Project 1

Exercise project

Bacteria Data Analysis

Introduction

In this project you will develop a computer program for handling data related to growthrates of different bacteria at different temperatures. You must implement the functions and main script described in the following according to the specifications.

Data load function

Interface

```
def dataLoad(filename):
    # Insert your code here
    return data
```

Input arguments — `filename`: A string containing the filename of a data file.

Output arguments — `data`: An $N \times 3$ matrix.

User input — No.

Screen output — Yes (error message, see specifications below.)

Description — The function must read the data in the data file `filename`. Each line in the data file consists of the following fields:

`Temperature`, `Growth rate`, and `Bacteria`.

The fields contain numeric values and are separated by a space character. The following illustrates an example excerpt of such a data file:

```
25  0.109  1
20  0.096  2
15  0.517  3
35  1.086  4
40  0.934  2
35  0.109  1
      ⋮
```

The `Bacteria` field is a numeric code that correspond to one of the following bacteria names:

`Bacteria`	Name
1	Salmonella enterica
2	Bacillus cereus
3	Listeria
4	Brochothrix thermosphacta

The data in the file must be stored in an $N \times 3$ matrix called `data`, where N is the number of valid rows in the data file.

Handling data errors — There might be one or more erroneous lines in the data file which the function must handle. If the data load function detects an erroneous line in the data file it must skip this line, output an error message to the screen, and continue with the next line. The function must only return the data from the the valid rows. The error message must explain in which line the error occurred and what the error was. The function should check for the following:

- The `Temperature` must be a number between 10 and 60.
- The `Growth rate` must be a positive number.
- The `Bacteria` must be one of the four mentioned in the table above.

Data statistics function

Interface

```
def dataStatistics(data, statistic):
    # Insert your code here
    return result
```

Input arguments

data: An $N \times 3$ matrix with columns Temperature, Growth rate, and Bacteria.
statistic: A string specifying the statistic that should be calculated.

Output arguments

result: A scalar containing the calculated statistic.

User input

No.

Screen output

No.

Description

This function must calculate one of several possible statistics based on the data. A "statistic" here denotes a single number which describes an aspect of the data, as for example a mean (average) value. The statistic that must be calculated depends on the value of the string statistic. The following table shows the differeninterfacet possible values of statistic and a description of how to calculate the corresponding statistic.

statistic	Description
'Mean Temperature'	Mean (average) Temperature.
'Mean Growth rate'	Mean (average) Growth rate.
'Std Temperature'	Standard deviation of Temperature.
'Std Growth rate'	Standard deviation of Growth rate.
'Rows'	The total number of rows in the data.
'Mean Cold Growth rate'	Mean (average) Growth rate when Temperature is less than 20 degrees.
'Mean Hot Growth rate'	Mean (average) Growth rate when Temperature is greater than 50 degrees.

You are encouraged to use suitable built-in functions to compute the statistics where possible.

Data plot function

Interface

```
def dataPlot(data):
    # Insert your code here
```

Input arguments

data: An $N \times 3$ matrix with columns Temperature, Growth rate, and Bacteria.

Output arguments

No.

User input

No.

Screen output

Yes (plots, see specifications below.)

Description

This function must display two plots:

1. "Number of bacteria": A bar plot of the number of each of the different types of Bacteria in the data.
2. "Growth rate by temperature": A plot with the Temperature on the x-axis and the Growth rate on the y-axis. The x-axis must go from 10 to 60 degrees, and the y-axis must start from 0. The plot should contain a single axis with four graphs, one for each type of Bacteria. The different graphs must be distinguished using e.g. different colors, markers, or line styles.

The plots should include a suitable title, descriptive axis labels, and a data legend where appropriate. You are allowed to present the plots in separate figure windows or as sub-plots in a single figure window.

Main script

Interface Must be implemented as a script.

Input arguments No.

Output arguments No.

User input Yes (see specifications below.)

Screen output Yes (see specifications below.)

Description The user of the data analysis program interacts with the program through the main script. When the user runs the main script he/she must have at least the following options:

1. Load data.
2. Filter data.
3. Display statistics.
4. Generate plots.
5. Quit.

The user must be allowed to perform these actions (see specifications below) in any order as long as he/she chooses, until he/she decides to quit the program. The details of how the main script is designed and implemented is for you to determine. It is a requirement that the program is interactive, and you should strive to make it user friendly by providing sensible dialogue options. Consider what you would expect if you were to use such a script, and what you think would be fun. Play around and make a cool script.

Error handling You must test that all input provided by the user are valid. If the user gives invalid input, you must provide informative feedback to the user and allow the user to provide the correct input.

It must not be possible to display statistics or generate plots before any data has been loaded. If the user attempts to do this, he/she should be given appropriate feedback stating that this is not possible.

1. Load data If the user chooses to load data, you must ask the user to input the filename of a data file. Remember to check if the file name is valid. Load in the data using the `dataLoad` function which will display information about any erroneous lines in the data file.

2. Filter data If the user chooses to filter data, he/she must be able to specify one or more conditions which must be satisfied in order for the data rows to be included in the calculation of statistics and generation of plots. As a minimum requirement the user must be able to specify two types of filters:

1. A filter for the Bacteria type, for example `Bacteria` =*Listeria*.
2. A range filter for the Growth rate, for example $0.5 \leq$ `Growth rate` ≤ 1.

Having specified such a filter, statistics and plots should be generated only for the subset of data rows where the condition is met. The user should also be able to disable the filter conditions, and when he/she does this, subsequently the statistics and plots should again be generated for the whole data. If a filter is active, information about the filter should at all times be visible in the user interface.

3. Display statistics If the user chooses to display statistics, you must ask the user which statistic to display. Use the `dataStatistics` function to compute the desired statistic and display it on the screen along with a description of the statistic. If a filter is active, the statistics must be computed only for data rows that satisfy the filter conditions.

4. Generate plots	If the user chooses to generate plots, call the `dataPlot` function to display the plots. If a filter is active, the plots must be generated based only on data rows that satisfy the filter conditions.
5. Quit	If the user chooses to quit the program, the main script should stop.
Data file	You are encouraged to make your own test data file in order to validate that your program functions correctly. It is important that you also ensure and document that your program works correctly in case of errors in the data file as described in section .

Project 2

Exam project

Lindenmayer systems

Introduction

In this project you must develop, test, and document a program for computing and plotting Lindenmayer systems. You must implement the functions and main script described in the following according to the specifications.

Lindenmayer systems A Lindenmayer system is defined iteratively, and it consists of: a) an alphabet of symbols which can be used to create strings, b) an initial string used to begin the iterative construction and c) replacement rules that specify how to replace selected symbols of the string by strings of symbols (from the same alphabet). Originally, these Lindenmayer systems were used to describe the behaviour of plant cells and to model the growth processes of plant development. In this exercise you will work with two systems called the Koch curve and the Sierpinski triangle.

Koch curve The Koch curve can be generated using a) an alphabet consisting of the symbols S, L and R, b) the initial string 'S' and c) the replacement rules

$$\begin{array}{lcl} \texttt{S} & \rightarrow & \texttt{SLSRSLS} \\ \texttt{L} & \rightarrow & \texttt{L} \\ \texttt{R} & \rightarrow & \texttt{R} \end{array}$$

The initial string is `S`. After the first iteration one obtains the string `SLSRSLS`. After the second iteration one obtains the string `SLSRSLSLSLSRSLSRSLSRSLSLSLSRSLS`. The first three iterations are visualized using turtle graphics (see below) in Fig. 2D.1.

Sierpinski triangle The Sierpinski triangle can be generated using a) an alphabet consisting of the symbols A, B, L and R, b) the initial string `A` and c) the replacement rules for each step of the iteration.

$$\begin{array}{lcl} \texttt{A} & \rightarrow & \texttt{BRARB} \\ \texttt{B} & \rightarrow & \texttt{ALBLA} \\ \texttt{L} & \rightarrow & \texttt{L} \\ \texttt{R} & \rightarrow & \texttt{R} \end{array}$$

The initial string is `A`. After the first iteration one obtains the string `BRARB`. After the second iteration one obtains the string `ALBLARBRARBRALBLA`. The inititalization and the result after iteration 4 and 8 is visualized using turtle graphics (see below) in Fig. 2D.2.

Visualization using turtle graphics The sequence of symbols in the string will be visualized graphically by translating each symbol in a command for a so-called turtle graphics: `S`, `A`, and `B` is interpreted as the line segment from an initial location (we will use the origin of the planar coordinate system for this) drawn along an initial direction — we use for this the canonical basis vector $(1,0)^T$. `L` is interpreted as turning left with the angle $\frac{1}{3}\pi$ and `R` as turning right with the angle $-\frac{2}{3}\pi$ (Koch) or $-\frac{1}{3}\pi$ (Sierpinski). The length of the line segment is scaled by a factor $\frac{1}{3}$ (Koch) or $\frac{1}{2}$ (Sierpinski) after each iteration step.

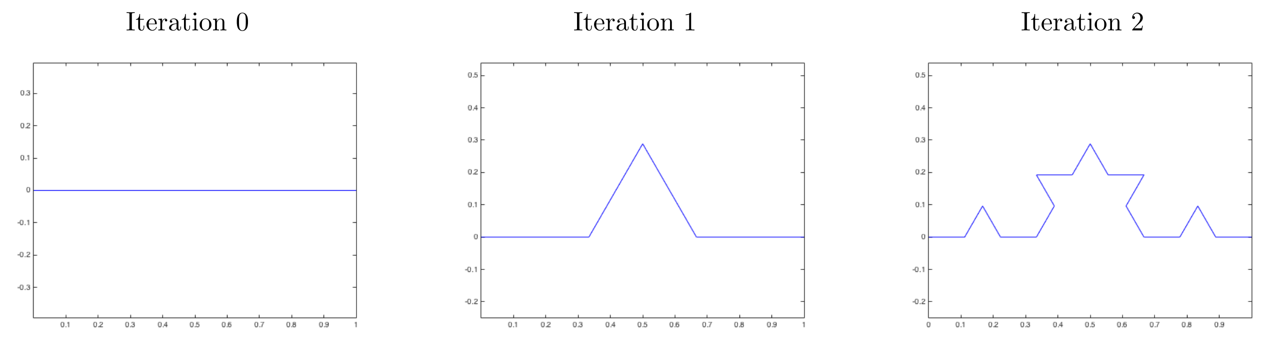

Figure 2D.1: Graphical representation of the first iterations of the Koch curve.

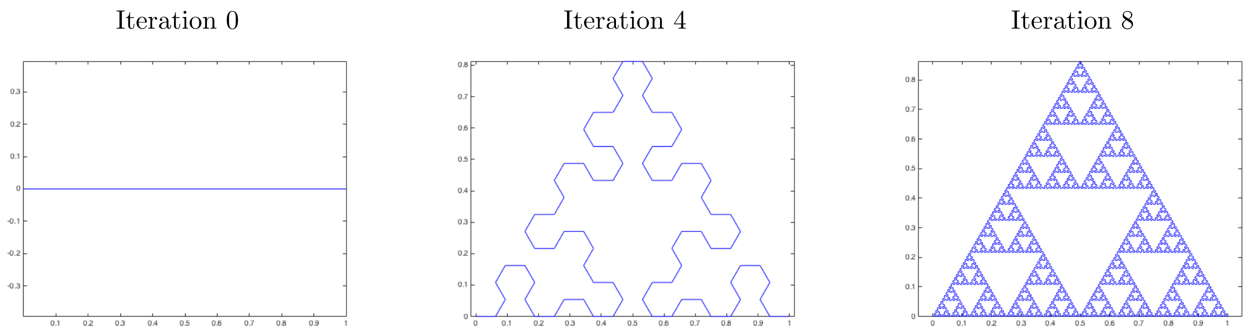

Figure 2D.2: Graphical representation of the initial, fourth and eights iteration of the Sierpinski triangle.

Lindenmayer Iteration

Interface	`def LindIter(System, N):`
	`    # Insert your code here`
	`    return LindenmayerString`

Input arguments	`System`: A string containing the name of the Lindenmayer system currently under scrutiny. The input can take the values `Koch` or `Sierpinski`. `N`: The number of iterations that should be calculated.
Output arguments	`LindenmayerString`: A string containing the output after `N` iterations of the chosen Lindenmayer system.
User input	No.
Screen output	No.
Description	The function must calculate N iterations of the system specified by `System` according to the replacement rules for the Koch curve and Sierpinski triangle respectively. The output of the iteration function should be stored in the string `LindenmayerString`.

Translation to turtle graphics commands

Interface	`def turtleGraph(LindenmayerString):`
	`    # Insert your code here`
	`    return turtleCommands`

Input arguments	`LindenmayerString`: A string of symbols representing the state of the system after the Lindemayer iteration.
Output arguments	`turtleCommands`: A row vector containing the turtle graphics commands consisting of alternating length and angle specifications $[l_1, \phi_1, l_2, \phi_2, \ldots]$.
User input	No.
Screen output	No.
Description	This function translates the string of symbols in `LindenmayerString` into a sequence of turtle graphics commands. The output consists of a row vector of numbers, which alternate between a number specifying the length of a line to be drawn and a number specifying the angle of the new line segment with respect to the drawing direction of the last line segment. Your function function should use the replacement rules consistent with the Lindenmayer system you choose to investigate. Which system this is can either be inferred

from the input `LindenmayerString` directly, or passed to the function by augmenting it by further input or output variables in the functions you program. You are free to choose how to implement this and creativity is appreciated.

Turtle graphics plot function

Interface

```
def turtlePlot(turtleCommands):
    # Insert your code here
```

Input arguments

`turtleCommands`: A row vector consisting of alternating length and angle specifications $[l_1, \phi_1, l_2, \phi_2, \ldots]$.

Output arguments

No.

User input

No.

Screen output

Yes (plot, see specifications below.)

Description

The plot function should turn the input vector `turtleCommands` into a graphical visualisation. This can be done by specifying the coordinates of the corners $\vec{x} = (x, y)$ of the straight line segments with a plot command. To find these coordinates, your function should follow the input vector, starting at the origin of the planar coordinate system and adding a new pair of coordinates after every line segment. The line segment has the length l_i, and must be drawn at an angle ϕ_i with respect to the previous line (as specified in the input vector of `turtleCommands`). The initial point is $\vec{x}_0 = (0, 0)^T$ and the initial direction is along a unit vector $\vec{d}_0 = (1, 0)^T$. Here a positive angle corresponds to turning left, and a negative one to turning right. The computation of the vector pointing to the point $\vec{x}_{i+1}$ by using the previous point $\vec{x}_i$ and previous drawing direction $\vec{d}_i$ with

$$\vec{d}_{i+1} = \begin{pmatrix} \cos(\phi_{i+1}) & -\sin(\phi_{i+1}) \\ \sin(\phi_{i+1}) & \cos(\phi_{i+1}) \end{pmatrix} \cdot \vec{d}_i \tag{2D.1}$$

$$\vec{x}_{i+1} = \vec{x}_i + l_{i+1} \cdot \vec{d}_{i+1} \tag{2D.2}$$

The results can be checked by comparing them with Fig. 2D.1 and 2D.2. The plots should include a suitable title, descriptive axis labels, and a data legend where appropriate. You are allowed to present the plots in separate figure windows or as sub-plots in a single figure window.

Main script

Interface

Must be implemented as a script.

Input arguments

No.

Output arguments

No.

User input

Yes (see specifications below.)

Screen output

Yes (see specifications below.)

Description

The user interacts with the program through the main script. When the user runs the main script he/she must have at least the following options:

1. Choose the type of Lindenmayer system and the number of iterations.
2. Generate plots.
3. Quit.

The user must be allowed to perform any reasonable number of Lindenmayer iterations for the chosen Lindenmayer system and subsequent graphical presentation of the obtained strings. The details of how the main script is designed and implemented is for you to determine. It is a requirement that the program is interactive, and you should strive to make it user friendly by providing sensible dialogue options. Consider what you would expect if you were to use such a script, and what you think would be fun. Play around and make a cool script.

Error handling

You must test that all input provided by the user are valid. If the user gives invalid input, you must provide informative feedback to the user and allow the user to provide the correct input. This includes also unreasonable choices with respect to the number of iterations (e.g. because of computational time).

1. Choose Lindenmayer system

When the user chooses a Lindenmayer system, you must ask the user to input one of the options listed in the following table.

`Input Option`	Name
1	Koch curve
2	Sierpinski triangle

2. Generate plots

If the user chooses to generate plots, call the `turtlePlot` function to display the plots corresponding to the current Lindenmayer system.

3. Quit

If the user chooses to quit the program, the main script should stop.

Possible extension

As a possible extension (not required) you could allow a user to define a new Lindenmayer system and to compute iterations of that and visualize this graphically. Further possibilities are to compute certain properties of the Lindenmayer systems like the curve length depending on the number of iterations.