PHYS 210: Introduction to Computational Physics Fall 2012 Homework 3
Due: Thursday, November 8, 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 two `octave` functions to solve tasks as described in the problem specifications.
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` and `tfdas` 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 be able to execute the instructor-supplied driver scripts in your `octave` session, ensure that the following line is in your `~/.octaverc` file.

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

   This should be the case if you have followed the instructions about `~/.octaverc` that we discussed in the labs.

   If you don't have an `~/.octaverc` file, copy it from `~phys210` as follows:

   ```
   % cp ~phys210/.octaverc .
   ```
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. Since both of the driver scripts produce Postscript files, recall that you can use `evince` and/or `okular` to view such files. Please use one of these viewers to ensure that all of the files that are to contain Postscript versions of plots, actually *do* contain Postscript plots. **I recommend that you use `evince`, especially for Problem 1.**
6. As usual, it is your responsibility to ensure that when your homework is complete, all requested files are in their correct locations with the correct names, and that **all code executes properly on the lab machines.**

   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`.
7. **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.
8. **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.
9. 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 (assuming that you *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 50 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 $\overline{V_iV_p}$ between the active point, V_p, and the randomly chosen triangle vertex, V_i.
3. Bisect $\overline{V_iV_p}$ to define a new point, V_q, that thus lies exactly at the midpoint of $\overline{V_iV_p}$.
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:
Type 'Enter' to continue:
This make take a few seconds ...
Please be patient!
Type 'Enter' to continue:

Done!!
```

Here (as well as in the subsequent problem) 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 check in the working directory for various `.ps` (Postscript) files that should have been generated, ensuring that they do contain plots. **You should use `evince` to do this, and look at the thumbnails, since the graphed points on some of the plots will be very difficult to discern if you use `okular`.** If you don't see a thumbnail when `evince` starts, select `View -> Side Pane` from the pull down menus.

You should also see various plots appear 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, $x_{\min} \le x \le x_{\max}$

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

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

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

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 introduced 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, $x_{\min}$, of the finite difference grid.
- `xmax`: Maximum coordinate, $x_{\max}$, 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 $n_x = 2^{\text{level}} + 1$ 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`