PHYS 410/555: Computational Physics Fall 2003 Homework 4
DUE: Tuesday, November 18, 10:00 AM Report bugs to choptuik@physics.ubc.ca

The following assignment involves writing and testing three Fortran 77 programs which use LAPACK routines to solve various linear systems. You must also prepare a short `Maple` *worksheet for Problem 3. Follow the instructions carefully and do all program development on the* `lnx` *machines. All files required by the assignment must reside in the correct places on your* `lnx` *account for the homework to be considered complete. Note that you are free—and in fact encouraged—to work from the relevant examples covered in class.*

Note: *This assignment is identical for both 410 and 555 students; i.e. all students must complete all three problems.*

Problem 1: In directory `~/hw4/a1` on your `lnx` account, create a source file `lsolve.f` and corresponding executable `lsolve` which solves a general linear system. `lsolve` is to accept exactly two arguments:

```
usage: lsolve <matrix file> <rhs file>
```

where `<matrix file>` is the name of a file which contains an $n \times n$ matrix (A) and `<rhs file>` is the name of a file which contains an n–element column vector (b). `lsolve` is to read the matrix and the right-hand-side vector, then compute the solution, x of the linear system

$$A\,x = b$$

using the `LAPACK` routine `DGESV`. If the solution is successful, the solution vector x is to be written to standard output, one number per line. Use the routines `dvfrom` and `dvto` (see Fortran Notes, Lecture 3) to read and write the vectors and `dmfrom` (Fortran Notes, Lecture 6) to read the matrix. It is probably best to include these routines in your `lsolve.f` source file. Recall that all source code covered in class is available in the `phys410` account on `lnx`. Also note that `dmfrom` reads array entries in standard Fortran order; i.e. column-by-column. Your implementation of `lsolve` should do a reasonable amount of error checking, including:

- verification that the input arguments refer to existing files
- verification that the input matrix, A, is square (i.e. $n \times n$ for some n)
- verification that the length of the rhs vector, b, is n, given that A is $n \times n$
- verification that sufficient storage exists to store A

If any of the above conditions are not satisfied, `lsolve` should print an error-message to standard error, then exit. Note that you need not test for file-existence in the main program since both `dvfrom` and `dmfrom` return information which can be used to deduce that the specified file did not exist. Be sure to test your program using, for example, a small–n problem such as that covered in Lecture 1 of the Linear Systems Notes, as well as various types of *invalid* input.

Hint: If your solution to this problem starts to get complicated, particularly in terms of how the matrix and vectors are dealt with, you should assume that there *is* an easier way!

Problem 2: Consider the following boundary value problem (BVP)

$$u''(x) + a(x)\,u'(x) + b(x)\,u(x) = f(x) \qquad 0 \le x \le 1 \qquad u(0), u(1) \;\text{ specified}$$

where $'$ denotes differentiation with respect to x, and $a(x)$, $b(x)$ and $f(x)$ are given functions. In directory `~/hw4/a2` on your `lnx` account, create a source file `gbvp1d.f` and corresponding executable `gbvp1d` which approximately solves this problem using second-order ($O(h^2)$) finite-difference methods. Specifically, discretize the continuum domain using a uniform mesh, $x_j, j = 1, \ldots, N$ with mesh spacing $h = 1/(N-1)$ (as discussed in class) and use the following approximations for the first and second derivatives;

$$\frac{u_{j+1} - u_{j-1}}{2h} = u'(x_j) + O(h^2)$$

$$\frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} = u''(x_j) + O(h^2)$$

Your program needs to first set up the tridiagonal system which results from discretizing the BVP using the above approximations and then solve the system using the LAPACK routine DGTSV. Test your implementation by taking:

$$u(x) \equiv u_{\text{exact}}(x) = \sin(4\pi x) \qquad a(x) = 16x^2 \qquad b(x) = 8x^3$$

then computing what $f(x)$ must be to satisfy the ODE. Your code should thus probably include a routine which calculates the appropriate values for a, b, f and u_{exact} on the finite-difference mesh x_j. Your program should have precisely the same usage and output behaviour as the example `bvp1d` covered in class. In particular, usage for `gbvp1d` should be

```
usage: gbvp1d <level> [<option>]

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

where the required integer argument, `<level>`, specifies the discretization level, l, such that $h = 1/2^l$ and $N = 2^l + 1$. If `<option>` is 0, or not specified, the standard output of `gbvp1d` is (x_j, u_j) (two numbers per line), $j = 1, ...N$, where u_j are the components of the computed solution. If `<option>` is non-zero, the standard output of `gbvp1d` is (x_j, e_j) (two numbers per line), $j = 1, ...N$, where $e = u_{\text{exact}} - u$ is the error in the computed solution. In all cases, the program should also compute the RMS error of the computed solution and output it to standard error. Using `gnuplot` or `supermongo`, make postscript plots showing (A) the level 6 solution and the exact solution as a function of x (`soln6.ps`) and (B) the error for level 5, 6 and 7 solutions, also as a function of x (`err567.ps`). (The postscript files should, of course, be located in the directory ~`/hw4/a2`.)

Problem 3: Consider the BVP problem defined in the previous question and, in ~`/hw4/a3` on your `lnx` account, create a source file `gbvp1d4.f` and executable `gbvp1d4` which approximately solves the ODE using a combination of $O(h^4)$ and $O(h^2)$ finite-difference approximations of the first and second derivatives. Specifically, use

$$\frac{-u_{j+2} + 8u_{j+1} - 8u_{j-1} + u_{j-2}}{12h} = u'(x_j) + O(h^4)$$

$$\frac{-u_{j+2} + 16u_{j+1} - 30u_j + 16u_{j-1} - u_{j-2}}{12h^2} = u''(x_j) + O(h^4)$$

for $j = 3, \ldots, N-2$, and the second-order approximations given previously for $j = 2$ and $j = N-1$. Before implementing the program, use `xmaple` to *verify* that the above difference formulae really *do* provide $O(h^4)$ approximations to $u'(x)$ and $u''(x)$. *Hint:* Consider `maple` expressions such as

```
> convert(taylor(u(x+h),h,8),polynom);
> convert(taylor(u(x+2*h),h,8),polynom);
```

Document your verification in a `maple` worksheet called ~`/hw4/a3/fd4.mws`. Your implementation of `gbvp1d4` must first set up the pentadiagonal system for the u_j, and then solve the linear system using the LAPACK banded-solver DGBSV. Use the same $a(x), b(x)$ and $u_{\text{exact}}(x)$ to test your program as you used in Problem 2. `gbvp1d4` *should accept precisely the same arguments and have precisely the same output behaviour as* `gbvp1d`. Using `gnuplot` or `supermongo`, make postscript plots showing (A) the level 6 solution and the exact solution, as a function of x (`soln6.ps`) and (B) the error for level 5, 6 and 7 solutions, also as a function of x (`err567.ps`).

Note: You may find this problem significantly more difficult to complete than the other two. Don't leave it until the last minute and don't be afraid to ask for help and/or advice.