Week 3

# Nonlinear Equations II

1. Review bisection method
2. Newton method
3. Secant method
4. Solve a system of two nonlinear equations

Week 3 part I

# Review bisection method

# Steps to solve problems

numerical schemes

↓

algorithm

↓

programs

↓

results

# Bisection Method (Bracketing method)

Given $x_1$ and $x_2$ such that

$f(x_1) \cdot f(x_2) < 0,$

Repeat

Set $x_3 = \frac{1}{2}(x_1 + x_2)$

If $f(x_1) \cdot f(x_3) < 0$

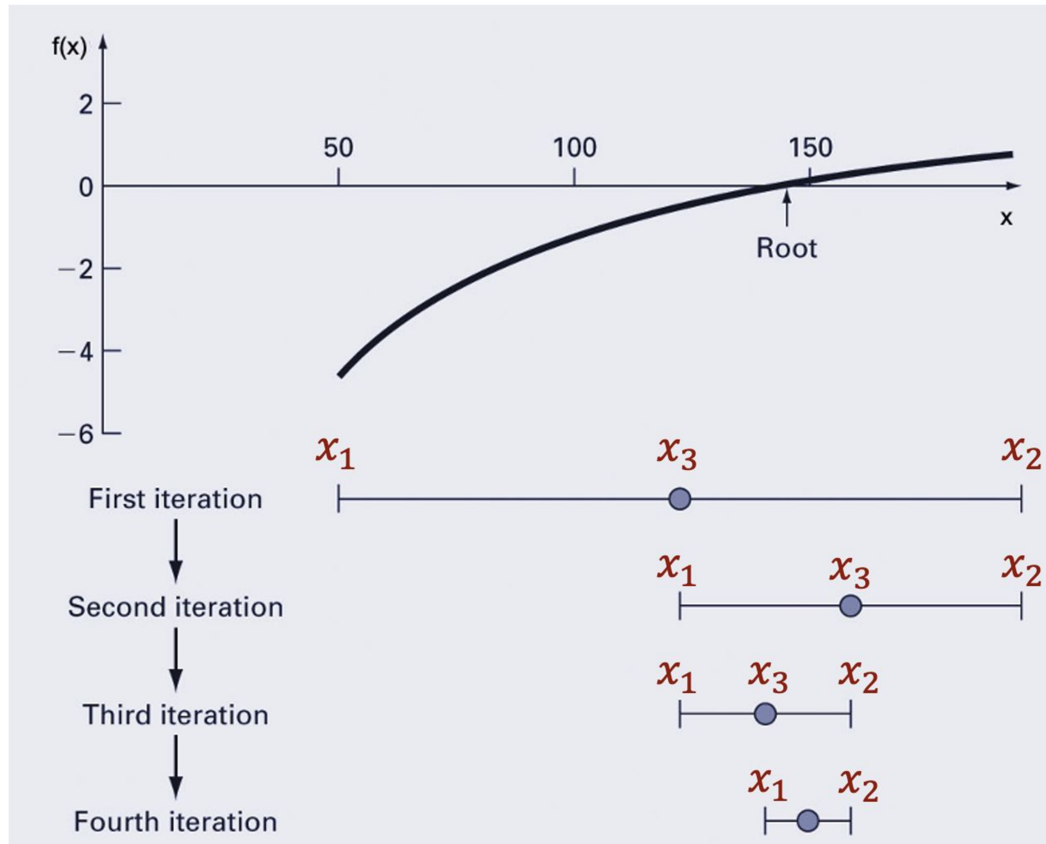
Set $x_2 = x_3$

Else

Set $x_1 = x_3$

End if

Until $(x_2 - x_1) < TOL$



# Programming Bisection (Python)

```
Import numpy as np

def F(x):
     y = 8-4.5*(x-np.sin(x))
return y

x1, x2, imax, TOL = 1, 3, 30, 0.001

if F(x1)*F(x2)>0:
     print('Error: The function has the same sign at points x1 and x2.')
else:
     for i in range(imax):
          x3 = (x1+x2)/2
          toli = (x2+x1)/2
          if F(x3) == 0:
               print('An exact solution x = %.6f was found'%x3)
               break
          if toli<TOL:
               break
          if i==imax:
               print('Solution was not obtained in %i iterations'%imax)
               break
          if F(x1)*F(x3)<0:
               x2 = x3
          else:
               x1 = x3
```

# Programming Bisection (MATLAB)

```
F=@(x) 8-4.5*(x-sin(x));

a = 1; b = 3; imax = 30; tol = 0.001
Fa=F(a); Fb=F(b);
if Fa*Fb > 0
    disp('Error: The function has the same sign at points a and b.’)
else
    disp('iteration    a    b (xNS) Solution    f(xNS)  Tolerance')
    for i = 1:imax
        xNS = (a + b)/2;
        toli=(b-a)/2;
        FxNS=F(xNS);
        fprintf('%3i    %11.6f %11.6f %11.6f  %11.6f %11.6f\n',i, a, b, xNS, FxNS, toli)
        if FxNS == 0
            fprintf('An exact solution x =%11.6f was found',xNS)
            break
        end
        if toli < tol
            break
        end
        if i == imax
        fprintf('Solution was not obtained in %i iterations',imax)
        break
        end
        if F(a)*FxNS < 0
            b = xNS;
        else
        a = xNS;
        end
    end
end
```

# Newton Method (Open method)

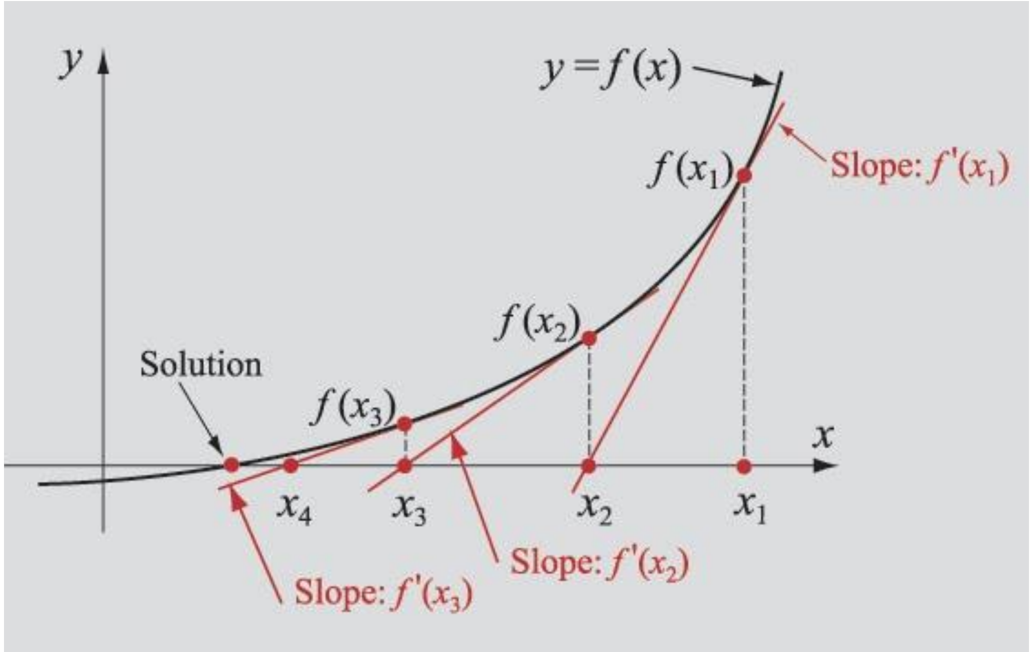
Given $x_1$ reasonably close to the root,

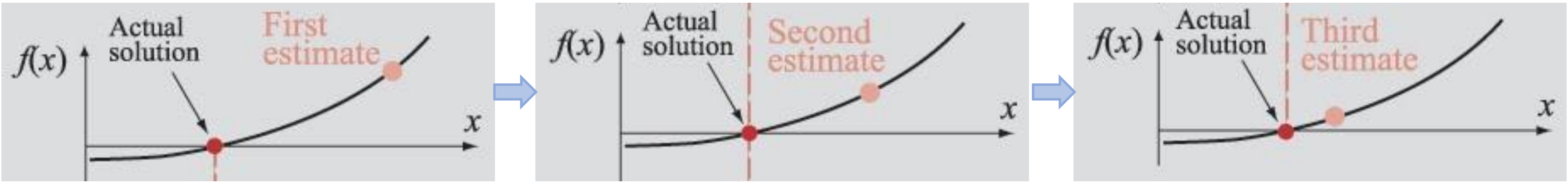
Repeat

Compute $f(x_1)$, $f'(x_1)$

Set $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$

Until $|x_2 - x_1| < TOL$



# Secant Method (Open method)

Given $x_0$ and $x_1$ that are near to the root,

Repeat

Set $x_2 = x_1 - f(x_1) \cdot \frac{x_0 - x_1}{f(x_0) - f(x_1)}$

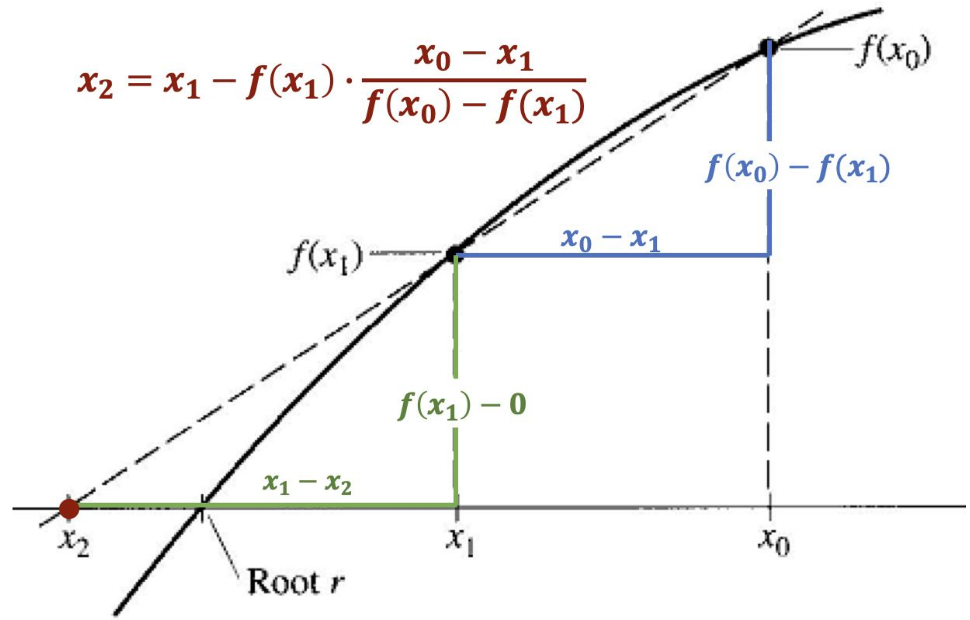
Set $x_0 = x_1$

Set $x_1 = x_2$

Until $|x_0 - x_1| < TOL$

$$\frac{x_1 - x_2}{f(x_1) - 0} = \frac{x_0 - x_1}{f(x_0) - f(x_1)}$$

$$x_2 = x_1 - f(x_1) \cdot \frac{x_0 - x_1}{f(x_0) - f(x_1)}$$



?

- When do we use Secant method instead of Newton method?
- Any exceptions for Bisection and Newton method?

# Introduction to functions in Python and MATLAB

**Python**

## **fsolve** from scipy

Roots of nonlinear equations defined by f(x)=0

Sol = fsolve(function,x0)

x0: The starting estimate for the roots

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html

**MATLAB**

## **fzero**

Root of nonlinear function

Sol = fzero(function,x0)

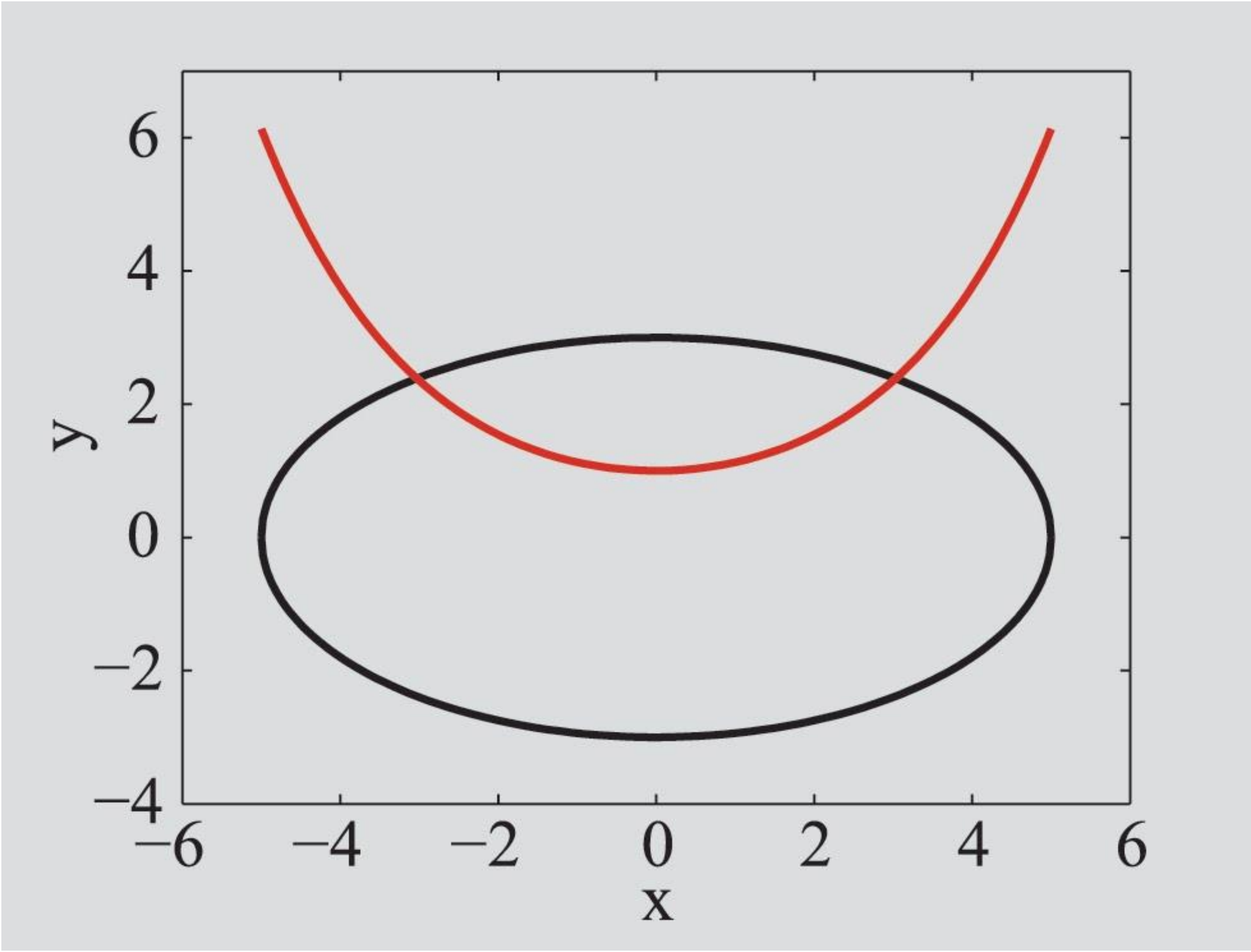
x0: value of x near to where the function crosses the axis

https://www.mathworks.com/help/matlab/ref/fzero.html

Week 3 part IV

# Solve A System of Two Nonlinear Equation

# Equation sets



# A system of two nonlinear equations

$$f_1(x,y)=0$$
$$f_2(x,y)=0$$

If x2 and y2 are the true solutions (but unknown) of the system and are sufficiently close to x1 and y1, then the values of f1 and f2 at x2 and y2 can be expressed using ***Taylor series expansion*** of the functions f1(x1,y1), f2(x1,y1):

$$\begin{aligned} f_1(x_2,y_2) &= f_1(x_1,y_1)+(x_2-x_1)\frac{\partial f_1}{\partial x}+(y_2-y_1)\frac{\partial f_1}{\partial y}+\ldots \\ f_2(x_2,y_2) &= f_2(x_1,y_1)+(x_2-x_1)\frac{\partial f_2}{\partial x}+(y_2-y_1)\frac{\partial f_2}{\partial y}+\ldots \end{aligned} \quad (1)$$

$$f_1(x_2, y_2) = f_1(x_1, y_1) + (x_2 - x_1)\frac{\partial f_1}{\partial x} + (y_2 - y_1)\frac{\partial f_1}{\partial y} + \ldots = 0$$

$$f_2(x_2, y_2) = f_2(x_1, y_1) + (x_2 - x_1)\frac{\partial f_2}{\partial x} + (y_2 - y_1)\frac{\partial f_2}{\partial y} + \ldots = 0 \tag{2}$$

$$-f_1(x_1, y_1) = (x_2 - x_1)\frac{\partial f_1}{\partial x} + (y_2 - y_1)\frac{\partial f_1}{\partial y}$$

$$-f_2(x_1, y_1) = (x_2 - x_1)\frac{\partial f_2}{\partial x} + (y_2 - y_1)\frac{\partial f_2}{\partial y} \tag{3}$$

$$-f_1(x_1, y_1) = \Delta x\frac{\partial f_1}{\partial x} + \Delta y\frac{\partial f_1}{\partial y}$$

$$-f_2(x_1, y_1) = \Delta x\frac{\partial f_2}{\partial x} + \Delta y\frac{\partial f_2}{\partial y} \tag{4}$$

$$-f_1(x_1, y_1) = \Delta x \frac{\partial f_1}{\partial x} + \Delta y \frac{\partial f_1}{\partial y} \tag{5}$$

$$-f_2(x_1, y_1) = \Delta x \frac{\partial f_2}{\partial x} + \Delta y \frac{\partial f_2}{\partial y}$$

$$\Delta x = \frac{-f_1(x_1, y_1)\frac{\partial f_2}{\partial y} + f_2(x_1, y_1)\frac{\partial f_1}{\partial y}}{\boxed{\frac{\partial f_1}{\partial x}\frac{\partial f_2}{\partial y} - \frac{\partial f_1}{\partial y}\frac{\partial f_2}{\partial x}}} \tag{6}$$

Jacobian matrix:

$$J_F(x, y) = \begin{bmatrix} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial y} \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y} \end{bmatrix}$$

$$\Delta y = \frac{-f_2(x_1, y_1)\frac{\partial f_1}{\partial x} + f_1(x_1, y_1)\frac{\partial f_2}{\partial x}}{\frac{\partial f_1}{\partial x}\frac{\partial f_2}{\partial y} - \frac{\partial f_1}{\partial y}\frac{\partial f_2}{\partial x}}$$

$$x_2 = x_1 + \Delta x$$

$$y_2 = y_1 + \Delta y \tag{7}$$

# Algorithm for solving a set of equations

1. Estimate an initial solutions, x1, y1
2. Calculate Jacobian and f1x, f1y, f2x, f2y
3. Solve for delta x, delta y
4. Estimate the new solutions, x2, y2, x3, y3, …
5. Compute the tolerance, then repeat or break