PHYS 410/555: Computational Physics Fall 2001 Homework 5
DUE: Tuesday, November 27, 10:00 AM Report bugs to choptuik@physics.ubc.ca

The following assignment involves writing and testing two Fortran 77 programs which use `LSODA` *to solve ordinary differential equations. As usual, all files required by the assignment must reside in the correct places on your* `lnx` *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: Consider the following ODE, known as the Van der Pol equation:

$$\frac{d^2u(t)}{dt^2} - \epsilon\left(1 - u(t)^2\right)\frac{du(t)}{dt} + u(t) = 0 \qquad t \geq 0 \tag{1}$$

where $\epsilon > 0$ is a specified *positive* constant. Physically, this ODE describes the voltage behaviour of a tunnel diode oscillator, although for the purposes of this homework we will simply be viewing it as providing an interesting example of non-linear oscillator dynamics.

In directory `~/hw5/a1`, write a `Fortran 77` program `vdp.f`, with corresponding executable `vdp` which solves the above ODE subject to the initial conditions:

$$u(0) = \mathtt{u0} \tag{2}$$

$$\frac{du}{dt}(0) = \mathtt{du0} \tag{3}$$

`vdp` must have the following usage:

```
usage: vdp <tmax> <u0> <du0> <epsilon> <tol> <olevel>
```

where

1. `tmax` is the `real*8` final integration time ($\mathtt{tmax} > 0$).
2. `u0` is the `real*8` initial oscillator displacement.
3. `du0` is the `real*8` initial oscillator velocity.
4. `epsilon` is the `real*8` value of ϵ ($\mathtt{epsilon} > 0$).
5. `tol` is the `real*8` `LSODA` tolerance. Use `itol = 1`, and pure absolute tolerance.
   ($10^{-12} \leq \mathtt{tol} \leq 10^{-2}$)
6. `olevel` is the `integer` output level. ($\mathtt{olevel} > 0$).

Your program must be commented, and must perform rudimentary error-checking of command-line arguments: arguments should be of the proper type, and should be in the appropriate ranges as given above (as usual, you can use `i4arg` and `r8arg` to detect whether an argument is of the correct type by using default values which a user is unlikely to enter).

At requested output times, `vdp` must write on standard output the time, and the computed displacement and velocity of the oscillator at that time, using an output statement such as

```
write(*,*) t, y(1), y(2)
```

The requested output times, t_j, are defined by

$$t_j = 0,\ h,\ 2h,\ \cdots\ \mathtt{tmax} \tag{4}$$

where h is given by

$$h \equiv \frac{\mathtt{tmax}}{2^{\mathtt{olevel}}} \tag{5}$$

To test your implementation, code an independent residual evaluation program, `chk-vdp.f`—with corresponding executable `chk-vdp`—which applies an $O(h^2)$ finite-difference approximation of (1) to the output of vdp. chk-vdp.f should be identical in construction, as well as what it inputs and outputs, to the chk-tlsoda.f example covered in class, except that chk-vdp.f is to accept a single, real*8 command-line argument, epsilon. Use the usual $O(h^2)$ difference approximations for the first and second time-derivatives in (1); namely:

$$\frac{du}{dt}(t_j) = \frac{u(t_{j+1}) - u(t_{j-1})}{2h} + O(h^2) \qquad (6)$$

$$\frac{d^2u}{dt^2}(t_j) = \frac{u(t_{j+1}) - 2u(t_j) + u(t_{j-1})}{h^2} + O(h^2) \qquad (7)$$

Create a script Do-chk which contains the following

```
#!/bin/sh

epsilon=1.0
vdp 10.0 0.0 1.0 $epsilon 1.0d-8 10 | nth 1 2 > vdp-out

for inc in 8 4 2 1; do
   nth 1 2 < vdp-out | lines 1 . $inc | chk-vdp $epsilon
done
```

Once you are satisfied that both vdp and chk-vdp are working properly, execute the script and save the output as chk-vdp-out.

In order to get a feel for the behaviour of the oscillator, use vdp to investigate the solution of (1) for a variety of values of ϵ ($0 \le \epsilon \le 5$ suggested), u0 ($10^{-3} \le$ u0 ≤ 10 suggested) and du0, ($-5 \le$ du0 ≤ 5 suggested). When performing your studies, ensure that you specify tmax large enough to determine the long-time behaviour of the oscillator. You can use gnuplot to plot the results of your computations.

An interesting way to examine the dynamics of an oscillator is to make a *phase-space plot*, i.e. a parametric plot (in t) of the oscillator's velocity versus its displacement. Use gnuplot to make a *single* phase-space Postscript plot called vdp-phase showing the du/dt vs u trajectories from the following three computations:

```
vdp 50.0   0.01  0.0  1.0 1.0e-8 11
vdp 50.0    1.0 -1.0  1.0 1.0e-8 11
vdp 50.0    3.0  1.0  1.0 1.0e-8 11
```

Using gnuplot, make a Postscript plot called vdp.ps showing the oscillator displacement versus t for the following computation

```
vdp 100.0 0.01 0.0 10.0 1.0e-8 12
```

What can you say about the long-time behaviour of the oscillator as a function of $u(0)$ and $du/dt(0)$ for fixed ϵ? Answer in ~/hw5/a1/README.

Problem 2: Consider the following partial differential equation, (frequently called the *diffusion equation*, or *heat equation*):

$$\frac{\partial u(x,t)}{\partial t} = \frac{\partial^2 u(x,t)}{\partial x^2} \qquad \text{on} \qquad 0 \le x \le 1 \quad t \ge 0 \qquad (8)$$

subject to (i) the initial conditions:

$$u(x,0) = u_0(x) \qquad (9)$$

(where $u_0(x)$ is a specified function), and (ii) the boundary conditions:

$$u(0,t) = u(1,t) = 0 \qquad (10)$$

In directory ~/hw5/a2, write a `Fortran 77` program, `diffusion.f`, with corresponding executable `diffusion`, which solves the above PDE using the method of lines, and `LSODA`. Specifically, convert (8) into a system of N coupled ODEs by (i) introducing a regular mesh x_j and discrete unknowns u_j:

$$x_j \equiv (j-1)h \quad j = 1, 2, \cdots N \qquad h \equiv (N-1)^{-1} \qquad u_j(t) \equiv u(x_j, t) \tag{11}$$

(ii) replacing $\partial^2 u/\partial x^2$ with the usual $O(h^2)$ finite difference approximation, yielding

$$\frac{d\,u_j(t)}{dt} = \frac{u_{j+1} - 2u_j + u_{j-1}}{h^2} \qquad j = 2, \cdots N-1 \tag{12}$$

and (iii) incorporating the Dirichlet boundary conditions (10) as follows:

$$\frac{du_1(t)}{dt} = 0 \tag{13}$$

$$\frac{du_N(t)}{dt} = 0 \tag{14}$$

`diffusion` must have the following usage:

```
usage: diffusion <xlevel> <olevel> <tfinal> <dtout> <tol>
```

where

1. `xlevel` is the `integer` level of spatial discretization: $N \equiv$ `nx` $= 2^{\texttt{xlevel}} + 1$. ($1 \le$ `xlevel` ≤ 9).
2. `olevel` is the `integer` level of spatial discretization for output to be subsequently plotted as a `gnuplot parametric` surface plot. See the discussion of the `stride` argument of the instructor-supplied `gnuout` routine below. ($1 \le$ `olevel` $\le$ `xlevel`).
3. `tfinal` is the `real*8` final integration time. (`tfinal` > 0).
4. `dtout` is the `real*8` output time interval: i.e. `diffusion` produces output at $t =$ `dtout`, 2 `dtout`, ..., $N_{\texttt{out}}$ `dtout` where $N_{\texttt{out}}$ is the largest integer satisfying $N_{\texttt{out}}$ `dtout` $\le$ `tfinal`. (`dtout` > 0).
5. `tol` is the `real*8` `LSODA` tolerance. Use `itol = 1`, and `atol = rtol = tol`. ($10^{-12} \le$ `tol` $\le 10^{-2}$)

For the initial conditions (9), use the following gaussian profile:

$$u(x,0) = u_0(x) = \exp(-((x-0.5)/0.1)^2) \qquad 0 < x < 1 \tag{15}$$

$$u(0,0) = u(1,0) = 0 \tag{16}$$

`diffusion` must output the approximate solution to standard output using the `gnuout` routine, which can be copied from ~phys410/hw5/diffusion/gnuout.f:

```
c===========================================================
c     Output to standard out for subsequent plotting via
c     gnuplot.
c===========================================================
      subroutine gnuout(u,x,nx,t,stride)
         implicit       none

         integer        nx,          stride
         real*8         u(nx),       x(nx),        t

         integer        j

         do j = 1 , nx , stride
            write(*,*) t, x(j), u(j)
```

```
      end do
      write(*,*)

      return

   end
```

Note that the `stride` argument is used to "down-sample" the data being written, since at high spatial resolutions, direct surface plotting via `gnuplot` will produce very large, mostly illegible plots. In the current case `stride` is defined as follows:

$$\texttt{stride} \equiv 2^{\texttt{xlevel}-\texttt{olevel}} \tag{17}$$

Your program must be commented and must perform error-checking of command-line arguments as in Problem 1.

In developing your program, you may find it useful to use the `vsxynt` interface, the `Scivis` visualization server, and the `sdftosv` utility program which sends `.sdf` files to `Scivis`. See the Course Software page for a brief description of these topics, as well as the *Supplementary Material for Homework 5* section of the ODE Notes page.

After your program has been de-bugged to your satisfaction, use it to generate a file `out8` which contains the standard output of the invocation

```
diffusion 8 6 0.20 0.005 1.0e-6
```

Using `gnuplot`, make a parametric surface plot of this data and save it as the Postscript file `out8.ps`.

Finally, in ~`hw5/a2/README`, briefly discuss how you might go about verifying that your program is correct (i.e. that as `tol` $\to 0$ and $h \to 0$ we have $u_j(t) \to u(t)$).