Lecture 3

Rootfinding

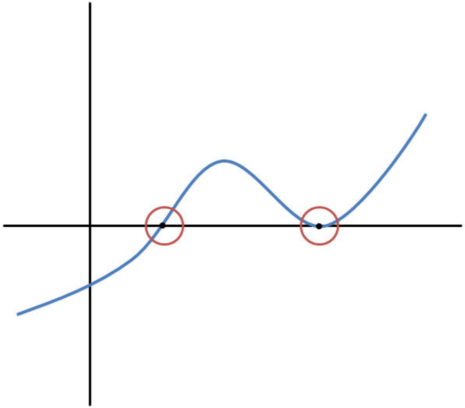
T. Gambill

Department of Computer Science
University of Illinois at Urbana-Champaign

February 14, 2012

Root Finding

Given a function $f(x)$, find x so that $f(x) = 0$



Rootfinding

Goals:

- Find roots to equations
- Compare usability of different methods
- Compare convergence properties of different methods

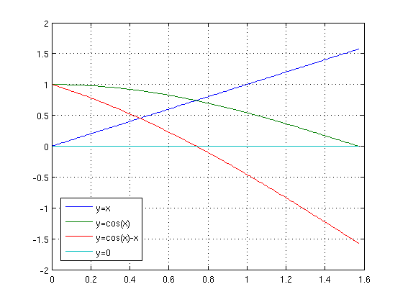
1. bracketing methods
2. Bisection Method
3. Newton's Method
4. Secant Method
5. (opt) fixed point iterations
6. (opt) special Case: Roots of Polynomials

Roots of $f(x)$

- Any single valued equation $g(x) = h(x)$ can be written as $f(x) = g(x) - h(x) = 0$

Example

- Find x so that $\cos(x) = x$
- That is, find where $f(x) = \cos(x) - x = 0$



Analyze your Application

- Is the function complicated to evaluate?
  - lots of expressions?
  - singularities?
  - simplify? polynomial?
- How accurate does our root need to be?
- How fast/robust should our method be?

!

From this, you can pick the right method...

Basic Root Finding Strategy

1. Plot the function
   - Get an initial guess
   - Identify problematic parts
2. Start with the initial guess and iterate

Iteration

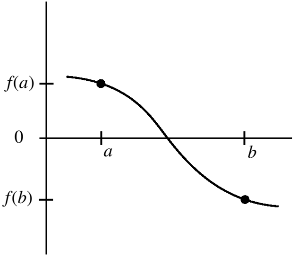
We need to study some iterations.

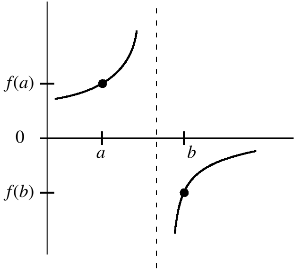
- iteratively finding a root to an equation
- iteratively finding the solution to an algebraic system
- iteratively finding solutions to Ordinary Differential Equations (ODEs)
- ...

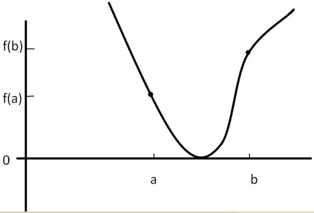
Bracket Basics

bó lại

- A root x is *bracketed* on $[a, b]$ if $f(a)$ and $f(b)$ have opposite sign.
- Changing signs does not guarantee bracketed, however: singularity







Bracket Algorithm

given: $f(x)$, x_min, x_max, n

Listing 1: Bracket Algorithm

```


dx = (x_max - x_min)/n
x_left = x_min
i=0

while i < n:
    i = i + 1
    x_right = x_left + dx
    if (f(x) changes sign in [x_left,x_right]):
        save [x_left,x_right]# as an interval with a root
    x_left = x_right
```

Testing Sign

$f(a) \times f(b) < 0$

Should we use?

```
fa = myfunc(a);
fb = myfunc(b);

if(fa*fb<0)
  (save)
end
```

Better Sign Test

!

Nope. Underflow...

sign()

Use Python's sign function

```
import numpy as np
fa = myfunc(a);
fb = myfunc(b);

if np.sign(fa) != np.sign(fb):
  (save)
```

Moving forward...

Bracketing is fine. But we need to find the actual root:

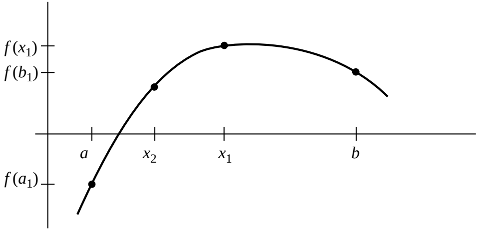
- Bisection
- Newton's Method
- Secant Method
- Fixed Point Iteration

Process:

1. Implement the bracket algorithm to get a visual and brackets
2. search brackets with these methods

Bisection

Given $f : \mathbb{R} \to \mathbb{R}$ and $f \in C([a, b])$ and $sign(f(a)) \neq sign(f(b))$ by the Intermediate Value Theorem we know we have a bracketed root on the interval $[a, b]$. Bisection Method: halve the interval while continuing to bracket the root.



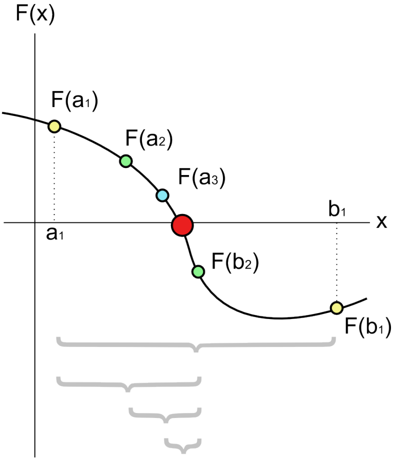
Bisection (2)

For the bracket interval $[a, b]$ the midpoint is

$$x_m = \frac{1}{2}(a + b)$$

idea:

1. split bracket in half
2. select the bracket that has the root
3. goto step 1



Bisection Algorithm

```python
import numpy as np
from scipy import optimize
import pprint

def bisection(f,a1,b1,tol):
    a = a1
    b = b1
    sfb = np.sign(f(b))
    k=1
    print('    k     a        b          x_mid       f(x_mid)    width')
    while b - a > tol:
        x = (a+b)/2.
        y = f(x)
        sfx = np.sign(y)
        w = np.abs(b-a)
        print('%5d %10.6f %10.8f %10.8f %11.8f %11.8f' % (k,a,b,x,y,w))
        if sfx == 0.:
            a = x
            b = x
            break
        elif sfx == sfb:
            b = x
        else:
            a = x
        k = k + 1


def f(x):
    return x - x**(1./3.) - 2

if __name__ == "__main__":
    bisection(f,3.,4.,1.e-3)
```

Bisection Example

Solve with bisection:

$$x - x^{1/3} - 2 = 0 \text{ solution from Matlab:} 3.521379706804568$$

k	a	b	x_mid	f(x_mid)	width
1	3.000000	4.00000000	3.50000000	-0.01829449	1.00000000
2	3.500000	4.00000000	3.75000000	0.19638375	0.50000000
3	3.500000	3.75000000	3.62500000	0.08884159	0.25000000
4	3.500000	3.62500000	3.56250000	0.03522131	0.12500000
5	3.500000	3.56250000	3.53125000	0.00845016	0.06250000
6	3.500000	3.53125000	3.51562500	-0.00492550	0.03125000
7	3.515625	3.53125000	3.52343750	0.00176150	0.01562500
8	3.515625	3.52343750	3.51953125	-0.00158221	0.00781250
9	3.519531	3.52343750	3.52148438	0.00008959	0.00390625
10	3.519531	3.52148438	3.52050781	-0.00074632	0.00195312

Analysis of Bisection

Let $\delta_n = x_{b_n} - x_{a_n}$ be the size of the bracketing interval $[x_{a_n}, x_{b_n}]$ with x_n the middle of the n^{th} stage of bisection. If r is the bracketed root then

$$|x_n - r| \leqslant \frac{1}{2}\delta_n \text{ where}$$

$$\begin{aligned}
\delta_1 &= b - a = \text{initial bracketing interval} \\
\delta_2 &= \frac{1}{2}\delta_1 \\
\delta_3 &= \frac{1}{2}\delta_2 = \frac{1}{4}\delta_1 \\
&\vdots \\
\delta_n &= \left(\frac{1}{2}\right)^{n-1}\delta_1 \qquad \text{thus}
\end{aligned}$$

$$|x_n - r| \leqslant \left(\frac{1}{2}\right)^{n}\delta_1$$

Analysis of Bisection

$$\frac{\delta_{n+1}}{\delta_1} = \left(\frac{1}{2}\right)^n = 2^{-n} \qquad \text{or} \qquad n = \log_2\left(\frac{\delta_1}{\delta_{n+1}}\right)$$

n	$\frac{\delta_{n+1}}{\delta_1}$	function evaluations
5	3.1×10^{-2}	7
10	9.8×10^{-4}	12
20	9.5×10^{-7}	22
30	9.3×10^{-10}	32
40	9.1×10^{-13}	42
50	8.9×10^{-16}	52

Remember the game Twenty questions?

Convergence Criteria

An automatic root-finding procedure needs to monitor progress toward the root and stop when current guess is close enough to the desired root.

- Convergence checking will avoid searching to unnecessary accuracy.
- Check how closeness of successive approximations

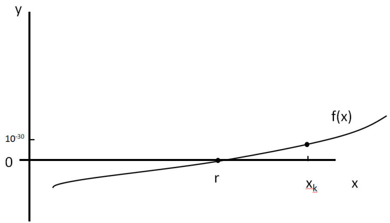
$$|x_k - x_{k-1}| < \delta_x$$

- Check how close $f(x)$ is to zero at the current guess.

$$|f(x_k)| < \delta_f$$

- Which one you use depends on the problem being solved

Convergence Criteria on x versus $f(x)$



Is x_k a sufficient approximation of a root at r? What if $r = 1$ and $x_k = 100$?

Alternative view

We have two views for finding roots

- Find r such that $f(r) = 0$
- Compute $r = f^{-1}(0)$

The two views give us two ways to determine errors.

Condition Number of Problem

Given a function $G : \mathbb{R} \to \mathbb{R}$,suppose we wish to compute $y = G(x)$. How sensitive is the solution to changes in x? We can measure this sensitivity in two ways:

- Absolute Condition Number = $\lim_{h\to 0} \frac{|G(x+h)-G(x)|}{|h|}$
- Relative Condition Number = $\lim_{h\to 0} \frac{\frac{|G(x+h)-G(x)|}{|G(x)|}}{\frac{|h|}{|x|}}$

Condition numbers much greater than one mean that the problem is inherently sensitive.

Condition Number Example

Given the problem of finding a root of a function $f : \mathbb{R} \to \mathbb{R}$,consider the absolute condition number applied to the problem of computing $f^{-1}(0)$.

$$\begin{aligned}
\text{Absolute Condition Number} &= \lim_{h \to 0} \frac{|f^{-1}(0+h) - f^{-1}(0)|}{|h|} \\
&= \left.\frac{df^{-1}(y)}{dy}\right|_{y=0} \quad \text{and from Calculus} \\
&= \frac{1}{\left.\frac{df(x)}{dx}\right|_{x=r}}
\end{aligned}$$

We conclude that the root finding problem is inherently sensitive to change if $\left|\frac{df(r)}{dx}\right| \approx 0$.

Condition Number Example

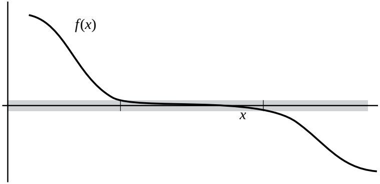
Given the problem of finding a root of a function $f : \mathbb{R} \rightarrow \mathbb{R}$,consider the absolute condition number applied to the problem of computing $f(r)$ where r is a root of f.

$$\text{Absolute Condition Number} = \lim_{h \to 0} \frac{|f(r+h) - f(r)|}{|h|} = \left. \frac{df(x)}{dx} \right|_{x=r}$$

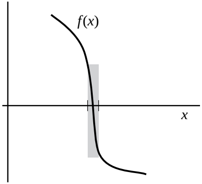
We conclude that the root finding problem is inherently sensitive to change if $\left| \frac{df(r)}{dx} \right| >> 1$.

Convergence Criteria Compared

If $f'(x)$ is small near the root, it is easy to satisfy tolerance on $f(x)$ for a large range of Δx.



If $f'(x)$ is large near the root, it is possible to satisfy the tolerance on Δx when $|f(x)|$ is still large.



Convergence rate of a root finding iteration

- Let $e_n = x^* - x_n$ be the error.
- In general, a sequence is said to converge with rate if r is the largest real for which the limit below is finite.

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^r} = C$$

Special Cases:

- If $r = 1$ and $C = 1$, then the rate is *sublinear*
- If $r = 1$ and $C < 1$, then the rate is *linear*
- If $r > 1$ (i.e. $r = 1$ and $C = 0$), then the rate is *superlinear*
- If $r = 2$ and $C > 0$, then the rate is *quadratic*

Convergence rate of the bisection method

When the bisection method ”converges” it can be shown that,

Bisection Method

The bisection method converges with rate $r = 1$ and $C = 0.5$.

Example

Convergence Rate

1. $10^{-2}, 10^{-3}, 10^{-4}, 10^{-5} \ldots$
2. $10^{-2}, 10^{-4}, 10^{-6}, 10^{-8} \ldots$
3. $10^{-2}, 10^{-3}, 10^{-5}, 10^{-8} \ldots$
4. $10^{-2}, 10^{-4}, 10^{-8}, 10^{-16} \ldots$
5. $10^{-2}, 10^{-6}, 10^{-18}, \ldots$

Example

Convergence Rate

1. $10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$... (linear with $C = 10^{-1}$)
2. $10^{-2}, 10^{-4}, 10^{-6}, 10^{-8}$... (linear with $C = 10^{-2}$)
3. $10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$... (superlinear, not quadratic)
4. $10^{-2}, 10^{-4}, 10^{-8}, 10^{-16}$...(quadratic)
5. $10^{-2}, 10^{-6}, 10^{-18}$, ... (cubic)

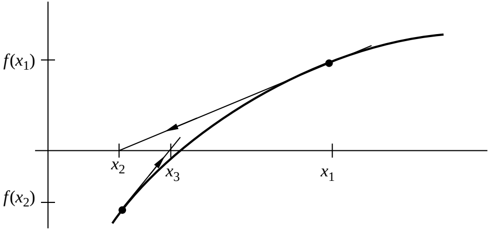
- Linear: Adds equal number of digits of accuracy at each step
- Quadratic: Doubles the number of digits at each step

Performing Division

- Ever wondered how a computer process performs division?
- “Long” division requires lookup, subtraction, shifts
- Generates one digit and a time. Can we do better?

To answer this, we need to look at faster methods than bisection

Newton's Method



For a current guess x_k, use $f(x_k)$ and the slope $f'(x_k)$ to predict where $f(x)$ crosses the x axis.

Newton's Method

Expand $f(x)$ in Taylor Series around x_k

$$f(x_k + \Delta x) = f(x_k) + \Delta x \left.\frac{df}{dx}\right|_{x_k} + \frac{(\Delta x)^2}{2} \left.\frac{d^2 f}{dx^2}\right|_{x_k} + \dots$$

Substitute $\Delta x = x_{k+1} - x_k$
and neglect 2^{nd} order terms to get

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

where

$$f'(x_k) = \left.\frac{df}{dx}\right|_{x_k}$$

Newton's Method

Goal is to find x such that $f(x) = 0$.
Set $f(x_{k+1}) = 0$ and solve for x_{k+1}

$$0 = f(x_k) + (x_{k+1} - x_k)f'(x_k)$$

or, solving for x_{k+1}

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

Newton's Method Algorithm

```
initialize:  x_1 = ...
for k = 2,3,...
   x_k = x_{k-1} - f(x_{k-1})/f'(x_{k-1})
   if converged, stop
end
```

Newton's Method Example

Solve:

$$x - x^{1/3} - 2 = 0$$

First derivative is

$$f'(x) = 1 - \frac{1}{3}x^{-2/3}$$

The iteration formula is

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

Newton's Method Example

```python
import numpy as np
from scipy import optimize
import pprint

def newton(f,fp, x,tol):
    k = 1
    print('    k                x_k              fp(x_k)     f(x_k)')
    print('%5d %22.20f %11.8f %11.8g' % (k,x,fp(x),f(x)))
    k = k + 1
    while np.abs( f(x) ) > tol:
        x = x - f(x)/fp(x)
        print('%5d %22.20f %11.8f %11.8g' % (k,x,fp(x),f(x)))
        k = k + 1


def f(x):
    return x - x**(1./3.) - 2.

def fp(x):
    return 1. - x**(-2./3.)/3.



if __name__ == "__main__":
    newton(f,fp, 3.,1.e-25)
```

Newton's Method Example

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

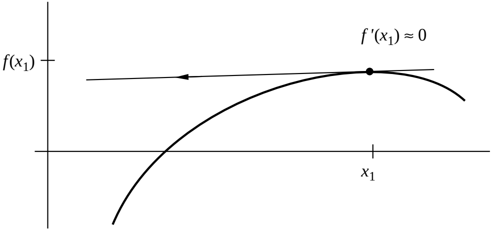
The approximate true root = 3.52137970680457046412926

k	x_k	fp(x_k)	f(x_k)
1	3.00000000000000000000	0.83975005	-0.44224957
2	3.52664429313903271535	0.85612976	0.0045067918
3	3.52138014739732829739	0.85598641	3.7714141e-07
4	3.52137970680457090822	0.85598640	2.6645353e-15
5	3.52137970680456779959	0.85598640	0

Conclusion

- Newton's method converges *much* more quickly than bisection
- Newton's method requires an analytical formula for $f'(x)$
- The algorithm is simple as long as $f'(x)$ is available.
- Iterations are not guaranteed to stay inside an ordinal bracket.

Divergence of Newton's Method



Since

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

the new guess, x_{k+1}, will be far from the old guess whenever $f'(x_k) \approx 0$

Newton's Method: Convergence

Recall

Convergence of a method is said to be of order r if there is a constant C such that

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^r} = C$$

If Newton's method converges then it is of order 2 (quadratic) when $f'(x_*) \neq 0$. (assuming f'' is continuous) For ξ_k between x_k and x_*

$$f(x_*) = f(x_k) + (x_* - x_k)f'(x_k) + \frac{1}{2}(x_* - x_k)^2 f''(\xi_k) = 0$$

So

$$\frac{f(x_k)}{f'(x_k)} + x_* - x_k + \frac{1}{2}(x_* - x_k)^2 \frac{f''(\xi_k)}{f'(x_k)} = 0$$

Then

$$x_* - x_{k+1} + \frac{1}{2}(x_* - x_k)^2 \frac{f''(\xi_k)}{f'(x_k)} = 0$$

Thus

$$\frac{|x_* - x_{k+1}|}{|x_* - x_k|^2} = \frac{1}{2}\left|\frac{f''(\xi_k)}{f'(x_k)}\right| \to \frac{1}{2}\left|\frac{f''(x_*)}{f'(x_*)}\right| \text{ as } x_k \to x_*$$

Reciprocal Approximation

- Consider the task of computing $1/q$ for some q without using division.
- We can write this as: find the root x of $f(x) = 1/(xq) - 1 = 0$.
- What is Newton's Method for this?
- $f'(x) = -1/(x^2q)$. Thus

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

or

$$x_{n+1} = x_n - \left(\frac{1/(x_nq) - 1}{-1/(x_n^2q)}\right)$$

Reciprocal Approximation

- Consider the task of computing $1/q$ for some q without using division.
- We can write this as: find the root x of $f(x) = 1/(xq) - 1 = 0$.
- What is Newton's Method for this?
- $f'(x) = -1/(x^2 q)$. Thus

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

or

$$x_{n+1} = x_n - \left(\frac{1/(x_n q) - 1}{-1/(x_n^2 q)}\right)\frac{x_n^2 q}{x_n^2 q}$$

$$x_{n+1} = x_n + x_n - x_n^2 q = 2x_n - x_n^2 q = 2x_n - x_n^2 q$$

Example: Compute 1/3 = 0.01010101... binary

- Find the bracket:
- $1/2 > 1/3 > 1/4$

1. $x_1 = 1/4$
2. $x_2 = 2x_1 - x_1^2 q = 1/2 - 3/16 = 5/16 = 0.0101 \quad (binary)$
3. $x_3 = 2 \times 5/2^4 - 3 \times 25/2^8 = (160 - 75)/2^8 = 85/2^8 = 0.01010101 \quad (binary)$
4. $x_4 = 2 \times 85/2^8 - 3 \times 85^2/2^{16} = 21845/2^{16} = 0.0101010101010101 \quad (binary)$

In 3 steps, computed 16 bits in $1/3$

How many binary digits are computed in the next step?

Instructor Notes

- Modification of Newton's Method for root finding when $\frac{df}{dx}(root) = 0$. Use the formula,
$$x_{n+1} = x_n - m * \frac{f(x_n)}{f'(x_n)}$$
where m is the multiplicity of the root.
- or solve $0 = g(x) = \frac{f(x)}{f'(x)}$