PHYS 410/555: Computational Physics   Fall 2005   Homework 4
DUE: December 1, 10:00 AM   Report bugs to choptuik@physics.ubc.ca

WARNING!!! *This is* BY FAR *the most challenging 410 assignment I have ever assigned, since it is, in fact, the base of a regular 410 term project.*

Why, you ask is it so challenging?

Fundamentally, since it requires the synthesis of everything that we're supposed to take out of the course, including the stuff that I have no hope in Hades of covering at anything less than supersonic speed, especially given the whole Thunderbirds thing from last week.

Specifically, to complete this assignment, and believe me, most or all of you *WILL* complete it, you must master the following hand-tooled techniques of numerical analyisis kung-foo-ery (you know, "when you can snatch the pebbles from my hand, Grasshopper, then you will be ready to leave")

1. Finite differencing of time dependent PDEs with quite arbitary spatial operators using uniformly $O(h^2)$ "Crank-Nicholson-type" schemes ($O(h^2)$ CN schemes, or sometimes simply "standard CN schemes in what follows).
2. The solution of non-linear banded systems, of the sort that result from application of standard schemes to equations such as the KdV equation using
   (a) Newton's method to solve *systems* of non-linear equations
   (b) The use of the LAPACK routine DGBSV to solve the banded (pentadiagonal, 5-diagonal) system that results from application of the standard CN scheme to the KdV equation

BUT WAIT, ISN'T MATT IN THIS HAPPY GO LUCKY LET'S DO XMAS YEAR ROUND KIND OF MOOD. *Indeed, this assignment is also unique in that you can choose to do it in groups of up to three, with only a single hand in, no questions asked–although I reserve the right to ask probing questions to test understanding on the part of any and all group members. Grading will take into account the composition of the groups, and believe me, that can work out more ways than you can start to imagine.*

Also note that this *was*, after all, only *half* of *one assigment* for the poor foobars that took Phys 381C back in Austin TX (yee hah) in 1997.

XXX, XXX, XXX, XXX and XXX you get the pleasure of doing the other (first) half of the project which is to strip down an FAS Multigrid code for nonlinear problems to produce an LCS scheme for linear problems. Stay tuned.
Problem 1 and only: Consider the Korteweg and de Vries (KdV) equation for $u \equiv u(x,t)$:

$$u_t + u_x + 12\,u\,u_x + u_{xxx} = 0 \qquad \text{on} \qquad -x_{\max} \le x \le x_{\max} \qquad 0 \le t \le t_{\max} \tag{2.1}$$

with initial and boundary conditions

$$u(x,0) = u_0(x) \qquad u(-x_{\max},t) = u(x_{\max},t) = 0 \tag{2.2}$$

This equation admits "wave-like" solutions (solitons) which propagate in *one* direction ($-x_{\max} \to x_{\max}$; i.e. "to the right"). The "vacuum" (or quiescent) state is $u = \kappa$, for an arbitrary real constant κ, which, without loss of generality, we can choose to be $\kappa = 0$. The *boundary* conditions (BCs) are thus compatible with quiescence, as should be the initial condition $u_0(x)$ (i.e. $u_0(x)$ should always satisfy—at least approximately—$u_0(-x_{\max}) = u_0(x_{\max}) = 0$). The right BC is *not* compatible with disturbances impinging on $x = x_{\max}$; thus, once any signal has reached $x = x_{\max}$, the "well-posedness" of the evolution is questionable, and you can expect "strange things" to happen. This problem could be remedied by working on a periodic domain (i.e. by identifying $-x_{\max}$ and $x_{\max}$), but this would also complicate the finite-difference solution of the equation. Furthermore, periodic boundary conditions are unnecessary, since by appropriate choice of $u_0(x)$ and $x_{\max}$, all of the interesting dynamics in the model can be studied on a finite spatial domain. Simply bear in mind that for any specific choice of initial data and $x_{\max}$, the amount of physical time,

$t_{\rm phys}$ for which the evolution can be meaningfully simulated will be finite, and thus $t_{\max}$ should normally be chosen (possibly empirically) so that $t_{\max} < t_{\rm phys}$.

Use an $O(h^2)$ "Crank-Nicholson" finite-difference scheme combined with a multi-dimensional Newton iteration to approximately solve (2.1). Implement your solution as a well-documented `f77` program `kdv` in `~/hw4/a2/` (source code `kdv.f`). `kdv` must have the following usage:

```
usage: kdv <xmax> <tmax> <level> <olevel> <dt/dx> <a> <x0> [<a> <x0> ...]
```

where

- `<xmax>` $\equiv x_{\max}$
- `<tmax>` $\equiv t_{\max}$
- `<level>` $\equiv$ discretization level. Number of spatial grid points, $\mathtt{nx} = 2^{\texttt{<level>}} + 1$
- `<olevel>` $\equiv$ output level. Output produced every $2^{\texttt{<level>}-\texttt{<olevel>}}$ time steps (see below).
- `<dt/dx>` $\equiv$ "Courant number". Ratio of time step $\triangle t$ to mesh spacing $\triangle x = h$. I recommend that you use `<dt/dx>` ≤ 0.5, and you may find that even smaller values are required (for stability) for high-amplitude pulses.
- `<a> <x0> [<a> <x0> ...]`. Initial data parameters: $a_i, x_{0i},\ i = 1, \cdots n_{\rm p}$ (interpretation described below). Note that these parameters come in pairs (all but the first pair are optional), and that your program may assume that at most 20 pairs will be specified on the command line.

Initial data: `kdv` must set initial data as follows:

$$u_0(x) = \sum_{i=1}^{n_{\rm p}} \Pi\left(x;\, a_i, x_{0i}\right) \qquad \text{where} \qquad \Pi\left(x;\, a, x_0\right) = \frac{1}{4} a^2 \cosh^{-2}\left[\frac{1}{2} a\left(x - x_0\right)\right] \tag{2.3}$$

Note that $\Pi\left(x;\, a, x_0\right)$ is a "pulse" profile whose maximum amplitude scales with a and which is centred at $x = x_0$. Thus, (2.3) generically represents the superposition of $n_{\rm p}$ separate pulses.

Finite differencing: Use an $O(h^2)$ two-level scheme—time-centred at $t = t^{n+1/2} \equiv t^n + \triangle t/2$—of the schematic form

$$\frac{u_j^{n+1} - u_j^n}{\triangle t} + \mu_t\left(D_x u_j^n\right) + 12\left(\mu_t\, u_j^n\right)\mu_t\left(D_x u_j^n\right) + \mu_t\left(D_{xxx} u_j^n\right) = 0 \tag{2.4}$$

where μ_t is the time averaging operator

$$\mu_t\, v_j^n \equiv \frac{1}{2}\left(v_j^n + v_j^{n+1}\right) \tag{2.4}$$

and D_x and D_{xxx} are centred, $O(h^2)$ FD approximations of ∂_x and ∂_{xxx} respectively.

Solving the algebraic equations: The discretization sketched above should yield a set of nonlinear equations:

$$F_j\left[u_{j'}^{n+1}\right] = 0 \qquad j = 3, 4, \ldots, \mathtt{nx} - 2\,; \quad j' = 1, 2, \ldots, \mathtt{nx} \tag{2.5a}$$

to which you should adjoin the following 4 equations:

$$u_1^{n+1} = u_2^{n+1} = u_{\mathtt{nx}-1}^{n+1} = u_{\mathtt{nx}}^{n+1} = 0 \tag{2.5b}$$

to yield a set of `nx` equations in the `nx` unknowns $u_j^{n+1},\ j = 1, 2, \ldots, \mathtt{nx}$. You should find that the Jacobian matrix of (2.5) is *5-diagonal* (*pentadiagonal*). At each time step then, solve for the u_j^{n+1} using an `nx`-dimensional Newton method. Use the `LAPACK` banded-solver, `dgbsv` (discussed in class), to solve the linear systems which arise in the Newton iteration and use

$$\frac{\|\delta \mathbf{u}\|_2}{\|\mathbf{u}\|_2} \leq 1.0^{-10}$$

as your convergence criterion for the Newton method.

Output: The command-line parameter `<olevel>` controls the frequency of standard output. Specifically, every $2^{\texttt{<level>}-\texttt{<olevel>}}$ time steps (including the 0th timestep $t^0 = 0$), `kdv` must produce output as follows (your variable names may differ, of course):

```
write(*,*) t,     nx
do i = 1 , nx
   write(*,*) x(i) ,    u(i)
end do
```

where `t` is the integration time, `nx` is the number of spatial grid points, `x(1:nx)` is the spatial coordinate vector, and `u(1:nx)` is the difference-solution vector (at time `t`). You may find it convenient to output other quantities in other fashions as you develop `kdv`, but your final program should produce *only* the above on standard output. If you are interested in using my SGI-specific visualization utility (`ser` aka `vs`) which was custom-built for this type of application, please see me personally.

Testing and results: Test your program thoroughly, particularly for convergence. Investigate the behaviour of the solution for single-pulse initial profiles of varying amplitude. Attempt to determine how the propagation speed of a pulse varies with amplitude. What typically happens to the difference solution if a disturbance is allowed to hit $x = x_{\max}$? Report your findings and anything else you find interesting in ~`/h4/a2/README`. Finally, using the output from

```
kdv 15.0 0.33  13 8  0.35    8.0 -11.0   6.0 -6.0   2.0 -1.0
```

make a figure consisting of nine plots arranged in a 3 x 3 configuration, which shows $u(x,t)$ at all nine output times. Save a Postscript version of your plot in ~`/h4/a2/soliton3.ps`. Let me know immediately if you have (or perceive) undue difficulty making such a plot. Also note that the above level-13 run is likely to take a minimum of several minutes on the `lnx`.

Once you have finished debugging your program (presumably *before* you make the level-13 run!), you should re-compile and re-link using the `f77` flags `-O2 -n32` (instead of `-g -n32`) in order to optimize your code, and thus minimize run-time.