# *MATLAB for Finance*

# 16
## Introduction to Numerical Methods

# Numerical Methods

- *Methods to approximate solution of mathematical problems through the performance of a finite number of elementary operations on numbers*

- The study of algorithms that use *numerical approximation* (instead of *symbolic manipulation*) is named Numerical Analysis

# Financial Example

- Financial problem: Internal Rate of Return ($r^*$)
- Given coupons $c$ and redemption $R$ the present value is:

$$V_0(v) = \sum_{k=1}^{n} c \cdot v^k + R \cdot v^n \qquad (1)$$

- where $v = d(r)$ for any suitable discount factor, like, e.g., $d(r) = (1+r)^{-1}$ or $d(r) = e^{-r}$

- The IRR $r^*$ is the rate that equals the present value $V_0$ to the spot price $P_0$: $V_0(d(r^*)) = P_0$

# Financial Example

- Rewriting (1) as follows:

$$V_0(v) = c \cdot v \cdot \sum_{k=0}^{n-1} v^k + R \cdot v^n$$

- we can recognize the summation to be the sum of a series:

$$\sum_{k=0}^{n-1} v^k = 1 + v + v^2 + \ldots + v^{n-1} = \frac{1 - v^n}{1 - v}$$

- that we can substitute in the present value

$$V_0(v) = c \cdot v \frac{1 - v^n}{1 - v} + R \cdot v^n$$

- The first derivative is

$$V_0'(v) = \frac{\partial V_0(v)}{\partial v} = c \cdot \left[ \frac{1 - v^n}{1 - v} + v \frac{-n \cdot v^{n-1}(1 - v) + (1 - v^n)}{(1 - v)^2} \right] + n \cdot R \cdot v^{n-1}$$

# Exercise

- Create a MATLAB function that compute the **present value**. Identify all the inputs needed to describe the coupon bond

- Create a MATLAB function that compute the **first derivative** of the present value given a certain bond

# Internal Rate of Return

- Note that often the IRR cannot be found analytically (polynomials with high grade), then numerical methods must be used

- Most common (basic) numerical methods:
  - Bisection method
  - Newton-Raphson method
  - Secant method

# *Bisection Method*

# Bisection Method

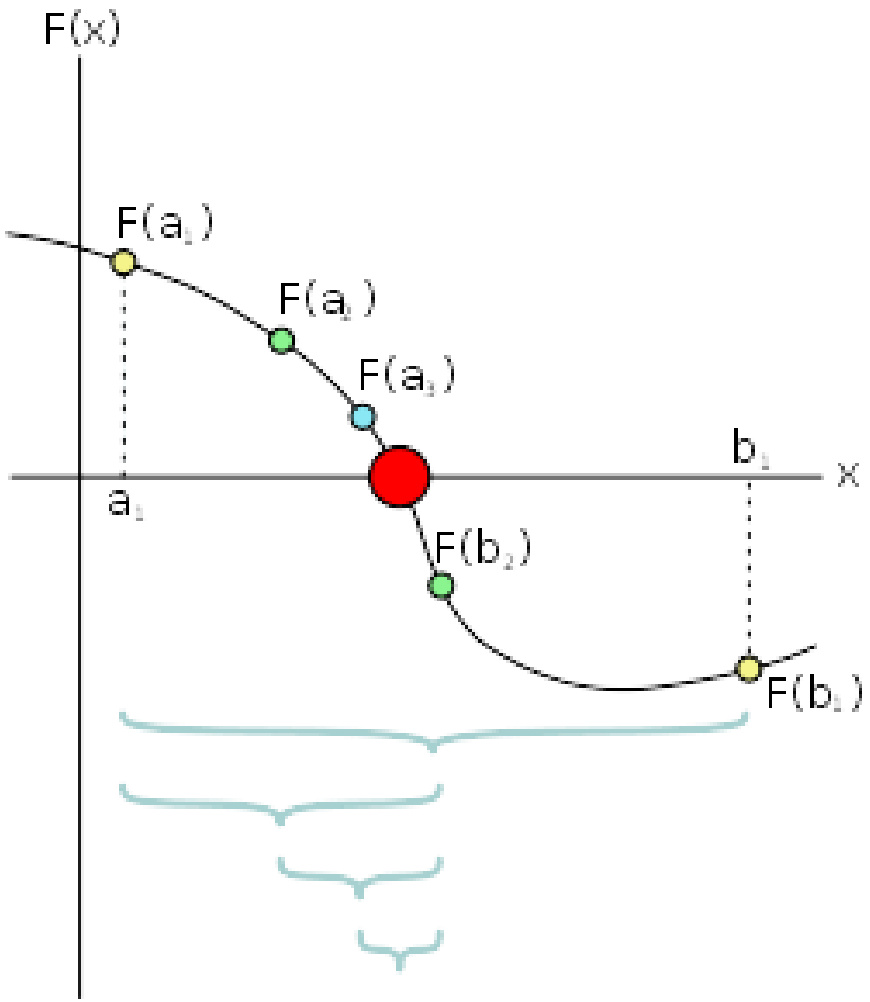
- Root-finding method which repeatedly bisects an interval and then selects a subinterval in which a root must lie for further processing
- It is a very simple and robust method, but it is also relatively slow
- Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods

# Bisection Method

- Finding $x$ such that $f(x)=0$, for any $f(\cdot)$ with at least a solution in interval $(a,b)$ and $\text{sign}(f(a)) \neq \text{sign}(f(b))$
- Iteratively compute the midpoint $c = (a+b)/2$ and compare $f(c)$ with $f(a)$ and $f(b)$ so that
  - if $\text{sign}(f(c)) \neq \text{sign}(f(a))$ then $c$ is the new $b$
  - if $\text{sign}(f(c)) \neq \text{sign}(f(b))$ then $c$ is the new $a$
- The iteration stops when $f(c) = 0$, i.e., $c$ is a solution

# Bisection Method

- Finding $x$ such that $f(x)=0$, for any $f(\cdot)$ with at least a solution in interval $(a,b)$ and $\text{sign}(f(a)) \neq \text{sign}(f(b))$
- Iteratively compute the midpoint $c = (a+b)/2$ and compare $f(c)$ with $f(a)$ and $f(b)$ so that
  - if $\text{sign}(f(c)) \neq \text{sign}(f(a))$ then $c$ is the new $b$
  - if $\text{sign}(f(c)) \neq \text{sign}(f(b))$ then $c$ is the new $a$
- The iteration stops when $f(c) = 0$, i.e., $c$ is a solution



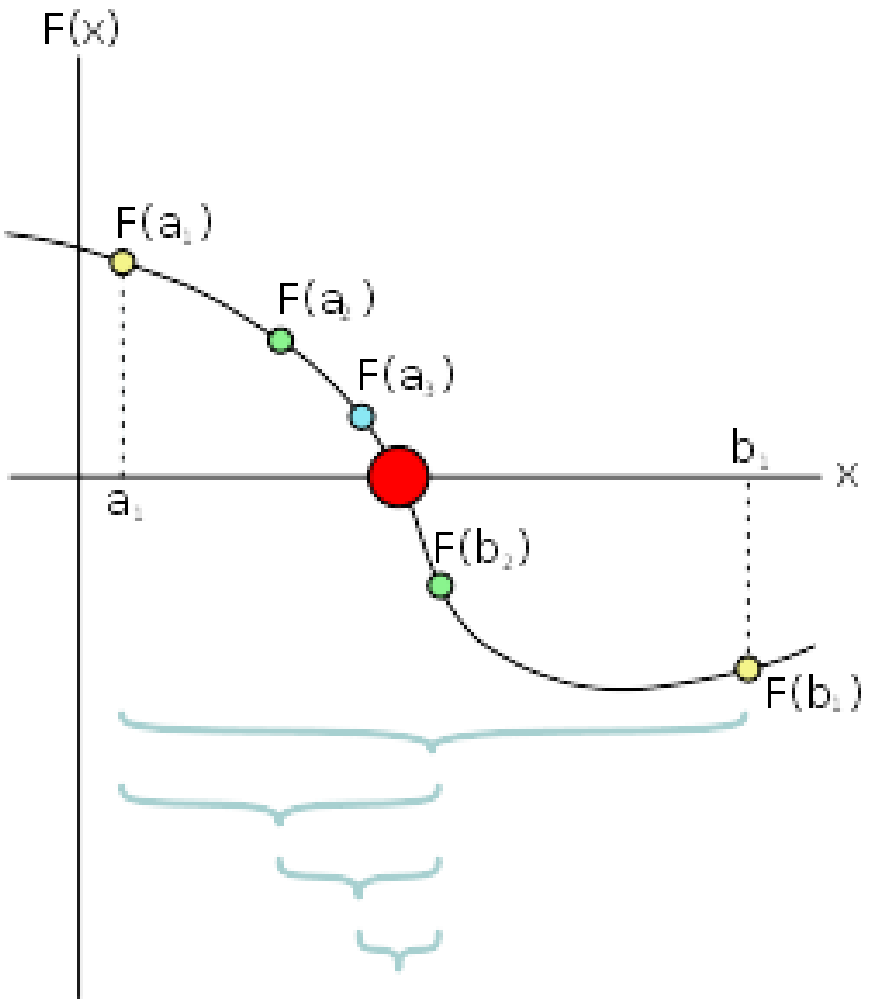
# Bisection Method

- Finding $x$ such that $f(x)=0$, for any $f(\cdot)$ with at least a solution in interval $(a,b)$ and $\text{sign}(f(a)) \neq \text{sign}(f(b))$
- Iteratively compute the midpoint $c = (a+b)/2$ and compare $f(c)$ with $f(a)$ and $f(b)$ so that
  - if $\text{sign}(f(c)) \neq \text{sign}(f(a))$ then $c$ is the new $b$
  - if $\text{sign}(f(c)) \neq \text{sign}(f(b))$ then $c$ is the new $a$
- The iteration stops when $f(c) = 0$, i.e., $c$ is a solution

```
while (f(c)~=0
  and |a-b|> ε)
  c ← (a+b) / 2
  if sign(f(a)) =
   sign(f(c)) then
      a ← c
  else
      b ← c
wend
```

# Bisection Method

```
while (f(c)~=0
  and |a-b|>ε)
  c ← (a+b) / 2
  if sign(f(a)) =
   sign(f(c)) then
      a ← c
  else
      b ← c
wend
```



# Exercise

- Implement in MATLAB a function that computes the internal rate of return (IRR) using the bisection method

# *Newton-Raphson Method*

# Newton-Raphson Method

- (Isaac Newton and Joseph Raphson)
- Is a method for finding successively better approximations to the roots (or zeroes) of a real-valued function.
- Given a function $f(x)$ and its derivative $f'(x)$, we begin with a first guess $x_0$ for a root of the function. Provided the function is reasonably well-behaved, a better approximation $x_1$ is

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

# Newton-Raphson Method

- Geometrically $x_1$ is the intersection with the $x$-axis of a line tangent to $f$ at $f(x_0)$

- The process $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$ is repeated

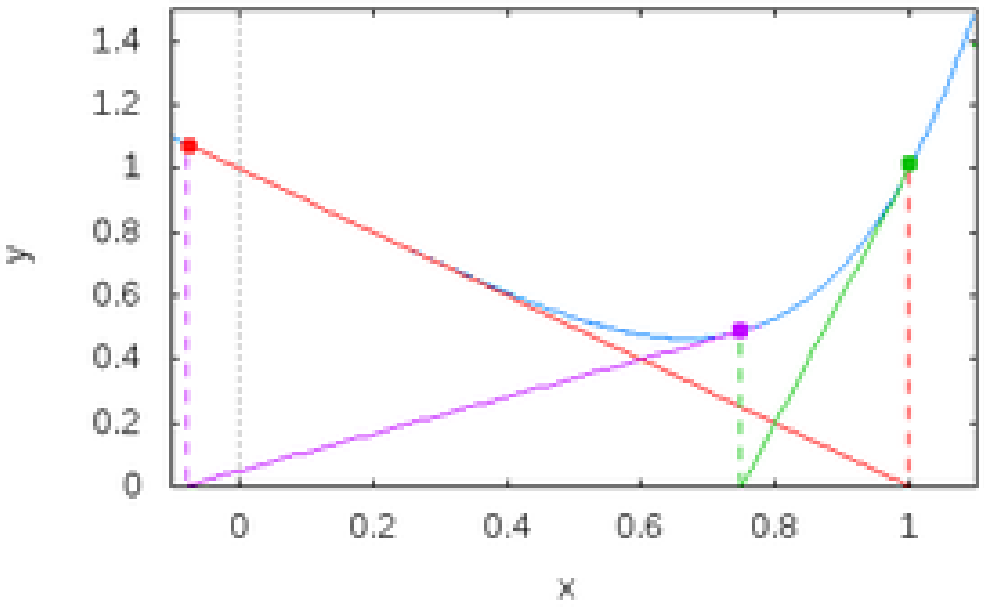
  until a sufficiently accurate value is reached:

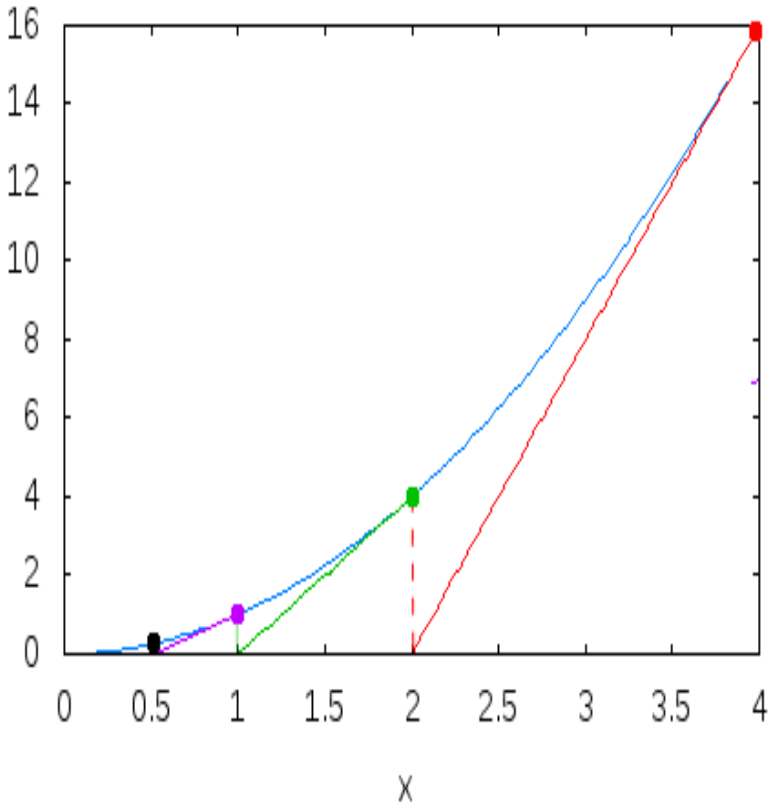
$$\left| f(x_n) \right| < \varepsilon$$

- Note that $f$ must be a differentiable function

# Newton-Raphson Method

```
while f(x) > ε
    x ← x – (f(x) / f' (x))
wend
```



# Exercise

- Implement in MATLAB a function that computes the internal rate of return (IRR) using the Newton-Raphson method. The algorithm is:

**while** $|V_0(v) - P_0| > \varepsilon$

$v \leftarrow v - (V_0(v) - P_0) / V_0'(v)$

**wend**

$P_0$ is constant

# *Secant Method*

# Secant Method

- Root-finding algorithm that uses a succession of roots of secant lines to better approximate a root of a function $f$
- The secant method can be thought of as a finite difference approximation of Newton's method
- This method is defined by the recurrence relation

$$x_n = x_{n-1} - f(x_{n-1})\frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})}$$

- The secant method requires two initial values, $x_0$ and $x_1$ , which should ideally be chosen to lie close to the root.