PHYS 210: Introduction to Computational Physics   Fall 2009   Homework 3

Final Version, November 6

Due: Friday, November 13, 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. The following assignment requires you to write three `octave` functions to accomplish various tasks.
2. For each problem, there is an associated instructor-supplied "driver" script that you will use to generate the final results for the question. In each case these results will consist of an inventory of files which is listed at the end of the problem description. Also note that for each problem you will have to prepare a *single* `.m` file that will contain the definition of the required `octave` function: that `.m` file will be part of the inventory.

   You can also use the instructor-supplied driver scripts as guides / templates for the design of your own scripts to test your `octave` functions as you code them.

   Warning!! If you *do* modify these scripts for your own purposes, ensure that you give them *different* names. That is, the scripts `tchaos`, `tfdas` and `tvdp` that you must eventually execute to complete the homework *must* be the ones defined in `/home/phys210/octave/hw3`, not your own versions.

   You will **definitely** *lose marks if you don't heed this warning, so let me know if there is anything about it that you don't understand!!*
3. **Important!!** In order to ensure that you can execute the driver scripts from within `octave` (executing on `hyper`), you need to add the following line to your `~/.octaverc` file.

   ```
   addpath('/home/phys210/octave/hw3')
   ```

   This needs to be done *before* you start `octave` if you want the driver scripts to be available in that session.
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/hw3` from `hyper` to some directory on your personal machine(s), and ensure that the target directory has been added to your `octave` path as above.
5. Although you *could* use `MATLAB` to develop and test the functions that you need to write to complete this homework, it is recommended that you do *not* do so, since the driver scripts will generally not work as expected under `MATLAB` due to differences in the plotting facilities between `octave` and `MATLAB`.
6. Since all of the driver scripts produce Postscript files, recall that you can use `gv` to view such files. Please use `gv` to ensure that all of the files that are to contain Postscript versions of plots, actually *do* contain plots.
7. As usual, it is your responsibility to ensure that when your homework it complete, all requested files are in their correct locations with the correct names, and that **all code executes properly on hyper**.

   Warning!! *At this stage in the course, the TA has the authority to begin deducting marks if you do not follow this protocol!!*

   Also as usual, note that any reference below to the directory `hw3` is implicitly a reference to `/phys210/$LOGNAME/hw3`.
8. **Comments and error checking**: Your functions should be commented at about the same level as my driver scripts are. Also, unless explicitly stated otherwise, your functions do *not* have to perform any checks for validity of input arguments.
9. **Loopless coding**: In the spirit of making use of `octave`'s powerful facilities for performing whole-array operations, you should attempt to solve problem 2 using as few `for` or `while` loops as possible.
10. Due to the amount of programming involved in this assignment relative to the previous two, as well as to the fact that it involves some mathematics that may be somewhat unfamiliar to you, this problem set may be quite challenging for some of you.

    However, the homework, it has also been designed to help you gain a level of proficiency in `octave` programming that will be necessary, in most cases, for the successful completion of your term projects (for those of you who *will* be using `octave` for your projects).

    Moreover, as you probably suspect, the length of this handout does not reflect the total length of code that must be written to complete the homework. Excluding comments (but including some tracing code), my solutions amount to about 80 lines of code.

    In any case I strongly recommend that you begin work on the assignment as soon as possible, and to seek help should you find yourself spending an undue amount of time (i.e. hours and hours) on any given problem.

PROBLEM 1: *The Chaos Game*

1.1 Mathematical specification

Consider the x-y plane and let

$$V_i \equiv (x_i, y_i) \quad i = 1, 2, 3 \tag{1}$$

be the vertices of an equilateral triangle that lies in that plane and which is inscribed in the unit circle defined by $x^2 + y^2 = 1$ (i.e. the circle with unit radius whose center is located at the origin $(0, 0)$). Specifically, the coordinates of the V_i are as follows:

$$V_1 = \left(\cos\left(\frac{\pi}{2}\right), \sin\left(\frac{\pi}{2}\right)\right) \tag{2}$$

$$V_2 = \left(\cos\left(\frac{7\pi}{6}\right), \sin\left(\frac{7\pi}{6}\right)\right) \tag{3}$$

$$V_3 = \left(\cos\left(\frac{11\pi}{6}\right), \sin\left(\frac{11\pi}{6}\right)\right) \tag{4}$$

The game is to be initialized by choosing, *at random*, some point—also in the xy plane—that lies on the circle centred at the origin, but with a radius $r = 1.25$. Call this randomly chosen point the *active point*, V_p. With this initialization, the game is played by repeating the following operations any desired number of times (steps):

1. Choose one of the three vertices, V_i, of the triangle *randomly.*
2. Consider the line segment $\overline{V_i V_p}$ between the active point, V_p, and the randomly chosen triangle vertex, V_i.
3. Bisect $\overline{V_i V_p}$ to define a new point, V_q, that thus lies exactly at the midpoint of $\overline{V_i V_p}$.
4. Make this new point V_q the active point V_p.

If one plots all the active points V_p which are generated, an interesting pattern appears after a sufficiently large number of iterations.

1.2 The problem *per se*

Make the directory ~`/hw3/a1` and within that directory create an `octave` source file `chaos.m` that defines an `octave` function with the following header:

```
function [x y] = chaos(nsteps)
```

The single input argument to `chaos` is defined as follows:

- `nsteps`: Number of steps to play. This should be an integer scalar greater than 0, and your implementation of `chaos` can *assume* that it is (i.e. you don't need to do any error checking of `nsteps`).

The output arguments of `chaos` are:

- `x` and `y`: Vectors of length `nsteps + 4` containing the x and y coordinates, respectively, of all of the active points generated by playing the game, as well as the vertices of the equilateral triangle. Specifically, on return `x` and `y` should be defined as follows
  - `x(k), y(k), k = 1, 2, 3`: Vertices of triangle as defined in (2)-(4)
  - `x(4), y(4)`: Coordinates of randomly chosen initial active point.
  - `x(k), y(k), k = 5, ... nsteps + 4`: Coordinates of subsequently generated active points.

The driver script for `chaos` is defined in

```
/home/phys210/octave/hw3/tchaos.m
```

As noted above, you can use this script as a template for the purposes of guiding the development and testing of your implementation of `chaos`, but, as emphasized in the preliminary comments, if you do so, *make sure that you don't call it* `tchaos.m`!—call it `mytchaos.m`, or some such, so that there can be no confusion with the true driver.

Once you are confident that your implementation of `chaos` is working properly, invoke the driver script `tchaos`: i.e. ensuring that `octave` is executing in `hw3/a1`, type `tchaos` at the `octave` prompt. As the script executes you should see the following on your terminal:

```
>> tchaos
Type 'Enter' to continue:
This will take a few seconds ... please be patient!
Type 'Enter' to continue:
This will take even longer (about half a minute) ...
Please be even more patient!
Type 'Enter' to continue:

Done!!
```

Here (as well as in subsequent problems) each time you see the text `Type 'Enter' to continue:`, you must do as instructed in order for the script to continue execution. Once you see the text `Done!!` you can exit `octave` and check the various `.ps` (Postscript) files to ensure that they contain what you expect. You should also see various plots appear in on your screen as the script executes.

File Inventory for directory `hw3/a1`

1. `chaos.m`
2. `chaos-1000.ps`
3. `chaos-10000.ps`
4. `chaos-100000.ps`

Optional! Generalize the game in various ways such as

1. Adding an option which controls the number of fixed vertices used (i.e. so that you can play with 4, 5, 6, ... fixed vertices, rather than just 3).
2. Adding an option which controls the placement of new points: new points should continue to be placed along a line segment connecting the previous point and the randomly chosen vertex, but need not be at the mid-point of the line segment.
3. Implementing the game in three dimensions.

If you do chose to generalize the game, be sure to create a *new* `.m` file that defines an `octave` function with a name different than `chaos`—`chaos` must execute precisely as specified above. Leave comments describing your generalization(s) in a `README` file in the solution directory.

Some bonus marks will be available for successful implementation of one or more of these extensions, as well as for suitably imaginative generalizations of your own invention.

PROBLEM 2: *Finite Difference Approximations*

2.1 Mathematical specification

Consider a uniform finite difference grid, x_j, defined on some domain, $x_{\rm min} \le x \le x_{\rm max}$

$$x_j = x_{\rm min} + (j-1)\Delta x \quad j = 1, 2, \cdots n_x \, , \tag{5}$$

where the *grid spacing* (or *mesh spacing* or *discretization scale*), Δx, is given by

$$\Delta x = \frac{x_{\rm max} - x_{\rm min}}{n_x - 1} \, . \tag{6}$$

We adopt the following notation (introduced in class) to denote the values of an arbitrary function $f(x)$ at the grid points, x_j:

$$f_j = f(x_j) \tag{7}$$

Then as also discussed in class, the following finite difference expressions define approximations to the first derivative, $f'(x) = df(x)/dx$, evaluated at the grid points x_j:

First order forward difference approximation of $f'(x_j)$

$$\frac{f_{j+1} - f_j}{\Delta x} = f'(x_j) + O(\Delta x) \approx f'(x_j) \tag{8}$$

First order backward difference approximation of $f'(x_j)$

$$\frac{f_j - f_{j-1}}{\Delta x} = f'(x_j) + O(\Delta x) \approx f'(x_j) \tag{9}$$

Second order centered difference approximation of $f'(x_j)$

$$\frac{f_{j+1} - f_{j-1}}{2\Delta x} = f'(x_j) + O(\Delta x^2) \approx f'(x_j) \tag{10}$$

In addition, we derived the following finite difference formula that approximates the second derivative, $f''(x) = d^2 f(x)/dx^2$, once again at the mesh points x_j:

Second order centered difference approximation of $f''(x_j)$

$$\frac{f_{j+1} - 2f_j + f_{j-1}}{\Delta x^2} = f''(x_j) + O(\Delta x^2) \approx f''(x_j) \tag{11}$$

2.2 The problem *per se*

Make the directory `hw3/a2`, and within that directory create an `octave` source file, `fdas.m`, that defines the function `fdas` having the following header

```
function [x df db dc ddc] = fdas(fcn, xmin, xmax, level)
```

The input arguments to `fdas` are as follows:

- `fcn`: Function handle for $f(x)$. This will typically be a handle to one of `octave`'s built-in math functions, such as `sin`, `sqrt`, `cosh` etc.
- `xmin`: Minimum coordinate, $x_{\rm min}$, of the finite difference grid.
- `xmax`: Maximum coordinate, $x_{\rm max}$, of the finite difference grid.
- `level`: Discretization level. As discussed in class, this provides a convenient way of specifying the number of grid points (or equivalently the grid spacing), particularly for the purpose of testing convergence of finite difference approximations. The finite difference grid will contain $n_x = 2^{\rm level} + 1$ points, and will have a mesh spacing, Δx, as defined by (6).

Your implementation of `fdas` must define the 5 output arguments as follows:

- `x`: Vector of length n_x containing the values x_j as defined by (5).
- `df`: Vector of length n_x containing the first order forward difference approximation of $f'(x_j)$ as defined by (8).
- `db`: Vector of length n_x containing the first order backward difference approximation of $f'(x_j)$ as defined by (9).
- `dc`: Vector of length n_x containing the second order centred difference approximation of $f'(x_j)$ as defined by (10).
- `ddc`: Vector of length n_x containing the second order centred difference approximation of $f''(x_j)$ as defined by (11).

Note that *all* of the vectors returned by `fdas` must be of length n_x. This means that you must evaluate all of the finite difference approximations (8)–(11) at *all* of the grid points, x_j, *including* the end points $x_1 = x_{\min}$ and $x_{n_x} = x_{\max}$.

Hint (may be helpful for implementing the computations in `fdas` *without the use of* `for` *loops):*

Assume that we have defined an octave vector `x` of length n_x using (5). We can then "temporarily" add additional values to the ends of the vector using a statement such as

```
   x = [(x(1) - deltax)  x  (x(nx) + deltax)];
```

and then later delete them using the sequence

```
   x(1) = [];
   x(nx+1) = [];
```

The driver script for this problem is

```
/home/phys210/octave/hw3/tfdas.m
```

When you are satisfied that you have implemented `fdas` correctly, execution of the driver should produce output as follows:

```
>> tfdas
Type 'Enter' to continue:
Type 'Enter' to continue:
Type 'Enter' to continue:
Type 'Enter' to continue:
Scaled RMS errors for O(h) forward FDA of d/dx
Level  Scaled Error
       .
       .
       .

Scaled RMS errors for O(h) backward FDA of d/dx
Level  Scaled Error
       .
       .
       .

Scaled RMS errors for O(h^2) centred FDA of d/dx
Level  Scaled Error
       .
       .
       .

Scaled RMS errors for O(h^2) centred FDA of d^2/dx^2
Level  Scaled Error
```

```
        .
        .
        .
```

where output that constitutes part of the solution has been suppressed. Again, various plots should appear on your screen as the script executes.

If you examine the output labelled

```
Scaled RMS errors for O(h^2) centred FDA of d^2/dx^2
```

you should notice something suspicious about the final few numbers.

Question to be answered in `a2/README`**:** What do you notice, and can you provide an explanation for the observed behaviour?

File Inventory for directory `hw3/a2`

1. `fdas.m`
2. `dsin-6.ps`
3. `edsin-6.ps`
4. `ddsin-6.ps`
5. `eddsin-6.ps`
6. `README`

PROBLEM 3: *The Van der Pol Equation*

3.1 Mathematical specification

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

$$\frac{d^2u(t)}{dt^2} - \epsilon\left(1 - u(t)^2\right)\frac{du(t)}{dt} + u(t) = 0\,, \quad 0 \le t \le t_{\max}\,, \tag{12}$$

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 (12) 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 (12) with the initial conditions:

$$\begin{aligned} u(0) &= u_0\,, & (13)\\ \frac{du}{dt}(0) &= v_0\,, & (14) \end{aligned}$$

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 (12)–(14) 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

$$t_n = (n-1)\Delta t, \quad n = 1, 2, \ldots, n_t\,. \tag{15}$$

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

$$\Delta t = \frac{t_{\max}}{n_t - 1}\,. \tag{16}$$

In addition, as in the previous 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

$$n_t = 2^\ell + 1 \tag{17}$$

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

$$u_n \equiv u(t_n) \tag{18}$$

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 (12)–(14)

$$u(t)|_{t=t_n}\,. \tag{19}$$

The next step in the discretization process involves replacement of the derivatives appearing in (12) with finite difference formulae. Here we will use the "standard" second-order, centred approximations for the first and second derivatives, namely:

$$\begin{aligned} \frac{u_{n+1} - u_{n-1}}{2\Delta t} &= \left.\frac{du}{dt}\right|_{t=t_n} + O(\Delta t^2) & (20)\\ \frac{u_{n+1} - 2u_n + u_{n-1}}{\Delta t^2} &= \left.\frac{d^2u}{dt^2}\right|_{t=t_n} + O(\Delta t^2) & (21) \end{aligned}$$

Using (18), (20) and (21) in (12) we get our FDA (finite difference approximation) of the oscillator equation

$$\frac{u_{n+1} - 2u_n + u_{n-1}}{\Delta t^2} - \epsilon\left(1 - u_n^2\right)\frac{u_{n+1} - u_{n-1}}{2\Delta t} + u_n = 0\,, \quad n+1 = 3, 4, ... n_t\,. \tag{22}$$

Note that in solving (22) 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 $n + 1 = 3, 4, ...n_t$. It also implies that in order to begin the process of solving (22) for discrete times $t_3, t_4, \ldots, t_{n_t}$, we must have the values $u_1 = u(0)$ and $u_2 = u(\Delta t)$ in hand.

The appropriate value for u_1 follows immediately from the initial condition (13)

$$u_1 = u(0) = u_0 \tag{23}$$

(be careful not to confuse the two uses of subscripts here, u_1 refers to the approximate value of $u(t)$ at the first mesh point, while u_0 is the freely specified initial condition for the oscillator's displacement).

Determining an appropriate value for u_2 is a little trickier. We proceed using Taylor series expansion and state without proof (although we *may* discuss this issue in class should time permit), 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

$$u_2 = u(\Delta t) = u(0) + \Delta t\frac{du}{dt}(0) + \frac{1}{2}\Delta t^2\frac{d^2u}{dt^2}(0) + O(\Delta t^3) \tag{24}$$

We note that we can evaluate the first 2 terms in (24) using the initial conditions (13) and (14). 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 (22)), we can use the ODE itself to replace the second time derivative with quantities that *can* be computed from the initial conditions.

Specifically, solving (12) for d^2u/dt^2 we have

$$\frac{d^2u}{dt^2} = \epsilon\left(1 - u^2\right)\frac{du}{dt} - u \tag{25}$$

Substituting this into (24), and neglecting the $O(\Delta t^3)$ and higher order terms, we find

$$u_2 = u(\Delta t) = u(0) + \Delta t\frac{du}{dt}(0) + \frac{1}{2}\Delta t^2\left[\epsilon\left(1 - u(0)^2\right)\frac{du}{dt}(0) - u(0)\right] \tag{26}$$

Then using the initial conditions (13) and (14) we can rewrite (26) as

$$u_2 = u_0 + \Delta t\, v_0 + \frac{1}{2}\Delta t^2\left[\epsilon\left(1 - u_0^2\right)v_0 - u_0\right] \tag{27}$$

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 (22) 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:

$$q \equiv \frac{1}{2}\epsilon\Delta t\left(1 - u_n^2\right) \tag{28}$$

then (22) implies that

$$u_{n+1} = (1 - q)^{-1}\left[\left(2 - \Delta t^2\right)\, u_n - (1 + q)\, u_{n-1}\right] \quad n + 1 = 3, 4, ...n_t \tag{29}$$

In summary, equations (29), (23) and (27) 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.

3.2 The problem *per se*

Make the directory `hw3/a3`, 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, v0, 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 (17), (16) and (15), respectively.
- `u0` and `v0`: The initial values $u(0)$ and $du/dt|_{t=0}$.
- `epsilon`: The control parameter ϵ.

You do not have to do any checks of the validity of the input arguments.

Your implementation of `vdp` must define the 3 output arguments 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 (23) and (27) and (29).
- `dudt`: A row vector of length n_t containing a finite difference approximation of $du/dt(t^n)$ computed using formula (20) 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 (14), 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/hw3/tvdp.m
```

and, again you are welcome to use it as an aid in designing your own script and/or interactive experimentation to develop and test your implementation of `vdp`. (Remember, though, do *not* name your script file `tdvp.m`!!) In particular, you are urged to make use of `octave`'s plotting facilities in your development script, since 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 which is also implemented in `tdvp.m`.

Once you are convinced that your `octave` function for approximately solving (12)–(14) 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 ($-5 \le v0 \le 5$ 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 `tdvp.m`, 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 `hw3/a3/README`.

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

```
>> tvdp
tvdp
These calculations will take some time (approximately 10 seconds).
Please be patient!
Type 'Enter' to continue:
Type 'Enter' to continue:
```

```
These calculations will take a few more seconds.
Please be patient!
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.

File Inventory for directory `hw3/a3`

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

And that's it for Homework 3! Whew!!