Lab2B
Newton's Method, Error, Newton's Fracal and Explorations

Lab1B, Lab2B, Lab3B, Lab4B will a series of Chapters about Numerical Analysis and its applications.

1 Title, Team Members, Abstract and Introduction

See the instructions for Lab1B.

2 Implement Newton's method and test your code

In this lab you will be implementing and applying Newton's method for solving nonlinear equations of the form:

$$f(x) = 0\,. \tag{1}$$

As you work through the lab you should use some examples which you know will work with each of the methods.

2.1 Implement Newton's method

Newton's method is a fixed point iteration (FPI) method, so here you use your work from Lab2A on FPI to help implement Newton's method:

$$\Delta x_k = -\frac{f(x_k)}{f'(x_k)} \tag{2}$$

$$x_{k+1} = x_k + \Delta x_k \tag{3}$$

Note that in this Lab you will be storing values in vectors with components x_k (or in Matlab notation $x(k)$). The header below outlines what your code should take as input and output:

```
function [x,flag]=mynewton(f,fx,x0,tol,maxiter)
% Function file: mynewton.m
% Author:
% Date:
% Purpose: Compute approximate solution to f(x)=0 via Newton's method
%
% Input arguments:
%   f -- A function handle for the function f(x) being solved.
%   fx -- A function handle for the derivative f'(x)
%   x0 -- Initial guess for the solution
%   tol -- Tolerance wanted in the solution.
%   maxiter -- Maximum number of iterations.
%
% Output arguments:
%   x -- Vector containing the iterates of the Newton's method.
%   flag -- Flag specifying if solution obtained within the tolerance, it
%           should take the values:
%           = The number of iterations taken to converge.
%           = -1  If the algorithm has not converged in maxiter iterations.
%
```

Note that you should implement this incrementally, not trying to add all the conditions at once, since this is much harder to debug. For example just start with the main loop and the maxiters stopping condition.

2.2 Testing your code

To ensure that your function `mynewton` is working correctly, you should test the function on some problems which you know the solution for. For example, the function

$$f(x) = (x+1)(x-1/2)$$

has roots at $x = -1$ and $x = 1/2$, and the derivative is

$$f'(x) = 2x + 1/2\,.$$

Thus, the MATLAB code

```
f=@(x)(x+1).*(x-1/2);
df=@(x)2*x+1/2;
[x1,fl]=mynewton(f,df,-1.2,0.001,10);
[x2,fl]=mynewton(f,df,0.6,0.001,10);
```

should give output vectors $x1$ and $x2$ of the iterates that yield $x_1 \approx -1$ and $x_2 \approx 0.5$.

2.3 Determine order of convergence and use log-log plot

You will also need to look at the order of convergence α, as discussed in lectures, we have

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|^\alpha} = \frac{|e_{k+1}|}{|e_k|^\alpha} \approx C \tag{4}$$

multiplying by the denominator and taking logarithms gives,

$$\log_e |e_{k+1}| \approx \alpha \log_e |e_k| + \log_e (C) \tag{5}$$

which is a linear model in $\log_e |e_k|$ with slope α and intercept $\log C$.

For your Newton method do the following, on your example:

- Plot the logarithms of the errors against each other.
- **By hand** draw what you think is the line which best fits the data.
- Estimate the slope of the line (the order of convergence) and compare it to the expected value. Do the different schemes meet expectations for their relative order of convergence? Why or why not?

2.4 Method to approximate and exploit the order of convergence α

The analysis in the previous subsection assumes that we have the exact solution x^* available in order to determine the order of convergence α. But in practise we would like to approximate the order of convergence from the iterates x_k during the execution of Newton's method, without knowing x^*. For well-conditioned problems with no multiplicities in their roots, we know $\alpha = 2$ for Newton's method and $C \approx |f''(x^*)/(2f'(x^*))|$ when we are close to a root. In particular the

convergence is quadratric in this case. Give and justify a method based on differences between iterates x_k (or equivalently the stable digits) that approximately detects quadratic convergence of Newton's method. Illustrate your approach with a couple of examples.

Similarly give a method based on iterates x_k produced by Newton's method to detect when its converging to a multiple root. How would you approximately determine the multiplicity m of such a root? Finally how would you improve Newton's method to more efficiently and accurately converge to the multiple root? Give an example to illustrate your approach.

3 Newton's fractals

The following material comes from the excellent Text, Numerical Methods for Engineers, by Jeffrey Chasnov.

Your task for section 3.1 is:

3.1 Find the solutions of $z^4 = 1$ as complex numbers and illustrate them in terms of exp and the graph below following the process in 3.1 below

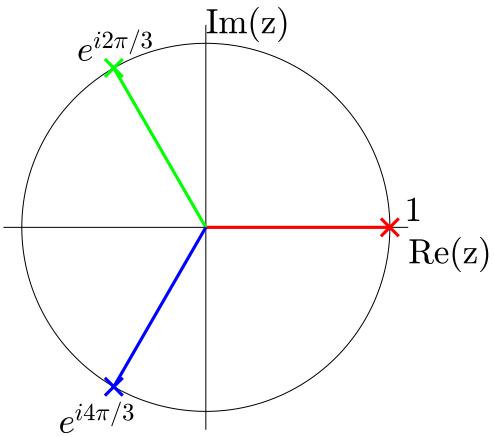
3.1 Roots in the Complex Plane

View this lecture on YouTube

The three cube roots of unity are the solutions of $z^3 = 1$. We can solve this equation using Euler's formula, $\exp(i\theta) = \cos\theta + i\sin\theta$. We write $z^3 = \exp(i2\pi n)$, with n an integer, and take the cube root to find the solutions $z = \exp(i\,2\,\pi n/3)$. The three unique complex roots correspond to $n = 0, 1$ and 2, and using the trig values of special angles, we find

$$r_1 = 1, \qquad r_2 = -\frac{1}{2} + \frac{\sqrt{3}}{2}i, \qquad r_3 = -\frac{1}{2} - \frac{\sqrt{3}}{2}i.$$

These roots can be located on the unit circle in the complex plane, and are shown below.



We now define the root-finding problem to be $f(z) = 0$, where $f(z) = z^3 - 1$, with derivative $f'(z) = 3z^2$. Newton's method for determining the three complex roots of unity is given by the iteration

$$z_{n+1} = z_n - \frac{f(z)}{f'(z)}.$$

Here, we want to determine which initial values of z_0 in the complex plane converge to which of the three cube roots of unity. So if for a given z_0, the iteration converges to r_1, say, we will color red the point z_0 in the complex plane; if it converges to r_2, we will color z_0 green; and if to r_3, blue.

In the next lecture, we show you how to construct a MATLAB program that grids up a rectangle in the complex plane, and determines the convergence of each grid point z_0. By coloring these grid points, we will compute a beautiful fractal.

3.2 Implement the Matlab below for $z^4 = 1$. Discuss aspects of the implementation. Produce the Fractal

Watching the 2 excellent videos by Chasnov will also be helpful. See Section 3.2 on the next page.

3.2 Coding the Newton fractal

View this lecture on YouTube

Our code first defines the function $f(z)$ and its derivative to be used in Newton's method, and defines the three cube roots of unity:

```
f = @(z) z.^3-1; fp = @(z) 3*z.^2;
root1 = 1; root2 = -1/2 + 1i*sqrt(3)/2; root3 = -1/2 - 1i*sqrt(3)/2;
```

We let $z = x + iy$. The complex plane is represented as a two-dimensional grid and we define `nx` and `ny` to be the number of grid points in the x- and y-directions, respectively. We further define `xmin` and `xmax` to be the minimum and maximum values of x, and similarly for `ymin` and `ymax`. Appropriate values were determined by numerical experimentation.

```
nx=2000; ny=2000;
xmin=-2; xmax=2; ymin=-2; ymax=2;
```

The grid in x and y are defined using `linspace`.

```
x=linspace(xmin,xmax,nx); y=linspace(ymin,ymax,ny);
```

We then use `meshgrid` to construct the two-dimensional grid. Suppose that the x-y grid is defined by `x=[x1 x2 x3]` and `y=[y1 y2 y3]`. Then `[X, Y]=meshgrid(x,y)` results in the matrices

$$\mathtt{X} = \begin{bmatrix} \mathtt{x1} & \mathtt{x2} & \mathtt{x3} \\ \mathtt{x1} & \mathtt{x2} & \mathtt{x3} \\ \mathtt{x1} & \mathtt{x2} & \mathtt{x3} \end{bmatrix}, \qquad \mathtt{Y} = \begin{bmatrix} \mathtt{y1} & \mathtt{y1} & \mathtt{y1} \\ \mathtt{y2} & \mathtt{y2} & \mathtt{y2} \\ \mathtt{y3} & \mathtt{y3} & \mathtt{y3} \end{bmatrix},$$

and we can grid the complex plane using

```
[X,Y]=meshgrid(x,y);
Z=X+1i*Y;
```

We now iterate Newton's method `nit` times. These lines constitute the computational engine of the code.

```
nit=40;
for n=1:nit
    Z = Z - f(Z) ./ fp(Z);
end
```

We next test to see which roots have converged. We use the logical variables `Z1`, `Z2` and `Z3` to mark the grid points that converge to one of the three roots. Grid points that have not converged are marked by `Z4`, and our convergence criteria is set by the variable `eps`. The function `abs` returns the modulus of a complex number.

```
eps=0.001;
Z1 = abs(Z-root1) < eps; Z2 = abs(Z-root2) < eps;
```

```
Z3 = abs(Z-root3) < eps; Z4 = ~(Z1+Z2+Z3);
```

Finally, we draw the fractal. The appropriate graphing function to use here is `image`, which can color pixels directly. We first open a figure and set our desired colormap. Here, our map will be a four-by-three matrix, where row one of the matrix corresponds with the RGB triplet `[1 0 0]` that specifies red; row two of the matrix `[0 1 0]` specifies green; row three of the matrix `[0 0 1]` specifies blue; and row four of the matrix `[0 0 0]` specifies black. The numbers 1-2-3-4 in our image file will then be colored red-green-blue-black.

```
figure;
map = [1 0 0; 0 1 0; 0 0 1; 0 0 0]; colormap(map);
```

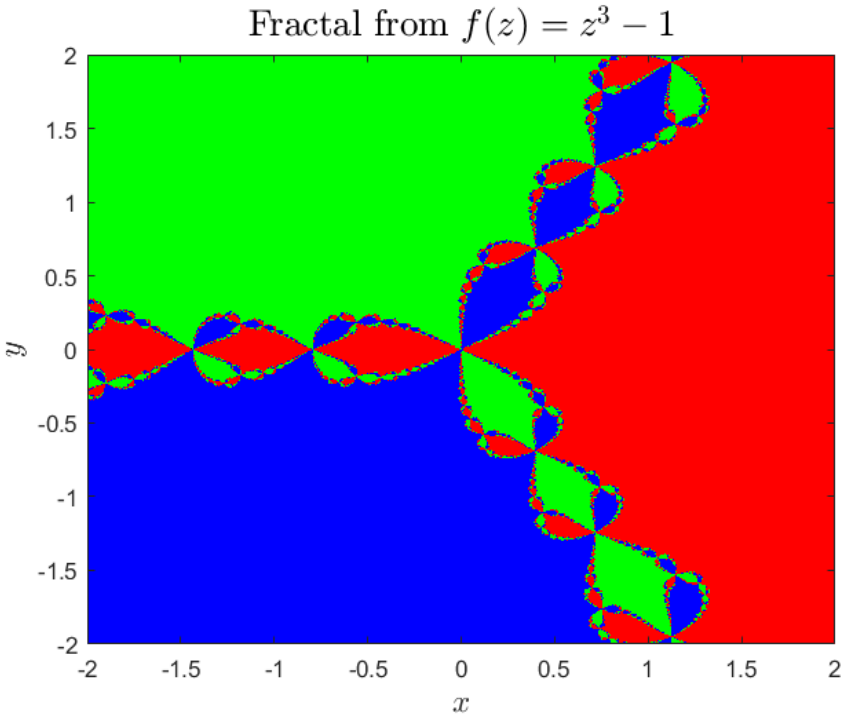
We construct the image file by combining the values of our four `Z` matrices into a single `Z` matrix containing elements 1, 2, 3, or 4.

```
Z=(Z1+2*Z2+3*Z3+4*Z4);
```

The graph is created in our final lines of code. We use the limits of x and y to specify the location of our image in the complex plane. One needs to be aware that the function `image` assumes that the first row of pixels is located at the top of the image. So by default, `image` inverts the y-axis direction by setting the 'YDir' property to 'reverse.' We need to undo this when plotting data from a computation because the usual convention is for the first row of pixels to be at the bottom of the image. We therefore set the 'YDir' property to 'normal.'

```
image([xmin xmax], [ymin ymax], Z); set(gca,'YDir','normal');
xlabel('$x$', 'Interpreter', 'latex', 'FontSize',14);
ylabel('$y$', 'Interpreter', 'latex', 'FontSize',14);
title('Fractal from $f(z)=z^3-1$', 'Interpreter', 'latex','FontSize', 16)
```

The resulting plot looks like



3.3 Create your own fractal and exploration