计算物理

Lecture 5

傅子文

Today's Lecture

- Functions and roots: Part I
  - Polynomials
  - General rooting finding
    - Bracketing & the bisection method
    - Newton-Raphson method

Functions & roots

- This is probably the first numerical task we encounter
  - Many graphical calculators have built-in routines for finding roots

- Solving the roots of $f(x)=0$ is a problem frequently encountered in physics

- First examine solutions for polynomials:

$$f_n(x) = \sum_{i=0}^{n} a_i x^i = 0 \quad \text{with } a_n = 1$$

$$n=1:\ f_1(x) = a_0 + x \quad ; \quad x = -a_0 \text{ is the only root}$$

$$n=2:\ f_2(x) = a_0 + a_1 x + x^2 \ ; x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_0}}{2}$$

Higher orders: cubic

Cubic formula was discovered by Tartaglia (1501-1545) but published by Cardan (~1530) *without permission!*

$$n=2:\ f_3(x)=a_0+a_1x+a_2x^2+x^3=(x-x_1)(x-x_2)(x-x_3)$$

$$x_1=\frac{1}{3}(-a_2+\delta_++\delta_-)$$

$$x_2=\frac{1}{3}(-a_2+\omega^2\delta_++\omega\delta_-)$$

$$x_1=\frac{1}{3}(-a_2+\omega\delta_++\omega^2\delta_-)\qquad \text{where}$$

$$\delta_\pm=\left(-a_2^3+\frac{9}{2}a_2a_1-\frac{27}{2}a_0\pm i\sqrt{3\Delta}\right)^{1/3}$$

$$\Delta=a_2^2a_1^2-4a_1^2-4a_2^3a_0-27a_0^2+18a_2a_1a_0$$

$$\omega=\frac{1}{2}(-1+i\sqrt{3})\quad ;\quad \mathrm{i}=\sqrt{-1}$$

Higher orders: quartic & higher

- The general solution to a quartic was discovered by Ferrari (1545) but is *extremely* lengthy
- Evariste Galois (1811-1832) showed that there is no general solution to a quintic & higher formulas
- Remember certain higher order equations may be rewritten as quadratics, cubics in x^2 (say) e.g.

$$x^4 + 3x^2 + 1 = (x^2)^2 + 3(x^2) + 1$$

- *What is the best approach for polynomials?*
- Always best to reduce to simplest form by dividing out factors
- If coefficients sum to zero then x=1 is a solution and (x-1) can be factored to give a (x-1)*(polynomial order n-1)
  - Can you think why?
- You may be able to repeatedly factor larger polynomials by inspection
  - Only useful for a single set of coefficients though

Example

- Consider $f(x)=x^5+4x^4-10x^2-x+6=0$
- Firstly sum of coefficients is 0, so $x=1$ is a solution – factor $(x-1)$ by long division of the polynomial
- Hence we find that

  $f(x)=(x-1)(x^4+5x^3+5x^2-5x-6)$
  - Notice that the coefficients in the second bracket sum to 0…
- In fact we can show that further division gives

  $f(x)=(x-1)^2(x+1)(x+2)(x+3)$

$$
\begin{array}{r}
x^4+5x^3+5x^2-5x-6 \\
(x-1)\overline{)\,x^5+4x^4+0x^3-10x^2-x+6} \\
x^5-x^4 \\
\hline
5x^4+0x^3 \\
5x^4-5x^3 \\
\hline
5x^3-10x^2 \\
5x^3-5x^2 \\
\hline
-5x^2-x \\
-5x^2+5x \\
\hline
-6x+6
\end{array}
$$

So regardless of whether an analytic general solution exists, we may still be able to find roots without using numerical root finders…

Advice on using the general quadratic formula

- For the quadratic $f(x)=x^2+a_1x+a_0$ the product of the roots must be a_0 since $f(x)=(x-x_1)(x-x_2)$ where x_1 & x_2 are the roots

- Hence $x_2=a_0/x_1$ so given one root, provided $a_0\neq 0$, we can then quickly find the other root

- Using the manipulation discussed in lecture 3

$$x_1 = \frac{-a_1+\sqrt{a_1^2-4a_0}}{2}\times\frac{-a_1-\sqrt{a_1^2-4a_0}}{-a_1-\sqrt{a_1^2-4a_0}} = \frac{2a_0}{-a_1-\sqrt{a_1^2-4a_0}}$$

If $a_0=0$ then we have divide by zero here.

Then define

$$d = -\frac{1}{2}\left(a_1+\mathrm{sgn}(a_1)\sqrt{a_1^2-4a_0}\right)$$

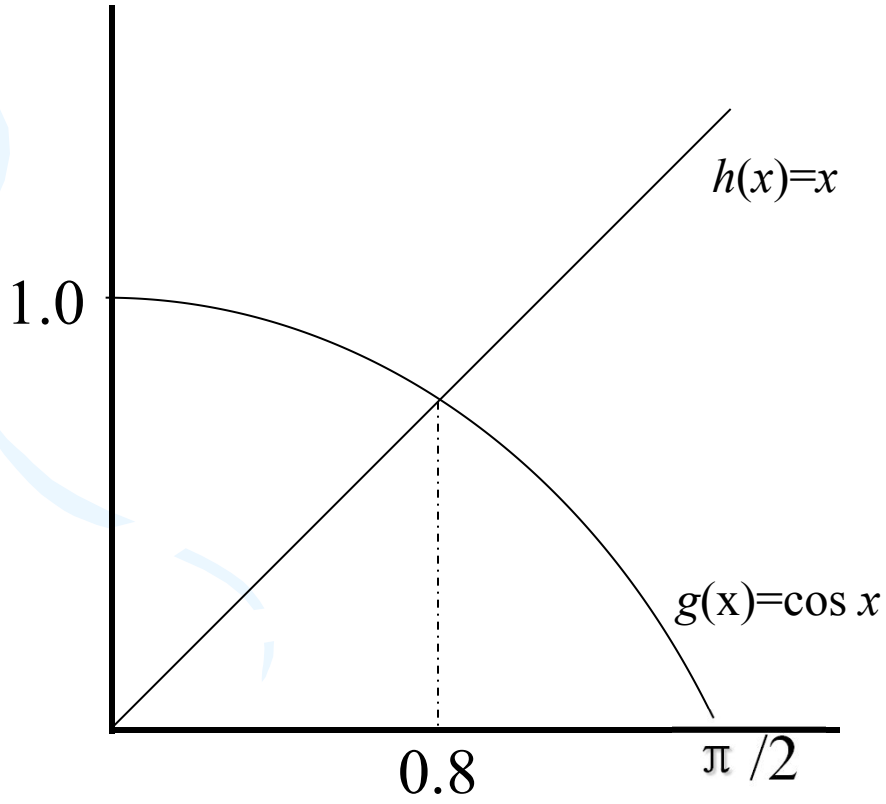
$$\Rightarrow x_1 = a_0/d$$

$$\Rightarrow x_2 = d$$

$\mathrm{sgn}(a_1)$ is a function that is 1 if a_1 is positive, and -1 if a_1 is negative. This ensures that the discriminant is added with the correct sign.

More general root finding

- For a more general function, say $f(x)=cos\ x - x$ the roots are non-trivial



In this case the solution is given where $g(x)=h(x)$

Graphing actually helps a great deal since you can see there must only be one root. In this case around 0.8.

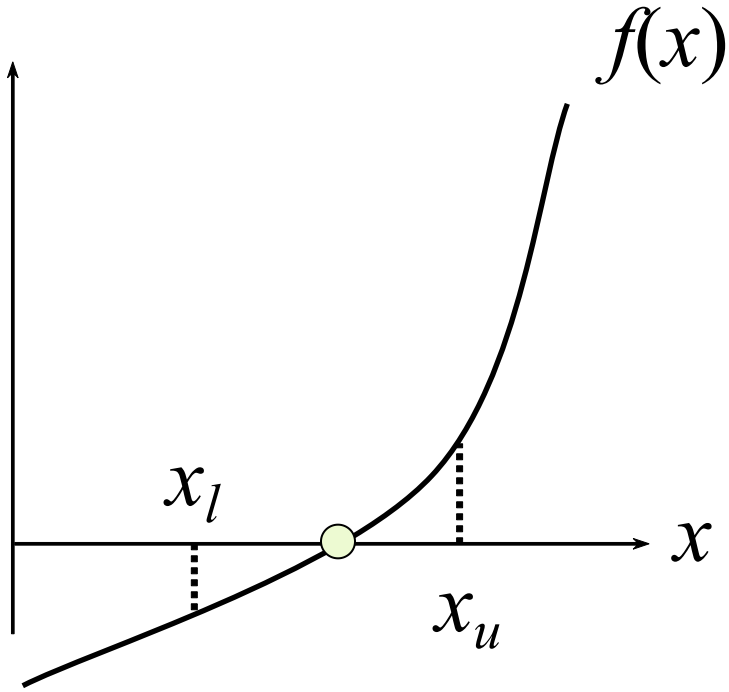
It is extremely beneficial to know what a given function will look like (approximately) – helps you to figure out whether your solutions are correct or not.

Iterative root finding methods

- It is clearly useful to construct numerical algorithms that will take any input $f(x)$ and search for solutions
  - However, there is no perfect algorithm for a given function $f(x)$
    - Some converge fast for certain $f(x)$ but slowly for others
    - Certain $f(x)$ may not yield numerical solutions
- Above all you should not use algorithms as black boxes
  - You need to know what the limitations are!
- We shall look at the following algorithms
  - Bisection
  - Newton-Raphson
  - Secant Method
  - Muller's Method

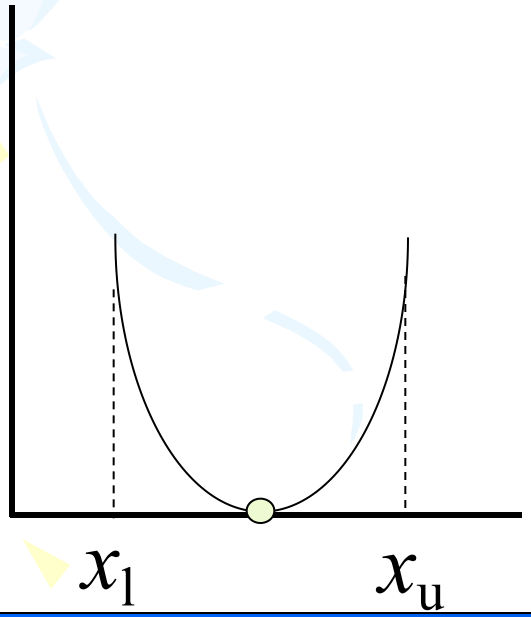
Bracketing & Bisection

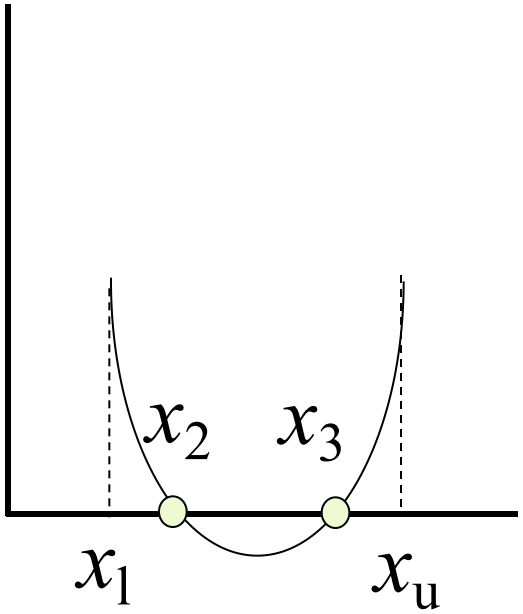
- Bracketing is a beautifully simple idea
- Anywhere region bounded by x_l and x_u, such that the sign of a function $f(x)$ changes between $f(x_l)$ and $f(x_u)$, there must be *at least* one root
  - This is provided that the function does not have any infinities within the bracket – $f(x)$ must be continuous within the region
- The root is said to be *bracketed* by the interval (x_l,x_u)



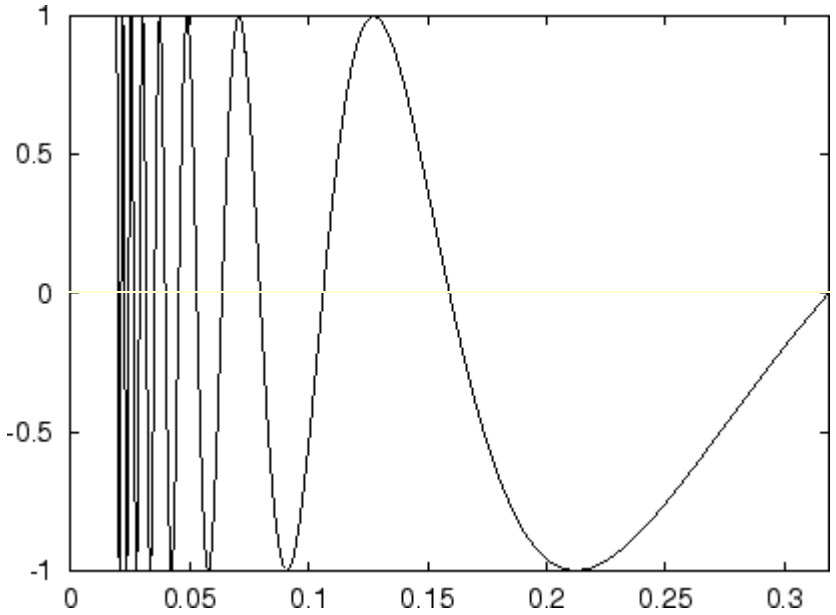
Other possibilities

- Even if a region is bracketed by two positive (or two negative) values a root may still exist
- There could even be $2n$ roots in this region!
  - Where n is a natural number





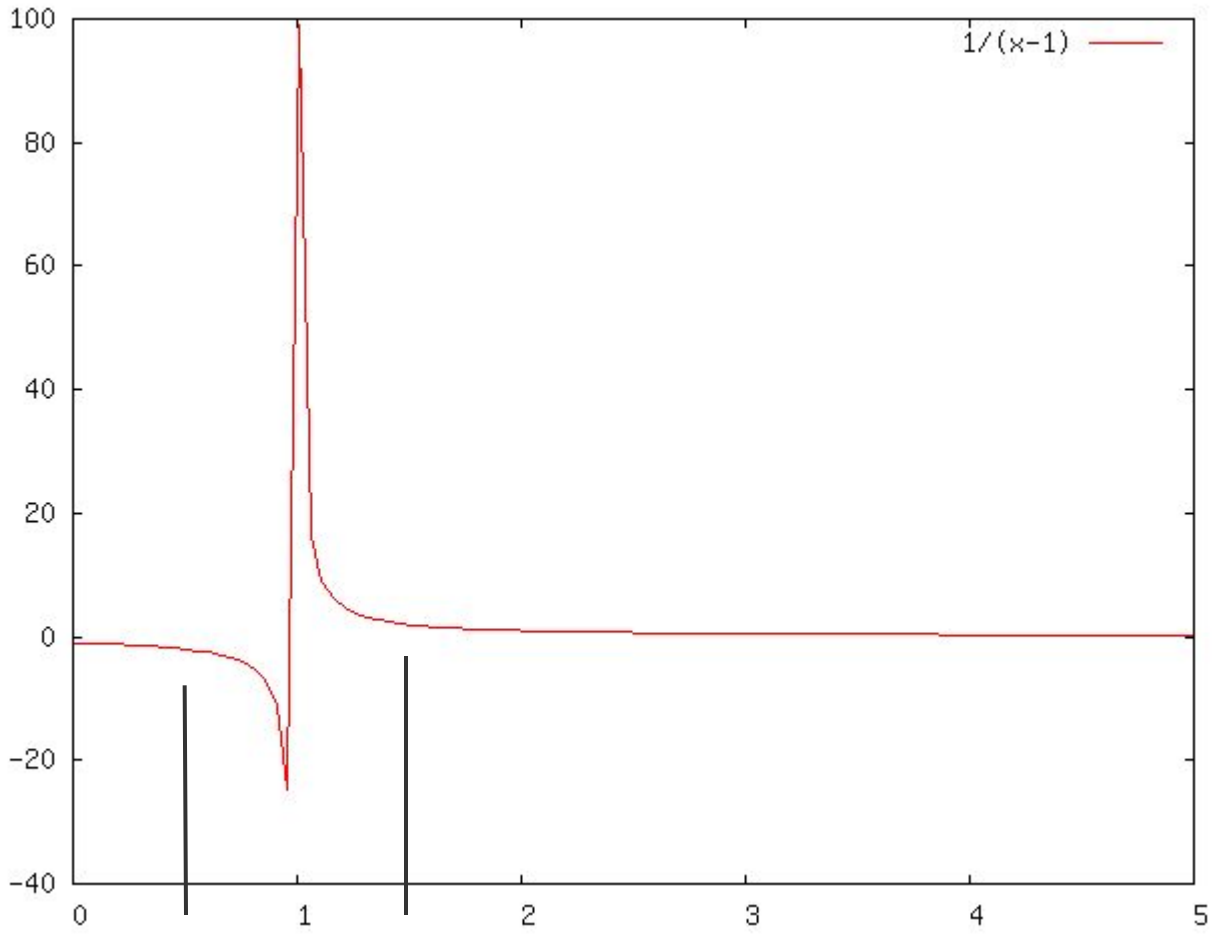
Pathological behaviour occurs



This is a graph of $\sin(1/x)$. As $x \to 0$ the function develops an infinite number of roots.

But don't worry! This is not a course in analysis…

Bracketing singularities

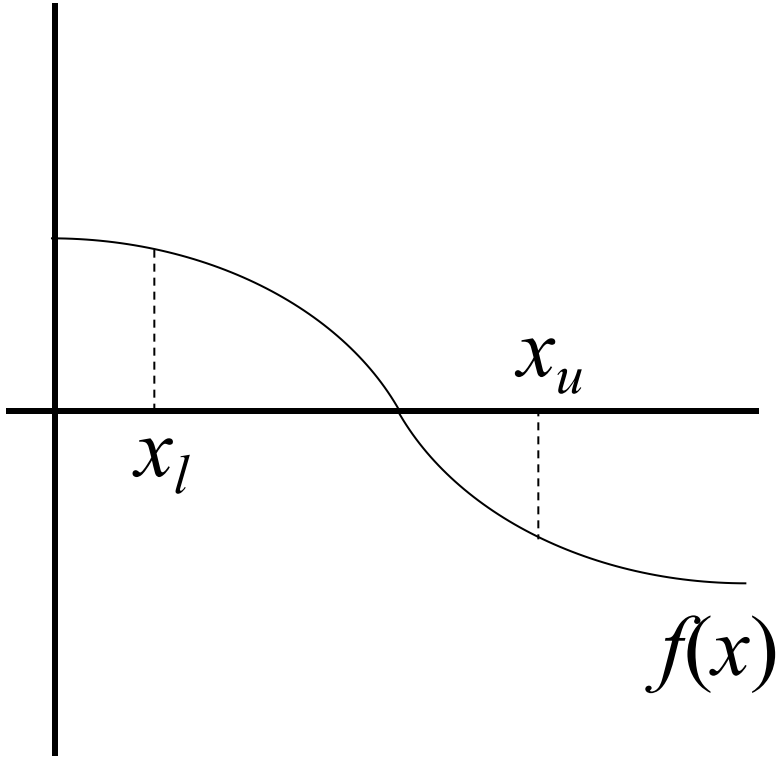


- $f(x)=1/(x-1)$ is not continuous within the bracketed region [0.5,1.5].Although $f(0.5)<0$ and $f(1.5)>0$ there is no root in this bracket (regardless of the fact that the plotting program connects the segments) $f(x)$ is not continuous within the bracket.

Root finding by bisection

Step 1: Given we know that there exists an $f(x_0)=0$ solution, choose x_l and x_u as the brackets for the root

- Since the *signs of* $f(x_l)$ & $f(x_u)$ *must be different*, we must have $f(x_l)\times f(x_u) < 0$
- This step may be non-trivial – we'll come back to it



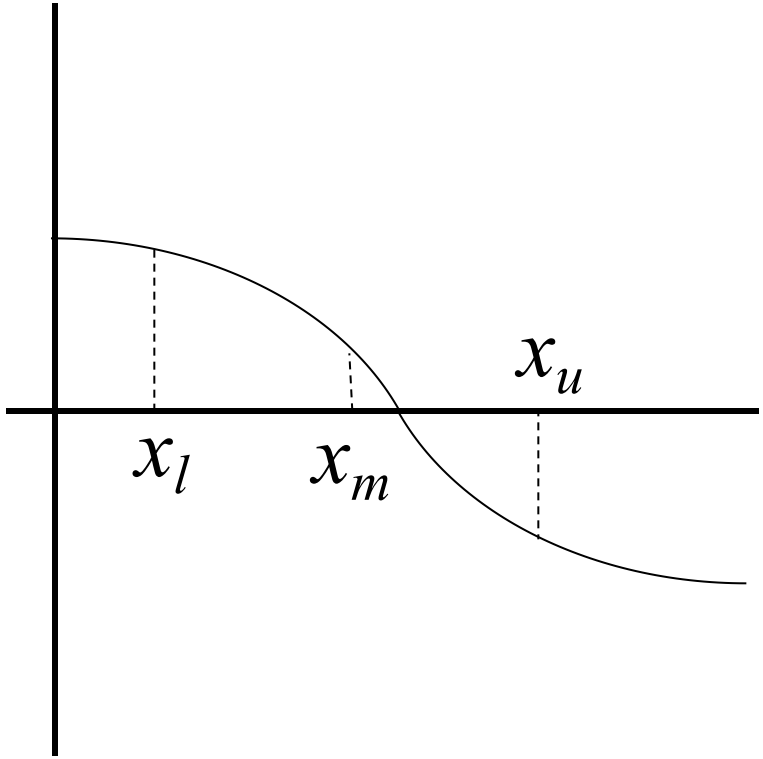
For example, for $f(x)=\cos x - x$ we know that $0<x_0<\pi/2$

Bisection: Find the mid-point

Step 2: Let $x_m=0.5(x_l+x_u)$

– *the mid point between the two brackets*

- This point must be closer to the root than one of x_l or x_u
- The next step is to select the new bracket from x_l, x_m, x_u



Final Step in Bisection

Step 4: *Test accuracy*

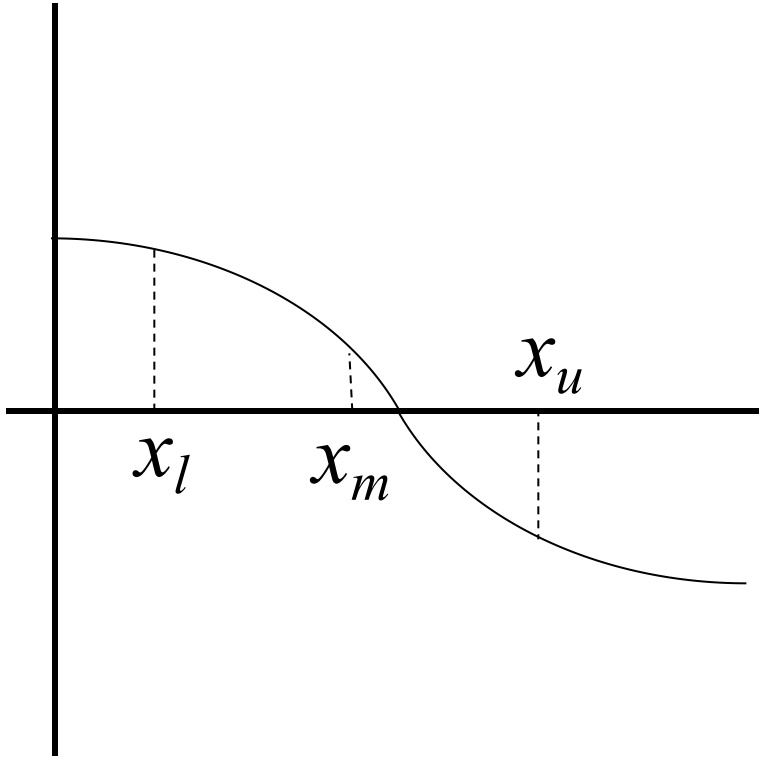
(we want to stop at some point)

- x_m is the best guess at this stage
- Absolute error is $|x_m - x_0|$ but we don't know what x_0 is!
- Error estimate: $e = |x_m^n - x_m^{n-1}|$ where n corresponds to nth iteration
- Check whether $e < d$ where d is the *tolerance* that you have to decide for yourself at the *start* of the root finding
  - If $e < d$ then stop
  - If $e > d$ then return to step 1

Bisection: Find the new bracket

Step 3:

- If $f(x_l)\times f(x_m)<0$
  - root lies between x_l and x_m
  - Set $x_l=x_l$; $x_u=x_m$
- If $f(x_l)\times f(x_m)>0$
  - root lies between x_m and x_u
  - Set $x_l=x_m$; $x_u=x_u$
- If $f(x_l)\times f(x_m)=0$
  - root is x_m you can stop



Example

- Let's consider $f(x)=\cos x - x$
- Pick the initial bracket as $x_l=0$ and $x_u=\pi/2$
  - $f(0)=1$, $f(\pi/2)= -\pi/2$ – check this satisfies $f(0)\times f(\pi/2)<0$ (*yes* - so good to start from here)
- First bisection: $x_m=0.5(x_l+x_u)= \pi/4=0.785398$
  - $f(x_l^{(1)})=f(0)=1$
  - $f(x_m^{(1)})=f(\pi/4)= -0.078$
  - $f(x_u^{(1)})=f(\pi/2)= -\pi/2$

  } $f(x_l^{(1)})\times f(x_m^{(1)})<0$

  - So select $x_l^{(1)}$ & $x_m^{(1)}$ as the new bracket

Repeat…

- After eleventh bisection:
  $x_m^{(11)}=963\pi/4096=0.738612$
- After twelfth bisection:
  $x_m^{(12)}=1927\pi/8192=0.738995$
- So after first 12 bisections we have only got the first three decimal places correct
- Typically to achieve 9 decimal places of accuracy will require 30-40 steps

Pros & cons of bisection

- Pro: Bisection is robust
  - You will *always* find a root
  - For this fact alone it should be taken seriously
- Pro: can be programmed very straightforwardly
- Con: bisection converges slowly…
  - Width of the bracket: $x_u^n - x_l^n = e^n = e^{n-1}/2$
    - Width halves at each stage
  - We know in advance if the initial bracket is width e^0 after n steps the bracket will have width $e^0/2^n$
  - If we have a specified tolerance d then this will be achieved when $n = \log_2(e^0/d)$
  - Bisection is said to converge linearly since $e^{n+1} \propto e^n$

Newton-Raphson Method

- For many numerical methods, the underlying algorithms begins with a Taylor expansion

- Consider the Taylor expansion of $f(x)$ around a point x_0

$$f(x) \cong f(x_0) + (x - x_0) f'(x_0) + \ldots$$

- If x is a root then

$$0 \approx f(x_0) + (x - x_0) f'(x_0)$$

$$\Rightarrow x \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

- This formula is applicable provided $f'(x_0)$ is not an extremum ($f'(x_0) \neq 0$))

Applying the formula

- If we interpret x_0 to be the current guess we use the formula to give a better estimate for the root, x
  - We can repeat this process iteratively to get better and better approximations to the root
    - We may even get the exact answer if we are lucky
- Take $f(x)=\cos x - x$, then $f'(x)=-\sin x - 1$
- It helps to make a good guess for the first root, so we'll take $x_0^{(1)} = \pi/4 = 0.785398$
  - The (1) signifies the first in a series of guesses
  - $f(x_0^{(1)}) = -0.0782914$
  - $f'(x_0^{(1)}) = -1.7071068$

Steps in the iteration

- Plugging in the values for $f(x_0^{(1)}), f'(x_0^{(1)})$ we get

$$x_0^{(2)} \approx x_0^{(1)} - \frac{f(x_0^{(1)})}{f'(x_0^{(1)})} = 0.785398 - \frac{0.0782914}{1.7071068} = 0.739536$$

- We repeat the whole step again, with the values for $x_0^{(2)}$

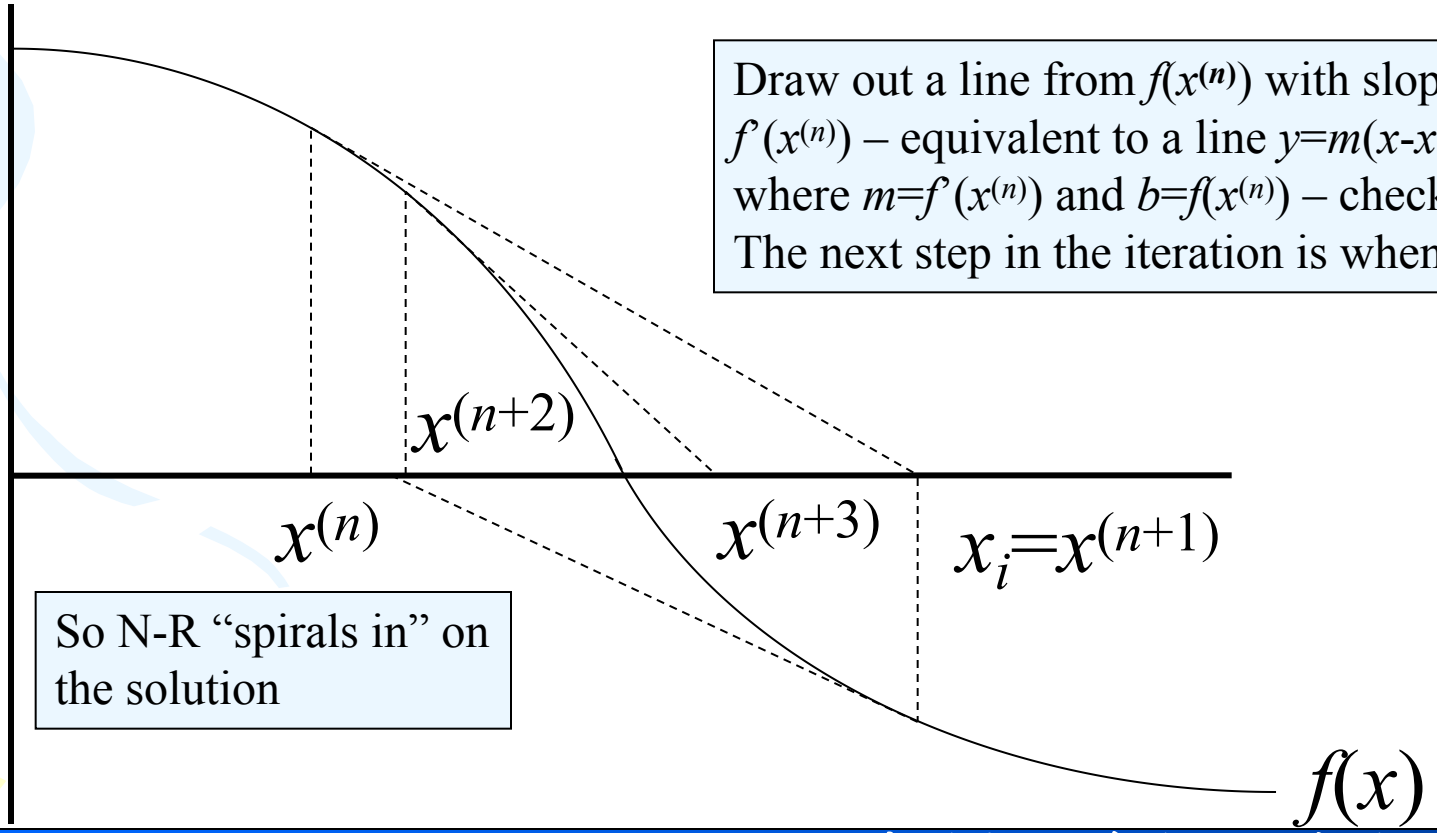
$$x_0^{(3)} \approx x_0^{(2)} - \frac{f(x_0^{(2)})}{f'(x_0^{(2)})} = 0.739536 - \frac{0.0007546}{1.673945} = 0.739085$$

- At this stage $f(x_0^{(3)})$=0.0000002, so we are already very close to the correct answer – after only two iterations
- It helps to see what is happening geometrically…

Geometric Interpretation of N-R

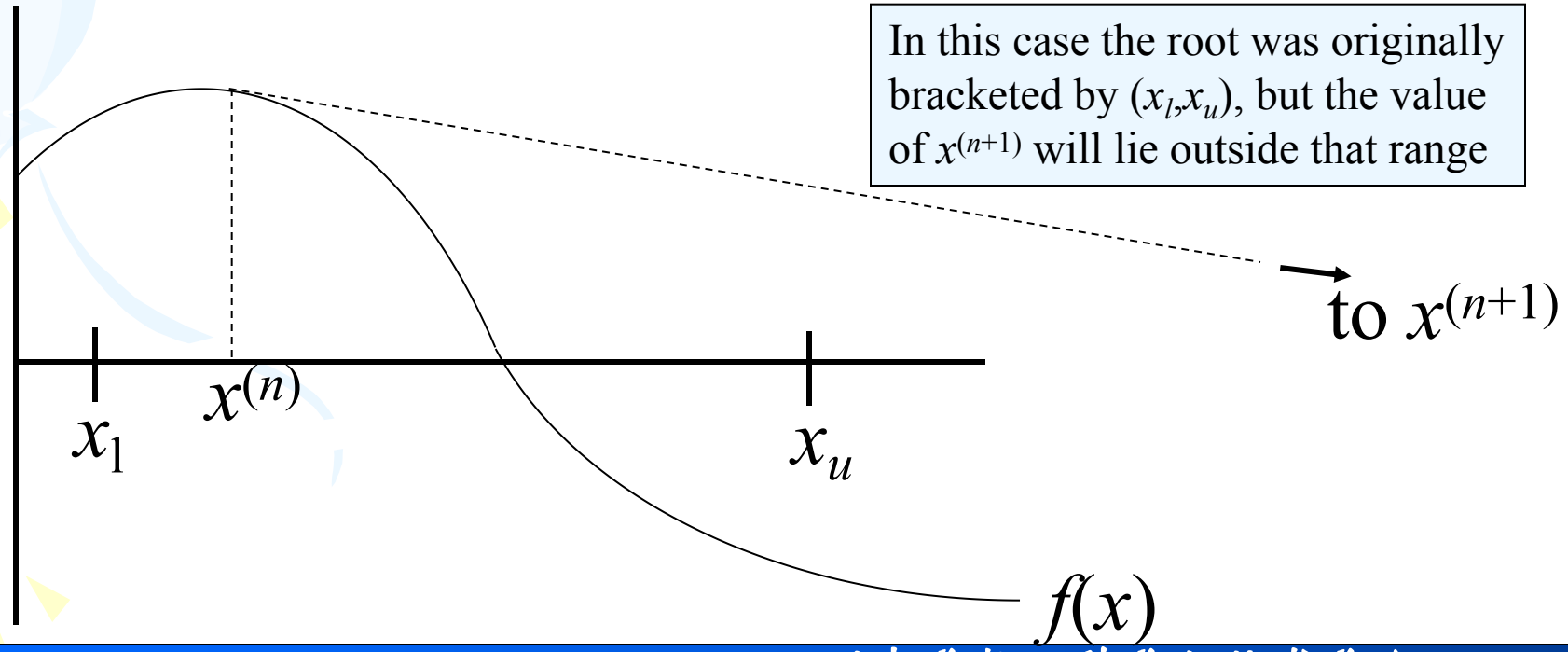
- The general step in the iteration process is

$$x^{(n+1)} \approx x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$$



Problems with N-R

- The primary problem with N-R is that if you are too close to an extremum the derivative $f'(x^{(n)})$ can become very shallow:



NR-Bisection Hybrid

- Let's combine the two methods
  - Good convergence speed from NR
  - Guaranteed convergence from bisection

- Start with $x_l < x^{(n)} < x_u$ as being the current best guess within the bracket

- Let
$$x_{NR}^{(n+1)} \approx x^{(n)} - \frac{f(x^{(n)})}{f'(x^{(n)})}$$

  - If $x_l < x_{NR}^{(n+1)} < x_u$ then take an NR step
  - Else set $x_{NR}^{(n+1)} = x_m^{(n+1)} = 0.5(x_l + x_u)$, check and update x_l and x_u values for the new bracket
    - *i.e.* we throw away any bad NR steps and do a bisection step instead

Example:Full NR-Bisection algorithm

- Let's look at how we might design the algorithm in detail

- Initialization:
  - Given x_l and x_u from user input evaluate $f(x_l)$ & $f(x_u)$
    - Test that $x_l \neq x_u$ – if they are equal report to user
  - Ensure root is properly bracketed:
    - Test $f(x_l) \times f(x_u) < 0$ – if not report to user
  - Initialize $x^{(n)}$ by testing values of $f(x_l)$ & $f(x_u)$
    - If $|f(x_l)| < |f(x_u)|$ then $x^{(n)} = x_l$ and set $f(x^{(n)}) = f(x_l)$
    - else let $x^{(n)} = x_u$ and set $f(x^{(n)}) = f(x_u)$
    - Evaluate the derivative $f'(x^{(n)})$
  - Set the counter for the number of steps to zero: icount=0

Full NR-Bisection algorithm cont.

- (1) Perform NR-bisection
  - Increment icount
  - Evaluate $x_{NR}^{(n+1)}=x^{(n)}-f(x^{(n)})/f'(x^{(n)})$
  - If ($x_{NR}^{(n+1)}<x_l$) or ($x_{NR}^{(n+1)}>x_u$) then require bisection
    - Set $x_{NR}^{(n+1)}=0.5(x_l+x_u)$
  - Calculate error estimate: $e=|x^{(n+1)}-x^{(n)}|$
  - Set $x(n)=x(n+1)$
  - Test error
    - If $|e/x(n)|<$ tolerance go to (2)
  - Test number of iterations
    - If icount>max # iterations then go to (3)
  - Prepare for next iteration
    - Set value of $f(x(n))$
    - Set value of $f'(x(n))$
    - Update the brackets using Step 3 of the bisection algorithm
  - Go to 1

Final parts of the algorithm

- (2) Report the root and stop
- (3) Report current best guess for root. Inform user max number of iterations has been exceeded

- While it is tempting to think that these reporting stages are pointless verbiage – they absolutely aren't!
  - Clear statements of the program state will help you to debug tremendously
  - Get in the habit now!

Newton-Raphson & Fractals

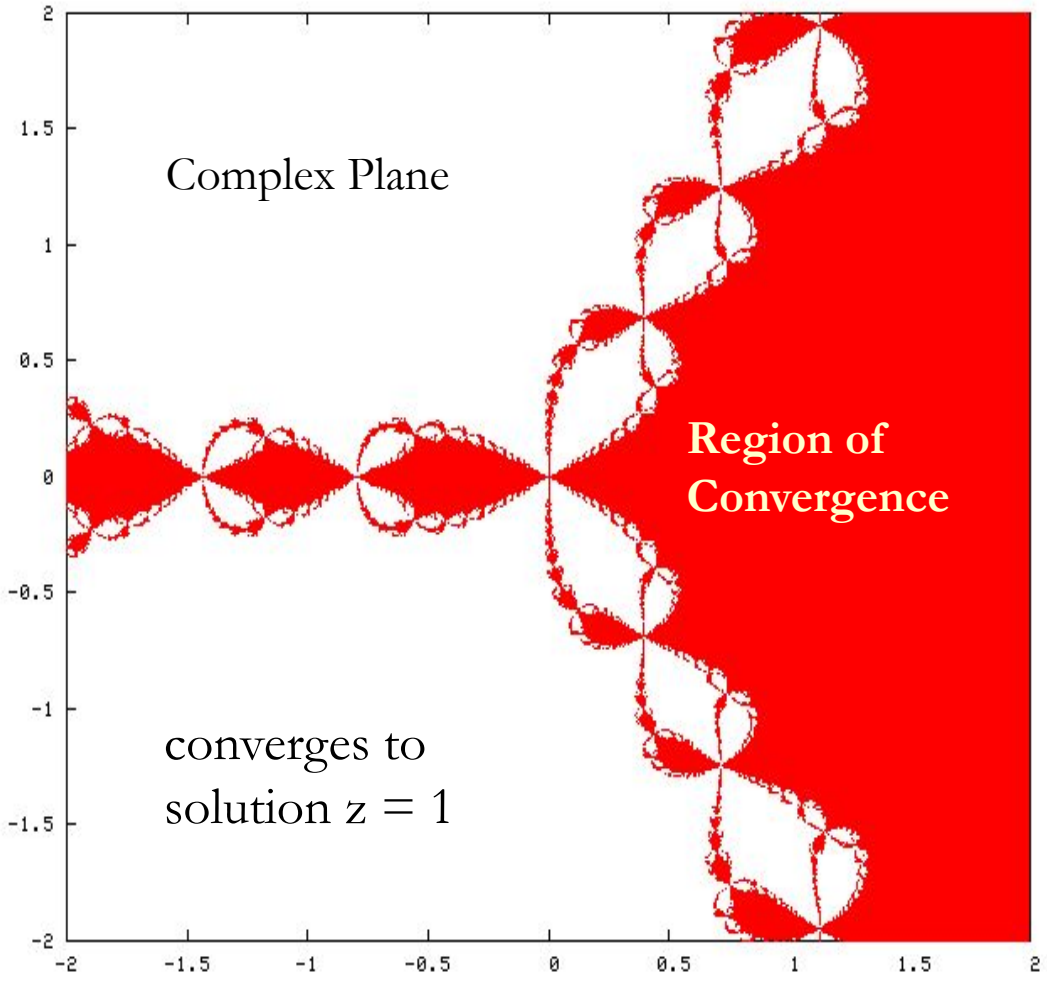
NR also works for Complex Functions

$$f(z)=0$$
$$f(z)=z^3-1=0$$
Roots: $z=1,\ z=e^{\pm 2\pi i/3}$

Newton-Raphson Method

$$z_{j+1}=z_{j+1}-\frac{z_j^3-1}{3z_j^2}$$

Look for Convergence

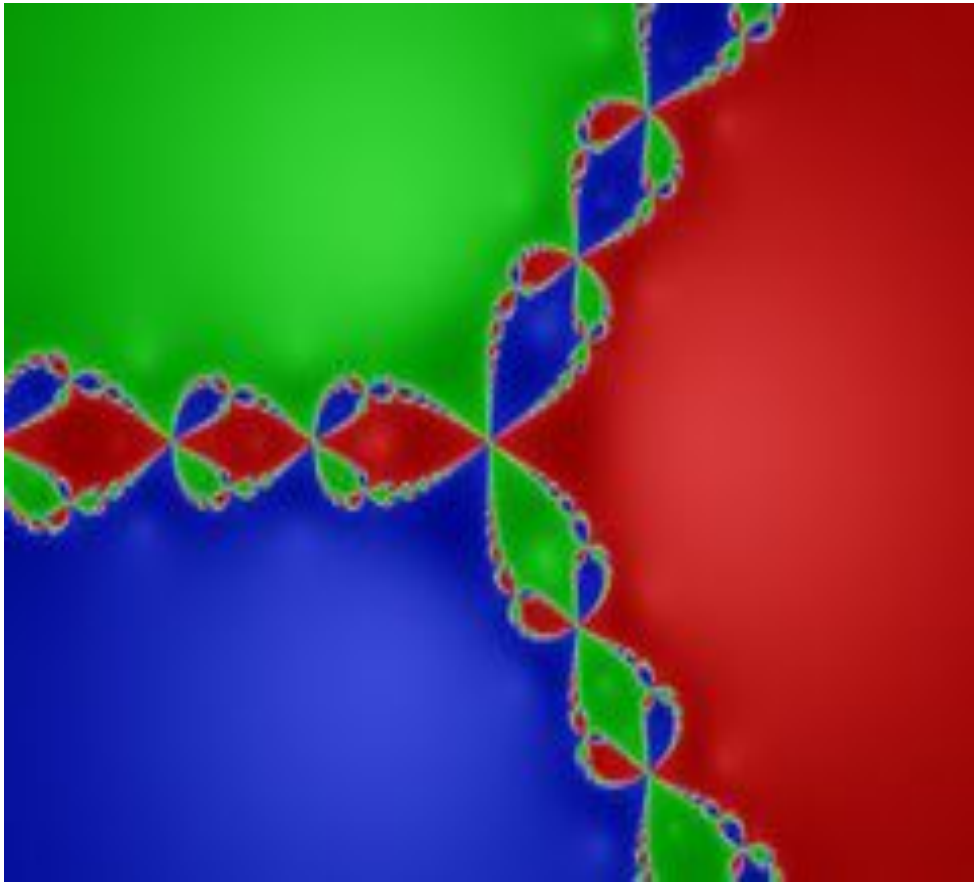


The following fractal images were created by applying the Newton-Raphson Method to various real-valued polynomials.

- Each pixel represents a point in the complex plane used by the NR Method as an initial approximation.

- The color of the pixel indicates which of the roots the point converged to, while the intensity is proportional to the number of iterations required to approximate that root to within a specified tolerance (.0001). Brighter pixels converged to the root more quickly than darker pixels, while white pixels did not converge at all after a specified number of iterations (usually 20 or 30), or encountered a derivative evaluation that was very close to zero.

- However between two solutions a chaotic region is seen, which follows no distinct pattern.

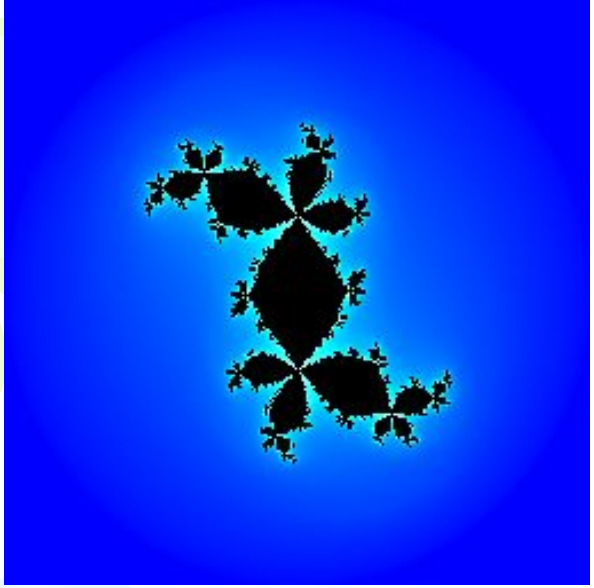


$$f(z) = z^3 - 1 = 0$$

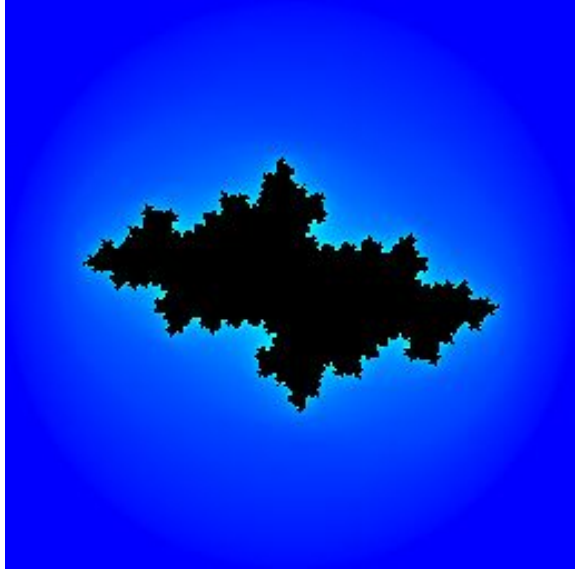
The second colouring scheme colours the pixels according to how many iterations are taken to approach a solution.

- There are dark circles around the solutions, indicating the areas where the solutions is approached quickly.
- In the fringes between two solutions, many iterations are required to converge to a solution, and a very small change in the starting position can cause the iteration to flip from one solution to another.

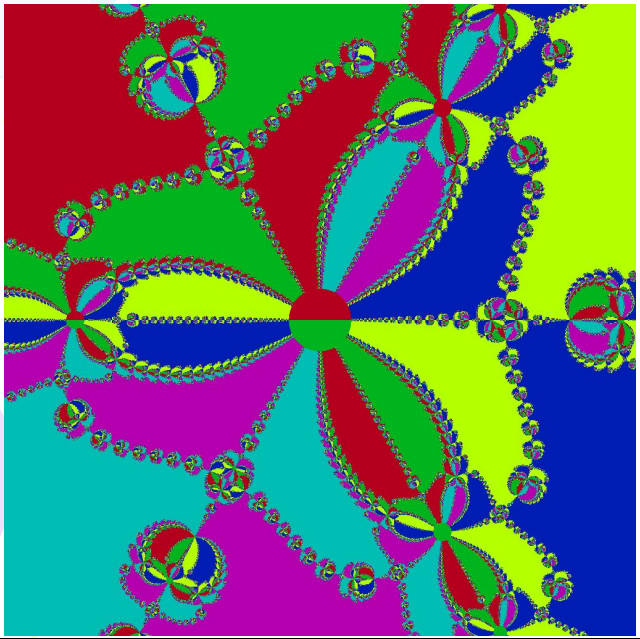
$$f(z) = z^6 - 1 = 0$$



Julia set for 0.28+0.528i



Julia set for -0.656+0.272i



Summary

- When dealing with high order polynomials don't rule out the possibility of being able to factorize by hand
- For general functions, bisection is simple but slow
  - Nonetheless the fact that it is guaranteed to converge from good starting points is extremely useful
- Newton-Raphson can have excellent convergence properties
  - Watch out for poor behaviour around extremums

Next Lecture

- More on Global bracket finding
- Müller-Brent method
- Examples

Exercise 2

- *Use the* **Bisection** *to calculate the roots of* $f(x)=\cos x - x$
- *seek the roots of* $f(x)=2-x-e^{-x}$

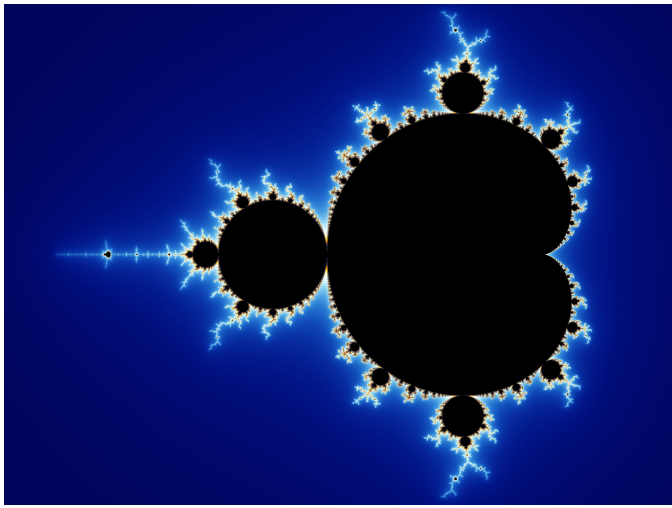
  1) NR method

  2) **Full NR-Bisection**

Homwork 4： 03/27/2019

Problem 1: **Use Newton-Raphson to get Fractals for**

$$f(z) = z^3 - 1 = 0$$



1) Adjust the specified tolerance (eg, 0.0001), to make sure the picture is nice?
2) Use two color scheme

Problem 2: Use **Full NR-Bisection algorithm** to find the roots of

$$f(x) = 4\cos x - e^x$$