

Lecture 06: Rootfinding and Optimization for functions of a single variable $f(x)$

Outline

1) Fail-safe hybrid methods

NewtonSafe (Newton + Bisection: Numerical Recipes)

Brent's method (Secant+IQI+Bisection)

Matlab's fzero (Brent's method)

2) Examples and Demo's

3) Optimization algorithms to find $\min(f(x))$ on $[a,b]$

Bracketing Algorithms: Golden Section Search

Interpolation Algorithms: Successive Parabolic Interpolation

Hybrid Methods: fminbnd

Hybrid Methods:

Design Goals:

- 1) Robustness: Given a bracket $[a,b]$, maintain the bracket
- 2) Efficiency: Use superlinear convergent methods when possible

Some Options:

Have derivatives:

NewtonSafe (or RootSafe, Numerical Recipes)

Newton's method within a bracket, Bisection otherwise

No Derivatives:

Brent's Algorithm (zbrent Numerical Recipes, **fzero** Matlab)

Returns minimum bracket using combination of

Bisection

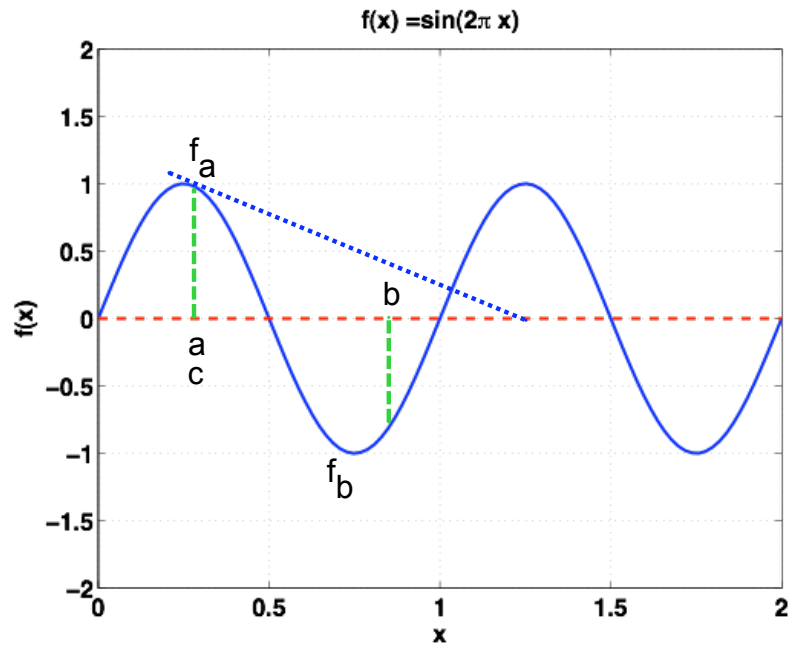
Secant method

Inverse Quadratic Interpolation

NewtonSafe: Bisection + Newton

The Geometric picture

Example: $f(x) = \sin(2\pi x)$



NewtonSafe Algorithm:

NewtSafe Code:

```

function [xFinal, xN, errorN] = newtSafe(func,a,b,ta)
% NEWTSafe - root-finder using hybrid Newton-bisection method to always maintain bracket
%
% [xFinal, xN, errorN] = newtSafe(func,a,b,ta)
%
% func: function() [ f, df]=func(x); returns both the function and its derivative
% a,b: initial bracket
% tol: stopping condition for error f(x) <= tol or b-a<=tol
% xFinal: final value
% xN: vector of intermediate iterates
% errorN: vector of errors
MAX_ITERATIONS = 100;
VERBOSE = true;

% initialize the problem

h = b-a;
fa = func(a);
fb = func(b);
if (sign(fa) == sign(fb))
    error('function must be bracketed to begin with');
end

c = a; % start on the left side (could also choose the middle)
[ f, df]=func(c);
xN(1) = c;
errorN(1,:) = [ abs(f(c)) h ];

% begin iteration until convergence or Maximum iterations
for i = 1:MAX_ITERATIONS
    % try a Newton step
    useNewton = true;
    c = c - f(c)/df(c);

    % if not in bracket choose bisection
    if (-fa <= c & b >= c)
        c = a + h/2;
        useNewton = false;
    end

    % Evaluate function and derivative at new c
    [f,df]=func(c);

    % check and maintain bracket
    if ( sign(f(c)) == sign(fb) )
        and =
            fa=f;
        else
            b = c;
            fb = f;
        end
        h = b-a;

    % calculate errors and track solutions
    absError = abs(f(c));
    relError = c;
    xN(i) = c;
    errorN(i,:) = [ absError, relError ];

    % do pretty
    if VERBOSE
        if useNewton
            disp(sprintf('Newt: a=%12.8e, b=%12.8e, c=%12.8e, f(c)=%12.8e, h=%12.8e', A,B,c,f(h), h));
        else
            disp(sprintf('Bisect: a=%12.8e, b=%12.8e, c=%12.8e, f(c)=%12.8e, h=%12.8e', A,B,c,f(h), h));
        end
    end

    % check if converged
    if ( absError < tol || relError < ta*h )
        break;
    end
end

% clean up
if ( i == MAX_ITERATIONS )
    warning('Maximum iterations exceeded');
end
xFinal = xN(end);
xN = xN'; % convert output to column vectors

```

Brent-Dekker Algorithm: Hybrid method using IQI+Secant+Bisection (foolproof)

Given: $f(x)$ and a bracket $[a,b]$

Initialize: use Secant method to find c between a and b

Until Converged: $\text{abs}(b-a) < \text{tol} * b$ or $f(c)=0$

Arrange a, b and c so that

- a and b form a bracket
- $|f(b)| \leq |f(a)|$
- c is the previous value of b

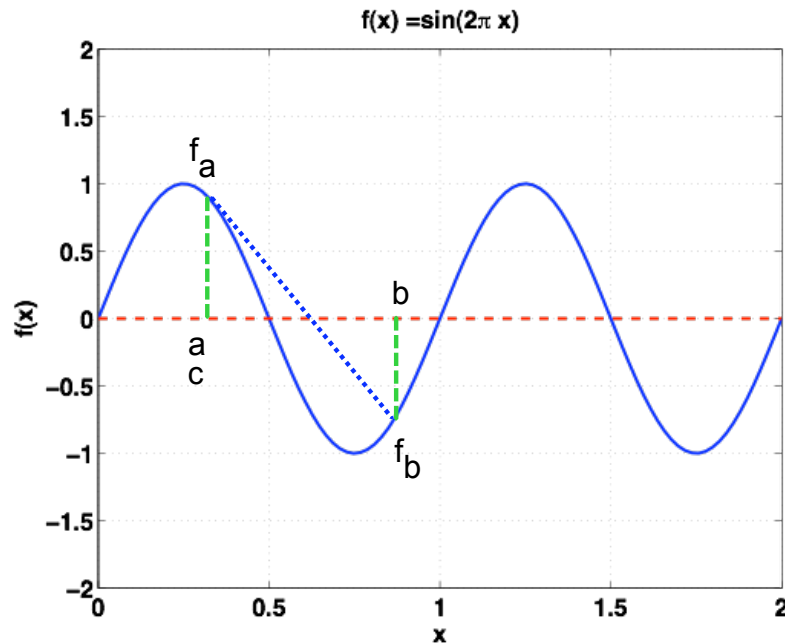
if $c \neq a$:
 Try $c=IQI$
elseif $c=a$
 Try $c=Secant$
end

if c in the bracket
 keep it
else
 use $c=Bisection$
end

Brent's method: Bisection + Secant + IQI

The Geometric picture

Example: $f(x)=\sin(2\pi x)$



Brent-Dekker Algorithm:

Comments:

This algorithm is Bullet-proof

It's guaranteed to always maintain a bracket

Doesn't require derivatives

Uses rapidly converging methods when reliable

Uses slow but sure method when necessary

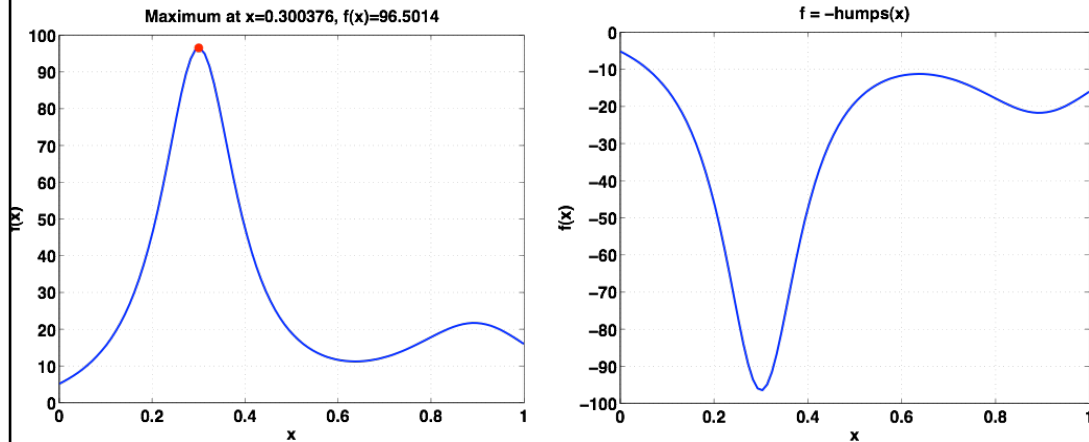
In **Matlab**: `fzero`...demonstrate with `fzerogui` (code in `fzerotxt`)

```
basic syntax (see help fzero, help optimset)
options = optimset('disp','iter')
x = fzero(@func,x0,options)
```

if `x0` is scalar, it searches for a bracket if `x0 = [a b]` it tests for a bracket and fails if $\text{sign}(f(a)) \neq \text{sign}(f(b))$

Optimization (finding extrema) for functions of one variable

Closely related problem to root finding, but rather than finding $f(x)=0$ on some interval. Find $\min(f(x))$ on some interval...



Optimization (finding extrema) for functions of one variable

General algorithms (similar to rootfinding algorithms):

Bracketing algorithms: Golden-Section Search (linear convergence)

Interpolation algorithms: repeated parabolic interpolation

Hybrid algorithms: Matlab's `fminbnd(func,a,b,tol)`

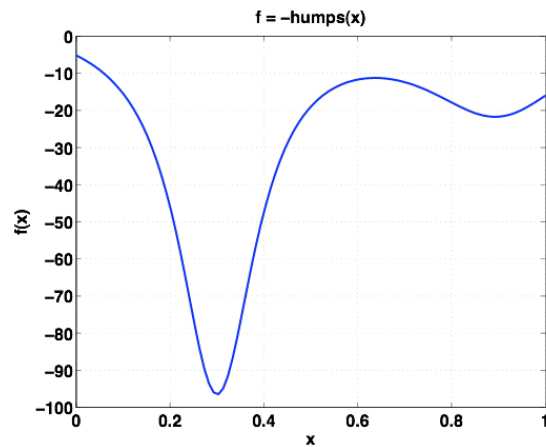
Bracketing Algorithm: Golden Section Search

Like Bisection: given $f(x)$ in $C[a,b]$ that is convex (uni-modal) over an interval $[a,b]$ reduce the interval size until it "brackets" the minimum

Note: bracketing not as well defined, you should always plot your function

Questions:

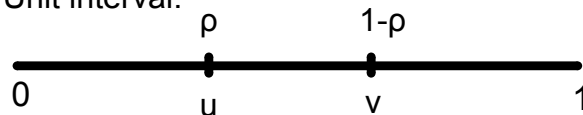
- 1) How many points are required to approximate a minimum?
- 2) How many points are needed to subdivide? $[a,b]$?
- 3) How to choose those points efficiently



Bracketing Algorithm: Golden Section Search

How to divide an interval for successive minima brackets...

Consider the Unit interval:



this 

or 

Golden Section Search: The Algorithm

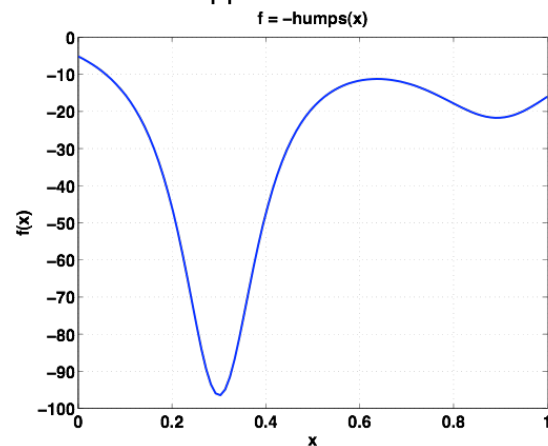
Given $f(x)$ and unimodal bracket $[a,b]$



Initialize:

Interpolation Algorithm: Successive Parabolic Interpolation

Like Secant: use multiple samplings of the function to approximate the function.



Successive Parabolic Interpolation: The Algorithm

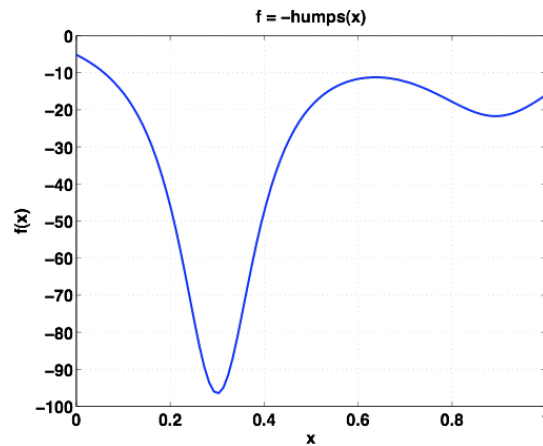
Given: $f(x)$ and $[a,b]$

initialize: $x = [a \ b \ (a+b)/2]$
 $n=2:-1:0;$

```

for i=1:MAX_ITERATIONS
    f = func(x);
    p = polyfit(x,f,2);
    pPrime = n.*p;
    xNew(i) = -pPrime(2)/pPrime(1);
    x = [ x(2:end) xNew(i) ];
    relErr = abs(xNew(i)-xNew(i-1))/abs(xNew(i));
    if relErr < tol
        break
    end
end
end

```



Hybrid schemes: fminbnd successive Parabolic interpolation + golden section search

in **Matlab**: use `fminbnd`

syntax:

```

options = optimset('disp','iter')
x = fminbnd(@func,x1,x2,options)

```

examples:

```

x = fminbnd(@(x) sin(2*pi*x),0,1,options)
x = fminbnd(@(x) -humps(x),.1,.4,options)

```

