SESG6025 Coursework: Non-linear Equations and Fractals

Due: Wednesday 12th January 2010 (12 midday). 20% weighting.

Aims: This coursework uses Newton-Raphson iteration on a single non-linear equation and a system of equations to improve the accuracy of an initial guess of the solution. From some initial guesses, the Newton-Raphson method may not converge to a real root of a non-linear equation: this is explored. A simple multi-dimensional non-linear equation solver is also written.

Objectives: Produce a plot of a simple fractal using Matlab or Python. Write a multi-dimensional non-linear equation solver in Matlab or Python.

Requirements and hand-in: You should email two files (together as attachments to a single email) to sesg6025@soton.ac.uk with the email subject "**Coursework**"

File 1: (Question 1). The file name must be "**fract.m**" or "**fract.py**"

File 2: (Question 2). The filename must be "**solver.**m" or "**solver.py**"

These files, when executed within Matlab/ Python, give the results required for each part of questions (1) and (2) on screen. Plots should be in separate figure windows.

Before attempting the questions, read the Matlab `help` on:

`for, format, i, meshgrid, shading, surf, colormap, view, feval`

Similar commands exist in Python (see course notes on Python and e.g. http://matplotlib.sourceforge.net/ for Python Plotting using Matlab commands.)

1) Non-linear equation solving

a) Using the starting value of z = 1.4, write a piece of Matlab/Python code to perform 5 Newton-Raphson steps to find a better solution of the equation

$$z^3 - 1 = 0 \qquad (1)$$

(The output should be to the screen when the .m / Python file is executed.)

b) In general there is no restriction on whether we start with a real value or a complex value as the initial guess. Give the 5 values which you obtain if you start with z = -1.0 – 1.5 i, where $i = \sqrt{-1}$. (Output results to the screen)

c) For equation (1) there are three roots at z = 1, -0.5 ± 0.866 i. You can check this using:

```
» roots([1,0,0,-1])

ans =

 -0.50000000000000 + 0.86602540378444i
 -0.50000000000000 - 0.86602540378444i
  1.00000000000000
```

In this section we repeat (b), but instead focus on which complex starting values converge to the root z = –1 to the equation in (1a). The solution will be in the form of a map

Set up a *grid* of 100 equally spaced x and y values in the range [-2, 2]. You should use the `linspace` and `meshgrid` functions (or equivalents). Set up a new array `z = x + i*y`, where i = $\sqrt{-1}$ and for each point in this array perform 10 iterations of the iteration scheme you derived in Q1. Form a second array, `colour`, as follows:

$$\texttt{colour(j,k)} = \begin{cases} 1 & |z_{10} - 1| \le 1 \times 10^{-6} \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

where `z0(j,k) = x(j,k) + i* y(j,k)` is the initial z value (you may find section 1.8 in the Matlab handout book useful for an easy way to set up the array `colour`).

Produce a surface whose height is given by the entries in the `colour` array at each `x` and `y` value. The shading for this plot should be interpolated using the 'cool' colormap in Matlab, or a suitable Python colormap. The output figure should be viewed from the top (`view(2)`).

[This part is for information only: The picture which you have produced is a fractal, the points which are shaded in are those which when used a starting values for the Newton-Raphson iteration converge to the root at 1. Other fractals, such as the Mandelbrot set, use a different iteration scheme and the colouring of the point at (x, y) is dependent on how quickly (if at all) the iteration scheme diverges when started from some initial guess $z_0 = x + i \times y$.

A fractal is a shape which has self-similar structure on all scales of magnification. Whilst it is *not* part of the coursework, you can zoom in on a small region of the set (e.g. centred near to the edge of the shaded region) and repeat the iterations to reveal further structure. If you are using a full version of Matlab then you can also use a much finer resolution to produce the plots and more detail will be visible.

For more information on fractals see: Peitgen, Jürgens, and Saupe (1992) "Chaos and Fractals. New Frontiers of Science" (Springer-Verlag).]

2) Multidimensional non-linear equation solver. The hand-in consists of a single '.m' or '.py' file which, as output, gives the solution to part (b) of the question.

a) Write a Matlab/ Python function to determine the roots of a set of non-linear equations. The following skeleton code in Matlab gives the help definition for the code. Your code should have all of the features listed and it should converge when any of the convergence tolerances are met, and an error message should be output to the screen if the maximum number of iterations is exceeded.

```
function [p_out] = solver
p=        ;
delta=    ;
epsilon=  ;
max1=     ;
[p_out,iter] = newton_md('F', 'JF', p, delta, epsilon, max1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [p, iter] = newton_md(F, JF, p, delta, epsilon, max1)
% Input – F is the system defined by the function F
%       - JF is the jacobian of F defined by the function JF
%       - p is the initial approximation to the solution
%       - delta is the convergence tolerance for P
%       - epsilon is the convergence tolerance for F(P)
%       - max1 is the maximum number of iterations
% Output – p is the output approximation to the solution
%        - iter is the number of iterations required
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [F_out] = F(p)
F_out = ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [JF_out] = JF(p)
JF_out = ;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

b) Use your procedure to solve the following system to > 7 decimal places

$$\begin{aligned} x^2 - 2x - y &= -0.5 \\ x^2 + 4y^2 &= 4 \end{aligned},$$

given an initial guess of (2, 0.25). If the solution is not found within 100 iterations the procedure should terminate.

Please ask if you need help.

Prof Simon Cox, sjc@soton.ac.uk. Building 25/2037 Phone Ext 23116.