APPLICATIONS OF MATLAB IN ENGINEERING

Yan-Fu Kuo
Dept. of Bio-industrial Mechatronics Engineering
National Taiwan University

Fall 2015

Today:

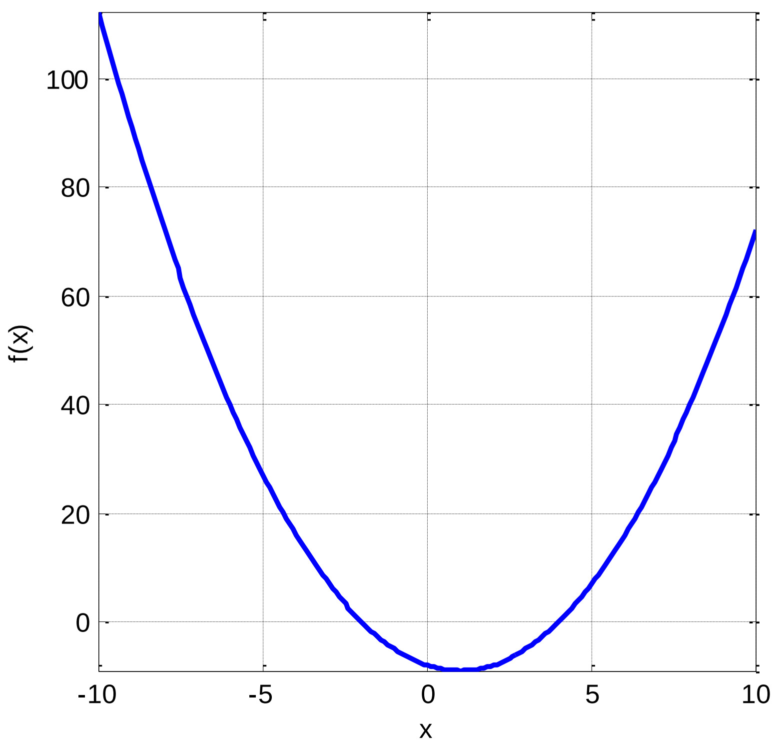
- Symbolic approach
- Numeric root solvers
- Recursive functions

Problem Statement

- Suppose you have a mathematical function $f(x)$ and you want to find x_0 such that $f(x_0) = 0$, e.g.

$$f(x) = x^2 - 2x - 8 = 0$$

- How do you solve the problem using MATLAB?
  - Analytical Solutions
  - Graphical Illustration
  - Numerical Solutions



Symbolic Root Finding Approach

- Performing mathematics on symbols, NOT numbers
- The symbols math are performed using “symbolic variables”
- Use `sym` or `syms` to create symbolic variables

```
syms x
x + x + x
(x + x + x)/4
```

```
x=sym('x');
x + x + x
(x + x + x)/4
```

- Define: $y = x^2 - 2x - 8$

Symbolic Root Finding: `solve()`

- Function `solve` finds roots for equations

$$y = x \cdot \sin(x) - x = 0$$

```
syms x
solve('x*sin(x)-x', x)
```

```
syms x
y = x*sin(x)-x;
solve(y, x)
```

- Find the roots for:

$$\cos(x)^2 - \sin(x)^2 = 0 \quad \text{and} \quad \cos(x)^2 + \sin(x)^2 = 0$$

Solving Multiple Equations

- Solve this equation using symbolic approach:

$$\begin{cases} x - 2y = 5 \\ x + y = 6 \end{cases}$$

```
syms x y
eq1 = x - 2*y - 5;
eq2 = x + y - 6;
A = solve(eq1,eq2,x,y)
```

Solving Equations Expressed in Symbols

- What if we are given a function expressed in symbols?

$$ax^2 - b = 0$$

```
syms x a b
solve('a*x^2-b')
```

- x is always the first choice to be solved
- What if one wants to express b in terms of a and x?

```
syms x a b
solve('a*x^2-b', 'b')
```

Exercise

- Solve this equation for x using symbolic approach

$$(x-a)^2+(y-b)^2=r^2$$

- Find the matrix inverse using symbolic approach

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Symbolic Differentiation: `diff()`

- Calculate the derivative of a symbolic function:

$$y = 4x^5$$

```
syms x
y = 4*x^5;
yprime = diff(y)
```

- Exercise:

$$f(x) = \frac{e^{x^2}}{x^3 - x + 3}, \qquad \frac{df}{dx} = ?$$

$$g(x) = \frac{x^2 + xy - 1}{y^3 + x + 3}, \qquad \frac{\partial f}{\partial x} = ?$$

Symbolic Integration:

- Calculate the integral of a symbolic function:

$$z = \int y dx = \int x^2 e^x dx, \qquad z(0) = 0$$

```
syms x;   y = x^2*exp(x);
z = int(y);   z = z-subs(z, x, 0)
```

- Exercise:

$$\int_0^{10} \frac{x^2 - x + 1}{x + 3} dx$$

Symbolic vs. Numeric

	Advantages	Disadvantages
Symbolic	• Analytical solutions • Lets you intuit things about solution form	• Sometimes can't be solved • Can be overly complicated
Numeric	• Always get a solution • Can make solutions accurate • Easy to code	• Hard to extract a deeper understanding

Review of Function Handles (@)

- A handle is a pointer to a function
- Can be used to pass functions to other functions
- For example, the `input` of the following function is another function:

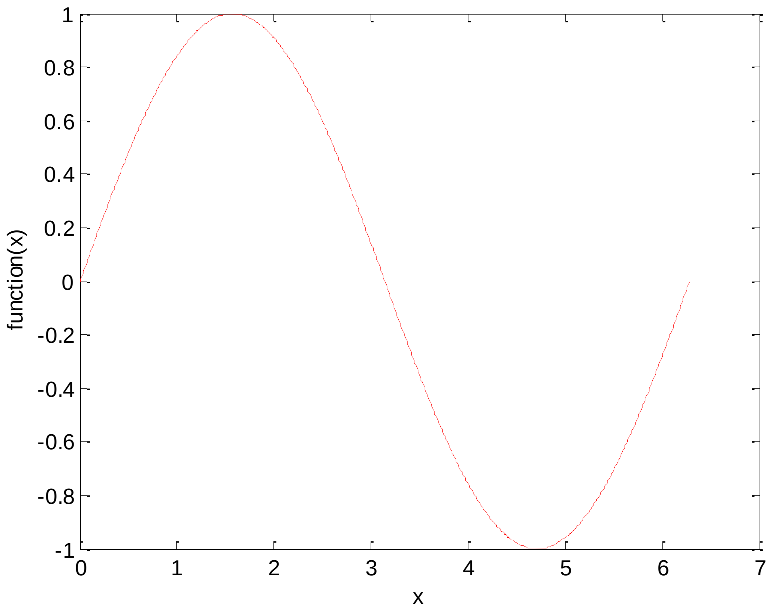
```
function [y] = xy_plot(input,x)
% xy_plot receives the handle of a function and plots that
% function of x
y = input(x);  plot(x,y,'r--');
xlabel('x');  ylabel('function(x)');
end
```

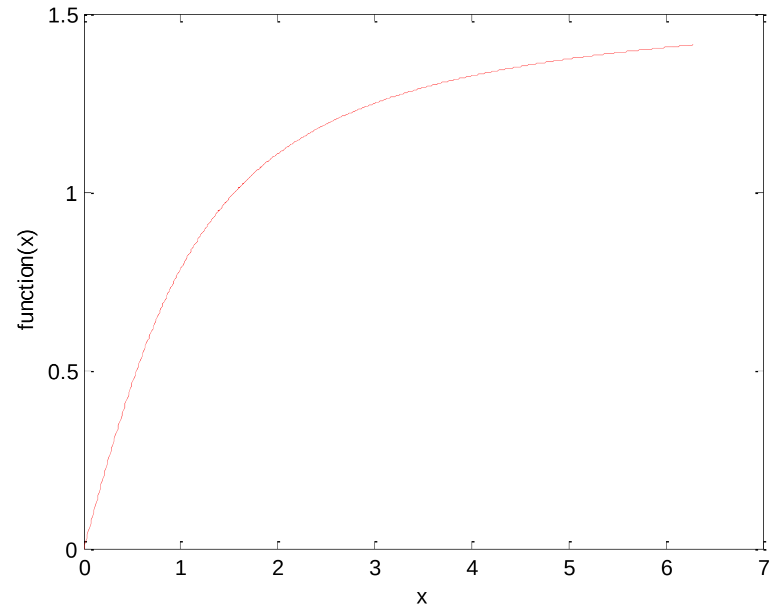
- Try: `xy_plot(@sin,0:0.01:2*pi);`

Using Function Handles

```
xy_plot(@sin,0:0.01:2*pi);
```

```
xy_plot(@atan,0:0.01:2*pi);
```





fsolve()

- A numeric root solver
- For example, solve this equation:

$$f(x) = 1.2x + 0.3 + x \cdot \sin(x)$$

```
f2 = @(x) (1.2*x+0.3+x*sin(x));
fsolve(f2,0)
```

A function handle

Initial guess

Exercise

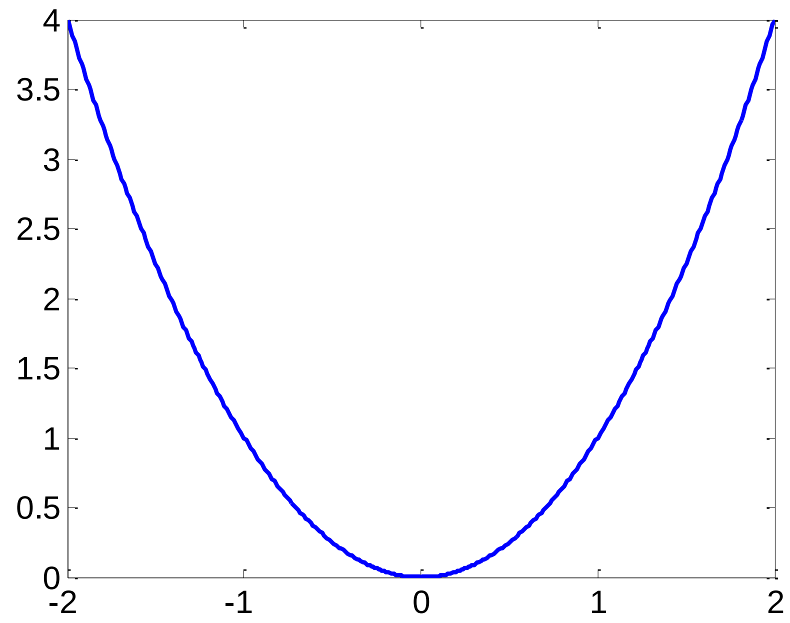
- Find the root for this equation :

$$f(x,y)=\begin{cases}2x-y-e^{-x}\\-x+2y-e^{-y}\end{cases}$$

using initial value $(x,y)=(-5,-5)$

fzero()

- Anther numeric root solver
- Find the zero if and only if the function crosses the x-axis




```
f=@(x)x.^2
fzero(f,0.1)
```

```
fsolve(f,0)
```

- Options:

Number of iterations Tolerance

```
f=@(x)x.^2
options=optimset('MaxIter',1e3,'TolFun',1e-10);
fsolve(f,0.1,options)
fzero(f,0.1,options)
```

Finding Roots of Polynomials: `roots()`

- Find the roots of this polynomial:

$$f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$$

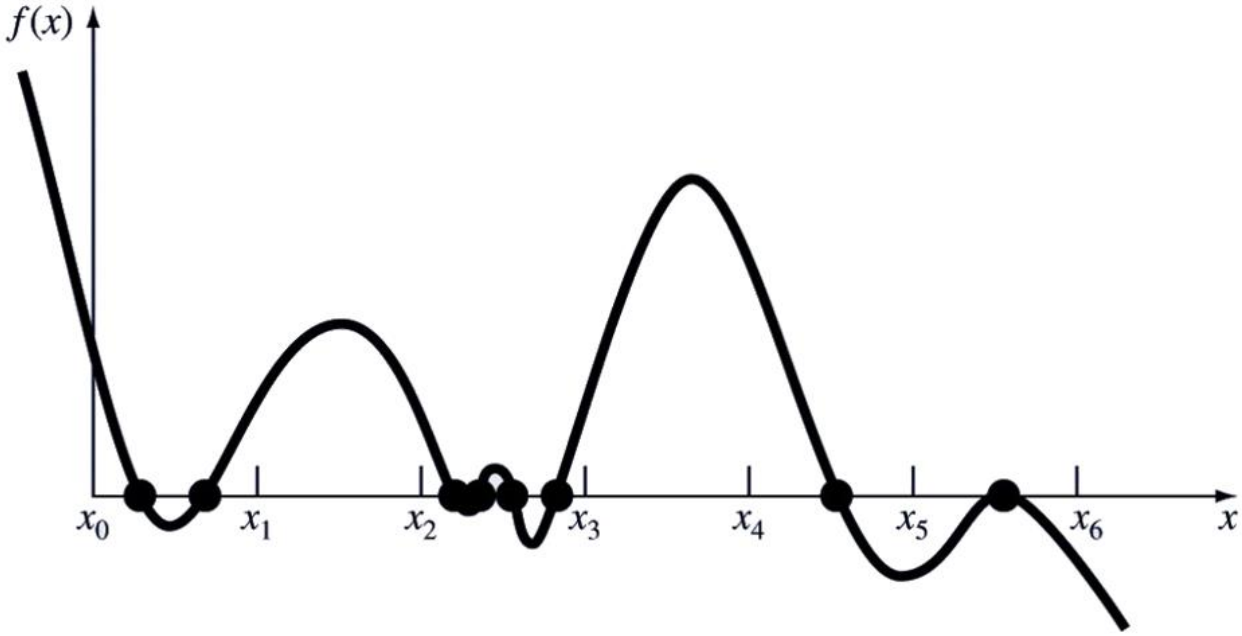
```
roots([1 -3.5 2.75 2.125 -3.875 1.25])
```

- `roots` only works for polynomials
- Find the roots of the polynomial:

$$f(x) = x^3 - 6x^2 - 12x + 81$$

How Do These Solvers Find the Roots?

- Now we are going to introduce more details of some numeric methods



Numeric Root Finding Methods

- Two major types:
  - Bracketing methods (e.g., bisection method)

    Start with an interval that contains the root
  - Open methods (e.g., Newton-Raphson method)

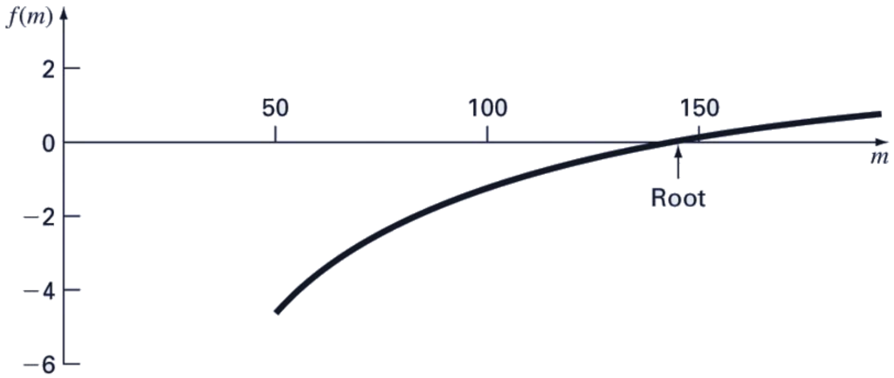
    Start with one or more initial guess points

- Roots are found iteratively until some criteria are satisfied:
  - Accuracy
  - Number of iteration

Bisection Method (Bracketing)

Assumptions:

- $f(x)$ continuous on $[l, u]$
- $f(l) \cdot f(u) < 0$



Algorithm:

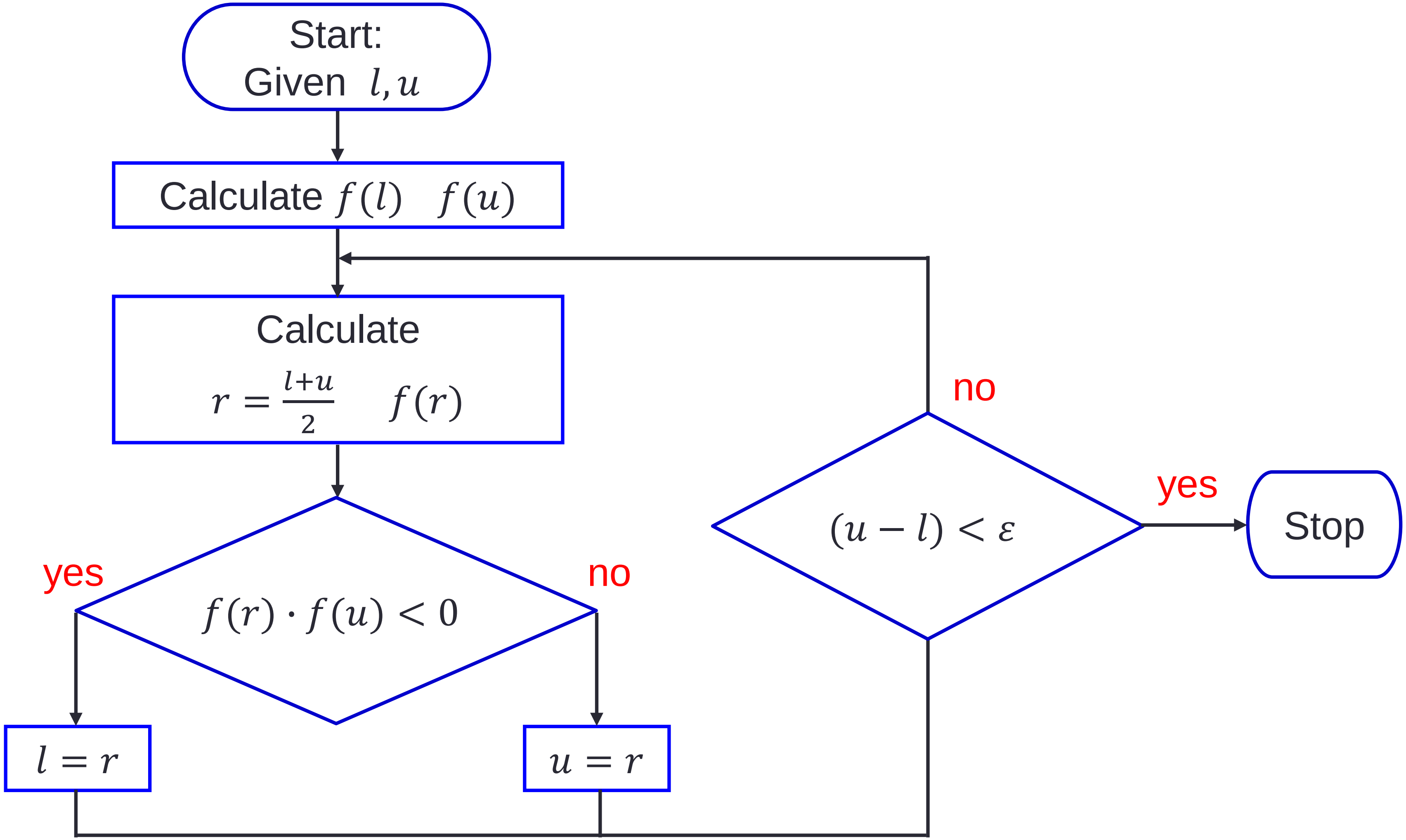
Loop

1. $r = (l + u)/2$
2. If $f(r) \cdot f(u) < 0$ then new interval $[r, u]$

   If $f(l) \cdot f(r) < 0$ then new interval $[l, r]$

End

Bisection Algorithm Flowchart



Newton-Raphson Method (Open)

Assumption:

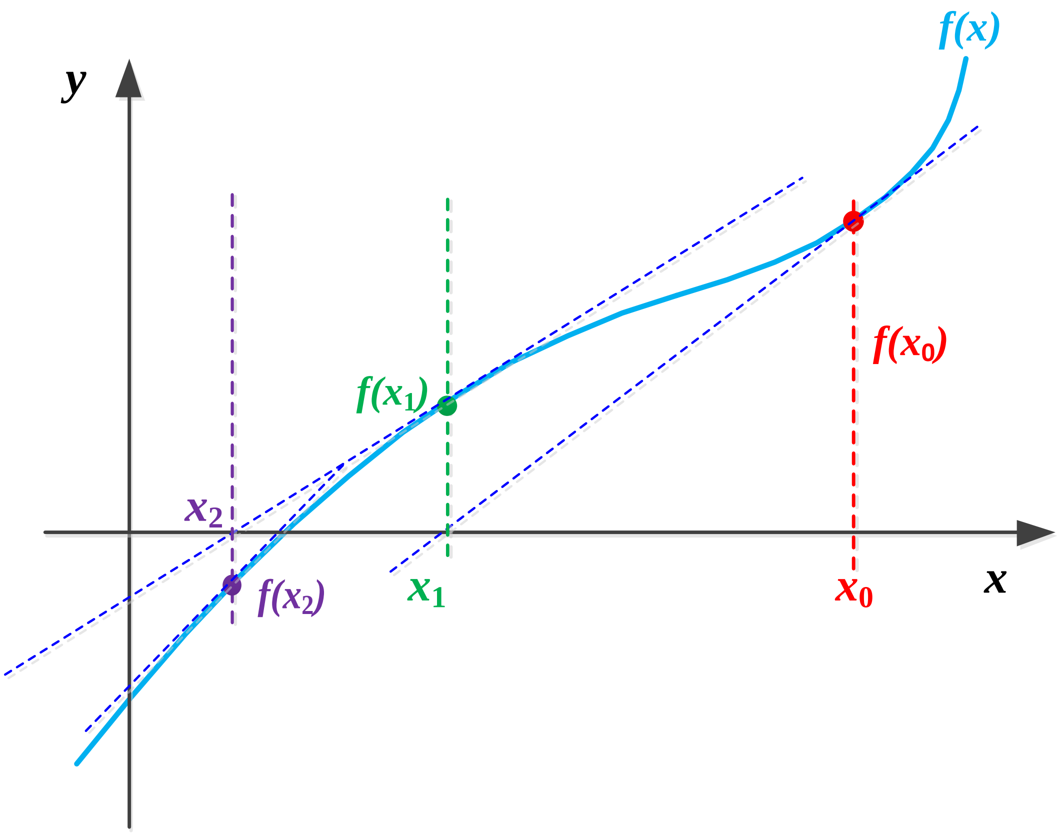
- $f(x)$ continuous
- $f'(x)$ known

Algorithm:

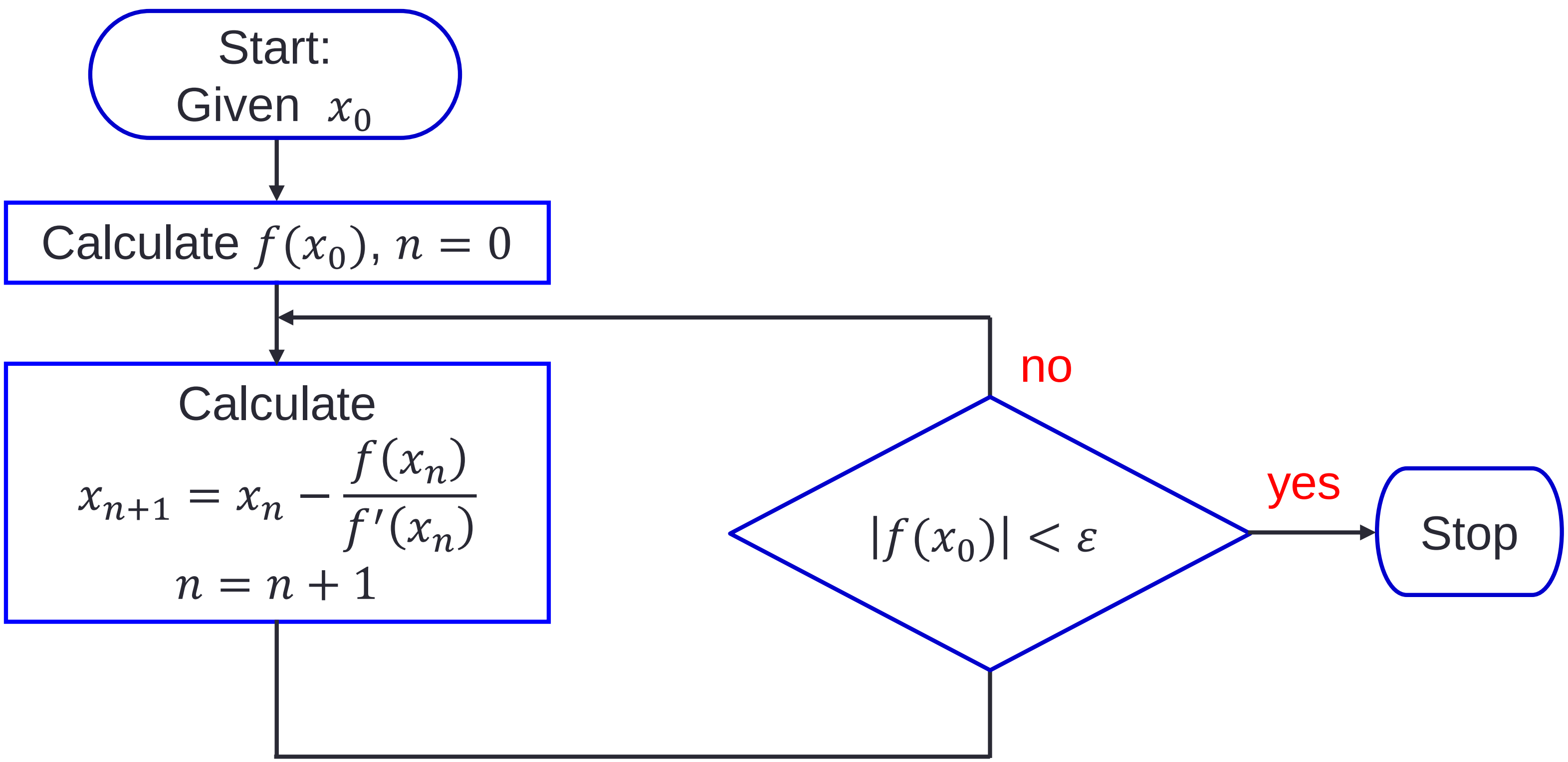
Loop

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

End



Newton-Raphson Algorithm Flowchart



Bisection vs. Newton-Raphson

Bisection	• Reliable • No knowledge of derivative is needed • Slow • One function evaluation per iteration • Needs an interval [a,b] containing the root, $f(a)\cdot f(b)<0$
Newton	• Fast but may diverge • Needs derivative and an initial guess x_0, $f'(x0)$ is nonzero

Recursive Functions

- Functions that call themselves
- Example, factorial of an integer n

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

- A factorial can be defined in terms of another factorial:

$$\begin{aligned} n! &= n \times (n-1)! \\ &= n \times (n-1) \times (n-2)! \\ &= n \times (n-1) \times (n-2) \times (n-3)! \\ &= n \times (n-1) \times (n-2) \times \cdots \end{aligned}$$

Factorial Recursive Function

- The function includes a recursive case and a base case
- The function stops when it reaches the base case

```
function output = fact(n)
% fact recursively finds n!
if n==1
    output = 1;
else
    output = n * fact(n-1);
end
end
```

Base case

Recursive case

End of Class