PHY/CAM 381C: Computational Physics   Homework 3
Due: Tuesday, March 18, 12:30pm

Important: *This assignment requires that you write four* `f77` *programs which involve the solution of ODEs and linear systems. As always, please follow the instructions carefully, particularly with regards to command-line arguments, standard input and standard output. Failure to implement programs which abide by the specifications given below will adversely affect your grade for the assignment.*

Problem 1: Modify the toy deuteron program `deut.f` discussed in class so that it outputs a *normalized* wave function $u_n(x)$. Specifically, in the directory `~/hw3/a1`, write a Fortran program `deutn.f`, with corresponding executable `deutn`, which has the same usage as `deut`:

```
usage: deutn <x_0> <E> [<tol>]
```

where <x_0> is the range of the potential, <E> is the trial eigenvalue, and <tol> is an optional tolerance for `LSODA`, which should default to 1.0×10^{-8}. Like `deut`, `deutn` should read requested output x-values, x_i, from standard input (one value per line). However, rather than outputting x_i, $u(x_i)$ and $u'(x_i)$, `deutn` should output *only* x_i, and $u_n(x_i)$ (two numbers per line) to standard output. $u_n(x)$ is to be computed so that

$$\int_0^\infty |u_n(x)|^2 \, dx = 1$$

Note that this is *not* equivalent to the true normalization condition

$$\int |\psi(r)|^2 dV = 1$$

but that it will ensure proper *relative* normalization between wave-functions computed for various values of x_0. Using the default LSODA tolerance (1.0×10^{-8}), compute eigenvalues (to about 7 significant digits), and corresponding normalized wave functions for

$$x_0 = 2.0, 3.0, 4.0 \text{ and } 8.0$$

Record your eigenvalues in `~/hw3/a1/README` and make a single postscript plot called `~/hw3/a1/deutn.ps` which shows all 4 computed eigenfunctions on the domain $0 \le x \le 20$. *Note:* You may find that you have to tune the values for E_n to considerably more than 7 or 8 digits *precision* in order to get a solution which behaves properly throughout the domain. Note that this does *not* mean that the computed eigenavalue is *accurate* to that number of digits. You should record both the *exact* (i.e to 16 digits) values of E_n which you used to make your plots, as well as values reflective of the likely accuracy in your computations in `README`.

Problem 2: Consider the differential equation $(' \equiv d/dx)$

$$\left(1 - x^2\right) \, C(x)'' - 1.628 \, x \, C(x)' + \lambda \, C(x) = 0 \tag{2.1}$$

on the interval

$$-1 \le x \le 1$$

with boundary conditions

$$C(-1) = 1 \qquad C(1) = \pm 1 \tag{2.2}$$

(i.e. C(1) is *either* +1 or -1, depending on the particular solution). Here, λ (which is a real constant) is to be viewed as an eigenvalue: solutions $C_n(x)$ of (2.1) satisfying the boundary conditions (2.2) will exist only for discrete values λ_n, where n is an integer which labels the eigenvalues and eigenfunctions, but which is also equal to the number of times $C(x)$ changes sign on the solution domain. In `~/hw3/a2`, write a `f77` program called `ortho` (source code `ortho.f`) which uses `LSODA` to solve this eigenvalue problem. `ortho` must have usage:

```
usage: ortho <lambda> [<tol>]
```

where <lambda> is a trial eigenvalue and <tol> is an optional tolerance for LSODA, which should default to 1.0×10^{-8}. ortho should read requested output x-values from standard input (one per line), and should output x_i, $C(x_i)$ (two numbers per line) to standard output. Use your program to compute

$$\lambda_n \text{ and } C_n(x) \text{ for } n = 1, 2, 3, 4, 5$$

The λ_n should be computed to about 7 significant digits. Record your computed λ_n in ~/hw3/a2/README and save your computed $C_n(x)$ (i.e. the standard output from ortho) in files c1, c2, c3, c4 and c5 and in ~/hw3/a2. Make a single postscript plot called ~/hw3/a1/allc.ps which shows all 5 eigenfunctions on the solution domain. *Minor hint:* You may find it convenient to ignore a return code (LSODA parameter istate) of -1 when integrating on the last interval (i.e. the interval containing $x = -1.0$.

Problem 3: *Gravitational n-body simulation.* In the directory ~/hw3/a3 write a well-commented f77 program nbody (source code nbody.f) which uses LSODA to integrate the equations of motion for n particles interacting via the Newtonian gravitational force. Specifically, consider n particles with masses

$$m_i \qquad i = 1, 2, \ldots n$$

and position vectors

$$\mathbf{r}_i(t) \equiv [x_i(t), y_i(t), z_i(t)] \qquad i = 1, 2, \ldots n$$

Then denoting differentiation with respect to time, t, by an overdot, the equations of motion are:

$$m_i \ddot{\mathbf{r}}_i = -G \sum_{j=1,\, j\neq i}^{n} \frac{m_i m_j}{r_{ij}^3} \mathbf{r}_{ij} \qquad i = 1, 2, \ldots n$$

where

$$\mathbf{r}_{ij} \equiv \mathbf{r}_i - \mathbf{r}_j \qquad r_{ij} = |\mathbf{r}_{ij}| = (\mathbf{r}_{ij} \cdot \mathbf{r}_{ij})^{1/2}$$

and G is Newton's gravitational constant which is to be set equal to unity in this calculation.

nbody must have the following usage:

```
usage: nbody <t final> <dt out> [<tol> <trace>]
```

where <t final> is the final integration time (the integration is assumed to start at $t = 0$), <dt out> is the output interval, <tol> is the LSODA tolerance which should default to 1.0×10^{-6}, and <trace>, which if present on the command line, enables tracing output to standard error as described below.

nbody must accept the following input from standard in:

```
m_1  x0_1  y0_1  z0_1  vx0_1  vy0_1  vz0_1
m_2  x0_2  y0_2  z0_2  vx0_2  vy0_2  vz0_2
                   .
                   .
                   .
m_n  x0_n  y0_n  z0_n  vx0_n  vy0_n  vz0_n
```

(seven numbers per line), where, for example, m_1, x0_1 and vx0_1 denote the mass, initial x-coordinate and initial x-component of the velocity, respectively, of the first particle. nbody must produce the following output on standard out:

```
n
m_1
m_2
  .
  .
  .
```

```
m_n
tout_0
x_1(tout_0)   y_1(tout_0)  z_1(tout_0)
x_2(tout_0)   y_2(tout_0)  z_2(tout_0)
                 .
                 .
                 .
x_n(tout_0)   y_n(tout_0)  z_n(tout_0)
tout_1
x_1(tout_1)   y_1(tout_1)  z_1(tout_1)
x_2(tout_1)   y_2(tout_1)  z_2(tout_1)
                 .
                 .
                 .
x_n(tout_1)   y_n(tout_1)  z_n(tout_1)

                 .
                 .
                 .

tout_nt
x_1(tout_nt)  y_1(tout_nt) z_1(tout_nt)
x_2(tout_nt)  y_2(tout_nt) z_2(tout_nt)
                 .
                 .
                 .
x_n(tout_nt)  y_n(tout_nt) z_n(tout_nt)
```

where the `tout_i, i = 1 ... nt` are the output times `0, dt_out, 2 dt_out, ... nt dt_out`, and the number of output times, `nt`, can be computed from the final integration time and the output interval. The line structure implied by the above schematic is important; make sure your program adheres to it. Your program can restrict the number of particles which can be integrated, but should *gracefully* handle input which specifies more than that number.

If output tracing is enabled (i.e. if the `<trace>` argument appears on the command-line), your program should produce, *on standard error*, the following output at each output time (including the initial time):

```
t   x_com   y_com   z_com   E_tot   KE_tot   PE_tot   P_tot   J_tot
```

Where `t` is the output time, `x_com`, `x_com` and `y_com` are the coordinates of the center of mass of the particle system:

$$\mathbf{r}_{\rm com} = [x_{\rm com}, y_{\rm com}, z_{\rm com}] = \frac{\sum_i m_i\,\mathbf{r}_i}{\sum_i m_i}$$

`E_tot` is the total mechanical energy of the system:

$$E_{\rm tot} = KE_{\rm tot} + PE_{\rm tot},$$

`KE_tot` is the total kinetic energy:

$$KE_{\rm tot} = \sum_{i=1}^{n} \frac{1}{2} m_i\, {v_i}^2$$

`PE_tot` is the total kinetic energy:

$$PE_{\rm tot} = -G \sum_{i=1}^{n} \sum_{j=1,\, j<i}^{n} \frac{m_i\, m_j}{r_{ij}}$$

P_tot is the magnitude of the total linear momentum:

$$P_{\rm tot} = |\mathbf{P}_{\rm tot}| \qquad \mathbf{P}_{\rm tot} = \sum_i \mathbf{p}_i = \sum_i m_i \mathbf{v}_i$$

and J_tot is the magnitude of the total angular momentum computed about the center of mass

$$J_{\rm tot} = |\mathbf{J}_{\rm tot}| \qquad \mathbf{J}_{\rm tot} = \sum_i (\mathbf{r}_i - \mathbf{r}_{\rm com}) \times \mathbf{p}_i$$

Make sure that all 9 numbers appear on a single line of the standard error: use a `format` statement such as

```
2000     format(1P,10E25.16)
```

Note that in the `csh`, you can redirect standard input and standard error to separate output files using the construct

```
% (command > stdoutfile) >& stderrfile
```

Although your implementation of `nbody` should treat particle motion in all three dimensions, you may find it convenient and instructive to consider the case of motion in the xy plane by using initial data with `z_i = 0`, `vx_i = 0`. For these cases you may find the instructor-supplied program `pp2d` useful for visualizing your calculations. For information on `pp2d` see the Course-Related Software Web page or type `pp2d -h` at a command prompt on one of the SGIs. Note that `pp2d` is `X`-based so that you can run `einstein` while visualizing your simulation on a remote workstation (running `X`) or X-terminal. Provided your `nbody` produces output as described above, and that the `DISPLAY` environment variable is set properly, you should be able to simply use a construct like

```
% nbody 2.0 0.001 < input | pp2d
```

Let me know if you have problems with `pp2d`, or if you have suggestions for improvements.

Once you have your program de-bugged and tested, there are an abundance of calculations you can perform with it. At a minimum, find initial conditions which describe:

- A stable orbit of two bodies with masses $m_1 = 9.0$ and $m_2 = 1.0$, respectively, with $P_{\rm tot} = 0$, and a separation of roughly 2.0. Leave your initial conditions in the file `~/hw3/a3/orbit`
- Initial data as described above, but with a third particle with mass $m_3 = 0.001$, which is in as large a (roughly) circular orbit as possible about m_2. As your critertion for stability, demand that m_1 and m_2 make at least 10 mutual orbits before the orbit of m_3 is disrupted. Again, motion should be restricted to the xy plane. Leave your initial conditions in the file `~/hw3/a3/satellite`, and document how you found the conditions in `~/hw3/a3/README`.

Feel free to investigate and document evolutions of your own choice, particularly with several particles. Report any particularly interesting findings in `~/hw3/a3/README`. I will further test and evaluate your program with my own input.

Problem 4: In the directory `~/hw3/a4`, write a Fortran program `nlbvp1d` (source code in `nlbvp1d.f`), which solves the following non-linear boundary value:

$$u_{xx} + (u u_x)^2 + \sin(u) = f(x) \qquad 0 \le x \le 1 \qquad \text{with } \; u(0) = u(1) = 0.$$

where $u \equiv u(x)$, and $f(x)$ is a specified function. Your program should use finite-difference techniques, Newton's method for non-linear systems and the `LAPACK` tridiagonal solver `dgtsv.f`. For the first and second derivatives, use the finite-difference approximations which were discussed in class. `nlbvp1d` must have the following usage:

```
usage: nlbvp1d <level> <guess_factor> [<option> <tol>]

        Specify option .ne. 0 for output
        of error instead of solution
```

The required `integer` argument, `level`, and optional `integer` argument, `option`, have the same interpretation and default value (for `option`) as in the `bvp1d` example discussed in class. The required `real*8` argument, `guess_factor`, is used to initialize the Newton iteration as described below, and `tol`, which should default to `1.0d-8`, specifies a convergence criteria for the Newton iteration. Iteration should continue until

$$\frac{\|\delta \mathbf{u}^{(n)}\|_2}{\|\mathbf{u}^{(n)}\|_2} \leq \texttt{tol}$$

where $\| \cdots \|_2$ denotes the ℓ_2 norm of a vector as defined in class. Test your program by taking

$$u(x) \equiv u_{\rm exact} = \sin(4\pi x),$$

computing what $f(x)$ must be so that the differential equation is satisfied, and supplying the appropriate values of $f(x)$ to your program. Initialize the Newton iteration by setting

```
u(i) = guess_factor * uexact(i)
```

Important: There are at least *three* distinct solutions of the differential equation given the right hand side $f(x)$ implicitly defined by the above choice of $u_{\rm exact}$. In order for the Newton method to converge to $u_{\rm exact}$, you will have to specify a value of `guess_factor` close to 1.0: in fact, I recommend that you use `guess_factor = 1.0` until you are sure that you have convergence, both of the Newton's method, and of the difference solution to $u_{\rm exact}$. Once you are confident that your difference solution is converging to $u_{\rm exact}$, make postscript plots showing (A) the level 6 numerical solution and the exact solution as function of x (`soln6.ps`) and (B) the error for level 5, 6, and 7 solutions, also as a function of x (`err567.ps`). Using different values of `guess_factor`, try to find at least two other solutions of the boundary value problem (keeping $f(x)$ fixed). Make a single postscript plot (`allsolns.ps`) showing all the solutions which you are able to find (computed at level 6). Note: All postscript plots should reside in the directory `~/hw3/a4`.