PHYS 410: Computational Physics Fall 2020
Homework 1
Due: Monday, October 5, 11:59 PM

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

General Notes

1. Refer to `http://laplace.phas.ubc.ca` for general information concerning homeworks/projects and the write-ups that must accompany the submission of each.
2. Your functions and scripts do not have to be extensively documented. For functions, an initial comment block that describes the input and output arguments is recommended, and you are free to use the comments below for that purpose.
3. Roots should be calculated to a mimimum of 10^{-12} relative precision.
4. Recall that you are welcome to discuss this and other assignments with your classmates, but that *any work you submit, including your writeup and any and all source code, must be original to you.*

Problem 1

Implement a hybrid algorithm that uses bisection and Newton's method to locate a root within a given interval $[x_{\min}, x_{\max}]$. Your top-level algorithm should be implemented as a function with the header

```
function x = hybrid(f, dfdx, xmin, xmax, tol1, tol2)
```

where the arguments to the routine are defined as follows:

```
%  f:    Function whose root is sought.
%  dfdx: Derivative function.
%  xmin: Initial bracket minimum.
%  xmax: Initial bracket maximum.
%  tol1: Relative convergence criterion for bisection.
%  tol2: Relative convergence criterion for Newton iteration.
```

The single output argument is given by

```
%  x:    Estimate of root.
```

Given the initial bracket (interval) $[x_{\min}, x_{\max}]$ such that

$$f(x_{\min})f(x_{\max}) < 0$$

your implementation should perform bisection until the root has been localized to a *relative* accuracy of `tol1`. Your code should then perform Newton iterations until the root has been determined to a *relative* tolerance of `tol2`.

Note that in MATLAB functions can be passed to other functions as arguments (e.g. `f` and `dfdx` above) using *function handles*, as in the following:

```
function fx = f(x)
   fx = cos(x)^2;
end

function val = caller(some_fcn, x)
   val = some_fcn(x);
end

result = caller(@f, 2.0)
```

Here, `@f` is a function handle and `result` will be assigned the value $\cos(2)^2$. In brief, to pass a function to another function, simply prepend a `@` to the function name in the argument list.

Test your implementation by determining all roots of the function

$$f(x) = 512x^{10} - 5120x^9 + 21760x^8 - 51200x^7 + 72800x^6 - 64064x^5 + 34320x^4 - 10560x^3 + 1650x^2 - 100x + 1$$

in the interval $[0, 2]$.

I leave it to you to determine how to choose the initial intervals for `hybrid`, but a brute force approach will suffice. Also, your solution may comprise more than one function—i.e. more functions than `hybrid` alone.

Problem 2

Implement a d-dimensional Newton iteration, but where the Jacobian matrix of the nonlinear system of equations is computed approximately using a finite-difference technique. Specifically, for

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_d(\mathbf{x})] = \mathbf{0}$$

approximate the Jacobian element $J_{ij}(\mathbf{x}) \equiv [\partial f_i/\partial x_j](\mathbf{x})$ using the difference quotient

$$\frac{f_i(x_1, x_2, \ldots, x_{j-1}, x_j + h, x_{j+1}, \ldots, x_d) - f_i(\mathbf{x})}{h}$$

where h is a specified parameter.

As part of your implementation, code a function with header

```
function x = newtond(f, jacfd, h, x0, tol)
```

where the input arguments are defined by

```
%  f:     Function which implements the nonlinear system of equations.
%         Function is of the form f(x) where x is a length-d vector, and
%         returns length-d column vector.
%  jacfd: Function which is of the form jacfd(f, x, h) where f is the above
%         function, x is a length-d vector, and h is the finite difference
%         parameter. jacfd returns the d x d matrix of approximate Jacobian
%         matrix elements.
%  h:     Finite differencing parameter.
%  x0:    Initial estimate for iteration (length-d column vector).
%  tol:   Convergence criterion: routine returns when relative magnitude
%         of update from iteration to iteration is <= tol.
```

and the output argument is

```
%  x:    Estimate of root (length-d column vector)
```

Use your implementation to find a root of the system

$$x^2 + y^4 + z^6 = 2$$
$$\cos(xyz^2) = x + y + z$$
$$y^2 + z^3 = (x + y - z)^2$$

starting from an initial estimate $\mathbf{x}^{(0)} = (x, y, z) = (-1.00, 0.75, 1.50)$ and using $h = 10^{-5}$.

Note that your implementation of `jacfd` (or whatever you wish to call it) does not have to compute the approximate Jacobian matrix for a general system of d equations: i.e. it can be specialized to the $d = 3$ case under consideration.

Your writeup should contain a brief description of the observed convergence behaviour of your solver for the above system and initial estimate. Here, you may wish to "trace" the approximate root values and residuals at each Newton step by removing appropriate semi-colons from your source code.