PHYS 210: Introduction to Computational Physics   Fall 2012   Homework 4

Due: Tuesday, November 20, 11:59 PM

PLEASE report all bug reports, comments, gripes etc. to Matt: `choptuik@physics.ubc.ca`

Please make careful note of the following information and instructions:

1. I struggled a bit with an appropriate due date for this assignment. I would *really* like you to complete it by November 15, but thought you might think (collectively) that that would be a bit too close to the Homework 3 deadline.

   The crucial thing to bear in mind here is that most of you will need a significant amount of time to complete your term projects. I thus encourage you to complete the homework as soon as you can, so that you can devote your attention to your projects.

2. That said, the following assignment requires you to write (only!) one `octave` function.

3. As in the previous homework, there is an associated instructor-supplied "driver" script that you will use to generate the final results for the question. As usual, these results will consist of an inventory of files that is listed at the end of the problem description.

   You can also use my driver as a guide/template for the design of your own script that can be used to test your function as you code it.

   Warning!! If you *do* modify my script for your own purposes, ensure that you give it a *different* name. That is, the script `tvdp` that you must eventually execute to complete the homework *must* be the one defined in `/home/phys210/octave/hw4`, not your own version.

4. If you want to work on this homework using `octave` on your laptop or home machine, then you will need to copy the contents of `/home/phys210/octave/hw4` from `hyper` to some directory on your personal machine(s), and, as necessary, ensure that the directory has been added to your `octave` path.

5. As usual, it is your responsibility to ensure that when your homework is complete, all requested files are in the correct directory with the correct names.

6. **Comments and error checking**: Your function should be commented at about the same level as my driver script is. You do *not* have to perform any checks for validity of input arguments.

Problem 1: *The Van der Pol Equation*

1.1 Mathematical specification

Consider the following non-linear ordinary differential equation (ODE), known as the Van der Pol equation,

$$\frac{d^2u(t)}{dt^2} - \epsilon\left(1 - u(t)^2\right)\frac{du(t)}{dt} + u(t) = 0\,, \quad 0 \le t \le t_{\max}\,, \tag{1}$$

where t is time, $u(t)$ is the displacement of the Van der Pols oscillator, $\epsilon > 0$ is a specified *positive* constant, and $t_{\max}$ is the time to which we wish to compute the dynamics of the oscillator.

Since (1) is a second order ODE, if we are to calculate a particular solution of it we must specific initial values for the displacement and the first time derivative of the displacement. That is, we must supplement (1) with the initial conditions:

$$u(0) = u_0\,, \tag{2}$$
$$\frac{du}{dt}(0) = v_0\,, \tag{3}$$

where u_0 and v_0 are values that we can choose freely.

Physically, the Van der Pol equation describes the voltage behaviour of a tunnel diode oscillator, although for the purposes of this homework we will primarily be viewing it as providing a simple yet interesting example of non-linear dynamics.

We note that ϵ is to be considered a *control parameter* of the problem: variation of ϵ will induce significant changes in the behaviour of the oscillator. We will return to this point in the following section. We will solve equations (1)–(3) using second order finite difference techniques. To that end, and following the basic procedure outlined in class for treating any differential problem using the finite difference approach, we first replace the continuum solution domain, $0 \le t \le t_{\max}$ with a finite difference grid, or mesh, denoted t^n and defined by

$$t^n = (n-1)\Delta t, \quad n = 1, 2, \ldots, n_t\,. \tag{4}$$

Thus there are a total of n_t grid points, and the grid spacing, Δt, is given by

$$\Delta t = \frac{t_{\max}}{n_t - 1}\,. \tag{5}$$

In addition, as in a previous homework problem—and although not necessary—we will find it convenient to make the number of mesh *intervals* a power of 2, and identify that power with the *level*, ℓ, of discretization, so that we have

$$n_t = 2^\ell + 1\,, \tag{6}$$

for some integer $\ell \ge 1$. Note that in practice, and depending on the value of $t_{\max}$, we will need to take ℓ significantly larger than 1 to ensure that the grid spacing is sufficiently small to provide a reasonable approximation of the oscillator's motion.

Having introduced the finite difference mesh, t^n, we define the associated discrete values of the oscillator displacement, u^n, by

$$u^n \equiv u(t^n)\,, \tag{7}$$

where it is to be understood that for any *finite* Δt, the u^n will only be an approximation to corresponding values of the continuum solution of (1)–(3)

$$u(t)|_{t=t^n}\,. \tag{8}$$

The next step in the discretization process involves replacement of the derivatives appearing in (1) with finite difference formulae. Here we will use the "standard" second-order, centred approximations for the first and second derivatives discussed in class, namely:

$$\frac{u^{n+1} - u^{n-1}}{2\Delta t} = \left.\frac{du}{dt}\right|_{t=t^n} + O(\Delta t^2)\,, \tag{9}$$
$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} = \left.\frac{d^2u}{dt^2}\right|_{t=t^n} + O(\Delta t^2)\,. \tag{10}$$

Using (7), (9) and (10) in (1) we get our FDA (finite difference approximation) of the oscillator equation

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} - \epsilon \left(1 - (u^n)^2\right) \frac{u^{n+1} - u^{n-1}}{2\Delta t} + u^n = 0\,, \quad n+1 = 3, 4, \ldots n_t\,. \tag{11}$$

Note that in solving (11) we will assume that the values u^n and u^{n-1} are known, leaving u^{n+1} as the single unknown in the equation. This is why we have written the discrete domain of applicability of the algebraic finite difference equations as $n+1 = 3, 4, \ldots n_t$. It also implies that in order to begin the process of solving (11) for discrete times $t^3, t^4, \ldots, t^{n_t}$, we must have the values $u^1 = u(0)$ and $u^2 = u(\Delta t)$. To avoid confusion of the last quantity with "u-squared", I will introduce the additional notation $u^{(2)} = u(\Delta t)$.

The appropriate value for u^1 follows immediately from the initial condition (2)

$$u_1 = u(0) = u_0\,. \tag{12}$$

Determining an appropriate value for $u^{(2)}$ is a little trickier. As for the case of the nonlinear pendulum discussed in class, we proceed using Taylor series expansion. We state without proof that we need to compute terms up to and including $O(\Delta t^2)$ in the expansion to ensure that the overall solution u^n is computed to second order accuracy. Thus, we write

$$u^{(2)} = u(\Delta t) = u(0) + \Delta t \frac{du}{dt}(0) + \frac{1}{2}\Delta t^2 \frac{d^2u}{dt^2}(0) + O(\Delta t^3)\,. \tag{13}$$

We note that we can evaluate the first two terms in (13) using the initial conditions (2) and (3). This leaves us the third, $O(\Delta t^2)$, term—which involves the second time derivative of u—to calculate, and the initial conditions will *not* directly allow us to do so.

However (and this is an approach that can be applied quite generally to second order differential equations that have been approximated using a three-time-level finite difference scheme such as (11)), we can use the ODE itself to replace the second time derivative with quantities that *can* be computed from the initial conditions.

Specifically, solving (1) for d^2u/dt^2 we have

$$\frac{d^2u}{dt^2} = \epsilon\left(1 - u^2\right)\frac{du}{dt} - u\,. \tag{14}$$

Substituting this into (13), and neglecting the $O(\Delta t^3)$ and higher order terms, we find

$$u^{(2)} = u(\Delta t) = u(0) + \Delta t \frac{du}{dt}(0) + \frac{1}{2}\Delta t^2 \left[\epsilon\left(1 - u(0)^2\right)\frac{du}{dt}(0) - u(0)\right]\,. \tag{15}$$

Then using the initial conditions (2) and (3) we can rewrite (15) as

$$u^{(2)} = u_0 + \Delta t\, v_0 + \frac{1}{2}\Delta t^2 \left[\epsilon\left(1 - u_0^2\right) v_0 - u_0\right]\,. \tag{16}$$

We are now almost done with our mathematical development of the problem. The last thing we need to do before proceeding to a specification of the `octave` function that you will code to solve the Van der Pols equation is to manipulate (11) so that it provides an explicit expression for the (advanced-time) unknown, u^{n+1}.

I thus leave it as an exercise for you (*please* do it!) to verify that if we define an auxiliary quantity, q, as follows:

$$q \equiv \frac{1}{2}\epsilon\Delta t\left(1 - (u^n)^2\right)\,, \tag{17}$$

then (11) implies that

$$u^{n+1} = (1-q)^{-1}\left[\left(2 - \Delta t^2\right) u^n - (1+q)\, u^{n-1}\right] \quad n+1 = 3, 4, \ldots n_t\,. \tag{18}$$

In summary, equations (18), (12) and (16) provide a complete set of algebraic equations for the unknowns u^n, $n = 1, 2, \ldots, n_t$, and we are now in a position to implement their solution as an `octave` function.

1.2 The problem *per se*

Make the directory `/phys210/$LOGNAME/hw4/a1`, and within that directory create an `octave` source file, `vdp.m`, that defines the function `vdp` having the following header:

```
function [t u dudt] = vdp(tmax, level, u0, dudt0, epsilon)
```

The input arguments to `vdp` are as follows:

- `tmax`: $t_{\max}$ as defined above.
- `level`: The integer-valued discretization level, ℓ, as described above. This argument is used to define the number of points, n_t, in the finite difference mesh, the discrete time step (mesh spacing), Δt, and the discrete times, t^n, via equations (6), (5) and (4), respectively.
- `u0` and `v0`: The initial values $u(0)$ and $du/dt|_{t=0}$.
- `epsilon`: The control parameter ϵ.

Again, you do not have to do any checks of the validity of the input arguments.

Your implementation of `vdp` must define the 3 return values as follows:

- `t`: A row vector of length n_t containing the discrete times, t^n.
- `u`: A row vector of length n_t containing the discrete solution, u^n, as given by equations (12) and (16) and (18).
- `dudt`: A row vector of length n_t containing a finite difference approximation of $du/dt(t^n)$ computed using formula (9) for $n = 2, 3, \ldots, n_t - 1$. For $n = 1$ and $n = n_t$ you should use the following

  ```
  dudt(1) = v0;
  dudt(nt) = 2 * dudt(nt-1) - dudt(nt-2);
  ```

  The first of these expressions follows immediately from the initial condition (3), while the second computes the final value of `dudt` using *linear extrapolation* of the second- and third-to-last values.

The driver script for this problem is

```
/home/phys210/octave/hw4/tvdp.m
```

and, again, you are welcome to use it as an aid in designing your own script to develop and test your implementation of `vdp`. (Remember, though, do *not* name your script file `tvdp.m`!!) You can also use interactive experimentation for this purpose should you find it more convenient. No matter how you proceed with development, you are urged to make use of `octave`'s plotting facilities: it is almost always much easier to understand what a program such as `vdp` is or is not doing through visualization, rather than staring at numbers.

As part of your testing process, you should try to establish that your finite difference solution is *converging* as expected, using the basic technique discussed in class and demonstrated in lab activities. Note that convergence testing is implemented in `tvdp`.

Once you are convinced that your `octave` function for approximately solving (1)–(3) is correct, and to the extent that time permits, you are encouraged to experiment with a variety of values of ϵ ($0 \le \epsilon \le 5$ suggested), u_0 ($10^{-3} \le u_0 \le 10$ suggested) and v_0 ($-5 \le v_0 \le 5$ suggested). When performing your studies, ensure that you specify $t_{\max}$ large enough to determine the long-time behaviour of the oscillator, and, dependent on the values of ϵ and $t_{\max}$, choose a minimum value for ℓ so that the oscillator's behaviour is well resolved.

An interesting way to view the output of `vdp` is through a *phase space* plot; i.e. a plot of $du(t)/dt$ vs $u(t)$. Again, this is something that is done in the driver `tvdp`, and you are encouraged to include phase space plotting in your own driver script.

You should briefly document what you find through your numerical experiments in `hw4/a1/README`.

To complete this problem, execute the supplied driver script, `tvdp`, which should produce output as follows

```
>> tvdp
These calculations should take a few seconds ...
Type 'Enter' to continue:
Type 'Enter' to continue:
These calculations should take a second or two more ...
Type 'Enter' to continue:

Done!!
```

You should also see three plots appear on your screen as the script executes.

Provide a summary of what you can deduce from the phase space plot (hardcopy in `phase_12.ps`) that is produced by running `tvdp` in your `README` file.

NOTE: You are free to use the function `pendulum`, defined in `~phys210/octave/pendulum.m`, as a template for your implementation of `vdp` (although you are encouraged to try to write the function on your own). If you *do* use `pendulum` for inspiration, you should acknowledge that fact in a comment block, e.g.

```
% This function based on the instructor-written function 'pendulum'
```

In fact, you should *always* cite your source(s) for code that you borrow, modify, etc., for the same reasons that you must always cite material that you use in writing papers etc.

File Inventory

1. `vdp.m`
2. `u_12_13_14.ps`
3. `du_12_13_14.ps`
4. `phase_12.ps`
5. `README`