A Tale of three sisters Julia R and Python

Anjan Kr Dasgupta

Table of contents

Preface

This is a book of three languages Python R and Julia. The book is written in a versatile platform *Quarto* https://quarto.org/docs/books. The compilation of the book is done in the open source platform *Vstool* https://visualstudio.microsoft.com/downloads/. The book is meant for chemists physicists biologists medical scientists and environment modellers and all those who want to communicate with the computer and compose reproducible code to solve their own problems. Instead of getting into what language suits , my basic contention in this book is to have a comparative appraisal of the three sister languages. In some cases we use the same end problem to highlight the difference in approaches in the three languages.
Typically the industry looks for modellers who deal with specific engineering and financial models. The examples given in a typical language based lecture often deal with problems related to such "industry-friendly" problems. For a change, I thought it will be interesting to start with problems that are simple but domain knowledge based. The person dealing with bank will understand the market based examples with ease. A chemist on the other hand would communicate more if the computational problem is related to chemistry. Often teh computational treatises in science (STEM) become specialised, e.g., molecular modelling problem. We have made a choice to restrict ourselves to problems which are simple intuitive and the coding or composing can provide us some additional insight. Long back I found a book in which there was a mixed language problem solving approach using Matlab, Maple and Mathematica (Moller (2007)). But the sister languages we have used (Python R and Julia) are used in more trendy applications like data science, AI and genomics research. A challenging task in this book will be to restrict ourselves to a limited space so that the basic purpose namely, the problem solving aptitude is retained. I hope we will be able to satisfy the reader who we assume will typically a researcher or a scientist with a curious mind or even a undergraduate student with some keen interest in computational problems and how they are solved using different language platforms.

Table 1: *Time elapsed (ms) for 100000 iterations with floating point variables*

Language	Time (ms)
Python	9.8
R	9.5
Julia	4.0

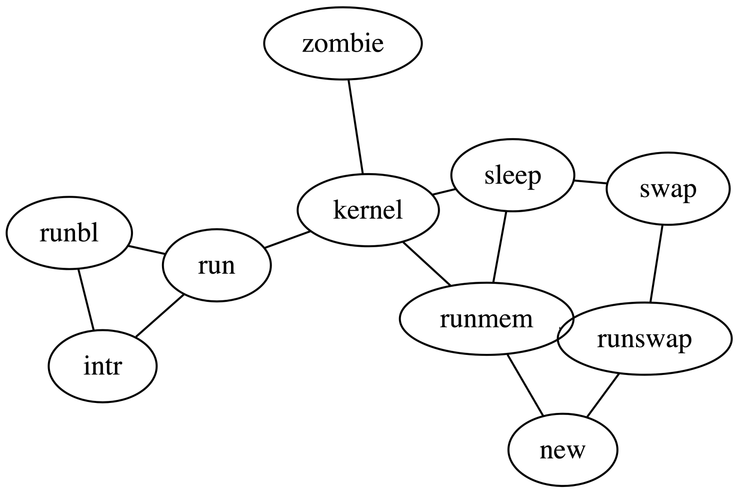
The Table 1 also highlights the "Two Language Problem" . High level langages are often goof in testing ideas, whereas, the low level languages often perform better but are lesser communicable to a random user. Julia as seen in the Table 1, serve the bindings of the low-level code to high-level version. Typically, Julia may nor be as fast as C++ or Fortran but is faster than the two sister counterparts R and Python. Interestingly, in the example above R is marginally faster than python, but the execution time differences are often dependent on the loop size.

Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

Simple Charts



Example of a mermaid chart

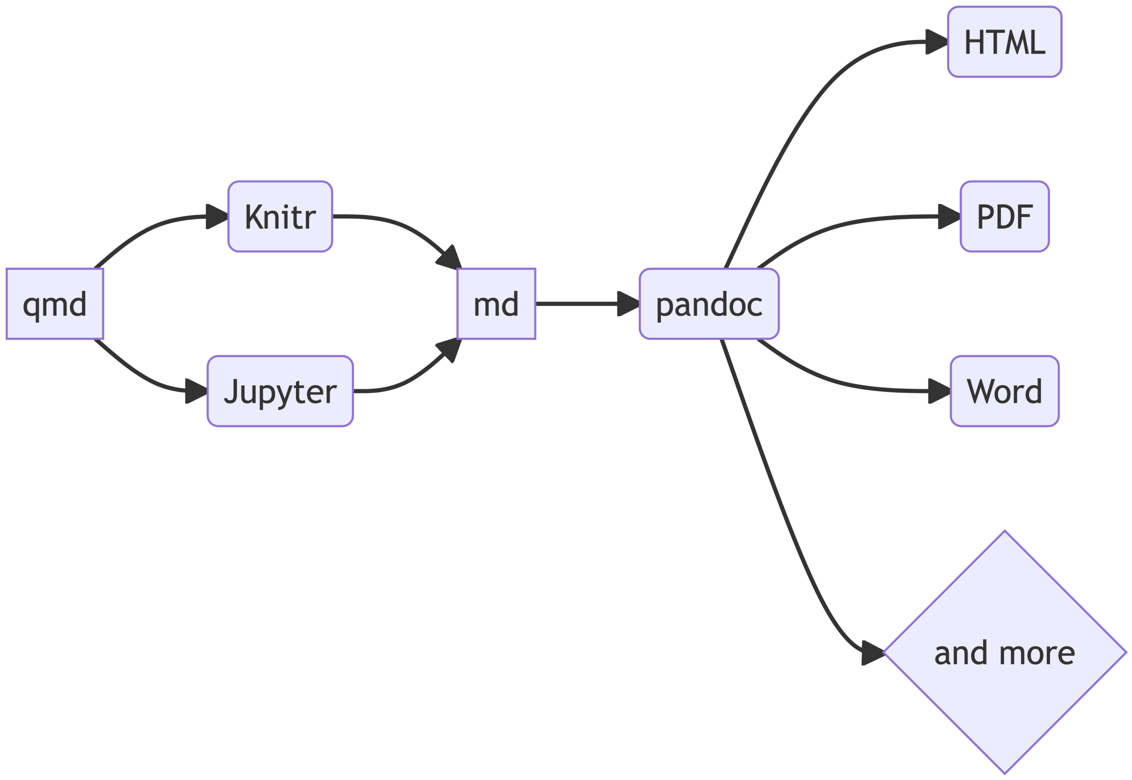


Figure 1: How Quarto orchestrates rendering of documents: start with a qmd file, use the Knitr or Jupyter engine to perform the computations and convert it to an md file, then use Pandoc to convert to various file formats including HTML, PDF, and Word.

1 Python

```
import numpy as np
#from numpy import arange
import matplotlib.pyplot as plt
from matplotlib import pyplot

import time
import matplotlib_venn as vn
import pandas  as pd
from pandas import read_csv
import scipy  as scipy
from scipy.optimize import curve_fit
import datetime as dt
# import maya
```

1.1 Loop time in python

```
start=time.time()
sum=0
for i in range(100000):
    sum=sum+0.001
time.time()-start
```

```
0.011715888977050781
```

1.2 Amino acids and nucleic acids as Python Lists

```
amino_all=['GAVLIPFYWSTNQCMDEHKR']
amino_all
```

```
['GAVLIPFYWSTNQCMDEHKR']
```

```
['GAVLIPFYWSTNQCMDEHKR']
```

1.3 Amino acids and nucleic acids as mathematical sets

We can use python to express the set of all amino acids.In formal terms this can be expressed as

$$\{G, A, V, L, I, P, F, Y, W, S, T, N, Q, C, M, D, E, H, K, R\} = amino\_all \tag{1.1}$$

Similarly,we can express the set of all nucleic acids participating in formation of DNA as:

$$\{A, G, C, T\} = nuc \tag{1.2}$$

Now let us for simplicity we express the set of amino acid and its subsets by

1.4 Amino acids as python sets

```
plt.rcParams.update({'font.size': 8})
amino_all = set(['G','A','V','L','I',
'P','F','Y','W','S','T','N','Q','C','M',
'D','E','H','K','R'])
```

1.5 Amino acid set as an object

```
class amino:
  def __init__(self,sequence):
    self.name = name
    self.sequence = sequence
```

```
amino.all='GAVLIPFYWSTNQCMDEHKR'
amino.polar='NQSTKRHDECY'
amino.Hbond='CWNQSTYKRHDE'
amino.S='CM'
amino.positive='KRH'
amino.negative='DEC'
amino.aromatic='FWYH'
amino.aliphatic='GAVLIP'
amino.dilsul='C'
amino.cyclic='P'
amino.amphiphatic='WYM'
amino.hydrophobic='AILMFVPG'
```

```
# print one of the objects
print(amino.positive)
```

```
KRH
```

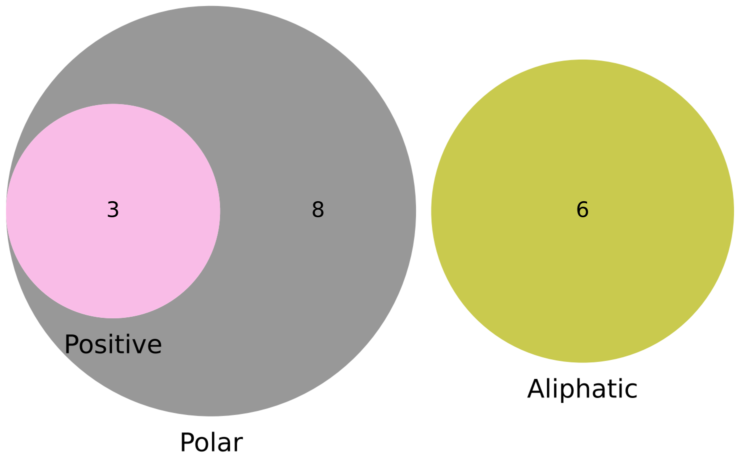
1.6 Cardianility of the amino acid set

```
len(amino.all)
```

20

```
#!pip install matplotlib_venn
```

```
from matplotlib_venn import venn2, venn2_circles, venn2_unweighted
from matplotlib_venn import venn3, venn3_circles
from matplotlib import pyplot as plt
vd3=venn3([set(amino.positive),set(amino.polar),set(amino.aliphatic)],
set_labels=('Positive', 'Polar','Aliphatic'),
set_colors=('#e377c2', '#7f7f7f', '#bcbd22'),
alpha = 0.8)
plt.show()
```



1.7 Inference

- None of the aliphatic amino acids are positive or polar
- While this may sound trivial, let us ask whether how many nonpolar amino acids show hydrogen bonding.
- We can solve it by
  - Venn Diagram
  - By simple set manipulation

1.8 Deriving the set of nonpolar amino acids

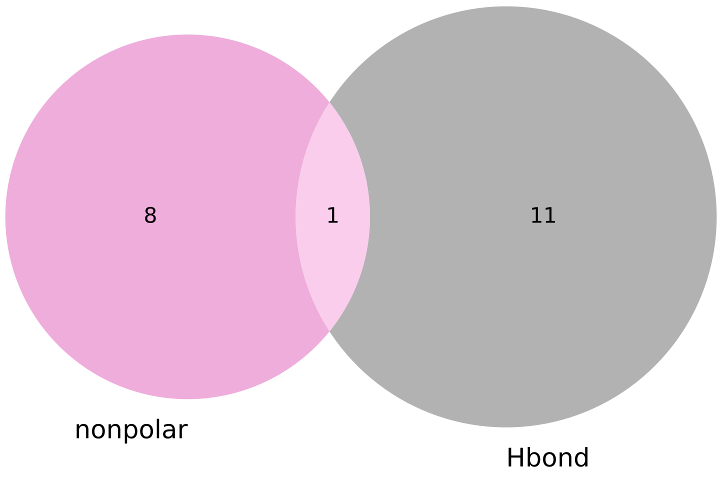
```
# Finding the set of nonpolar amino acids

N=set(amino.all)-set(amino.polar)
```

```
N
```

```
{'A', 'F', 'G', 'I', 'L', 'M', 'P', 'V', 'W'}
```

```
N.intersection(set(amino.Hbond))
```

```
vd2=venn2([N,set(amino.Hbond)],
set_labels=('nonpolar', 'Hbond'),
set_colors=('#e377c2', '#7f7f7f'),
alpha = 0.6)
```



1.9 Inference

Trp (w) is the only nonpolar amino acid showing hydrogen bonding properties

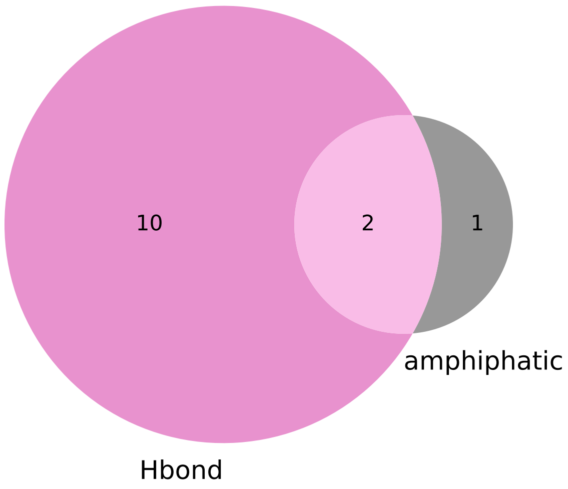
```
result = set(amino.Hbond).intersection(set(amino.amphiphatic))
print(result)# Should be a finite set
```

```
{'W', 'Y'}
```

1.10 Inference

Among the ampipathic amino acids which are also hydrogen bonded. Turns out to be tyrosine and tryptophan,both being aromatic.

```
vd2=venn2([set(amino.Hbond),set(amino.amphiphatic)],
 set_labels=('Hbond', 'amphiphatic'),
 set_colors=('#e377c2', '#7f7f7f'),
 alpha = 0.8)
plt.show()
```



1.11 Another simple amino-set problem

How many aromatic amino acids exist other than the ampipathic and Hbonded?

```
nonH=set(amino.aromatic)- (set(amino.Hbond).intersection(set(amino.amphiphatic)))
nonH
```

```
{'F', 'H'}
```

1.12 A problem on disulfide bridges

How to find an amino acid containing sulfur that does not form di-sulfide bridges

1.13 Solution - Compare the two sets amino.S and amino.disul

```
result = set(amino.S)-set(amino.dilsul)
print(result)
```

```
{'M'}
```

1.14 Comments on the above

The implication of the above is that the singular amino acid C is responsible for the dsulphide bridge formation although there are two amino acids C and M containing S.

1.15 A structural insight from the python based set theoretic analysis

Structurally, it may be interesting to ask why M does not play a role in the disulphide bridge formation. Cysteine doesn't have any methyl group linked to its sulfur. It has a sulfhydryl group. This group is very reactive and can react with another cysteine to form a dimer, which is called cystine. Methionine is not only larger, but it has a large methyl group attached to sulfur. This not only reduces the reactivity, but it also causes molecular hindrance in real time,called steric hindrance. This is the basic reason why cysteine can dimerize, but methionine cannot. But cysteine can form a disulfide bridge with other sulfur containing AA such as methione as well.

1.16 String Examples

```
'TATA' in 'TATATATATATATATATATATATA'
```

```
True
```

```
'aaa' + 'ccc' + 'ttt' + 'ggg'
```

```
'aaaccctttggg'
```

```
'TA' * 12
```

```
'TATATATATATATATATATATATA'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[0]
```

```
'M'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[-5]
```

```
'D'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[1:4]
```

```
'NKM'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[4:-1]
```

```
'DLVADVAEKTDLSKAKATEVIDAVF'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[-5:-4]
```

```
'D'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[-1: :-1]
```

```
'AFVADIVETAKAKSLDTKEAVDAVLDMKNM'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[:8]
```

```
'MNKMDLVA'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[9:]
```

```
'VAEKTDLSKAKATEVIDAVFA'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[9:-1]
```

```
'VAEKTDLSKAKATEVIDAVF'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[:]
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[16:0:-4]
```

```
'SKDD'
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'[16::-4]
```

```
'SKDDM'
```

```
len('MNKMDLVADVAEKTDLSKAKATEVIDAVFA')
```

```
30
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'.find('DL')
```

```
4
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'.find('DL', 5)# find DL after the 6th position
```

```
14
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'.find('DL', 5, 12)
```

```
-1
```

```
'MNKMDLVADVAEKTDLSKAKATEVIDAVFA'.startswith('DL')
```

```
False
```

```
'AAAAATCCCGAGGCGGCTATATAGGGCTCCGGAGGCGTAATATAAAA'.find('TCCGGA')
```

```
27
```

```
first="Anjan"
last="Dasgupta"
print(first[-1: :-1],last[-1: :-1])
```

```
najnA atpugsaD
```

1.17 List Comprehension

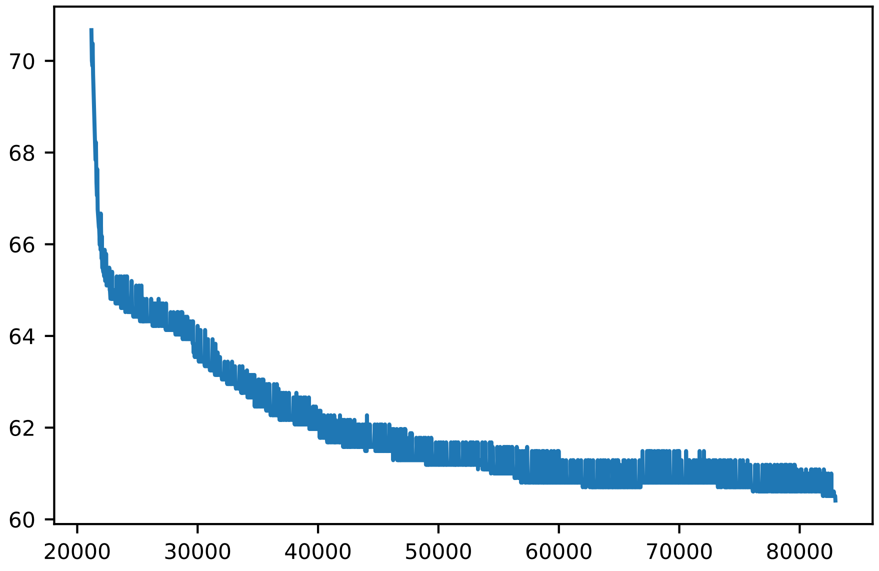
```
S = [x**2 for x in range(10)]
V = [2**i for i in range(13)]
M = [x for x in S if x % 2 == 0]
print(S,"\n",V,"\n",M)
```

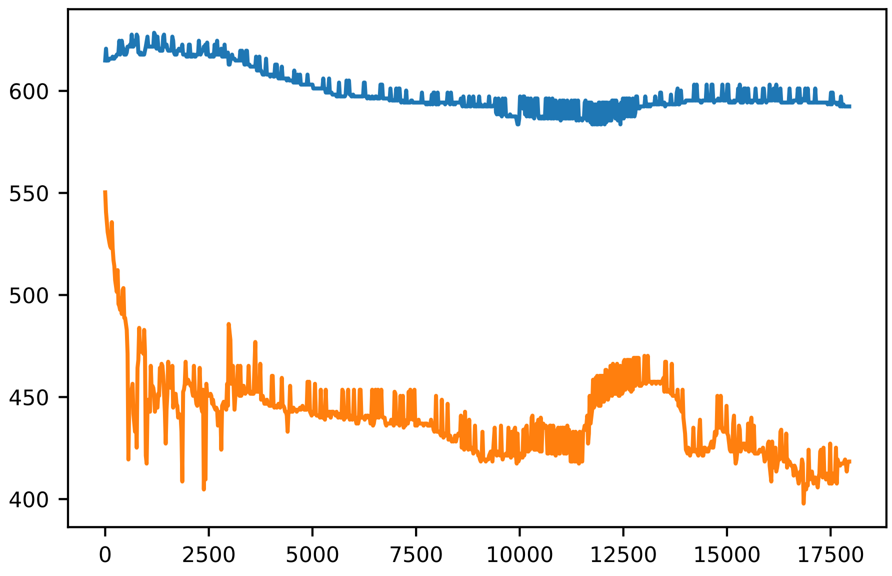
```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
 [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]
 [0, 4, 16, 36, 64]
```

1.18 Reading CSV files (IoT Data)

<

```
# df1=pd.read_csv('nov_15_22_1.csv')
df1=pd.read_csv('nov_15_22_2.csv')
df2=pd.read_csv('nov_21_22_1.csv')
df3=pd.read_csv('nov_25_22.csv')
# def to_sec(x):
#     return int(x[0]) * 3600 + int(x[1]) * 60 + int(x[2])

df2.iloc[:,0]=[int(round(dt.datetime.strptime(i[:-6], '%Y-%m-%dT%H:%M:%S').timestamp())) f
df1.iloc[:,0]=[int(round(dt.datetime.strptime(i[:-6], '%Y-%m-%dT%H:%M:%S').timestamp())) f
df3.iloc[:,0]=[int(round(dt.datetime.strptime(i[:-6], '%Y-%m-%dT%H:%M:%S').timestamp())) f
t1=df1.iloc[:,0]
t2=df2.iloc[:,0]
t3=df3.iloc[:,0]
humid1=df1['humidity value']
humid21=df2['field1']
humid22=df2['field2']
humid3=df3['Node2']
plt.plot(t1[1000:]-t1[0],humid1[1000:])
plt.show()
plt.figure()
plt.plot(t2-t2[0],humid21)
plt.plot(t2-t2[0],humid22)
plt.show()
```





1.19 Curve fitting in python

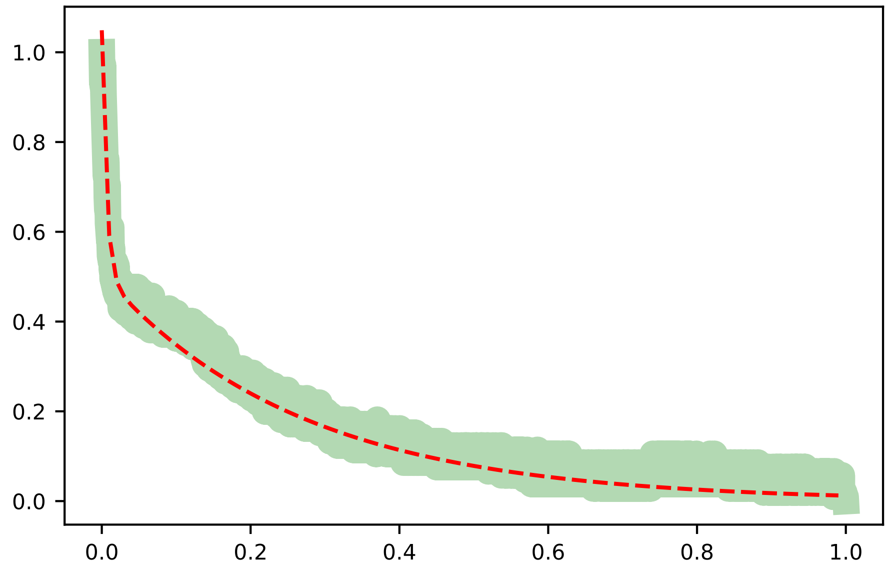
https://machinelearningmastery.com/curve-fitting-with-python/>

```
def func(x, q,a, b, c):
    return a * np.exp(-b * x) + c * np.exp(-q * x)
def scale(x, out_range=(0, 1)):
    domain = np.min(x), np.max(x)
    y = (x - (domain[1] + domain[0]) / 2) / (domain[1] - domain[0])
    return y * (out_range[1] - out_range[0]) + (out_range[1] + out_range[0]) / 2
```

```
xdata= (t1[1000:]-t1[0])/max(t1)
x = np.array(xdata.values.tolist())
ydata=humid1[1000:]
y=np.array(ydata.values.tolist())
```

```
type(x)
type(y)
x=scale(x)
y=scale(y)
#plt.plot(x,y)
#plt.show()
```

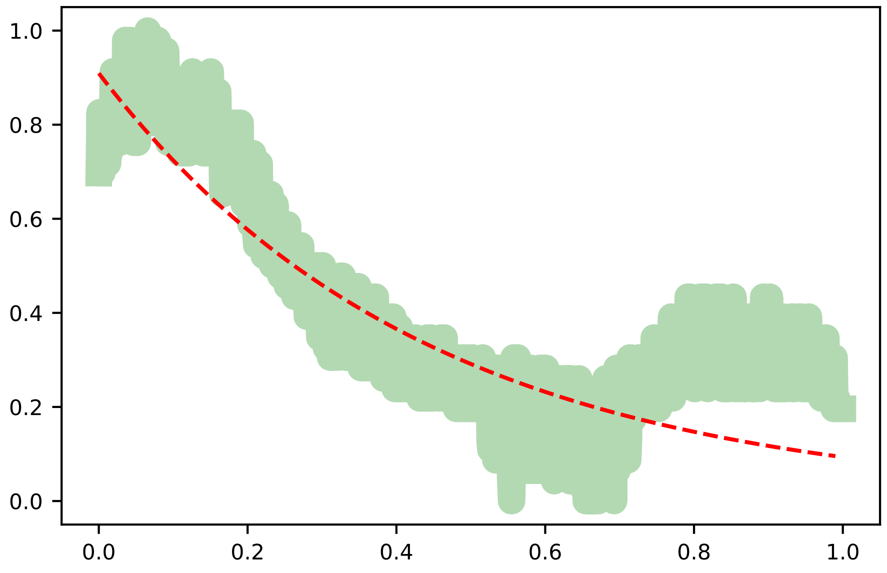
```
popt, _ = curve_fit(func, x, y)
q,a,b,c=popt
#pyplot.scatter(x, y)
plt.plot(x, y, c='green', lw=10, alpha=0.3, label="Lighten")
x_line = np.arange(min(x), max(x), 0.01)
y_line = func(x_line, q,a, b, c)
pyplot.plot(x_line, y_line, '--', color='red')
pyplot.show()
print("a * np.exp(-b * x) + c * np.exp(-q * x)","\n")
print(['q','a','b','c'],"\n")
print(popt)
```



```
a * np.exp(-b * x) + c * np.exp(-q * x)

['q', 'a', 'b', 'c']

[  3.72826723   0.5433054  165.12319896   0.50569432]
```

```
xdata= (t2-t2[0])/max(t2)
x = np.array(xdata.values.tolist())
ydata=humid21
y=np.array(ydata.values.tolist())
```

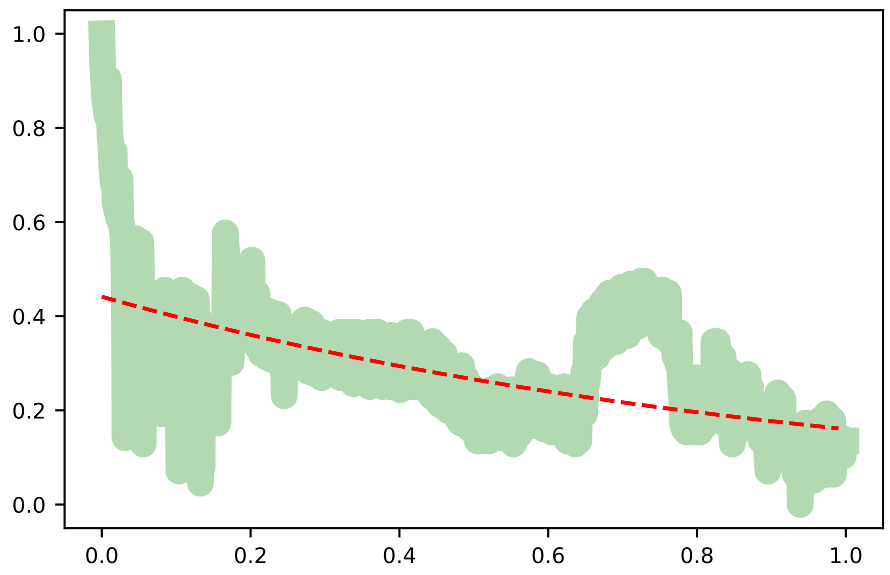
```
y=np.array(ydata.values.tolist())
x=scale(x)
y=scale(y)
#plt.plot(x,y)
#plt.show()
popt2, _ = curve_fit(func, x, y)
q,a,b,c=popt2
#pyplot.scatter(x, y)
plt.plot(x, y, c='green', lw=10, alpha=0.3, label="Lighten")
x_line = np.arange(min(x), max(x), 0.01)
y_line = func(x_line, q,a, b, c)
pyplot.plot(x_line, y_line, '--', color='red')
pyplot.show()
print("a * np.exp(-b * x) + c * np.exp(-q * x)","\n")

print(['q','a','b','c'],"\n")


print(popt2)
```



```
a * np.exp(-b * x) + c * np.exp(-q * x)

['q', 'a', 'b', 'c']
```

```
[ 2.27608078 -0.38982791  2.2759168   1.29918115]
```

```
xdata= (t2-t2[0])/max(t2)
x = np.array(xdata.values.tolist())
ydata=humid22
y=np.array(ydata.values.tolist())

y=np.array(ydata.values.tolist())
x=scale(x)
y=scale(y)
#plt.plot(x,y)
#plt.show()
popt3, _ = curve_fit(func, x, y)
q,a,b,c=popt3
#pyplot.scatter(x, y)
plt.plot(x, y, c='green', lw=10, alpha=0.3, label="Lighten")
x_line = np.arange(min(x), max(x), 0.01)
y_line = func(x_line, q,a, b, c)
pyplot.plot(x_line, y_line, '--', color='red')
pyplot.show()
print("a * np.exp(-b * x) + c * np.exp(-q * x)","\n")

print(['q','a','b','c'],"\n")


print(popt3)
```



```
a * np.exp(-b * x) + c * np.exp(-q * x)

['q', 'a', 'b', 'c']

[ 1.01452245  0.96023685  1.01440371 -0.51869714]
```

2 R

```r
remotes::install_github("allisonhorst/palmerpenguins")
```

```
Skipping install of 'palmerpenguins' from a github remote, the SHA1 (c19a9044) has not change
  Use `force = TRUE` to force installation
```

```r
library(tidyverse)
```

```
-- Attaching packages --------------------------------------- tidyverse 1.3.2 --

v ggplot2 3.4.0     v purrr   0.3.4
v tibble  3.1.7     v dplyr   1.0.9
v tidyr   1.2.0     v stringr 1.4.0
v readr   2.1.2     v forcats 0.5.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
```

```r
library(palmerpenguins)
```

2.1 R loop time

```r
sum=0
start_time <- Sys.time()
for (i in 1:100000) {
  sum=sum+0.001
}
end_time <- Sys.time()
end_time - start_time
```

```
Time difference of 0.009418011 secs
```

2.2 Meet Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

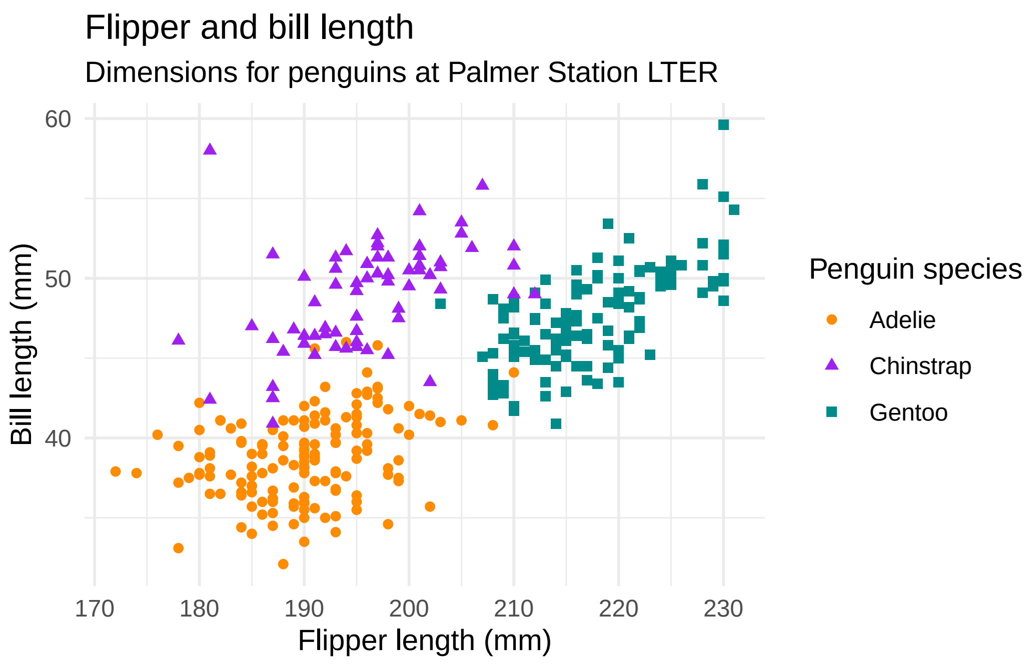
2.3 Meet the penguins

{style="float:right;" fig-alt="Illustration of three species of Palmer Archipelago penguins: Chinstrap, Gentoo, and Adelie. Artwork by Trivelpiece (1981)," width="401"}

The `penguins` data from the **palmerpenguins** package contains size measurements for 344 penguins from three species observed on three islands in the Palmer Archipelago, Antarctica.

The plot below shows the relationship between flipper and bill lengths of these penguins.

```
ggplot(penguins,
       aes(x = flipper_length_mm, y = bill_length_mm)) +
  geom_point(aes(color = species, shape = species)) +
  scale_color_manual(values = c("darkorange","purple","cyan4")) +
  labs(
    title = "Flipper and bill length",
    subtitle = "Dimensions for penguins at Palmer Station LTER",
    x = "Flipper length (mm)", y = "Bill length (mm)",
    color = "Penguin species", shape = "Penguin species"
  ) +
  theme_minimal()
```


Flipper and bill length
Dimensions for penguins at Palmer Station LTER
Bill length (mm)
60
50
40
170
180
190
200
210
220
230
Flipper length (mm)
Penguin species
Adelie
Chinstrap
Gentoo

3 Julia

3.1 Loop time in Julia

```
sum=0
@time for i in 1:100000 ; sum+=0.001;end;
```

```
  0.004005 seconds (100.03 k allocations: 1.528 MiB, 45.19% compilation time)
```

3.2 Polynomials in Julia

```
using Polynomials
Polynomial([1,2,3], :s)
```

$1 + 2 \cdot s + 3 \cdot s^2$

3.2.1 Construct Polynomial from Roots

```
using Polynomials
fromroots([1,2,3]) # (x-1)*(x-2)*(x-3)
```

$-6 + 11 \cdot x - 6 \cdot x^2 + x^3$

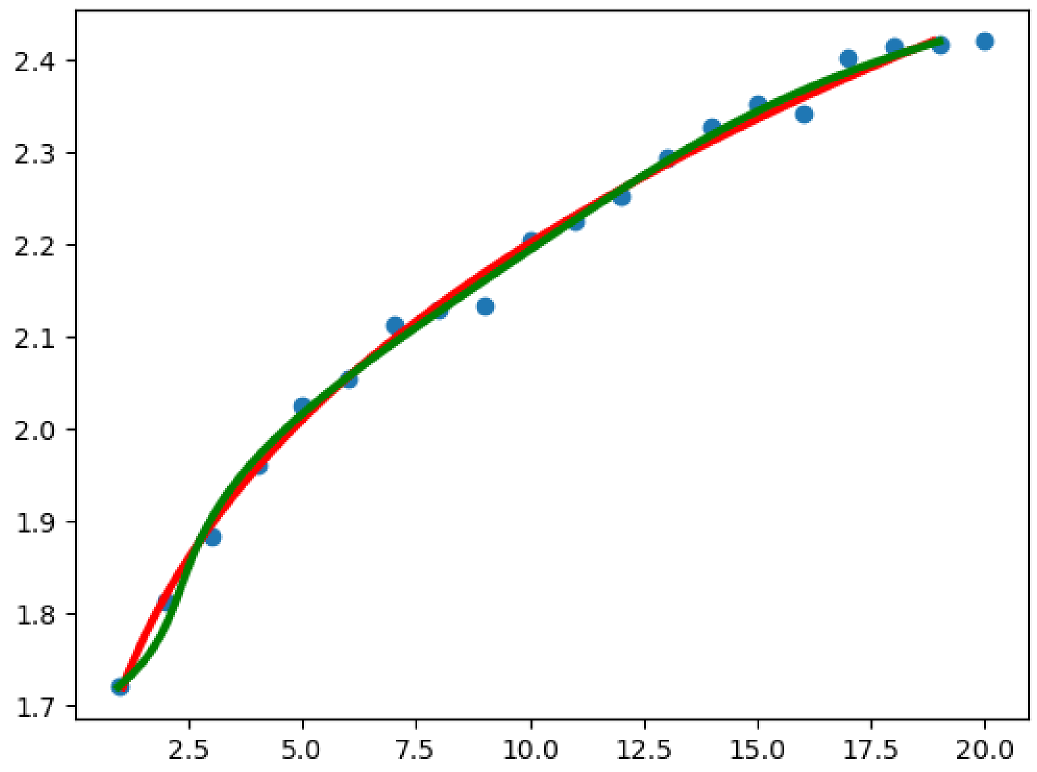
3.2.2 Evaluate polynomial from roots

```
p = Polynomial([1, 0, -1]);
p(0.1)
```

```
0.99
```

3.2.3 Curve Fitting

```julia
using PyPlot
using CurveFit

U = 1.0:20.0
E = @. sqrt(2 + 1 * U ^ 0.45) + randn()/60
e = range(minimum(E), maximum(E), length=50)

f1 = curve_fit(KingFit, E, U)
# see httpsf2://juliahub.com/ui/Packages/CurveFit/yfYeb/0.3.2  for explanation of kingfit

U1 = f1.(e)

f2 = curve_fit(Polynomial, E, U, 5)
U2 = f2.(e)



plot(U, E, "o", U1, e, "r-", U2, e, "g-", linewidth=3)
```



```
3-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x2a00a71f0>
 PyObject <matplotlib.lines.Line2D object at 0x2a00a71c0>
 PyObject <matplotlib.lines.Line2D object at 0x2a00a7400>
```

3.3 Structure in Julia

```
struct Person
    name::String
    age::Int64
end

me = Person("Anjan", 34)


typeof(me)
```

```julia
me.name



fieldnames(Person)
fieldtypes(Person)

# mutable structs

mutable struct MutablePerson
    name::String
    age::Int64
end

jose_mutable = MutablePerson("Jose", 34)

jose_mutable.name = "José"
jose_mutable.age = 35

jose_mutable

function newborn!(x::MutablePerson)
    x.age = 0
end

newborn!(jose_mutable)

jose_mutable



# abstract types

abstract type Pet end

struct Dog <: Pet end

struct Cat <: Pet end

function encounter(x::Pet, y::Pet)
    return "fallback"
end
```

```
function encounter(x::Dog, y::Cat)
    return "Oh, there's a chase"
end

function encounter(x::Cat, y::Dog)
    return "Oh, there's hissing"
end

rex = Dog()
meow = Cat()

encounter(rex, meow)
encounter(meow, rex)

struct Giraffe <: Pet end

godfried = Giraffe()

encounter(rex, godfried)
```

```
"fallback"
```

Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. "Literate Programming." *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.

Moller, Karl Dieter. 2007. *OPTICS Learning by Computing, with Examples Using Mathcad®, Matlab®, Mathematica®, and Maple®: With 308 Illustrations*. Springer.

Trivelpiece, Wayne Zebulon. 1981. *Ecological Studies of Pygoscelid Penguins and Antarctic Skuas*. State University of New York College of Environmental Science; Forestry.