PHYS 410/555 Computational Physics: Solution of ODEs

(Reference *Numerical Recipes*, Chapters 16, 17)

Overview

- "Theory"
  - Casting systems of ODEs in first order form (canonical form)
  - Boundary / initial conditions
- Some Basic Numerical Techniques
  - Euler method
  - Second-order Runge-Kutta
- Using "Canned" Software
  - `ODEPACK` routine `lsoda`
- Applications
  - Quadrature (definite integrals)
  - Initial value problems (dynamics)
  - Boundary value problems

Note: There are *many* applications in virtually every sub-field of physics.

Casting Systems of ODEs in First Order Form

- Can *always* reduce systems of ODEs to set of first order DEs by introducing appropriate new (auxiliary) variables.

Example 1

$$y''(x) + q(x)y'(x) = r(x) \qquad {}' \equiv \frac{d}{dx} \tag{1}$$

- Introduce new variable $z(x) \equiv y'(x)$, then (1) becomes

$$y' = z \tag{2}$$
$$z' = r - qz \tag{3}$$

Example 2

$$y''''(x) = f(x) \tag{4}$$

- Introduce new variables

$$y_1(x) \equiv y'(x) \tag{5}$$
$$y_2(x) \equiv y''(x) \tag{6}$$
$$y_3(x) \equiv y'''(x) \tag{7}$$

then (4) becomes

$$y' = y_1 \tag{8}$$
$$y_1' = y_2 \tag{9}$$
$$y_2' = y_3 \tag{10}$$
$$y_3' = f \tag{11}$$

- Thus, the generic problem in ODEs is reduced to study of a set of N coupled, *first-order* DEs for the functions, $y_i, i = 1, 2, \ldots, N$

$$y_i'(x) \equiv \frac{dy_i}{dx}(x) = f_i(x, y_1, y_2, \cdots, y_N) \qquad i = 1, 2, \ldots N \tag{12}$$

where the $f_i(\cdots)$ are *known functions* of x and y_i

- *Equivalent forms:* $\mathbf{y} \equiv (y_1, y_2, \cdots, y_N)$

$$\mathbf{y}'(x) = \mathbf{f}(x, \mathbf{y}) \tag{13}$$
$$\dot{\mathbf{y}}(t) = \mathbf{f}(t, \mathbf{y}) \tag{14}$$

Boundary / Initial Conditions

- ODE problem not completely specified by DEs themselves
- Nature of boundary conditions is crucial aspect of problem
- Generally, BCs are *algebraic* conditions on certain values of the y_i in (12) that are to be satisfied at discrete specified points.
- Generally will need N conditions for N-th order system
- BCs divide ODE problems into 2 broad classes

1) Initial Value Problems

- All the y_i are given at some starting (initial) value, $t_{\min}$ and we wish to find the y_i at some final value, $t_{\max}$, or at some *set* of values

$$t_n, \qquad t_{\min} \le t_n \le t_{\max} \qquad n = 0, 1, 2, \cdots \tag{15}$$

2) (Two-point) Boundary Value Problems

- BCs are specified at more than one value of x. Typically some will be specified at $x = x_{\min}$, the remainder at $x = x_{\max}$.
- Have already considered some 2-pt BVPs, and their solution via finite difference techniques

We will focus on general techniques / software for solving IVPs, and some simple BVPs.

Some Basic Numerical Techniques for IVPs

We adopt the notation of *Numerical Recipes*, and illustrate the methods for the case of a scalar equation. The generalization to systems is straightforward.

1) The Euler Method

- Consider two values of x, x_n and $x_{n+1} = x_n + h$ (h is often called the "step size", and is completely analogous to the mesh spacing, h, used in our previous work on FD approximations)
- Then the (forward) Euler method is given by

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{16}$$

- Note that we use this formula to "advance" solution from $x = x_n$ to $x = x_{n+1} = x_n + h$
- Can easily derive from $O(h)$ (forward) finite difference approximation

$$\frac{y_{n+1} - y_n}{h} = y'_n + O(h) \tag{17}$$

$$y' = f(x, y) \longrightarrow \frac{y_{n+1} - y_n}{h} = f(x_n, y_n) \tag{18}$$

- *Accuracy*: $O(h^2)$ *per step*. For fixed final $x = x_f$, number of steps scales as h^{-1}, so *global* accuracy is $O(h)$
- OK for demonstration purposes, but should *never* be used in practice—not very accurate, not very *stable*!

2) Second-order Runge-Kutta (Mid-point Method)

- The second-order Runge-Kutta method is given by

$$\begin{aligned} k_1 &= hf(x_n, y_n) && (19) \\ k_2 &= hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) && (20) \\ y_{n+1} &= y_n + k_2 && (21) \end{aligned}$$

- *Global accuracy*: $O(h^2)$
- *Derivation*

$$\frac{y_{n+1} - y_n}{h} = f(x_{n+1/2}, y_{n+1/2}) + O(h^2) \qquad (22)$$

where $x_{n+1/2} \equiv x_n + h/2$, $y_{n+1/2} \equiv y(x_{n+1/2})$. (Exercise: Verify the above, and compute the actual form of the leading order error term.)

To retain $O(h^2)$ accuracy, need to evaluate $f(x_{n+1/2}, y_{n+1/2})$ to $O(h^2)$ (i.e. can neglect $O(h^2)$ terms), so, in turn, need to know $y_{n+1/2}$ to $O(h^2)$; proceed via Taylor series expansion

$$\begin{aligned} y_{n+1/2} &= y_n + \frac{1}{2}hy'_n + O(h^2) \\ &= y_n + \frac{1}{2}k_1 + O(h^2) \\ \Longrightarrow y_{n+1} &= y_n + hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right) \end{aligned}$$

as advertised.

Although it is “good for you” to understand some of the theory that underlies a modern ODE solver, the state of such solvers is very high, and, as with linear system solvers, can frequently be used as “black boxes”—with the important proviso that we *always* make every reasonable attempt to *validate* our results (convergence tests, independent residual tests, conserved quantities, etc.)

ODEPACK

- Public-domain collection of routines for solution of systems of ODEs (IVPs)

- We will focus on one routine, `lsoda`, which has the following header:

```
 subroutine lsoda(f, neq, y, t, tout, itol, rtol, atol, itask,
&                  istate, iopt, rwork, lrw, iwork, liw, jac, jt)
    external    f, jac
    integer     neq, itol, itask, istate, iopt, lrw, liw, jt
    real*8      t, tout, rtol, atol
    real*8      y(neq), rwork(lrw)
    integer     iwork(liw)
```

  See source code and sample "driver" program (`tlsoda.f`) for full description of parameters and routine operation

- `f, jac`: Names of routines (subroutines) for evaluating right hand side of ODES (`f`), and Jacobian of system (`jac`). `f` is *required*, `jac` is *optional*, typically a "dummy" routine

- `neq`: number of equations / size of system (canonical first-order form)

- `y`: On input, (approximate) values of unknowns at $t = t$ `( y(i) , i = 1 , neq )`; On output, (approximate) values of unknowns at $t = t_{\text{out}}$

- `t, tout`: Limits of current integration interval

- `itol, rtol, atol`: Tolerance (error-control) parameters (see `lsoda.f`, `tlsoda.f` for details)

- `itask`: Set = 1 for normal operation

- `istate`: Set = 1 intially for normal operation, thereafter set = 2 for normal operation (routine will automatically do this if integration on first interval is successful); check for negative value on return to detect abnormal completion

- `iopt`: Normally set = 0 (no optional inputs, but, again, refer to the source code for full details)

- `rwork(lrw)`: `real*8` work array of length `lrw`;
  minimum value of `lrw` is `22 + 16 * neq`

- `iwork(liw)`: `integer` work array of length `liw`;
  minimum value of `liw` is `20 + neq`

- `jt`: Set = 2 for normal operation—supply "dummy" Jacobian routine, `lsoda` will approximately compute Jacobian numerically if and when necessary

Crucial User-supplied Routine Called by `lsoda`

- `f`: Evaluates "RHS" of system of ODEs (12); *must* have header as follows

```
      subroutine f(neq, t, y, ydot)
         implicit   none
         integer    neq
         real*8     t, y(neq), ydot(neq)
```

- *Inputs:* `neq, t, ( y(j) , j = 1 , neq )`
- *Output:* `( ydot(j) , j = 1 , neq )`

`lsoda` *Tolerance Parameters:* `itol, atol, rtol`

- `lsoda` will control step-size, order of method and type of method so that estimated local error in `y(i)` is less than

```
ewt(i) = rtol * abs(y(i)) + atol       itol .eq. 1
ewt(i) = rtol * abs(y(i)) + atol(i)    itol .eq. 2
```

  Thus, local error tests passes if, for *each* component `y(i)`, either the absolute error is less than `atol` (or `atol(i)`), or the relative error is less than `rtol`

Choosing Error Tolerances

- Can experiment, but `rtol = atol = tol` (single control parameter) often works well, particularly for y_i that exhibit significant dynamical range
- Some exceptions (of course); for example, consider 2-d motion in polar coordinates, (r, θ). If we use relative control, then for $\theta \gg 2\pi$, "acceptable local error" $\delta\theta$ will increase. Better idea to try to keep $\delta\theta$ constant via "pure absolute" control (`rtol = 0.0d0`)
- Solution eror will almost certainly grow with time, so for fixed final integration time, t_f, will need to *calibrate* error estimates, i.e. assume that

$$\left\| y_{\text{computed}}(t_f) - y_{\text{exact}}(t_f) \right\| \approx \kappa(t_f)\texttt{tol} \tag{23}$$

  where $\kappa(t_f)$ can be determined via calibration *if* y_{exact} *is known*

- However, even if y_{exact} is *not* known (typical case!), (23) tells us that we can expect error (at fixed time) to be *proportional* to `tol`; e.g. if `tol` goes from `1.0d-6 -> 1.0d-10`, should expect solution error to be down by about 4 orders of magnitude
- *Caveat emptor!* ("User beware!")

Checking/validating Results From ODE Integrators

1) Monitoring Conserved Quantities

- Example: For dynamical systems with a Lagrangian (Hamiltonian), total energy, $E(t)$ is conserved: $dE/dt = 0$
- Monitor variation $\delta\hat{E}(t, \epsilon)$ of computed energy $\hat{E}(t, \epsilon)$:

$$\delta\hat{E}(t, \epsilon) = \hat{E}(t, \epsilon) - \hat{E}(t_{\min}, \epsilon) \tag{24}$$

where ϵ is the error tolerance for the integrator.

- Should find that this is an $O(\epsilon)$ quantity, i.e. for ϵ sufficiently small, should have

$$\delta \hat{E}(t, \epsilon) = \epsilon f(t) + \text{higher order terms} \tag{25}$$

- Thus, e.g., if we take $\epsilon \to \epsilon/10$, should see $\delta\hat{E} \to \delta\hat{E}/10$ (approximately, so long as $\epsilon \gg \epsilon_{\text{machine}}$)

2) Independent Residual Evaluation

- *Idea*: Attempt to directly verify that approximate solution, $\hat{u}$ (u previously y!) satisfies the ODE(s) through the use of an *independent discretization* of the ODE (i.e. a discretization distinct from that used by the ODE integrator).
- *Note:* In numerical analysis, a *residual* quantity is one that should tend to 0 in some appropriate limit
- Let

$$L\left[u(t)\right] \equiv Lu(t) = 0 \tag{26}$$

be our ODE, where L is a differential operator, and u, in general can be a *vector* of functions; will assume that L is *linear*, but technique generalizes to non-linear case

- Let $\hat{u}(t, \epsilon)$ be the solution computed by our ODE integrator for tolerance ϵ, and consider computing $\hat{u}$ on a regular mesh of output times

$$t^h \equiv t_n = t_{\min},\ t_{\min} + h,\ t_{\min} + 2h,\ \cdots \tag{27}$$

and consider, for concreteness, a second-order (in h) finite difference approximation to the ODE

$$L^h u^h = 0 \qquad L^h = L + O(h^2) \tag{28}$$

- Note that (28) *defines* u^h, and that

$$u^h(t) \neq \hat{u}\left(t^h, \epsilon\right) \tag{29}$$

- The finite difference operator L^h can be expanded as follows

$$L^h = L + h^2 E_2 + h^4 E_4 + \cdots \tag{30}$$

where, as discussed previously, E_2, E_4, etc. are higher order differential operators (involve higher order derivatives than L).

- Now, we can write

$$\hat{u}(t, \epsilon) = u(t) + e(t, \epsilon) \tag{31}$$

where $e(t, \epsilon)$ is the error in the solution computed using the ODE integrator

- Next, consider the action of L^h on $\hat{u}(t, \epsilon)$; suppressing explicit t-dependence, we have

$$\begin{aligned} L^h \hat{u}(\epsilon) &= \left(L + h^2 E_2 + h^4 E_4 + \cdots\right)(u + e(\epsilon)) && (32) \\ &= Lu + h^2 E_2 u + \cdots + L^h e(\epsilon) && (33) \\ &\approx h^2 E_2 [u] + L^h [e(\epsilon)] && (34) \end{aligned}$$

- Now, assume that

$$h^2 E_2 [u] \gg L^h [e(\epsilon)] \tag{35}$$

  then

$$L^h \hat{u} \approx h^2 E_2 [u] = O(h^2) \tag{36}$$

- With a high-accuracy ODE solver such as `lsoda`, it is usually possible to satisfy (35), at least over *some* time interval $(t_{\min}, t_{\max})$, and as long as h is not chosen too small
- *Note:* Key idea is to show/check *correctness* of implementation; e.g. checking for errors in coding of equations.

Example:

- Consider the ODE describing simple harmonic motion, (with the gross abuse of notation, $' \equiv d/dt$!):

$$u''(t) = -u(t) \tag{37}$$

  that we will solve on $0 \le t \le t_{\max}$ with the initial values $u(0)$ and $u'(0)$ given

- General solution of (37) is

$$\begin{aligned} u(t) &= A\sin(t) + B\cos(t) && (38) \\ u'(t) &= A\cos(t) - B\sin(t) && (39) \end{aligned}$$

  Evaluating (39) at $t = 0$ yields

$$\begin{aligned} A &= u'(0) && (40) \\ B &= u(0) && (41) \end{aligned}$$

  So specific solution satisfying initial conditions is

$$u(t) = u'(0)\sin(t) + u(0)\cos(t) \tag{42}$$

- Cast (37) in canonical form; define

$$\begin{aligned} y_1 &\equiv u && (43) \\ y_2 &\equiv u' && (44) \end{aligned}$$

  Then (37) becomes

$$\begin{aligned} y_1' &\equiv y_2 && (45) \\ y_2' &\equiv -y_1 && (46) \end{aligned}$$

- *RHS routine called by* `lsoda`

```
c===========================================================
c     Implements differential equations:
c
c     u'' = -u
c
c     y(1) := u
c     y(2) := u'
c
c     y(1)' :=  y(2)
c     y(2)' := -y(1)
c
c     Called by ODEPACK routine LSODA.
c===========================================================
      subroutine fcn(neq,t,y,yprime)
         implicit    none

         integer     neq
         real*8      t,     y(neq),    yprime(neq)

         yprime(1) =  y(2)
         yprime(2) = -y(1)

         return
      end
```

Independent Residual Evaluator

- First, rewrite (37) in form (26)

$$u''(t) + u(t) = 0 \tag{47}$$

- Next, using e.g. `lsoda`, generate solution $\hat{u}(t^h, \epsilon)$ on a level-ℓ uniform mesh:

$$t^h_n = 0, h, 2h, \cdots t_{\max} \tag{48}$$

with

$$h = \frac{t_{\max}}{2^\ell} \tag{49}$$

- Then, apply $O(h^2)$ finite-difference discretization of (47) to $\hat{u}$ to compute *residual* R_n:

$$R_n \equiv \frac{\hat{u}_{n+1} - 2\hat{u}_n + \hat{u}_{n-1}}{h^2} + \hat{u}_n \qquad n = 1, 2, \cdots 2^\ell - 1 \tag{50}$$

- In particular, should find that RMS value (ℓ_2 norm) of R_n is an $O(h^2)$ quantity:

$$\left[\frac{\sum_n |R_n|^2}{2^\ell - 1}\right]^{\frac{1}{2}} \equiv \|\mathbf{R}\|_2 = O(h^2) \tag{51}$$

See `tlsoda.f`, `chk-tlsoda.f` for implementation.

Note on Solution Sensitivity/Ill-conditioning

- In integrating from t to $t_{\rm out}$, `lsoda` will typically evaluate RHS of ODEs at many intermediate values t_I, $t \le t_I \le t_{\rm out}$ according to the details of the algorithm, and the user-specified tolerances; these t_I are typically "invisible" to the user
- If, as is frequently the case, one wants to tabulate the solution at many values, e.g. on a grid

$$t_n \equiv t_{\rm min},\ t_{\rm min} + h,\ \cdots t_{\rm max} - h,\ t_{\rm max} \tag{52}$$

  then will generally find that, for fixed tolerance, the computed value at $t = t_{\rm max}$, e.g., will depend on specifics of the output values of t_n requested

- If results are *highly* dependent on choice of t_n, this is a sign that problem is *sensitive* (poorly conditioned); the gravitational n-body problem is a classic example
- In such a case, will also tend to find significant dependence of results on small changes in error tolerances

BOTTOM LINE: Need to be CAREFUL in use of "black box" software!

IVP Applications

1) "Quadrature"/Definite integrals

- Suppose we wish to evaluate definite integral

$$\int_{x_1}^{x_2} f(x)dx \tag{53}$$

- Consider $I(x)$ such that

$$\frac{dI}{dx} = f(x) \tag{54}$$

  Then, we have

$$\begin{aligned} \int_{x_1}^{x_2} \frac{dI}{dx}dx &= \int_{x_1}^{x_2} f(x)dx & (55)\\ \Longrightarrow I(x_2) - I(x_1) &= \int_{x_1}^{x_2} f(x)dx & (56)\end{aligned}$$

  So, with the initial condition

$$I(x_1) = 0 \tag{57}$$

  we have

$$I(x_2) = \int_{x_1}^{x_2} f(x) dx \tag{58}$$

Example:

- Use above technique and `lsoda` to compute approximate value of

$$I(x; x_1, x_2) = \int_{x_1}^{x_2} e^{-x^2} dx \tag{59}$$

  where, for example, $I(x, 0, \infty) = \sqrt{\pi}/2$.

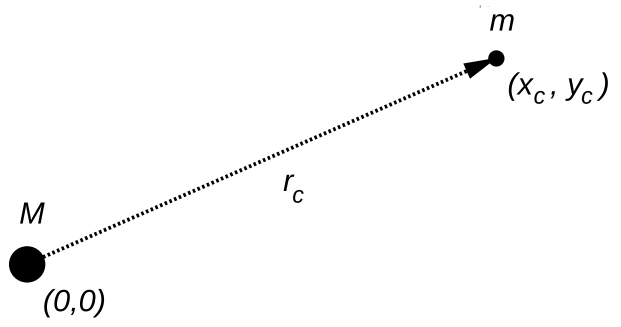
- *RHS routine called by* `lsoda`

```
subroutine fcn(neq,x,y,yprime)
   implicit    none

   integer     neq
   real*8      x,     y(neq),    yprime(neq)

   yprime(1) = exp(-x**2)
   return
end
```

- Should expect *local* tolerance to provide better estimate of *global* accuracy in this case (quadrature)—why?

2) Restricted 2-body problem

- Consider point particle with mass m, interacting with another mass, M, with $M \gg m$—treat M as *fixed*, study dynamics of m (test particle)



- *Dynamical variables:* coordinates of test particle— x_c, y_c

- *Equations of motion*

$$\sum \mathbf{F} = m\,\mathbf{a} \tag{60}$$

$$m\,\mathbf{a} = -G\,\frac{Mm}{|\mathbf{r}_c|^2}\,\hat{\mathbf{r}}_c = -G\,\frac{Mm}{{r_c}^3}\,\mathbf{r}_c \tag{61}$$

- Divide by m, and resolve into x and y components:

$$\ddot{x}_c = -\frac{GM}{{r_c}^3}\,x_c \tag{62}$$

$$\ddot{y}_c = -\frac{GM}{{r_c}^3}\,y_c \tag{63}$$

- 2 second-order ODEs $\longrightarrow$ 4 first order ODEs
- Rewrite in canonical form; define

$$y_1 = x_c \tag{64}$$

$$y_2 = y_c \tag{65}$$

$$y_3 = \dot{x}_c \tag{66}$$

$$y_4 = \dot{y}_c \tag{67}$$

Then we have

$$\dot{y}_1 = y_3 \tag{68}$$

$$\dot{y}_2 = y_4 \tag{69}$$

$$\dot{y}_3 = -\frac{GM}{{r_c}^3}\,y_1 \tag{70}$$

$$\dot{y}_4 = -\frac{GM}{{r_c}^3}\,y_2 \tag{71}$$

where

$${r_c}^3 = \left({y_1}^2 + {y_2}^2\right)^{3/2} \tag{72}$$

- Initial values:

$$y_1(0),\ y_2(0): \qquad \text{Initial position of particle} \tag{73}$$

$$y_3(0),\ y_4(0): \qquad \text{Initial velocity of particle} \tag{74}$$

- Initial conditions for circular orbit: $\mathbf{v} \perp \mathbf{r}_c$

$$|\mathbf{a}| = \frac{v^2}{r_c} = \frac{GM}{{r_c}^2} \Longrightarrow v = \left(\frac{GM}{r_c}\right)^{1/2} \tag{75}$$

  Then, setting $G = M = 1$ (choice of units)

$$\Longrightarrow v = {r_c}^{-1/2} \tag{76}$$

- Typical circular orbit

$$r_c = 1, \qquad v = 1 \tag{77}$$
$$\mathbf{r}_c(0) = (1.0, 0.0) \qquad \mathbf{v}(0) = (0.0, 1.0) \tag{78}$$

- Will get elliptical orbits by changing any of $x_c(0), y_c(0), v_x(0), v_y(0)$ (If changes too drastic, may get hyperbolic or parabolic (unbound) orbits)

"Quality assessment" (calibration)

- Make use of existence of *conserved* total energy, $E_{\rm tot}$ and angular momentum (w.r.t. $(0,0)$), $J_{\rm tot}$

$$\begin{aligned} E_{\rm tot} &= T + V_{\rm grav} = \frac{1}{2}mv^2 - G\,\frac{Mm}{r_c} && (79)\\ J_{\rm tot} &= |\mathbf{r} \times m\,\mathbf{v}| && (80) \end{aligned}$$

- Particle mass enters as arbitrary parameter (test particle limit), compute *specific* quantities, E, J:

$$\begin{aligned} E &= \frac{E_{\rm tot}}{m} = \frac{1}{2}v^2 - G\,\frac{M}{r_c} && (81)\\ J &= \frac{J_{\rm tot}}{m} = |\mathbf{r} \times \mathbf{v}| && (82) \end{aligned}$$

  Get

$$\begin{aligned} E &= \frac{1}{2}\left({v_x}^2 + {v_y}^2\right) - \frac{GM}{({x_c}^2 + {y_c}^2)^{1/2}} && (83)\\ J &= x v_y - y v_x && (84) \end{aligned}$$

- As discussed previously, should expect

$$\begin{aligned} \Delta E(t) &\equiv E(t) - E(0) \approx \epsilon\,\kappa_E(t) && (85)\\ \Delta J(t) &\equiv J(t) - J(0) \approx \epsilon\,\kappa_J(t) && (86) \end{aligned}$$

  where ϵ is the `lsoda` tolerance; e.g. if we make the tolerance 10 times more stringent, should find roughly factor of 10 improvement in energy, angular momentum conservation

- *RHS routine called by* `lsoda`

```
      subroutine fcn(neq,t,y,yprime)
         implicit    none

c-----------------------------------------------------------
c        Problem parameters (G, M) passed in via common
c        block defined in 'fcn.inc'
c-----------------------------------------------------------
         include    'fcn.inc'

         integer     neq
         real*8      t,     y(neq),    yprime(neq)

         real*8      c1

         c1 = -G * M / (y(1)**2 + y(2)**2)**1.5d0

         yprime(1) = y(3)
         yprime(2) = y(4)
         yprime(3) = c1 * y(1)
         yprime(4) = c1 * y(2)

         return
      end
```

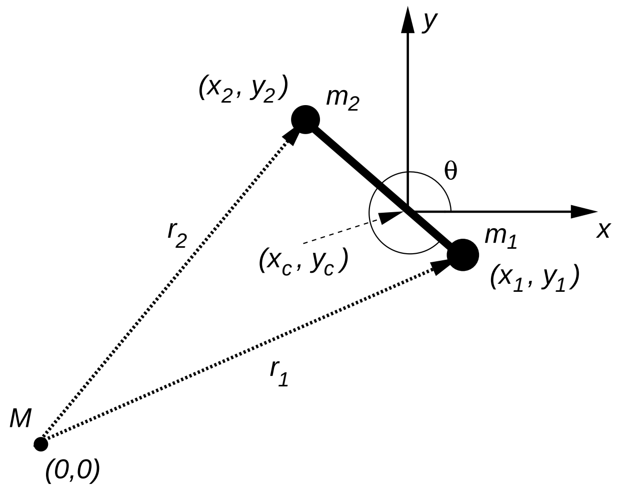
- *Include file defining additional parameters*

```
c-------------------------------------------------------------
c     Application specific common block for communication with
c     derivative evaluating routine 'fcn' (optional) ...
c-------------------------------------------------------------

      real*8  G,         M
      common / com_fcn /
     &        G,         M
```

PHYS 410/555 Computational Physics: The Orbiting Dumbbell
(Following Giordano, *Computational Physics*, Section 4.6)

Background: With the exception of Hyperion, which is one of Saturn's satellites, all of the moons in the solar system are "spin-locked"; a moon which is spin-locked has a rotational frequency, ω about its own spin axes which is the same as its orbital frequency, Ω. The supposed mechanism by which the spin-locking comes about is somewhat involved; however the point is that Hyperion is somehow exceptional—study of its $\omega(t)$ suggests that it is tumbling *chaotically* in its orbit about Saturn, which is presumably due to both to its peculiar shape (like that of an egg), and the fact that it is in an elliptical orbit about Saturn.

To investigate the effects of a non-spherical distribution of mass on a satellite's spin as it orbits its parental body, we consider the model of an "orbiting dumbbell".



Consider two test masses, m_1, m_2 connected by a massless rigid rod of length d, in orbit about a mass, $M \gg m_1, m_2$ as shown in the figure above. Let $(x_i, y_i), i = 1, 2$ be the coordinates of the two test masses, let (x_c, y_c) be the coordinates of the dumbbell's center of mass, and let θ be the angle the rod makes with the x-axis. Defining

$$\mu \equiv \frac{m_2}{m_1 + m_2}$$

the distances of the masses from the center of mass are

$$d_1 = \mu\, d \qquad d_2 = (1-\mu)d$$

then

$$x_i = x_c \pm d_i \cos\theta \qquad y_i = y_c \pm d_i \sin\theta$$

The moment of inertia of the dumbbell about (x_c, y_c) is

$$I = m_1 {d_1}^2 + m_2 {d_2}^2 = \frac{m_1 {m_2}^2}{(m_1+m_2)^2} d^2 + \frac{m_2 {m_1}^2}{(m_1+m_2)^2} d^2 = \frac{m_1 m_2}{m_1+m_2} d^2$$

The equations of motion for the body are

$$(m_1 + m_2)\, \mathbf{a_c} = (m_1 + m_2)\, \ddot{\mathbf{r}}_c = \sum \mathbf{F} = \mathbf{F}_1 + \mathbf{F}_2$$

$$I\alpha = I\ddot{\theta} = \sum \tau = \mathbf{d}_1 \times \mathbf{F}_1 + \mathbf{d}_2 \times \mathbf{F}_2$$

where

$$\mathbf{F}_1 = -\frac{GMm_1}{{r_1}^3}\left[x_1, y_1\right]$$
$$\mathbf{F}_2 = -\frac{GMm_2}{{r_2}^3}\left[x_2, y_2\right]$$

are the gravitational forces acting on m_1 and m_2 respectively. The translational equations yield:

$$(m_1 + m_2)\ \ddot{x}_c = -GM\left(\frac{m_1}{{r_1}^3}x_1 + \frac{m_2}{{r_2}^3}x_2\right)$$
$$(m_1 + m_2)\ \ddot{y}_c = -GM\left(\frac{m_1}{{r_1}^3}y_1 + \frac{m_2}{{r_2}^3}y_2\right)$$

or

$$\ddot{x}_c = -GM\left(\frac{1-\mu}{{r_1}^3}x_1 + \frac{\mu}{{r_2}^3}x_2\right)$$
$$\ddot{y}_c = -GM\left(\frac{1-\mu}{{r_1}^3}y_1 + \frac{\mu}{{r_2}^3}y_2\right)$$

The rotational equation gives:

$$\begin{aligned}
I\ddot{\theta} = \mathbf{d}_1 \times \mathbf{F}_1 + \mathbf{d}_2 \times \mathbf{F}_2 &= -GM\frac{m_1}{{r_1}^3}d_1\left(\cos\theta y_1 - \sin\theta x_1\right) + GM\frac{m_2}{{r_2}^3}d_2\left(\cos\theta y_2 - \sin\theta x_2\right) \\
&= -GM\frac{m_1}{{r_1}^3}d_1\left(\cos\theta y_c - \sin\theta x_c\right) + GM\frac{m_2}{{r_2}^3}d_2\left(\cos\theta y_c - \sin\theta x_c\right) \\
&= GM\left(\frac{m_2}{{r_2}^3}d_2 - \frac{m_1}{{r_1}^3}d_1\right)\left(\cos\theta y_c - \sin\theta x_c\right) \\
&= GM\frac{m_1 m_2}{m_1 + m_2}d\left(\frac{1}{{r_1}^3} - \frac{1}{{r_2}^3}\right)\left(\sin\theta x_c - \cos\theta y_c\right)
\end{aligned}$$

so

$$\ddot{\theta} = \frac{GM}{d}\left(\frac{1}{{r_1}^3} - \frac{1}{{r_2}^3}\right)\left(\sin\theta x_c - \cos\theta y_c\right)$$

Summarizing, we have:

$$\begin{aligned}
\ddot{x}_c &= -GM\left(\frac{1-\mu}{{r_1}^3}x_1 + \frac{\mu}{{r_2}^3}x_2\right) \\
\ddot{y}_c &= -GM\left(\frac{1-\mu}{{r_1}^3}y_1 + \frac{\mu}{{r_2}^3}y_2\right) \\
\ddot{\theta} &= \frac{GM}{d}\left(\frac{1}{{r_1}^3} - \frac{1}{{r_2}^3}\right)\left(\sin\theta x_c - \cos\theta y_c\right)
\end{aligned}$$

where

$$\begin{aligned}
\mu &\equiv \frac{m_2}{m_1 + m_2} \\
d_1 &= \mu\, d \\
d_2 &= (1-\mu)d \\
x_i &= x_c \pm d_i \cos\theta \\
y_i &= y_c \pm d_i \sin\theta \\
{r_i}^3 &= \left({x_i}^2 + {y_i}^2\right)^{3/2}
\end{aligned}$$

The total (conserved) energy of the system is

$$E_{\text{tot}} = T_{\text{trans}} + T_{\text{rot}} + V_{\text{grav}}$$

where

$$\begin{aligned}
T_{\text{trans}} &\equiv \frac{1}{2}\left(m_1 + m_2\right)\left(\dot{x}_c^2 + \dot{y}_c^2\right) \\
T_{\text{rot}} &\equiv \frac{1}{2}\frac{m_1 m_2}{m_1 + m_2}\, d^2\, \dot{\theta}^2 \\
V_{\text{grav}} &\equiv -GM\left(\frac{m_1}{r_1} + \frac{m_2}{r_2}\right)
\end{aligned}$$

PHYS 410/555 Computational Physics

The Method of Lines for the Wave Equation

One approach to the numerical solution of time-dependent *partial differential equations* (PDEs) is to use a discretization technique, such as finite-differencing, but only apply it explicitly to the *spatial* part(s) of the PDE operator(s) under consideration. Following the spatial discretization, one is left with a set of coupled *ordinary differential equations* in t, which can then often be solved by a "standard" ODE integrator such as `LSODA`.

As an example of this technique, consider the *wave equation* in one space dimension (often called the "1D wave equation"):

$$\frac{\partial^2}{\partial t^2} u(x,t) = c^2 \frac{\partial^2}{\partial x^2} u(x,t) \tag{1}$$

Introducing the notation that a subscript denotes partial differentiation, and suppressing the explicit x and t dependence, (1) can also be written as

$$u_{tt} = c^2 u_{xx} \tag{2}$$

As you probably know, the wave equation describes propagation of disturbances, or waves, at a speed c: waves can either travel to the right (velocity $+c$), or to the left (velocity $-c$). Without loss of generality, we can always choose units such that $c = 1$, and, for convenience, we will do so. Our wave equation then becomes:

$$u_{tt} = u_{xx} \tag{3}$$

As with any differential equation, boundary conditions play a crucial role in fixing a solution of (3). Here, we will solve the wave equation on the domain

$$0 \le x \le 1 \qquad t \ge 0 \tag{4}$$

and will thus have to provide boundary conditions at $x = 0$ and $x = 1$, as well as initial conditions at $t = 0$.

For concreteness, we will prescribe *Dirichlet boundary conditions*:

$$u(0,t) = u(1,t) = 0 \tag{5}$$

as well as the following *initial conditions*:

$$\begin{aligned} u(x,0) &= u_0(x) = \exp\left(-\left(\frac{x-x_0}{\Delta}\right)^2\right) && (6) \\ u_t(x,0) &= 0 && (7) \end{aligned}$$

where x_0 $(0 < x_0 < 1)$ and Δ are specified constants.

If we think in terms of small-amplitude waves propagating on a string, then the Dirichlet conditions correspond to keeping the ends of the string fixed. The interpretation of the initial conditions is as follows: In solving (3) we have the freedom to specify the amplitude of the disturbance for all values of x, as well as the time-rate of change of that amplitude, again for all values of x.

We thus have two functions worth of freedom in specifying our initial conditions. We set the initial amplitude to some functional form given by $u_0(x)$; here we use a "gaussian pulse" that is centred at x_0, and that has an overall effective width of a few $\times$ Δ. We also set the initial time rate of change of the amplitude to be 0 for all x.

Such data is known as *time symmetric*, since it defines an instant in the evolution of the wave equation where there is a $t \to -t$ symmetry. In other words, with time symmetric initial data, if we integrate *backward* in time, we will see exactly the same solution as a function of $-t$ as we see integrating forward in time.

Since the wave equation describes propagating disturbances, and given that the initial conditions *are* time symmetric, a little reflection might convince you that the initial conditions (6) and (7) must represent a superposition of equal amplitude right-moving and left-moving pulses. Thus, we should expect the solution of (3), subject to (5), (6) and (7) to describe the propagation of two equal-amplitude pulses that are initially coincident, but that subsequently move apart reflect off $x = 0$ and $x = 1$ respectively, move together, through each other, then apart, etc. etc. Indeed, this is precisely the behaviour we will observe in our subsequent numerical solution.

As mentioned above, the *method of lines*, involves an explicit discretization *only of the spatial part of the PDE operator*. Here we will use the familiar $O(h^2)$ finite-difference approach to the treatment of $u_{xx} \equiv \partial^2 u/\partial x^2$.

However, before proceeding to the spatial discretization, we first note that (3) is a second-order-in-time equation. In order that our approach eventually produce a set of *first order* ODEs in t, we introduce an auxiliary variable, $v(x,t)$,

$$v(x,t) \equiv u_t(x,t) \equiv \frac{\partial u}{\partial t}(x,t) \tag{8}$$

and then rewrite (3) as the *system:*

$$\begin{aligned} u_t &= v && (9) \\ v_t &= u_{xx} && (10) \end{aligned}$$

The boundary conditions become

$$u(0,t) = u(1,t) = v(0,t) = v(1,t) = 0 \tag{11}$$

while the initial conditions are now

$$\begin{aligned} u(x,0) &= u_0(x) = \exp\left(-\left(\frac{x-x_0}{\Delta}\right)^2\right) && (12) \\ v(x,0) &= 0 && (13) \end{aligned}$$

We can now proceed with the spatial discretization. To that end, we replace the continuum spatial domain $0 \le x \le 1$ by a uniform finite difference mesh, x_j:

$$x_j \equiv (j-1)h \qquad j = 1, 2, \cdots N \qquad h \equiv (N-1)^{-1} \tag{14}$$

and introduce the discrete unknowns, u_j and v_j:

$$\begin{aligned} u_j &\equiv u_j(t) \equiv u(x_j,t) && (15) \\ v_j &\equiv v_j(t) \equiv v(x_j,t) && (16) \end{aligned}$$

Using the usual centred, $O(h^2)$ approximation for the second spatial derivative,

$$u_{xx}(x_j) = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} + O(h^2) \tag{17}$$

eqs. (9) and (10) become a set of $2(N-2)$ coupled ODEs for the $2(N-2)$ unknowns $u_j(t)$ and $v_j(t)$, $j = 2, \cdots N-1$:

$$\begin{aligned} \frac{du_j}{dt} &= v_j \qquad\qquad j = 2, \cdots N-1 && (18) \\ \frac{dv_j}{dt} &= \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} \qquad j = 2, \cdots N-1 && (19) \end{aligned}$$

We can implement the Dirichlet boundary conditions as follows: if the boundary conditions are satisfied at the initial time, $t = 0$, then they will be satisfied at all future times provided that the time derivatives of u and v vanish at the boundaries. Using this observation, we can now write down a complete set of $2N$ coupled ODEs in the $2N$ unknowns $u_j(t)$ and $v_j(t)$ which can then be solved using `LSODA`:

$$\begin{aligned} \frac{du_1}{dt} &= 0 && (20) \\ \frac{du_j}{dt} &= v_j \qquad\qquad j = 2, \cdots N-1 && (21) \\ \frac{du_N}{dt} &= 0 && (22) \end{aligned}$$

$$\begin{aligned} \frac{dv_1}{dt} &= 0 && (23) \\ \frac{dv_j}{dt} &= \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} \qquad j = 2, \cdots N-1 && (24) \\ \frac{dv_N}{dt} &= 0 && (25) \end{aligned}$$

This solution procedure is implemented by the the program `wave` (See ~`phys410/ode/wave`). You will follow an analogous approach to solve the *diffusion equation* in the final homework.

PHYS 410/555 Computational Physics

Solving Two-Point Boundary Value Problems Using "Shooting"
A Toy Model for the Deuteron

Recall that, by definition, *two-point boundary value problems* (BVPs), are ODE's for which boundary conditions are supplied at two distinct points—typically the end points of the solution domain—rather than at some single point, as in the case of initial value problems (IVPs).

In addition, we observe that two-point BVPs are often (but not always) *eigenvalue problems*, that is, BVPs are often characterized by one or more parameters such that only for specific parameter values (eigenvalues of the problem) will solutions satisfying the boundary conditions exist. In such a case, the solution of the BVP also becomes a problem in *search*—in general we will *not* be able to construct an algorithm which in "one go" results in the eigenvalue and the associated *eigenfunction.* Rather, we will have to provide some initial estimate (guess) of the eigenvalue, and then successively refine it according to some criterion, until we have computed the eigensolution to some acceptable accuracy. Also, solutions of eigenvalue problems are generally *not* unique; typically an eigenvalue problem admits a countable infinity of eigenfunctions, each with an associated eigenvalue (the eigenvalues are often, but not always, distinct).

The technique we will briefly consider here is known as *shooting.* Shooting is based on the observation that any two-point BVP can also be solved as an IVP. Consider, for example, a BVP of the form:

$$u''(x) = f(u, u', x) \qquad 0 \le x \le 1 \tag{1}$$

subject to the boundary conditions

$$u(0) = u_0 \tag{2}$$
$$u(1) = u_1 \tag{3}$$

where u_0 and u_1 are specified constants. Assume that we have somehow determined a function $U(x)$ which satisfies (1), (2) and (3). We then note that if we now consider the *initial value problem* defined by (1) subject to the *initial conditions*:

$$u(0) = U(0) = u_0 \tag{4}$$
$$u'(0) = U'(0) \tag{5}$$

then its solution must also be $U(x)$. That is, we can solve the BVP as an IVP, by "guessing" what value of U' we must specify at $x = 0$ so that when we are done integrating from $x = 0$ to $x = 1$, we have $u(1) = u_1$. The term "shooting" comes from analogy with the problem of setting the elevation, θ, of an artillery gun (i.e. θ is the angle the gun's barrel makes with the horizontal), so that the shell hits a target at some given range. Assuming that $\theta < \pi/4$, if the elevation if too low/high, the shell will fall short/long of the target respectively. The gunner can use information about where his/her current shot lands to adjust θ so that the next shot comes closer to the target.

Similarly, when integrating a BVP such as (1) via shooting, we will typically find that if we specify $u'(0) > U'(0)$ then we will have $u(1) > u_1$, while if we choose $u'(0) < U'(0)$, then we will find $u(1) < u_1$ (equally as likely is that $u'(0) > U'(0) \longrightarrow u(1) < u_1$ and $u'(0) < U'(0) \longrightarrow u(1) > u_1$). As long as we can find an initial pair of "bracketing" values, $[u'_-(0), u'_+(0)]$, such that separate integrations of (1) with these two values leads to values of $u(1)$ which similarly bracket the desired boundary value, u_1, then we can narrow the bracket using, for example, the technique of bisection search described below, to determine $U'(0)$, and hence the solution of the BVP, to whatever precision is desired.

We note that it is not always an initial value *per se* which we tune in a shooting problem. For some second order BVPs (such as the one considered below), we can deduce a second initial value at one of the boundary points (in addition to the one given by the boundary conditions) from mathematical or physical considerations, but, as discussed above, there is a parameter, λ, in the specification of the BVP which must have certain values in order that the boundary condition at the other boundary is satisfied. The idea is still the same; we look for an initial bracket $[\lambda_-, \lambda_+]$ such that separate integration with the two parameter values gives end-point solution values which are too small and too large respectively. We can then refine the bracket until we determine the eigenvalue and eigenfunction to the required precision.

We now illustrate this technique by using it to solve the toy model for a deuteron which is discussed in Chapter 9 of *Mathematical Methods for Physicists* by Arfken. A deuteron, as most of you probably know, is a bound state of a proton and a neutron (the nucleus of deuterium, an isotope of hydrogen). The model is *highly* idealized; we assume that the deuteron wave function, $\psi(\vec{r})$, is a *spherically symmetric* solution of the time-independent Schrödinger equation, with the proton-neutron interaction described by a square wave potential.

Thus we wish to solve

$$-\frac{\hbar^2}{2M}\nabla^2\psi(\vec{r}) + V(\vec{r})\psi(\vec{r}) = E\psi(\vec{r}) \tag{6}$$

where M is the deuteron "mass", and E is the energy eigenvalue which will shortly become the "shooting parameter" in our BVP solution of the model.

From the assumption of spherical symmetry, we have

$$\psi(\vec{r}) \to \psi(r) \tag{7}$$

and

$$\nabla^2\psi(r) = \frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\psi}{dr}\right) \tag{8}$$

Further, defining

$$u(r) \equiv r\psi(r) \tag{9}$$

we have (as you should verify)

$$\nabla^2\psi(r) \to \frac{1}{r}\frac{d^2u(r)}{dr^2} \tag{10}$$

Thus, (6) can be rewritten:

$$\frac{d^2u(r)}{dr^2} + \frac{2M}{\hbar^2}\left(E - V(r)\right)u(r) = 0 \tag{11}$$

As noted above, we will model the proton-neutron interaction as a finite-range square potential. Thus, we take

$$V(r) = V_0 \qquad 0 \le r \le a \tag{12}$$

$$V(r) = 0 \qquad r > a \tag{13}$$

where V_0 is a negative constant, so $|V_0|$ is the depth of the potential well, while a is its width.

As is the case for any solution of the Schrödinger equation, we must demand that our solution of (6) be normalizable, i.e. that

$$\int \psi\psi^* dV = 1 \tag{14}$$

so that there is unit probability that our deuteron is found *somewhere* in the universe. In the current spherically symmetric case this means that we must have

$$4\pi \int_0^\infty r^2 \psi(r)^2 dr = 4\pi \int_0^\infty u(r)^2 dr = 1 \tag{15}$$

Clearly, a necessary condition for normalizability is that

$$\lim_{r\to\infty} u(r) = 0 \tag{16}$$

and this, in fact, is one of the boundary conditions for our ODE. Furthermore, we assert that for fixed values of the parameters of the model (a, M and V_0), a normalizable solution will only exist for certain discrete values of E—the eigenvalues of our Schrödinger equation.

Before we consider the solution of (11) using shooting, we rewrite the equation in an equivalent form in which the minimum number of free parameters (which, if desired, can be made explicitly dimensionless) becomes evident. To this end, we define a rescaled radial coordinate, x

$$x \equiv \sqrt{2M} r \tag{17}$$

so that, among other things, we have

$$\frac{d^2u}{dr^2} \to 2M \frac{d^2u}{dx^2} \tag{18}$$

Further, we choose units such that $\hbar = 1$ and $V_0 = -1$ (you should establish that this *is* always possible if it isn't immediately obvious to you).

With these choices, we are left with one free parameter—the width, a, of the potential well. Given that we have adopted the rescaled radial coordinate x, it is more convenient to use x_0, defined by

$$x_0 \equiv \sqrt{2M} a \tag{19}$$

as the free parameter.

Thus, our Schrödinger equation (11) becomes

$$\frac{d^2u(x)}{dx^2} + (E - V(x))\, u(x) = 0 \tag{20}$$

where

$$V(x) = -1 \qquad 0 \le x \le x_0 \tag{21}$$

$$V(x) = 0 \qquad x > x_0 \tag{22}$$

and where we again note that $E = E(x_0)$ is an *eigenvalue* of (20); i.e., for a given value of x_0, only for discrete values of E will we have a normalizable wave function.

The boundary conditions for (20) are derived from the demands that

1. $\psi(r)$ be regular (analytic) at $r = 0$.
2. $\lim_{r\to\infty} u(r) = 0$.

The regularity condition at $r = 0$ means that $\psi(r)$ admits an expansion

$$\lim_{r\to 0} \psi(r) = \psi_0 + r^2\psi_2 + O(r^4) \tag{23}$$

where the ψ_i are constants (the power series expansion cannot have terms which are odd in r, since $\psi(r)$ would not have a well-defined derivative at $r = 0$ in that case). From this, it follows that $u(r)$ has an expansion

$$\lim_{r \to 0} u(r) = r\psi(r) = r\psi_0 + r^3\psi_2 + O(r^5) \tag{24}$$

Thus, we have

$$\begin{aligned} u(0) &= 0 && (25) \\ \frac{du}{dr}(0) &= \psi_0 && (26) \end{aligned}$$

We now observe that (6) (like *all* Schrödinger equations) is *linear*; given any solution, $\psi(r)$ we have that $c\psi(r)$, where c is an arbitrary positive constant, is also a solution. The *particular* solution we seek is fixed by the normalization condition:

$$\int \psi\psi^* dV = 1 \tag{27}$$

Operationally, this means that we can choose $\psi(0)$ *arbitrarily* (say $\psi(0) = 1$ for convenience), then, for specified x_0, vary E until we find a solution which satisfies

$$\lim_{r \to \infty} u(r) = \lim_{r \to \infty} r\psi(r) = 0 \tag{28}$$

(In other words, the eigenvalue is independent of the normalization of the eigenfunction).

In preparation for a solution of our problem using `LSODA` we rewrite (20) in canonical first order form by introducing

$$w(x) \equiv \frac{du(x)}{dx} \tag{29}$$

We then have

$$\begin{aligned} \frac{du(x)}{dx} &= w(x) && (30) \\ \frac{dw(x)}{dx} &= (V(x) - E)u(x) && (31) \end{aligned}$$

subject to

$$\begin{aligned} u(0) &= 0 && (32) \\ w(0) &= 1 && (33) \end{aligned}$$

and with E to be determined so that

$$\lim_{x \to \infty} u(x) = 0 \tag{34}$$

We will now assume that for any given value of x_0, we are able to determine values E_- and E_+ (perhaps by trial-and-error) such that

$$\lim_{x \to \infty} u(x; E_-) = -\infty \tag{35}$$

$$\lim_{x \to \infty} u(x; E_+) = +\infty \tag{36}$$

where the notation $u(x; E_i)$ means the trial solution $u(x)$ computed using eigenvalue estimate E_i. We further assume that properties (35) and (36) hold for *any* values $E_{\rm HI}$ and $E_{\rm LO}$ that bracket the desired eigenvalue, namely:

$$\lim_{x \to \infty} u(x; E_{\rm LO}) = -\infty \tag{37}$$

$$\lim_{x \to \infty} u(x; E_{\rm HI}) = +\infty \tag{38}$$

with either

$$E_{\rm LO} < E < E_{\rm HI} \tag{39}$$

or

$$E_{\rm HI} < E < E_{\rm LO} \tag{40}$$

Given the initial bracket $[E_-, E_+]$, then, we can compute the desired eigenvalue, E, accurate to some desired tolerance δ, using a *bisection search* (also known as a *binary search*). Here is a typical implementation of a bisection search written in pseudo-code:

```
$E_{\rm LO} := E_-$
$E_{\rm HI} := E_+$
while $|E_{\rm HI} - E_{\rm LO}| > \delta$ do
    $E_{\rm MID} := (E_{\rm HI} + E_{\rm LO})/2$
    if $u(x; E_{\rm MID}) \to +\infty$ as $x \to \infty$ then
        $E_{\rm HI} := E_{\rm MID}$
    else
        $E_{\rm LO} := E_{\rm MID}$
    end if
end while
$E := E_{\rm MID}$
```

We note that the convergence rate of the bisection search is completely pre-determined; the size of the bracketing interval after n bisections is

$$\frac{1}{2^n}\,(E_+ - E_-) \tag{41}$$

The solution of equations (30)-(33) using `LSODA` is implemented as the program `deut.f`; the bisection search for the eigenvalues $E(x_0)$ is implemented via the shell-script `Shoot-deut`, which itself uses the following collection of scripts which can be used to implement shell-level bisection searches:

```
bsnew <lo> <hi>      # Initializes a new search
bscurr               # Returns the current (mid) value for the search
bslo                 # Replaces the low bracket value with ‘bscurr‘
bshi                 # Replaces the high bracket value with ‘bscurr‘
bsdone [<tol>]       # Returns completion code 0 if search bracket
                     #   has been narrowed to a relative precision <tol>
                     #   (which defaults to 10(-14), returns completion
                     #   code 1 otherwise
bsnotdone [<tol>]    # Logical negation of bsdone.
```

In practice, of course, we do not (can not!) integrate all the way to $x = \infty$, instead, we integrate to some finite $x = x_{\rm max}$ where $x_{\rm max}$ is chosen sufficiently large so that, for the specfied convergence criterion δ, we can determine for all possible E whether the solution $u(x; E)$ is diverging to $+\infty$ or to $-\infty$ at $x = x_{\rm max}$.

Finally, as with many of the problems we have discussed in this course, the toy deuteron problem can be solved "analytically"—however, as in the cases of those other exactly soluble problems we have discussed, the numerical technique which we use to approximately solve this BVP can be extended very easily to solve entire classes of problems which are *not* amenable to exact solution.