Non-linear equations

One dimensional case

Dr.ir. Ivo Roghair, Prof.dr.ir. Martin van Sint Annaland

Chemical Process Intensification group
Eindhoven University of Technology

Numerical Methods (6BER03), 2024-2025

S

Today's outline

- Introduction
    - General
- Direct Iteration Method
    - Passing functions
- Bracketing
- Bisection method
- Secant/False Position
- Brent's method
- Python solvers
- Newton-Raphson method
- Multi-dimensional Newton-Raphson

Today's outline

- Introduction
  - General
- Direct Iteration Method
  - Passing functions
- Bracketing
- Bisection method
- Secant/False Position
- Brent's method
- Python solvers
- Newton-Raphson method
- Multi-dimensional Newton-Raphson

Content

Root finding

How to solve $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ for arbitrary functions $\mathbf{f}$ (i.e., $\mathbf{f}(\mathbf{x})$ move all terms to the left)

- One-dimensional case: 'Bracket' or 'trap' a root between bracketing values, then hunt it down like a rabbit.
- Multi-dimensional case:
  - *N* equations in *N* unknowns: You can only hope to find a solution.
  - It may have no (real) solution, or more than one solution!
  - Much more difficult!! "You never know whether a root is near, unless you have found it"

Outline

One-dimensional case:

- Direct iteration method
- Bisection method
- Secant and false position method
- Brent's method
- Newton-Raphson method

Multi-dimensional case:

- Newton-Raphson method
- Broyden's method

Outline

One-dimensional case:

- Direct iteration method
- Bisection method
- Secant and false position method
- Brent's method
- Newton-Raphson method

Multi-dimensional case:

- Newton-Raphson method
- Broyden's method

In this course we will:

- Introduction to underlying ideas and algorithms
- Exercises in how to program the methods in Excel and Python.

Outline

One-dimensional case:

- Direct iteration method
- Bisection method
- Secant and false position method
- Brent's method
- Newton-Raphson method

Multi-dimensional case:

- Newton-Raphson method
- Broyden's method

In this course we will:

- Introduction to underlying ideas and algorithms
- Exercises in how to program the methods in Excel and Python.

Warning

Do not use routines as black boxes without understanding them!!!

Today's outline

General Idea

Root finding proceeds by iteration:

- Start with a good initial guess (crucially important!!)
- Use an algorithm to improve the solution until some predetermined convergence criterion is satisfied

Pitfalls:

- Convergence to the wrong root...
- Fails to converge because there is no root
- Fails to converge because your initial estimate was not close enough...

General Idea

Root finding proceeds by iteration:

- Start with a good initial guess (crucially important!!)
- Use an algorithm to improve the solution until some predetermined convergence criterion is satisfied

Pitfalls:

- Convergence to the wrong root...
- Fails to converge because there is no root
- Fails to converge because your initial estimate was not close enough...

Tips:

- It never hurts to inspect your function graphically
- Pay attention to carefully select initial guesses

General Idea

Root finding proceeds by iteration:

- Start with a good initial guess (crucially important!!)
- Use an algorithm to improve the solution until some predetermined convergence criterion is satisfied

Pitfalls:

- Convergence to the wrong root...
- Fails to converge because there is no root
- Fails to converge because your initial estimate was not close enough...

Tips:

- It never hurts to inspect your function graphically
- Pay attention to carefully select initial guesses

Hamming's motto

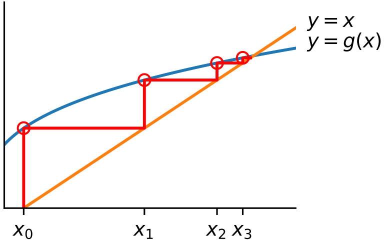
The purpose of computing is insight, not numbers!!

Direct Iteration Method/Successive Substitutions

Rewrite $f(x) = 0 \Rightarrow x = g(x)$

- Start with an initial guess: x_0
- Calculate new estimate with: $x_1 = g(x_0)$
- Continue iteration with: $x_2 = g(x_1)$
- Proceed until: $|x_{i+1} - x_i| < \varepsilon$

When the process converges, taking a smaller value for $x_{i+1} - x_i$ results in a more accurate solution, but more iterations need to be performed.



Direct Iteration Method - Exercise 1

Find the root of

$$f(x) = x^3 - 3x^2 - 3x - 4$$

Direct Iteration Method - Exercise 1

Find the root of

$$f(x) = x^3 - 3x^2 - 3x - 4$$

Attempt 1

Rewrite as $x = (3x^2 + 3x + 4)^{(1/3)}$

- Solve in Excel
- Solve in Python

Direct Iteration Method - Exercise 1

Find the root of

$$f(x) = x^3 - 3x^2 - 3x - 4$$

Attempt 1

Rewrite as $x = (3x^2 + 3x + 4)^{(1/3)}$

- Solve in Excel
- Solve in Python

Attempt 2

Rewrite as: $x = (x^3 - 3x^2 - 4)/3$

- Solve in Excel
- Solve in Python

Intermezzo: Functions Revisited

- In Python, you can define your own functions to reuse certain functionalities. We can define a mathematical function at the top of a file, or in a separate file with `.py` extension:

```python
def demo_f1(x):
    return x**2 + np.exp(x)
```

- The first line contains the function name, in this case `demo_f1`
- The return statement defines the output, `x` is defined as input
- It can use `x` as a scalar as well as a vector by using NumPy: e.g. `np.exp()`
  - If `x` is a vector, the output is also a vector.
- In case you define your function in a separate file, e.g. `nonlin_functions.py`, you can import the function into another file through:

```python
from nonlin_functions import demo_f1
```

Passing Functions in Python

- To solve $f(x) = x^2 - 4x + 2 = 0$ numerically, we can write a function that returns the value of $f(x)$:

```python
def MyFunc(x): # Note: case sensitive!!
    return x**2 - 4*x + 2
```

- The function can be assigned to a variable as an alias:

```python
f = MyFunc
a = 4
b = f(a)
```

```
2
```

- We can then call a solving routine (e.g., `fsolve` from SciPy):

```python
from scipy.optimize import fsolve
ans = fsolve(MyFunc, 5)
ans = fsolve(lambda x: x**2 - 4*x + 2, 5)
```

```
array([3.41421356])
array([3.41421356])
```

Passing Functions in Python

- We can also make our own function, that takes another function as an argument:

```
import matplotlib.pyplot as plt
import numpy as np

def draw_my_function(func):
    # Draws a function in the range [0, 10] using 20 data points.
    # 'func' is a function that can be any actual function.
    x = np.linspace(0, 10, 20)
    y = func(x)
    plt.plot(x, y, "-o")
    plt.show()
```

- Now we can call the function with another function, either a lambda function or a common function:

```
f = lambda x: x**2 - 4*x + 2
draw_my_function(f)
```

Direct Iteration Method - Exercise 1

Find the root of

$$f(x) = x^3 - 3x^2 - 3x - 4$$

Attempt 1

Rewrite as $x = (3x^2 + 3x + 4)^{(1/3)}$

- Solve in Excel
- Solve in Python

Attempt 2

Rewrite as: $x = (x^3 - 3x^2 - 4)/3$

- Solve in Excel
- Solve in Python

Direct Iteration Method - Exercise 1

Find the root of $f(x) = x^3 - 3x^2 - 3x - 4$ with the direct iteration method in Excel:

First attempt:

Iteration	Formula	Result
1	$(3x^2 + 3x + 4)^{(1/3)}$	2
2		3.115
3		3.489
$\vdots$		$\vdots$
10		3.990

Converges!

Second attempt:

Iteration	Formula	Result
1	$x = (x^3 - 3x^2 - 4)/3$	-1
2		-2.375
3		-11.439
$\vdots$		$\vdots$
10		#NUM!

Diverges!

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Direct Iteration Method - Exercise 1

Find the root of $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ with the direct iteration method in Python:
A simple script:

```python
x = 2.5
print(f"i: {0}, x: {x:.6e}")
for i in range(1, 21):
    x = (3*x**2 + 3*x + 4)**(1/3)
    print(f"i: {i}, x: {x:.6e}")
```

```
i: 0, x: 2.500000e+00
i: 1, x: 3.115840e+00
i: 2, x: 3.489024e+00
...
i: 19, x: 3.999970e+00
i: 20, x: 3.999983e+00
```

Lesson

Not very flexible/reusable $\longrightarrow$ use functions

Direct Iteration Method - Exercise 1

Find the root of the equation $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ using the direct iteration method in Python.

- First, define the functions.

```python
def MyFnc1(x):
    return (3*x**2 + 3*x + 4)**(1/3)

def MyFnc2(x):
    return (x**3 - 3*x**2 - 4) / 3
```

Direct Iteration Method - Exercise 1

Find the root of the equation $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ using the direct iteration method in Python.

- First, define the functions.

```python
def MyFnc1(x):
    return (3*x**2 + 3*x + 4)**(1/3)

def MyFnc2(x):
    return (x**3 - 3*x**2 - 4) / 3
```

- Then, create a function to carry out the Direct Iteration algorithm.

```python
def DirectIterationMethod(g, x, eps):
    itmax = 100
    it = 0
    y = g(x)
    print(f"i: {0}, x: {x:.6e}")
    while (abs(y - x) > eps) and (it < itmax):
        it += 1
        x = y
        y = g(x)
        print(f"i: {it}, x: {x:.6e}")
```

Direct Iteration Method - Exercise 1

Find the root of the equation $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ using the direct iteration method in Python.

- Finally, call the Direct Iteration function with the appropriate parameters.

```
DirectIterationMethod(MyFnc1, 2.5, 1e-3)
DirectIterationMethod(MyFnc2, 2.5, 1e-3)
```

Direct Iteration Method - Exercise 1

Find the root of the equation $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ using the direct iteration method in Python.

- Finally, call the Direct Iteration function with the appropriate parameters.

```
DirectIterationMethod(MyFnc1, 2.5, 1e-3)
DirectIterationMethod(MyFnc2, 2.5, 1e-3)
```

```
i: 0, x: 2.500000e+00
i: 1, x: 3.115840e+00
i: 2, x: 3.489024e+00
i: 3, x: 3.708113e+00
...
i: 9, x: 3.990573e+00
i: 10, x: 3.994696e+00
i: 11, x: 3.997016e+00
i: 12, x: 3.998321e+00
```

Direct Iteration Method - Exercise 1

Find the root of the equation $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ using the direct iteration method in Python.

- Finally, call the Direct Iteration function with the appropriate parameters.

```
DirectIterationMethod(MyFnc1, 2.5, 1e-3)
DirectIterationMethod(MyFnc2, 2.5, 1e-3)
```

```
i: 0, x: 2.500000e+00
i: 1, x: 3.115840e+00
i: 2, x: 3.489024e+00
i: 3, x: 3.708113e+00
...
i: 9, x: 3.990573e+00
i: 10, x: 3.994696e+00
i: 11, x: 3.997016e+00
i: 12, x: 3.998321e+00
```

```
i: 0, x: 2.500000e+00
i: 1, x: -2.375000e+00
i: 2, x: -1.143945e+01
i: 3, x: -6.311875e+02
i: 4, x: -8.421961e+07
i: 5, x: -1.991216e+23
i: 6, x: -2.631687e+69
Traceback (most recent
    call last):
```

Direct Iteration Method - Exercise 1

Find the root of the equation $f(x) = x^3 - 3x^2 - 3x - 4 = 0$ using the direct iteration method in Python.

- Finally, call the Direct Iteration function with the appropriate parameters.

```
DirectIterationMethod(MyFnc1, 2.5, 1e-3)
DirectIterationMethod(MyFnc2, 2.5, 1e-3)
```

```
i: 0, x: 2.500000e+00
i: 1, x: 3.115840e+00
i: 2, x: 3.489024e+00
i: 3, x: 3.708113e+00
...
i: 9, x: 3.990573e+00
i: 10, x: 3.994696e+00
i: 11, x: 3.997016e+00
i: 12, x: 3.998321e+00
```

```
i: 0, x: 2.500000e+00
i: 1, x: -2.375000e+00
i: 2, x: -1.143945e+01
i: 3, x: -6.311875e+02
i: 4, x: -8.421961e+07
i: 5, x: -1.991216e+23
i: 6, x: -2.631687e+69
Traceback (most recent
    call last):
```

Thinking

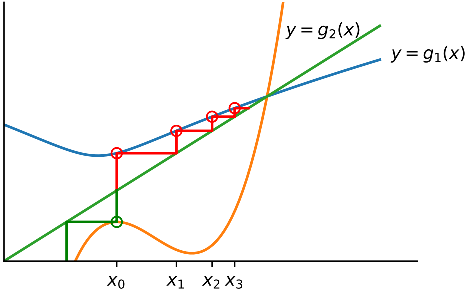
Discuss why it converges with `MyFnc1` and diverges with `MyFnc2`

Direct Iteration Method

- Exercise 1: Find the root of the equation

$$f(x) = x^3 - 3x^2 - 3x - 4 = 0$$

  using the direct iteration method.
- Observe that the method only works effectively when $g'(x_i) < 1$. Even then, it may not converge quickly.



Point

The iterations can be represented using the following relations:

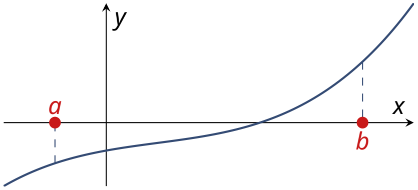
$$x_{i+1} = g(x_i) + g'(x_i)(x - x_i)$$
$$x_{i+2} = g(x_{i+1}) + g'(x_{i+1})(x_{i+1} - x_i)$$
$$|x_{i+2} - x_{i+1}| = |g'(x_i)||x_{i+1} - x_i|$$
$$\text{Convergence if } |g'(x_i)| \leq 1$$

Today's outline

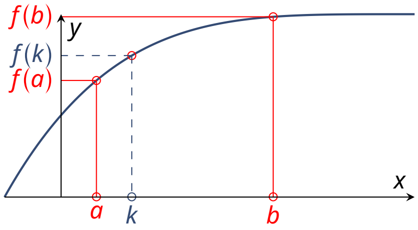
- Introduction
  - General
- Direct Iteration Method
  - Passing functions
- Bracketing
- Bisection method
- Secant/False Position
- Brent's method
- Python solvers
- Newton-Raphson method
- Multi-dimensional Newton-Raphson

Bracketing

Bracketing a root involves identifying an interval (a, b) within which the function changes its sign.



- If $f(a)$ and $f(b)$ have opposite signs, it indicates that at least one root lies in the interval (a, b), assuming the function is continuous in the interval.



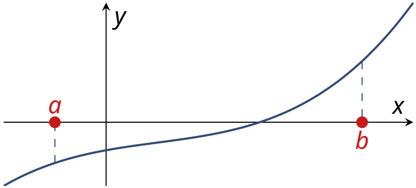
Intermediate value theorem

States that if $f(x)$ is continuous on $[a, b]$ and k is a constant lying between $f(a)$ and $f(b)$, then there exists a value $x \in [a, b]$ such that $f(x) = k$.

Bracketing

What's the point?

Bracketing a root = Understanding that the function changes its sign in a specified interval, which is termed as bracketing a root.

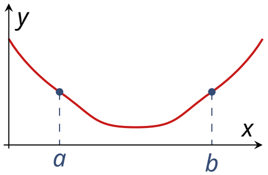


General best advice:

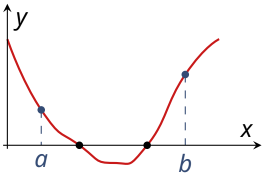
- Always bracket a root before attempting to converge on a solution.
- Never allow your iteration method to get outside the best bracketing bounds...

General Idea

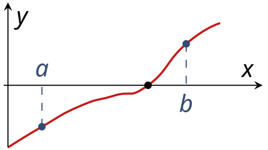
Potential issues to be cautious of while bracketing:



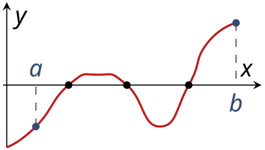
No answer (no root found)



Oops! Encountering two roots



Ideal scenario with one root found



Finding three roots (might work temporarily)

Bracketing - exercise 2

1. Write a Python function to bracket a function, starting with an initially guessed range x_1 and x_2 through the expansion of the interval.
2. Develop a program to ascertain the minimum number of roots existing within the x_1 and x_2 interval.
3. Note: These functions can be integrated to formulate a function that yields bracketing intervals for diverse roots.
4. Test the function for $f(x) = x^2 - 4x + 2$

Bracketing - exercise 2

- Initially, if feasible, draft a graph using the following Python commands:

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 5, 50)
y = x**2 - 4*x + 2
plt.figure()
plt.plot(x, y, x, np.zeros(len(x)))
plt.axis('tight')
plt.grid(True)
plt.show()
```

- This graphical representation instantly reveals the existence of two roots, evaluated as:

$$x_1 = 2 - \sqrt{2} \approx 0.59 \quad , \qquad x_2 = 2 + \sqrt{2} \approx 3.41$$

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Bracketing - exercise 2

```python
def find_root_by_bracketing(func, x1, x2, tol=1e-6, max_iter=1000):
  # Ensure the bracket is valid
  if func(x1) * func(x2) > 0:
     print('The bracket is invalid. The function must have opposite signs at the two endpoints.')
     return False

  # Loop until we find the root or exceed the maximum number of iterations
  for i in range(max_iter):
     # Find the midpoint
     x_mid = (x1 + x2) / 2

     # Check if we found the root
     if abs(func(x_mid)) < tol:
        print(f'Root found: {x_mid}')
        return True

     # Narrow down the bracket
     if func(x_mid) * func(x1) < 0:
        x2 = x_mid
     else:
        x1 = x_mid

  # If we reach here, we did not find the root within the maximum number of iterations
  print('Failed to find the root within the maximum number of iterations.')
  return False
```

Steps:

- Formulate a function to augment the interval (x_1, x_2) up to a maximum of 250 iterations or until a root is discovered.
- The function should:
  - Return true if a root is found, and false otherwise.
  - Showcase the results.

Bracketing

Exercise 2: Function to Bracket a Function

```python
def brak(func, x1, x2, n):
    nroot = 0
    dx = (x2 - x1) / n
    xb1 = []
    xb2 = []

    x = x1
    for i in range(n):
        x += dx
        if func(x) * func(x - dx) <= 0:
            nroot += 1
            xb1.append(x - dx)
            xb2.append(x)

    for i in range(nroot):
        print(f'Root {i+1} in bracketing interval
              [{xb1[i]}, {xb2[i]}]')
    else:
        if nroot == 0:
            print('No roots found!')
```

Steps:

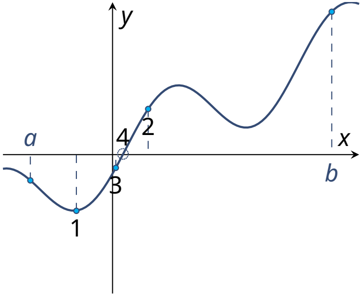
- The function subdivides the interval (x_1, x_2) into n parts to check for at least one root.
- It returns the left and right boundaries of the intervals where roots are found in arrays xb1 and xb2.

Today's outline

Bisection Method

Bisection Algorithm:

- Within a certain interval, the function crosses zero, indicated by a change in sign.
- Evaluate the function value at the midpoint of the interval and examine its sign.
- The midpoint then supersedes the limit sharing its sign.



Properties

- Pros: The method is infallible.
- Cons: Convergence is relatively slow.

Bisection Method

Exercise 3

- Write a function in Excel to find a root of a function using the bisection method.
- Assume that an initial bracketing interval (x_1, x_2) is provided.
- Specify the required tolerance.
- Output the required number of iterations.
- Implement the same in Python.

Exercise 3

Bisection Method in Excel:

it	x_1	x_2	f_1	f_2	xmid	fmid	Interval Size
0	-2	2	14	-2	0	2	4
1	0	2	2	-2	1	-1	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
25	0.585786	0.585786	1×10^{-7}	-6.8×10^{-8}	0.585786	1.58×10^{-8}	5.96×10^{-8}

Note: The table represents a sequence of iterations showing how the bisection method converges to a root with each step, demonstrating variable updates and interval size reduction.

Bisection Method

Exercise 3: Python Implementation

```python
def bisection(func, a, b, tol, maxIter):
    if func(a) * func(b) > 0:
        print('Error: f(a) and f(b) must have different signs.')
        return None

    iter = 0
    while (b - a) / 2 > tol:
        iter += 1
        if iter >= maxIter:
            print('Maximum iterations reached')
            return None

        c = (a + b) / 2
        print(f'Iteration {iter}: Current estimate: {c}')

        if func(c) == 0:
            return c

        if np.sign(func(c)) != np.sign(func(a)):
            b = c
        else:
            a = c

    return (a + b) / 2
```

- Criterion used for both the function value and the step size.
- While loop usually requires protection for a maximum number of iterations.
- Bisection is sure to converge.
- Root found in 25 iterations. Can we optimize it further?

Bisection Method

Required Number of Iterations:

- Interval bounds containing the root decrease by a factor of 2 after each iteration.

$$\varepsilon_{n+1} = \frac{1}{2}\varepsilon_n \quad \Rightarrow \quad \boxed{n = \log_2 \frac{\varepsilon_0}{tol}}$$

$\varepsilon_0 =$ initial bracketing interval,
$tol =$ desired tolerance.

- After 50 iterations, the interval is decreased by a factor of $2^{50} = 10^{15}$.
- Consider machine accuracy when setting tolerance.
- Order of convergence is 1:

$$\boxed{\varepsilon_{n+1} = K\varepsilon_n^m}$$

  - $m = 1$: linear convergence.
  - $m = 2$: quadratic convergence.

- Bisection method will:
  - Find one of the roots if there is more than one.
  - Find the singularity if there is no root but a singularity exists.

Today's outline

EINDHOVEN UNIVERSITY OF TECHNOLOGY

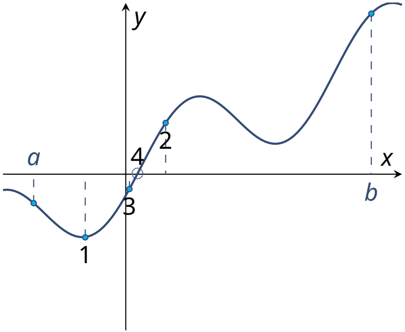
Secant and False Position Method

Secant/False Position (Regula Falsi) Method

- Provides faster convergence given sufficiently smooth behavior.
- Differs from the bisection method in the choice of the next point:
  - **Bisection**: selects the mid-point of the interval.
  - **Secant/False position**: chooses the point where the approximating line intersects the axis.
- Adopts a new estimate by discarding one of the boundary points:
  - **Secant**: retains the most recent of the previous estimates.
  - **False position**: maintains the prior estimate with the opposite sign to ensure the points continue to bracket the root.

Secant and False Position Method: Comparison

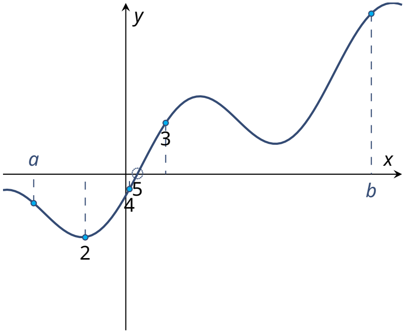
Secant Method



- Slightly faster convergence:

$$\lim_{n\to\infty} |\varepsilon_{n+1}| = K|\varepsilon_n|^{1.618}$$

False Position Method



- Guaranteed convergence

Secant and False Position Method

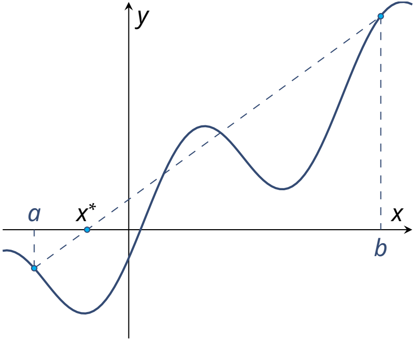
Exercise 4:

- Write a function in Excel and Python to find a root of a function using the Secant and False position methods.
- Assume that an initial bracketing interval (x_1, x_2) is provided.
- Specify the required tolerance.
- Output the required number of iterations.
- Compare the bisection, false position, and secant methods.

Secant and False Position Method

Exercise 4:

- Determination of the abscissa of the approximating line:



- Determine the approximating line using the expression:

$$f(x) \approx f(a) + \frac{f(b) - f(a)}{b - a}(x - a)$$

- Determine the abscissa where $f(x^*) = 0$:

$$x^* = a - \frac{f(a)(b-a)}{f(b) - f(a)}$$
$$= \frac{af(b) - bf(a)}{f(b) - f(a)}$$

Note: In the above equations, a and b are the initial guesses/boundaries where the root is suspected to be, and $f(x)$ is the function for which we are finding the root.

Secant and False Position Method

Exercise 4:

- Write a function in Excel and Python to find a root of a function using the Secant and the False position methods.
- Assume that an initial bracketing interval (x_1, x_2) is provided.
- Specify the required tolerance.
- Output the required number of iterations.
- Compare the bisection, false position, and secant methods.

Secant and False Position Method

Exercise 4: False Position Method in Excel

iteration	xa	xb	fa	fb	x absc	fabsc	interval
0	-1.5000	4.0000	-0.3895	2.1628	-0.6606	-0.8455	5.5000
1	-0.6606	4.0000	-0.8455	2.1628	0.6493	0.6896	4.6606
2	-0.6606	0.6493	-0.8455	0.6896	0.0609	-0.1972	1.3099
3	0.0609	0.6493	-0.1972	0.6896	0.1917	0.0070	0.5884
4	0.0609	0.1917	-0.1972	0.0070	0.1873	-0.0001	0.1308
5	0.1873	0.1917	-0.0001	0.0070	0.1874	0.0000	0.0045
6	0.1874	0.1917	0.0000	0.0070	0.1874	0.0000	0.0044
7	0.1874	0.1917	0.0000	0.0070	0.1874	0.0000	0.0044

Relevant expressions:

- `a=IF((a*fa)<0,a,xbsc)`
- `b=IF((b*fb)<0,b,xbsc)`
- `xbsc=a − fa*(xb−xa)/(fb−fa)`

Secant and False Position Method

Exercise 4:

- Write a function in Excel and Python to find a root of a function using the Secant and the False position methods.
- Assume that an initial bracketing interval (x_1, x_2) is provided.
- Also the required tolerance is specified.
- Also output the required number of iterations.
- Compare the bisection, false position, and secant methods.

Secant and False Position Method

Exercise 4: Secant method in excel

iteration	x	f
-1	2.0000	0.5895
0	-1.0000	-0.7591
1	0.6886	0.7368
2	-0.1431	-0.4819
3	0.1857	-0.0026
4	0.1875	0.0002
5	0.1874	0.0000

Relevant expressions:

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

$$f_n = f(x_n)$$

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Secant and False Position Method

Exercise 4: False position method in Python

```python
def false_position(f, x0, x1, tol, max_iter):
    if f(x0) * f(x1) > 0:
        raise ValueError('f(x0) and f(x1) must have different signs.')

    history = []

    for i in range(max_iter):
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        history.append(x2)

        if abs(f(x2)) < tol:
            break

        if f(x2) * f(x0) < 0:
            x1 = x2
        else:
            x0 = x2

    root = x2
    return root, history
```

Calling the function:

```python
secant_method(lambda x: x**2 - 4*x + 2, 0, 2, 1e-7, 100)
```

Secant and False Position Method

Exercise 4: Secant method in Python

```python
def secant_method(f, x0, x1, tol, max_iter):
    history = [x0, x1]

    for i in range(1, max_iter):
        x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
        history.append(x2)

        if abs(x2 - x1) < tol:
            break

        x0 = x1
        x1 = x2

    root = x1
    return root, history
```

Calling the function:

```python
false_position(lambda x: x**2 — 4*x + 2, 0, 2, 1e—7, 100)
```

Comparison of Methods

Exercise 4:

- $\mathrm{tol}_{\mathrm{eps}}, \mathrm{tol}_{\mathrm{func2}} = 1e-15, \quad \text{and} \quad (x_1, x_2) = (0, 2)$
- $f(x) = x^2 - 4x + 2 = 0$

Method	Nr. of iterations
Bisection	52
False position	22
Secant	9

```python
from scipy.optimize import root_scalar

root_scalar(lambda x: x**2 - 4*x + 2, method='brentq', bracket=[0, 2], xtol=1e-15)
```

Note the initial bracketing steps in root_scalar!

Today's outline

- Introduction
  - General
- Direct Iteration Method
  - Passing functions
- Bracketing
- Bisection method
- Secant/False Position
- **Brent's method**
- Python solvers
- Newton-Raphson method
- Multi-dimensional Newton-Raphson

Brent's Method

Features of Brent's method:

- Superlinear convergence with the sureness of bisection
- Keeps track of superlinear convergence, and if not achieved, alternates with bisection steps, ensuring at least linear convergence
- Implemented in MATLAB's `scipy.optimize.fzero` function:
  - Utilizes root-bracketing
  - Bisection/secant/inverse quadratic interpolation
- Inverse quadratic interpolation:
  - Uses three prior points to fit an inverse quadratic function ($x(y)$)
  - Involves contingency plans for roots falling outside the brackets

Brent's method

Formulas:

$$x = b + \frac{P}{Q}, \qquad R = \frac{f(b)}{f(c)}$$

$$P = S\left[T(R-T)(c-b) - (1-R)(b-a)\right], \qquad S = \frac{f(b)}{f(a)}$$

$$Q = (T-1)(R-1)(S-1), \qquad T = \frac{f(a)}{f(c)}$$

- b = current best estimate
- P/Q = a 'small' correction

Note: If P/Q does not land within the bounds or if bounds are not collapsing quickly enough, a bisection step is taken.

Brent's method script

```python
def brent_method(f, a, b, tol=1e-6, max_iter=100):
    if f(a) * f(b) >= 0:
        raise ValueError("f(a) and f(b) must have different signs.")
    # Initialize variables
    c = a
    fa = f(a)
    fb = f(b)
    fc = fa
    history = [a, b]
    d = e = b - a
    for _ in range(max_iter):
        if fa * fc > 0:
            c = a
            fc = fa
            d = e = b - a
        if abs(fc) < abs(fb):
            a, b, c = b, c, a
            fa, fb, fc = fb, fc, fa
        tol1 = 2 * 1.0e-16 * abs(b) + 0.5 * tol
        xm = 0.5 * (c - b)
        if abs(xm) <= tol1 or fb == 0:
            return b, history
        if abs(e) >= tol1 and abs(fa) > abs(fb):
            s = fb / fa
            if a == c:
                # Linear interpolation (Secant method)
                p = 2 * xm * s
                q = 1 - s
            else:
                # Inverse quadratic interpolation
                q = fa / fc
                r = fb / fc
                p = s * (2 * xm * q * (q - r) - (b - a) * (r - 1))
                q = (q - 1) * (r - 1) * (s - 1)
            if p > 0:
                q = -q
            p = abs(p)

            if 2 * p < min(3 * xm * q - abs(tol1 * q), abs(e * q)):
                e = d
                d = p / q
            else:
                d = xm
                e = d
        else:
            d = xm
            e = d
        a = b
        fa = fb
        if abs(d) > tol1:
            b += d
        else:
            b += tol1 if xm > 0 else -tol1

        fb = f(b)
        history.append(b)
    raise ValueError("Maximum number of iterations reached.")
```

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Using Excel for Solving Non-linear Equations: Goal-Seek and Solver

Setting up Goal-Seek and Solver in Excel:

- Available in Excel with some prerequisites installation.
- For Excel 2010:
  - Install via `Excel → File → Options → Add-Ins → Go` (at the bottom) → `Select solver add-in`.
  - Accessible through the 'data' menu ('Oplosser' in Dutch).

Procedure for solving:

- Select the goal-cell.
- Specify whether you want to minimize, maximize, or set a certain value.
- Define the variable cells for Excel to adjust to find the solution.
- Set the boundary conditions (if any).
- Click 'solve', possibly after setting advanced options.

Excel: Goal-Seek Example

Using Goal-Seek to find a solution:

- The Goal-Seek function can set the goal-cell to a desired value by adjusting another cell.
- Steps:

1. Open Excel and input the following data:

A	x	B
1	x	3
2	f(x)	f(x) = -3*B1^2 - 5*B1 + 2
3		

2. Navigate to `Data → What-if Analysis → Goal Seek` and input:
   - Set cell: B2
   - To value: 0
   - By changing cell: B1
3. Press OK to find a solution of approximately 0.3333.

Excel: Solver Example

Using Solver to Find Solutions with Boundary Conditions:

- Solver can adjust values in one or more cells to reach a desired goal-cell value, respecting specified boundary conditions.
- Example sheet setup:

	A	B	C
1		x	f(x)
2	x1	3	=2*B2*B3-B3+2
3	x2	4	=2*B3-4*B2-4

- Procedure:
  1. Navigate to `Data → Solver`.
  2. Set the goal function to C2 with a target value of 0.
  3. Add a boundary condition: C3 = 0.
  4. Specify the cells to change as `$B$2:$B$3`.
  5. Click "Solve" to find B2 = 0 and B3 = 2 as solutions.