Phy 329: Computational Physics Homework 6 Due: Monday, November 18, 3:00pm

The following assignment involves writing and testing three Fortran 77 programs which solve non-linear equations. Do all development and execution on `einstein`*. As usual, all files required by the assignment must reside in the correct places on your SGI account for the homework to be considered complete. Contact me immediately if you are having undue difficulties with any part of the homework.*

Problem 1: In directory `~/hw6/a1`, write a Fortran program called `mybisect.f`, with corresponding executable `mybisect`, which finds a root of

$$\sin(x/7) = \cos(7x)$$

using bisection. `mybisect` should accept exactly two `real*8` arguments:

```
usage: mybisect <xmin> <xmax>
```

which specify an initial bracket for the search, and should write the located root (one number) to standard output. Make sure the root is accurate to at least 8 digits. You are free to work from source code covered in class and available on the `phy329` account on `einstein`. Test your routine by finding a root in the interval $0.0 < x < 0.5$. How many roots can you find in the interval $-2.0 < x < 2.0$? Answer in `~/hw6/a1/README`. *Hint:* you may find it useful to plot $\sin(x/7) - \cos(7x)$ (use `Maple` or `gnuplot`) in order to locate brackets for the roots.

Problem 2: In directory `~/hw6/a2`, write a Fortran program `newt3` (source code in `newt3.f`), which finds a root of the following system using Newton's method in 3-dimensions:

$$\begin{aligned} x^2 + y^3 + z^4 &= 1 \\ \sin(xyz) &= x + y + z \\ x &= yz \end{aligned}$$

`newt3` should accept 3 or 4 arguments:

```
usage: newt3 <x0> <y0> <z0> [<tol>]
```

where `x0`, `y0` and `z0` are the initial guesses for x, y, and z respectively, and `tol` is an optional convergence criteria which should default to `1.0d-8`. Implement the test for convergence following the `newt2` example covered in class. Like `newt2`, your program should use the `LAPACK` routine, `dgesv`, to solve the linear system arising in the Newton iteration. Your program should trace the Newton iteration to standard error (again, as `newt2` does), mostly to aid you in determining when you have implemented the algorithm correctly. The only output to standard output should be the final estimate of the root (three numbers, x, y, z, on one line). Test your program by finding a root near $x = 3.0$, $y = -2.0$, $z = -1.0$ and recording what you find in `~/hw6/a2/README`.

Problem 3: In directory `~/hw6/a3`, write a Fortran program `nlbvp1d` (source code in `nlbvp1d.f`), which solves the following non-linear boundary value problem discussed in class:

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

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`. Use the finite-difference approximation which was discussed in class. (Also note that the discretization technique and $O(h^2)$ approximations of the first and second derivatives are the same as those used in Problem 2 of Homework 4 (H4.2)). `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 H4.2. 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

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

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_{exact}. In order for the Newton method to converge to u_{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_{exact}. Once you are confident that your difference solution is converging to u_{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).