PHY/CAM 381C: Computational Physics Homework 4
Due: Tuesday, April 22, 12:30pm

Important: *This assignment requires you to write two* `f77` *programs which involve the FD solution of PDEs. As always, please follow the instructions carefully, particularly with regards to command-line arguments and required input/output.*

Problem 1: Implement the Linear Correction Scheme (LCS) multi-grid algorithm described in class for the 2d model problem:

$$u_{xx} + u_{yy} = f(x,y) \qquad \text{on} \qquad 0 \le x, y \le 1 \qquad \text{with} \qquad u(x,0) = u(x,1) = u(0,y) = u(1,y) = 0 \tag{1}$$

Specifically, in `~/hw4/a1`, write an adequately commented `f77` program `mglcs2d` with usage:

```
usage: mglcs2d <level> [<nvcycle> <preswp> <pstswp>]
```

where all arguments have the same interpretation and default values as for the FAS code, `mgfas2d`, which was discussed in class. You may borrow freely from the FAS code and its supporting routines, although you should provide proper attribution should you choose to do so.

Configure your code to solve (1) with

$$f(x,y) = -5\pi^2 \sin(\pi x)\sin(2\pi y)$$

so that

$$u_{\text{exact}} = \sin(\pi x)\sin(2\pi y)$$

and note that `mgcls2d`, like `mgfas2d`, is to implement a full-multilevel algorithm—i.e. for an invocation with $l_{\max} =$ `<level>`, `mgcls2d` must solve problems on levels, $l = 1, 2, \ldots, l_{\max}$, where each level $l+1$ solution is initialized via interpolation of the level l solution.

`mgcls2d` should produce tracing output on standard error similar (or identical) in format to that produced by `mgfas2d`. The last output produced on standard error *must* be a suitable norm of the error in the computed solution (i.e. in the $l =$ `<level>` solution). There is no other *required* output, although you may find it useful to trace other quantities as you develop your program. In this regard, you may find the `bbhutil` routines—briefly discussed in class and documented via the Course Software Web page—useful if you wish to use `Explorer` on `einstein`'s console to visualize 2d grid-functions. Also note that `gnuplot` can make parametric "surface plots", $f(x,y)$. See `help splot` in `gnuplot` for more information

Test your code thoroughly, particularly for convergence. I will do the same.

Problem 2: 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

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

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, $\texttt{nx} = 2^{\texttt{<level>}} + 1$
- `<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:

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

to which you should adjoin the following 4 equations:

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

to yield a set of `nx` equations in the `nx` unknowns u_j^{n+1}, $j = 1, 2, \ldots, \texttt{nx}$. 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

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

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 `einstein`. 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.