Lecture 4

Numerical Techniques: Root Searching

Blackbody Radiation

IKC-MH.55 *Scientific Computing with Python* at November 03, 2023

Dr. Cem Özdoğan
Engineering Sciences Department
İzmir Kâtip Çelebi University

Contents

Main Topics I

- Solve “$f(x) = 0$”
  - where $f(x)$ is a function of x.
  - The values of x that make $f(x) = 0$ are called the roots of the equation.
- The following non-linear equation can compute the friction factor, f:

$$\frac{1}{\sqrt{f}} = \left(\frac{1}{k}\right) ln(RE\sqrt{f}) + \left(14 - \frac{5.6}{k}\right)$$

- The equation for f is not solvable except by the numerical procedures.

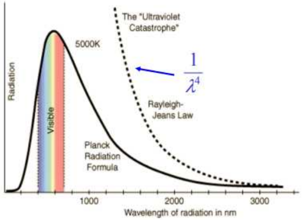
Main Topics II

1. **Interval Halving (Bisection)**. Describes a method that is very simple and foolproof but is not very efficient. We examine how the error decreases as the method continues.
2. **Linear Interpolation Methods**. Tells how approximating the function in the vicinity of the root with a straight line can find a root more efficiently. It has a better "rate of convergence".
3. **Newton-Raphson Method**. Explains a still more efficient method that is very widely used but there are pitfalls that you should know about. Complex roots can be found if complex arithmetic is employed.

Blackbody Radiation I

- Some observations in physics experiments at the beginning of the 20th century could not be explained by classical theories (Blackbody Radiation, The Photoelectric Effect, The Hydrogen Atom, ...).
- Among them, the phenomenon called **Blackbody Radiation** has a special place.
- By definition, a blackbody is an object that absorbs any heat radiation falling on it.
- Rayleigh and Jean proposed that infinitesimal amounts of energy were continuously added to the system when the frequency was increased.
- **Classical physics assumed that energy emitted by atomic oscillations could have any continuous value**.
- If we try and sum the energies at each frequency we find that there is an **infinite** energy in this system!
- This paradox was called the ULTRAVIOLET CATASTROPHE.

Blackbody Radiation II



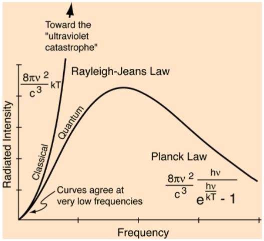


Figure: Variation of energy density with wavelength/frequency in blackbody radiation.

- Spectrum of the radiation inside the blackbody can be measured experimentally.

$$dU = u(\lambda, T)d\lambda$$

- The experimentally observed curve $u(\lambda, T)$ at T=5000 K is shown in the upper figure.
- It has not been possible to explain this observed spectrum of blackbody radiation with classical theories (in the classical limit of large λ, Rayleigh-Jeans Law).

Blackbody Radiation III

- Later, in 1901, Planck was able to come up with the formula that fully explained this curve, with an assumption that predicted the *quantum nature of light*:

$$u(\lambda, T) = \frac{8\pi hc}{\lambda^5} \frac{1}{e^{hc/\lambda k_B T} - 1} \quad (1)$$

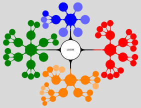
  Here c is the speed of light and $h = 6.63 \times 10^{-34}$ *joule.s* becomes a new constant in physics by the name of Planck's constant.
- Planck's assumption was as follows: **The energy was quantized and could be emitted or absorbed only in integral multiples of a small unit of energy, known as a quantum.**
- The energy distribution of a radiation with a frequency $\nu = hc/\lambda$ is a multiple of an amount of $h\nu$:

$$E = nh\nu \ \ (n = 1, 2, 3, \ldots)$$

- Blackbody radiation helped to move our understanding in physics from a classical approach to a quantum one.

Blackbody Radiation IV

- Some notable features of the Planck distribution are:
  1. Total radiant energy (ie the area under the curve in the Figure)
  $$\int_0^\infty u(\lambda, T) d\lambda = \sigma T^4$$
  is proportional to the 4th power of the temperature T. This is called the **Stefan-Boltzmann formula**.
  2. There is a simple relation between the wavelength λ_{max} at which each curve has a maximum value and the equilibrium temperature T
  $$\lambda_{max} T = 0.0029 \; m.K$$
  This equation is called the **Wien's displacement law**.
- Here we will obtain the Wien's displacement law by numerical method.



Blackbody Radiation V

- To find the maximum wavelength, it is sufficient to take the derivative of Equation 1 with respect to the wavelength λ and set it to zero. However, it is convenient to do this based on the dimensionless variable $x = hc/k_B T\lambda$:

$$u(x) = A\frac{x^5}{e^x - 1} \quad (x = hc/k_B T\lambda \text{ and } A = 8\pi(k_B T)^5/(hc)^4)$$

$$\frac{du}{dx} = A\frac{5x^4(e^x - 1) - x^5 e^x}{(e^x - 1)^2} = 0$$

$$\boxed{(5 - x) - 5e^{-x} = 0 \qquad (2)}$$

- If this equation is solved, x_{max} and then λ_{max} can be found. However, Equation 2 has no analytical solution.
- We can find the answer with the numerical root finding methods.

Bisection I

- Interval halving (bisection), an ancient but effective method for finding a zero of $f(x)$.
- It begins with two values for x that bracket a root.
- **The function** $f(x)$ **changes signs at these two x-values** and, if $f(x)$ is continuous, there must be at least one root between the values.
- The test to see that $f(x)$ does change sign between points a and b is to see if $f(a) * f(b) < 0$ (see Fig.).

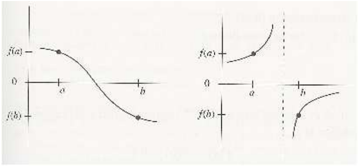


Figure: Testing for a change in sign of f(x) will bracket either a root or singularity.

The bisection method then

- successively divides the initial interval in half,
- finds in which half the root(s) must lie,
- and repeats with the endpoints of the smaller interval.

Bisection II

- A plot of $f(x)$ is useful to know where to start.
- **Example**: The function; $f(x) = 3x + sin(x) - e^x$
- Look at to the plot of the function to learn where the function crosses the x-axis.

```python
import numpy as np
from math import *
def f(x):
    return 3*x+sin(x)-exp(x)
xval=[]
funval=[]
kfirst=0.0; klast=2.0; kincrement=0.01
for j in np.arange(kfirst, klast +
        kincrement, kincrement):
    xval.append(j)
    funval.append(f(j))
import matplotlib.pyplot as plt
plt.title('$f(x)=3x+sin(x)-e^x$')
plt.xlabel('X Value')
plt.ylabel('Function Value')
plt.plot(xval, funval, '-')
plt.grid()
plt.savefig('function_plot.eps')
plt.show()
```

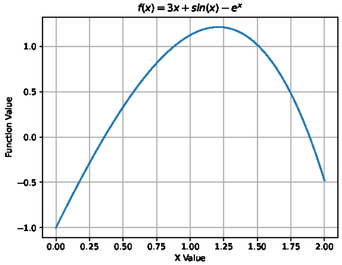
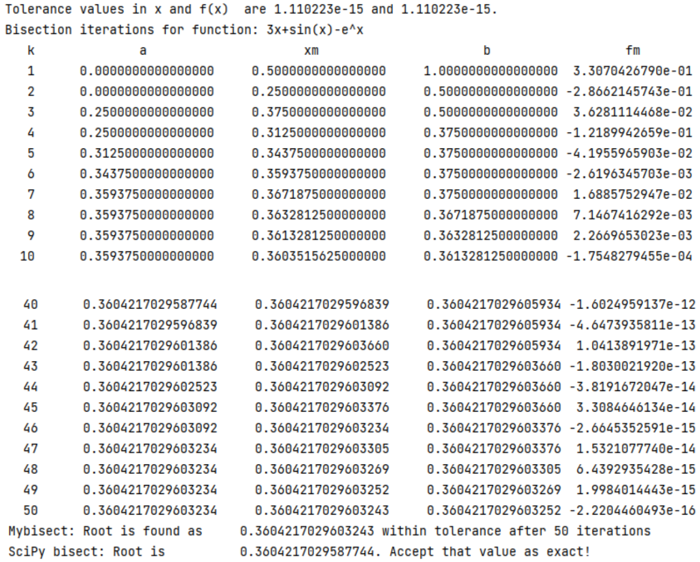


Figure: Code and plot of the function: $f(x) = 3x + sin(x) - e^x$

- We see from the figure that indicates there are zeros at about $x = 0.35$ and 1.9.

Bisection III

- Apply Bisection method to $f(x) = 3x + sin(x) - e^x$. (**Example py-file:** mybisect.py)



```
Tolerance values in x and f(x)  are 1.110223e-15 and 1.110223e-15.
Bisection iterations for function: 3x+sin(x)-e^x
  k          a                  xm                  b                fm
  1     0.0000000000000000   0.5000000000000000   1.0000000000000000  3.3070426790e-01
  2     0.0000000000000000   0.2500000000000000   0.5000000000000000 -2.8662145743e-01
  3     0.2500000000000000   0.3750000000000000   0.5000000000000000  3.6281114468e-02
  4     0.2500000000000000   0.3125000000000000   0.3750000000000000 -1.2189942659e-01
  5     0.3125000000000000   0.3437500000000000   0.3750000000000000 -4.1955965903e-02
  6     0.3437500000000000   0.3593750000000000   0.3750000000000000 -2.6196345703e-03
  7     0.3593750000000000   0.3671875000000000   0.3750000000000000  1.6885752947e-02
  8     0.3593750000000000   0.3632812500000000   0.3671875000000000  7.1467416292e-03
  9     0.3593750000000000   0.3613281250000000   0.3632812500000000  2.2669653023e-03
 10     0.3593750000000000   0.3603515625000000   0.3613281250000000 -1.7548279455e-04


 40     0.3604217029587744   0.3604217029596839   0.3604217029605934 -1.6024959137e-12
 41     0.3604217029596839   0.3604217029601386   0.3604217029605934 -4.6473935811e-13
 42     0.3604217029601386   0.3604217029603660   0.3604217029605934  1.0413891971e-13
 43     0.3604217029601386   0.3604217029602523   0.3604217029603660 -1.8030021920e-13
 44     0.3604217029602523   0.3604217029603092   0.3604217029603660 -3.8191672047e-14
 45     0.3604217029603092   0.3604217029603376   0.3604217029603660  3.3084646134e-14
 46     0.3604217029603092   0.3604217029603234   0.3604217029603376 -2.6645352591e-15
 47     0.3604217029603234   0.3604217029603305   0.3604217029603376  1.5321077740e-14
 48     0.3604217029603234   0.3604217029603269   0.3604217029603305  6.4392935428e-15
 49     0.3604217029603234   0.3604217029603252   0.3604217029603269  1.9984014443e-15
 50     0.3604217029603234   0.3604217029603243   0.3604217029603252 -2.2204460493e-16
Mybisect: Root is found as    0.3604217029603243 within tolerance after 50 iterations
SciPy bisect: Root is         0.3604217029587744. Accept that value as exact!
```

Bisection IV

The root is (almost) never known exactly, since it is extremely unlikely that a numerical procedure will find the precise value of x that makes $f(x)$ exactly zero in floating-point arithmetic.

- The main advantage of interval halving is that it is **guaranteed to work** (continuous & bracket).
- The algorithm must decide how close to the root the guess should be before stopping the search (see Fig.).
- The major objection of interval halving has been that it is **slow to converge**.

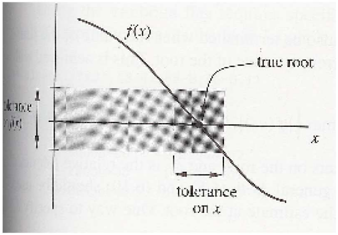


Figure: The stopping criterion for a root-finding procedure should involve a tolerance on x, as well as a tolerance on $f(x)$.

Bisection is generally recommended for finding an approximate value for the root, and then this value is refined by more efficient methods.

Linear Interpolation Methods - The Secant Method I

- Bisection is simple to understand but it is <u>not</u> the most <u>efficient</u> way to find where $f(x)$ is zero.
- Most functions can be approximated by a straight line over a small interval.

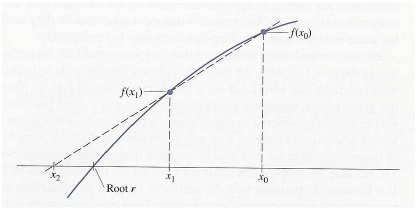


Figure: Graphical illustration of the Secant Method.

- The secant method begins by finding *two points on the curve* of $f(x)$, hopefully near to the root.
- As Figure illustrates, we draw the line through these two points and find where it intersects the x-axis.
- If $f(x)$ were truly linear, the straight line would intersect the x-axis at the root.

Linear Interpolation Methods - The Secant Method II

- From the obvious similar triangles we can write

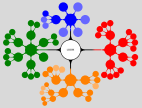
$$\frac{(x_1 - x_2)}{f(x_1)} = \frac{(x_0 - x_1)}{f(x_0) - f(x_1)} \Longrightarrow x_2 = x_1 - f(x_1)\frac{(x_0 - x_1)}{f(x_0) - f(x_1)}$$

- Because f(x) is not exactly linear, x_2 is not equal to r,
- but it should be closer than either of the two points we began with. If we repeat this, we have:

$$\boxed{x_{n+1} = x_n - f(x_n)\frac{(x_{n-1} - x_n)}{f(x_{n-1}) - f(x_n)},\ n = 1, 2, \ldots}$$

- The net effect of this rule is to set $x_0 = x_1$ and $x_1 = x_2$, after each iteration.

The technique we have described is known as, the <u>secant method</u> because the <u>line through two points on the curve</u> is called the <u>secant line</u>.



Linear Interpolation Methods - The Secant Method III

- Apply Secant method to $f(x) = 3x + sin(x) - e^x$. (**Example py-file:** mysecant.py)

Table: The Secant method for $f(x) = 3x + sin(x) - e^x$, starting from $x_0 = 1, x_1 = 0$, using a tolerance value of 1E-16.

```
Tolerance values in x and f(x)  are 1.110223e-15 and 1.110223e-15.
Secant iterations for function: 3x+sin(x)-e^x
   k            a                    xm                    b                  fm
   1     0.0000000000000000     0.4709895945962973     1.0000000000000000  2.6515881591e-01
   2     1.0000000000000000     0.3075084610161186     0.4709895945962973 -1.3482201684e-01
   3     0.4709895945962973     0.3626132418960405     0.3075084610161186  5.4785286937e-03
   4     0.3075084610161186     0.3604614817438951     0.3626132418960405  9.9517712217e-05
   5     0.3626132418960405     0.3604216717772825     0.3604614817438951 -7.8014178007e-08
   6     0.3604614817438951     0.3604217029607673     0.3604216717772825  1.1080025786e-12
   7     0.3604216717772825     0.3604217029603244     0.3604217029607673  0.0000000000e+00
Mysecant: Root is found as     0.3604217029603244 within tolerance after 7 iterations
SciPy secant: Root is          0.3604217029603243. Accept that value as exact!
```

- Table shows the results from the secant method for the same function that was used to illustrate bisection method.
- If $f(x)$ is far from linear near the root or not continuous, the method may fail. A plot of $f(x)$ is useful to know where/how to start.

Newton's Method I

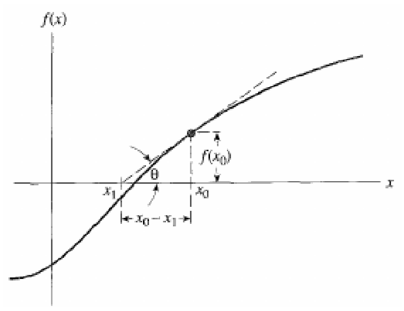


Figure: Graphical illustration of the Newton's Method.

- One of the most widely used methods of solving equations is Newton's method.
- Like the previous ones, this method is also based on a linear approximation of the function, but does so using a tangent to the curve (see Figure).

- Starting from a single initial estimate, x_0, that is not too far from a root, we move along the tangent to its intersection with the x-axis, and take that as the next approximation.
- This is continued until either the successive x-values are sufficiently close or the value of the function is sufficiently near zero.

Newton's Method II

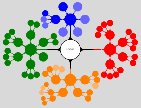
- The calculation scheme follows immediately from the right triangle shown in Figure.

$$tan\theta = f^{'}(x_0) = \frac{f(x_0)}{x_0 - x_1} \Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

and the general term is:

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},\ n = 0, 1, 2, \ldots}$$

- Newton's algorithm is widely used because, it is more rapidly convergent than any of the methods discussed so far. **Quadratically convergent**
- The error of each step approaches a constant K times the square of the error of the previous step.



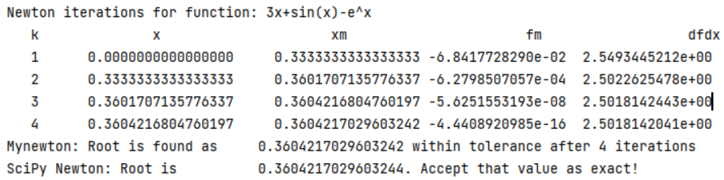
Newton's Method III

- The number of decimal places of accuracy nearly doubles at each iteration.
- There is the need for two functions evaluations at each step, $f(x_n)$ and $f'(x_n)$ and we must obtain the derivative function at the start.
- If a difficult problem requires many iterations to converge, the number of function evaluations with Newton's method may be many more than with linear iteration methods. Because Newton's method always uses two per iteration whereas the others take only one.
- The method may converge to a root different from the expected one or diverge if the starting value is not close enough to the root.

Newton's Method IV

- Apply Newton's method to $f(x) = 3x + sin(x) - e^x$. (**Example py-file:** mynewton.py)
- Table shows the results from Newton's method for the same function that was used to illustrate bisection and secant methods.

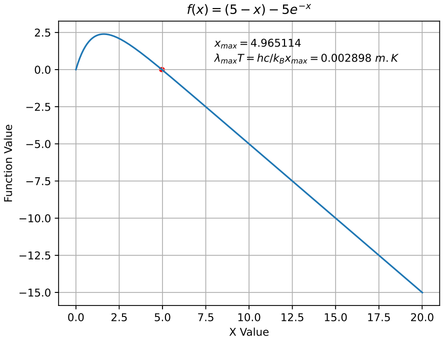
Table: Newton's method for $f(x) = 3x + sin(x) - e^x$, starting from $x_0 = 0$, using a tolerance value of 1E-16.



```
Newton iterations for function: 3x+sin(x)-e^x
  k          x                      xm                    fm                dfdx
  1      0.0000000000000000     0.3333333333333333 -6.8417728290e-02  2.5493445212e+00
  2      0.3333333333333333     0.3601707135776337 -6.2798507057e-04  2.5022625478e+00
  3      0.3601707135776337     0.3604216804760197 -5.6251553193e-08  2.5018142443e+00
  4      0.3604216804760197     0.3604217029603242 -4.4408920985e-16  2.5018142041e+00
Mynewton: Root is found as    0.3604217029603242 within tolerance after 4 iterations
SciPy Newton: Root is         0.3604217029603244. Accept that value as exact!
```

Blackbody Radiation V
(**Example py-file:** blackbodyradiation.py)

```
-------------------------------------------------------------------------------------
Mybisect: Root is found as     4.9651142317442760 within tolerance after 50 iterations
SciPy bisect: Root is          4.9651142317439962. Accept that value as exact!
-------------------------------------------------------------------------------------
Mysecant: Root is found as     4.9651142317442760 within tolerance after 6 iterations
SciPy secant: Root is          4.9651142317442760. Accept that value as exact!
-------------------------------------------------------------------------------------
Mynewton: Root is found as     4.9651142317442760 within tolerance after 4 iterations
SciPy Newton: Root is          4.9651142317442760. Accept that value as exact!
-------------------------------------------------------------------------------------
```


$f(x) = (5 - x) - 5e^{-x}$
$x_{max} = 4.965114$
$\lambda_{max} T = hc/k_B x_{max} = 0.002898\ m.K$
Function Value
X Value