INDEX PAGE
Pages
Lab 1
FAQ
Lab 2
Lab 3
Lab 7
Lab 6
Testing tips
Lab 4
Lab 10
Lab 5
Module overview
Support
Installing and finding Python
Snippets
Lecture slides
Lab 8
Lab 9
First steps
Testing demo

Home | **Lab 10**

Laboratory 10: Cup of tea

Prerequisits: curve fitting, data analysis, inverting function (root find)

Contents

PEP8

To monitor the coding style we turn on the PEP8 style guide monitoring in Spyder by:

1. Going to preferences in Spyder menu
2. Clicking on `Editor` in the selection list on the left
3. Clicking on `Code Introspection/Analysis` (top right corner)
4. Ticking the box `Real-time code style analysis`
5. Clicking `Apply` and `OK` (bottom right corner)

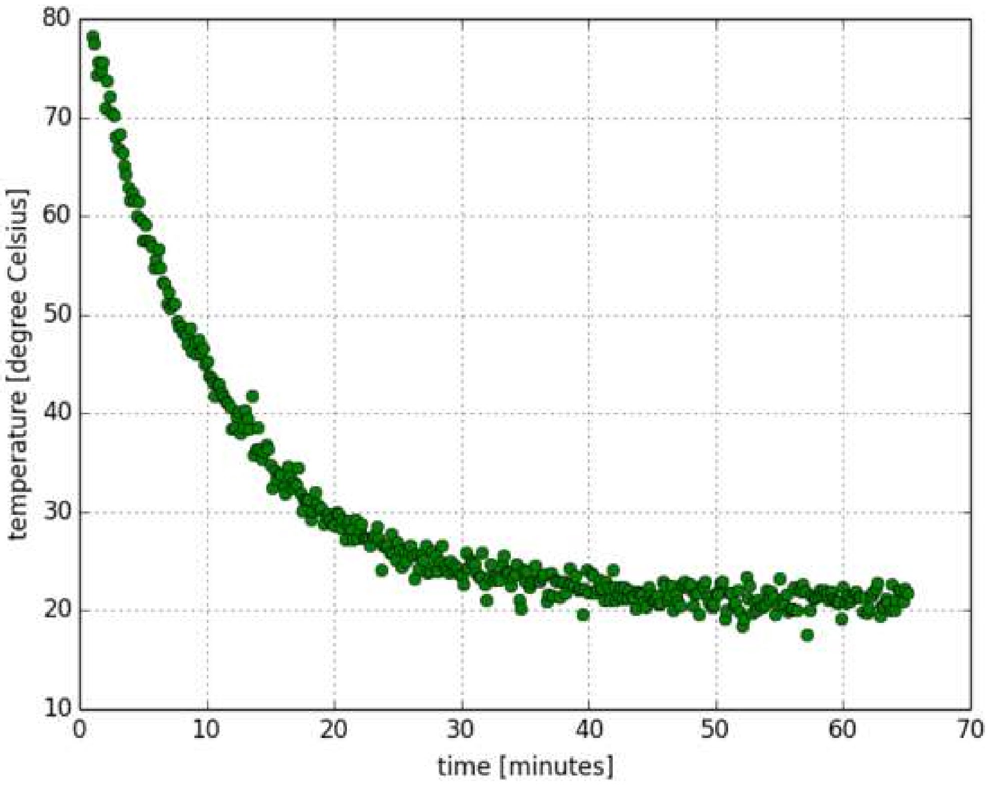
Preparation

In this lab, we will be using the curve_fit function from `scipy`, which is located at:

```
scipy.optimize.curve_fit
```

Background

The task is to study how a hot cup of tea or coffee cools down once the liquid has been poured into the cup. A dedicated research unit in a little known university in central Java, Indonesia, has conducted temperature measurements for a cup of tea and made the data available:



The specialised equipment takes one temperature reading (in degree Celsius) every 10 seconds, although the measurements are somewhat noisy. Unfortunately, something went wrong in the beginning of the measurement, so the data for the first minute are missing.

Research questions

The questions we aim to answer are:

1. what was the initial temperature of the tea in the cup at time t=0s?
2. how quickly does the tea cool down? In particular: after what time is it safe to drink it (we assume that 60C is a safe temperature).
3. what will be the final temperature of the tea if we wait infinitely long (presumably this will be the room temperature in this particular lab in Java).

Strategy

To make progress with this task, we define some variable names and have to make a few simplifying assumptions.

- We assume that *initial temperature*, `Ti`, is the temperature with which the tea has been poured into the cup.
- If we wait infinitely long, the final temperature will reach the value of the *ambient temperature*, `Ta`, which will be the environmental temperature in the lab where the measurements have been taken.
- We further assume that the cup has no significant heat capacity to keep the problem simple.
- We assume that the cooling process follows a particular model. In particular we assume that the rate with which the temperature `T` changes as a function of time `t` is proportional to the difference between the current temperature `T(t)` and the temperature of the environment `Ta`, i.e.:

  $$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{1}{c}(T_\mathrm{a} - T)$$

  We can solve this differential equation analytically and obtain a *model equation*:

  $$T(t) = (T_\mathrm{i} - T_\mathrm{a}) \exp\left(\frac{-t}{c}\right) + T_\mathrm{a} \qquad (1)$$

  where `c` is the time constant for the cooling process (which is expressed in seconds). The larger `c`, the longer it takes for the hot drink to cool down. Over a period of `c` seconds, the drink's temperature will decrease by roughly 2/3.

Exercises

Create a file `lab10.py` which you populate with the following functions:

1. A function `model(t, Ti, Ta, c)` which implements equation (1).

   Examples

   ```
   In [ ]: model(0, 100, 0, 10)
   Out[ ]: 100.0

   In [ ]: model(10, 100, 0, 10)
   Out[ ]: 36.787944117144235

   In [ ]: import math

   In [ ]: math.exp(-1) * 100
   Out[ ]: 36.787944117144235

   In [ ]: model(10, 100, 100, 10)
   Out[ ]: 100.0

   In [ ]: model(1000000, 100, 25, 10)
   Out[ ]: 25.0
   ```

   The function `model(t, Ti, Ta, c)` should return a single value if `t` is a floating point number, and an `array` if `t` is an array. For example:

```
In [ ]: model(0, 100, 20, 500)
Out[ ]: 100.0

In [ ]: from numpy import linspace

In [ ]: ts = linspace(0, 3600, 4)

In [ ]: ts
Out[ ]: array([    0.,  1200.,  2400.,  3600.])

In [ ]: model(ts, 100, 20, 500)
Out[ ]: array([ 100.        ,  27.25743626,   20.65837976,   20.05972686])
```

You achieve this behaviour by using the exponential function from `numpy` (i.e. `numpy.exp` rather than the exponential function from the math module) when you implement equation (1) in the `model` function.

2. `read2coldata(filename)` which opens a text file with two columns of data. The columns have to be separated by white space. The function should return a tuple of two numpy-arrays where the first contains all the data from the first column in the file, and the second all the data in the second column.

   Example: for a data file testdata.txt which contains

```
1.5  4
8    5
16   6
17   6.2
```

   we expect this behaviour:

```
In [ ]: read2coldata('testdata.txt')
Out[ ]: (array([  1.5,   8. ,  16. ,  17. ]), array([ 4. ,  5. ,  6. ,  6.2]))

In [ ]: a, b = read2coldata('testdata.txt')

In [ ]: a
Out[ ]: array([  1.5,   8. ,  16. ,  17. ])

In [ ]: b
Out[ ]: array([ 4. ,  5. ,  6. ,  6.2])
```

3. A function `extract_parameters(ts, Ts)` which expects a numpy array `ts` with time values and a numpy array `Ts` of the same length as `ts` with corresponding temperature values. The function should estimate and return a tuple of the three model parameters `Ti`, `Ta` and `c` (in this order) by fitting the model function as in equation (1) to the data `ts` and `Ts`.

   Hints:

   - The `curve_fit` function may need some initial guesses (through the optional function parameter `p0`) for the model parameters to be able to find a good fit.
   - You are strongly encouraged to plot your fitted curve together with the raw data to check that the fit is reasonable.

     You can use a function like this:

```
def plot(ts, Ts, Ti, Ta, c):
    """Input Parameters:

      ts : Data for time (ts)
                (numpy array)
      Ts : data for temperature (Ts)
                (numpy arrays)
      Ti : model parameter Ti for Initial Temperature
                (number)
      Ta : model parameter Ta for Ambient Temperature
```

```
                (number)
       c  : model parameter c for the time constant
                (number)

    This function will create plot that shows the model fit together
    with the data.

    Function returns None.
    """

    pylab.plot(ts, Ts, 'o', label='data')
    fTs = model(ts, Ti, Ta, c)
    pylab.plot(ts, fTs, label='fitted')
    pylab.legend()
    pylab.savefig('testcompare.pdf')  # or pylab.show()
```

4. A function `sixty_degree_time(Ti, Ta, c)` which expects the model paramaters `Ti` (initial temperature), `Ta` (ambient temperature) and `c` the cooling rate time constant. The function should return an estimate of the number of seconds after which the temperature of the drink has cooled down to 60 degree Celsius (60 degree is a temperature that is generally considered low enough not to damage tissue).

   You have at least two different possible ways of obtaining this number of seconds for a given set of model parameters (`Ti, Ta, c`). One involves a root finding algorithm. You can assume that `Ti` > 60 degree Celsius, and that `Ti > Ta`.

When you have completed your work, check and test the code carefully (including documentation strings). Submit your file by email with subject line `lab 10`.

Can you now answer the three research questions, i.e. (No need to submit these answers.)

- what was the initial temperature of the tea in the cup at time t=0s?
- how quickly does the tea cool down? In particular: after what time is it safe to drink it (we assume that 60C are a safe temperature).
- what will be the final temperature of the tea if we wait infinitely long (presumably this will be the room temperature in this particular lab in Java).

Return to Top