3}\right) = x_{n-1}\left(1 - \frac{hA}{3}\right) - \frac{4hA}{3} x_n .$$

This is a linear difference equation with constant coefficients, whose solution may be sought in the form

$$x_n = \alpha\, \beta^n ;$$

we find a quadratic equation for β, hence there are two roots:

$$\beta = \left(1 + \frac{Ah}{3}\right)^{-1}\left(-\frac{2Ah}{3} \pm \sqrt{1 + \frac{A^2h^2}{3}}\right) \approx \begin{cases} 1 - Ah \\ -1 - Ah/3 \end{cases}$$

The first of these, for small enough step-size h, is smaller than unity and therefore leads to the solution we seek, namely a decreasing exponential. The second root exceeds unity in magnitude and leads to a growing, oscillatory term. This is rather unfortunate, since the limitations of numerical precision on digital computers with finite registers guarantees the presence of some small amount of the growing solution. That is, after a finite number of integration steps the solution is guaranteed to contain an error that continues to grow.

Thus we shall have to take care that methods we introduce for numerical integration of differential equations do not introduce instabilities through inadvertance.

4. Calculus of finite differences

In discussing numerical integration it is useful to introduce the difference operator Δ defined as

$$\Delta x_n \stackrel{df}{=} x_{n+1} - x_n .$$

Now clearly,

$$\Delta f(t) \stackrel{df}{=} f(t+h) - f(t) \equiv h\frac{df}{dt} + \frac{h^2}{2!}\frac{d^2f}{dt^2} + \ldots$$

where we have expanded $f(t+h)$ in Taylor's series about t. Formally, if we define an operator

$$D = \frac{d}{dt}$$

we may write the Taylor's series as an exponential:

$$\Delta f = h\frac{df}{dt} + \frac{h^2}{2!}\frac{d^2f}{dt^2} + \ldots = \left(e^{hD} - 1\right)f$$

or abstracting to a relation between operators,

$$\Delta = e^{hD} - 1 \qquad (5)$$

or

$$D = \frac{1}{h}\log(1+\Delta) . \qquad (6)$$

5. Runge-Kutta method

One standard class of methods that had fallen into disfavor but now are popular again, are the Runge-Kutta[3] algorithms. The algorithms can be classified according to order n—that is, if h is the step size, the error at each step will be $O(h^{n+1})$.

Second-order Runge-Kutta

From Eq. 6 we find

$$D\,x = \frac{1}{h} log(1+\Delta)\,x = f(x, t)$$

or

$$\left(\Delta - \frac{\Delta^2}{2} + \ldots\right)x = h\,f\,.$$

Keeping only the second difference term $\Delta^2/2$ we may write this as

$$\Delta x \approx h\,f + \frac{1}{2}\Delta^2\,x \approx h\,f + \frac{1}{2}\Delta\,(h\,f) = \frac{1}{2}\Big[h\,f(x, t) + h\,f(x+hf, t+h)\Big]\,.$$

This is, in fact, the second order Runge-Kutta quadrature formula[4]

$$\begin{aligned} k &= h\,f(x_n\,, nh) \\ x_{n+1} &= x_n + \tfrac{1}{2}\Big[k + h\,f(x_n+k, nh+h)\Big] + O(h^3)\,. \end{aligned} \qquad (7)$$

We can check this formula by expanding in Taylor's series and comparing terms. Clearly,

$$k + h\,f(x+k, t+h) \approx 2h\,f + h\,k\frac{\partial f}{\partial x} + h^2\frac{\partial f}{\partial t} = 2h\,\dot{x} + h^2\,\ddot{x} + O(h^3)\,,$$

hence

$$x_{n+1} \overset{df}{=} x(nh+h) \approx x_n + h\,\dot{x}_n + \frac{h^2}{2}\ddot{x}_n\ ;$$

that is, the Runge-Kutta x_{n+1} agrees with the Taylor's series expansion to $O(h^3)$.

It is worth noting that a formula[5] equivalent to Eq. 7 is

$$x_{n+1} = x_n + h\,f(x_n + k/2, t + h/2)\,;$$

that is, as we shall see below, there is a certain latitude in choosing the combinations of derivatives and function values, as well as in the intermediate values of the independent variable.

---

3. ***HMF***, p896.
4. *HMF*, §25.5.6
5. *HMF*, §25.5.7

```
FALSE [IF]
   Numerical solution of first-order ordinary differential equation by 2nd order
   Runge-Kutta algorithm, following Abramowitz & Stegun p. 896, 25.5.6

   Solves    dx/dt = f(x,t)

   This is an ANS Forth program requiring FLOATING and FLOATING EXT
   wordsets. This program assumes a separate floating point stack.

   For simplicity and clarity, the program uses a FORmula TRANslator


Example
   : fnb   ( f: x t -- t^2*exp[-x])    f^2  FSWAP  FNEGATE FEXP   F*  ;
   use( fnb  0e0 0e0 5e0 0.1e0 )runge
[THEN]

MARKER  -runge
INCLUDE ftran111.f         \ use FORmula TRANslator

\ multiple variable definitions
: fvariables     ( n —)  0 DO  FVARIABLE  LOOP  ;

5 fvariables t h x tf  xk

v: fdummy        \ dummy function name

: x’    \ integration step
        f" xk = h*fdummy(x,t)"          \ compute k
        f" t = t+h"                     \ increment t
        f" xk + h*fdummy(x+xk,t)"  F2/  ( f: — dx)
        x F@   F+   x F!                \ x = x + dx ;

: display       CR   t F@  FS.  5 SPACES   x F@  FS.  ;

: done?  t F@  tf F@  F>  ;

: )runge   ( xt —)  ( f: x0 t0 tf h — )
        defines fdummy              \ vector fn_name
        tf F!  t F!  x F!  h F!     \ initialize variables
        BEGIN   display   x’        \ indefinite loop
        done?   UNTIL  ;
```

A program for 2nd-order Runge-Kutta integration (in Forth) is shown on the preceding page. As an example we solve numerically the equation:

$$\dot{x} = t^2\, e^{-x}$$

with the initial condition $x(t=0) = 0$, whose exact solution is

$$x(t) = \log_e(1 + \frac{1}{3}t^3)\ .$$

To run and compare with the exact solution, modify the word `display` as follows:

```
: display      CR   t F@  FS.  5 SPACES   x F@  FS.
                5 SPACES  f" ln(1+t^3/3)"  FS.   ;
```

The resulting output is shown below:

t	x	exact	t	x	exact
0.00000E-1	0.00000E-1	0.00000E-1	2.60000E0	1.92543E0	1.92551E0
1.00000E-1	5.00000E-4	3.33278E-4	2.70000E0	2.02287E0	2.02300E0
2.00000E-1	2.99675E-3	2.66312E-3	2.80000E0	2.11817E0	2.11834E0
3.00000E-1	9.45945E-3	8.95974E-3	2.90000E0	2.21133E0	2.21153E0
4.00000E-1	2.17714E-2	2.11090E-2	3.00000E0	2.30236E0	2.30259E0
5.00000E-1	4.16399E-2	4.08220E-2	3.10000E0	2.39129E0	2.39154E0
6.00000E-1	7.04869E-2	6.95261E-2	3.20000E0	2.47817E0	2.47844E0
7.00000E-1	1.09341E-1	1.08256E-1	3.30000E0	2.56305E0	2.56333E0
8.00000E-1	1.58756E-1	1.57573E-1	3.40000E0	2.64597E0	2.64627E0
9.00000E-1	2.18777E-1	2.17528E-1	3.50000E0	2.72700E0	2.72731E0
1.00000E0	2.88963E-1	2.87682E-1	3.60000E0	2.80619E0	2.80651E0
1.10000E0	3.68461E-1	3.67186E-1	3.70000E0	2.88360E0	2.88393E0
1.20000E0	4.56125E-1	4.54890E-1	3.80000E0	2.95930E0	2.95962E0
1.30000E0	5.50634E-1	5.49469E-1	3.90000E0	3.03333E0	3.03365E0
1.40000E0	6.50614E-1	6.49544E-1	4.00000E0	3.10575E0	3.10608E0
1.50000E0	7.54732E-1	7.53772E-1	4.10000E0	3.17663E0	3.17696E0
1.60000E0	8.61759E-1	8.60919E-1	4.20000E0	3.24601E0	3.24634E0
1.70000E0	9.70612E-1	9.69895E-1	4.30000E0	3.31395E0	3.31427E0
1.80000E0	1.08036E0	1.07977E0	4.40000E0	3.38049E0	3.38081E0
1.90000E0	1.19025E0	1.18977E0	4.50000E0	3.44569E0	3.44601E0
2.00000E0	1.29965E0	1.29928E0	4.60000E0	3.50960E0	3.50991E0
2.10000E0	1.40808E0	1.40781E0	4.70000E0	3.57225E0	3.57256E0
2.20000E0	1.51516E0	1.51498E0	4.80000E0	3.63369E0	3.63400E0
2.30000E0	1.62061E0	1.62051E0	4.90000E0	3.69397E0	3.69427E0
2.40000E0	1.72423E0	1.72419E0	5.00000E0	3.75312E0	3.75342E0
2.50000E0	1.82586E0	1.82589E0	ok		

Fourth-order Runge-Kutta

The key to deriving Runge-Kutta formulas is to write the difference operator as a linear combination of the first derivative values of the function:

$$x_{n+1} - x_n \approx \sum_{s=0}^{M} w_s \, k_s \tag{8}$$

where

$$k_s = h \, f\left(x_n + \sum_{r=1}^{s-1} \beta_{sr} \, k_r \, , \, t + \alpha_s \, h \right)$$

and the coefficients w_s , α_s and β_{sr} can be determined by expanding both sides of Eq. 8 and comparing term-by-term. This leads, in general, to fewer equations than unknowns, hence the available degrees of freedom may be chosen either to minimize some bound on the error term, or with some other purpose.

The fourth-order Runge-Kutta formula we shall use here, accurate to $O(h^5)$, is

$$x_{n+1} = x_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} \tag{9}$$

where

$$k_1 = h \, f(x_n \, , nh) \qquad k_3 = h \, f\left(x_n + \frac{k_2}{2}, nh + \frac{h}{2} \right)$$

$$k_2 = h \, f\left(x_n + \frac{k_1}{2}, nh + \frac{h}{2} \right) \qquad k_4 = h \, f(x_n + k_3 \, , nh + h) \, .$$

We leave it as an exercise to show that the preceding formulae are equivalent to a Taylor's series expansion to the requisite order; as well as to solve the same example equation over the same range, comparing it with the exact solution. To convert the program for second-order Runge-Kutta to one using the fourth-order formula Eq. 6, we need alter but one subroutine (of course we must also allocate space for the extra intermediate variables):

```
: x'    \ integration step    ( note: h2 = h/2)
   f" xk1 = h2*fdummy(x,t)"            \ compute k1
   f" t = t+h2"                        \ increment t by a half-step
   f" xk2 = h*fdummy(x+xk1,t)"
   f" xk3 = h*fdummy(x+xk2/2,t)"
   f" t = t+h2"                        \ increment t by a half-step
   f" xk4 = h2*fdummy(x+xk3, t)"
   f" x = x + (xk1 + xk2 + xk3 + xk4)/3"
;
```

The same example, run with this algorithm, gives the results shown on the next page. We see that the numerical integration now agrees with the exact result to six (6) significant figures, over the entire range, in contrast to the second-order formula, which—with this step size—yielded significant

```
   t            x           exact

0.00000E-1   0.00000E-1   0.00000E-1
1.00000E-1   3.33281E-4   3.33278E-4
2.00000E-1   2.66312E-3   2.66312E-3
3.00000E-1   8.95975E-3   8.95974E-3
4.00000E-1   2.11090E-2   2.11090E-2
5.00000E-1   4.08220E-2   4.08220E-2
6.00000E-1   6.95261E-2   6.95261E-2
7.00000E-1   1.08256E-1   1.08256E-1
8.00000E-1   1.57574E-1   1.57573E-1
9.00000E-1   2.17528E-1   2.17528E-1
1.00000E0    2.87682E-1   2.87682E-1
1.10000E0    3.67187E-1   3.67186E-1
1.20000E0    4.54890E-1   4.54890E-1
1.30000E0    5.49470E-1   5.49469E-1
1.40000E0    6.49544E-1   6.49544E-1
1.50000E0    7.53772E-1   7.53772E-1
1.60000E0    8.60919E-1   8.60919E-1
1.70000E0    9.69895E-1   9.69895E-1
1.80000E0    1.07977E0    1.07977E0
1.90000E0    1.18977E0    1.18977E0
2.00000E0    1.29928E0    1.29928E0
2.10000E0    1.40781E0    1.40781E0
2.20000E0    1.51498E0    1.51498E0
2.30000E0    1.62051E0    1.62051E0
2.40000E0    1.72419E0    1.72419E0
2.50000E0    1.82589E0    1.82589E0
2.60000E0    1.92551E0    1.92551E0
2.70000E0    2.02300E0    2.02300E0
2.80000E0    2.11834E0    2.11834E0
2.90000E0    2.21153E0    2.21153E0
3.00000E0    2.30259E0    2.30259E0
3.10000E0    2.39154E0    2.39154E0
3.20000E0    2.47844E0    2.47844E0
3.30000E0    2.56333E0    2.56333E0
3.40000E0    2.64627E0    2.64627E0
3.50000E0    2.72731E0    2.72731E0
3.60000E0    2.80651E0    2.80651E0
3.70000E0    2.88393E0    2.88393E0
3.80000E0    2.95962E0    2.95962E0
3.90000E0    3.03365E0    3.03365E0
4.00000E0    3.10608E0    3.10608E0
4.10000E0    3.17696E0    3.17696E0
4.20000E0    3.24634E0    3.24634E0
4.30000E0    3.31427E0    3.31427E0
4.40000E0    3.38081E0    3.38081E0
4.50000E0    3.44601E0    3.44601E0
4.60000E0    3.50991E0    3.50991E0
4.70000E0    3.57256E0    3.57256E0
4.80000E0    3.63400E0    3.63400E0
4.90000E0    3.69427E0    3.69427E0
5.00000E0    3.75342E0    3.75342E0
  ok
```

disagreement in the fourth or fifth significant figures. In fact, setting the floating point display to ten significant figures reveals that the calculated and exact results of the fourth-order Runge-Kutta formula disagree by a few parts in 10^7.

Implicit Runge-Kutta integration

A variation on the explicit Runge-Kutta algorithm discussed above is the *implicit* algorithm[6]. For example, in the second-order formula given above, suppose $x_n + k_n$ were replaced by x_{n+1} :

$$k = h\,f(x_n\,,\,nh)$$
$$x_{n+1} = x_n + \frac{1}{2}\Big[k + h\,f(x_{n+1}\,,\,nh+h)\Big] + O(h^3)\;.$$

and the resulting (transcendental) equation solved for x_{n+1} by—say—*regula falsi*. Since we have already written a *regula falsi* program, we can apply it here to get the algorithm shown diagrammatically below:

```
)runge
  initialize          \ get h, tf, t0, x0

  BEGIN

       xk = hf(x,t)   \ integration step

       SOLVE:  x' = x + [xk + hf(x',t)]/2

       display

       done?

  UNTIL
```

The task of implementing this algorithm is left as exercise.

Now why would one bother with an implict Runge-Kutta scheme? The answer is that when the differential equation becomes singular; or when the equation is "stiff", the implicit scheme will not develop instabilities. That is, it permits us to obtain a solution without making the step size excessively small.

6. See, *e.g.*, A. Ralston, *A First Course in Numerical Analysis* (McGraw-Hill Book Company, New York, 1965) Ch. 5. Implicit methods increase the stability of numerical solution, compared with explicit methods. The formula below is exact for second-order polynomials. The error is of the same order as the explicit formula, but the coefficient may be smaller.

6. Predictor-corrector methods

The fourth-order Runge-Kutta method requires four evaluations of the function $f(x, t)$ per integration step. This can be unnecessarily time-consuming. A second disadvantage of Runge-Kutta methods is that they provide no estimate of how well they are doing; that is, if the precision is too great (because the function is changing only slowly), it would be advantageous to increase the step-size. Conversely, if the precision is worsening, a decrease in step-size is in order. The Runge-Kutta method is both stable and easy to program, however, since it integrates forward from the current value x_n . This is why it remains popular.

Another genre of methods, the so-called *predictor-corrector* algorithms, has been studied by many authors. Its advantages are first, that (in a fourth-order procedure) $f(x, t)$ is evaluated only twice per integration step; and second, that two different algorithms (the predictor and the corrector) are employed, hence they check each other. That is, comparing two different algorithms provides a running estimate of precision, that can be used to adjust the step-size for optimal results. The chief drawback of predictor-corrector methods is that they are more difficult to program than Runge-Kutta.

Like all numerical integration algorithms, the predictor-corrector approach is based on the integral form of the differential equation,

$$x(t + \Delta t) \;=\; x(t) \;+\; \int_t^{t+\Delta t} ds\, \dot{x}(s) \;\; . \tag{10}$$

The trick is to use one quadrature formula for the predictor and another for the corrector. The two formulas must, of course, have error terms of the same order. Thus consider a second-order procedure based on closed and open Newton-Cotes quadrature rules:

$$x_{n+1} \;=\; x_{n-2} \;+\; \frac{3h}{2}\left[\dot{x}_{n-1} + \dot{x}_n\right] \;+\; \mathrm{O}(h^3) \qquad \text{(open Newton-Cotes[7])} \tag{11a}$$

$$x_{n+1} \;=\; x_n \;+\; \frac{h}{2}\left[\dot{x}_n \;+\; \dot{x}_{n+1}\right] \;+\; \mathrm{O}(h^3) \qquad \text{(trapezoidal rule[8])} \tag{11b}$$

The first thing we ought to examine is the stability of these procedures. Consider again our decaying exponential,

$$\dot{x} \;=\; -Ax \;;$$

the open Newton-Cotes formula leads to the difference equation

$$x_{n+1} \;=\; x_{n-2} \;-\; \frac{3hA}{2}\left[x_{n-1} + x_n\right]$$

7. *HMF*, p. 886, §25.4.21
8. *HMF*, p. 885, §25.4.2

whose solution is

$$x_n = \alpha \beta^n ,$$

where

$$\beta^3 + \frac{3Ah}{2}\beta^2 + \frac{3Ah}{2}\beta - 1 = 0 .$$

For $|Ah| << 1$ we find one real root,

$$\beta_1 \approx 1 - Ah ,$$

and two complex conjugate roots

$$\beta_{2,3} \approx \left(1 + \frac{Ah}{2}\right) e^{\pm 2i\pi/3} .$$

That is, the formula is unstable because in addition to the decreasing solution (the one that leads to $x = x(0)\, e^{-At}$), there are two that increase more slowly, as well as oscillate. (On the other hand, if A had the opposite sign, corresponding to an *increasing* exponential, the two complex roots would be smaller than unity in magnitude so the errors would die out.)

What about the closed Newton-Cotes formula? Here the secular equation is

$$\left(1 + \frac{Ah}{2}\right)\beta = 1 - \frac{Ah}{2}$$

with one root, $\beta \approx 1 - Ah$. This is a well-behaved equation.

With predictor-corrector algorithms the best strategy is to choose the predictor to be the equation with possible instability, and the corrector to be stable. Thus we choose the open Newton-Cotes formula for the predictor and the trapezoidal rule for the corrector. We leave the task of implementing the predictor-corrector algorithm to the student.

7. Second order equations

Second order equations like

$$\ddot{x} + f(x, \dot{x}, t) = 0$$

can be put in the form of two first-order equations and then solved by applying a first-order differential equation solver to each. Let

$$y \overset{df}{=} \dot{x} \;;$$

then we have the system

$$\dot{y} + f(x, y, t) = 0$$

$$\dot{x} - y = 0 .$$

Many of the second order differential equations encountered in practice are linear but have non-constant coefficients. Such equations can be transformed so as to have no first-derivative term, that is, to have the form

$$\ddot{x} + F(x, t) = 0 .$$

And of course, some nonlinear equations already have this form.

there are special methods for second-order equations (linear or nonlinear) that lack a first-derivative term. The best-known of these is the Numerov algorithm[9]:

$$\Delta^2 x_{n-1} \equiv x_{n+1} - 2x_n + x_{n-1} = h^2 \ddot{x} + \frac{h^4}{12} x^{(4)} + \ldots$$

$$h^2 \Delta^2 \ddot{x}_{n-1} \equiv \ddot{x}_{n+1} - 2\ddot{x}_n + \ddot{x}_{n-1} = -h^2 \left(F_{n+1} - 2F_n + F_{n-1}\right) \approx -h^4 x^{(4)}$$

or in other words,

$$x_{n+1} - 2x_n + x_{n-1} \approx -h^2 F_n - \frac{h^2}{12}\left(F_{n+1} - 2F_n + F_{n-1}\right) . \tag{12}$$

If the differential equation is linear and of second order,

$$\ddot{x} = f(t)\, x + g(t) , \tag{12$'$}$$

then Eq. (12′) can be solved explicitly for the next value, x_{n+1} :

$$\Delta^2 \left[x_{n-1} \left(1 - \frac{h^2}{12} f_{n-1} \right) \right] \approx h^2 f_n x_n + h^2 g_n + \frac{h^2}{12} \Delta^2 g_{n-1} + O(h^6) . \tag{13}$$

The error is $O(h^6)$; the price is that one needs the first two values of $x(t)$, x_0 and x_1, in order to procede with this algorithm. The fact that one also needs g_{n+1} at every step imposes no extra computational burden since it merely gets computed one step earlier than it would have been.

On the following page is the numerical solution of the equation

$$\ddot{x} + x = 0$$

with initial conditions

$$x_0 = 0, \dot{x}_0 = 1 \text{ and } h = 0.1$$

The exact solution is, of course, $\sin(t)$. The Numerov algorithm definitely yields the advertised accuracy (*i.e.* to 1 part in 10^6).

---

9. B.V. Numerov, *Mon. Not. Roy. Astron. Soc.* **84** (1924) 180; *ibid.*, 592.
See also R.W. Hamming, *Numerical Methods for Scientists and Engineers* (McGraw-Hill Book Co., Inc., New York, 1962) p. 215.

```
   t              x(t)           sin(t)
1.00000E-1      9.98334E-2      9.98334E-2
2.00000E-1      1.98669E-1      1.98669E-1
3.00000E-1      2.95520E-1      2.95520E-1
4.00000E-1      3.89418E-1      3.89418E-1
5.00000E-1      4.79426E-1      4.79426E-1
6.00000E-1      5.64642E-1      5.64642E-1
7.00000E-1      6.44218E-1      6.44218E-1
8.00000E-1      7.17356E-1      7.17356E-1
9.00000E-1      7.83327E-1      7.83327E-1
1.00000E0       8.41471E-1      8.41471E-1
1.10000E0       8.91207E-1      8.91207E-1
1.20000E0       9.32039E-1      9.32039E-1
1.30000E0       9.63558E-1      9.63558E-1
1.40000E0       9.85450E-1      9.85450E-1
1.50000E0       9.97495E-1      9.97495E-1
1.60000E0       9.99573E-1      9.99574E-1
1.70000E0       9.91665E-1      9.91665E-1
1.80000E0       9.73847E-1      9.73848E-1
1.90000E0       9.46300E-1      9.46300E-1
2.00000E0       9.09297E-1      9.09297E-1
2.10000E0       8.63209E-1      8.63209E-1
2.20000E0       8.08496E-1      8.08496E-1
2.30000E0       7.45705E-1      7.45705E-1
2.40000E0       6.75463E-1      6.75463E-1
2.50000E0       5.98472E-1      5.98472E-1
2.60000E0       5.15501E-1      5.15501E-1
2.70000E0       4.27379E-1      4.27380E-1
2.80000E0       3.34988E-1      3.34988E-1
2.90000E0       2.39249E-1      2.39249E-1
3.00000E0       1.41119E-1      1.41120E-1
3.10000E0       4.15800E-2      4.15807E-2
3.20000E0      -5.83748E-2     -5.83741E-2
3.30000E0      -1.57746E-1     -1.57746E-1
3.40000E0      -2.55542E-1     -2.55541E-1
3.50000E0      -3.50784E-1     -3.50783E-1
3.60000E0      -4.42521E-1     -4.42520E-1
3.70000E0      -5.29837E-1     -5.29836E-1
3.80000E0      -6.11858E-1     -6.11858E-1
3.90000E0      -6.87767E-1     -6.87766E-1
4.00000E0      -7.56803E-1     -7.56802E-1
4.10000E0      -8.18277E-1     -8.18277E-1
4.20000E0      -8.71576E-1     -8.71576E-1
4.30000E0      -9.16166E-1     -9.16166E-1
4.40000E0      -9.51602E-1     -9.51602E-1
4.50000E0      -9.77530E-1     -9.77530E-1
4.60000E0      -9.93691E-1     -9.93691E-1
4.70000E0      -9.99923E-1     -9.99923E-1
4.80000E0      -9.96164E-1     -9.96165E-1
4.90000E0      -9.82452E-1     -9.82453E-1
5.00000E0      -9.58924E-1     -9.58924E-1
5.10000E0      -9.25814E-1     -9.25815E-1  ok
```

Next we investigate the stability of the Numerov algorithm. Suppose the inhomogeneous term vanished and the function $f(t)$ were constant: then letting

$$a^2 \overset{df}{=} \frac{h^2 f}{1 - h^2 f/12}$$

and

$$X_n \overset{df}{=} x_n \left(1 - \frac{h^2}{12} f_n\right)$$

we find that the difference equation has the solution $X_n = \alpha \beta^n$ where the two roots of the secular equation are

$$\beta = 1 + a^2/2 \pm a\sqrt{1 + a^2/4} \ .$$

If $a^2 < 0$ as in the preceding example, then $\beta = e^{i\varphi}$, *i.e.* its magnitude is unity. and the solution is asymptotically sinusoidal. If, on the other hand, $a^2 > 0$, the solutions are decreasing and increasing exponentials, approximating

$$x(t) \sim e^{\pm t\sqrt{f}}$$

in the limit as $h \to 0$.

Thus we may expect that if we must integrate in the positive-t direction, and we are hoping for a solution with decreasing exponential behavior, sooner or later any initially small component of the increasing solution will win, at which point the accuracy will decrease radically.

8. "Stiff" equations

Certain differential equations can be extremely difficult to solve because they embody several quite different time scales. An example is the equation

$$\ddot{x} - 400(1 - e^{-t})^2 x = 0 \, , \qquad (14)$$

in which the two time scales are 1 and $1/\sqrt{400} = 0.05$. Such equations were first encountered by engineers designing systems containing several both soft and stiff springs and the name "stiff" has stuck.

Suppose we use any of the algorithms we have developed to integrate Eq. 14 to the right—that is, in the direction of increasing t. For $t > 1$ we see the equation has approached the behavior

$$\ddot{x} - 400\, x \approx 0$$

whose solutions are $e^{\pm 20\, t}$. Our numerical solution will rapidly approach the exponentially growing solution, no matter what initial conditions we impose. That is, no matter how fine a step size we choose, if we desire the solution that asymptotically falls as $e^{-20\,t}$, we are bound to be disappointed since it will be swamped by the growing solution.

There are several ways to handle such difficulties. In the present case, we can start the solution at $e^{-20\,t}$ for some large t and integrate leftward. The desired solution is now the growing one, and the undesired one dies out exponentially quickly. The computed values can then be renormalized (because the equation is homogeneous) to give any desired (generally non-zero) value at $t = 0$.

This method does not always work, hence it is worthwhile to have some additional methods that can be tried when it fails. Acton[10] suggests the Riccati transformation: let

$$\dot{x} = \eta(t)\,x\;;$$

then equations of the form

$$\ddot{x} = f(t)\,x$$

are transformed into two first-order equations,

$$\dot{\eta} + \eta^2 = f(t)$$

and

$$\dot{x} = \eta(t)\,x\;,$$

one of them nonlinear. The second can be solved by a simple quadrature:

$$x(t) = \exp\left[\int^{t} dt'\;\eta(t')\right]$$

whereas the first is relatively smooth.

A second technique for solving stiff equations in which the undesired solution diverges is based on the following idea: if we integrate our example in the direction of increasing t we are guaranteed to obtain the "wrong" solution, $w(t)$. Since we are dealing with a second order linear differential equation, we can obtain the second solution (the "right" one, $r(t)$) by the substitution

$$r(t) = w(t)\,u(t)$$

which (since $w(t)$ is a solution) gives

$$\ddot{u}\,w + 2\dot{u}\,\dot{w} = 0$$

or (within a multiplicative constant)

$$u(t) = \int_t^{\infty} dt'\,\frac{1}{w^2(t')}\quad .$$

Thus we may write

10. See, *e.g.*, F.S. Acton, *Numerical Methods that Work* (Math. Ass'n of America, Washington, DC, 1990), p. 148ff.

$$r(t) = A\,w(t)\int_t^{\infty} dt'\,\frac{1}{w^2(t')} \tag{15}$$

where A is adjusted to give the desired initial value, $r(0)$. This method requires only that we integrate rightward until $w(t)$ attains its asymptotic behavior. Obviously this will occur at some finite value of t. If we save the values $w_n = w(nh)$ so attained, it is straightforward then to construct the desired solution (from right to left) by quadrature.

Not all stiff equations are stiff because of divergent solutions: an example is the system[11]

$$\begin{pmatrix}\dot{x}\\ \dot{y}\end{pmatrix} = \begin{pmatrix}998 & 1998\\ -999 & -1999\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix}. \tag{16}$$

with initial conditions $x(0) = 1$ and $y(0) = 0$. It is not difficult to determine the exact solutions:

$$\begin{aligned} x &= 2\,e^{-t} - e^{-1000\,t} \\ y &= -\,e^{-t} + e^{-1000\,t}\,. \end{aligned}$$

The disease of the above system, from the numerical point of view, is that although the rapidly decreasing solution disappears almost instantly, the step size h required for numerical stability, in the usual integration schemes, will be determined by the short time scale 0.001 rather than the slow scale, 1.000 . Thus, the third approach, advocated by Press, *et al.*[12], is an implicit scheme based on (here A is a matrix and x a vector)

$$\dot{x} = -A\,x$$

so that

$$x_{n+1} \approx x_n + h\,\dot{x}_{n+1} \equiv x_n - h\,A\,x_{n+1}\,.$$

To solve we write

$$x_{n+1} \approx \left(1 + hA\right)^{-1} x_n\,;$$

if the matrix A is positive-definite, then the inverse matrix has eigenvalues that are always smaller than unity, hence the method is stable even for large step size (although it may lose accuracy in that case). Let us apply it to the case

$$A = \begin{pmatrix}-998 & -1998\\ 999 & 1999\end{pmatrix}$$

whose eigenvalues are 1 and 1000, respectively. The inverse $(1 + hA)^{-1}$ is then given by

11. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations* (Prentice-Hall, Englewood Cliffs, NJ, 1971).

12. W.H. Press, B.P. Flannery, S. Teukolsky and W.T. Vetterling, *Numerical Recipes* (Cambridge U. Press, Cambridge, 1986), p. 572ff.

$$\left(1 + hA\right)^{-1} = \begin{pmatrix} \dfrac{1 + 1999h}{1 + 1001h + 1000h^2} & \dfrac{1998h}{1 + 1001h + 1000h^2} \\ \dfrac{-999h}{1 + 1001h + 1000h^2} & \dfrac{1 - 998h}{1 + 1001h + 1000h^2} \end{pmatrix}$$

However, although Press, *et al.* recommend this approach, simple experiments indicate that it loses a great deal of accuracy, even for very small step size. The BASIC program given below demonstrates this clearly.

```
' solution of the "stiff" equations
'   dx/dt = 998 * x  +  1998 * y
'   dy/dt = -999 * x  -  1999 * y
'   by the inverse method (Press, et al., p. 572 ff
DEFDBL C-H, X-Y
DEFINT K CLS h = .0001
DET = 1 + h * (1001! + 1000! * h)
C = (1 + 1999! * h) / DET: D = 1998! * h / DET
E = -999! * h / DET: F = (1 - 998! * h) / DET
x = 1!: y = 0!: k = 0
FOR t = h TO 4 STEP h
   x = C * x + D * y
    y = E * x + F * y
    k = k + 1
    IF k MOD 1000 = 0 THEN
        k = 0
        PRINT t, x, y
    END IF
NEXT
END
```

The problem arises from the fact that the eigenvalues of the matrix A appearing in Eq. 16 are extremely different in size: 1 and 1000, respectively. To put it in somewhat different terms, we are trying to calculate the matrix

$$B = e^{-At} = \lim_{h \to 0} \left(1 + Ah\right)^{-t/h} ;$$

if A is "large" in some sense, then we will need to go to values of h considerably smaller than $\|A\|^{-1}$ before we can expect to achieve the above limit.

This fact suggests a possibly better alternative, whose exploration we leave to the interested reader. The idea is to subtract from the matrix A a term of low rank such that the difference is in some sense "small". Thus write

$$A = |a\rangle\, \alpha\, \langle b| + R$$

where R is much smaller than A, and treat hR as a perturbation. We can approximate the vector $|a\rangle$ by the usual iterative method, and similarly for $\langle b|$. Then the computation of $(1 + hA)^{-1}$ is greatly simplified and the solution closer to pure quadrature.

9. Boundary value problems

A typical problem arising in atomic, nuclear and particle physics is the need to determine the energies of (quantum-mechanical) bound states of particles in potentials. Suppose such a potential is spherically symmetric, and the particle nonrelativistic. Then the ordinary differential equation that must be solved is[13]

$$\frac{d^2\psi}{dr^2} = \left[\kappa^2 + U(r)\right]\psi$$

subject to the boundary conditions $\psi(0) = 0$ and $\lim_{r \to \infty} \psi(r) = 0$. The potential $U(r)$ is assumed to vanish faster than $1/r$ for large r, hence the solution manifestly has asymptotic behavior

$$\psi(r) \underset{r \to \infty}{\longrightarrow} A(\kappa)\, e^{-\kappa r} + B(\kappa)\, e^{\kappa r}.$$

It is then obvious that the condition that ψ represent a bound state is that κ should satisfy the transcendental equation

$$B(\kappa) = 0 .$$

The problem we are faced with, is how can we evaluate $B(\kappa)$ and seek its zeros, since we do not know a closed-form expression for it?

Several algorithms have been developed for solving this conundrum. The so-called "shooting method" requires us to choose a value of κ, assume $B(\kappa) = 0$, then integrate inward from some large distance (that is, in the direction of decreasing r). This will be stable, since the desired (exponentially decreasing) solution will be increasing in magnitude. At the same time we integrate outward from $r=0$ (starting with the appropriate small-r behavior). We then match the two solutions at some convenient point[14]. Naturally they will not match, so depending on which logarithmic derivative is larger, one increases or decreases κ and tries again.

The algorithm we shall expound here is a very useful one described by Krell and Ericson[15]. The basic idea is to guess a value for κ and integrate outward. We know that beyond the range of the potential the solution of the (Numerov) difference equation will be

$$A\left[1 + a^2/2 - a\left(1 + a^2/4\right)^{½}\right]^n + B\left[1 + a^2/2 + a\left(1 + a^2/4\right)^{½}\right]^n$$

where $a = \kappa h$. The strategy is therefore to integrate outward to some distance where the potential has become negligible, multiply the solution by

$$\beta_<^n = \left[1 + a^2/2 - a\left(1 + a^2/4\right)^{½}\right]^n$$

13. The potential $U(r)$ is assumed to be appropriately behaved at $r=0$ and $r=\infty$.
14. In fact, to avoid having to normalize the wave function, we match the logarithmic derivatives.
15. M.Krell and T.E.O. Ericson, *J. Comp. Phys.* **3** (1968) 202.

in order to isolate $B(\kappa)$, and search for a zero. Krell and Ericson's search method involves picking three initial values of κ, squaring the resulting values of B, and fitting a quadratic function to the resulting three values in order to extrapolate to zero[16]. That is, they write

$$\varphi(\kappa) \overset{df}{=} B^2(\kappa) \approx \varphi_1 \frac{(\kappa-\kappa_2)(\kappa-\kappa_3)}{(\kappa_1-\kappa_2)(\kappa_1-\kappa_3)} + \varphi_2 \frac{(\kappa-\kappa_1)(\kappa-\kappa_3)}{(\kappa_2-\kappa_1)(\kappa_2-\kappa_3)} + \varphi_3 \frac{(\kappa-\kappa_2)(\kappa-\kappa_1)}{(\kappa_3-\kappa_2)(\kappa_3-\kappa_1)} = 0$$

and solve the quadratic equation. This gives the next guess; at the next iteration the three values of κ are taken to be the new value and the two old values closest to it. Each iteration only requires one new evaluation of $B(\kappa)$ since we keep the two values of B (corresponding to the two old values of κ that we retained).

10. Accurate numerical orthogonality

One way experimental physicists investigate complex systems such as condensed matter or isolated atoms or atomic nuclei (or even, for that matter, isolated nucleons) is to measure their response to a weak external probe. By weak we mean that if the Hamiltonian operator of system and probe is

$$H = H_0 + K + V$$

where H_0 is the hamiltonian of the isolated system, K that of the isolated probe, and V is their interaction, then the amplitude for a transition from the initial state $|n, k\rangle = |\Phi_n\rangle \otimes |\chi_k\rangle$ to the final state $|n', k'\rangle = |\Phi_{n'}\rangle \otimes |\chi_{k'}\rangle$ is something like

$$M_{f,i} \overset{df}{=} \langle n', k' | V | n, k\rangle = \int d^3r\, \Phi^\dagger_{n'}(\vec r)\; V_{k'k}(\vec r)\; \Phi_n(\vec r),$$

where, typically,

$$V_{k'k}(\vec r) \overset{df}{=} \int d^3r'\; \chi^\dagger_{k'}(\vec r\,')\; V(\vec r\,' - \vec r)\; \chi_k(\vec r\,').$$

If the probe states can be described as plane waves of definite momentum, the transition amplitude neatly factorizes into

$$M_{f,i} = \tilde V(q) \int d^3r\, \Phi^\dagger_{n'}(\vec r)\, \Phi_n(\vec r)\, e^{i\vec q\cdot\vec r},$$

where $\vec q = \vec k - \vec k'$. That is, the factor $\tilde V(q)$ is generic, characteriing the experimental technique; whereas the information we hope to obtain about the system we are probing is contained in the *response amplitude*[17]

---

16. The function $B(\kappa)$ presumably has a *simple* zero at the eigen-energy. Thus squaring it generates a positive quadratic function.
17. What is often measured is the *inclusive response function*, which is the squared modulus of the amplitude, summed over all final states, and perhaps including an energy-conserving δ-function.

$$S_{n',n} \stackrel{df}{=} \int d^3r\, \Phi^{\dagger}_{n'}(\vec{r}\,)\, \Phi_n(\vec{r}\,)\, e^{i\vec{q}\cdot\vec{r}}\ .$$

Now, when the momentum transfer $q = 0$, the above amplitude vanishes for $n' \neq n$ because of the orthogonality of the eigenstates of an Hermitian operator such as H_0. The problem is that when such integrals are computed numerically, by numerical integration of differential equations for the eigenstates, they often do not vanish, leading to spurious results at small q. The reason for the non-orthogonality is that when we replace differential operators (acting on a continuum) with difference operators (acting on a mesh), the latter may not be Hermitian, in the sense of maintaining a symmetric inner product.

Let us elaborate the latter statement in the one dimensional case (that would apply if, say, the Schroedinger equation was separable in spherical coordinates). After partial wave analysis the response amplitude would involve terms of the form

$$S_{fi} = \int_0^{\infty} dr\, r^2\, \varphi^{\dagger}_f(r)\, \varphi_i(r)\, j_0(qr) \quad ;$$

the Hermiticity of the Hamiltonian is reflected in the fact that the matrix elements of the differential operator

$$-\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\psi}{dr}\right) \equiv -\frac{1}{r}\frac{d^2}{dr^2}(r\psi)$$

may, integrating by parts, be written

$$\begin{aligned} M_{fi} &= \int_0^{\infty} dr\, r^2\, \varphi^{\dagger}_f(r)\left[-\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d}{dr}\right)\right]\varphi_i(r) \\ &= \int_0^{\infty} dr\, \frac{d}{dr}\Big(r\varphi^{\dagger}_f(r)\Big)\frac{d}{dr}\Big(r\varphi_i(r)\Big) - r\,\varphi^{\dagger}_f(r)\,\frac{d}{dr}\Big(r\varphi_i(r)\Big)\Bigg|_0^{\infty} . \end{aligned}$$

The contribution from the end points of the integral vanishes assuming the initial and final states are well-behaved—which is certainly the case for the wavefunctions of potentials that are non-singular at $r = 0$ and which vanish sufficiently rapidly as $r \to \infty$.

When we naively calculate the initial- and final state wavefunctions by solving numerically the appropriate differential equations, a strange thing happens. Since we solve on a mesh—typically one of constant spacing—we would expect to evaluate the integral for S_{fi} using a quadrature rule of uniform spacing, for example the trapezoidal rule. Thus we would write

$$S_{fi} \approx \delta r \sum_{k=0}^{N} \left[r_k\, \varphi^{\dagger}_f(r_k)\right]\left[r_k\, \varphi_i(r_k)\right] j_0(qr_k)\ .$$

The problem is that typically the sum, evaluated at $q = 0$ is not zero, and in fact is not even very small. What do we mean by "not very small"? For cases of interest the initial state represents a low-lying bound state of the system, hence is rather smooth; the final state, conversely, is often a scattering state at fairly high energy, hence oscillates sinusoidally. The overlap intgral will therefore have significant cancellations (this is an example of the Riemann-Lebesgue lemma in action) and be small whether the states are truly orthogonal or not. This means that criterion of smallness should be comparison of $S_{fi}(q{=}0)$ with the plane wave Born approximation (that is, replacing $r\varphi_f(r)$ with $\sin(Kr)$). On this scale, we typically find that to make the above sum significantly smaller than the PWBA requires solving the differential equation with an inordinately small mesh spacing δr.

Why should this be so? Investigation revealed that the wave functions, considered as eigenstates of a difference operator on a mesh of non-zero spacing, were not truly orthogonal because the method of solution had produced energy eigenvalues that were subtly in error. That is, if we want the expression

$$M_{fi} \approx \delta r \sum_{k=0}^{N} \left[r_k \, \varphi_f^{\dagger}(r_k) \right] \left[r_k \, \varphi_i(r_k) \right]$$

to be the exact orthogonality relation for these states, for a given mesh, then we must enforce the hermiticity of the difference operator Δ^2 on the discrete space. This amounts to insisting that the boundary condition for $r_n = n \, \delta r \to \infty$ is that bound state functions behave as

$$\beta_<^n = \left[1 + a^2/2 - a\left(1 + a^2/4\right)^{1/2} \right]^n .$$

If the energy eigenvalues are determined from this condition rather than the usual one employed by canned Schroedinger equation solvers, namely $r\varphi(r) \to e^{-\kappa r}$, the overlap integral will be small, even for fairly coarse meshes. That is, the solutions will be typically 5 to 10 orders of magnitude smaller than the PWBA, depending on the numerical precision used in the solution. This will ensure adequate precision in the computed response amplitudes.

On the following pages are BASIC and Forth versions of a program to solve the bound state Schrödinger eigenvalue problem.

```
' BASIC program for finding bound state wave functions of the          p. 1
' radial Schrödinger equation

DECLARE SUB make.y (B#)
DECLARE FUNCTION BE# (B1#, B2#, B3#)
DECLARE FUNCTION u# (r#)
DECLARE FUNCTION Yfin# (B#)
' test of numerical orthogonality in the Schroedinger equation
DEFDBL A-Z
DIM SHARED yy(81), rr(81)

CLS
B1 = 2: B2 = 5: B3 = 7
BB = BE(B1, B2, B3)
PRINT BB

CALL make.y(BB)
FOR i% = 0 TO 80 STEP 5
   PRINT rr(i%), yy(i%)
NEXT
END

FUNCTION BE (B1, B2, B3)
WHILE ABS(B3 - B1) > .0000001
  R1 = Yfin#(B1): R2 = Yfin#(B2): R3 = Yfin#(B3)
  a1 = R1 * (B2 - B3)
  a2 = R2 * (B3 - B1)
  a3 = R3 * (B1 - B2)
  ' B = (a1 * (B2 + B3) + a2 * (B1 + B3) + a3 * (B1 + B2)) / (a1 + a2 + a3) / 2
  alpha = a1 + a2 + a3
  beta = a1 * (B2 + B3) + a2 * (B1 + B3) + a3 * (B1 + B2)
  gamma = a1 * B2 * B3 + a2 * B1 * B3 + a3 * B1 * B2
  B = (beta - SQR(ABS(beta * beta - 4 * alpha * gamma))) / 2 / alpha
  ' PRINT B1; " "; B2; " "; B3; " "; B
  SELECT CASE B
         CASE IS < B1
             B3 = B2: B2 = B1: B1 = B
         CASE B1 TO B2
             B1 = B: B3 = (B2 + B3) / 2
         CASE B2 TO B3
             B3 = B: B1 = (B1 + B2) / 2
         CASE ELSE
             B1 = B2: B2 = B3: B3 = B
  END SELECT
WEND
BE = B
END FUNCTION
```

```
' BASIC program for finding bound state wave functions of the            p. 2
' radial Schrödinger equation

SUB make.y (B#)
    k2 = 2 * 939 * B# / 197.32 ^ 2
    h# = .2
    h2# = h# * h#: hk2# = h2# * k2
    x = 1 + hk2 / 2 - SQR(hk2 * (1 + hk2 / 4))
    y0 = 0    y1 = h * h: y1 = y1 * y1
    sum = 0
    yy(0) = 0: rr(0) = 0: i% = 0
    FOR r = h TO 16 STEP h
       i% = i% + 1
       y = (2 + h2 * (k2 + u(r))) * y1 - y0
       rr(i%) = r
       yy(i%) = y
       y0 = y1
       y1 = y
       sum = sum + y * y
    NEXT
     ' FOR r = 8 + h TO 16 STEP h
     '  i% = i% + 1
     '  y = y * x
     '  sum = sum + y * y
     '  yy(i%) = y
     '  rr(i%) = r
     ' NEXT    sum = 1 / SQR(sum * h)
    FOR i% = 1 TO 80
       yy(i%) = yy(i%) * sum
    NEXT
END SUB

FUNCTION u# (r#)
    u# = -2.45 / (1 + EXP((r# - 4.1) / .5)) + 12# / r# / r#
END FUNCTION

FUNCTION Yfin# (B#)
    k2 = 2 * 939 * B# / 197.32 ^ 2
    h = .2
    h2 = h * h
    hk2 = h2 * k2
    x = 1 + hk2 / 2 - SQR(hk2 * (1 + hk2 / 4))
    r = 0
    y0 = 0
    y1 = h2 * h2
    FOR r = h TO 16 STEP h
        y = (2 + h2 * (k2 + u(r))) * y1 - y0
        y0 = y1 * x
        y1 = y * x
    NEXT
    Yfin# = y1 * y1
    PRINT y1 * y1, B#
END FUNCTION
```

```
FALSE [IF]
Program to solve the nonrelativistic Schroedinger equation
for bound states ( E < 0 )
---------------------------------------------------
    (C) Copyright 1999  Julian V. Noble.
      Permission is granted by the author to
      use this software for any application pro-
      vided this copyright notice is preserved,
      as per GNU Public License agreement.
---------------------------------------------------
This is an ANS Forth compatible program with the following
environmental dependence:
      ANS FLOAT and FLOAT EXT wordsets
      ANS TOOLS EXT wordsets
      Assumes separate floating point stack
      Uses a FORmula TRANslator for ease of porting
      to other languages
[THEN]

MARKER  -sch
include ftran111.f      \ load FORmula TRANslator
include ansfalsi.f      \ load root finder

: fvariables    ( n --)  0 DO  FVARIABLE  LOOP  ;

9  fvariables kappa alpha r0 r dr psi psi0 psi1 drsq
3  fvariables chi chi0 chi1

: coul   ( f: r -- coul)  r f! f" 0.417/r "   ;   \ pure Coulomb potential

: U     ( f: r -- U)    \ Coulomb + nuclear potential
    r F!    r F@  r0 F@  F
    IF      f" 0.417*(3-(r/r0)^2)/r0 " F2/
    ELSE    f" 0.417/r "
    THEN    f" -2.414/(1+exp(2*(r-r0)) ) "   F+ ;

: startup
    f" r0 = 1.2*12^(1/3)"
    f" dr = 0.1"
    f" r = dr"
    f" drsq = dr^2"
    f" psi0 = 0"  f" psi1 = dr"
    f" chi0 = 0"  f" chi1 = dr" ;

: psi_step
    f" psi = ( 2+drsq*( U(r)+kappa^2) )*psi1 - psi0"
    f" chi = ( 2+drsq*( coul(r)+kappa^2) )*chi1 - chi0"
    f" psi0 = psi1"    f" psi1 = psi"
    f" chi0 = chi1"    f" chi1 = chi"
    f" r=r+dr" ;

: display   r f@ f.  psi1 f@ f.  psi0 f@ f.   chi1 f@ fs.
            f" psi1 / abs(chi1) " fs. ;
```

```
2  fvariables OldB  NewB

: beta  ( f: kappa -- B[k])
    kappa F!
    startup
    psi_step
    f" NewB = psi1/abs(chi1)"
    BEGIN   f" OldB = NewB "
            psi_step
            f" NewB = psi1/abs(chi1)"
            f" abs(NewB-OldB)/(NewB+OldB) "  1e-7 F<
    UNTIL
    NewB F@ ;

: energy   ( f: -- energy)   f" 20.71 * kappa^2"  ;

: wf      \ calculate the wavefunction
    startup
    BEGIN   r F@ 10e0 F<
    WHILE   CR display psi_step
    REPEAT  ;

\ say use{ beta 0.2e0 1e0  1e-4 )falsi  energy fs.
```