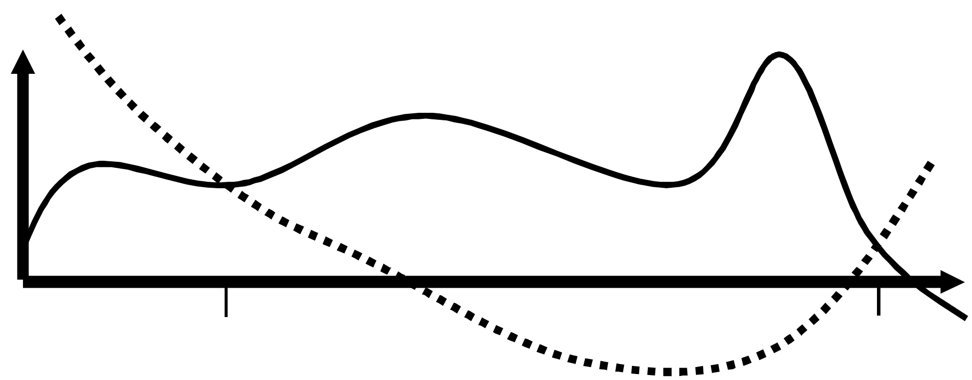
Root Finding Topics

- Bi-section Method
- Newton’s method
- Uses of root finding for sqrt() and reciprocal sqrt()
- Secant Method
- Generalized Newton’s method for systems of non-linear equations
  - The Jacobian matrix
- Fixed-point formulas, Basins of Attraction and Fractals.

Motivation

- Many problems can be re-written into a form such as:
  - $f(x,y,z,\ldots) = 0$
  - $f(x,y,z,\ldots) = g(s,q,\ldots)$



Motivation

- A root, r, of function f occurs when $f(r) = 0$.
- For example:
  - $f(x) = x^2 - 2x - 3$

  has two roots at $r = -1$ and $r = 3$.
    - $f(-1) = 1 + 2 - 3 = 0$
    - $f(3) = 9 - 6 - 3 = 0$
  - We can also look at f in its factored form.

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

Factored Form of Functions

- The factored form is not limited to polynomials.
- Consider:

  $f(x) = x \sin x - \sin x.$

  A root exists at $x = 1$.

  $f(x) = (x - 1) \sin x$

- Or,

  $f(x) = \sin \pi x \Rightarrow x\,(x - 1)\,(x - 2)\,\ldots$

Examples

- Find x, such that
  - $x^p = c, \Rightarrow x^p - c = 0$
    - Calculate the sqrt(2)
      - $x^2 - 2 = 0 = \left(x - \sqrt{2}\right)\left(x + \sqrt{2}\right)$
- Ballistics
  - Determine the horizontal distance at which the projectile will intersect the terrain function.

Root Finding Algorithms

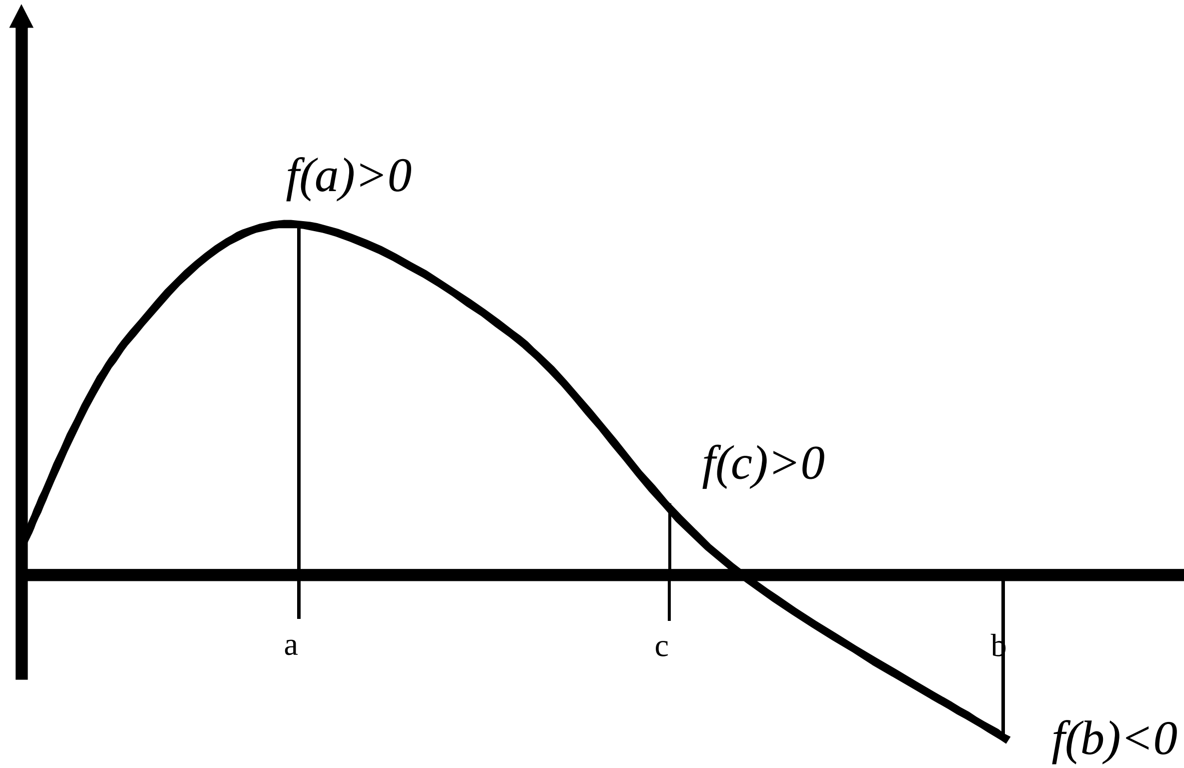
- **Closed or Bracketed techniques**
  - **Bi-section**
  - **Regula-Falsi**
- Open techniques
  - Newton fixed-point iteration
  - Secant method
- Multidimensional non-linear problems
  - The Jacobian matrix
- Fixed-point iterations
  - Convergence and Fractal Basins of Attraction

Bisection Method

- Based on the fact that the function will change signs as it passes thru the root.
    - $f(a)*f(b) < 0$
- Once we have a root ***bracketed***, we simply evaluate the mid-point and halve the interval.

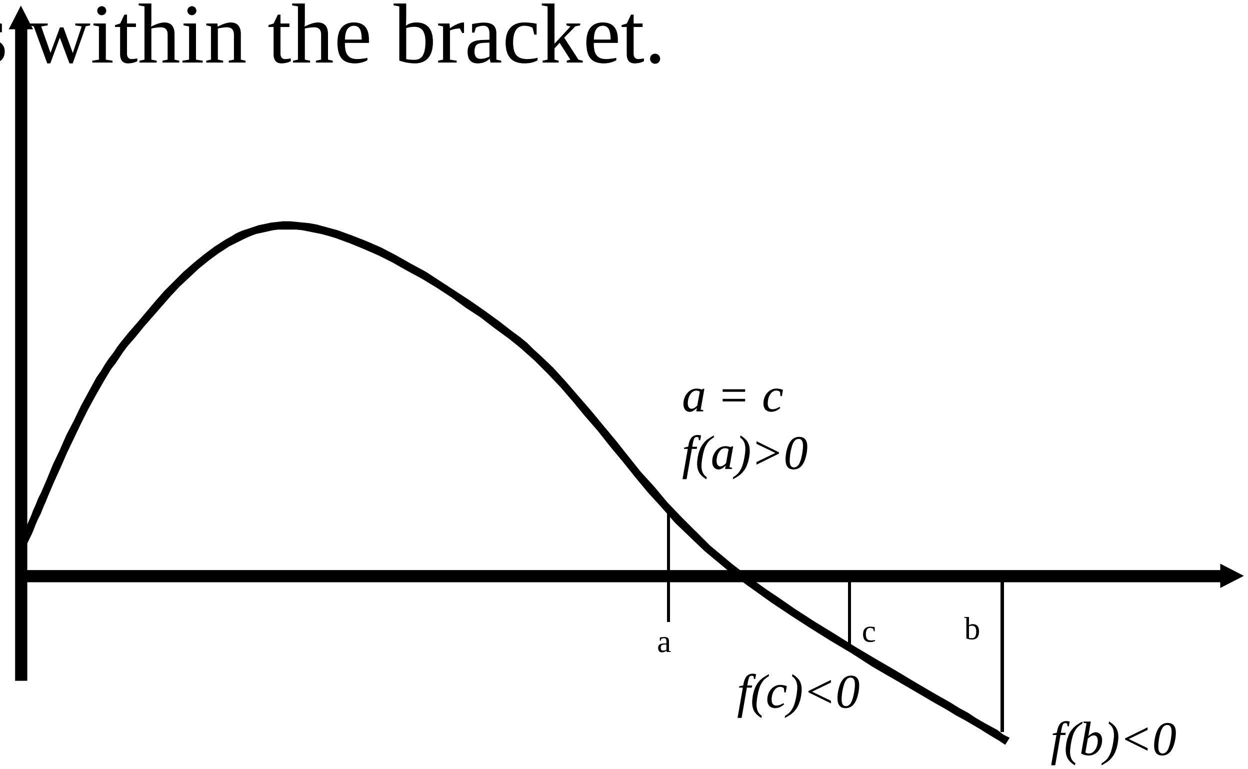
Bisection Method

- c=(a+b)/2



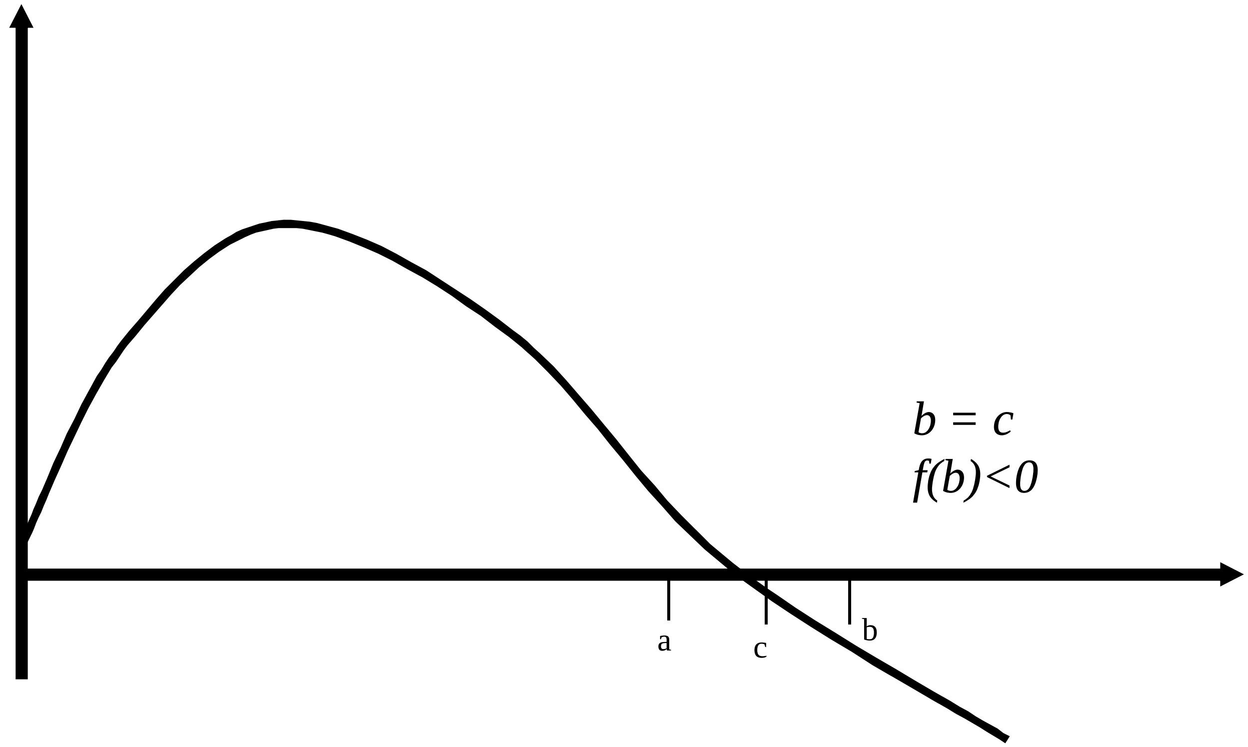
Bisection Method

- Guaranteed to converge to a root if one exists within the bracket.



Bisection Method

- Slowly converges to a root



Bisection Method

- Simple algorithm:

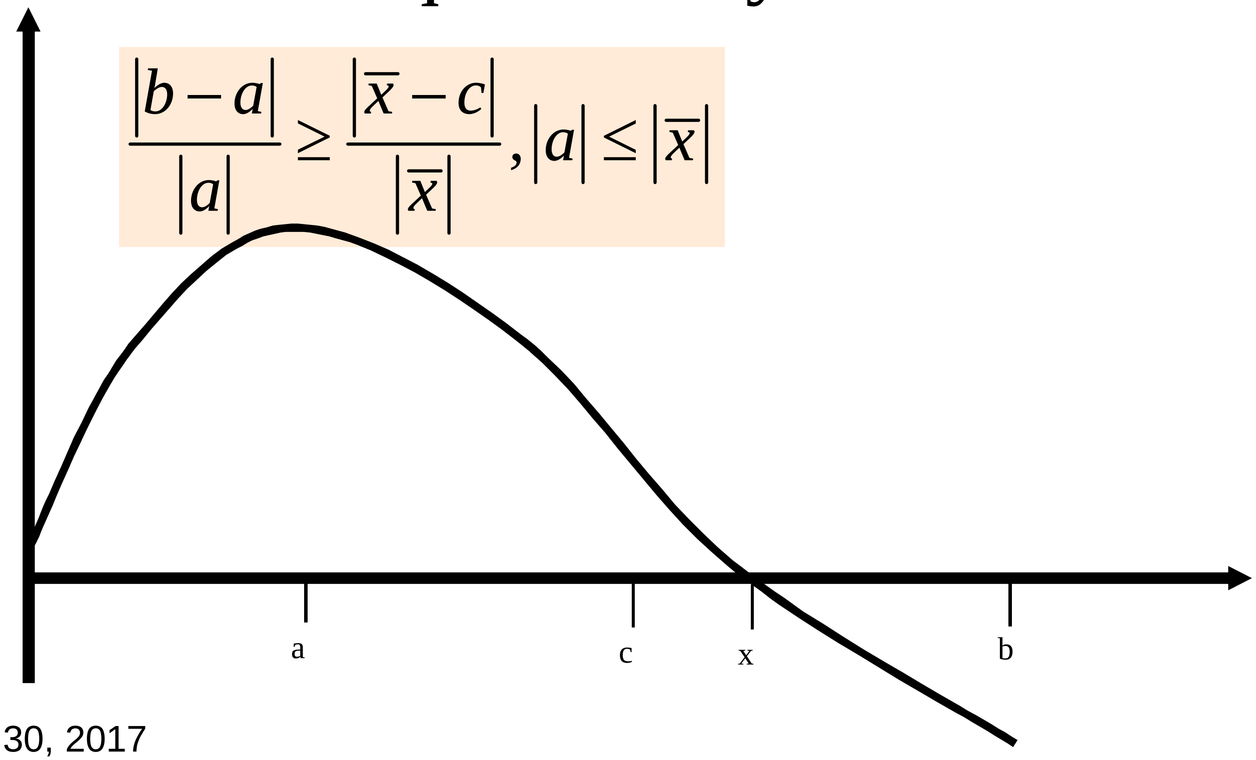
```
Given: a and b, such that f(a)*f(b)<0
Given: error tolerance, err

c=(a+b)/2.0; // Find the midpoint
While( |f(c)| > err ) {
    if( f(a)*f(c) < 0 )  // root in the left half
            b = c;
    else                 // root in the right half
            a = c;
    c=(a+b)/2.0; // Find the new midpoint
}
return c;
```

Relative Error

- We can develop an upper bound on the relative error quite easily.

$$\frac{|b-a|}{|a|} \geq \frac{|\bar{x}-c|}{|\bar{x}|}, |a| \leq |\bar{x}|$$



Absolute Error

- What does this mean in binary mode?
  - $err_0 \leq |b\text{-}a|$
  - $err_{i+1} \leq err_i/2 = |b\text{-}a|/2^{i+1}$
- We gain an extra bit each iteration!!!
- To reach a desired **absolute** error tolerance:
  - $err_{i+1} \leq err_{tol} \Rightarrow$

$$\frac{|b-a|}{2^n} \leq err_{tol}$$

$$n \geq \log_2\left(\frac{|b-a|}{err_{tol}}\right)$$

Absolute Error

- The bisection method converges linearly or first-order to the root.
- If we need an accuracy of 0.0001 and our initial interval (b-a)=1, then:

  $2^{-n} < 0.0001 \Rightarrow$ 14 iterations

- Not bad, why do I need anything else?

A Note on Functions

- Functions can be simple, but I may need to evaluate it many many times.
- Or, a function can be extremely complicated. Consider:
    - Interested in the configuration of air vents (position, orientation, direction of flow) that makes the temperature in the room at a particular position (teacher's desk) equal to 72°.
    - Is this a function?

A Note on Functions

- This function may require a complex three-dimensional heat-transfer coupled with a fluid-flow simulation to *evaluate* the function. $\Rightarrow$ hours of computational time on a supercomputer!!!
- May not necessarily even be computational.
- Techniques existed before the Babylonians.

Root Finding Algorithms

- **Closed or Bracketed techniques**
  - Bi-section
  - **Regula-Falsi**
- Open techniques
  - Newton fixed-point iteration
  - Secant method
- Multidimensional non-linear problems
  - The Jacobian matrix
- Fixed-point iterations
  - Convergence and Fractal Basins of Attraction

Regula Falsi

- In the book under computer problem 16 of section 3.3.
- Assume the function is linear within the bracket.
- Find the intersection of the line with the x-axis.

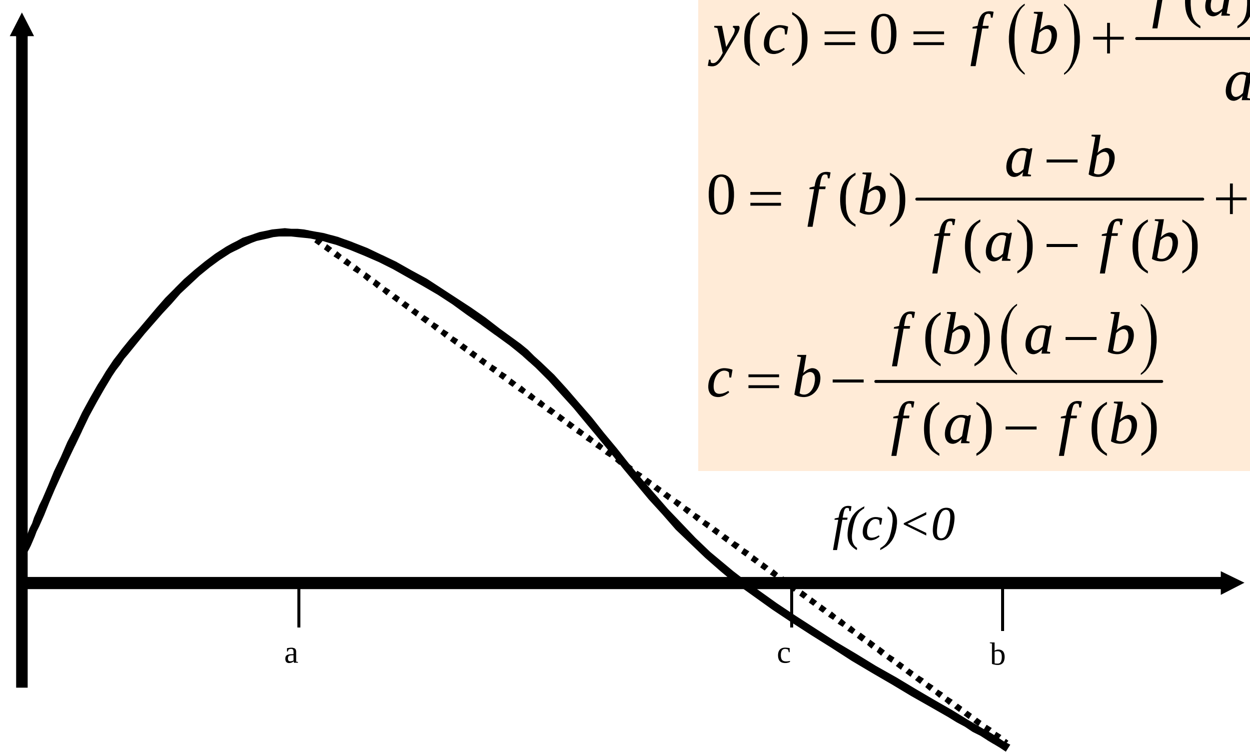
Regula Falsi

$$y(x) = f(b) + \frac{f(a) - f(b)}{a - b}(x - b)$$

$$y(c) = 0 = f(b) + \frac{f(a) - f(b)}{a - b}(c - b)$$

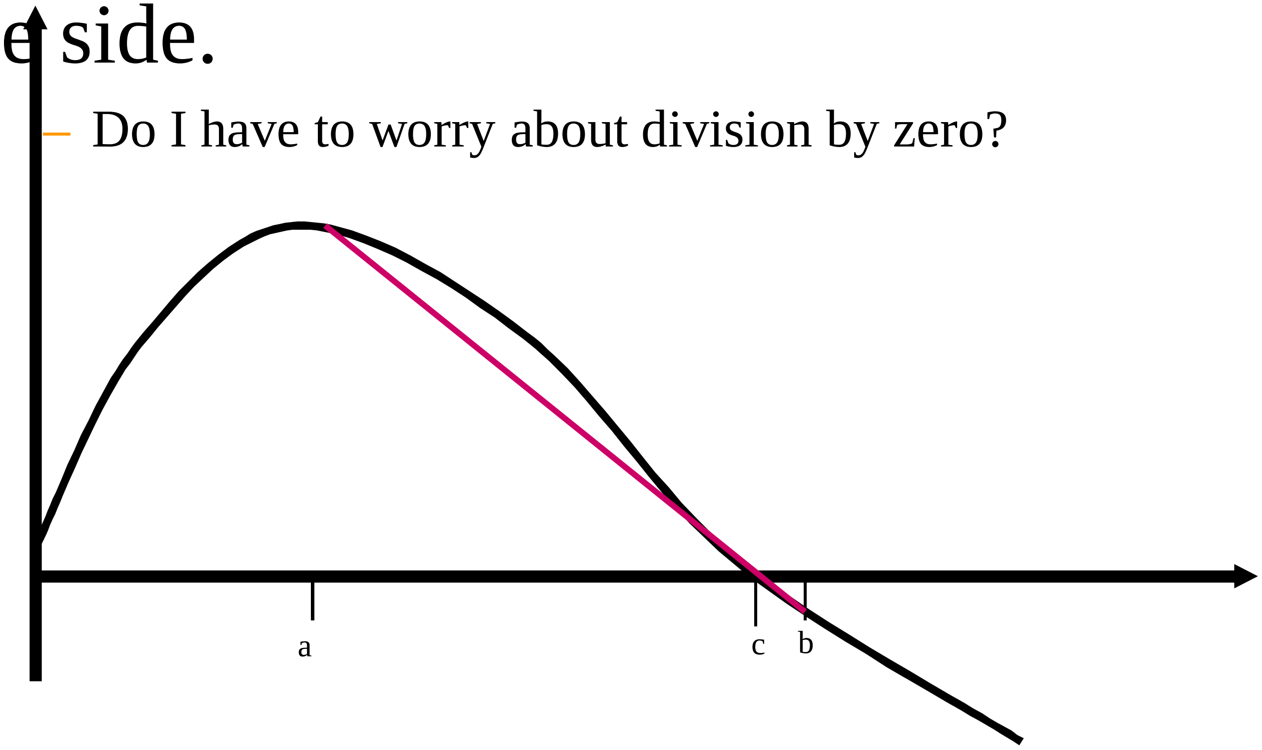
$$0 = f(b)\frac{a - b}{f(a) - f(b)} + c - b$$

$$c = b - \frac{f(b)(a - b)}{f(a) - f(b)}$$



Regula Falsi

- Large benefit when the root is much closer to one side.
  - Do I have to worry about division by zero?



Regula Falsi

- More generally, we can state this method as:

  $c = wa + (1-w)b$

  – For some weight, w, $0 \le w \le 1$.
  – If $|f(a)| >> |f(b)|$, then $w < 0.5$
    - Closer to **b**.

Bracketing Methods

- Bracketing methods are robust
- Convergence typically slower than open methods
- Use to find approximate location of roots
- “Polish” with open methods
- Relies on identifying two points a,b initially such that:
    - $f(a)\, f(b) < 0$
- Guaranteed to converge

Root Finding Algorithms

- Closed or Bracketed techniques
  - Bi-section
  - Regula-Falsi
- **Open techniques**
  - **Newton fixed-point iteration**
  - Secant method
- Multidimensional non-linear problems
  - The Jacobian matrix
- Fixed-point iterations
  - Convergence and Fractal Basins of Attraction

Newton's Method

- Open solution, that requires only one current guess.
- Root does not need to be bracketed.
- Consider some point x_0.
  - If we approximate *f(x)* as a line about x_0, then we can again solve for the root of the line.

$$l(x) = f'(x_0)(x - x_0) + f(x_0)$$

Newton's Method

- Solving, leads to the following iteration:

$$l(x) = 0$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Newton's Method

- This can also be seen from Taylor's Series.
- Assume we have a guess, x_0, close to the actual root. Expand *f(x)* about this point.

$$\bar{x} = x_i + \Delta x$$

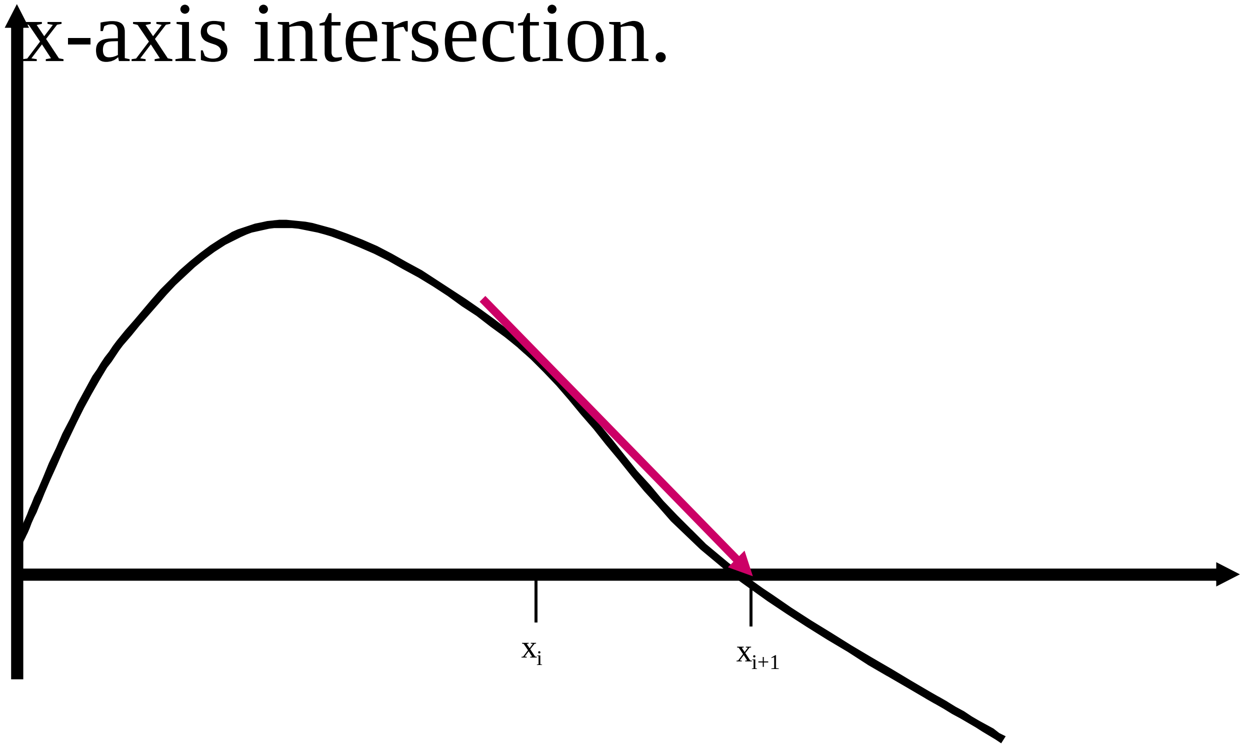
$$f(x_i + \Delta x) = f(x_i) + \Delta x f'(x_i) + \frac{\Delta x^2}{2!} f''(x_i) + \cdots \equiv 0$$

- If *dx* is small, then dx^n quickly goes to zero.

$$\Delta x \approx x_{i+1} - x_i = -\frac{f(x_i)}{f'(x_i)}$$

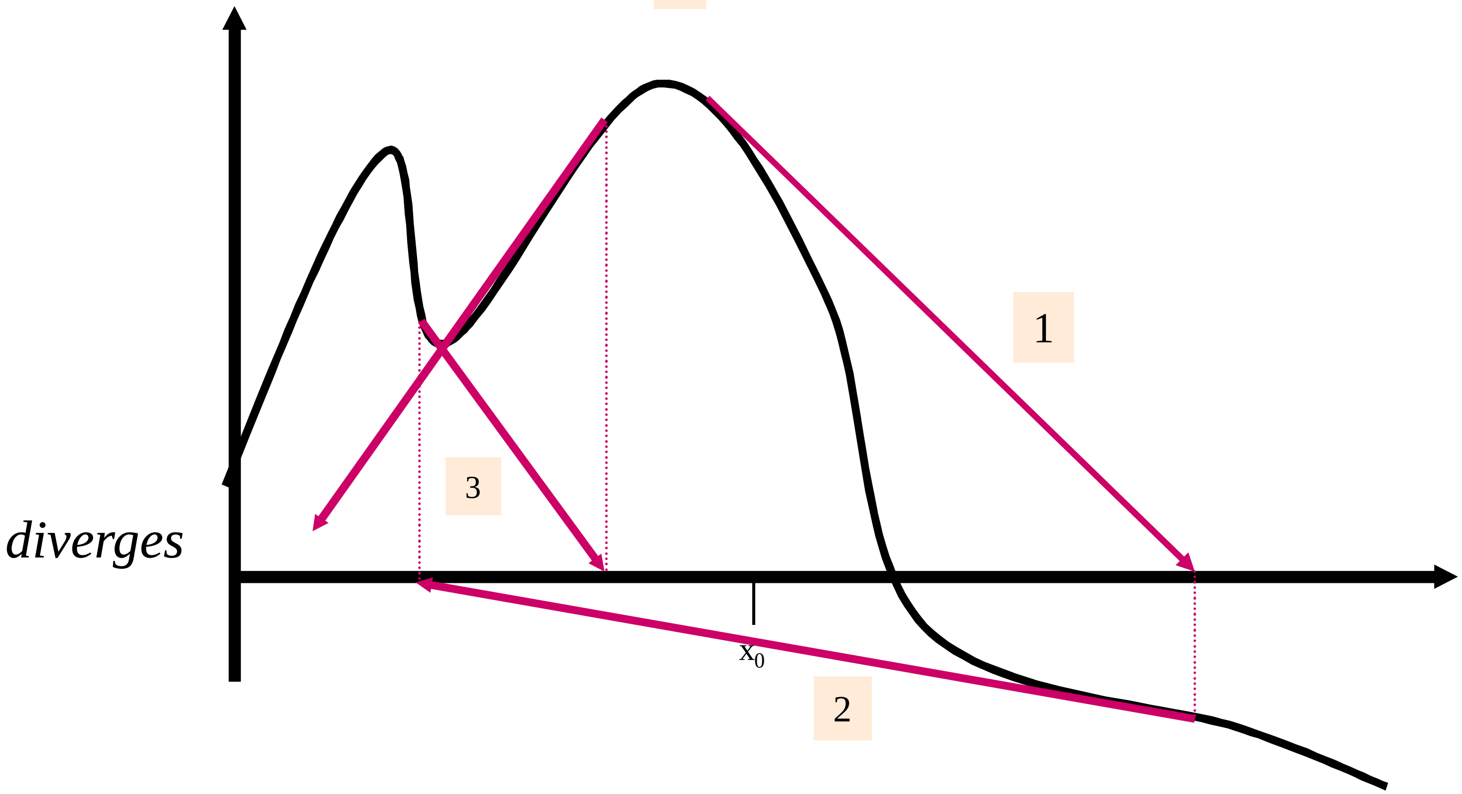
Newton's Method

- Graphically, follow the tangent vector down to the x-axis intersection.



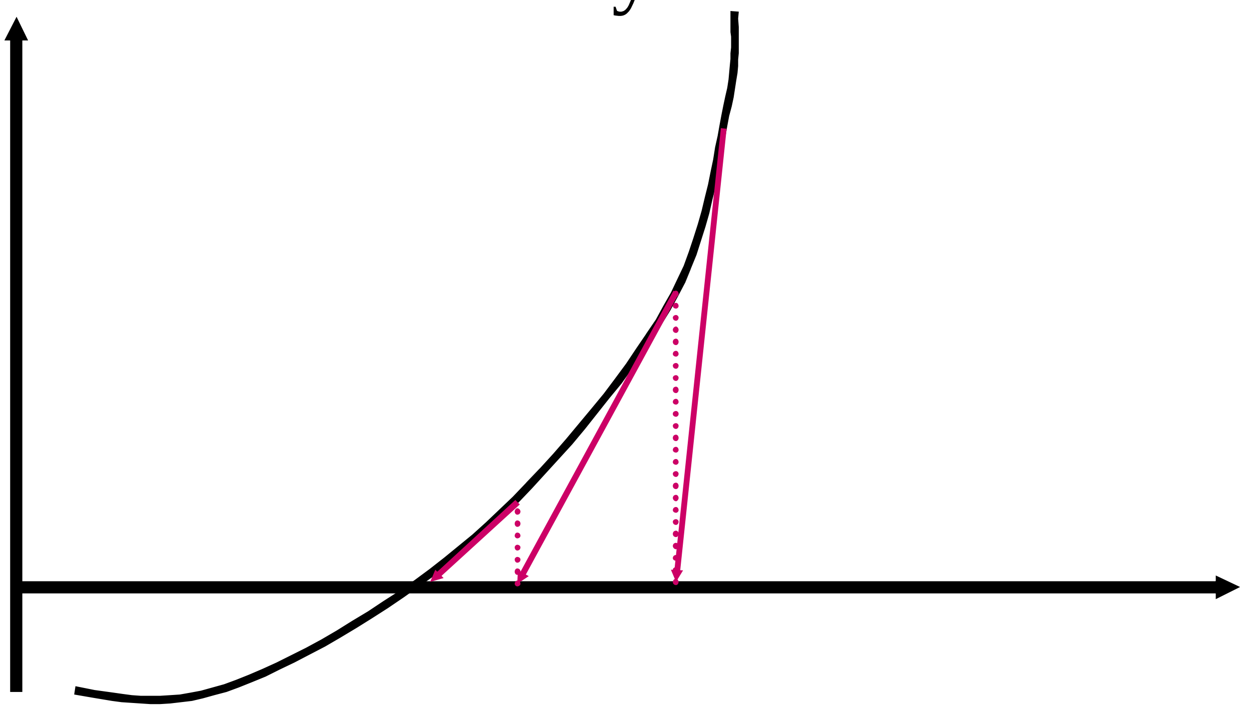
Newton's Method

- Problems



Newton's Method

- Need the initial *guess* to be close, **or**, the function to behave nearly linear within the range.



Finding a square-root

- Ever wonder why they call this a *square-root?*
- Consider the *roots* of the equation:
  - $f(x) = x^2$-a
- This of course works for any power:

$$\sqrt[p]{a} \Rightarrow x^p - a = 0, \quad p \in R$$

Finding a square-root

- Example: √2 = 1.4142135623730950488016887242097
- Let x_0 be one and apply Newton's method.

$$f'(x) = 2x$$

$$x_{i+1} = x_i - \frac{x_i^{\,2} - 2}{2x_i} = \frac{1}{2}\left(x_i + \frac{2}{x_i}\right)$$

$$x_0 = 1$$

$$x_1 = \frac{1}{2}\left(1 + \frac{2}{1}\right) = \frac{3}{2} = 1.5000000000$$

$$x_2 = \frac{1}{2}\left(\frac{3}{2} + \frac{4}{3}\right) = \frac{17}{12} \approx 1.4166666667$$

Finding a square-root

- Example: √2 = 1.4142135623730950488016887242097
- Note the rapid convergence

$$x_3 = \frac{1}{2}\left(\frac{17}{12} + \frac{24}{17}\right) = \frac{577}{408} \approx 1.414215686$$

$$x_4 = 1.414213562374\underset{\smile}{6}$$

$$x_5 = 1.41421356237309504880168\underset{\smile}{9}\underset{\smile}{6}$$

$$x_6 = 1.4142135623730950488016887242097$$

- Note, this was done with the standard Microsoft calculator to maximum precision.

Finding a square-root

- Can we come up with a better initial guess?
- Sure, just divide the exponent by 2.
  - Remember the bias offset
  - Use bit-masks to extract the exponent to an integer, modify and set the initial guess.
- For $\sqrt{2}$, this will lead to $x_0=1$ (round down).

Convergence Rate of Newton's

$$e_n = \bar{x} - x_n \quad or \quad \bar{x} = x_n + e_n$$

$$0 \equiv f(\bar{x}) = f(x_n + e_n)$$

$$f(x_n + e_n) = f(x_n) + e_n f'(x_n) + \frac{1}{2} e_n^2 f''(\xi_n),\ for\ some\ \xi_n \in (\bar{x}, x_n)$$

$$\therefore f(x_n) + e_n f'(x_n) = -\frac{1}{2} e_n^2 f''(\xi_n)$$

- Now,

$$e_{n+1} = \bar{x} - x_{n+1} = \bar{x} - x_n + \frac{f(x_n)}{f'(x_n)} = e_n + \frac{f(x_n)}{f'(x_n)}$$

$$= \frac{e_n f'(x_n) + f(x_n)}{f'(x_n)}$$

$$\therefore e_{n+1} = -\frac{1}{2}\left(\frac{f''(\xi_n)}{f'(x_n)}\right) e_n^2$$

Convergence Rate of Newton's

- Converges **quadratically**.

$$if\ |e_n| \le 10^{-k} \quad then,$$

$$|e_{n+1}| \le c10^{-2k}$$

Newton's Algorithm

- Requires the derivative function to be evaluated, hence more function evaluations per iteration.
- A robust solution would check to see if the iteration is stepping too far and limit the step.
- Most uses of Newton's method assume the approximation is pretty close and apply one to three iterations blindly.

Division by Multiplication

- Newton's method has many uses in computing basic numbers.
- For example, consider the equation:

$$\frac{1}{x} - a = 0$$

- Newton's method gives the iteration:

$$\begin{aligned} x_{k+1} &= x_k - \frac{\frac{1}{x_k} - a}{-\frac{1}{x_k^2}} = x_k + x_k - a x_k^2 \\ &= x_k \left(2 - a x_k\right) \end{aligned}$$

Reciprocal Square Root

- Another useful operator is the reciprocal-square root.
  - Needed to normalize vectors
  - Can be used to calculate the square-root.

$$a\frac{1}{\sqrt{a}} = \sqrt{a}$$

Reciprocal Square Root

$$Let \;\; f(x) = \frac{1}{x^2} - a = 0$$

$$f'(x) = -\frac{2}{x^3}$$

- Newton's iteration yields:

$$x_{k+1} = x_k + \frac{x_k}{2} - a\frac{x_k^3}{2}$$

$$= \frac{1}{2}x_k\left(3 - ax_k^2\right)$$

1/Sqrt(2)

- Let's look at the convergence for the reciprocal square-root of 2.

$$
\begin{aligned}
x_0 &= 1 \\
x_1 &= 0.5(1)\left(3-2\cdot 1^2\right) = 0.5 \\
x_2 &= 0.5(0.5)\left(3-2\cdot 0.5^2\right) = 0.625 \\
x_3 &= 0.693359375 \\
x_4 &= 0.706708468496799468994140625 \\
\mathrm{x}_5 &= 0.707106444695907075511730676593228 \\
\mathrm{x}_6 &= 0.707106781186307335925435931237738 \\
\mathrm{x}_7 &= 0.70710678118654752440084423972481
\end{aligned}
$$

If we could only start here!!

1/Sqrt(x)

- What is a good choice for the initial seed point?
  - Optimal – the root, but it is unknown
  - Consider the normalized format of the number:

$$(-1)^s \cdot 2^{e-127} \cdot (1.m)_2$$

  - What is the reciprocal?
  - What is the square-root?

1/Sqrt(x)

- Theoretically,

$$\frac{1}{\sqrt{x}} = x^{-\frac{1}{2}} = (1.m)^{-\frac{1}{2}} \bullet \left(2^{e-127}\right)^{-\frac{1}{2}}$$

$$= (1.m)^{-\frac{1}{2}} \bullet \left(2^{\frac{127-e}{2}}\right)$$

$$= (1.m)^{-\frac{1}{2}} \bullet \left(2^{\frac{3\bullet 127-e}{2}-127}\right)$$

New bit-pattern for the exponent

- Current GPU's provide this operation in as little as 2 clock cycles!!! How?
- How many significant bits does this estimate have?

1/Sqrt(x)

- GPU's such as nVidia's FX cards provide a 23-bit accurate reciprocal square-root in two clock cycles, by only doing 2 iterations of Newton's method.
- Need 24-bits of precision =>
  - Previous iteration had 12-bits of precision
  - Started with 6-bits of precision

1/Sqrt(x)

- Examine the mantissa term again (1.*m*).
- Possible patterns are:
  - 1.000…, 1.100…, 1.010…, 1.110…, …
- Pre-compute these and store the results in a table. Fast and easy table look-up.
- A 6-bit table look-up is only 64 words of on chip cache.
- Note, we only need to look-up on *m*, not 1.*m*.
- This yields a reciprocal square-root for the first seven bits, giving us *about* 6-bits of precision.

1/Sqrt(x)

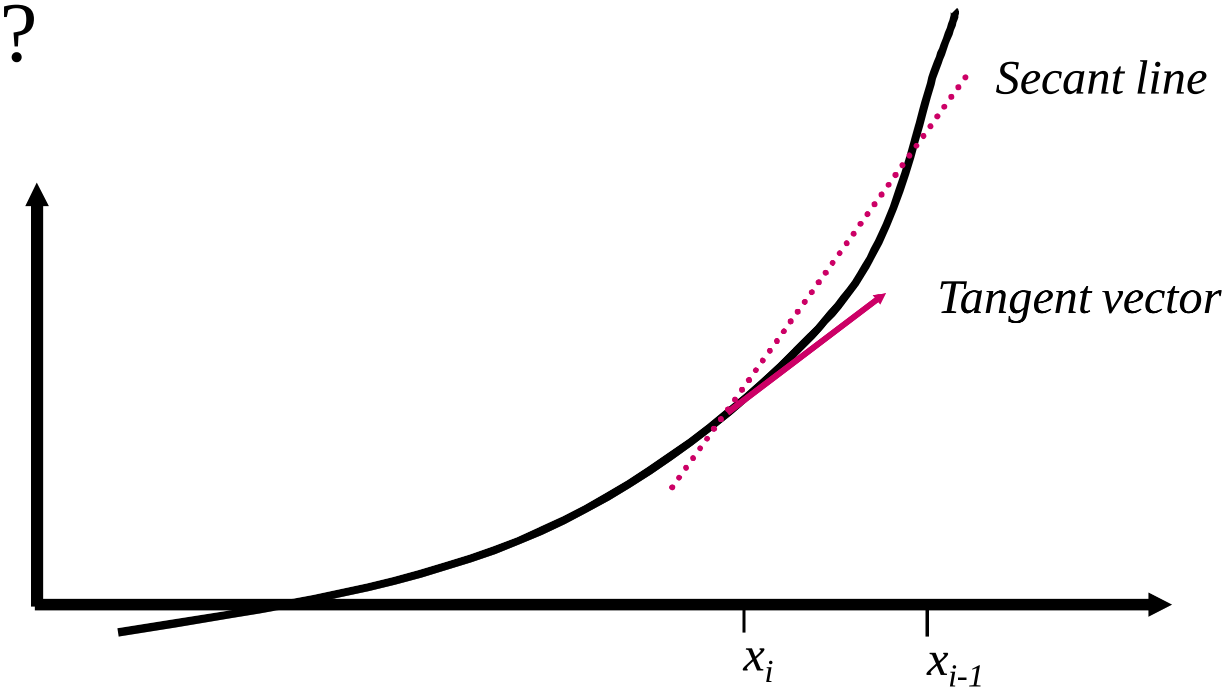
- Slight problem:
  - The $\sqrt{1.m}$ produces a result between 1 and 2.
  - Hence, it remains normalized, 1.*m'*.
  - For $\frac{1}{\sqrt{x}}$, we get a number between ½ and 1.
  - Need to shift the exponent.

Root Finding Algorithms

- Closed or Bracketed techniques
  - Bi-section
  - Regula-Falsi
- **Open techniques**
  - Newton fixed-point iteration
  - **Secant method**
- Multidimensional non-linear problems
  - The Jacobian matrix
- Fixed-point iterations
  - Convergence and Fractal Basins of Attraction

Secant Method

- What if we do not know the derivative of *f(x)*?



Secant Method

- As we converge on the root, the secant line approaches the tangent.
- Hence, we can use the secant line as an estimate and look at where it intersects the x-axis (its root).

Secant Method

- This also works by looking at the definition of the derivative:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

- Therefore, Newton's method gives:

$$x_{k+1} = x_k - \left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \right) f(x_k)$$

- Which is the Secant Method.

Convergence Rate of Secant

- Using Taylor's Series, it can be shown (proof is in the book) that:

$$e_{k+1} = \overline{x} - x_{k+1}$$
$$= -\frac{1}{2}\left(\frac{f''(\xi_k)}{f''(\zeta_k)}\right) e_k e_{k-1} \approx c \cdot e_k e_{k-1}$$

Convergence Rate of Secant

- This is a recursive definition of the error term. Expressed out, we can say that:

$$\left|e_{k+1}\right| \leq C\left|e_k\right|^{\alpha}$$

- Where α=1.62.
- We call this super-linear convergence.

Root Finding Algorithms

- Closed or Bracketed techniques
  - Bi-section
  - Regula-Falsi
- Open techniques
  - Newton fixed-point iteration
  - Secant method
- **Multidimensional non-linear problems**
  - **The Jacobian matrix**
- Fixed-point iterations
  - Convergence and Fractal Basins of Attraction

Higher-dimensional Problems

- Consider the class of functions

  $$f(x_1,x_2,x_3,\ldots,x_n)=0,$$

  where we have a mapping from $\Re^n \rightarrow \Re$.
- We can apply Newton's method separately for each variable, x_i, holding the other variables fixed to the current *guess*.

Higher-dimensional Problems

- This leads to the iteration:

$$x_i \rightarrow x_i - \frac{f\left(x_1, x_2, \ldots, x_n\right)}{f_{x_i}\left(x_1, x_2, \ldots, x_n\right)}$$

- Two choices, either I keep of complete set of old guesses and compute new ones, or I use the new ones as soon as they are updated.
- Might as well use the more accurate new guesses.
- Not a unique solution, but an infinite set of solutions.

Higher-dimensional Problems

- Example:
  - x+y+z=3
  - Solutions:
    - x=3, y=0, z=0
    - x=0, y=3, z=0
    - …

Systems of Non-linear Equations

- Consider the set of equations:

$$
\begin{aligned}
&f_1(x_1, x_2, \ldots, x_n) = 0 \\
&f_2(x_1, x_2, \ldots, x_n) = 0 \\
&\vdots \\
&f_n(x_1, x_2, \ldots, x_n) = 0
\end{aligned}
$$

Systems of Non-linear Equations

- Example:

$$
\begin{aligned}
x + y + z &= 3 \\
x^2 + y^2 + z^2 &= 5 \\
e^x + xy - xz &= 1
\end{aligned}
$$

Plane intersected with a sphere, intersected with a more complex function.

- Conservation of mass coupled with conservation of energy, coupled with solution to complex problem.

Vector Notation

- We can rewrite this using vector notation:

$$\vec{\mathbf{f}}(\vec{\mathbf{x}}) = \vec{\mathbf{0}}$$

$$\mathbf{f} = \left(f_1, f_2, \ldots, f_n \right)$$

$$\mathbf{x} = \left(x_1, x_2, \ldots, x_n \right)$$

Newton's Method for Non-linear Systems

- Newton's method for non-linear systems can be written as:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left[\mathbf{f}'\left(\mathbf{x}^{(k)}\right)\right]^{-1}\mathbf{f}\left(\mathbf{x}^{(k)}\right)$$

$$\textit{where } \mathbf{f}'\left(\mathbf{x}^{(k)}\right) \textit{ is the Jacobian matrix}$$

The Jacobian Matrix

- The Jacobian contains all the partial derivatives of the set of functions.

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \dfrac{\partial f_1}{\partial x_2} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \dfrac{\partial f_2}{\partial x_1} & \dfrac{\partial f_2}{\partial x_2} & \cdots & \dfrac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_n}{\partial x_1} & \dfrac{\partial f_n}{\partial x_2} & \cdots & \dfrac{\partial f_n}{\partial x_n} \end{bmatrix}$$

- Note, that these are all functions and need to be evaluated at a point to be useful.

The Jacobian Matrix

- Hence, we write

$$\mathbf{J}(\mathbf{x}^{(i)}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}\left(\mathbf{x}^{(i)}\right) & \frac{\partial f_1}{\partial x_2}\left(\mathbf{x}^{(i)}\right) & \cdots & \frac{\partial f_1}{\partial x_n}\left(\mathbf{x}^{(i)}\right) \\ \frac{\partial f_2}{\partial x_1}\left(\mathbf{x}^{(i)}\right) & \frac{\partial f_2}{\partial x_2}\left(\mathbf{x}^{(i)}\right) & \cdots & \frac{\partial f_2}{\partial x_n}\left(\mathbf{x}^{(i)}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}\left(\mathbf{x}^{(i)}\right) & \frac{\partial f_n}{\partial x_2}\left(\mathbf{x}^{(i)}\right) & \cdots & \frac{\partial f_n}{\partial x_n}\left(\mathbf{x}^{(i)}\right) \end{bmatrix}$$

Matrix Inverse

- We define the inverse of a matrix, the same as the reciprocal.

$$a\frac{1}{a}=1$$

$$\mathbf{A}\mathbf{A}^{-1}=\mathbf{I}=\begin{bmatrix}\mathbf{1} & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{1} & \cdots & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{1}\end{bmatrix}$$

Newton's Method

- If the Jacobian is non-singular, such that its inverse exists, then we can apply this to Newton's method.
- We rarely want to compute the inverse, so instead we look at the problem.

$$
\begin{aligned}
\mathbf{x}^{(i+1)} &= \mathbf{x}^{(i)} - \left[\mathbf{f}'\left(\mathbf{x}^{(i)}\right)\right]^{-1} \mathbf{f}\left(\mathbf{x}^{(i)}\right) \\
&= \mathbf{x}^{(i)} + \mathbf{h}^{(i)}
\end{aligned}
$$

Newton's Method

- Now, we have a linear system and we solve for **h**.

$$\left[\mathbf{J}\left(\mathbf{x}^{(k)}\right)\right]\mathbf{h}^{(k)} = -\mathbf{f}\left(\mathbf{x}^{(k)}\right)$$

$$\mathbf{x}^{(i+1)} = \mathbf{x}^{(i)} + \mathbf{h}^{(i)}$$

- Repeat until **h** goes to zero.

We will look at solving linear systems later in the course.

Initial Guess

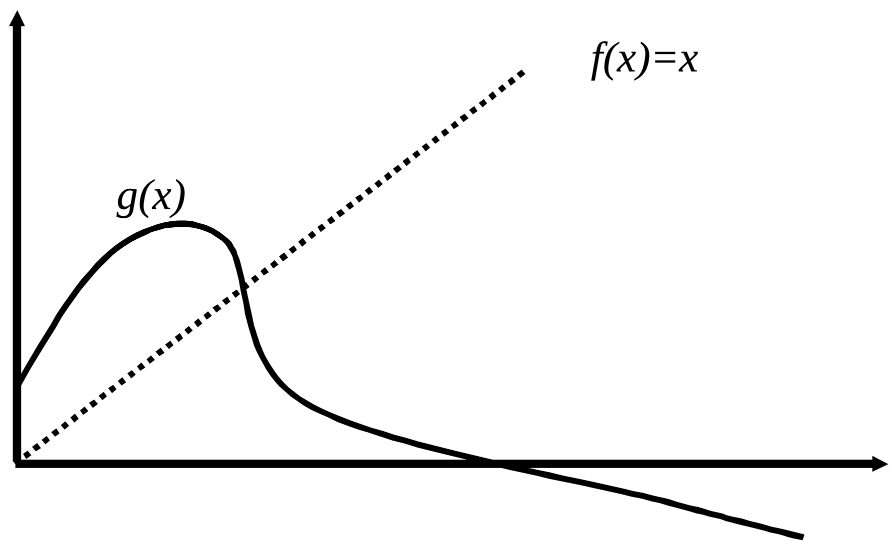
- How do we get an initial guess for the root vector in higher-dimensions?
- In 2D, I need to find a region that contains the root.
- Steepest Decent is a more advanced topic not covered in this course. It is more stable and can be used to determine an approximate root.

Root Finding Algorithms

- Closed or Bracketed techniques
  - Bi-section
  - Regula-Falsi
- Open techniques
  - Newton fixed-point iteration
  - Secant method
- Multidimensional non-linear problems
  - The Jacobian matrix
- **Fixed-point iterations**
  - **Convergence and Fractal Basins of Attraction**

Fixed-Point Iteration

- Many problems also take on the specialized form: $g(\mathbf{x})=\mathbf{x}$, where we seek, $\mathbf{x}$, that satisfies this equation.



Fixed-Point Iteration

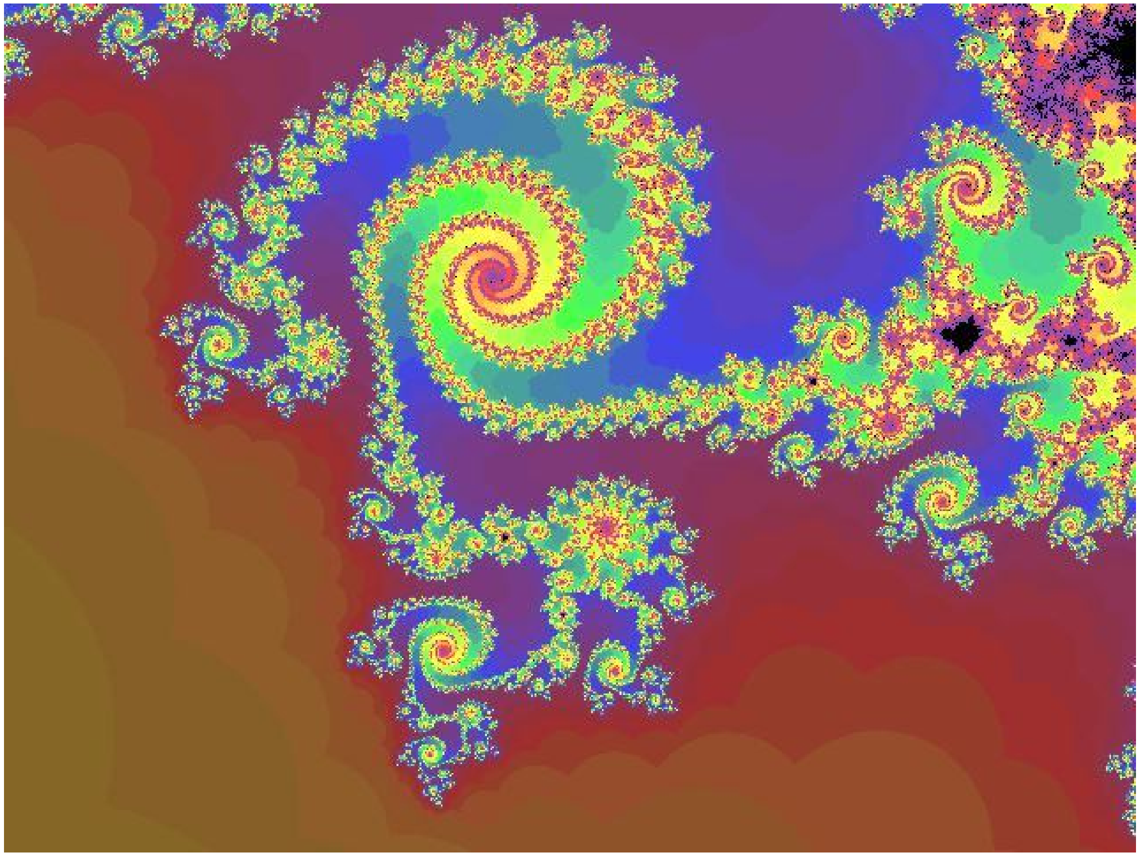
- Newton's iteration and the Secant method are of course in this form.
- In the limit, $f(x_k)=0$, hence $x_{k+1}=x_k$

Fixed-Point Iteration

- Only problem is that that **assumes** it converges.
- The pretty fractal images you see basically encode how many iterations it took to either converge (to some accuracy) or to diverge, using that point as the initial seed point of an iterative equation.
- The book also has an example where the roots converge to a finite set. By assigning different colors to each root, we can see to which point the initial seed point converged.

Fractals

- Images result when we deal with 2-dimensions.
- Such as complex numbers.
- Color indicates how quickly it converges or diverges.



Fixed-Point Iteration

- More on this when we look at iterative solutions for linear systems (matrices).