NUMERICAL MODELING

STUDENT MANUAL

ERASMUS+ project "Green Science Teaching Materials for Digital Learning" 2022. - 2025.

Table of Contents

- Introduction
  - What is numerical modeling and why it is important?
  - Why study numerical modeling?
  - What to use numerical modeling for?
- Chapter 1: What do we need to know before starting numerical modeling?
  - 1.1. Plotting graphs with *SageMath*
  - 1.2. Plotting graphs with *Python*
  - 1.3. Plotting graph from data
    - 1.3.1. Reading CSV files
    - 1.3.2. Plotting graphs from scattered data
- Chapter 2: Numerical methods for determining roots of a function
  - 2.1. Root of a function - general
  - 2.2. Bisection method
    - 2.2.1. How to construct program that executes bisection method?
  - 2.3. Secant method
    - 2.3.1. Algorithm for performing the secant method
- Chapter 3: Determination of the maximum and minimum of the function by numerical methods
- Chapter 4: Numerical derivation
  - 4.1. Finite difference method
- Chapter 5: Numerical integration
  - 5.1. Numerical integration methods
  - 5.2. How to approximate integrals using the described methods?
    - 5.2.1. Midpoint rule
    - 5.2.2. Trapezoidal rule
    - 5.2.3. Simpson's rule
  - 5.3. Physics problem: Determination of the potential for a linear charge distribution
- Chapter 6: Numerical solution of ordinary differential equations
- Chapter 7: Numerical methods used for linear regression and least square method
- Chapter 8: Numerical solution of systems of linear equations
- Task solutions
- Literature

Introduction: What is numerical modeling and why it is important?

Although we know that there is no science without an experiment, it is not always possible to simply conduct an experiment to confirm or disprove a hypothesis. However, the principles of scientific disciplines that make science what it is still require conducting experiments. As much as designing and conducting an experiment is an important part of the scientific discipline, it is even more important to explain its results. Even when it is possible to carry out an experiment, the result is usually a large amount of, at first glance, unrelated data that needs to be analyzed and given some kind of meaning. Mathematics gives us many ways in which experimental data can be analyzed, but these can be time-consuming processes. That changed with the invention and development of computers because they can analyze large groups of data faster and more precisely. And when we put all these problems and their solutions together, we get what is known as the numerical modeling technique. It is a computational technique used to simulate and analyze complex systems or processes using mathematical models and computer simulations. Today, it is used daily in natural and social sciences and engineering in order to predict the behavior of complex systems and gain insight into their behavior in different conditions. Numerical modeling allows us to analyze those systems that are difficult or impossible to analyze using a traditional experiment, but even in cases where this is possible, it is useful for optimizing the experiment before conducting it in a traditional way (thus reducing errors when conducting a physical experiment and saving time and resources).

Why study numerical modeling?

This technique is an integral part of modern science and every future scientist will encounter it during his or her career. So why not start as early as possible and give future scientists an insight into how modern science works. Numerical modeling is based on interdisciplinarity - when we use numerical modeling for a system or a problem, mathematical and computer knowledge are simultaneously used for the purpose of describing a specific problem within a certain natural or social science. In this way, we apply and connect knowledge from areas that we normally look at separately, which helps us develop divergent thinking. Also, to apply the principles of numerical modeling, it is necessary to use a scientific approach to the problem, which includes setting up a hypothesis, modeling the situation/system (retaining what is essential for a specific problem), searching for the best way to solve the problem, analyzing the results and comparing or applying them to problems of a similar type. Although with numerical modeling we do not put emphasis on setting up a hypothesis but on the way of testing hypotheses (because it is used as a tool), this technique still provides an insight into the scientific method and the problems that scientists face during the process.

What to use numerical modeling for?

There are many tools that can be used for numerical modeling, from the programs we use every day to process data such as MS Excel or LibreCalc Office, to more advanced computer software such as Wolfram Mathematica, MatLab or similar alternatives to Wolfram Mathematica. Many of these more advanced mathematical software that we can use for numerical modeling must be purchased to use, but there are free alternatives. One of the most widely used free math software is SageMath. SageMath is a free and open-source mathematical software that integrates many open-source packages such as NumPy, SciPy, matplotlib, SymPy and many others (almost 100 packages). The user interface is a notebook that opens in a web browser or command line. The notebook used by SageMath is Jupyter Notebook which supports more than 40 programming languages, one of programming language Jupyter Notebook supports is Python, which is great for all our numerical modeling needs. Inside the notebook, you can write text and code, create graphics, write equations and share everything with other users. You can find more about SageMath and Jupyter Notebook from their websites www.sagemath.org (https://www.sagemath.org/index.html) and www.jupyter.org (https://jupyter.org/).

Chapter 1: What do we need to know before starting numerical modeling?

> *"A picture is worth a thousand words."* Frederick R. Barnard

The famous saying also tells us a lot within the context of science because an essential way of visualizing scientific problems is the graphical representation of data. Although we mostly display data in tables, graphs are a much more powerful way of representing given data and good graphical representation of a problem helps us with a simpler data interpretation. With the use of computers and computer programs, drawing graphs has become much simpler and faster, especially when dealing with a large amount of data. Graphical representation of data is also essential for numerical modeling, therefore it is important to know how to correctly and simply display data in *SageMath* and using *Python*.

1.1. Plotting graphs with *SageMath*

Drawing graphs within SageMath will be shown on a few simple examples. Using *SageMath*, we can display a function graphically using the `plot()` function, within which we can use some additional arguments. Let's show some of the possibilities with examples.

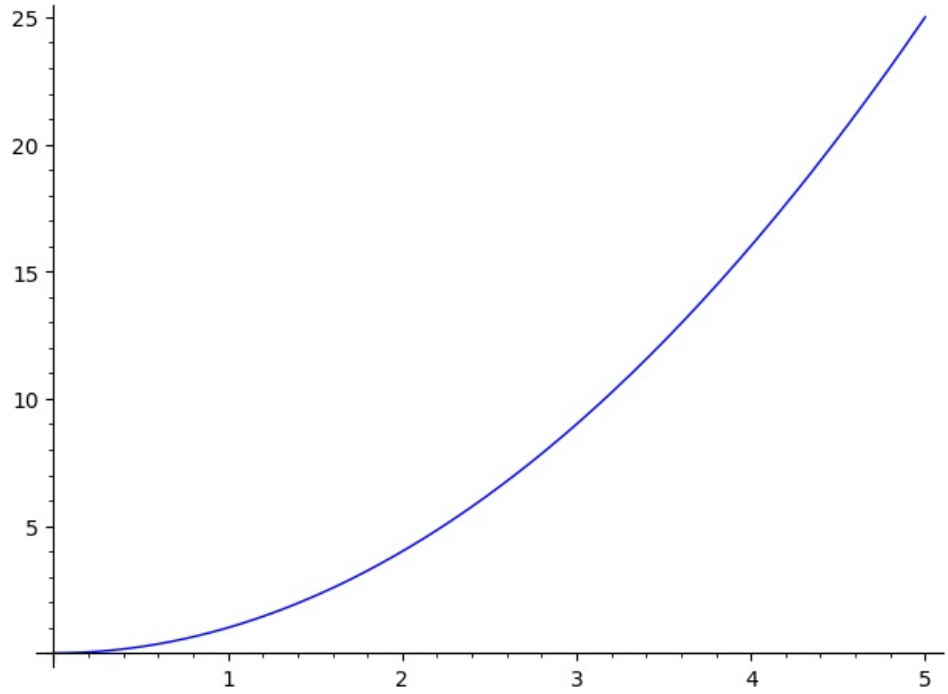
Example 1.1.

Let's draw the graph of the function x^2 for x in the interval $[0, 5]$.

In [1]:

```
plot(x^2,(x,0,5))
```

Out[1]:



If we want to upgrade our graph, it is necessary to specify arguments. Some of the arguments we use most often are listed in the Table below.

title=''	**axes_labels=[]**	**color=''**	**thickness**	**linestyle**	**legend_label=''**
graph title	axes names\	line color\	line thickness\	line appearance\	function name on the legend\

You can see a list of other possible arguments and how they are used using the `help(plot)` command.

Task 1.1.

Draw the graph of the function x^2 for x in the interval $[0, 5]$. a. Add a graph title and axis names. b. Change the line color, thickness and appearance of the line as desired. (We can change the color of the line in several ways - try changing the color in different ways).

TASK SOLUTION

```
plot(x^2,(-5,5),color='purple',thickness='2', linestyle= '--', title=('$x^{2}$'), axes_labels=(['$x$','$f(x)$']))
```

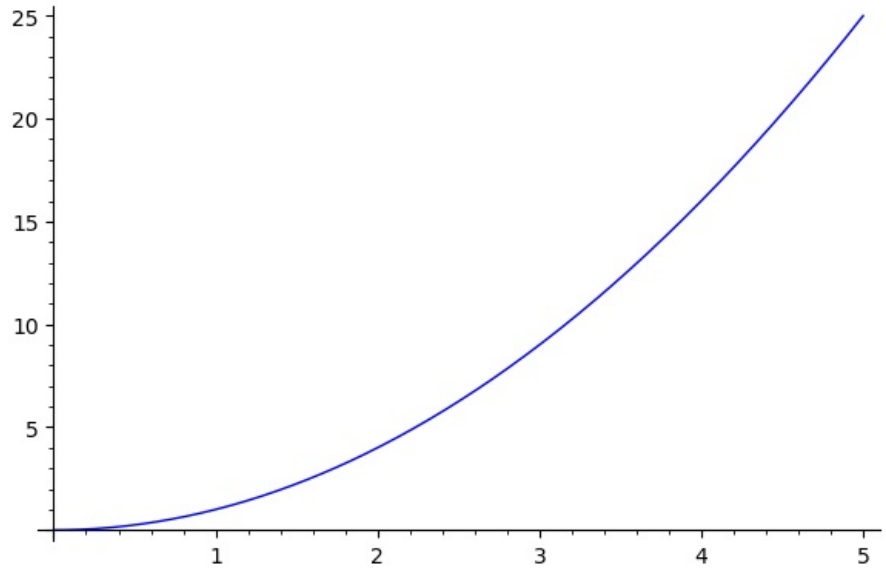
Graphs can also be drawn by attaching a variable to the graphic object, which can then be called later or simply display multiple functions on the same graph. In the following example, in addition to the graphical representation of the quadratic function, we will also add the graphical representation of the function x^3.

Example 1.2.

In [2]:

```
P_quadratic = plot(x^2, (x,0,5))
P_cubic = plot(x^3, (x,0,5))
P_quadratic
```

Out[2]:

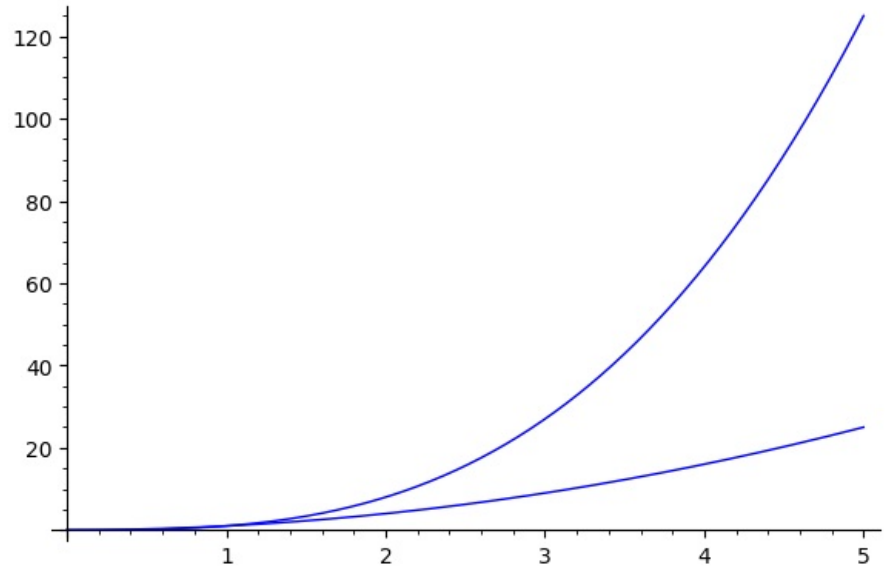


We present both functions as follows.

In [3]:

```
P_quadratic + P_cubic
```

Out[3]:

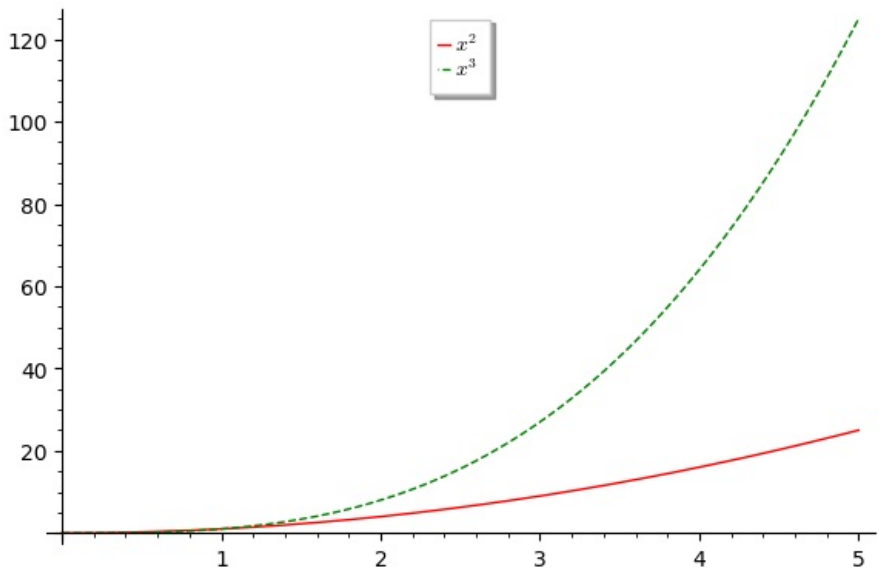


By showing several different functions, we come to the problem of their visual distinction and naming. We can distinguish them simply by changing their representation (color, thickness, appearance,...), but the problem still remains as to which line refers to which displayed function. In the following example, we will show how we can add a legend to the graph.

In [4]:

```
P_quadratic = plot(x^2, (x,0,5), color = 'red', legend_label='$x^2$') #within the plot() function we add an argum
ent with which we assign a name to the graphic display
P_cubic = plot(x^3, (x,0,5), color = 'green',  linestyle = '--', legend_label='$x^3$')
graph = P_quadratic + P_cubic #we save the graphic displays that we want to display together in a new variable
graph.set_legend_options(loc='upper center') #we define the arguments for the legend (in this case the position o
f the legend)
graph
```

Out[4]:



You can also save your graph on your computer in different formats: PDF, png and jpg. The PDF or image of the graph is saved in the directory where your Jupyter notebook is located. We save the graphs using `save('file_name.format')` . If we want to save the graph showing the functions x^2 and x^3 in the form of .png, we would do it by calling the command:

```
graph.save('graph.png')
```

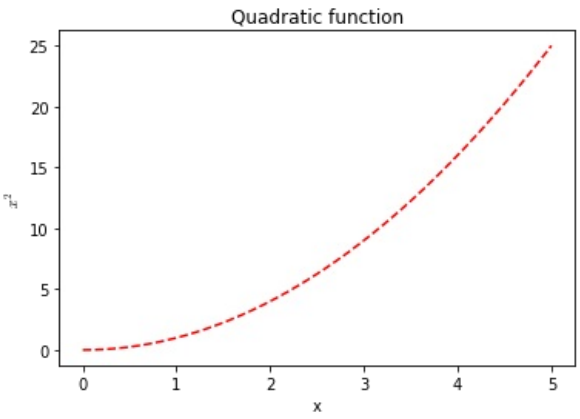
1.2. Plotting graphs with *Python*

For those who are more comfortable with Python, they have come across different modules and libraries that are used within this programming language. As SageMath is based on the Python programming language, all these modules and libraries are integrated within it. So *Python*'s *Matplotlib* library can also help us to draw graphs. Documentation for that library, as well as various tutorials, can be found on its website (https://python-graph-gallery.com/matplotlib/). Within Python, we need the *NumPy Python* package to calculate function values and any other scientific calculations. We can find necessary documentation at its website (https://numpy.org/).

The following example shows how to draw a graph using the *Matplotlib* library.

In [5]:

```
import matplotlib.pyplot as plt #accessing required modules
import numpy as np
x = np.linspace(0, 5, 10000) #defining the x coordinate range (start, stop, number of generated samples)
plt.plot(x,x^2, color='red', linestyle='--')
plt.title("Quadratic function")
plt.xlabel("x")
plt.ylabel("$x^{2}$");
```



Task 1.2.

We know that the average speed of the body can be determined as:

$$v = \frac{\Delta x}{\Delta t},$$

and in the case of uniform motion, the instantaneous speed is equal to the average speed. Knowing that $\Delta x = x - x_0$ and $\Delta t = t - t_0$, we can arrive at the following expression for the dependence of position on time for uniform motion:

$$x = vt + x_0 - vt_0.$$

Draw a graph of the dependence of the position of the body on time for a body moving uniformly along the line for two different bodies for which you will choose the values v, t_0 and x_0 yourself. Name the axes (with units of measurement), show the difference between the motion displays in some way (different line thickness, color or shape), add a legend and position it to best fit your graph. Do the task in two ways:

1. using *SageMath* commands and
2. using the *Python* library *Matplotlib* (you need to research how to add a legend in this case).

TASK SOLUTION

```
# using SageMath commands
v = 10
x_0 = 10
t_0 = 10
var('t')
p1 = plot(v*t+x_0-v*t_0,(0,30), color='purple', linestyle= '--', axes_labels=(['$t/s$','$x/m$']), legend_l
abel='$v=10$ $m/s$, $x_{0}=10$ $m$, $t_{0}=10$ $s$')
v1 = 5
x_01 = 5
t_01 = 0
p2 = plot(v1*t+x_01-v*t_01,(0,30), color='orange', linestyle= '-', axes_labels=(['$t/s$','$x/m$']), legend
_label='$v=5$ $m/s$, $x_{0}=5$ $m$, $t_{0}=0$ $s$')
p1 + p2
```

```
# using Python library Matplotlib
import matplotlib.pyplot as plt
import numpy as np

t = var ('t')
t = np.linspace(0, 30, 100)
v = 10
x_0 = 10
t_0 = 10
plt.plot(t, v*t+x_0-v*t_0, color='purple', linestyle='--', label = '$v=10$ $m/s$, $x_{0}=10$ $m$, $t_{0}=1
0$ $s$')
v1 = 5
x_01 = 5
t_01 = 0
plt.plot(t, v1*t+x_01-v*t_01, color='orange', linestyle= '-', label = '$v=5$ $m/s$, $x_{0}=5$ $m$, $t_{0}=
0$ $s$')
plt.xlabel("$t/s$")
plt.ylabel("$x/m$")
plt.legend()
plt.show()
```

1.3. Plotting graph from data

When we experiment in science we do not get continuous data values as we have shown so far, but we get points in the coordinate system. Also, if we process measurement data on a computer, they can be saved in a file. This is especially the case when dealing with a large amount of data. In this subsection, we will see how we can retrieve ("read") the data saved in the file and how we can display the given data graphically.

When exchanging data between different applications and programs, most often used type files for text are CSV files (*Comma Separated Values*) or text files with the `.txt` extension. Within *SageMath* itself, reading such files is a bit more complicated, but as *SageMath* is based on programming language *Python*, its libraries and built-in functions make it easier for us to manipulate CSV files. Within *Python*, there are two libraries that help us read text files, as well as write data to such files. When we work with a smaller amount of data (which we will do most of the time), it is enough to use *Python*'s `CSV module`. On the other hand, if we need to work with a large amount of data or numerical analysis on a large amount of data, then we can use the `Pandas library`. Here we will show how to read files using the `CSV module` and, if necessary, you can find the documentation for the `Pandas library` at the official website (https://pandas.pydata.org/).

1.3.1. Reading CSV files

Let's say that we want to read the file `SI.csv`, which is attached to this manual. Inside the file are the basic quantities of the International System of units (SI), their symbols, the corresponding units of measurement and the symbols of these units of measurement. For the simplest reading of a file, that file must be saved in the same directory as the Jupyter notebook in which we open the desired file. Otherwise, we must specify the path to the file along with the file name. The simplest way to read a file is shown in the following example:

In [6]:

```
SI = open('SI.csv', 'r') #opens the "SI.csv" file and saves the data in the SI parameter
for line in SI:
    print (line)
SI.close() #closes file
```

```
base quantity, base quantity symbol, unit of measurement, unit of measurement symbol

lenght, l, meter, m

mass, m, kilogram, k

time, t, second, s

thermodynamic themperature, T, kelvin, K

amount of substance, n, mole, mol

electric current, I, amper, A

luminous intensity, I, candela, cd
```

Using the `readline()` function we can read only the first line inside the file.

In [7]:

```
SI = open('SI.csv', 'r')
SI1 = SI.readlines()
new_list=[]
for line in SI1:
    new_list.append(line.rstrip('\n'))
print(new_list)
SI.close()
```

```
['base quantity, base quantity symbol, unit of measurement, unit of measurement symbol', 'lenght, l,
meter, m', 'mass, m, kilogram, k', 'time, t, second, s', 'thermodynamic themperature, T, kelvin, K',
'amount of substance, n, mole, mol', 'electric current, I, amper, A', 'luminous intensity, I, candel
a, cd']
```

The method shown above will work well for files that contain little data. As soon as we have a file with a large amount of data, this method of reading the file will take up a lot of memory, so it is better to avoid it. A simpler way of working and a more elegant way to manipulate data is via *Python*'s `CSV module`. If we work in *Python*, then first of all we have to import the `CSV module` with the command

```
import csv
```

and only then we can access its options. Let's look at this way of reading files and some of the possibilities through an example.

In [8]:

```
import csv

with open('SI.csv', 'r') as file: #opens file for reading (and closes after completion)
    SI = csv.reader(file)
    for line in SI:
        print(line)
```

```
['base quantity', ' base quantity symbol', ' unit of measurement', ' unit of measurement symbol']
['lenght', ' l', ' meter', ' m']
['mass', ' m', ' kilogram', ' k']
['time', ' t', ' second', ' s']
['thermodynamic themperature', ' T', ' kelvin', ' K']
['amount of substance', ' n', ' mole', ' mol']
['electric current', ' I', ' amper', ' A']
['luminous intensity', ' I', ' candela', ' cd']
```

As we can see in this example, the values are already saved in lists by lines, so we can access a single line or column more easily. Another advantage of this method is that the file is closed immediately after reading.

In [9]:

```
import csv

with open('SI.csv', 'r') as file:
    SI = csv.reader(file)
    for line in SI:
        print(line[0]) #prints the values found in each line at position 1
```

```
base quantity
lenght
mass
time
thermodynamic themperature
amount of substance
electric current
luminous intensity
```

So that we don't have to constantly open the file and read it when we want to manipulate the data written in it, it is useful to save the data in a list after reading the file. In this way, we can access all data even when the file is closed.

In [10]:

```
import csv

with open('SI.csv') as file:
    SI = list(csv.reader(file)) #reads the file and saves the data in a list

print(SI)
```

```
[['base quantity', ' base quantity symbol', ' unit of measurement', ' unit of measurement symbol'],
['lenght', ' l', ' meter', ' m'], ['mass', ' m', ' kilogram', ' k'], ['time', ' t', ' second', ' s']
, ['thermodynamic themperature', ' T', ' kelvin', ' K'], ['amount of substance', ' n', ' mole', ' mo
l'], ['electric current', ' I', ' amper', ' A'], ['luminous intensity', ' I', ' candela', ' cd']]
```

We now have a list whose elements are lists containing data from a single line. Now we can easily access an individual list element within a list by calling:

```
listname[x][y]
```

where x represents the list we want to access within the list `listname` , and y is the element of the list we want to access. Let's take a look at the example of the list `SI` in which the data from the file `SI.csv` is saved.

In [11]:

```
print(SI[1][0]) #we access the 1st element in the 1st line
print(SI[1][1]) #we access the 2nd element in the 2nd line
```

```
lenght
 l
```

TASK 1.3.

Download to your computer the file `measurements.csv` , which contains the measurement data for the dependence of the acceleration on the mass of the body under the action of a constant force of 2 N (save the file in the same directory where you will save the notebook in which you are working). The measurement was carried out for four different body masses and four acceleration values were measured for each mass.

1. Read the `measurements.csv` file and save the read data in a sheet.
2. Using `for loop` or `list comprehension` separate each line so that you get 5 lists in which each line will be located separately.
3. Find how you can remove the first element of the list so that we only have measured values in each list and apply that to each list.
4. The data is stored in the form of a string within the list, and we need its numerical value for calculation. Determine the numerical value for each element of each list and store the resulting values in a new or the same list.

TASK SOLUTION:

```
import csv

with open('measurements.csv') as values:
    measurements = list(csv.reader(values))

for a in measurements:
    element_number = len(a)

mass = [float(measurements[0][i]) for i in range(1, element_number)]
a1 = [float(measurements[1][i]) for i in range(1, element_number)]
a2 = [float(measurements[2][i]) for i in range(1, element_number)]
a3 = [float(measurements[3][i]) for i in range(1, element_number)]
a4 = [float(measurements[4][i]) for i in range(1, element_number)]

print(mass)
print(a1)
```

Plotting graphs from scattered data

As we said earlier, with scientific measurements, the data we get are not continuous values but scattered values. That's why it's important to know how to draw a graph from scattered data. Here is an example where it is necessary to plot an $x-t$ graph from data saved in lists using *SageMath* and *Python* built-in functions.

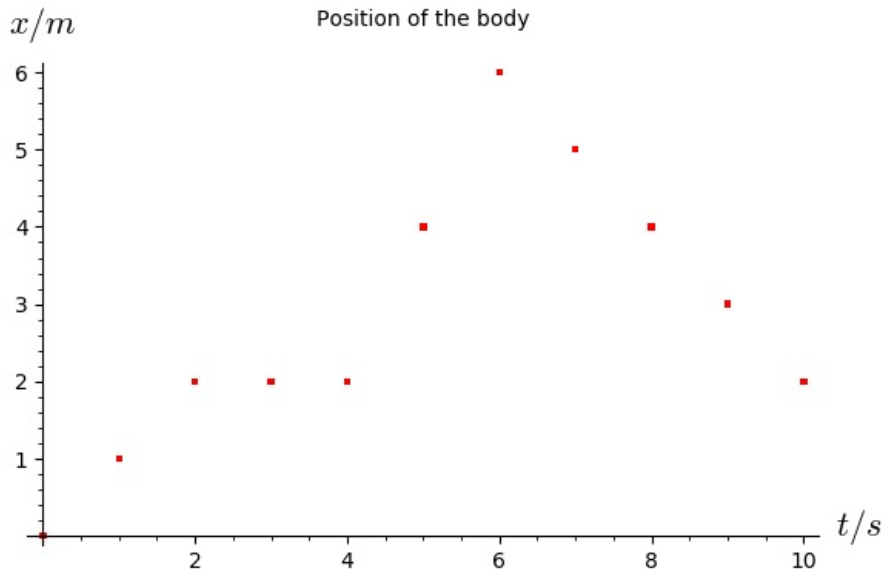
In [12]:

```
#example shown using SageMath functions
t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = [0, 1, 2, 2, 2, 4, 6, 5, 4, 3, 2]

x_t = list(zip(t,x)) #returns a list of tuples which in this case represents coordinates

list_plot(x_t, color = 'red', title = 'Position of the body', axes_labels = ['$t/s$','$x/m$'], marker = "s")
#plots a graph from the list, where we added the name of the graph and axis names and changed the color and shape
of the maker
```

Out[12]:

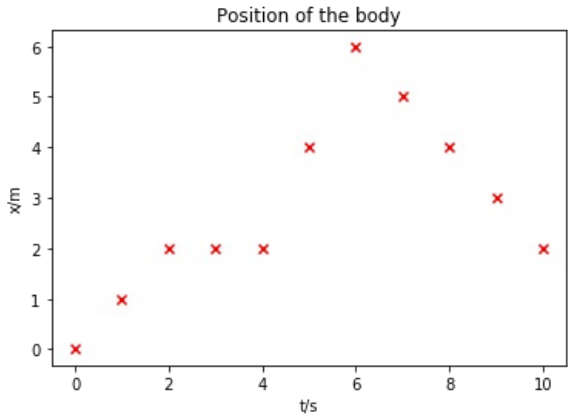


In [13]:

```
#example using Python's Matplotlib library
import matplotlib.pyplot as plt

t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = [0, 1, 2, 2, 2, 4, 6, 5, 4, 3, 2]

plt.scatter(t,x, c = 'red', marker = 'x') #drawing a scatter plot from the list elements t and x and defining add
itional parameters
plt.title("Position of the body")
plt.xlabel("t/s")
plt.ylabel("x/m")
plt.show;
```



See the following link (https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.scatter.html) for more possible parameters.

We can also draw a scatterd plot when the data is stored in a list of *tuples*, which was the case in the first example of drawing a graph with *SageMath* functions. In order to be able to draw a graph from a list of *tuples*, we must first "unpack" that list and that zip in which the data is saved. We can do this using the unpacking operator * which we place before the object we want to unpack. It unpacks the values of any iterable object. For more details about the unpacking operator and how we can use it, you can visit following link (https://realpython.com/python-kwargs-and-args/#unpacking-with-the-asterisk-operators).

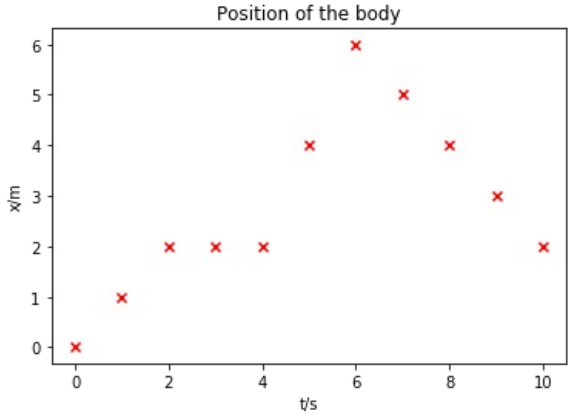
In [14]:

```
import matplotlib.pyplot as plt

t = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x = [0, 1, 2, 2, 2, 4, 6, 5, 4, 3, 2]

x_t = list(zip(t,x))

plt.scatter(*zip(*x_t), c = 'red', marker = 'x')
plt.title("Position of the body")
plt.xlabel("t/s")
plt.ylabel("x/m")
plt.show;
```



TASK 1.4.

After solving the 3rd task, you should have 5 lists containing the numerical values of the measured data for the mass of the body and for the values of the accelerations measured for each mass.

1. Calculate the average values of the accelerations measured for each individual mass and save the obtained average values in a new list. The new list should have the same number of elements as the list in which the mass values are stored.
2. Draw a graph of the dependence of acceleration on body mass. Data on the graph must not be connected by lines. The picture must contain the name of the graph and the names of the axes with the corresponding units of measurement.
3. For mass values in the mass range given by measurements, determine the theoretical acceleration values using Newton's second law of motion if you know that a constant force of $2N$ acted on the bodies. Save the obtained values in a new list.
4. On the same graph as before, add the dependence of acceleration on mass for theoretical values. Show the theoretical curve as a solid line. Add a legend to the graph.

TASK SOLUTION:

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

average = [sum(a1)/len(a1), sum(a2)/len(a2), sum(a3)/len(a3), sum(a4)/len(a4)]
mass_range = np.linspace(mass[0], mass[3], 20)

plt.scatter(average,mass, c = 'red', marker = 'x')
plt.plot(2/mass_range, mass_range)
plt.title("Acceleration-mass dependence")
plt.xlabel("m[kg]")
plt.ylabel("a[$m/s^2$]")
plt.legend(["measurement", "theory"], loc ='upper right')
plt.show()
```

Chapter 2: Numerical methods for determining roots of a function

As important as graphical representations for a simpler presentation of a certain problem are in science, it is equally important to know how to analyze the obtained graphs. Through analysis, we can describe given data in more detail, and perhaps find a solution to our problem. All graphs show some kind of function (the dependence of one variable on another), so the analysis of the graph is actually the analysis of the function that the graph shows. The most basic components of function analysis are finding the roots of the function and its extremes (minimum and maximum). In this chapter we will describe ways of finding the roots of a function using numerical methods, and in the next chapter we will deal with the function extrema.

2.1. Root of a function - general

Roots of a function (or zero points) are the points where the given function intersects the abscissa axis, and we get them by equating the function to zero. Most of the functions you have encountered so far during your education are **polynomials**, functions of the form $a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$ where $n \in \mathrm{N}$ represents the degree of the polynomial. When dealing with a polynomial of low degree ($n = 1.2$), determining the roots of the function is not a problem. If, for example, we have the function $f(x) = 2x + 4$, the procedure for determining the root of the function is very simple: $2x + 4 = 0 \rightarrow 2x = -4 \rightarrow x = -2$. In the case of polynomials of the second degree (quadratic function), the roots of a function $f(x) = ax^2 + bx + c$ are obtained by solving the quadratic equation. And the solution of the quadratic equation is generally determined, by the well-known expression, $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Similar general, although more complicated, expressions also exist for polynomials of the third and fourth degree. However, there is no such general solution for polynomials of the fifth (or higher) degree. Also, functions are not just polynomials. There are also trigonometric functions, exponential functions, or functions that are a combination of the previously mentioned functions. And for many of these functions there is no analytical solution (a solution written using elementary functions) and even if analytical solution exists it is very complicated. Of course, this is not only the case with finding the roots of a function - this problem also occurs with other elements of the analysis of functions and their graphical representations. This is where the need for numerical analysis arises - for finding a satisfactory numerical solution so we could get better understanding of our problem.

In this chapter, we will describe two numerical methods that can be used to determine roots of a function: the **bisection method** and the **secant method**. In addition to these two, there are some other numerical methods for finding the roots of a function, the most famous of which is the *Newton-Raphson method*. Unlike bisection and secant method, the Newton-Raphson method requires fewer iterations to reach the desired precision, but we will not show it here because it requires greater knowledge of mathematics.

2.2. Bisection method

The basic idea is very simple - let's say that it is known that the root of the function $f(x)$ is in the interval $[a, b]$. In the case of functions that we can display graphically, we can determine that interval from the graphical display. If the graphic representation of the function is complicated to determine, we can find the interval in which the root of the function is located by looking at the values in which the function changes its sign. In that case, we need to choose two

arbitrary values $x = a$ and $x = b$ and determine their value, $f(a)$ and $f(b)$. If $f(a) > 0$ (positive) and $f(b) < 0$ (negative) or vice versa, then there is a root in the interval $[a, b]$ because within that interval the function had to pass through abscissa axis to change the sign. A general example for finding the appropriate interval is shown in Fig. 1.

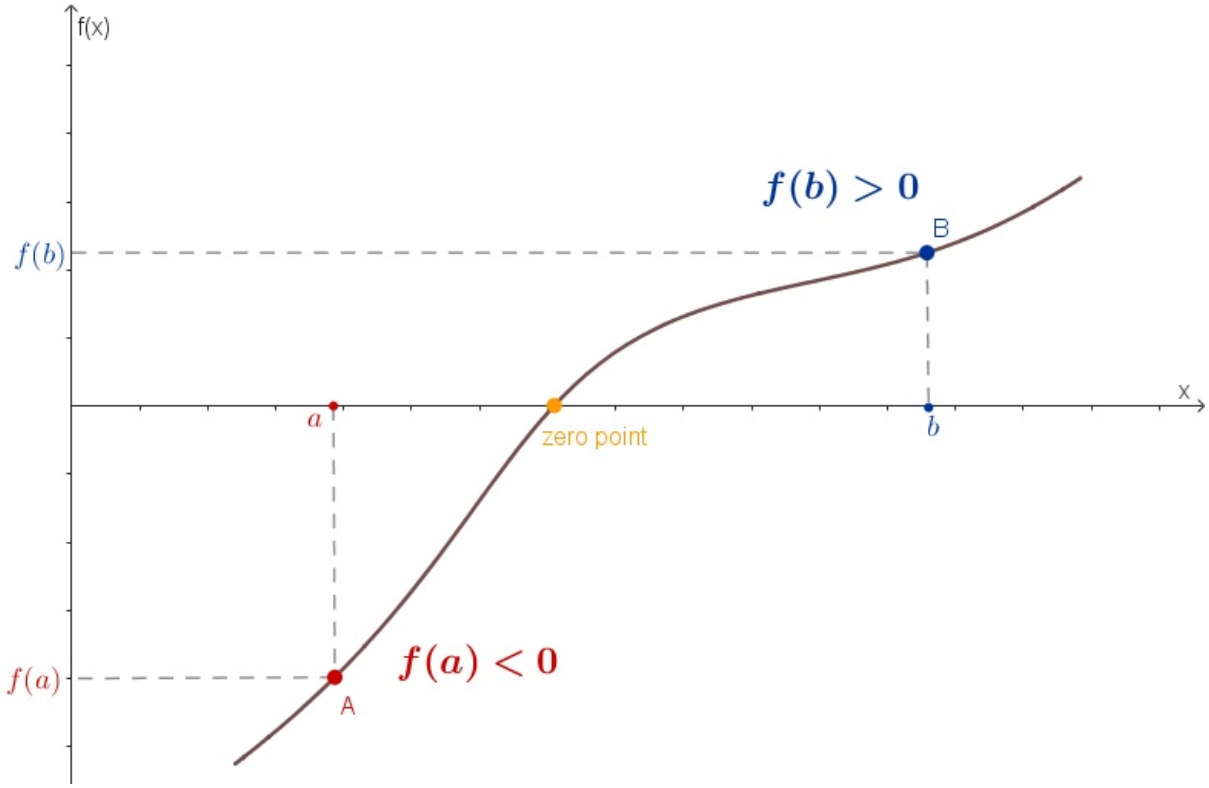


Figure 1.
An interval that bounds the root/zero point for some arbitrary function.

When we have found the interval $[a, b]$ in which the root of the function $f(x)$ is located, we can apply the bisection method by doing the following:

1. Cut the interval $[a, b]$ in half, where the point $c = \frac{a+b}{2}$ is the middle of that interval. If $f(c) = 0$, then the point c is the root and we are done with the application of the algorithm. Otherwise, there will be a root in one of the intervals $[a, c]$ or $[c, b]$.
2. We find which of those two intervals contains the root by comparing the signs of the function at point c and point a. Information about the relationship between the signs of the function at point c and point a (or point b) will reveal to us through which interval our function passes and with which interval we continue executing algorithm. If $f(c)f(a) > 0$ then both functions have the same prefix and there is no root in the interval $[a, c]$ (because the function does not intersect the abscissa) so we choose the interval $[c, b]$. If $f(c)f(a) < 0$ then the sign changes, that is, the function intersects the abscissa axis in that interval and in that case we choose the interval $[a, c]$. You can see a general example in Figure 2.

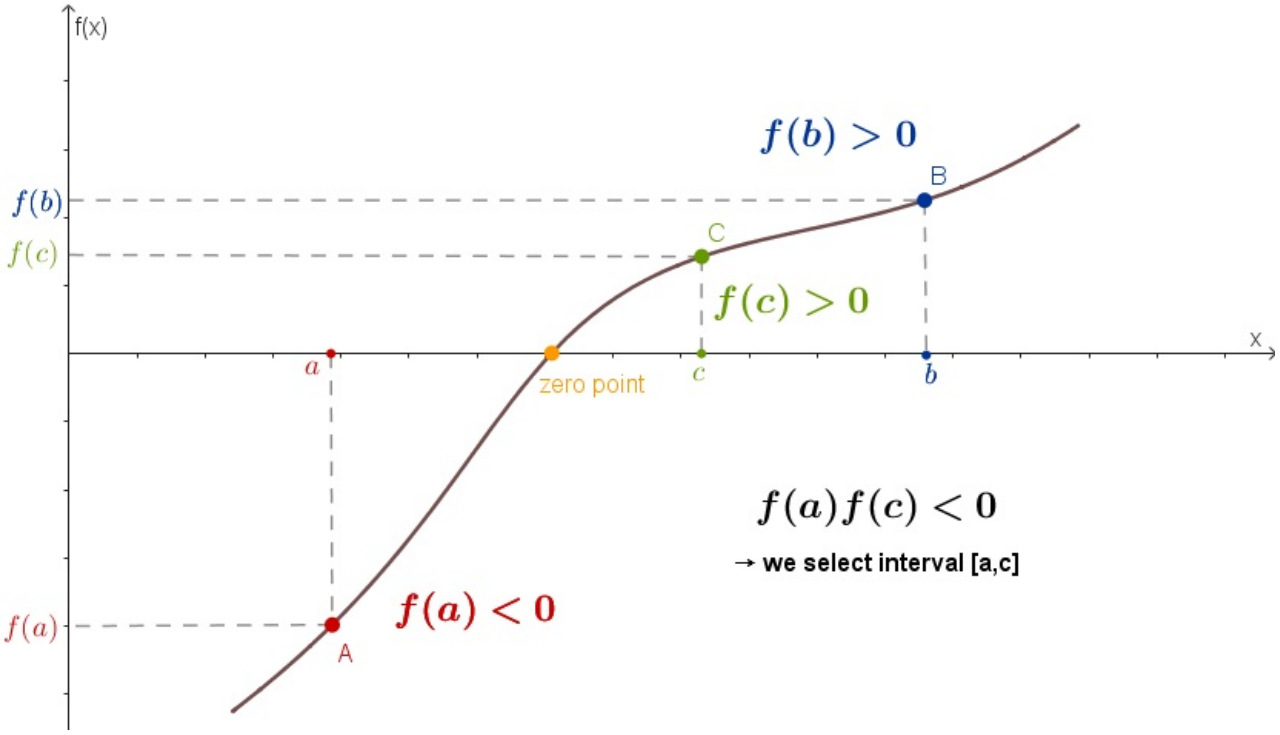


Figure 2.
General idea of determining the interval in which the root is located with the bisection method.

1. We continue the bisection process until we get a sufficiently small interval in which the root of the function is located, that is, until we get an interval that corresponds to the desired precision of determining the root. The interval becomes small enough when the difference between the limits of the interval is less than or equal to the chosen precision, that is, we iterate until $|b - a| > precision$ holds. In each of the iterations again we must know in

which interval our root is located.

2.2.1. How to construct program that executes bisection method?

1. We define a function that receives 3 variables: **a** - the lower limit of the initial interval, **b** - the upper limit of the initial interval and **p** - precision. We are looking for the middle of the initial interval, $c = \frac{a+b}{2}$: if this point is the root ($f(c) = 0$) then the program stops and returns the value of the root, and if it is not, we check the signs of one of the interval (eg whether $f(a)f(c) > 0$ or whether $f(a)f(c) < 0$). If $f(a)f(c) < 0$ then we need to select the interval $[a, c]$, that is, we set $a \rightarrow a$ and $b \rightarrow c$. If that condition does not hold, then we set $a \rightarrow c$ and $b \rightarrow b$. Such iterations are repeated until we satisfy the condition for precision, that is, as long as it holds that $|b - a| > p$. Finally, the function returns a specified root.
2. Let's check if the signs of the function are opposite within the limits of the initial interval, i.e. if it holds that $f(a)f(b) < 0$. If they are not, the algorithm stops, and if they are, then the bisection method is implemented.

TASK 2.1.

Based on the description above write a function that that implements the bisection method. To test how it works, define a known function (for which you know the zero points) inside that same cell.

TASK SOLUTION

```
def bisection (a, b, p):
    step = 1
    condition = abs(b-a)
    while condition > p: #we iterate as long as the condition holds
        c = (a+b)/2.0 #we are looking for the middle of the initial interval
        print('Iteration-%d, c = %0.6f and f(c) = %0.6f' % (step, c, f(c)))
        if f(c) == 0.0: #we check if point c is zero point
            print ('The root of the function is found at the point c=%0.6f and its value is f(c)=%0.6f' %(
c, f(c)))
            break
        elif f(a)*f(c) < 0.0: #if point c is not the root, we check if the root is in the interval a,c
            b = c #if the condition is valid, we set b=c, that is, we select the interval a,c
        else:
            a = c #if the condition is not valid, then we set a=c, that is, we select the interval b,c
        step = step + 1
        condition = abs (b-a) #we update the precision according to the new interval
    print ('\nRoot of the function is: %0.8f' %c)

#input of values for the a, b and p
a = float(input('Lower limit of the initial interval: '))
b = float(input('Upper limit of the initial interval: '))
p = float(input('Precision (entry with decimal point): '))

""" we check whether there is a root in the initial interval: if it is not found, the program prompts us t
o select new  values, and if it is found, then the bisection function (a,b,p) is applied """
#example for the function with known roots
f = x-2

if f(a)*f(b) >= 0.0:
        print ("\nThere is no zero point within that interval!")
        print ("Try again with new values.")
else:
    bisection (a, b, p)
```

Although the bisection method is easy to understand and implement, it always converges to the zero point and is therefore reliable, there are also disadvantages. It always converges to the zero point but the convergence is very slow, that is, many iterations are needed to find the root of the function. And this is only valid if the function is continuous (if there are no interruptions of the function in that interval) and if we apply it at points where the function has opposite signs. Due to this need, we cannot find the root for all functions using the bisection method. For example, the bisection method will not help us if we want to find the root of the function $f(x) = x^2$. Although there are faster and better numerical methods for finding the root of a function, given our limited knowledge of mathematics and for our purposes, the bisection method serves the purpose.

TASK 2.2. - PHYSICS TASK

The player shoots the ball at a speed of 5 m/s at an angle of 40° relative to the ground. At what distance from the player will the ball hit the ground?

```
a. Define the trajectory function of this projectile motion for the conditions given in the problem.
b. Draw a graph of the trajectory of this projectile motion so that you can approximatelly determine the i
nterval in which the zero point of the function is located.
c. Apply the bisection method to find the ball's range.
```

TASK SOLUTION

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

g = 9.81
z = np.linspace (0,5,50)

v0 = float(input("Initial velocity (m/s): "))
alpha = float(input("Angle(°): "))

y = z*np.tan(alpha*np.pi/180)-(z^2 * g)/(2*v0^2*(np.cos(alpha*np.pi/180))^2)

plt.plot(z,y)
plt.xlabel ('x/m')
plt.ylabel ('y/m')
plt.grid(True)
plt.show()

def bisection (a, b, p):
    step = 1
    condition = abs(b-a)
    while condition > p:
        c = (a+b)/2.0
        if f(c) == 0.0:
            break
        elif f(a)*f(c) < 0.0:
            b = c
        else:
            a = c
        step = step + 1
        condition = abs (b-a)
    print ('\nRoot of the function is: %0.8f' %c)

print ('Determening the root of the function!')
a = float(input('Lower limit of the initial interval: '))
b = float(input('Upper limit of the initial interval: '))
p = float(input('Precision (entry with decimal point): '))

f = x*np.tan(alpha*np.pi/180)-(x^2 * g)/(2*v0^2*(np.cos(alpha*np.pi/180))^2)


if f(a)*f(b) >= 0.0:
        print ("\nThere is no zero point within that interval!")
        print ("Try again with new values.")
else:
    bisection (a, b, p)
```

2.3. Secant method

The secant method is a variation of the previously mentioned Newton-Raphson's method, but basic knowledge of mathematics is enough for the implementation of secant method. Just like with the bisection method we begin with two starting points, that is, with the interval in which we think our root is located, but the general idea of the secant method is a little different. One of the advantages of this method, compared to the bisection method, is that the function in the starting points should not have different signs (the selected interval at the beginning should not contain a zero point).

After selecting the initial interval, we look for the secant of the function at those two points (the line that intersects the function at two points). The principle is shown geometrically in the following Figures 3 and 4. Figure 3 shows the case when the function has opposite signs at the starting points, and Figure 4 shows the case when the function has the same signs at the starting points. In both cases, the secant intersects the abscissa at some point c.

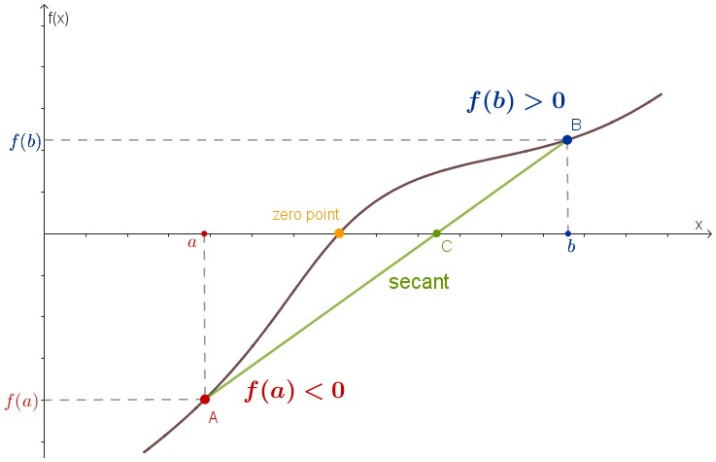


Figure 3.

Determining the secant in the case when the function has opposite signs at the starting points.

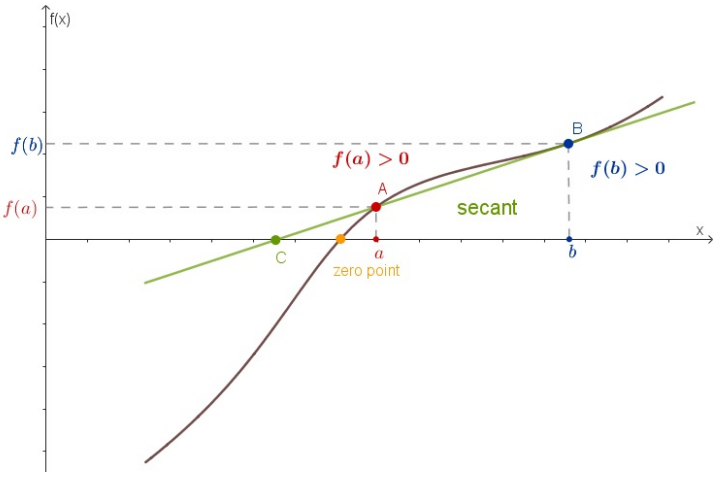


Figure 4.

Determining the secant in the case when the function has the same signs at the starting points.

To determine the point where the secant intersects the abscissa axis, we first need to determine the equation of the secant (the equation of the line passing through the points a and b). By equating the equation of the secant to zero, we get the position of the point c. More precisely, we can calculate the point c as:

$$c = b - \frac{(b-a)f(b)}{f(b) - f(a)}$$

After determining the point where the secant intersects the abscissa axis, we repeat the procedure for that point (eg c) and point b, as shown in Figure 5. The new secant is marked in purple in the picture and we can see that its slope changes so that it approaches more to the slope of the tangent at the zero point of the function.

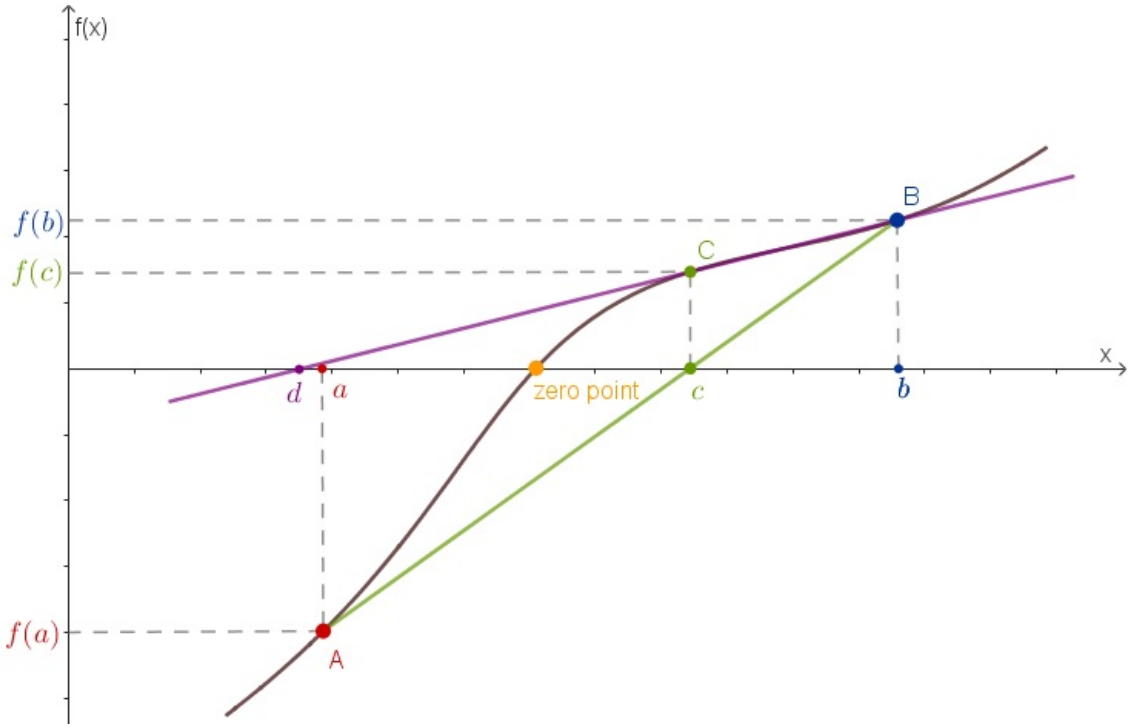


Figure 5.
Searching for the second secant (highlighted in purple) through the new points.

We stop with the iterations, as with the bisection method, when we reach satisfactory precision (as long as the condition $|f(c)| > p$ is satisfied).

Unlike with the bisection method, we have seen here that we do not need to have an initial interval such that the function in the edge values of the interval has opposite signs. Another positive thing about the secant method is that we will arrive at the solution for the zero point after a smaller number of iterations because the convergence to the zero point is faster than with the bisection method. However, the big disadvantage of the secant method is that it can happen that we do not have convergence towards the root of the function and that we have to be careful that the values of the function at the points by which we determine the secant are not equal, because in that case the secant does not intersect the abscissa axis. Nevertheless, even with these disadvantages, the secant method is a good option when determining the root in physical problems where continuous non-oscillating functions mainly appear.

2.3.1. Algorithm for performing the secant method

1. Define f(x).
2. Define input values: lower and upper limits of the interval (a, b), precision (p) and maximum number of steps (N). NOTE: While a, b and p are *float* values, N must be an *integer*.
3. Initialize the step counter: $i = 1$.

4. If $f(a) = f(b)$ print "Error: impossible to divide by zero." and go to (10), otherwise go to (5).
5. Calculate $c = b - \frac{(b-a)f(b)}{f(b)-f(a)}$.
6. Increase the number of steps: $i = i + 1$.
7. If $i \geq N$ print "Does not converge." and go to (10), otherwise go to (8).
8. If $|f(c)| > p$ set $a = b$ and $b = c$ and go to (5), otherwise go to (9).
9. Print the root as c.
10. Stop the program.

TASK 2.3.

Based on the above algorithm, write a function that will determine the root of the function using the secant method. To test the program determine roots of the function whose roots you searched with the bisection method in Task 2.1.

TASK SOLUTION

```
def secant (a, b, p, N):
    i = 1
    condition = True
    while condition > p:
        if f(a)==f(b):
            print ('Error: impossible to divide by zero!')
            break
        c = b -((b-a)*f(b))/(f(b)-f(a))
        a = b
        b = c
        i = i + 1
        if i > N:
            print ('It does not converge!')
            break
        condition = abs (f(c))
    print ('\nRoot of the function is: %0.8f' %c)

#input of values for the a, b, p and N
a = float(input('Lower limit of the initial interval: '))
b = float(input('Upper limit of the initial interval: '))
p = float(input('Precision (entry with decimal point): '))
N = int(input('Maximum number of iterations: '))


f = x-2

if f(a)==f(b):
    print ('Error: impossible to divide by zero!')
    print ('Try again with new values.')
else:
    secant(a, b, p, N)
```

As you could see here, if the function has more than one root you can't find them all at once with bisection and secant method but you have to search for them one by one and for each determine different starting interval. In cases like that and with more complex functions it is easier and faster to use Python built-in root finding functions. The function that is used for root finding is *fsolve* for the *scipy.optimize*.

TASK 2.4.

Explore the documentation for Python's built-in function *fsolve* and use it to find the roots of the function $f(x) = x^3 + 2 * x - 1$.

TASK SOLUTION

```
import numpy as np
from scipy.optimize import fsolve
def func(x):
    return x^3 +2*x -1
root = fsolve(func,0)
root
```

Chapter 4: Numerical derivation

When analyzing a graphic representation in physical problems we often have to determine the tangent of a function at a certain point which brings us a lot of useful information when analyzing the problem. In the x-t graph, the slope of the tangent gives us information about the speed of the body or in the v-t graph about the acceleration of the body at a certain moment. These are perhaps the most well-known examples from physics where knowing the slope of

a tangent is useful for analyzing a problem, but they are not the only ones. Generally, from the slope of the tangent it's possible to find out the rate of change of the displayed function in relation to the change of the independent variable. Of course, this is not only applicable in physics, but in almost all natural sciences as well as some social sciences such as economics. The solution to the problem that is geometrically brought to us by the tangent of the function at some point of the function, analytically represents the reduction of the interval in which we observe some change in the function until the moment when that interval is small enough to determine the value of the function at some point. One of such examples in physics is the determination of the instantaneous speed and instantaneous acceleration of a body during its motion, which we can determine geometrically through the tangent function at a particular moment of motion. One general example of determining the tangent at point A of the function is shown in Figure 1. **Derivations** are those that enable us to determine the equation of the tangent and its value at the observed point of the function.

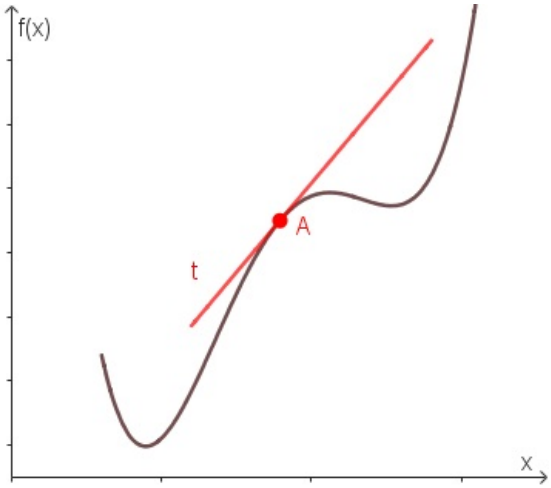


Figure 1.

The line t is the tangent of the function at the point A, and the derivative of the function corresponds to the slope of the line t at that point.

Although we currently do not have sufficient mathematical knowledge to determine the derivatives of functions, they can be analytically determined for many functions. However, there are still functions where the analytical determination of the derivative is not simple or not possible. This is where numerical methods are used again, so in this chapter we will describe the determination of the derivative of a function at a point.

4.1. Finite difference method

The term derivative of a function was simultaneously and independently arrived at by two scientists: Isaac Newton and Gottfied Wilhelm Leibniz. Newton studied the velocity of a body at a given moment, and Leibniz dealt with the issue of the tangent of a function at a certain point. We have already mentioned that the derivative represents the rate of change of the function in relation to the change of the independent variable, that is, geometrically, the derivative of the function at a point can be interpreted as the slope of the tangent at that point of the function. The simplest method for numericala determination of the derivative at a point works on the same principle on which Leibniz came up with the concept of derivative, by observing the secant of the function (Figure 2) and reducing the interval between the two points where the secant intersects the graph of the function. In Figure 3, we can see that by decreasing the observed interval Δx, the slope of the secant approaches the slope of the tangent at the point A.

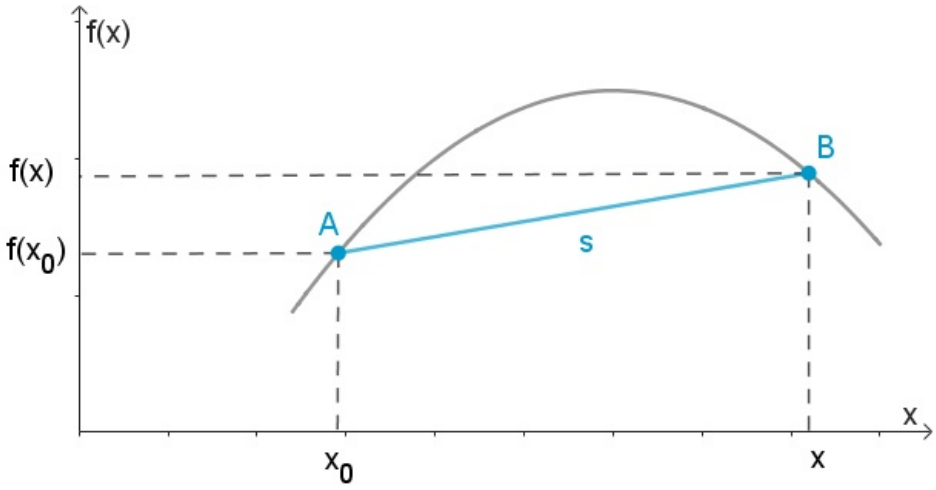


Figure 2.

The secant intersects the shown function at the points A and B.

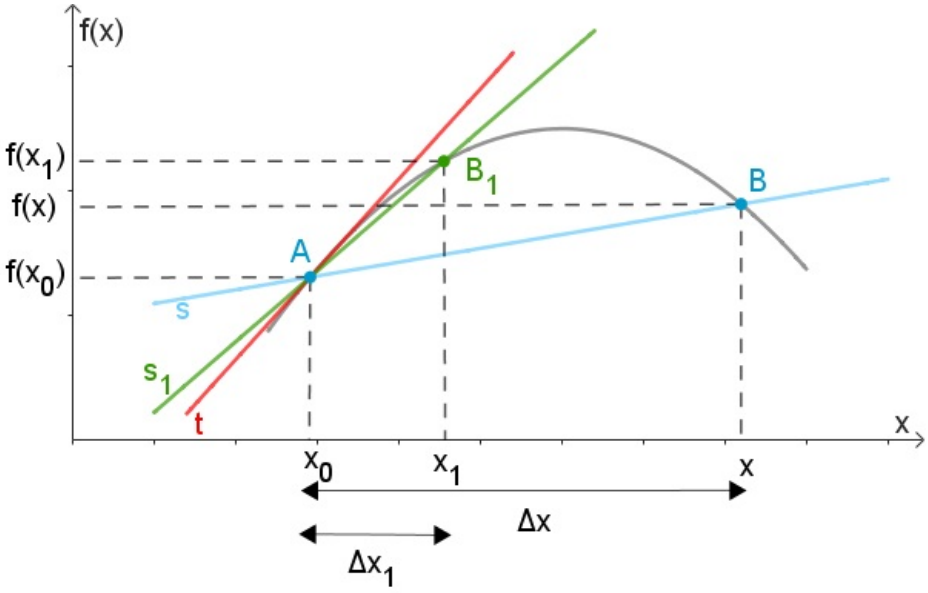


Figure 3.
By decreasing the interval Δx, the slope of the secant approaches the slope of the tangent at the point A.

The slope of the secant can be determined numerically as:

$$k_s = \frac{f(x) - f(x_0)}{x - x_0},$$

where $\Delta x = x - x_0$ represents **change of variable**, and $f(x) - f(x_0)$ **change of functional value of variable**. By summarizing the change of the variable, i.e. when $\Delta x \to 0$, the secant turns into a tangent at the point x_0 as shown in Figure 3. Based on this, the derivative of the function $f'(x)$ at some point x_0 can be written as

$$f'(x_0) = \lim_{\Delta x \to 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}.$$

Based on the previous expression, it is evident that for sufficiently small distances between points, that is, sufficiently small Δx, the slope of the secant k_s is a sufficiently good approximation of the derivative of the function. If the chosen step Δx is too small, a rounding error appears - when subtracting terms in the denominator, they are canceled and we get zero in the denominator. On the other hand, if the step Δx is too large, the slope of the secant will not be a good enough approximation of the derivative (slope of the tangent). The described method of approximating the derivative of a function at a point is called the **finite difference method**.

For simpler analysis we will observe the derivative of the function at the point a where the length of the interval between the two observed points is h. That is, when we set $x_0 \to a$ and $\Delta x \to h$ with the assumption that $h > 0$ the expression for the slope of the secant k_s that we'll use to approximate the derivative $f'(a)$ becomes:

$$f'(a) \approx \frac{f(a + h) - f(a)}{h}.$$

The described substitution of variables is shown graphically in Figure 4. This expression represents the formula for **forward difference approximation**. The name comes from the fact that, according to Figure 4, the surrounding point is chosen to the right of the point where we are looking for the derivative of the function (in this case it is the point $(a, f(a))$).

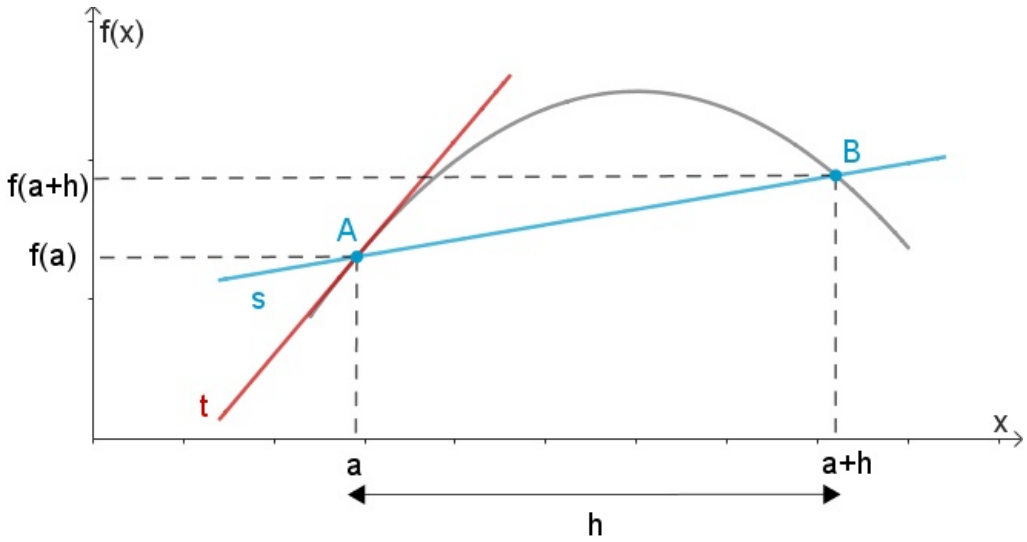


Figure 4.
By decreasing the distance between the two points, h, the slope of the shading approaches the slope of the tangent at point A.

We can arrive at a similar expression as for the forward difference method if we choose the surrounding point to the left of the point where we want to determine the derivative. In this case, the coordinates of the surrounding point are $(a-h, f(a-h))$, so the expression for approximating the derivative is given as follows

$$f'(a) \approx \frac{f(a) - f(a-h)}{h}$$

and represents the formula for ***backward difference***.

Figure 5 shows the graphical interpretation of the formulas for approximation by forward difference (marked in blue) and backward difference (marked in green). The slope of the line marked as *tangent* (red) corresponds to the exact value of the derivative at the point A. The slope of the line t_-, which passes through the points A and C, corresponds to the formula for approximating the derivative by backward difference and the slope of the line t_+, which passes through the points A and B, corresponds to the formula for derivative approximation by forward difference. Also, we can see in the figure that we will get a better approximation of the slope of the tangent at point A if we observe the slope of the line through points B and C (marked orange in Figure 5). The slope of that line, that is, the approximation of the derivative at point A, can be determined as

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h}$$

and that expression represents the formula for ***central difference***. This formula gives a better approximation of the derivative because it uses three points (the point $(a, f(a))$ and one point on each side of it).

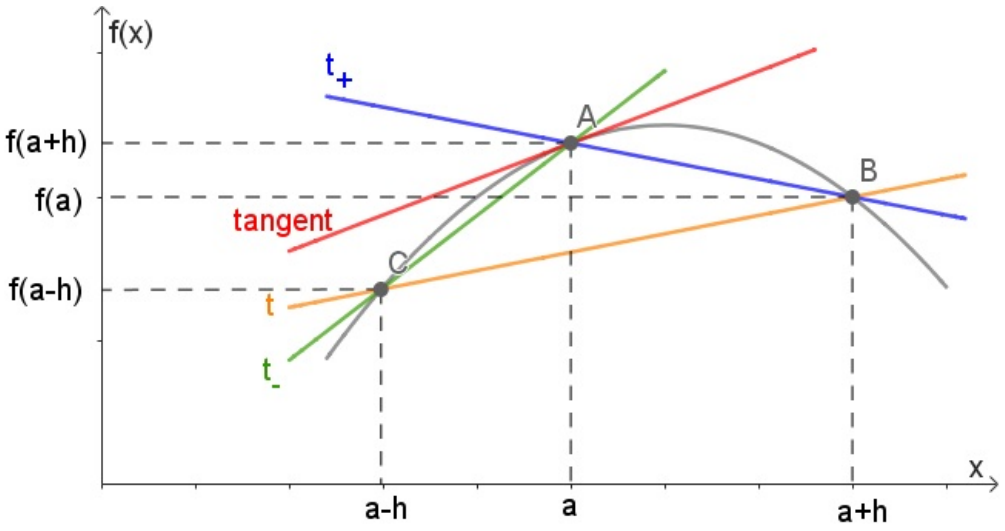


Figure 5.
Graphic interpretation of the formula for approximating the derivative by forward (blue), backward (green) and central (orange) differences.

TASK 4.1.

1. Write a function to perform the finite difference method for approximating the derivative where you can choose which method to use to determine the derivative (forward, backward or central).
2. Find a built-in *Python* function that approximates the derivative of a function at a given point.
3. Apply the finite difference method to calculate the approximation of the derivative and compare with the actual value of the derivative at that point and the value obtained via Python's built-in function. For application use function $f(x) = x^2 + 2x$ whose derivative is given by the function $f'(x) = 2x + 2$. Which method gives the best approximation of the derivative, and through which method does *Python*'s derivative function calculate the approximation of the derivative?

TASK SOLUTION

```
from scipy.misc import derivative

def f(x):
    return x**2 + 2*x

def h(x): #the function h(x) represents the derivative of the function f(x)
    return 2*x + 2

def der (f, a, method ='central', h = 0.01):
    if method == 'central':
        return (f(a + h) - f(a - h))/(2*h)
    elif method == 'forward':
        return (f(a + h) - f(a))/h
    elif method == 'backward':
        return (f(a) - f(a - h))/h
    else:
        raise ValueError("Method must be 'central', 'forward' or 'backward'.")


print(der (f, 1)) #the derivative value obtained using the written function der
print(derivative (f, 1, dx = 0.01)) #derivative value obtained using a function built into the scipy.misc
module
print(h (1.0))
```

TASK 4.2.

1. The position values of some body at a certain moment in time are saved in the file `accelerated_data.csv` . Using that data write a program that will determine the derivatives of the position in dependence on time for each given point.

NOTE: To be able to determine the derivatives, you must load the data from the file into a list and convert it into two lists: one containing the time values (eg the `t` list) and the other containing the position values (eg the `x` list).

1. Plot obtained values as a function of time. What physical quantity is represented by the derivative of the position of the body in time?

TASK SOLUTION

```
import csv
import matplotlib.pyplot as plt

with open('accelerated_data.csv') as file:
    data = list(csv.reader(file))

for a in data:
    no_elements = len (a)

t = [float(data[0][i]) for i in range(no_elements)]
x = [float(data[1][i]) for i in range(no_elements)]

derivative = []
first = (x[1]-x[0])/(t[1]-t[0])
derivative.append(first)


for i in range(1, len(x)-1):
    f_deriv = (x[i+1] - x[i-1])/(t[i+1] - t[i-1])
    derivative.append(f_deriv)
    i += 1

last = (x[len(x)-1]-x[len(x)-2])/(t[len(x)-1]-t[len(x)-2])
derivative.append (last)

print(derivative)

plt.scatter(t, derivative, s = 10)
plt.title('Derivative of the position as function of time')
plt.xlabel('t [s]')
plt.ylabel('dx/dt = v [m/s]')
plt.show()
```

Chapter 5: Numerical integration

Very often, when analyzing graphical representations some additional information can be found from the area under the curve that represents the dependence of the two quantities. Some of the most famous examples in physics are the determination of the area in $v-t$ graph where the value of the surface corresponds to the distance traveled or the displacement of the body (Figure 1a) or in $F-x$ graph where the surface represents the work W (that is, the change in body energy ΔE) as swown in Figure 1b. In cases where the dependence of two quantities is simple (eg linear) or can be divided into simple dependencies in certain intervals, determining the area under the curve is not a problem - the calculation comes down to elementary knowledge of mathematics.

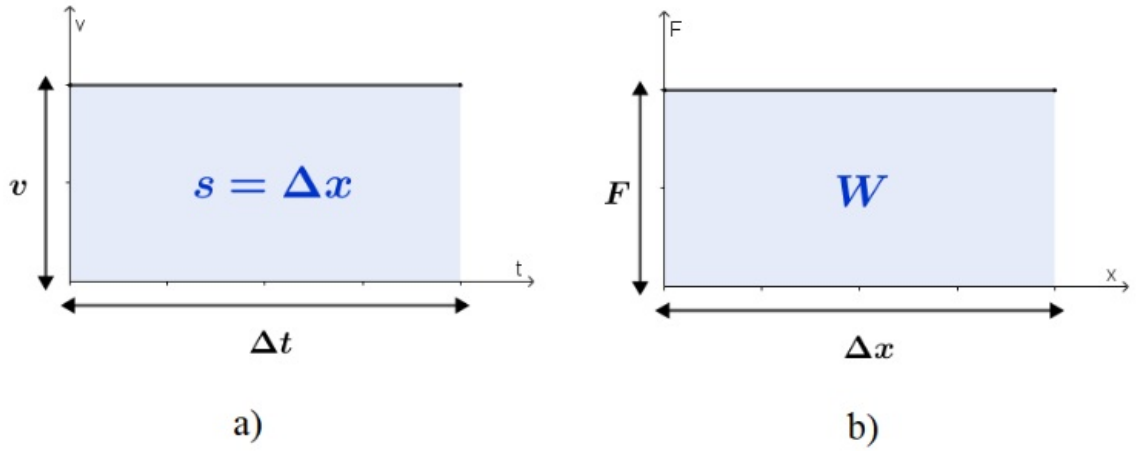


Figure 1.
a) The area between the curve and the abscissa axis in $v-t$ graph represents the traveled path/displacement of the body and b) the area between the curve and the abscissa in $F-x$ graph represents the work done on the body.

But what if we cannot divide the area under the curve into simple shapes whose area we know how to calculate? Let's say, as in the case shown in Figure 2. The answer to this problem is given by **integrals**.

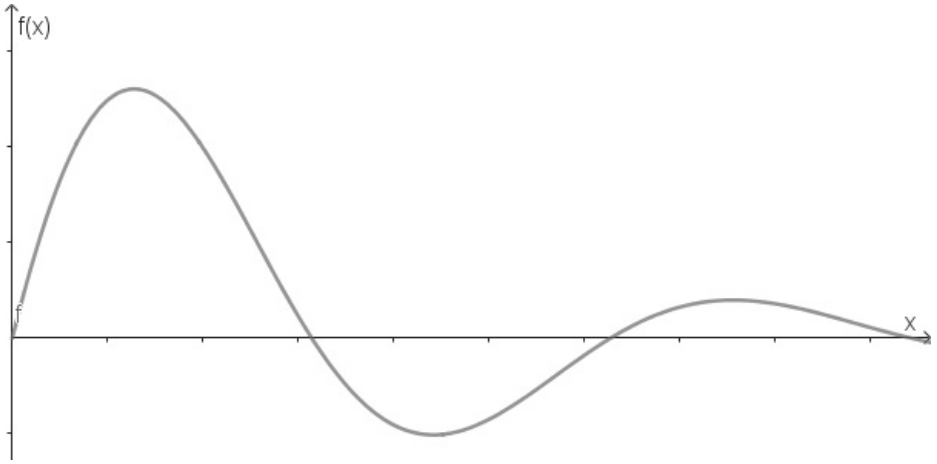


Figure 2.
Graphic representation of the function where we can't easily determine surface under the curve.

The idea of integration was shaped by Isaac Newton and Gottfried Wilhelm Leibniz at the end of the 17th century. The term integration has two meanings. Integrating, in the simplest terms, is finding the surface of some bounded function, which we achieve by calculating the so-called **definite integral**. A definite integral of the form $$\int_{a}^{b} f(x)dx \$$ represents the surface bounded by the function $f(x)$, the x-axis and the lines $x=a$ and $x=b$ (limits of the definite integral). The described area for the function $sin(x)$ is shown in Figure 3. We can imagine finding such a specific integral as dividing the required area into rectangles (or some other simpler shapes) - by adding up their individual areas, we get the total area we are looking for. The smaller the width of the individual rectangles (sections), the closer the sum of the areas of the sections is to the required area as shown in Figure 4.

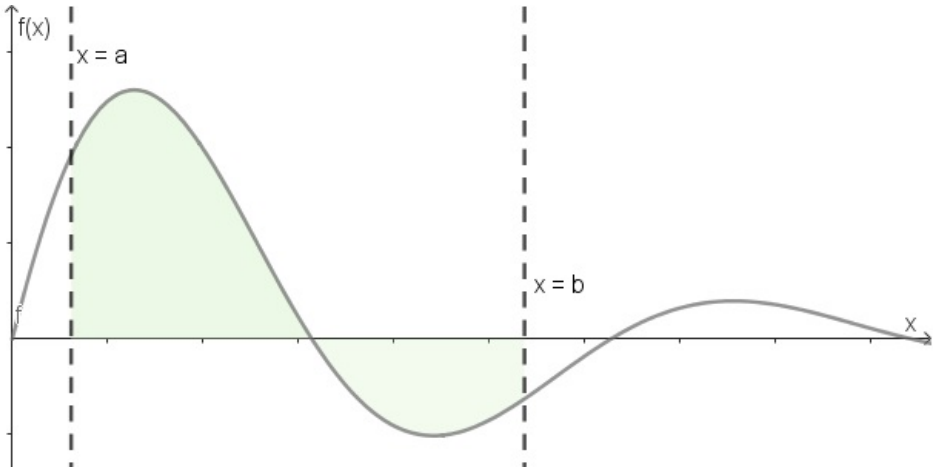


Figure 3.
Graphic representation of the function $sin(x)$ and the surface in the interval [a,b].

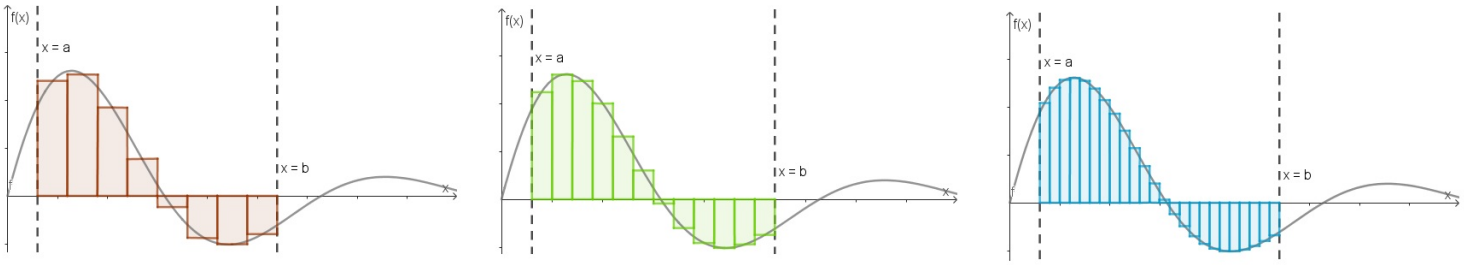


Figure 4.
Graphic representations that show approaching the required surface by summing the surfaces of increasingly narrow sections.

Another meaning of determining the integral is to find the *antiderivative* of a function $f(x)$, i.e. its *primitive function* of the form $F(x)$. In that case, we calculate the **indefinite integral** (no limits of integration are defined, but a general function $F(x)$ is obtained, by derivation of which we obtain the initial function $f(x)$)

$$F(x) = \int f(x)dx.$$

In some cases, it is difficult or even impossible to analytically find the antiderivative of a function, so we cannot even determine a specific integral, that is, calculate a bounded surface. In these cases, we use the numerical integration for such definite integrals.

5.1. Numerical integration methods

There are several methods of numerical integration:

1. *midpoint rule*
2. *trapezoidal rule*
3. *Simpson's rule*

The midpoint rule is the simplest of the mentioned methods. Within this method, we approximate the area between the curve of the function $f(x)$ and x-axis by summing the areas of rectangles of equal width whose central points are also the points located on the function $f(x)$. This central points determine the height of the rectangle as shown in Figure 6.

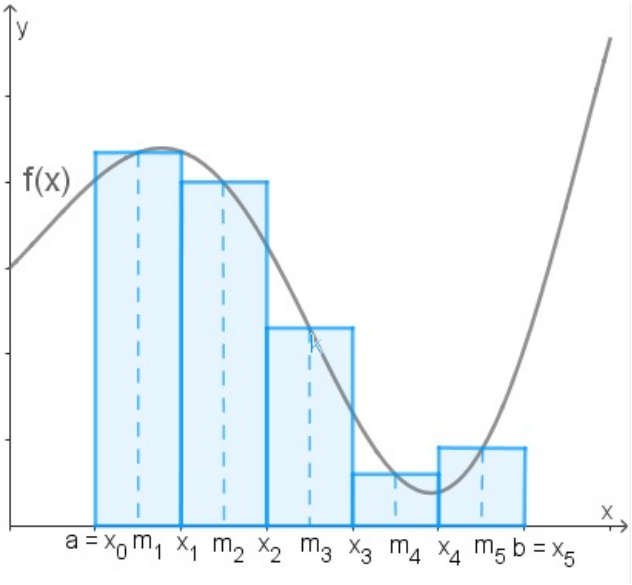


Figure 6.
The principle of the midpoint rule for approximating area under the curve given by function $f(x)$.

When approximating the integral for data obtained by measurement, where the functional dependence of two quantities is not known, it's not possible to use the midpoint rule because in that case we can't determine the "height of the rectangle". Instead of rectangles, we can use trapezoids to approximate the surface at equal subintervals (shown in Figure 7) which is applied within the trapezoidal rule. This method is similar to the midpoint rule approximation, but in order to reach the desired precision with the midpoint rule, we need a large number of rectangles, while the trapezoidal approximation achieves the same precision faster. An interesting presentation of the differences in these two methods can be found at the following link (https://www.whitman.edu/mathematics/calculus_online/section08.06.html).

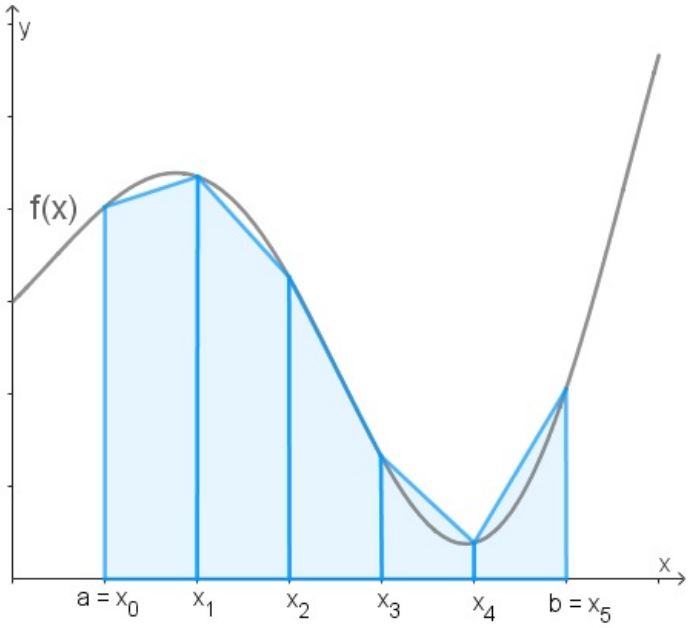


Figure 7.
The principle of the trapezoidal rule for approximating area under the curve given by function $f(x)$.

If we look at Figure 8, we can see that the trapezoidal rule overestimates the value of the integral on the intervals where the function is concave and underestimates the value on the intervals where the function is convex. On the other hand, with the midpoint rule, these errors are "averaged" because the overestimation and underestimation of the value of the integral occurs on the same parts of the function. This is precisely one of the reasons why, in general, the midpoint rule is more accurate than the trapezoidal rule when approximating the value of integral.

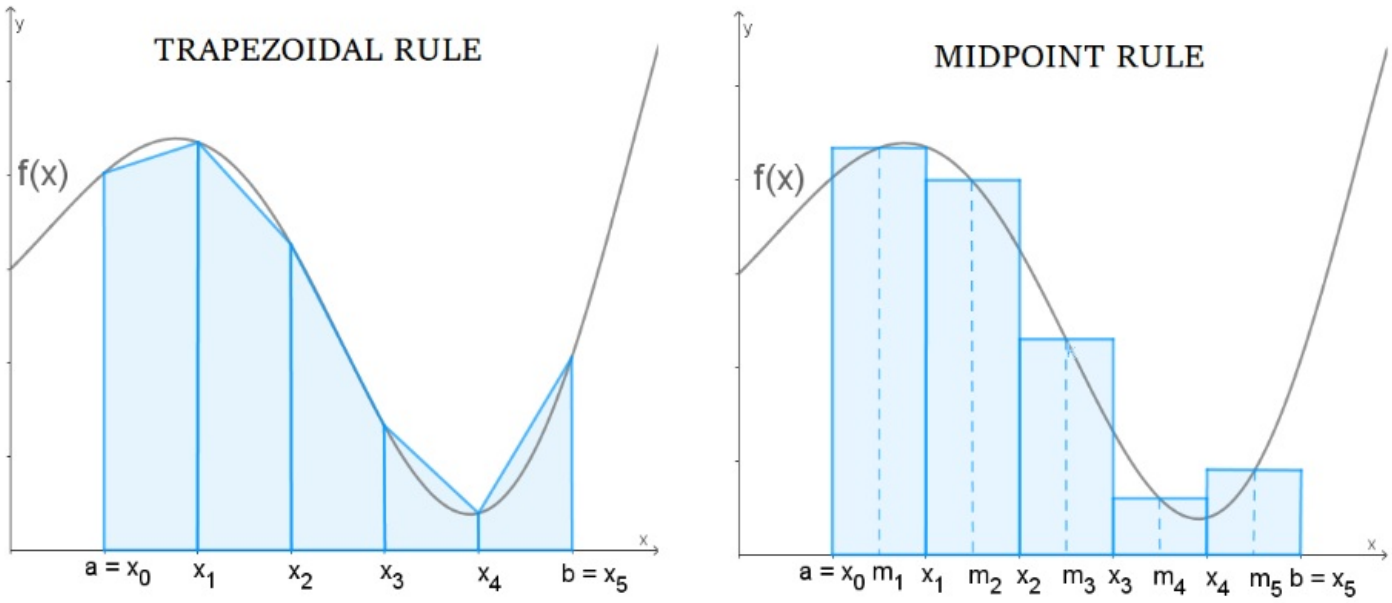


Figure 8.
A comparison of the trapezoidal rule (left) and the midpoint rule (right) showing why the midpoint rule gives us greater accuracy than the trapezoidal rule.

With the midpoint and the trapezoidal rule, the integral is approximated by parts of the function using a constant function (midpoint rule) or a linear function (trapezoidal rule). Simpson's rule for approximating the value of the integral uses a polynomial of the second degree, i.e. a quadratic function. We divide the interval within which we want to determine the integral into an even number of subintervals of equal width. The integral of the function $f(x)$ on the first pair of subintervals is approximated by the integral of the polynomial $p(x) = Ax^2 + Bx + C$ which passes through the points $(x_0, f(x_0))$, $(x_1, f(x_1))$ and $(x_2, f(x_2))$. On the next pair of subintervals, we approximate the integral by the integral of the new polynomial p_2 that passes through the next three points. The procedure is repeated for each subsequent pair of subintervals. The principle on which Simpson's rule works is shown in Figure 9. In Figure 9 you can also see how we achieve the desired precision even faster than with midpoint or trapezoidal rule.

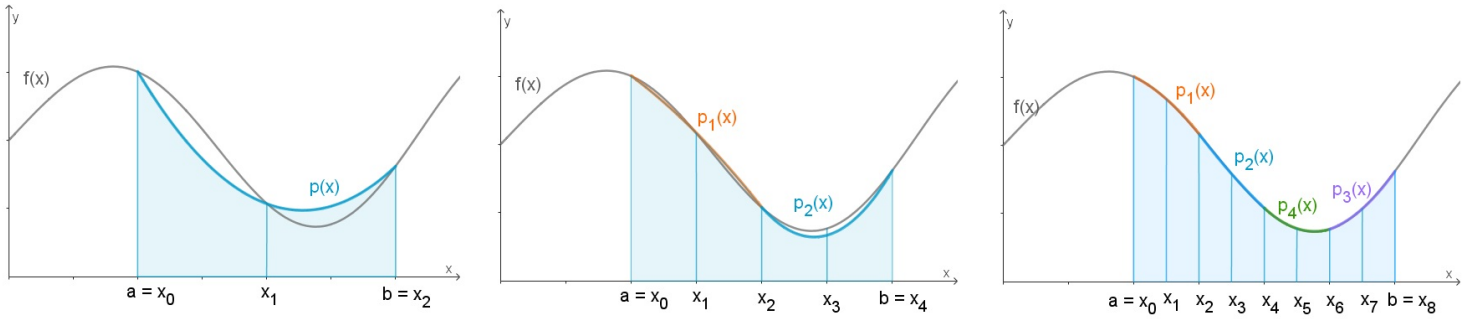


Figure 9.
The principle of the Simpson's rule for approximating area under the curve given by function $f(x)$.

5.2. How to approximate integrals using the described methods?

Each of these approximation methods uses subintervals of equal width h into which the region of integration, the interval $[a, b]$, is divided. To get the width of each subinterval, we must divide the width of the interval $[a, b]$ into the desired number of intervals $(n - 1)$, where n represents the number of points by which we divide the interval. Of course, the condition that $b > a$ must be satisfied.

$$h = \frac{b - a}{n - 1}$$

5.2.1. Midpoint rule

In the midpoint rule, the width of the rectangle is given as $h = \frac{b-a}{n-1}$, as described earlier, and the height of the rectangle for each subinterval is determined with the value of the function at the point that is the midpoint of the subinterval boundary (as shown in Figure 6). Generally, if we have a subinterval $[x_i, x_{i+1}]$, the center point of that subinterval is given as:

$$y_i = \frac{x_{i+1} + xi}{2}$$

. Of course, the area of the rectangle is then determined as the product of width and height for each individual subinterval. By summing the areas of the obtained rectangles, we finally get an approximation of a definite integral, that is

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} hf(y_i)$$

.

5.2.2. Trapezoidal rule

As with the midpoint rule, the width of the subinterval (the width of each trapezoid) is h. In this case, it is necessary to determine the area of each trapezoid and sum all these areas in order to obtain an approximation of the integral. If we look at one of the trapezoids in Figure 9, we can see that its surface can be divided into the surface of a rectangle and a right triangle, as shown in Figure 10. The following applies for the surfaces of a rectangle and a right triangle:

$$P_{rectangle} = hf(x_i),$$

$$P_{triangle} = \frac{h(f(x_{i+1}) - f(x_i))}{2}.$$

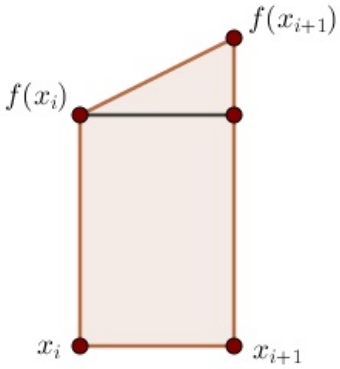


Figure 10.
Representation of a trapezoid that can be divided into a rectangle and a right triangle.

Looking at it this way, the area of all trapezoids is determined as $P = \sum_{i=0}^{n-1}(P_{rectangle} + P_{triangle})$. This expression can be sorted, so we finally get

$$\int_a^b f(x)\,dx \approx \frac{h}{2}[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)]$$

.

The points x_0 and x_n actually represent the limits of the definite integral a and b.

5.2.3. Simpson's rule

The last method approximates the definite integral using a polynomial of the second degree (which is graphically a parabola). In this case, the approximation is done through three points, i.e. through pairs of subintervals. To determine this approximation, we need a little more complex math, so we'll just state it in its final form:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}[f(x_0) + 4\sum_{i=1,\,odd}^{n-1} f(x_i) + 2\sum_{i=2,\,even}^{n-2} f(x_i) + f(x_n)]$$

.

With this method, it is very important that there is an even number of subintervals (an odd number of points) by which the interval $[a, b]$ is divided. As in the previous method, it holds that $x_0 = a$ and $x_n = b$.

TASK 5.1.

1. Define a function that calculates the integral of some function $f(x)$ using the midpoint rule. Let a function have input parameters a, b and n, where a and b represent the lower and upper limits of integration, and n is the number of points by which we divide the region of integration into intervals.
2. Calculate the integral of the function $f(x) = sin(x)$ in the interval $[0, \pi]$.
3. Compare the obtained value with the exact value of the integral within that interval: $\int_0^{\pi} sin(x)\,dx = 2$.

TASK SOLUTION

```
import numpy as np

def f(x):
    return np.sin(x)

def int_mid(a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1) #n is the number of points, and the number of intervals is (n-1)
    sum_der = 0
    i = 0
    for i in range (n-1):
        sum_der = sum_der + h*f((x[i+1]+x[i])/2)
    print(sum_der)

int_mid(0, np.pi, 11)
```

TASK 5.2.

1. Define a function that calculates the integral of a function $f(x)$ using the trapezoidal rule. Let a function have input parameters a, b and n, where a and b represent the lower and upper limits of integration, and n is the number of points by which we divide the region of integration into intervals.
2. Calculate the integral of the function $f(x) = sin(x)$ in the interval $[0, \pi]$.
3. You calculate the value of that same integral via Python's built-in function *quad* within the package ***scipy.integrate***. Compare the obtained value with your value.

TASK SOLUTION

```
import numpy as np
import scipy.integrate as integrate

def f(x):
    return np.sin(x)

def int_trap(a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1)
    sum_der = 0
    i = 1
    for i in range (n-1):
        sum_der = sum_der + h/2 * (f(a) + 2*f(x[i]) + f(b))
    print (sum_der)

int_trap(0, np.pi, 11)
print(integrate.quad(f,0,np.pi))
```

TASK 5.3.

1. Define a function that calculates the integral of a function $f(x)$ using Simpson's rule. Let a function have input parameters a, b and n, where a and b represent the lower and upper limits of integration, and n is the number of points by which we divide the region of integration into intervals.
2. Calculate the integral of the function $f(x) = sin(x)$ in the interval $[0, \pi]$.
3. You calculate the value of that same integral via Python's built-in function *quad* within the package ***scipy.integrate***. Compare the obtained value with your value.

TASK SOLUTION

```
import numpy as np
import scipy.integrate as integrate

def f(x):
    return np.sin(x)

def int_simp (a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1) # n must be odd for the number of intervals to be even - the condition for this rule!
    sum_even = 0
    sum_odd = 0
    i = 1
    for i in range (n-1):
        if i % 2 == 0:
            sum_even = sum_even + 2*f(x[i])
        else:
            sum_odd = sum_odd + 4*f(x[i])
        sum_all =  h/3 *(f(x[0]) + sum_odd + sum_even + f(x[n-1]))
    print (sum_all)


int_simp (0, np.pi, 100)
integrate.quad(f,0,np.pi)
```

Physics problem: Determination of the potential for a linear charge distribution

The charge is distributed on the rod of length l so that the linear charge density (that is, the charge per unit length) is given as

$$\lambda(x) = \frac{Q}{l} e^{-\frac{x^2}{l^2}}.$$

For the specified charge density, it is necessary to determine the electric potential at point T at the height y exactly above the halfway point of the rod. For the sake of simplicity, we can set the stick so that its halfway point is at the origin of the coordinate system, as shown in Figure 11.

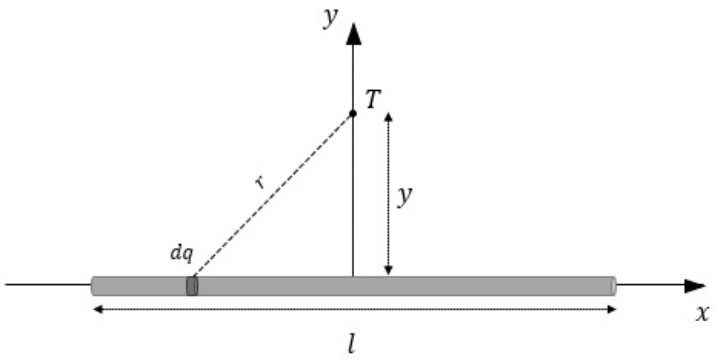


Figure 11.
Problem setup shown in the coordinate system.

We know that the potential of a point charge depends on the amount of charge that creates that potential q and on the distance of the observed point of space from the charge r, or

$$V(r) = \frac{1}{4\pi\varepsilon_0}\frac{q}{r}.$$

In our case, we do not have just one point charge, but many of them distributed along the rod (which is determined by the linear charge density $\lambda(x)$), where each charge on the rod contributes to the total potential. We can say it another way, some small fraction of the charge dq in the point T creates a potential $dV = \frac{1}{4\pi\varepsilon_0}\frac{dq}{r}$. The symbols dq and dV represent *differentials* of charge and potential, i.e. their arbitrarily small sizes. So, the total potential at point T will be equal to the sum of all those arbitrarily small potentials, which leads us to the intergal

$$V = \int_a^b dV = \int_a^b \frac{1}{4\pi\varepsilon_0}\frac{dq}{r}$$

where the limits of the intergal a and b represent the left and right ends of the stick, i.e. $a = -\frac{l}{2}$ and $b = \frac{l}{2}$ because we are integrating over x coordinates. To see how the potential changes depending on the x coordinate, we can write the charge differential over the linear charge density. Given that we know that in general it holds for the linear charge density $\lambda(x) = \frac{Q}{x}$, we can see that accordingly $Q = \lambda(x)x$ or in our case where we observe charge differential $dq = \lambda(x)dx$. We also know that the potential depends on the distance of the observed point from the charge that creates it, and in our case the distance changes as we move along the stick. More precisely, the distance depends on the x coordinate, which we can see from Figure 11. where the distance r can be obtained using the Pythagorean theorem, where it is valid that $r = \sqrt{x^2 + y^2}$. When we fit all this, we get the integral that needs to be calculated in order to determine the potential at point T,

$$V = \int_{-l/2}^{l/2} \frac{1}{4\pi\varepsilon_0}\frac{\lambda(x)}{\sqrt{x^2+y^2}}\,dx$$

$$V = \int_{-l/2}^{l/2} \frac{1}{4\pi\varepsilon_0}\frac{\frac{Q}{l}e^{-\frac{x^2}{l^2}}}{\sqrt{x^2+y^2}}\,dx$$

$$V = \frac{1}{4\pi\varepsilon_0}\frac{Q}{l}\int_{-l/2}^{l/2} \frac{e^{-\frac{x^2}{l^2}}}{\sqrt{x^2+y^2}}\,dx.$$

The presented intergal is not easy to calculate analytically, nor do we have enough mathematical knowledge to be able to calculate it analytically, but we can calculate it numerically under certain conditions (knowing the values of the constants ε_0, l, Q and knowing the distance y from the rod on which we want to determine the value of the potential).

TASK 5.4.

Determine numerical value of the electric potential (either via Python's built-in function or via a function you wrote yourself) at the point T located at the height at the point

$$y = 4m$$

above the halfway point of the rod for the physics problem shown above for the linear charge density of the form $\lambda(x) = \frac{Q}{l}e^{-\frac{x^2}{l^2}}$ knowing the values:

$$l = 2m$$

$$\frac{Q}{4\pi\varepsilon_0 l} = 1V.$$

TASK SOLUTION

```
import math
import scipy.integrate as integrate

y = 4
l = 2
factor = 1

def f(x):
    return (math.exp(-(x*x)/(l*l)))/(math.sqrt(x*x+y*y))

value, error = integrate.quad(f,-l/2,l/2)

potential = factor * value

print(potential)
print(error)
```

TASK 5. 5. (not so easy task)

Determine numerical value of the electric potential at point T for the previously presented physical problem of determining the electric potential for a linear charge density of the form $\lambda(x) = \frac{Q}{l}e^{-\frac{x^2}{l^2}}$ knowing the values:

$$l = 2m$$

$$\frac{Q}{4\pi\varepsilon_0 l} = 1V,$$

but in the interval 1 m $< y < 10$ m. Save the obtained values in a list and graphically display the dependence of the potential on the distance y from the middle of the stick.

TASK SOLUTION

```
import math
import scipy.integrate as integrate

l = 2
factor = 1
y0 = 1
yn = 10
potential =[]
distance =[]

for y in range (y0,yn+1,1):
    distance.append(y)

def f(x):
    return (math.exp(-(x*x)/(l*l)))/(math.sqrt(x*x+y*y))


for y in range (y0,yn+1,1):
    value, error =  integrate.quad(f,-l/2,l/2)
    pot = factor * value
    potential.append(pot)

print (distance)
print (potential)

V_y = list(zip(distance,potential))
list_plot(V_y, color = 'blue', title = 'Potential dependence on the distance from the rod', axes_labels =
['$y/m$','$V/V$'], marker = "s")
```

Task 5.6.

The force $F(x) = F_0 - kx$ acts in the $+x$ direction on a body of mass m located on a frictionless horizontal surface. The box was initially at rest at position x_0. How much work was done on the box if the force acted on the body until the moment when it reached position x?

1. Think about how to determine the work done on the box using numerical integration (write down the expression for the work in the form of the work differential according to the example with the charge and determining the potential) and write a program that calculates the work in general for this problem.
2. Calculate the work for the following constant values: $F_0 = 18$ N, $k = 0.53$ N/m, $m = 6$ kg, $x_0 = 0$ m and $x = 14$ m.

TASK SOLUTION

```
import numpy as np
import scipy.integrate as integrate

F0 = float(input("For F(x)=F0-kx, enter the value of F0 in [N]:"))
k = float(input("For F(x)=F0-kx enter the value of k in [N/m]:"))


def f(x):
    return F0-k*x

def int_simp (a,b,n):
    x = np.linspace (a, b, n)
    h = (b-a)/(n-1)
    sum_even = 0
    sum_odd = 0
    i = 1
    for i in range (n-1):
        if i % 2 == 0:
            sum_even = sum_even + 2*f(x[i])
        else:
            sum_odd = sum_odd + 4*f(x[i])
        sum_all =  h/3 *(f(x[0]) + sum_odd + sum_even + f(x[n-1]))
    print (sum_all)


int_simp (0, 14, 10000)
integrate.quad(f,0,14)
```

Chapter 7: Numerical methods used for linear regression and least square method

In science, we often encounter a large amount of data that needs to be analyzed. The basis of analyzing a dataset involves searching for dependencies between observed variables - whether one variable depends on another and what is the type of tehir dependence. This dependence is often unknown and difficult to directly observe from experimental data due to dispersion of data as a result of random errors. Therefore, it is necessary to use statistical data processing to select the dependency that best fits the given set of measurement data. The determination of the functional relationship between two or more variables is examined through regression analysis. By using regression analysis, based on the established correlation and known values of the independent variable, we can predict the value of the dependent variable - more precisely, it provides us with a relationship between the dependent and independent variables. Such analysis is often used for predicting the behavior of certain phenomena, as well as for inferring causal relationships between the dependent and independent variables. Depending on the number of variables whose dependence we are examining, regression analysis can be divided into simple and multiple regression. Regarding simple regression analysis, it involves analyzing the dependence of one dependent variable on an independent variable and with multiple regression we can analyze relationship between independent variable and several dependant variables. It can also be further divided into linear and nonlinear regression analysis (Figures 1 and 2 illustrate linear and nonlinear relationships between two variables).

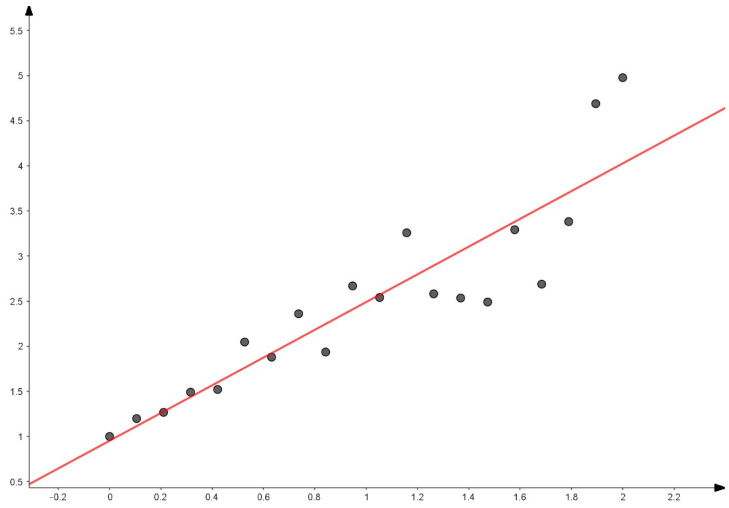


Figure 1.

Linear relationship between two variables.

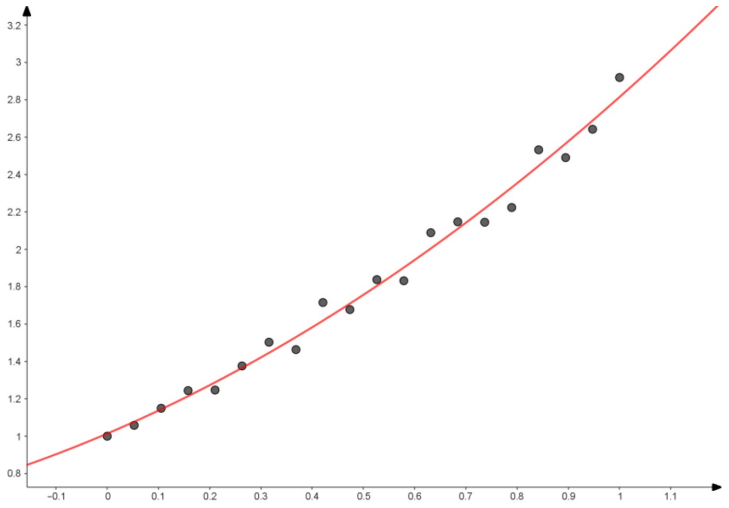


Figure 2.

Nonlinear relationship between two variables.

The most well-known methods of regression analysis are **linear regression** and the **least square method**. The simplest of this methods is simple linear regression which examines the influence of an independent variable (x) on the dependent variable (y). In this case, we seek a line of the form $y = a_0 + a_1 x + u$ that best fits the given set of experimental data, or in other words, we look for the most likely values of the slope coefficients a_0 and a_1 such that the line, with the smallest deviation, passes through the experimental data. The term u represents the random error, which indicates the difference between the experimentally measured values of the independent variable and the theoretical prediction (Figure 3), known as the *residual*. In the least square method, the function that best fits the experimental data is determined by minimizing the sum of the squares of the differences between the measured and calculated values. In other words, it determines the function whose curve approaches the given points as closely as possible. This method can be applied for both linear and nonlinear relationships between two variables.

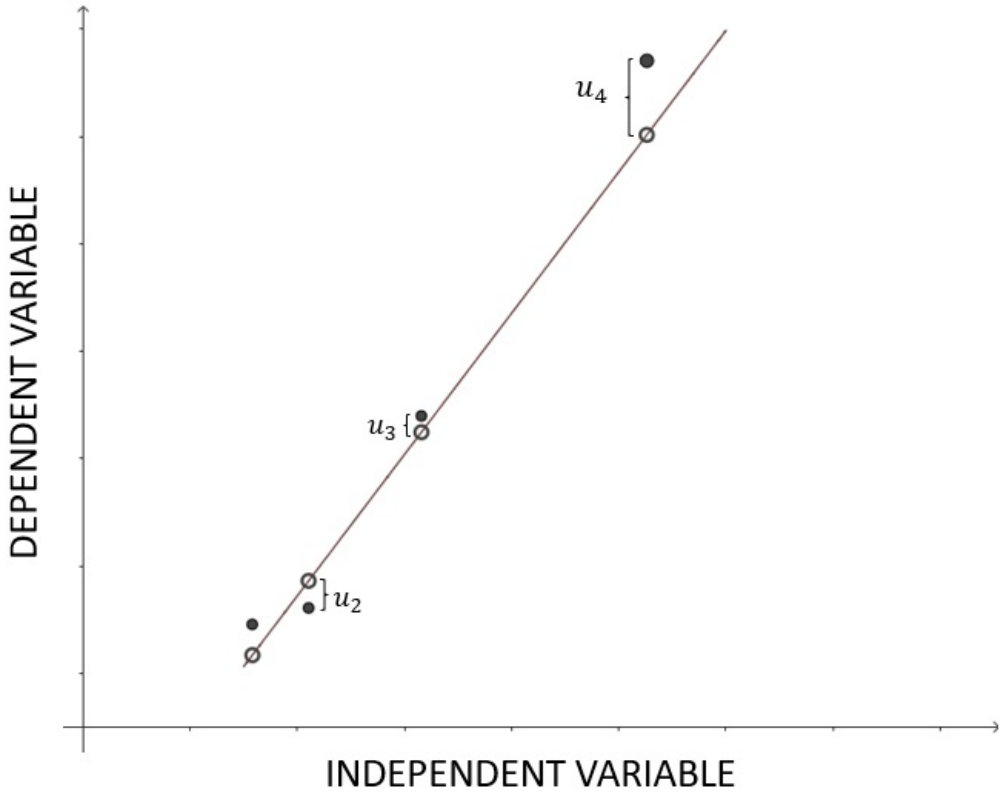


Figure 3.
The principle of simple linear regression.

With the method of least squares, as with linear regression, we look for the most likely values of the coefficients of the function such that the obtained curve approaches as close as possible to the experimentally determined values. The simplest form of the function for which the least square method is applied is the linear function of the form $y = a_0 + a_1 x$ and in that case analytical solutions for the coefficients a_0 and a_1 and their errors M_{a_0} and M_{a_1} can be obtained. We obtain the coefficients and their errors using the following expressions:

$$a_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$a_1 = \frac{n \cdot \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$M_{a_0} = \sqrt{\frac{1}{n-2}\left[\frac{n \cdot \sum_{i=1}^{n} y_i^2 - \left(\sum_{i=1}^{n} y_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} - a_1^2\right]}$$

$$M_{a_1} = M_{a_0} \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$$

Task 7.1.

For the data set defined at the end of this task, write a function `estimate` whose input variables will be x and y. The function should calculate the coefficients a_0 and a_1 and their errors M_{a_0} and M_{a_1} using the previously mentioned expressions. Plot values of x and y (as points) and on the same plot show the line whose coefficients were obtained by the least square method. Label the axes correctly and set the legend.

```
import numpy as np

x = np.linspace(0, 2, 20)
y = x + x * np.random.random(len(x))
```

TASK SOLUTION

```
import math

def estimate(x,y):
    n = len(x)
    sum_x=np.sum(x)
    sum_y=np.sum(y)
    sum_x2=np.sum(x*x)
    sum_xy=np.sum(x*y)
    sum_y2=np.sum(y*y)

    a0=(sum_x2 * sum_y - sum_x * sum_xy)/(n*sum_x2-sum_x * sum_x)
    a1=(n*sum_xy - sum_x * sum_y)/(n*sum_x2-sum_x * sum_x)
    Ma1=math.sqrt(1/(n-2)*((n*sum_y2-(sum_y)^2)/(n*sum_x2-(sum_x)^2) - a1^2))
    Ma0=Ma1*math.sqrt(1/n * sum_x2)

    return a0,a1,Ma0,Ma1

import matplotlib.pyplot as plt

a0 = estimate(x,y)[0]
a1 = estimate(x,y)[1]

plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.', label='Data')
plt.plot(x, a0 + a1*x, 'r', label='Least squeare method fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="upper left")
plt.show()
```

These expressions are already quite complicated, and they become even more complicated in the case of polynomials of the third degree (where the highest power of the variable is 3). For some higher powers of polynomials, as well as for many functions, it is not possible to find an analytical solution for determining the coefficients and their errors. In this case, we rely on numerical solutions and numerical methods for determining the coefficients. *Python*'s built-in `optimize.curve_fit()` function, which is built into the `scipy` library, helps us with this. It uses the least squeare method to determine the coefficients of any function (linear or non-linear).

Task 7.2.

Look online for the documentation for the function `optimize.curve_fit()` and study how it is used. Apply it to the previous problem with the same defined points and plot this solution as well.

TASK SOLUTION

```
import numpy as np
from scipy import optimize
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 20)
y = 1 + x + x * np.random.random(len(x))

def f(x, a1, a0):
    y = a0 + a1*x
    return y

coef, err = optimize.curve_fit(f, xdata = x, ydata = y)


plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.', label='Data')
plt.plot(x, coef[0]*x + coef[1], 'r', label='Least squeare method fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="upper left")
plt.show()
```

Task 7.3.

Load the *method1.csv* file attached to this manual and save the data to lists. Plot data and assess what kind of dependence this could be and use the least square method to determine the coefficients for the assumed form of the function. Finally, show the function obtained by the least squares method on the same graph as the data. Add the names on the margins and the legend.

TASK SOLUTION

```
import csv

with open('method1.csv') as f:
    m = list (csv.reader(f))
    for a in m:
        x = [float(m[0][i]) for i in range (len(a))]
        y = [float(m[1][i]) for i in range (len(a))]
    f.close()

def func(x, a, b, c):
    y = a*np.exp(b*x) + c
    return y

coef, err = optimize.curve_fit(func, xdata = x, ydata = y)


plt.figure(figsize = (10,8))
plt.plot(x, y, 'b.', label = 'Data')
plt.plot(x, coef[0]*np.exp(coef[1]*np.array(x)) + coef[2], 'r', label='Least square method fit')
plt.xlabel('x')
plt.ylabel('y')
plt.legend(loc="upper left")
plt.show()
```

Chapter 8: Numerical solution of systems of linear equations

When solving mathematical problems, we often have to solve a system of (usually two, sometimes three, practically never four or more) linear equations. Sometimes, the system of (two) quations is so straightforward that one can guess the correct solutions just by looking at the equations. More often, it takes a simple calculation to arrive to a solution. We will briefly summarize some general guidelines and well-known methods for solving systems of linear equations and give some examples.

Introduction

It is important to always keep in mind a few basic principles. Firstly, in order to get a specific non-parameterized solution, we need as many equations as there are unknown variables. For example, if the problem asks for values of x and y, we need two equations; if the problem asks for a, b and c, there have to be three equations to work with and so on. In fact, in the examples we just gave, we should have specified another condition, namely that the equations have to be linearly independent, meaning "different in a significant way", if we use informal language. For example, equations $4x - 3y = 7$ and $8x - 6y = 14$ are obviously dependent and as such insufficient to get x and y. Informally, the second equation does not carry any additional information about unknown variables.

Secondly, there are three possible outcomes when solving a system of linear equations - a solution may not exist, there may be infinitely many solutions or there is a single solution. In case of a single solution, the solution is of course not a number but an ordered set of numbers. In an example with two variables and two equations, the solution is a pair of numbers (x, y). Geometrically, this pair can be interpreted as the intersection point of two lines in a plane, the lines are in this case given by the original equations. Geomatrical interpretation of a two-variable system that does not have a solution is that the lines given by the equations are parallel and therefore do not have an intersection. If the equations give the same line, the system is said to have infinitely many solutions, as every point on the line is evidently a solution. If there are three unknown variables, we can visualise the equations as planes in 3D space, the intersection of two planes that are not parallel is a line, and the intersection of this line with the third plane (again, if they are not parallel) is a point in 3D space, which is represented by a set of its coordinates (x, y, z). Obviously, the order is important, as $(x, y) \neq (y, x)$ and similarly in higher dimensions. With four or more variables, the principle remains the same, eventhough we can no longer visualise the equations as geometric objects.

Thirdly, solving the systems usually follows the same generally applicable steps that we will summarize shortly. With some practice, one can solve small systems (two or three variables) with "nice" coefficients relatively quickly. With larger systems (or "difficult" coefficients), however, working with pencil and paper can take quite some time and requires a great deal of attention. A system of ten equations, for example, is practically unsolvable on paper. In practice, some problems require solving huge systems of linear equations, sometimes containing hundreds or thousands of variables and equations. In this case, the answer is to use computers, rewrite the systems in matrix form, which we will explain in more detail later, and use one of many numerical methods available.

Methods for solving systems of linear equations with pen and paper

Substitution method - two variables

Since we are dealing with linear equations, it is always possible to find an explicit expression for at least one variable. To clarify, if both equations contain both variables, it is easy to see that either one of the variables can be expressed. However, it could happen that one of the equations (or both!) contain only one variable - in this case, it is obviously not possible to express both variables. When we have selected a variable and an equation that will be rearranged in such a way that the selected variable is "isolated", we substitute this variable in the other equation with the expression we got from the first equation (see examples). In the two-variables case, this is relatively straightforward, although sometimes students struggle with this simple step when dealing with three-variable systems. The substitution we have just described results in obtaining only one equation with one variable that we solve as we normally would. In the end, if there is a solution to this equation, we must use it to obtain the value of the other variable. Only the pair of both variables is the solution to the system, as discussed above.

Elimination method - two variables

Solving the system via elimination method is closer to solving the system in matrix form. For this method, we choose a variable that appears in both of the equations and multiply both sides of one or both of the equations in such a way that we are left with two equations (linearly dependant on the original two, as they differ only in the multiplication factor) that have the same coefficient related to the chosen variable (again, see example). When we have obtained such a pair of equations, we substract one from the other, resulting in the "disappearance" of the chosen variable and leaving us with only one variable in one equation. Alternatively, it is sometimes easier for the students to add equations (as opposed to substracting them), in this case, we multiply both sides of the original equations in such a way that we are left with one positive and one negative coefficient (both having the same absolute value of course) related to the chosen variable. When adding the equations, such coefficients again result in eliminating one of the variables. As mentioned in the substitution method, we must not forget to obtain a pair of values, as only such a pair is a true solution to the given system (if the system is uniquely solvable).

Substitution method - more than two variables

Solving two-variable systems is the foundation on which we build to solve bigger systems with pen and paper. When dealing with multiple variables, we syistematically reduce the number of variables and equations by one in each step. For the first step, we choose a variable and express it from one of the equations. Then we substitute the chosen variable in all of the other equations besides the one we have originally chosen with the expression we obtained. After the substitution (and some calculations) we have reduced the number of variables by one (as our originally chosen variable does not appear in any of the equations any mor) and the number of equations by one (the first equation is no longer needed as it "served its purpose" by providing the substitution). We repeat the process as many times as necessary (as many times as there are different variables) until we obtain only one equation with one variable. If it is solvable, we obtain its value. Then we begin the work in the reverse order of substitutions and one by one we obtain the values for the other variables. The variable that was chosen as first to be substituted, is the last to have its value explicity computed. The solution to the system is an ordered set of such values.

Elimination method - more than two variables

Again, the goal is to slowly reduce the number of variables and equations. As always, we begin by selecting an equation and a variable. We then multiply all of the equations in such a way that we get the same (or, alternatively, the same in absolute value and differenf in its sign) coefficients related to the chosen variable. We than substract (alternatively, add) equations in pairs - the selected equation and one other equation, then the selected equation (the same as before!) and another equation etc. We alway substract from the same equation. As discussed before, this results in a smaller system as we have again eliminated one of the variables and also have one less equation to work with. We repeat the process until the system is solved.

Example in two variables

SUBSTITUTION:

ELIMINATION:

GRAPH:

Example in three variables

SUBSTITUTION:

ELIMINATION:

GRAPH:

TASK

Solve the following systems of equations:

a) [has solution] b) [no solution] c) [has solution] d) [infinitely many solutions]

Rewriting systems as matrices