Root Finding

Prepared by Jakir Hasan
CSE'18, SUST
05/12/21

Content

Bracketing Method

If f(x) is real and continuous in the interval of xl to xu and f(xl) and f(xu) have opposite signs, that is,

$$f(xl) * f(xu) < 0$$

Then there is at least one real root between xl and xu.

Bisection Method

Pseudocode

```
Step 1:
Choose lower xl and upper xu guesses for the root such that the function
changes sign over the interval. This can be checked by ensuring that
        f(xl) * f(xu) < 0

Step 2:
An estimate of the root is determined by xr = (xl + xu)/2

Step 3:
Make the following evaluations to determine in which subinterval the root
lies:
```

```
a. If f(xl)*f(xr) < 0, the root lies in the lower subinterval. Therefore
set xu = xr and return to step 2

b. If f(xl)*f(xr) > 0, the root lies in the upper subinterval. Therefore
set xl = xr and return to step 2

c. If f(xl)*f(xr) == 0, the root equals xr, terminate the computation.
```

Implementation

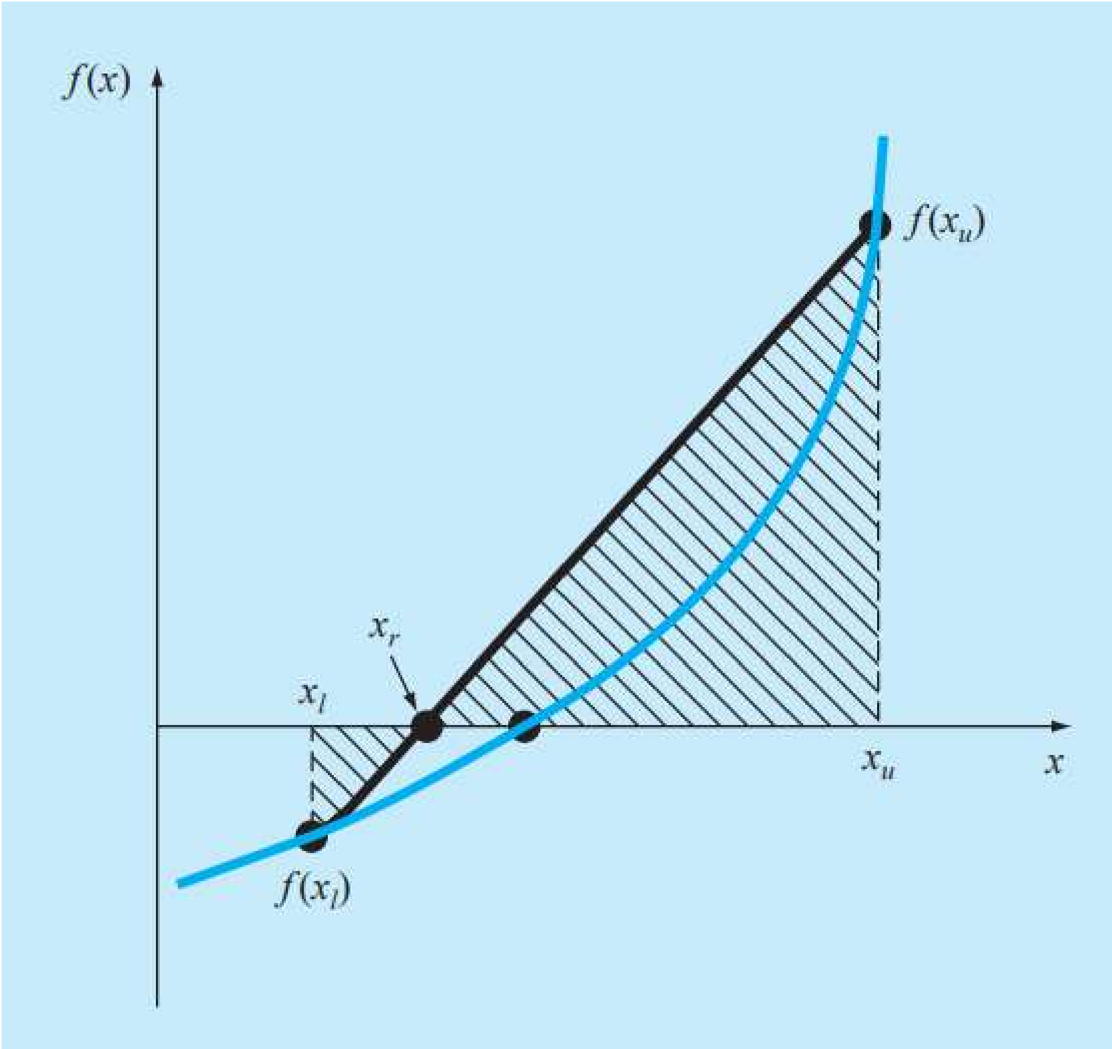
```
def function(x):
    """
    Assumes x is a float.
    Return a float evaluating the expression below.

    """
    return pow(x, 3) - 0.165 * pow(x, 2) + 3.993 * pow(10, -4)


def bisection_method(xl, xu, tolerance, max_iteration):
    """
    Parameters
    xl (float): Lower limit of the interval
    xu (float): Upper limit of the interval
    tolerance (float): Tolerance value.
    max_iteration (int): Maximum number of iterations

    Returns root (float) of the function

    """

    # xr (float): Midpoint of lower and upper limit
    xr = (xl + xu)/2.0

    # total_iteration (int): Total number of iterations
    total_iteration = 0

    while True:
```

```
        # test (float)
        test = function(xl) * function(xr)

        if (test < 0):
            xu = xr
        elif (test > 0):
            xl = xr
        else:
            break

        # xr_previous (float): Previous value of root
        xr_previous = xr
        xr = (xl + xu)/2

        # approximate_relative_error (float): Approximate relative error
        approximate_relative_error = (abs(xr - xr_previous)/xr) * 100.0
        total_iteration += 1

        if (approximate_relative_error < tolerance or total_iteration >
max_iteration):
            break

    return xr


# tolerance -> 0.0005%
root = bisection_method(0.0, 0.11, 0.0005, 20)

print("Approximate value of root is", root)
```

@Credit: Numerical Methods for Engineers by Steven C. Chapra, Batch Drive of 16

False Position Method

Process



Formula For Guessing Root

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

Implementation

```
def f(x):
```

```python
    """
    Assumes x is float
    Return a float evaluating the expression
    """
    return pow(x, 3) - 0.165 * pow(x, 2) + 3.993 * pow(10, -4)


def false_position_method(xl, xu, tolerance, max_iteration):
    """



    Parameters
    ----------
    xl (float): Lower limit
    xu (float): Upper limit
    tolerance (float): Tolerance value
    max_iteration (int): Maximum number of iteration

    Returns root (float) of the given function

    """

    # xr (float): Estimated value of root
    xr = xu - f(xu) * ((xl - xu)/(f(xl) - f(xu)))
    total_iteration = 0

    while True:

        test = f(xl) * f(xr)

        if test < 0:
            xu = xr
        elif test > 0:
            xl = xr
        else:
            break

        xr_previous = xr
        xr = xu - f(xu) * ((xl - xu)/(f(xl) - f(xu)))
        approximate_relative_error = (abs(xr - xr_previous)/xr) * 100.0

        total_iteration += 1
```

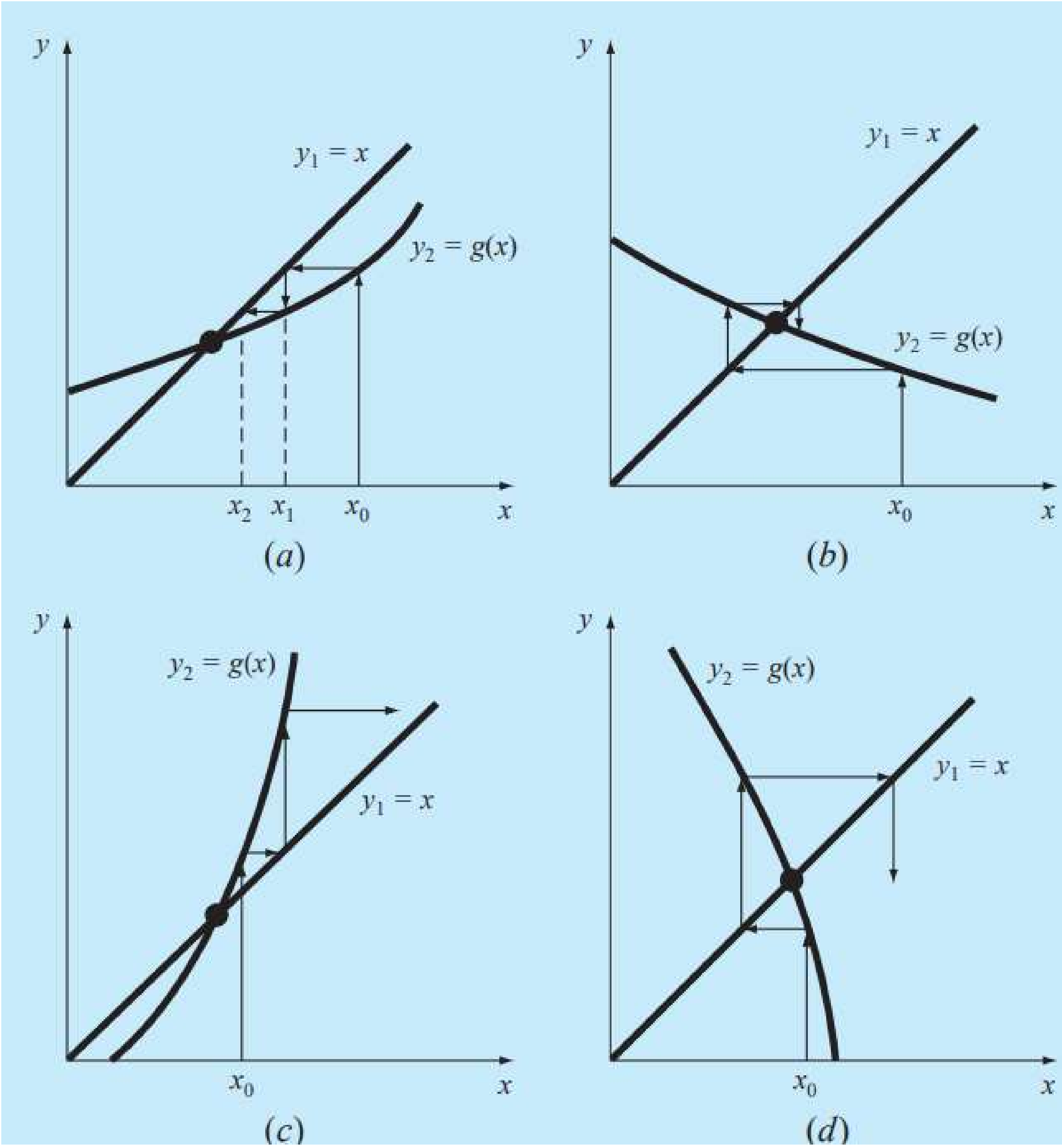
```
        if approximate_relative_error < tolerance or total_iteration >
max_iteration:
            break

    return xr


# tolerance -> 0.000005%
root = false_position_method(0.0, 0.11, 0.000005, 20)
print("Approximate value of root is", root)
```

@Credit: Numerical Methods for Engineers by Steven C. Chapra, Batch Drive of 16

Open method

Simple Fixed Point Iteration

Process



Formula For guessing root

Xi+1 = g(Xi)

Implementation

```
import math
```

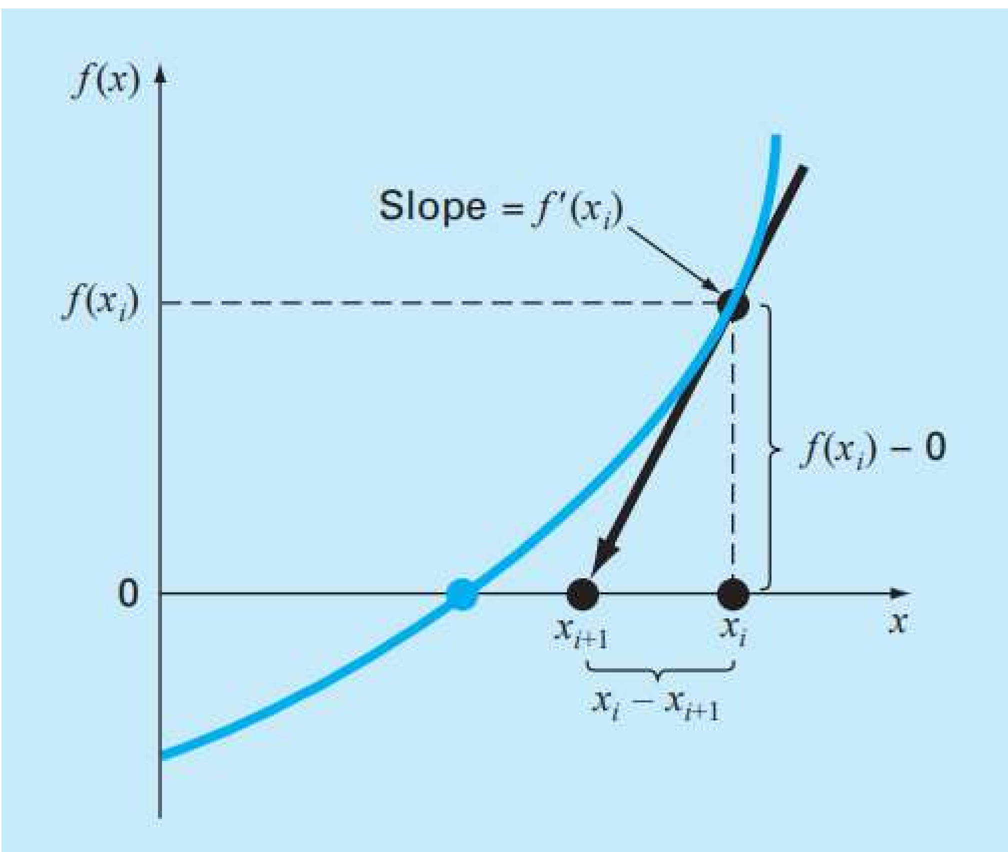
```python
def f(x):
    """
    Assumes x is a float.
    Returns a float evaluating the expression below

    """
    return pow(x, 3) + pow(x, 2) - 1


def g(x):
    """
    Assumes x is a float.
    Returns a float evaluating the expression below.(this expression is
achieved
    by converting f(x) = 0 to x = g(x))

    """
    return 1 / math.sqrt(x+1)


def simple_fixed_point_iteration(x0, tolerance, max_iteration):
    """
    Parameters
    ----------
    x0 (float): Initial guess of root
    tolerance (float): Error below this value is acceptable
    max_iteration (int): Maximum number of iteration

    Return the root (float) of the function
    """

    # x1 (float)
    x1 = x0
    # total_iteration (int): Total number of iteration
    total_iteration = 0

    while True:

        x1_previous = x1
        x1 = g(x1)

        approximate_relative_error = (abs(x1 - x1_previous)/x1) * 100.0
        total_iteration += 1
```

```
        if approximate_relative_error < tolerance or total_iteration > max_iteration:
            break

    return x1


root = simple_fixed_point_iteration(2, 0.00001, 10)
print("Approximate value of root is", root)
```

@Credit: Numerical Methods for Engineers by Steven C. Chapra, Batch Drive of 16

Newton Raphson Method

Process



Formula For guessing root

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

Implementation

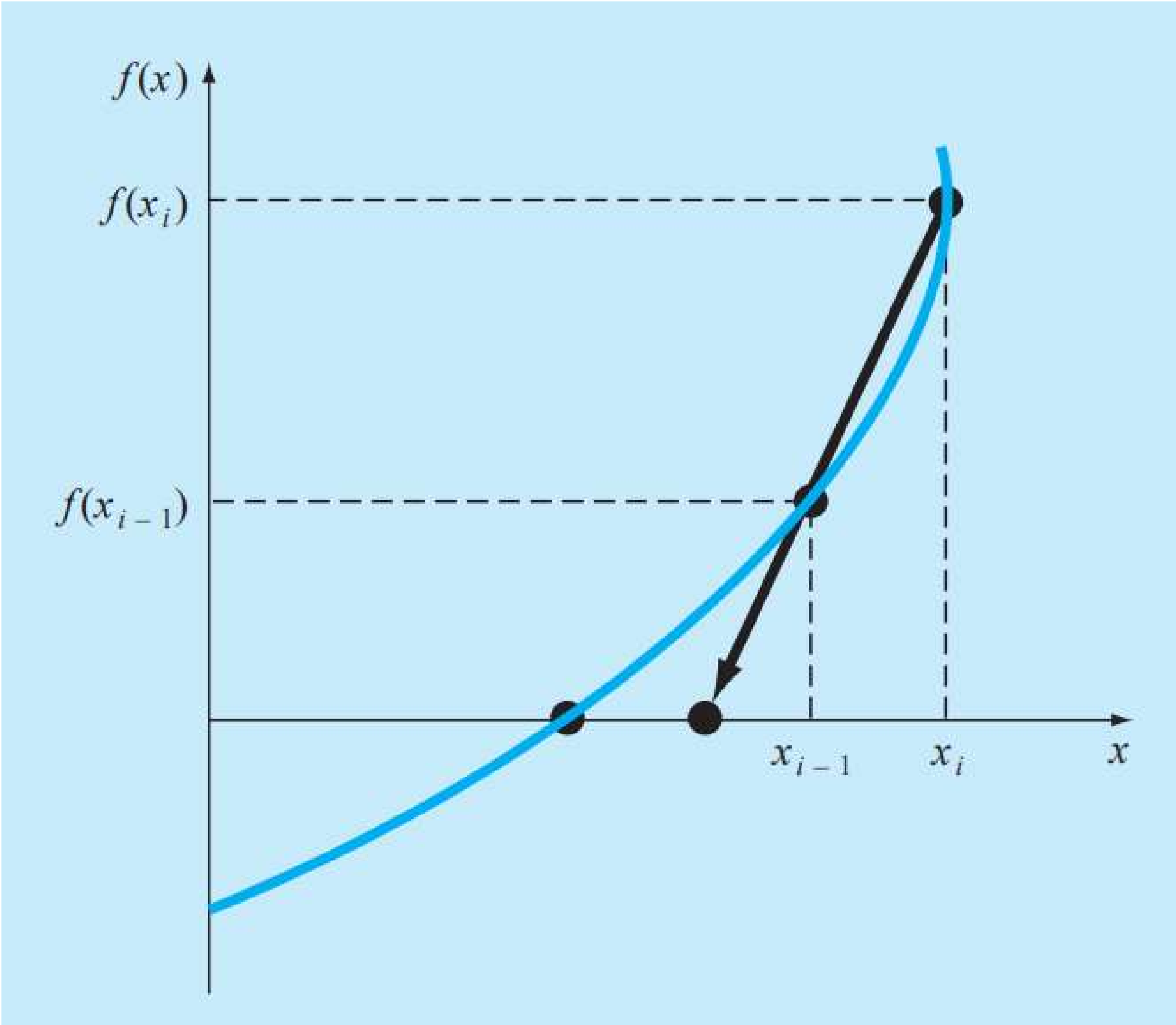
```
def f(x):
    """
    Returns a float evaluating the expression

    """
    return pow(x, 3) - 0.165*pow(x, 2) + 3.993*pow(10, -4)


def derivative(x):
    """
    Returns the derivative of f(x)

    """
    return 3*pow(x, 2) - 0.33*x


def newton_raphson_method(x0, tolerance, max_iteration):
    """
    x0 (float): Initial guess
    tolerance (float): Error below this value is acceptable
    max_iteration (int): Maximum number of iteration

    Return the root of f(x)
    """

    x1 = x0
    total_iteration = 0

    while True:

        x1_previous = x1
        x1 = x1 - f(x1) / derivative(x1)

        approximate_relative_error = (abs(x1 - x1_previous)/x1) * 100.0
        total_iteration += 1
```

```
        if approximate_relative_error < tolerance or total_iteration >
max_iteration:
            break

    return x1


root = newton_raphson_method(0.05, 0.00005, 20)
print("Approximate value of root is", root)
```

@Credit: Numerical Methods for Engineers by Steven C. Chapra, Batch Drive of 16

Secant Method

Process



Formula For guessing root

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

Implementation

```
def f(x):
    """
```

```
    Parameters
    ----------
    Assumes x is a float

    Returns a float evaluating the expression

    """
    return pow(x, 3) - 0.165*pow(x, 2) + 3.993*pow(10, -4)


def secant_method(x0, x1, tolerance, max_iteration):
    """
    Parameters
    ----------
    x0 (float): Initial guess
    x1 (float): Initial guess
    tolerance (float): Error below this value is acceptable
    max_iteration (int): Maximum number of iteration

    Return the root of the function
    """

    # x2 (float): Approximate root of the function
    x2 = x1 - f(x1) * ((x1 - x0) / (f(x1) - f(x0)))

    x0 = x1
    x1 = x2
    total_iteration = 0

    while True:

        x2_previous = x2
        x2 = x1 - f(x1) * ((x1 - x0) / (f(x1) - f(x0)))

        x0 = x1
        x1 = x2

        approximate_relative_error = (abs(x2 - x2_previous) / x2) * 100.0
        total_iteration += 1

        if approximate_relative_error < tolerance or total_iteration >
max_iteration:
            break
```

```python
    return x2

root = secant_method(0.02, 0.05, 0.000005, 20)

print("Approximate value of root is", root)
```

@Credit: Numerical Methods for Engineers by Steven C. Chapra, Batch Drive of 16