Root finding

Root finding

- suppose you have a mathematical function **`f(x)`** and you want to find x_0 such that $f(x_0) = 0$
  - why would you want to do this?
  - many problems in computer science, science, and engineering reduce to optimization problems
    - find the shape of an automobile that minimizes aerodynamic drag
    - find an image that is similar to another image (minimize the difference between the images)
    - find the sales price of an item that maximizes profit
  - if you can write the optimization criteria as a function **`g(x)`** then its derivative **`f(x) = dg/dx = 0`** at the minimum or maximum of **`g`** (as long as **`g`** has certain properties)

Roots of polynomials

- for roots of polynomials MATLAB has a function named **`roots`**
- **`roots`** finds all of the roots of a polynomial defined by its coefficients vector; e.g.,
  - the roots of the polynomial $x^3 - 6x^2 - 72x - 27$:

```
p = [1 -6 -72 -27];
r = roots(p)
r =
    12.1229
    -5.7345
    -0.3884
```

Roots of non-polynomials

- we've already seen Newton's method for root finding (Day 12)

1. start with an initial estimate of the root x_0
2. $i = 0$
3. while $|f(x_i)| > \epsilon$

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

   $$i = i + 1$$

- this requires computation of both $f(x)$ and $f'(x)$

Newton's method

- our previous implementation used local functions to represent both $f(x)$ and $f'(x)$
- the problem with this approach is that we can only find roots of the local function that defines $f(x)$
  - e.g., our previous implementation can only find the roots of $f(x) = x^2 - 1$

```
function [ root, xvals ] = newton(x0, epsilon)
%NEWTON Newton's method for x^2 - 1
%   ROOT = NEWTON(X0, EPSILON) finds a root of f(x) = x^2 - 1 using
%   Newton's method starting from an initial estimate X0 and a tolerance EPSILON
%
%   [ROOT, XVALS] = NEWTON(X0, EPSILON) also returns the iterative estimates
%   in XVALS

xvals = x0;
xi = x0;
while abs(f(xi)) > epsilon
   xj = xi - f(xi) / fprime(xi);
   xi = xj;
   xvals = [xvals xi];
end
root = xi;

end

function [ y ] = f(x)
y = x * x - 1;
end

function [ yprime ] = fprime(x)
yprime = 2 * x;
end
```

can only find the root of this function

Function handles

- it would be nice if we could tell our implementation of Newton's method what function to use for $f(x)$ and $f'(x)$
- MATLAB does not allow you to pass a function directly to another function
  - instead you must pass a *function handle* to the function

- a function handle is a value that you can use to call a function (instead of using the name of the function)
  - because it is a value, you can store it in a variable!

Function handles

- you can create a handle for any function by using @ before the function name

```
linspaceHandle = @linspace; % handle for linspace
cosHandle = @cos;           % handle for cos
plotHandle = @plot;         % handle for plot
```

Function handles

- you can use the handle to call the function exactly the same way that you would use the function name to call the function

```
% use handle to call linspace
x = linspaceHandle(-1, 1, 50);

% use handle to call cos
y = cosHandle(2 * pi * x);

% use handle to call plot
plotHandle(x, y, 'b:');
```

Function functions

- by using function handles, we can modify our implementation of Newton's method to find the root of any function
  - we just have to supply two function handles, one for $f(x)$ and a second for $f'(x)$
  - we also need MATLAB functions that implement $f(x)$ and $f'(x)$

function handles

```
function [ root, xvals ] = newton(f, fprime, x0, epsilon)
%NEWTON Newton's method for root finding
%   ROOT = NEWTON(F, FPRIME, X0, EPSILON) finds a root of the
%   function F having derivative FPRIME using Newton's method
%   starting from an initial estimate X0 and a tolerance EPSILON
%
%   [ROOT, XVALS] = NEWTON(F, FPRIME, X0, EPSILON) also returns
%   the iterative estimates in XVALS

xvals = x0;
xi = x0;
while abs(f(xi)) > epsilon
   xj = xi - f(xi) / fprime(xi);
   xi = xj;
   xvals = [xvals xi];
end
root = xi;

end
```

local functions have been removed

Function functions

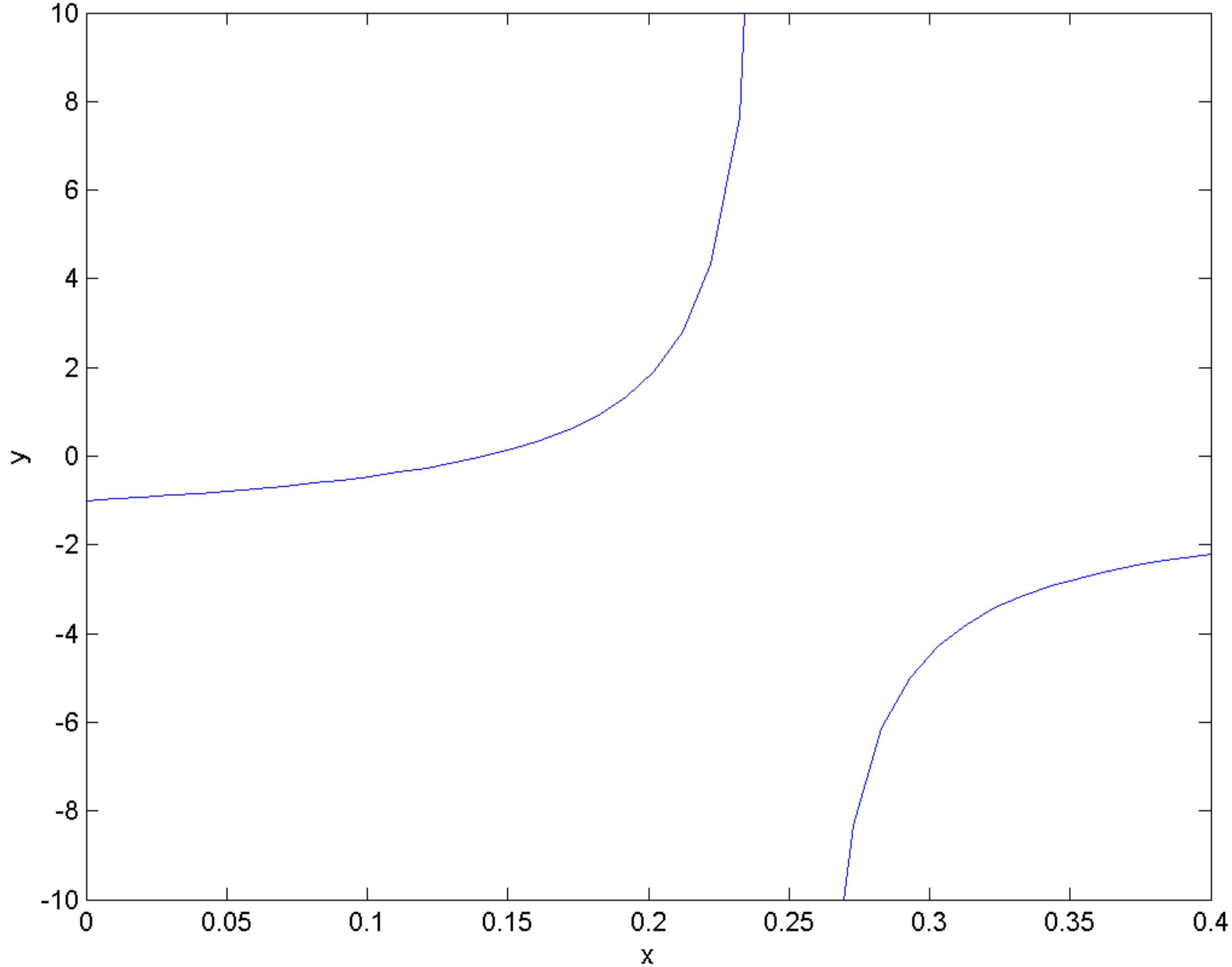
- let's find a root of

$$f(x) = \sqrt{x}\tan(\pi\sqrt{x}) - \sqrt{1-x}$$

which has the derivative

$$f'(x) = \frac{1}{2}\left(\frac{1}{\sqrt{1-x}} + \frac{\tan(\pi\sqrt{x})}{\sqrt{x}} + \pi\sec^2(\pi\sqrt{x})\right)$$

plot of $f(x)$



```
function [y] = myf(x)
%MYF Function to find the root of

y = sqrt(x) .* tan(pi * sqrt(x)) - sqrt(1 - x);

end
```

```matlab
function [y] = myfprime(x)
%MYFPRIME Derivative of MYF

sqrtx = sqrt(x);
a = 1 / sqrt(1 - x);
b = tan(pi * sqrtx) / sqrtx;
c = pi * (sec(pi * sqrtx))^2;
y = 0.5 * (a + b + c);

end
```

Function functions

- we can now use our Newton's method implementation by passing in function handles for **`f`** and **`fprime`**

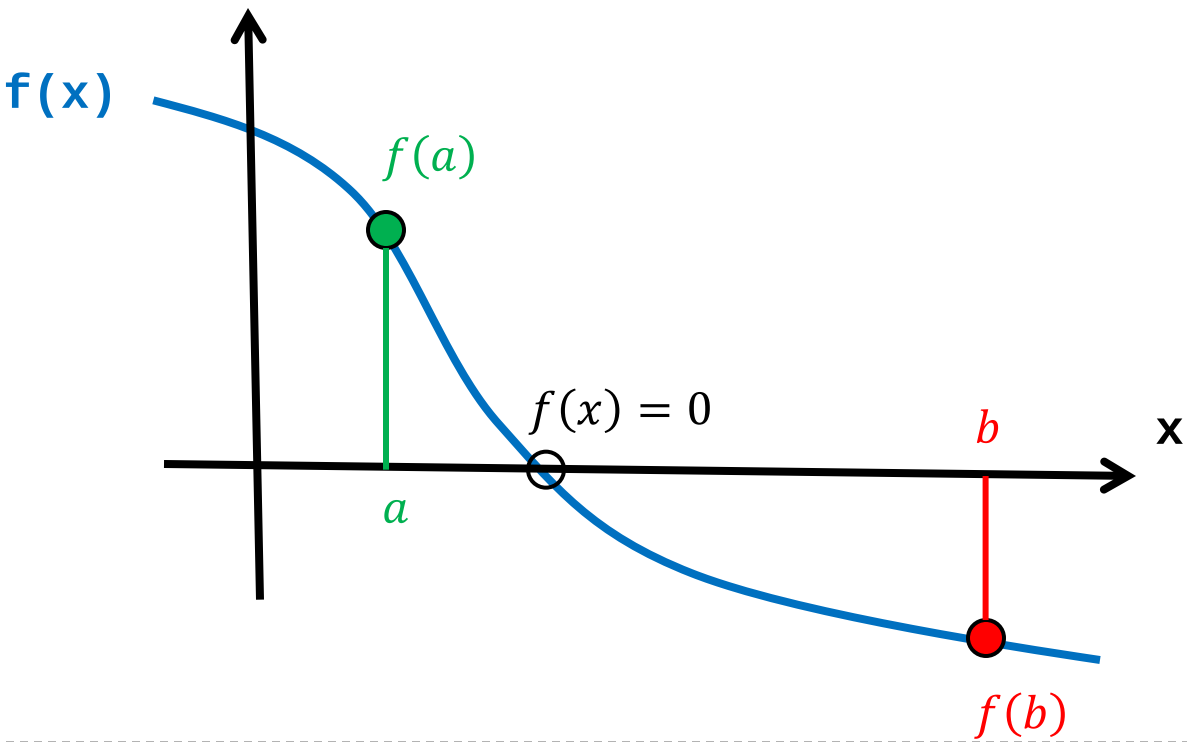
```
newton(@myf, @myfprime, 0.1, 1e-6)
```

Bracketing methods

- Newton's method requires an initial estimate of the root and the derivative of the function that we want to find the roots for
- bracketing methods do not require the derivative

Bracketing methods

- bracketing methods require two estimates $x_1 = a$ and $x_2 = b$ such that the root lies between the two estimates

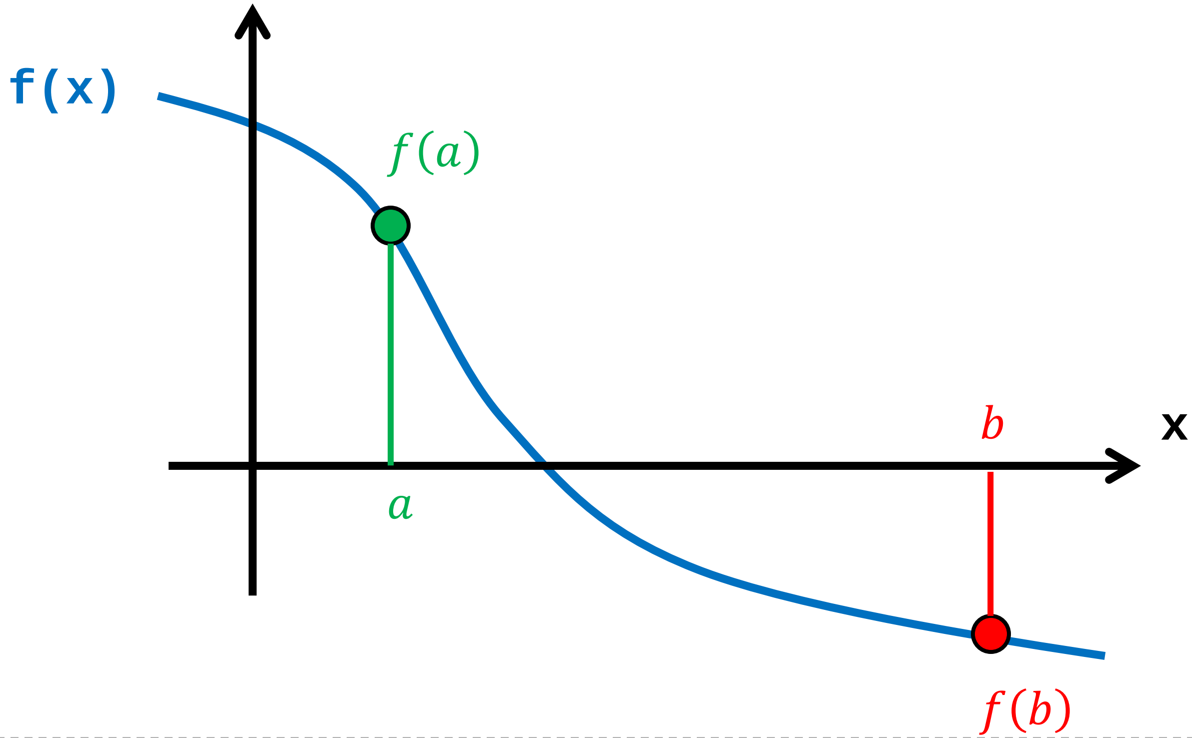


Bisection method

- the bisection method repeatedly evaluates f at the midpoint c_i of the interval $[a_i, b_i]$
  - c_i becomes one of the new interval endpoints $[a_{i+1}, b_{i+1}]$ depending of the sign of $f(c_i)$

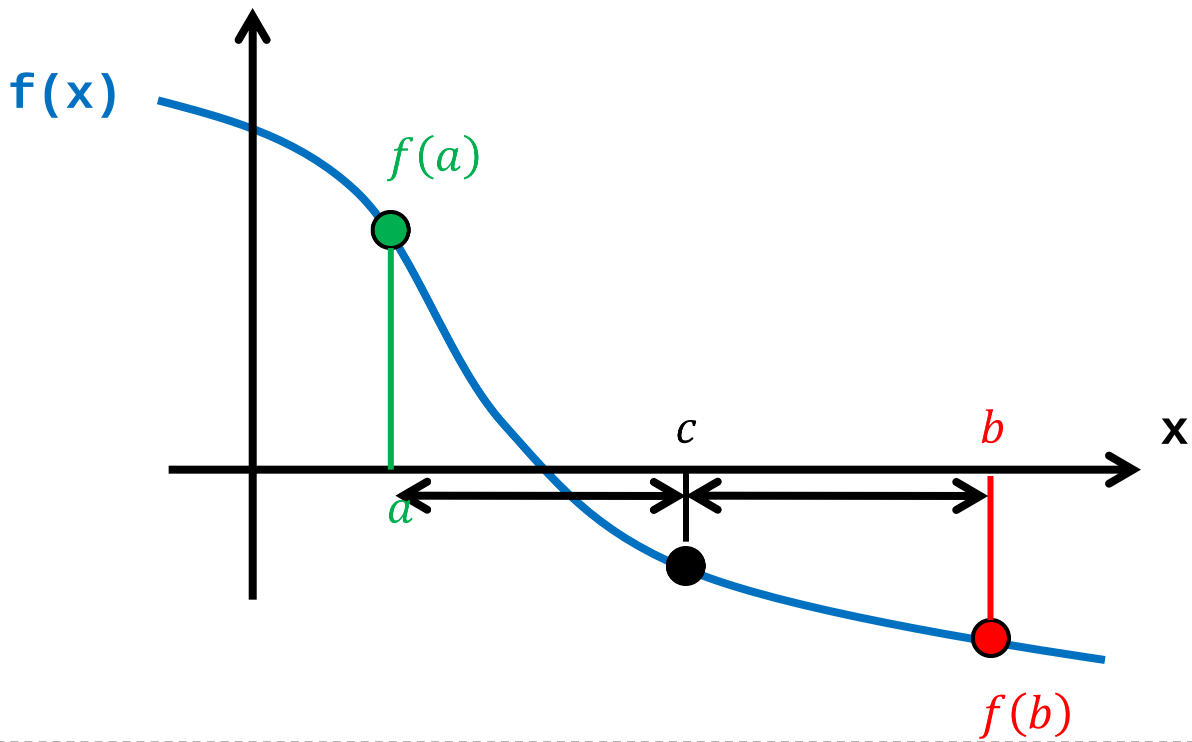
Bisection Method

- evaluate **`f(x)`** at two points **`x = a`** and **`x = b`** such that
  - **`f(a) > 0`**
  - **`f(b) < 0`**



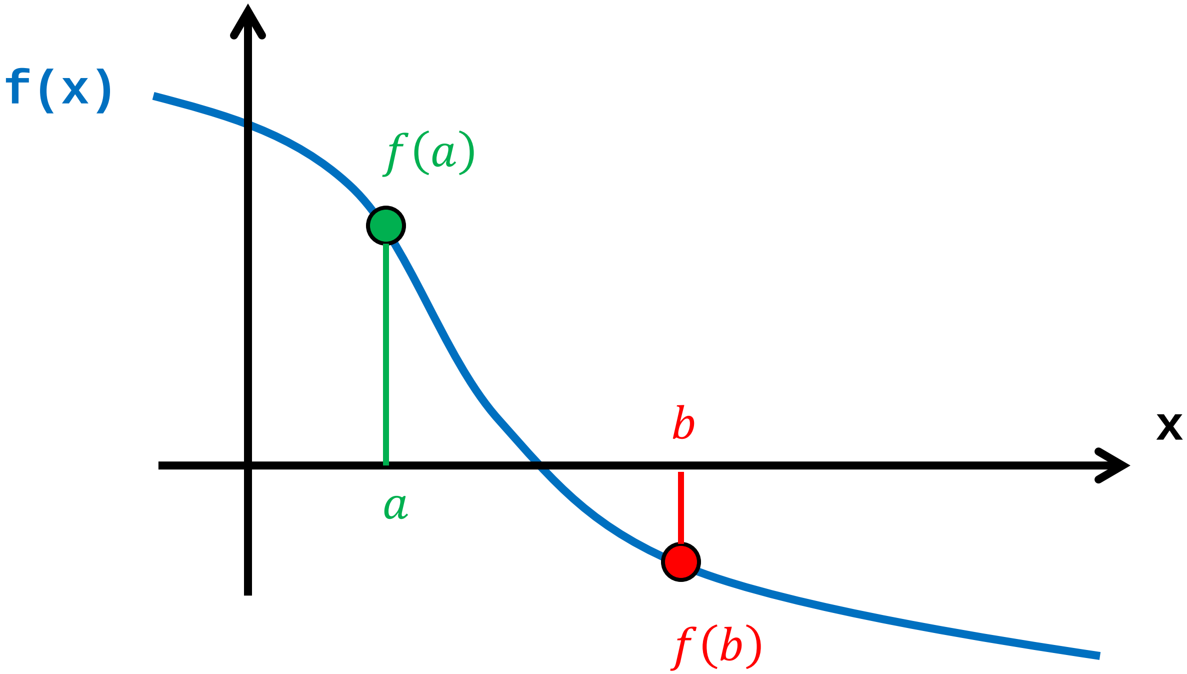
Bisection method

- evaluate **`f(c)`** where **`c`** is halfway between **`a`** and **`b`**
  - if **`f(c)`** is not close to zero, repeat the bisection using **`c`** as one of the new endpoints



Bisection method

- in this example, the value of **`f(c)`** is not yet close enough to zero
  - **`c`** becomes the new **`b`** (because the sign of **`f(c)`** is negative) and the process repeats



Bisection method

- the method stops when **`f(c)`** becomes close enough to zero, and **`c`** is the estimate of the root of **`f`**