13 Roots

Let f be a function. The number x for which $f(x) = 0$ is called a **zero of a function** or a **root of a function**. In other words, we need to solve the equation of $f(x) = 0$ – determine x value which satisfy this equation, therefore solving equations, whether linear or nonlinear is often called **root-finding**.

Finding roots of the equation or system of the equations is a common problem in engineering practice. The zeroes of the function can be real or complex numbers, but the complex roots are seldom computed, because they usually have no physical significance. It is not always possible to find the zero of a function in an analytical way. A function may have an arbitrary number of roots, including no roots, e.g. $f(x) = \sin x$ has an infinite number of roots.

All methods of finding roots are iterative and require a starting point which is an estimate of the root. The estimate is crucial – a bad starting value may fail to converge or converge to a different root than the sought. There is no universal method of finding a root estimate. Sometimes knowledge of the problem is enough to guess the location of the zero of a function. A numerical method for root finding typically requires a function to be evaluated at successive points until the algorithm has narrowed the range of the solution solution to the desired accuracy (in this chapter also referred as tolerance).

Plotting a function is usually a good visual aid, as it gives some clues about the number of solutions and their approximate locations. However, locating the root on the plot cannot be programmed. It is recommended to **bracket** the root, i.e. determine its lower and upper bounds, before passing the problem to a root-finding algorithm. The root is bracketed in the interval $(a,\ b)$ if $f(a)$ and $f(b)$ have opposite signs. If the function is continuous, at least one root must lie in the interval (according to the Intermediate Value Theorem). If the function is discontinuous, then instead of a root, a discontinuity may be bracketed.

13.1 Incremental search method

As it was stated before, plotting a function, even based on a few points, can be enough to provide good starting values for the root-searching algorithm. Another useful method for bracketing roots is the incremental search method. The idea of this algorithm is simple and uses the Intermediate Value Theorem – if $f(x_1)$ and $f(x_2)$ have different signs, there has to be at least one root in the interval $x_1,\ x_2)$. If the interval is small enough, it likely contains a single zero of a function. The roots of $f(x)$ can be detected by evaluating the function at intervals Δx and looking for a change in sign.

There are, however, potential drawbacks of the incremental search:

- a double root will not be detected,
- it is possible to miss two closely spaced roots (if Δx is larger than the root spacing),
- singularities in discontinuous functions can be mistaken for roots.

13.2 Convergence criteria

To determine if the number of iterations already finished is enough for the solution to be accurate, it is necessary to define a **convergence criterium** – a condition which fulfilled would define the stopping point of the computations. Usually, one of the criteria below is used for root-finding:

- limited number of steps (iterations),
- $|f(x)| \leq \varepsilon$ – check if the function value is small enough (close to zero within tolerance of ε),
- $|x_i - x_{i+1}| \leq \varepsilon$ – check if the results of the subsequent iterations are close to each other,

where ε defines the desired tolerance (accuracy) of the result.

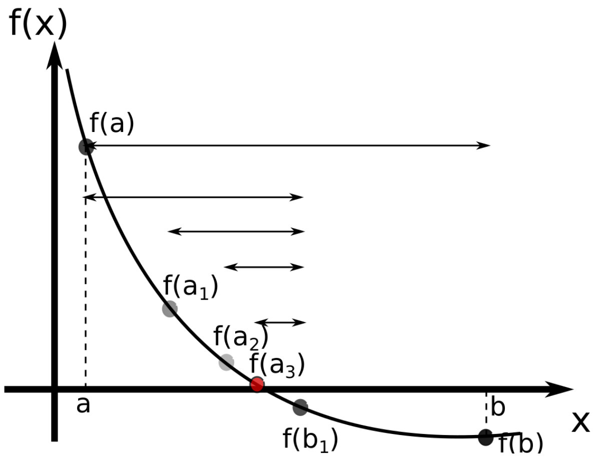

Figure 13.1. Bisection method

13.3 Bisection method

After the root of $f(x)$ has been bracketed in the interval $(x_1,\ x_2)$, there are several methods which can be used to find the zero. The method of **bisection** accomplishes this task by halving the interval until it becomes sufficiently small (the technique is also known as **interval halving method**). Although bisection is not the most efficient method used for computing roots, it is the most reliable – if the root has been bracketed correctly, it will always close on it.

The method of bisection uses the same principles as incremental search: if there is a root in the interval $(x_1,\ x_2)$, then $f(x_1)$ and $f(x_2)$ have opposite signs. Then, the interval needs to be halved to narrow down the search. This step is repeated until the solution with sufficient accuracy is found (Fig. 13.1). The algorithm can be described as follows:

1. Given the function $f(x)$ and the root bracketing interval $(x_1,\ x_2)$, halve the interval at point $x_3 = \frac{x_1+x_2}{2}$. Compute $f(x_3)$.
2. If $f(x_2)$ and $f(x_3)$ have opposite signs, the root must be in the interval $(x_3,\ x_2)$ and we replace x_1 with x_3. Otherwise, the zero lies in $(x_1,\ x_3)$ and x_2 is replaced with x_3. The new interval is half the size of the original interval.
3. Repeat steps 1–2 until the solution is found or one of the convergence criteria is fulfilled, e.g.:

$$|x_2 - x_1| \leq \varepsilon.$$

   where ε is the desired accuracy of the result.

Since in each iteration the starting interval is divided in half, the maximal number of repetitions n to reach desired accuracy ε can be easily estimated:

$$k = \frac{\ln(\Delta x/\varepsilon)}{\ln 2}. \tag{13.1}$$

13.4 Methods based on linear interpolation

There are two commonly used methods based on linear interpolation: **the secant** and false position (**regula falsi**) method. Those two are closely related, both require two starting estimates of the root (x_1, x_2), but they use them differently. The function $f(x)$ is assumed to be close to linear near the root, so the improved value of the root (x_3) can be approximated using linear interpolation between x_1 and

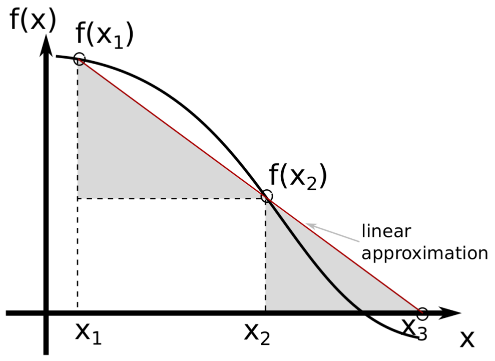

Figure 13.2. Linear interpolation between x_1 and x_2

x_2. Using symbols from Figure 13.2, the similar triangles yield the relationship:

$$\frac{f_2}{x_3 - x_2} = \frac{f_1 - f_2}{x_2 - x_1}. \tag{13.2}$$

Therefore, the improved estimate of the root can be defined as

$$x_3 = x_2 - f_2 \frac{x_2 - x_1}{f_2 - f_1}. \tag{13.3}$$

Equation 13.3 is a basis of computing the subsequent solutions using the secant rule.

The regula falsi method requires that x_1 and x_2 bracket the root. After the improved estimate is computed, either x_1 or x_2 is replaces by x_3 in a similar manner as in a bisection method: if $f(x_3)$ has the same sign as $f(x_1)$, then $x_1 = x_3$, otherwise $x_2 = x_3$. The procedure is repeated until the solution is found.

The secant method does not require bracketing of the root and discards the oldest prior estimate of the root ($x_1 = x_2$, $x_2 = x_3$ in the next iteration). The convergence of this method can be shown to be superlinear. The false position method's convergence is close to linear, but the method is more reliable, since it always brackets the root.

13.5 Ridder's method

While methods based on linear interpolation are a good choice, Ridder's algorithm will usually yield better results. Ridder's method is a variant of a false position algorithm. When a root is bracketed in an interval $(x_1,\ x_2)$, Ridder's method first evaluates the function at midpoint $x_3 = \frac{x_1+x_2}{2}$. Next a function $g(x)$ is introduced:

$$g(x) = f(x)e^{(x-x_1)Q}, \tag{13.4}$$

where Q is the constant determined by the condition that the points $(x_1,\ g(x_1))$, $(x_2,\ g(x_2))$, $(x_3,\ g(x_3))$ lie on the straight line. The improved estimate of the root is computed by the linear interpolation of $g(x)$ rather than $f(x)$.

The final formula for a root update based on linear interpolation of $g(x)$ is

$$x = x_3 + (x_3 - x_1)\frac{\text{sign}\left[f(x_1) - f(x_2)\right] f(x_3)}{\sqrt{f(x_3)^2 - f(x_1)f(x_2)}}. \tag{13.5}$$

x is guaranteed to lie in the interval (x_1, x_2). After each iteration x_0 and x_1 are redefined. x is used as one of the two bracketing values, the other one is taken to be x_2 if $f(x_2)f(x) < 0$ or whichever of two values, x_1 and x_3 has a function value with an opposite sign to $f(x)$.

The computation process is repeated until the difference between two successive values of x is negligible.

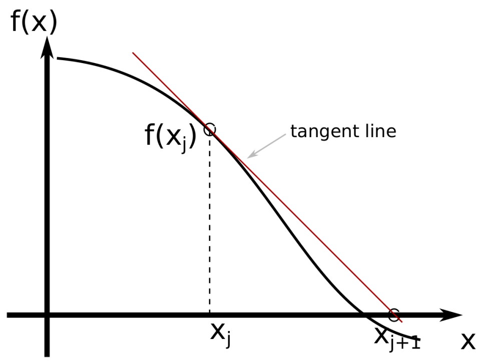

Figure 13.3. Graphical interpretation of the Newton-Raphson method

If the root was bracketed, x_4 is guaranteed to stay in the interval $(x_1,\ x_2)$, which makes the method very reliable. The downside is that each iteration requires two function evaluations. The order of the method is superlinear, can be shown to converge quadratically – faster than both the regula falsi and the secant method. The number of significant numbers in the result approximately doubles in each run.

13.6 Newton-Raphson method

The Newton-Raphson algorithm is probably the best known method for finding zeroes of a function: it is fast and simple. The drawback is that it uses a derivative of a function, as well as the function itself. Hence, the method is recommended for continuous functions, which have easily computable derivative.

The Newton-Raphson method can be derived from the Taylor series expansion of $f(x)$ about x:

$$f(x_{i+1}) = f(x_i) + f'(x_i)(x_{i+1} - x(i)) + O(x^2_{i+1-x_i}) \tag{13.6}$$

If x_{i+1} is a root of $f(x) = 0$ and assuming that x_i is close to x_{i+1}, we can solve for x_{i+1}:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{13.7}$$

The algorithm approximates $f(x)$ with a straight line that is tangent to the curve at point $(x_i,\ f(x_i))$, x_{i+1} is the intersection of the tangent line and the x-axis (Fig. 13.3).

Starting with an initial value x_0, the algorithm repeatedly applies eq. 13.7 until it reaches the defined convergence criterium, for example:

$$|x_{i+1} - xi| \leq \varepsilon \tag{13.8}$$

with ε being the error tolerance. The method converges **quadratically** (the error is the square error of the error in the previous step), the number of significant figures roughly doubles in every iteration. The algorithm converges fast near the root, however, its global convergence characteristics are poor – the tangent line is not always the best approximation of the function. There is also no guarantee, that the algorithm will converge to the closest root to the initial guess. It is also not possible to know in advance if the root found using the algorithm is larger or smaller than the given starting point.

13.7 Check your computations in Python

The `SciPy.optimize`[1] module provides some functions for numerical equation-solving, including the methods discussed in the previous sections: bisection, Ridder's method and Newton-Raphson algorithm.

Each of the methods presented below have some additional configarable parameters like the tolerance for the spacing between the subsequent values of x (`xtol`) or values of $f(x)$ `rtol`, maximal number of iterations (`maxiter`).

[1]`https://docs.scipy.org/doc/scipy/reference/optimize.html`

`optimize.bisect` takes three arguments: a Python function that represents the mathematical equation for which we are computing the root (given as a Python function name or a lambda function). The second and third arguments are the bounds of the interval bracketing the root (the sign of the function has to be different for those two values in order for bisection to work). Example `[1]` contains some Python code for computing a root in the interval (-1, 0) for function $f(x) = x^4 - (2x + 1)$ using a Python function as an argument. A lambda statement was given in example `[2]`.

In [1]:
```
import scipy.optimize as optimize

def fx(x):
    return x**4 - (2*x+1)

optimize.bisect(fx, 1, 2)
```

Out[1]:
```
1.395336994466561
```

In [2]:
```
import scipy.optimize as optimize

optimize.bisect(lambda x: x**4 - (2*x+1), 1, 2)
```

Out[2]:
```
1.395336994466561
```

Example `[3]` shows the usage of the `optimize.ridder` algorithm. Once again, the second and third argument define the bracketing interval and the first argument is a solved equation.

In [3]:
```
import scipy.optimize as optimize

def fx(x):
    return x**4 - (2*x+1)

optimize.ridder(fx, 1, 2)
```

Out[3]:
```
1.3953369944679295
```

Newton-Raphson method does not fail, when $f(a)$ and $f(b)$ have the same sign. `optimize.newton` computes a root of a function, based on the function itself (again, defined using `def` or as `lambda`) and an initial guess of the root. Optionally, if you specify the derivative of a function using `fprime` keyword argument the Newton's method is used. Otherwise, the secant method is used to determine the value of the root (examples `[4]` and `[5]`). Using the Newton's method we have less control of which root is computed.

In [4]:
```
import scipy.optimize as optimize

def fx(x):
    return x**4 - (2*x+1)

def fprimex(x): # derivative of fx
    return 4 * x**3 - 2

optimize.newton(fx, 1, fprime=fprimex)
```

Out[4]:
```
1.395336994467073
```

In [5]:
```python
import scipy.optimize as optimize

def fx(x):
    return x**4 - (2*x+1)

optimize.newton(lambda x: x**4 - (2*x+1), 1)
```

Out[5]:
```
1.3953369944670728
```

In Example [6], an automatic computation of the derivative using SymPy module is shown.

In [6]:
```python
# automatic computation of a derivative
import sympy
import scipy.optimize as optimize

s_x = sympy.symbols("x") # define variables of the function
s_f = s_x ** 4 - (2 * s_x +1) # define a symbolic function using the variable
f = lambda x: sympy.lambdify(s_x, s_f, 'numpy')(x) # convert function to lambda for
    quick evaluation
fp = lambda x: sympy.lambdify(s_x, sympy.diff(s_f, s_x), 'numpy')(x) # compute
    symbolic derivative and convert to

optimize.newton(f, 1.5, fprime=fp)
```

Out[6]:
```
1.395336994467073
```

13.8 Exercises

Exercise 13.1. A root of
$$x^3 - 10x^2 + 5 = 0$$
lies in the interval $(0,\ 1)$. Compute this root to within 10^{-4}.

Exercise 13.2. Sketch the graphs of $y = x$ and $y = 2\sin x$. Use the bisection method to find an approximation to within 10^{-5} to the first positive value of x with $x = 2\sin x$.

Exercise 13.3. Use bisection to find the root of
$$x^3 - 10x^2 + 5 = 0$$
knowing that it lies in the interval $(0,\ 1)$. Find the solution to four-digit accuracy. How many function evaluations are involved in the procedure?

Exercise 13.4. Use the Newton-Raphson method to obtain successive approximations of $\sqrt{2}$. Stop computation when:

- $|f(x)| < 10^{-6}$
- $x_{k+1} - x_k < 10^{-4}$

Exercise 13.5. Find the smallest positive zero of
$$f(x) = x^4 - 6.4x^3 + 6.45x^2 + 20.538x - 31.752.$$
Find brackets of the roots by visualizing the function first. Note that this function has a double root, and both, the incremental search and bisection would fail.

Exercise 13.6. Determine the root of

$$f(x) = x^3 - 10x^2 + 5 = 0$$

with Ridder's method, knowing that it lies in $(0.6, 0.8)$. Compare the number of iterations needed with one of the linear interpolation methods.

13.9 Useful links

- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.bisect.html#scipy.optimize.bisect`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html#scipy.optimize.newton`
- `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.ridder.html#scipy.optimize.ridder`

References

Burden, R. L. and Faires, J. D. (2011). *Numerical Analysis.* Cengage, 9th edition.

Gerald, C. F. and Wheatley, P. O. (2004). *Applied Numerical Analysis.* Pearson Addison Wesley.

Kiusalaas, J. (2013). *Numerical methods in Python.*

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical recipes. The art of scientific computing.* Cambridge University Press, Cambridge, 3rd edition.

Sauer, T. (2012). *Numerical Analysis.* Pearson, USA, 2nd edition.

Stoer, J. and Bulirsch, R. (2013). *Introduction to Numerical Analysis*, volume 12. Springer Science & Business Media, 2nd edition.