EUROKIN MEETING, FEB 24TH, 2021

Parameter Estimation Using Python

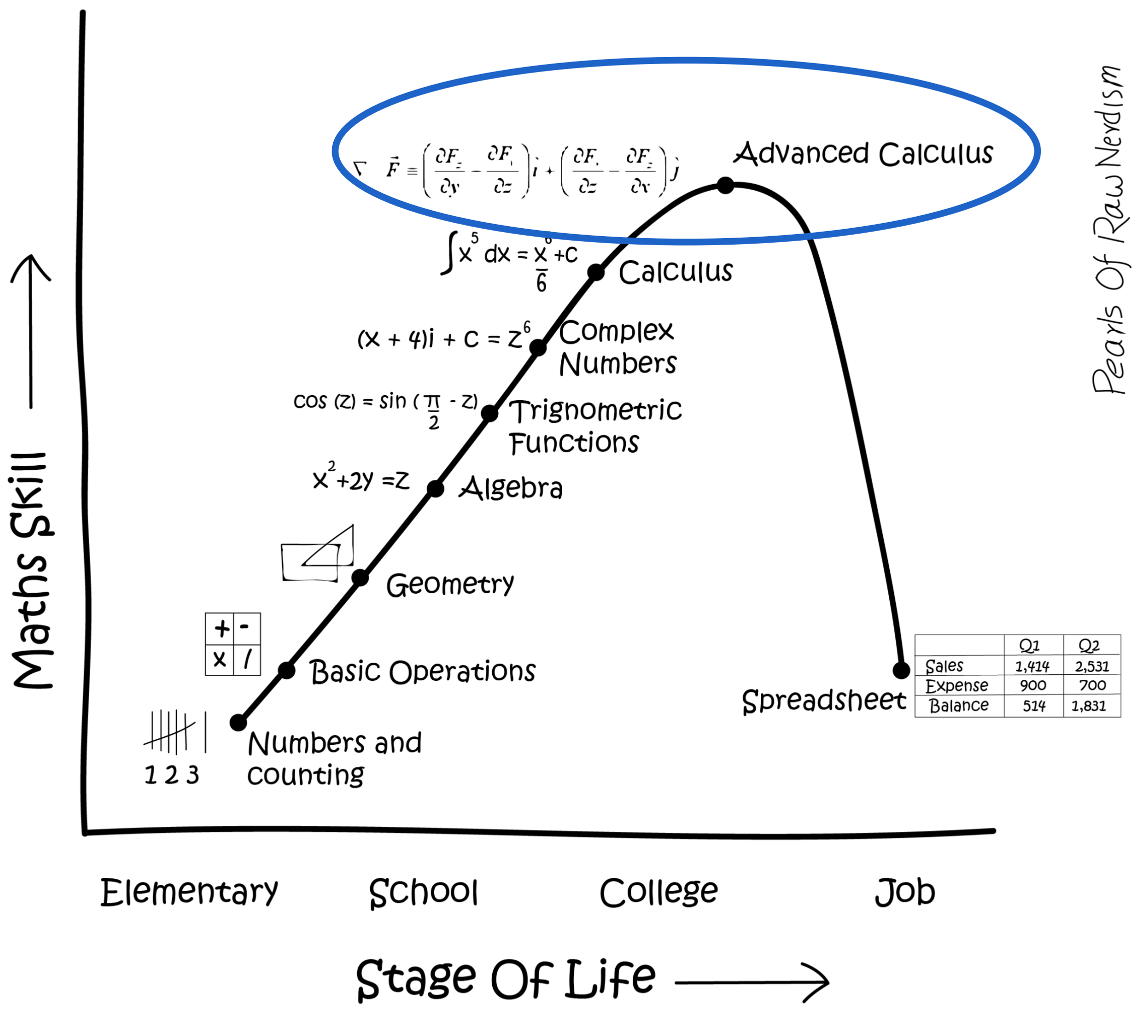
Reza Monjezi[1], Javier Ibanez Abad[1], Ana Bjelic[1], Joris W. Thybaut[1]

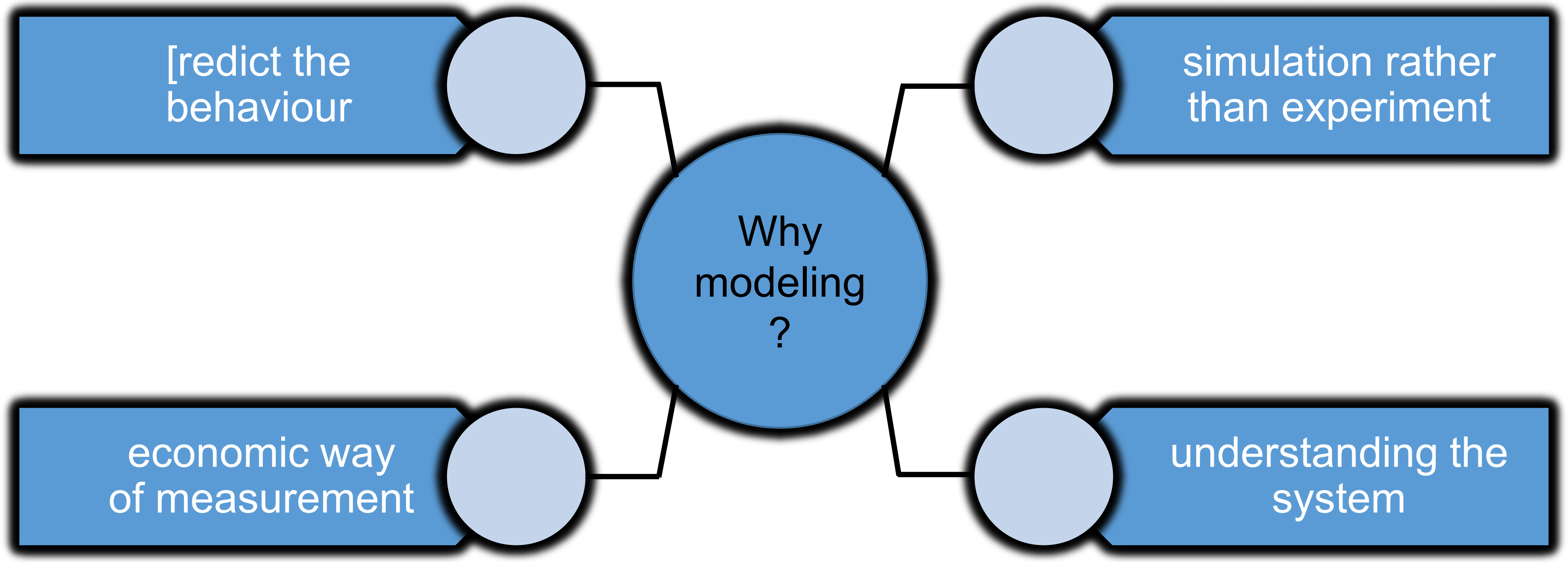
[1]Laboratory for Chemical Technology

GHENT UNIVERSITY

LCT DRIVING CHEMICAL TECHNOLOGY

importance of mathematical modelling

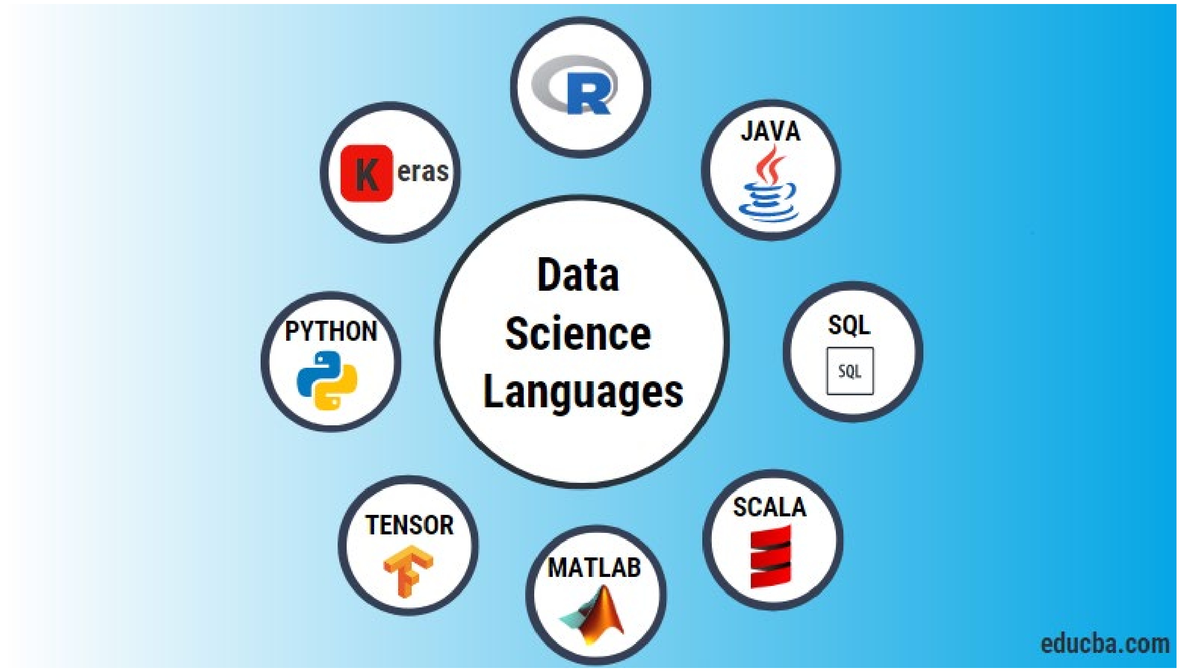



[redict the behaviour
simulation rather than experiment
Why modeling ?
economic way of measurement
understanding the system

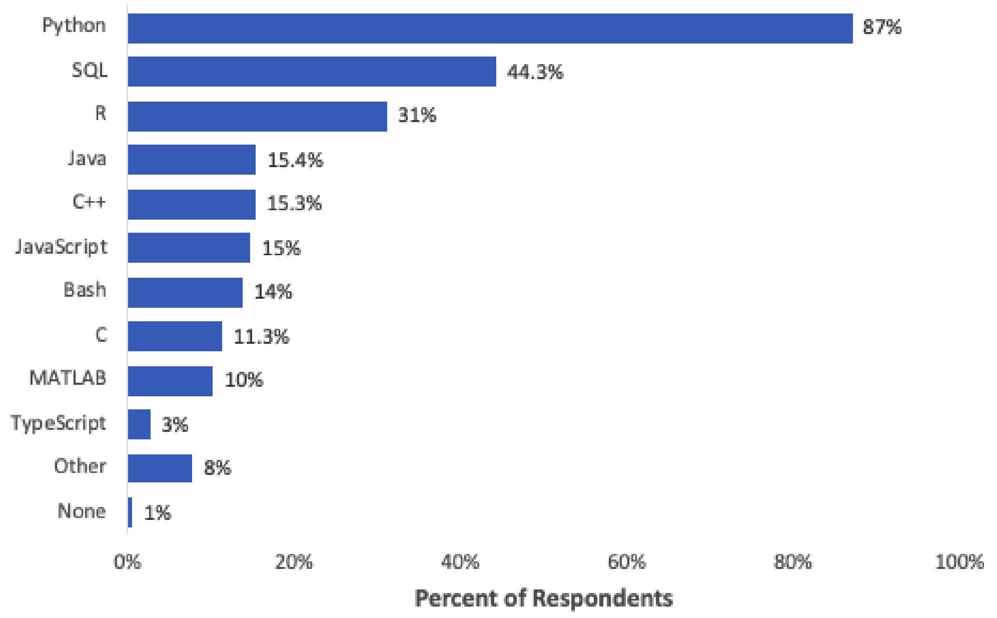
GHENT UNIVERSITY

LCT
DRIVING CHEMICAL TECHNOLOGY

programing language



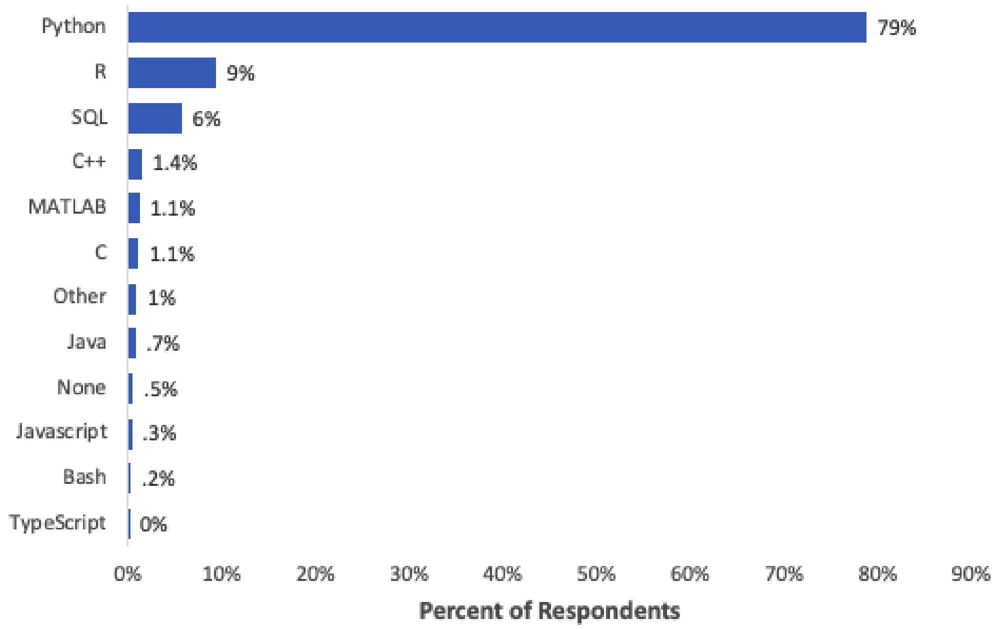
What programming languages do you use on a regular basis?



Note: Data are from the 2019 Kaggle ML and Data Science Survey. You can learn more about the study here: https://www.kaggle.com/c/kaggle-survey-2019.
A total of 19717 respondents completed the survey; the percentages in the graph are based on a total of 14762 respondents who provided an answer to this question.

Copyright 2020 Business Over Broadway

What programming language would you recommend an aspiring data scientist to learn first?

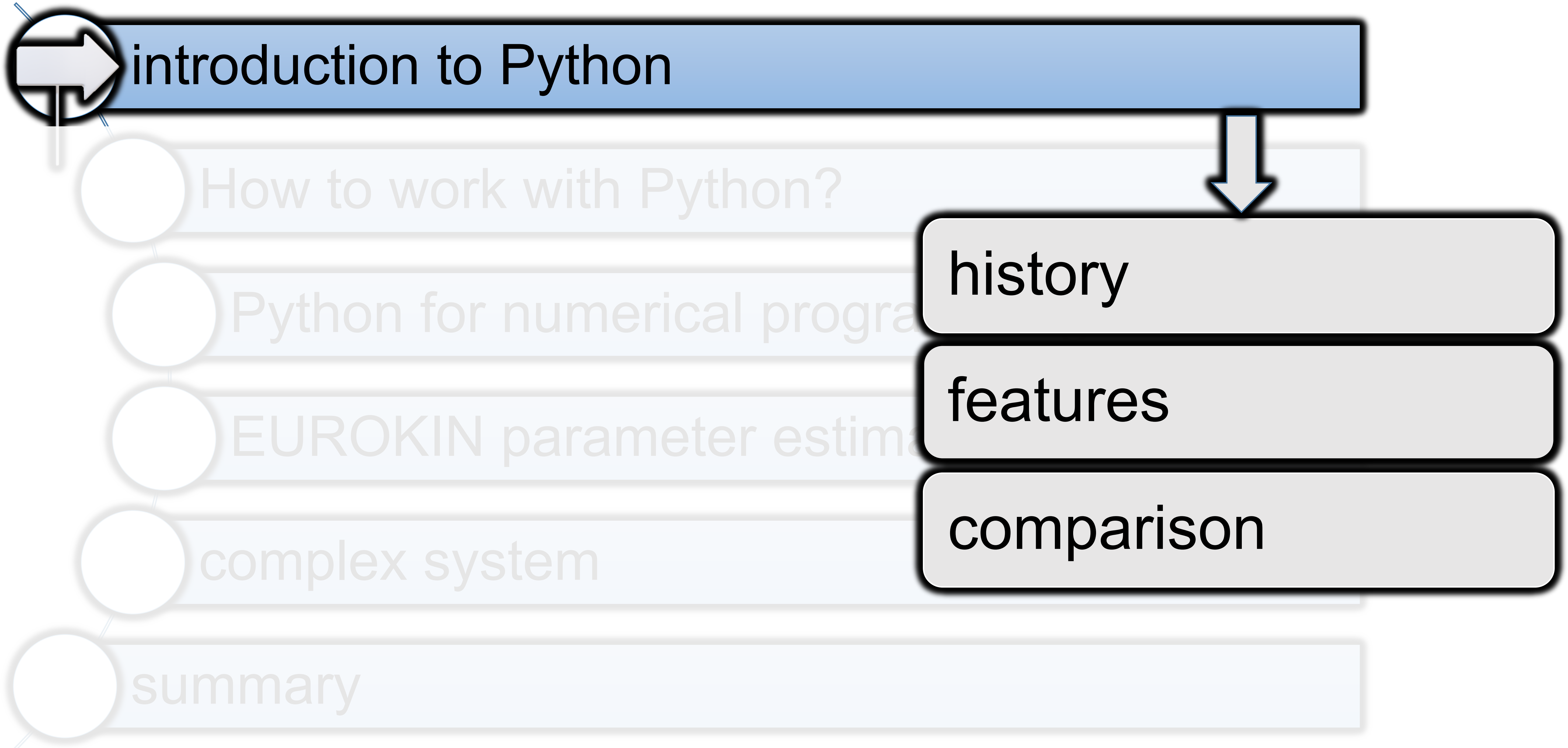


Note: Data are from the 2018 Kaggle ML and Data Science Survey. You can learn more about the study here: https://www.kaggle.com/c/kaggle-survey-2019.
A total of 19717 respondents completed the survey; the percentages in the graph are based on a total of 14377 respondents who provided an answer to this question.

Copyright 2020 Business Over Broadway

programing language

overview



introduction to Python

How to work with Python?

Python for numerical progra

EUROKIN parameter estima

complex system

summary

history

features

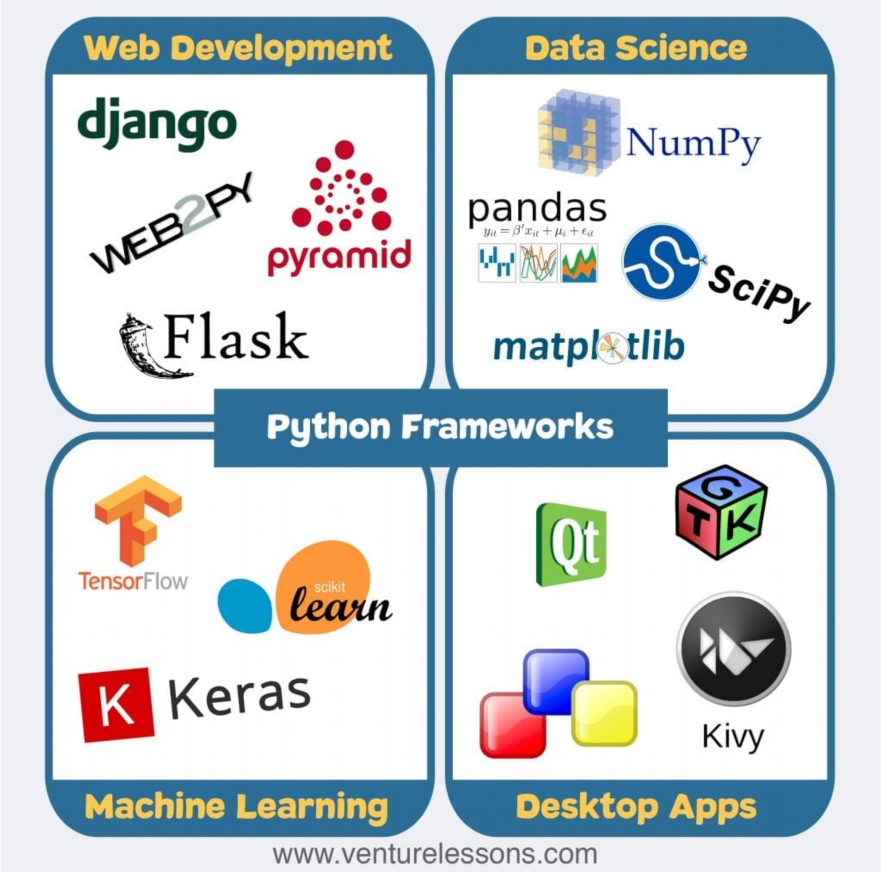
comparison

introduction to Python

Guido van Rossum

one of the fastest growing programming languages in terms of

- No. of developers
- No. of libraries
- No. of areas
- No. of companies

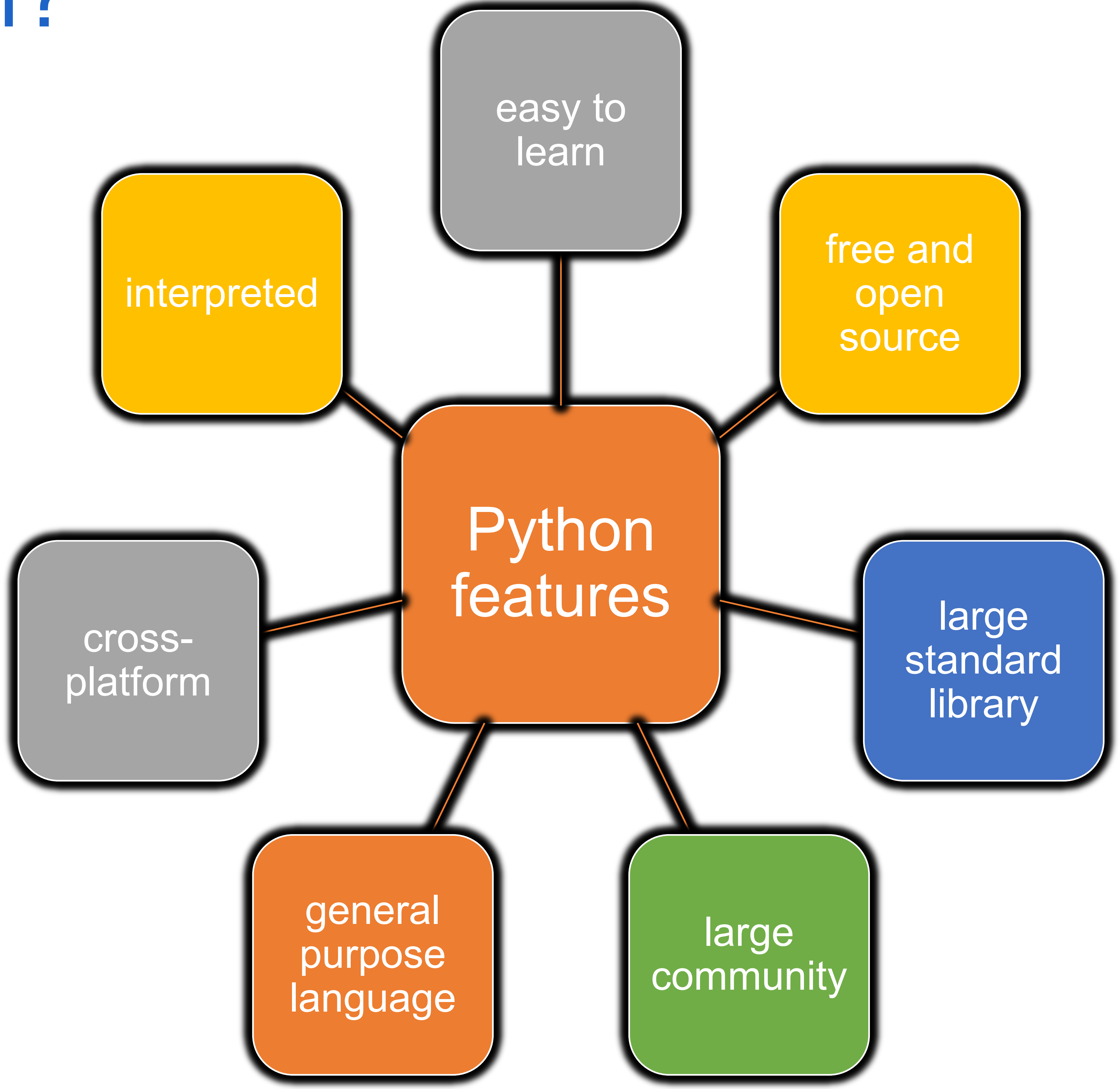


UBER
Quora
Spotify
Google
bitly
hipmunk
reddit
Pinterest
instacart
python™
NASA
YouTube
Dropbox
NETFLIX
Instagram

GHENT UNIVERSITY

LCT DRIVING CHEMICAL TECHNOLOGY

Why Python?



Python features

easy to learn

free and open source

large standard library

large community

general purpose language

cross-platform

interpreted

Python vs. Fortran and Matlab

		Python	Fortran	Matlab
→	free?	✓	✓	✗
→	open source?	✗	✗	✗
→	interpreted?	✓	✓	✓
→	all in one?	✓	✓	✓
→	support from company?	✓	✓	✓
✓	easy import/export data?	✓	✓	✓
✓	easy data visualization?	✓	✓	✓

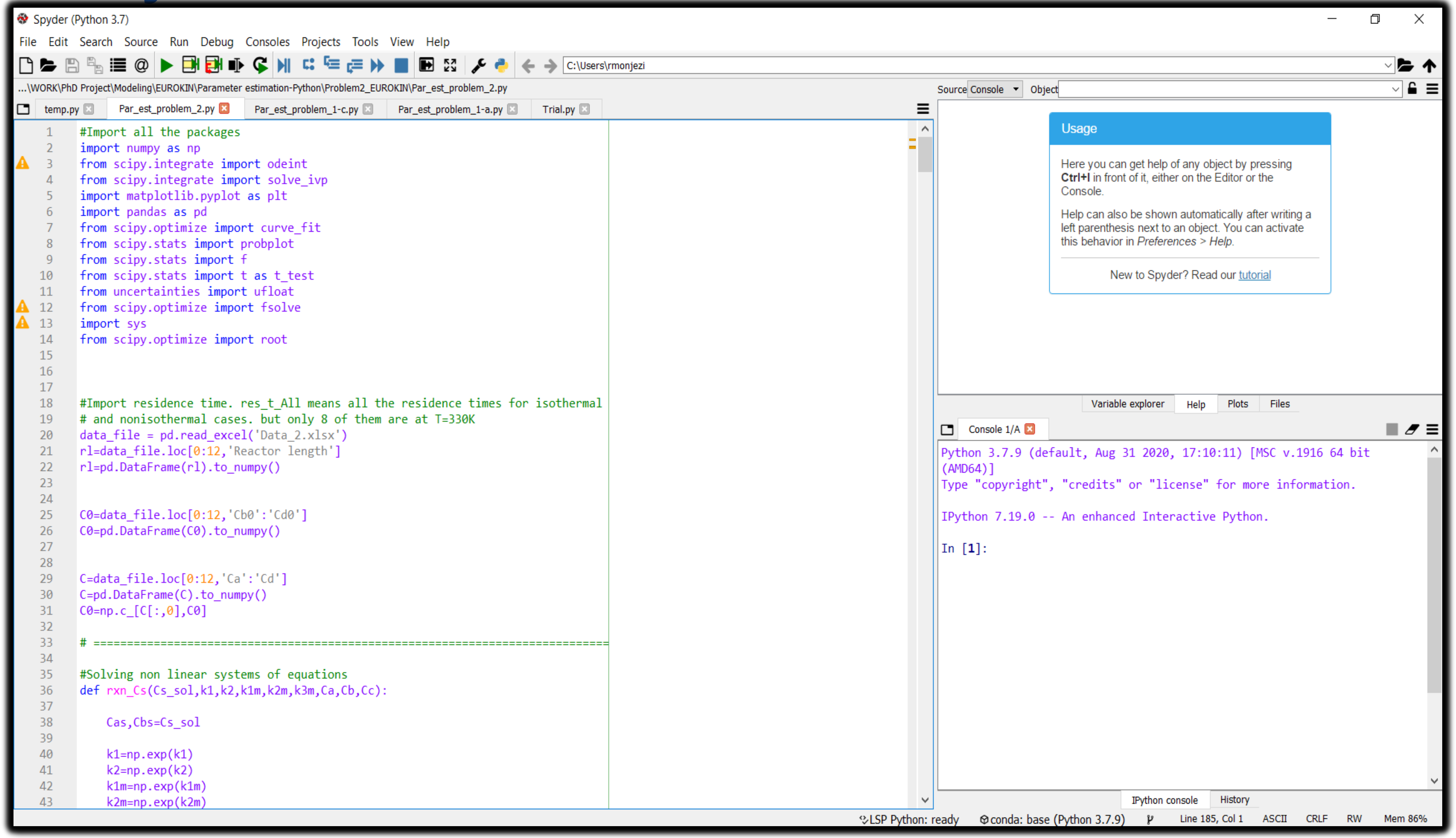
overview

- ✓ introduction to Python
- How to work with Python?
  - IDE
  - code structure
  - help
- Python for numerical programming
- EUROKIN parameter estim
- complex system
- summary

IDEs for Python

IDLE
SPYDER
PyCharm
eric
vim
Atom
jupyter
Anaconda
PyDev
Thonny
Python IDE for beginners

IDEs for Python



```python
#Import all the packages
import numpy as np
from scipy.integrate import odeint
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import curve_fit
from scipy.stats import probplot
from scipy.stats import f
from scipy.stats import t as t_test
from uncertainties import ufloat
from scipy.optimize import fsolve
import sys
from scipy.optimize import root



#Import residence time. res_t_All means all the residence times for isothermal
# and nonisothermal cases. but only 8 of them are at T=330K
data_file = pd.read_excel('Data_2.xlsx')
rl=data_file.loc[0:12,'Reactor length']
rl=pd.DataFrame(rl).to_numpy()


C0=data_file.loc[0:12,'Cb0':'Cd0']
C0=pd.DataFrame(C0).to_numpy()


C=data_file.loc[0:12,'Ca':'Cd']
C=pd.DataFrame(C).to_numpy()
C0=np.c_[C[:,0],C0]

# ============================================================================

#Solving non linear systems of equations
def rxn_Cs(Cs_sol,k1,k2,k1m,k2m,k3m,Ca,Cb,Cc):

    Cas,Cbs=Cs_sol

    k1=np.exp(k1)
    k2=np.exp(k2)
    k1m=np.exp(k1m)
    k2m=np.exp(k2m)
```

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

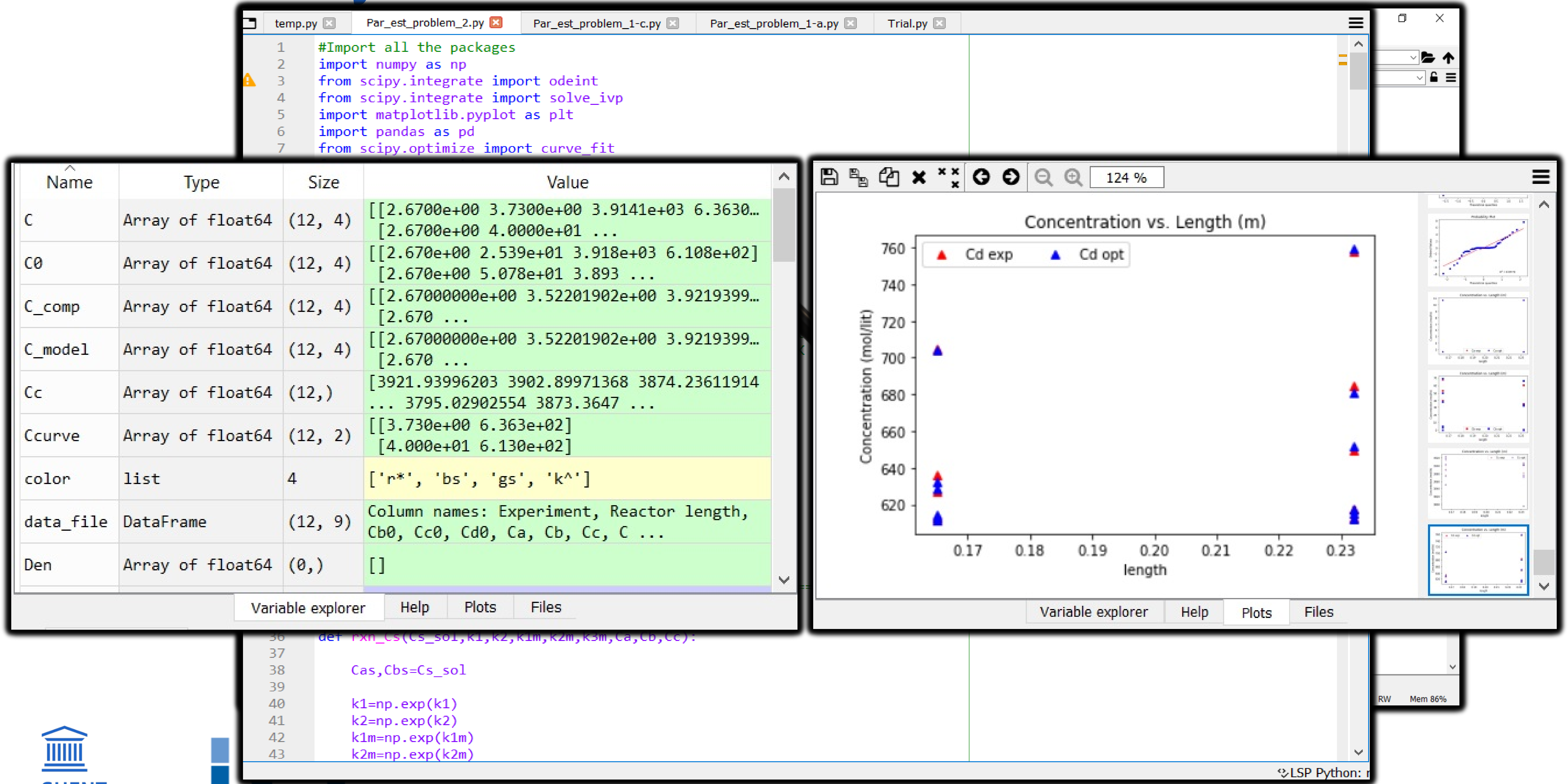
```
Python 3.7.9 (default, Aug 31 2020, 17:10:11) [MSC v.1916 64 bit
(AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.19.0 -- An enhanced Interactive Python.

In [1]:
```

LSP Python: ready   conda: base (Python 3.7.9)   Line 185, Col 1   ASCII   CRLF   RW   Mem 86%

GHENT UNIVERSITY

DRIVING CHEMICAL TECHNOLOGY

IDEs for Python



```python
#Import all the packages
import numpy as np
from scipy.integrate import odeint
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt
import pandas as pd
from scipy.optimize import curve_fit
```

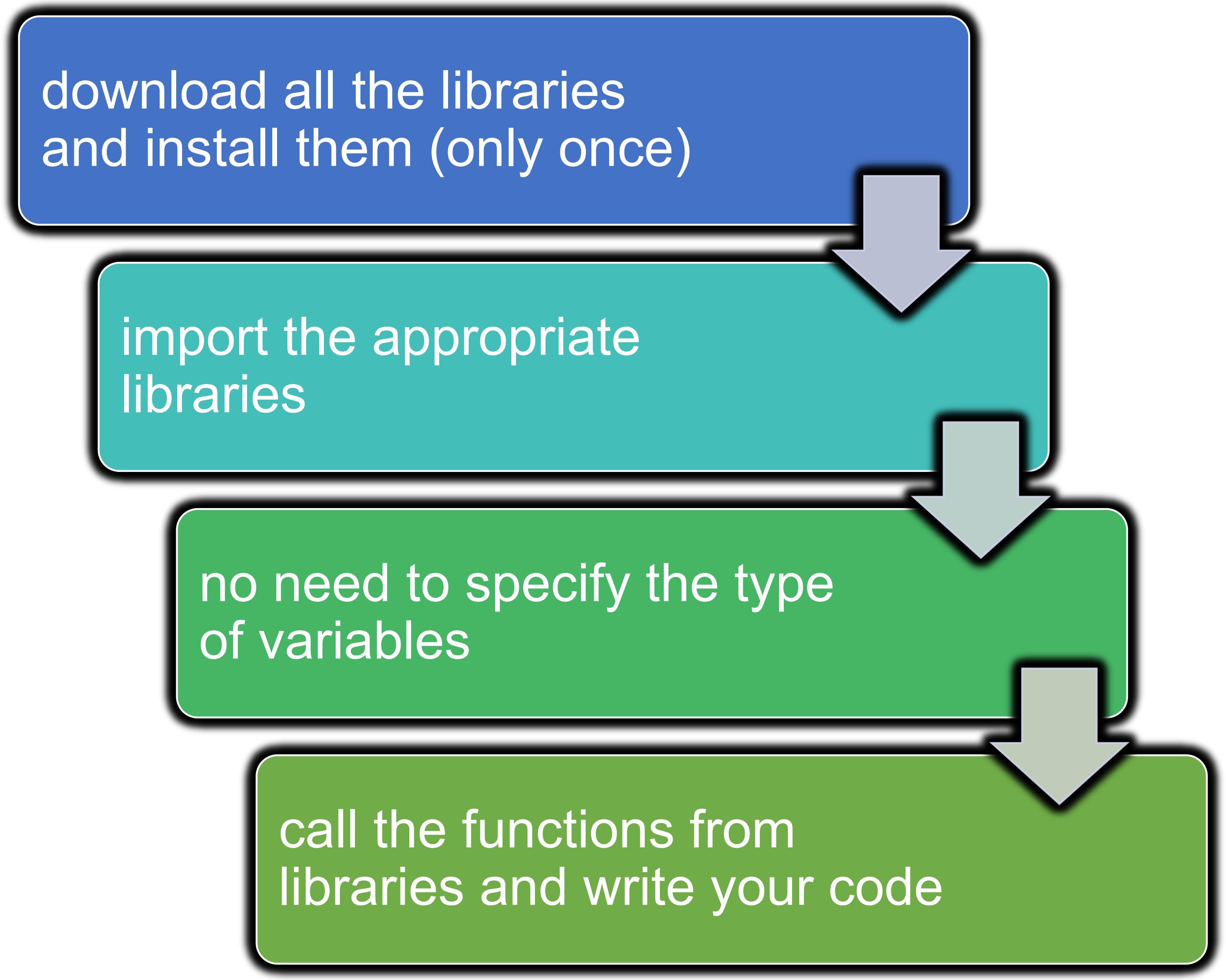
Name	Type	Size	Value
C	Array of float64	(12, 4)	[[2.6700e+00 3.7300e+00 3.9141e+03 6.3630… [2.6700e+00 4.0000e+01 ...
C0	Array of float64	(12, 4)	[[2.670e+00 2.539e+01 3.918e+03 6.108e+02] [2.670e+00 5.078e+01 3.893 ...
C_comp	Array of float64	(12, 4)	[[2.67000000e+00 3.52201902e+00 3.9219399… [2.670 ...
C_model	Array of float64	(12, 4)	[[2.67000000e+00 3.52201902e+00 3.9219399… [2.670 ...
Cc	Array of float64	(12,)	[3921.93996203 3902.89971368 3874.23611914 ... 3795.02902554 3873.3647 ...
Ccurve	Array of float64	(12, 2)	[[3.730e+00 6.363e+02] [4.000e+01 6.130e+02]]
color	list	4	['r*', 'bs', 'gs', 'k^']
data_file	DataFrame	(12, 9)	Column names: Experiment, Reactor length, Cb0, Cc0, Cd0, Ca, Cb, Cc, C ...
Den	Array of float64	(0,)	[]

```python
def rxn_Cs(Cs_sol,k1,k2,k1m,k2m,k3m,Ca,Cb,Cc):

    Cas,Cbs=Cs_sol

    k1=np.exp(k1)
    k2=np.exp(k2)
    k1m=np.exp(k1m)
    k2m=np.exp(k2m)
```

GHENT UNIVERSITY

DRIVING CHEMICAL TECHNOLOGY

Code in Python



download all the libraries and install them (only once)

import the appropriate libraries

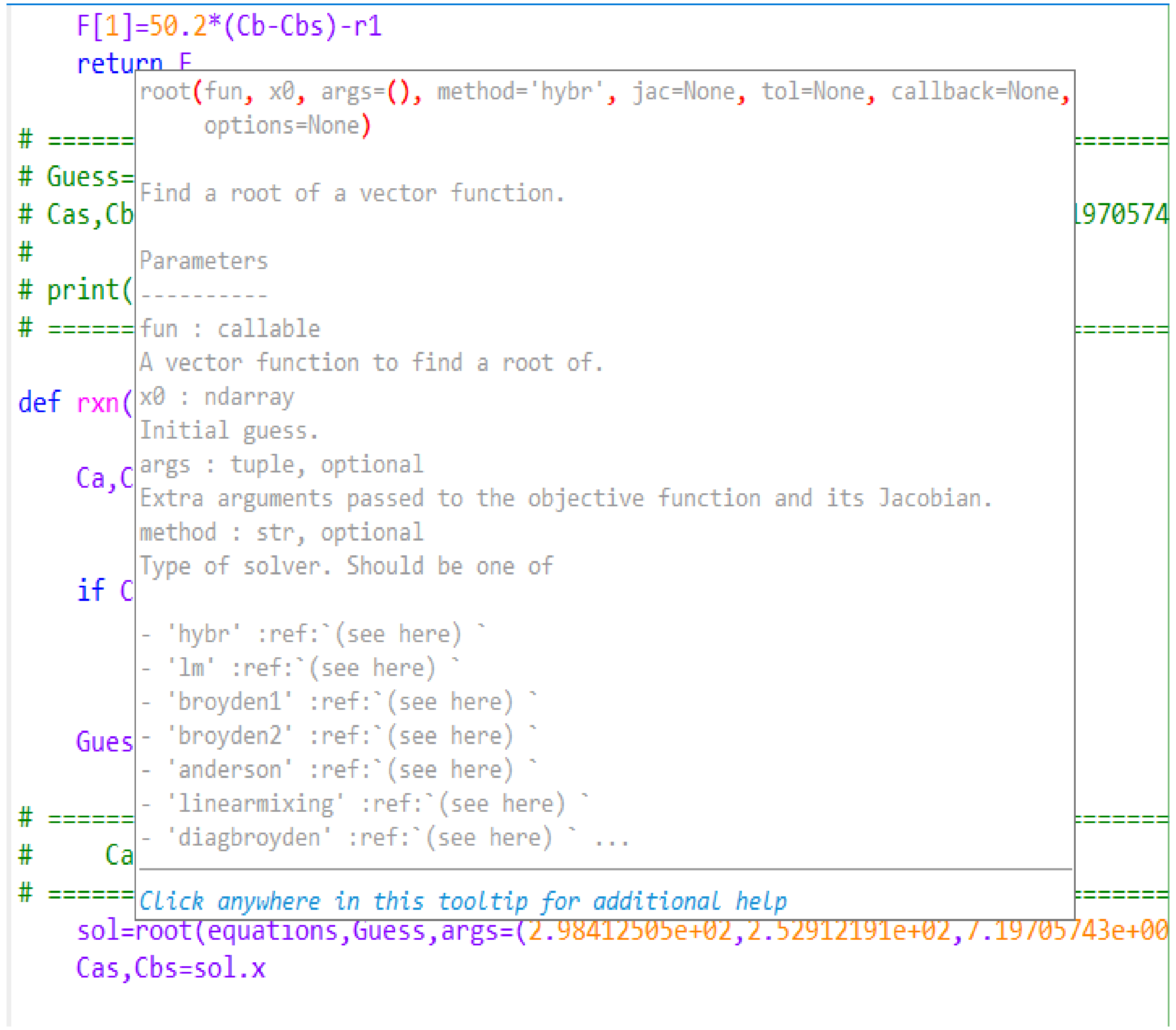
no need to specify the type of variables

call the functions from libraries and write your code

```
#Import the numpy package. np is
#the abbreviation for numpy.
import numpy as np
#Creating a matrix
M=np.array([[1,2,3],[4,5,6]])
print(M)
```

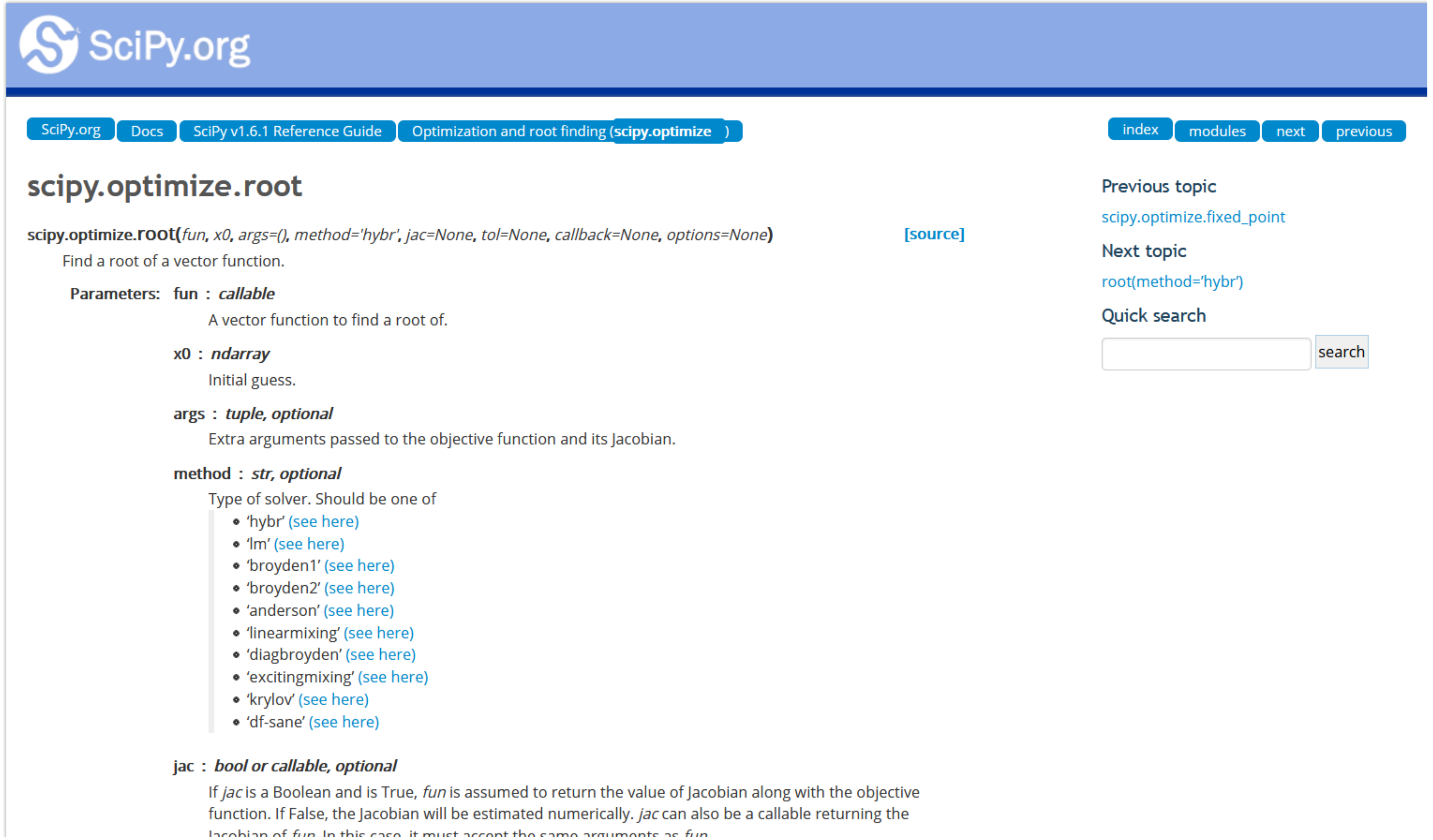
```
[[1,2,3]
[4,5,6]]
```

help function



```
    F[1]=50.2*(Cb-Cbs)-r1
    return F

# ======                                                         =======
# Guess=
# Cas,Cb                                                        1970574
#
# print(
# ======                                                         =======

def rxn(

    Ca,C

    if C



    Gues

# ======                                                         =======
#     Ca
# ======                                                         =======
    sol=root(equations,Guess,args=(2.98412505e+02,2.52912191e+02,7.19705743e+00
    Cas,Cbs=sol.x
```

```
root(fun, x0, args=(), method='hybr', jac=None, tol=None, callback=None,
     options=None)

Find a root of a vector function.

Parameters
----------
fun : callable
A vector function to find a root of.
x0 : ndarray
Initial guess.
args : tuple, optional
Extra arguments passed to the objective function and its Jacobian.
method : str, optional
Type of solver. Should be one of

- 'hybr' :ref:`(see here) `
- 'lm' :ref:`(see here) `
- 'broyden1' :ref:`(see here) `
- 'broyden2' :ref:`(see here) `
- 'anderson' :ref:`(see here) `
- 'linearmixing' :ref:`(see here) `
- 'diagbroyden' :ref:`(see here) ` ...

Click anywhere in this tooltip for additional help
```



SciPy.org
SciPy.org
Docs
SciPy v1.6.1 Reference Guide
Optimization and root finding (scipy.optimize)
index
modules
next
previous

scipy.optimize.root

scipy.optimize.root(*fun*, *x0*, *args=()*, *method='hybr'*, *jac=None*, *tol=None*, *callback=None*, *options=None*) [source]

Find a root of a vector function.

Parameters: **fun** : *callable*

A vector function to find a root of.

x0 : *ndarray*

Initial guess.

args : *tuple, optional*

Extra arguments passed to the objective function and its Jacobian.

method : *str, optional*

Type of solver. Should be one of

- 'hybr' (see here)
- 'lm' (see here)
- 'broyden1' (see here)
- 'broyden2' (see here)
- 'anderson' (see here)
- 'linearmixing' (see here)
- 'diagbroyden' (see here)
- 'excitingmixing' (see here)
- 'krylov' (see here)
- 'df-sane' (see here)

jac : *bool or callable, optional*

If *jac* is a Boolean and is True, *fun* is assumed to return the value of Jacobian along with the objective function. If False, the Jacobian will be estimated numerically. *jac* can also be a callable returning the Jacobian of *fun*. In this case, it must accept the same arguments as *fun*.

Previous topic

scipy.optimize.fixed_point

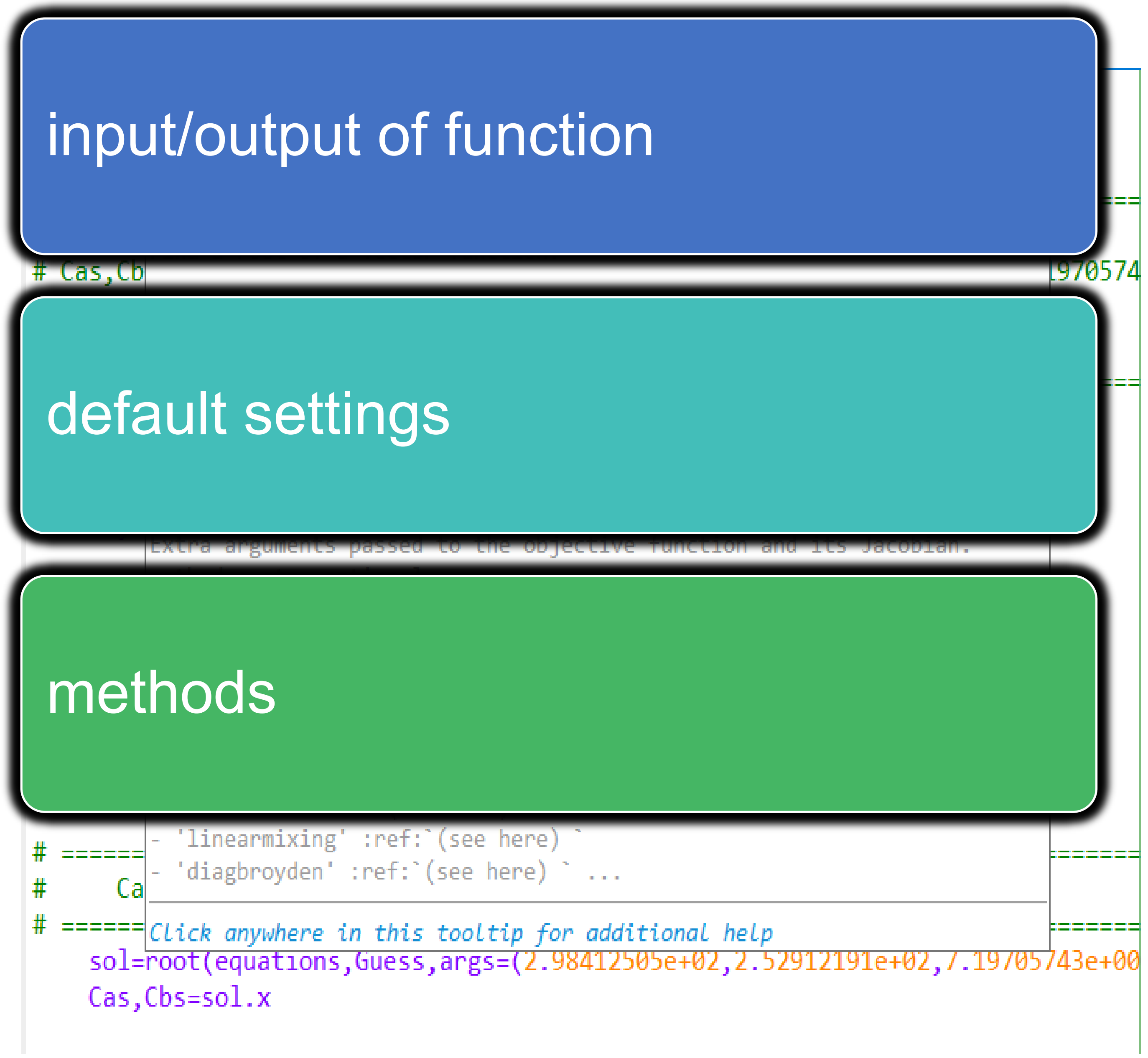
Next topic

root(method='hybr')

Quick search

search

help function

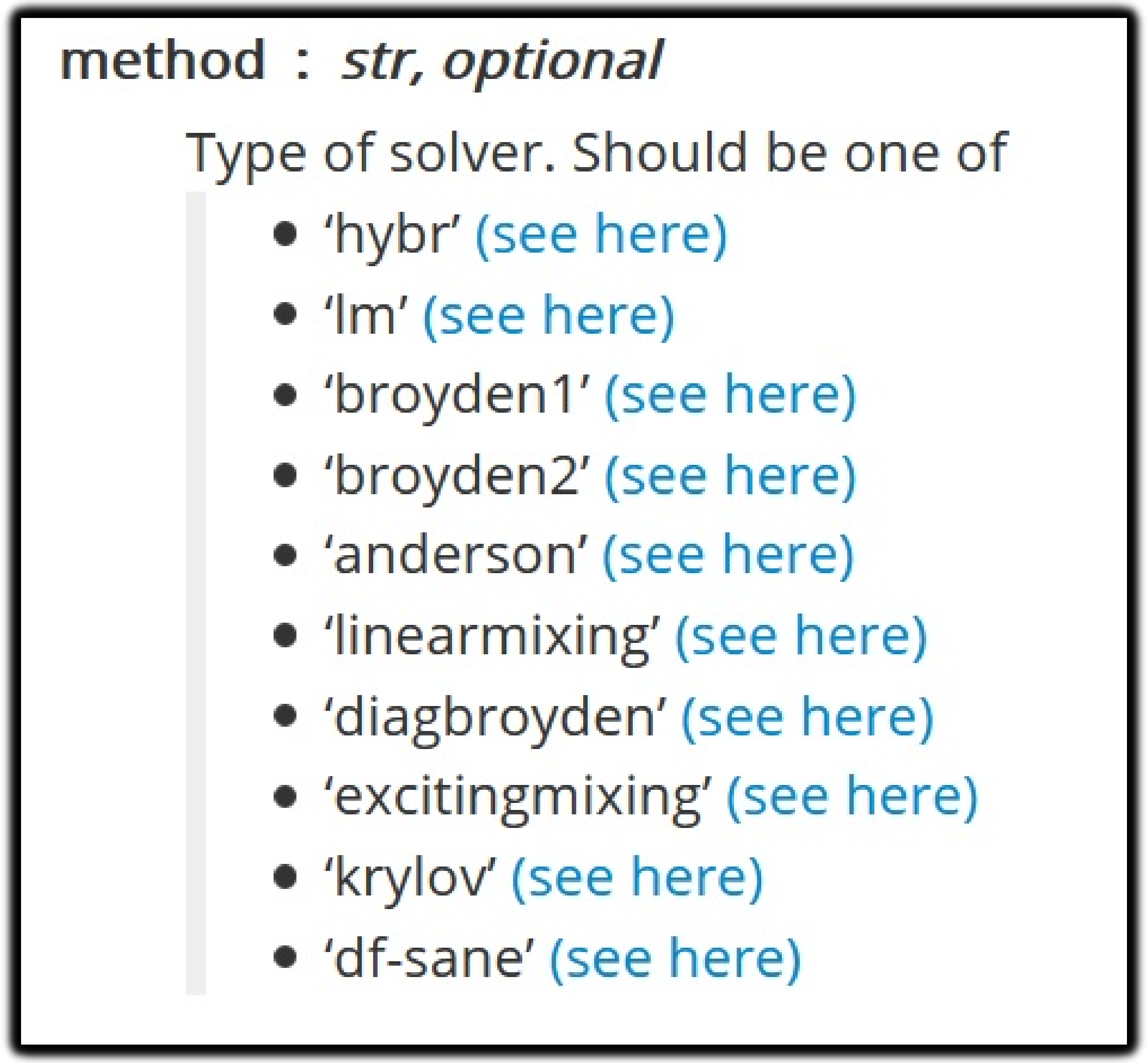


input/output of function

default settings

methods

```
# Cas,Cb                                                                      970574
   Extra arguments passed to the objective function and its Jacobian.
   - 'linearmixing' :ref:`(see here) `
# ======
   - 'diagbroyden' :ref:`(see here) ` ...
#     Ca
# ======
   Click anywhere in this tooltip for additional help
    sol=root(equations,Guess,args=(2.98412505e+02,2.52912191e+02,7.19705743e+00
    Cas,Cbs=sol.x
```



method : *str, optional*

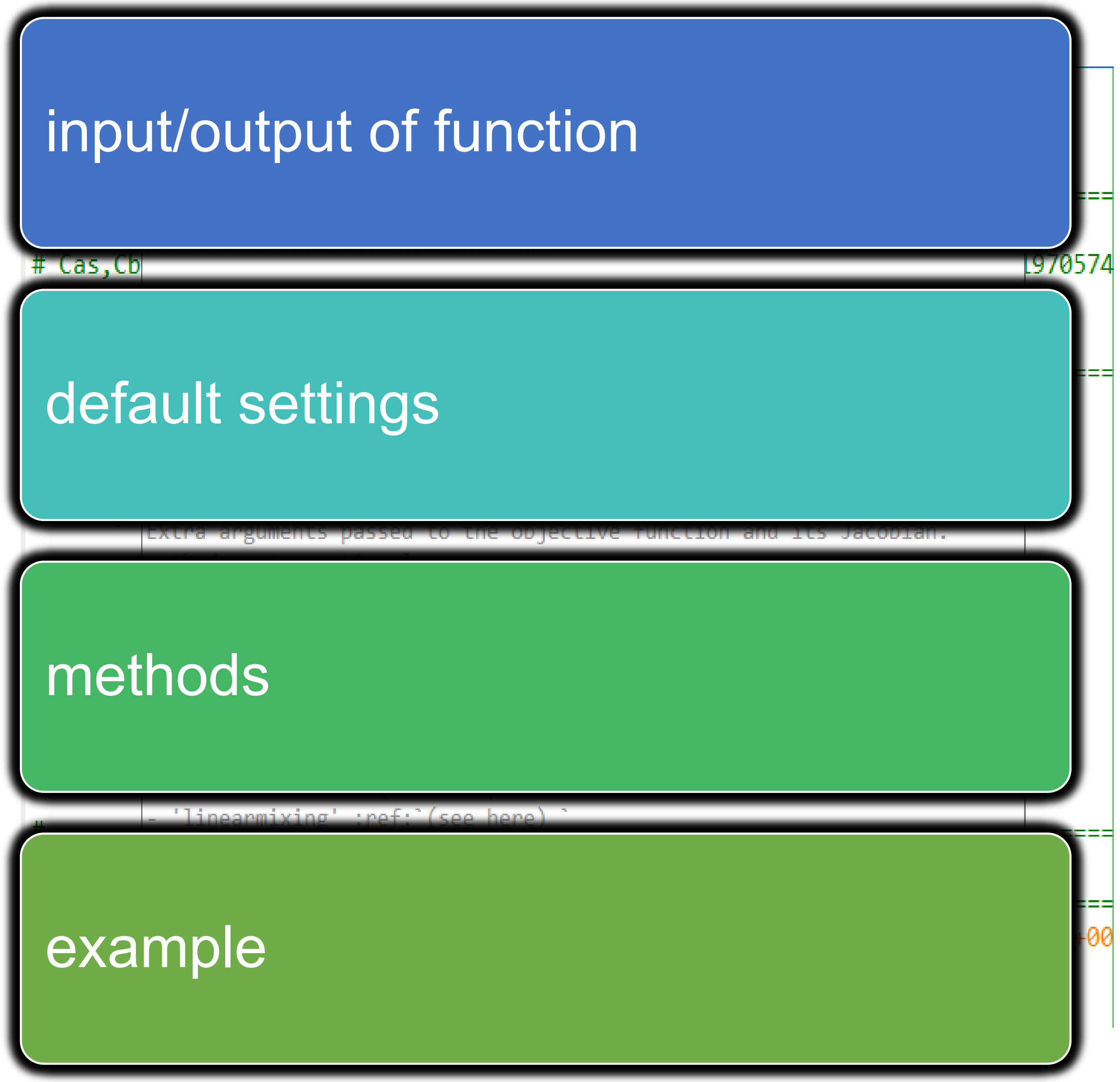
Type of solver. Should be one of

- 'hybr' (see here)
- 'lm' (see here)
- 'broyden1' (see here)
- 'broyden2' (see here)
- 'anderson' (see here)
- 'linearmixing' (see here)
- 'diagbroyden' (see here)
- 'excitingmixing' (see here)
- 'krylov' (see here)
- 'df-sane' (see here)

GHENT UNIVERSITY

LCT DRIVING CHEMICAL TECHNOLOGY

help function



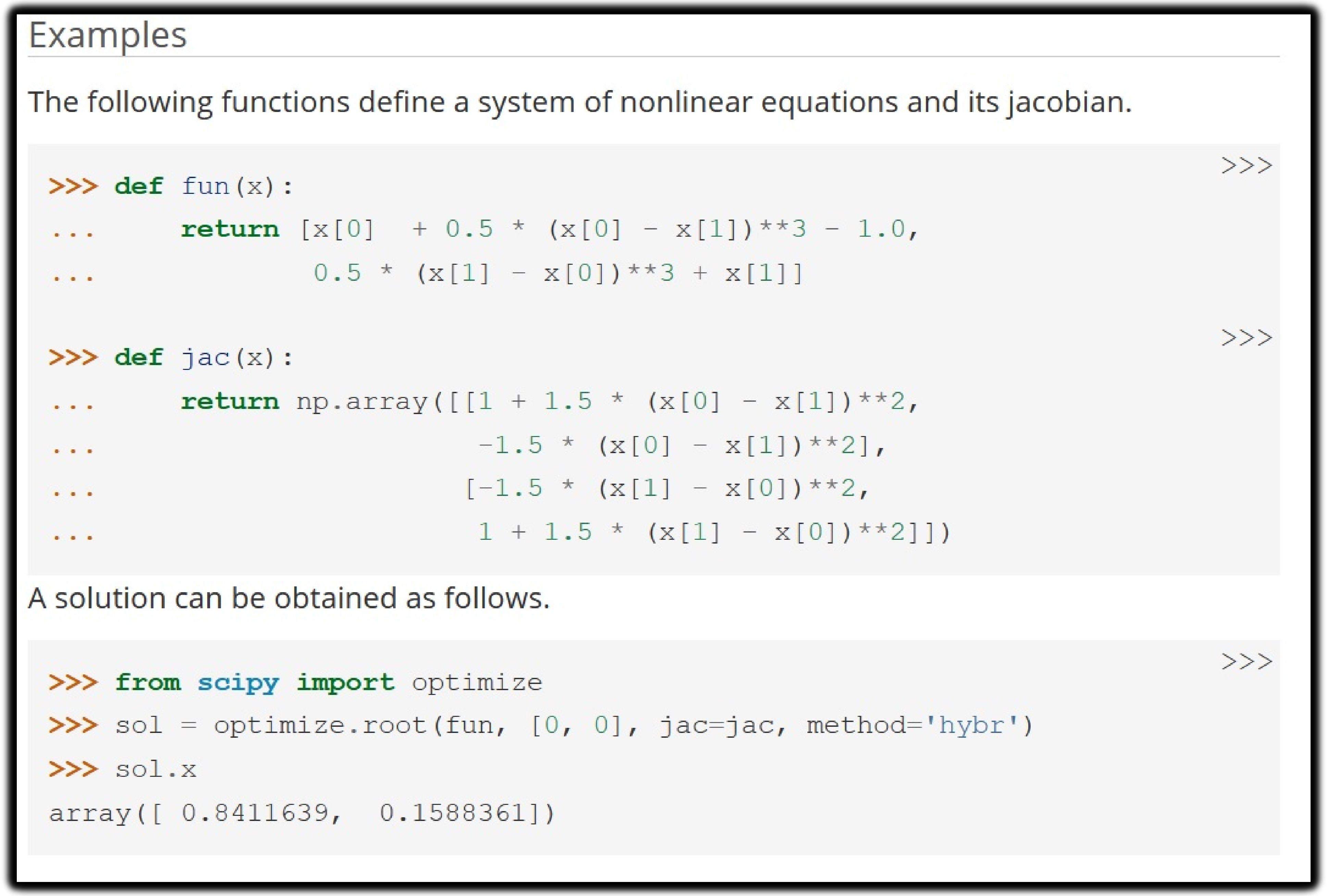
input/output of function

default settings

methods

example

# Cas,Cb
970574
Extra arguments passed to the objective function and its Jacobian.
- 'linearmixing' :ref:`(see here)`



Examples

The following functions define a system of nonlinear equations and its jacobian.

```
>>> def fun(x):
...     return [x[0]  + 0.5 * (x[0] - x[1])**3 - 1.0,
...             0.5 * (x[1] - x[0])**3 + x[1]]

>>> def jac(x):
...     return np.array([[1 + 1.5 * (x[0] - x[1])**2,
...                       -1.5 * (x[0] - x[1])**2],
...                      [-1.5 * (x[1] - x[0])**2,
...                       1 + 1.5 * (x[1] - x[0])**2]])
```

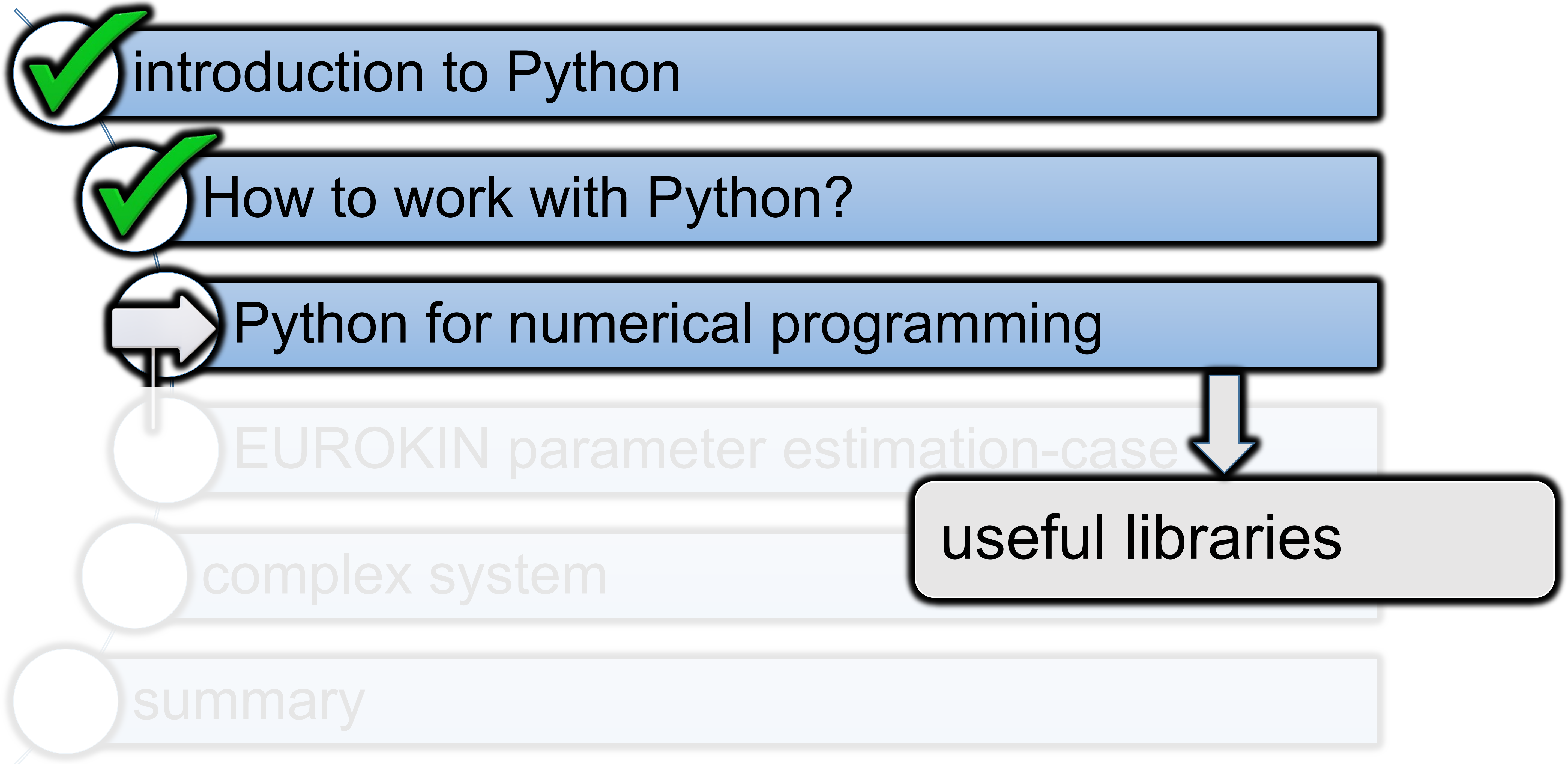
A solution can be obtained as follows.

```
>>> from scipy import optimize
>>> sol = optimize.root(fun, [0, 0], jac=jac, method='hybr')
>>> sol.x
array([ 0.8411639,  0.1588361])
```

GHENT UNIVERSITY

LCT DRIVING CHEMICAL TECHNOLOGY

overview



introduction to Python

How to work with Python?

Python for numerical programming

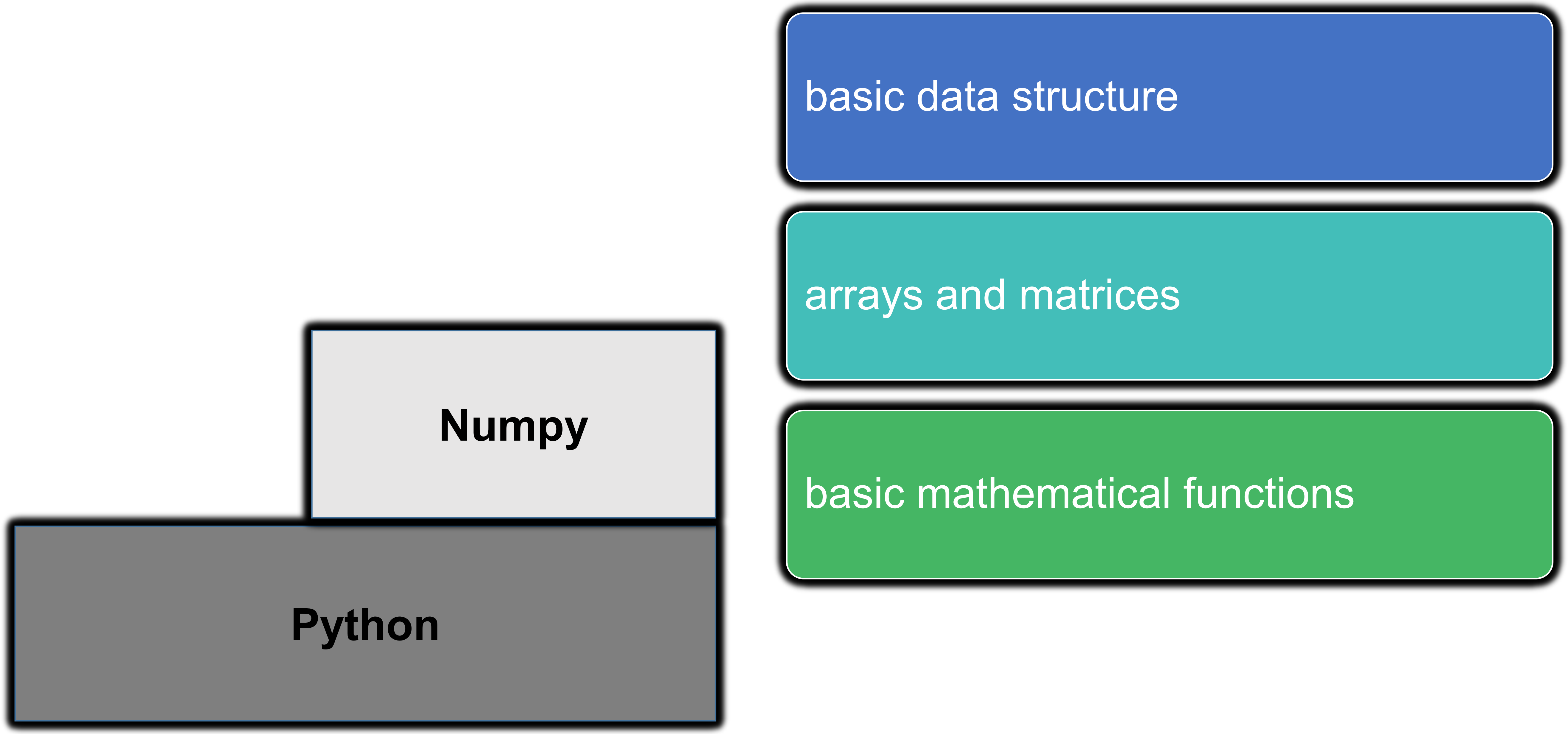
EUROKIN parameter estimation-case

complex system

summary

useful libraries

Python for numerical programming



Numpy

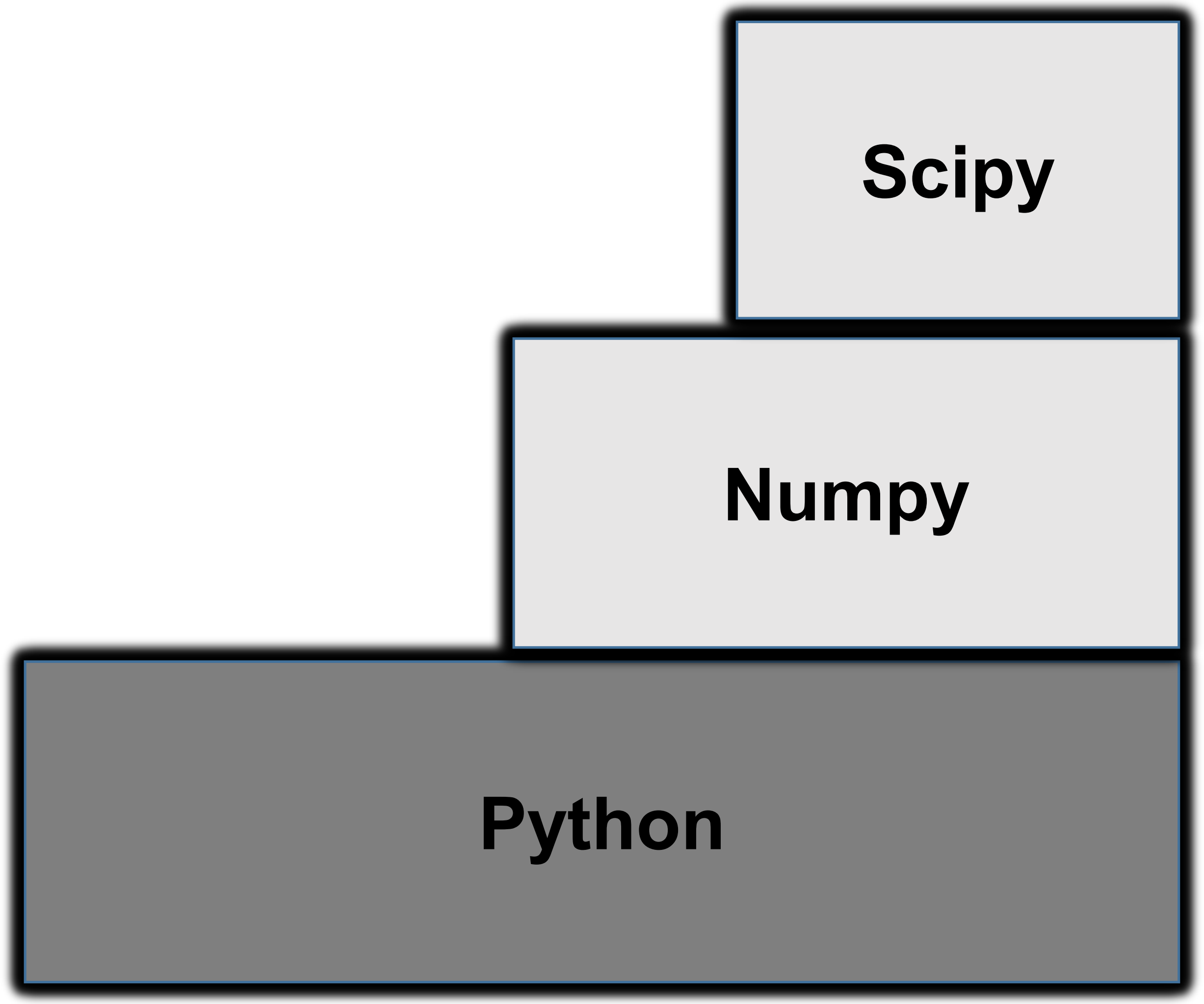
Python

basic data structure

arrays and matrices

basic mathematical functions

Python for numerical programming



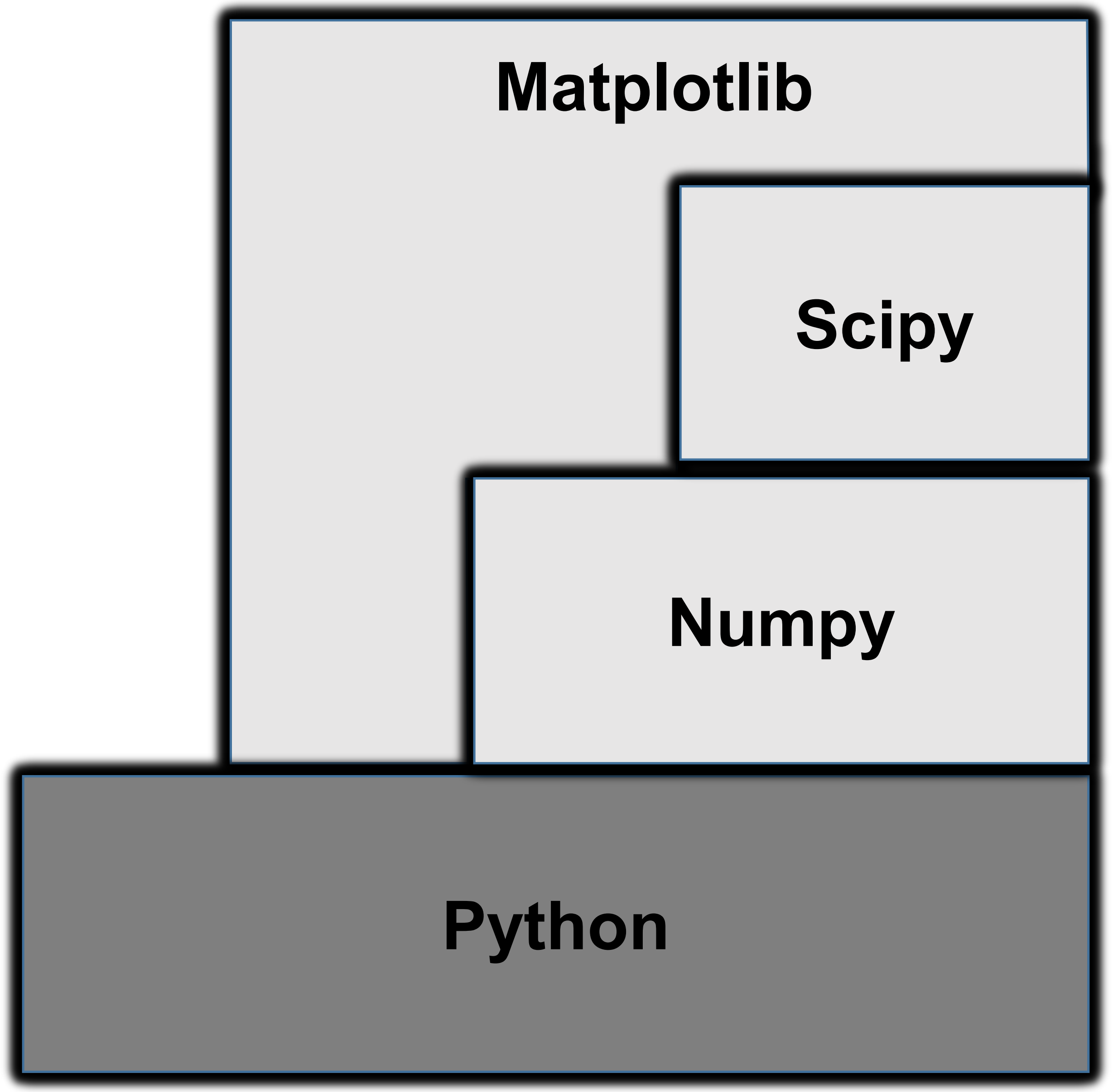
Scipy

Numpy

Python

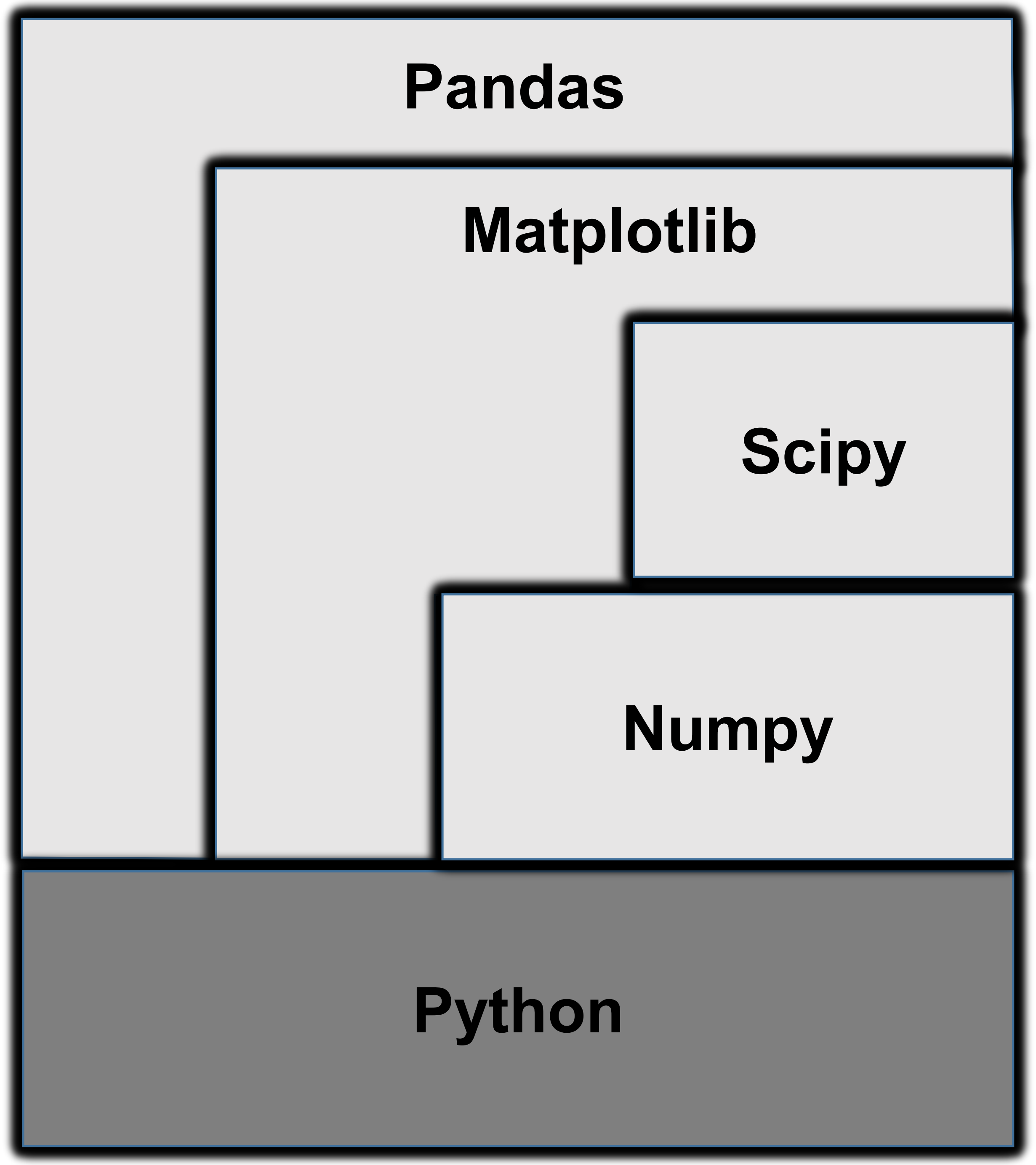
- integration
- optimization
- statistical calculation

Python for numerical programming



Data visualization

Python for numerical programming



Data structure

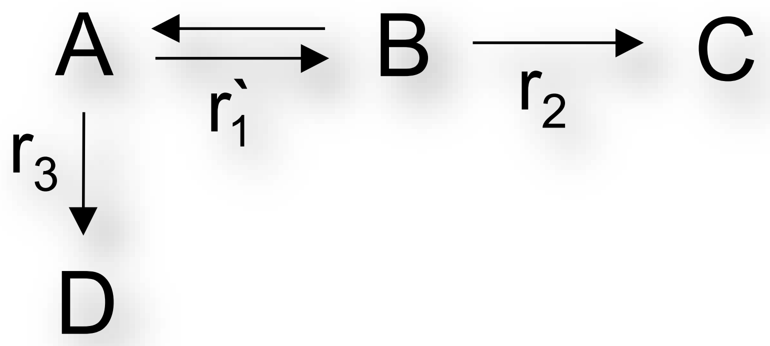
Import/export data

overview

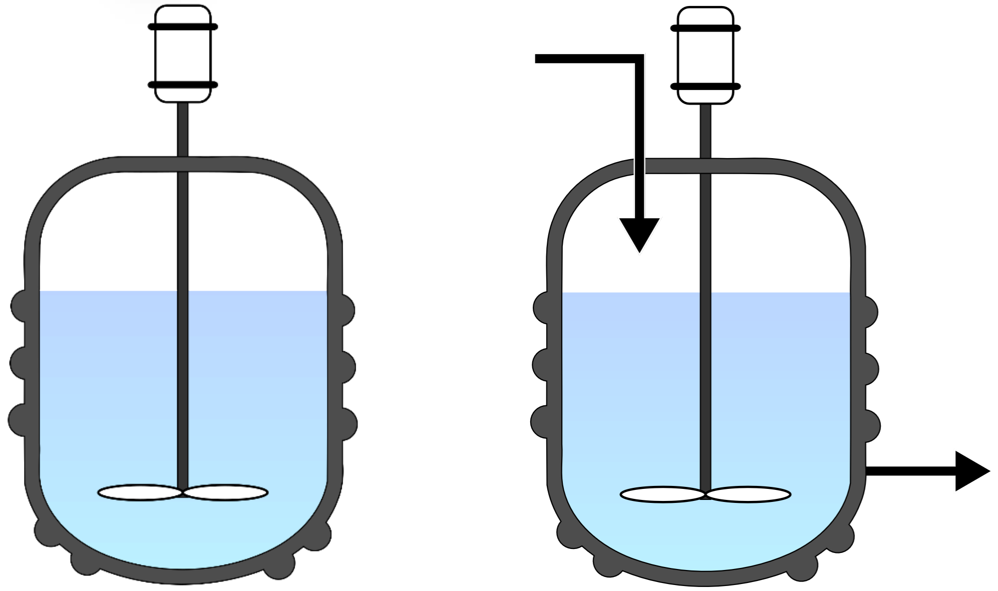
- ✓ introduction to Python
- ✓ How to work with Python?
- ✓ Python for numerical programming
- ➡ EUROKIN parameter estimation-case 1
  - problem statement
  - Solution procedure for different scenarios:
    - Isothermal batch reactor data
    - Non-isothermally assessment of batch reactor data
    - CSTR and batch reactor data
- Complex system
- Summary

problem statement

reaction?



reactors?



temperature?

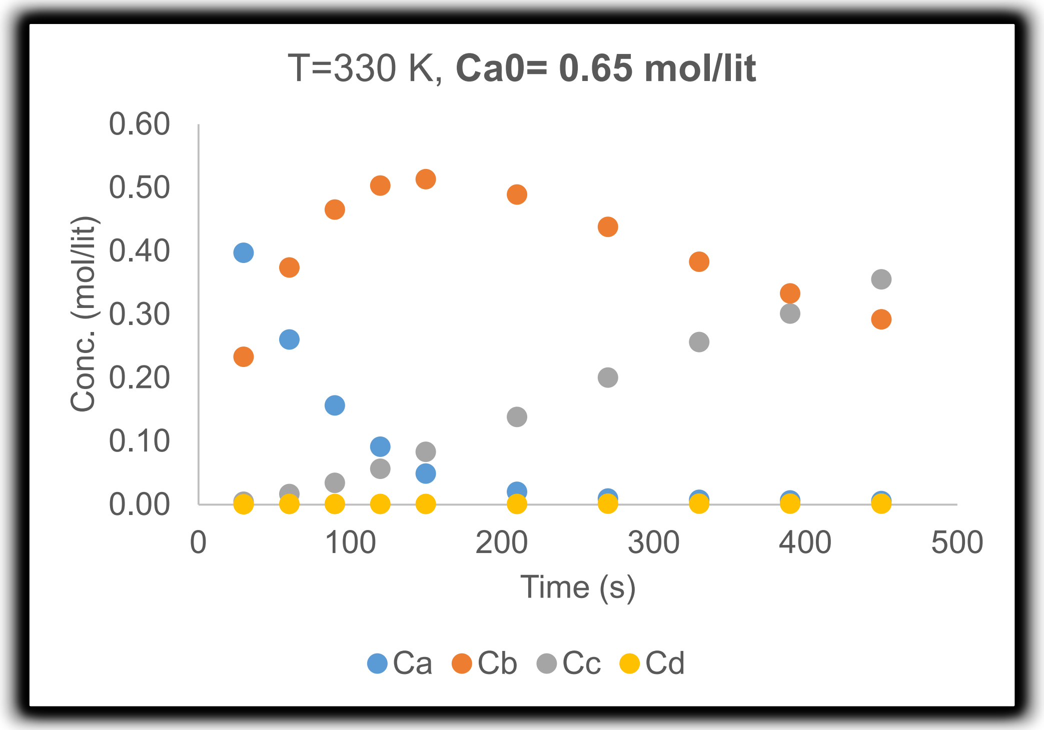
310 K

330 K

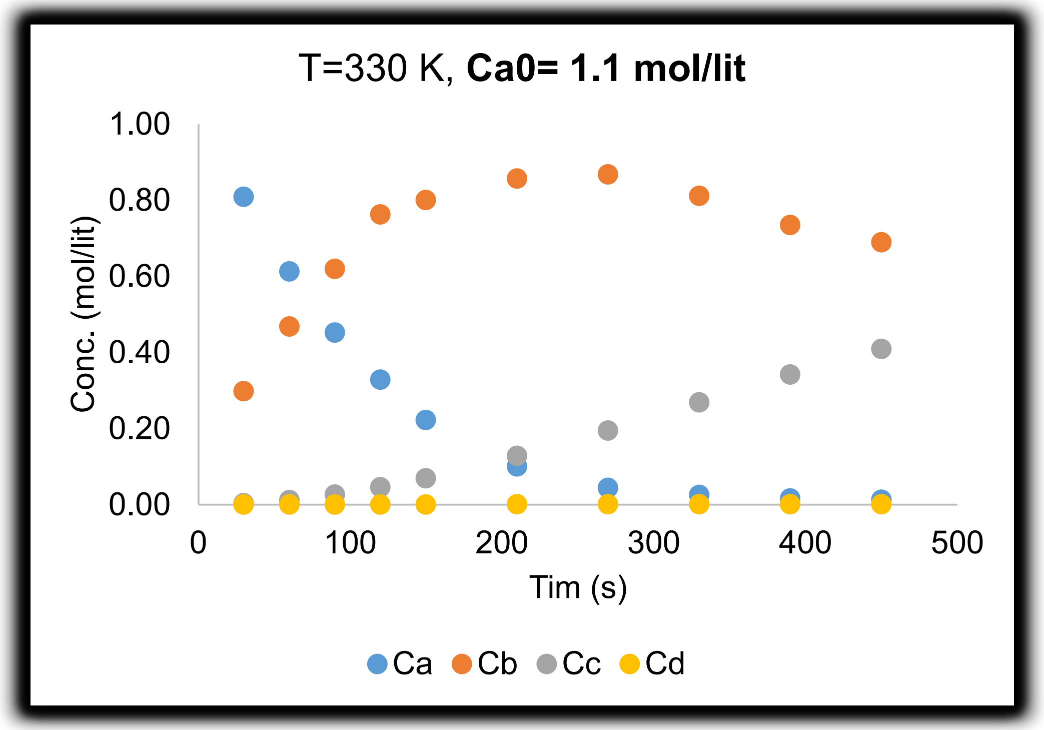
360 K

	part 1	part 2	part 3
batch reactor?	✓	✓	✗
CSTR?	✗	✗	✗
isothermal data?	✓	✗	✗
non-isothermal data?	✗	✗	✗

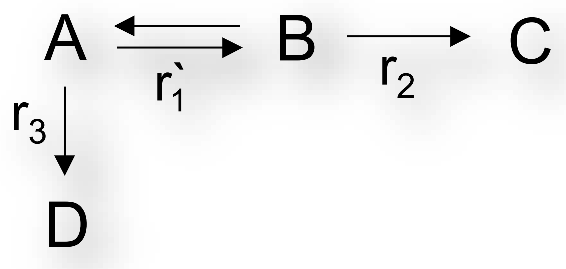
part1: batch reactor + isothermal data



T=330 K, **Ca0= 0.65 mol/lit**



T=330 K, **Ca0= 1.1 mol/lit**

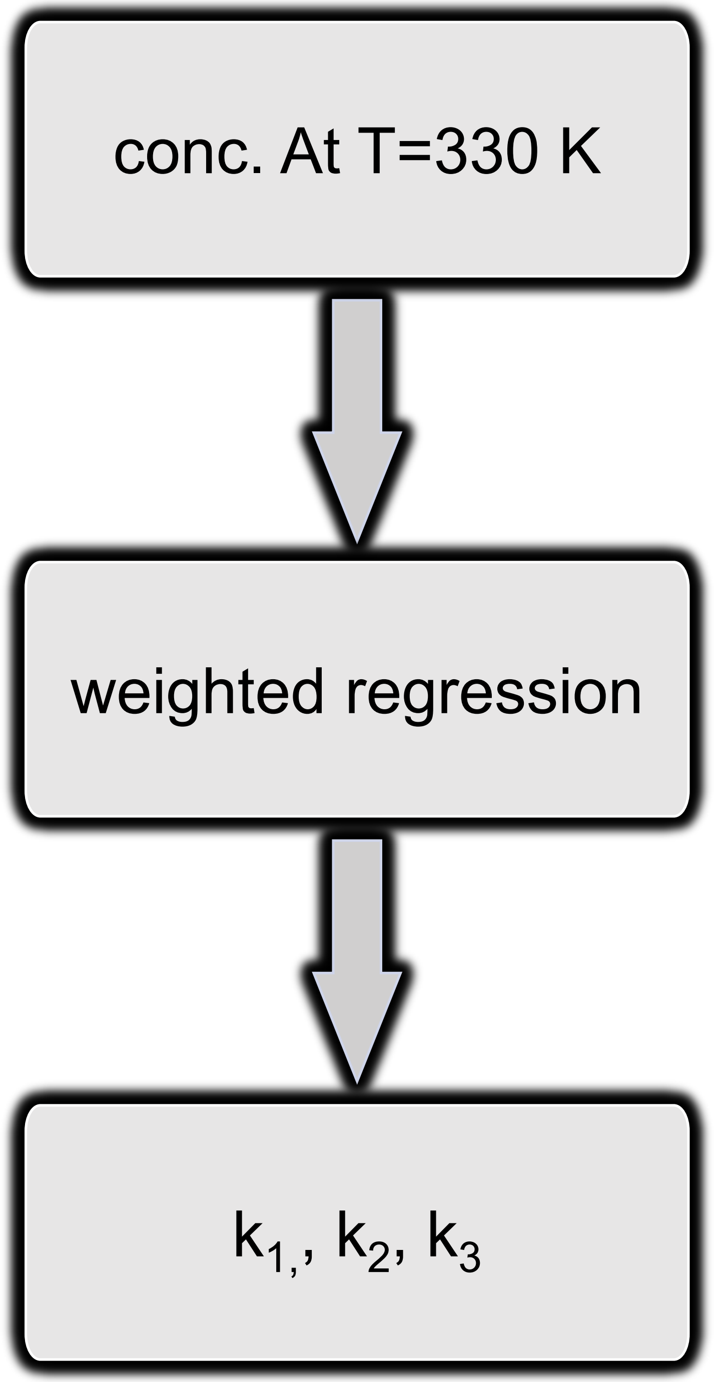


$$\frac{dC_A}{dt} = \frac{-k_1 K_A \left(C_A - \frac{C_B}{K_{eq1}}\right) - k_3 K_A C_A}{1 + K_A C_A + K_B C_B}$$

$$\frac{dC_C}{dt} = \frac{k_2 K_B C_B}{1 + K_A C_A + K_B C_B}$$

$$\frac{dC_D}{dt} = \frac{k_3 K_A C_A}{1 + K_A C_A + K_B C_B}$$

$$C_B = C_{A0} - C_A - C_C - C_D$$

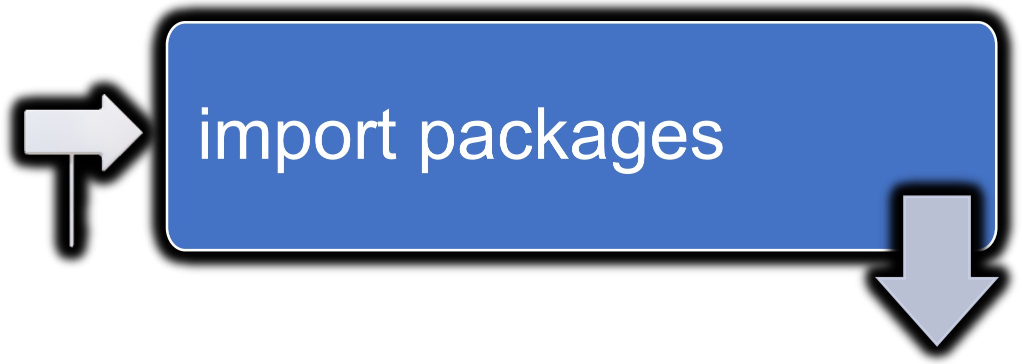


conc. At T=330 K

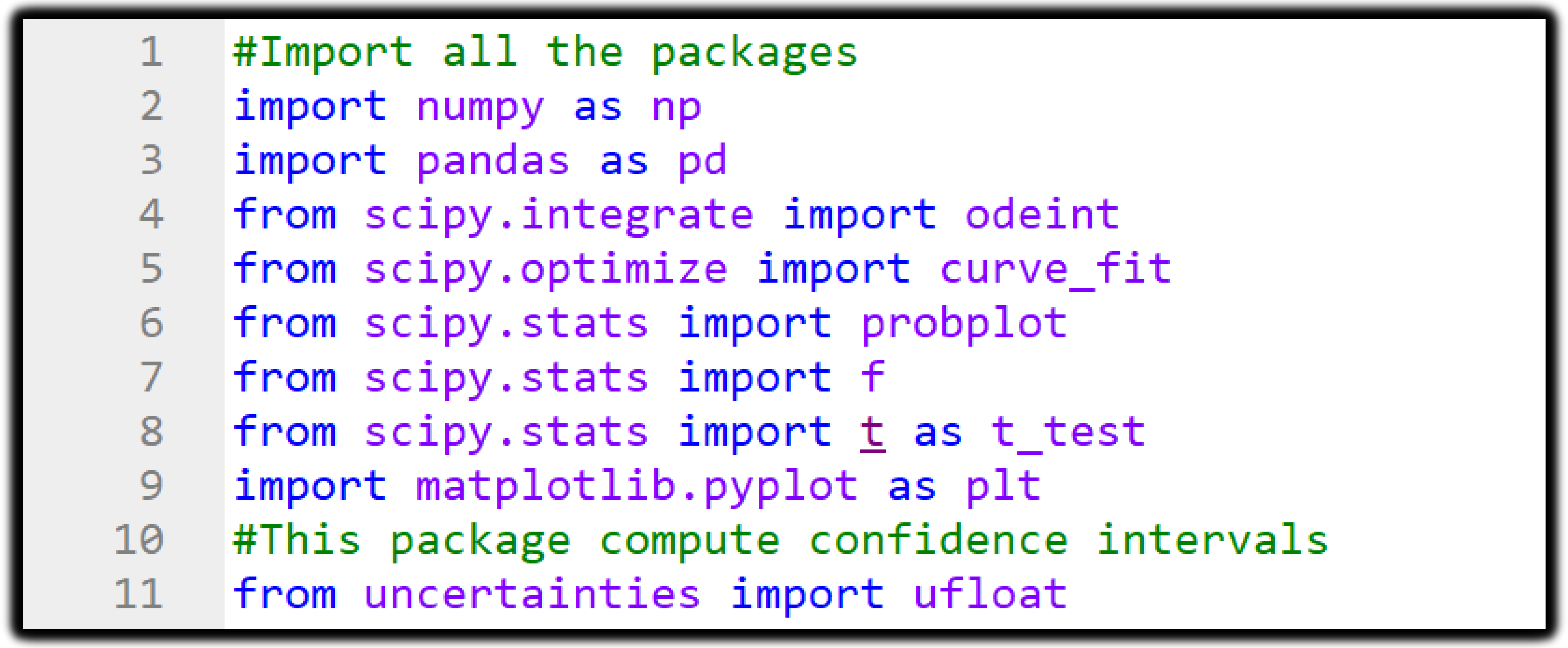
weighted regression

$k_{1,}$, k_2, k_3

part1: batch reactor + isothermal data (code)

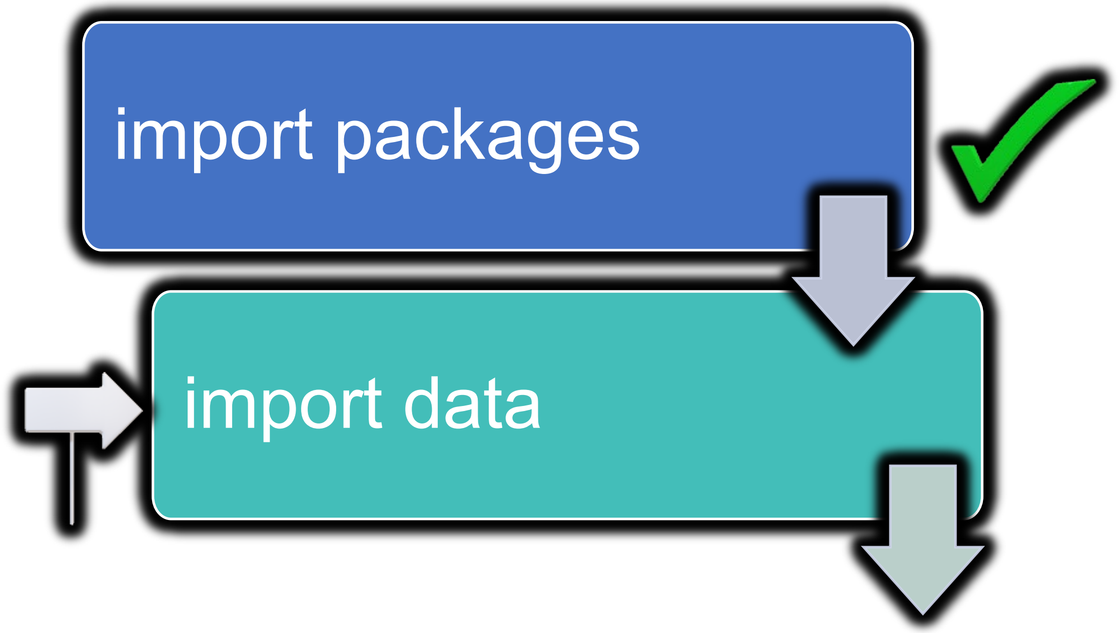


import packages



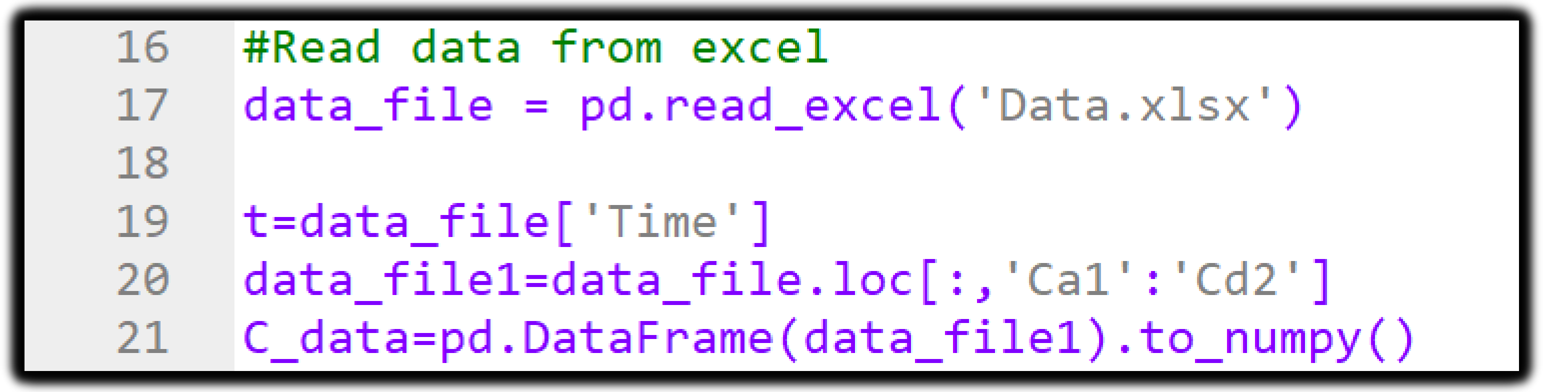
```python
#Import all the packages
import numpy as np
import pandas as pd
from scipy.integrate import odeint
from scipy.optimize import curve_fit
from scipy.stats import probplot
from scipy.stats import f
from scipy.stats import t as t_test
import matplotlib.pyplot as plt
#This package compute confidence intervals
from uncertainties import ufloat
```

part1: batch reactor + isothermal data (code)



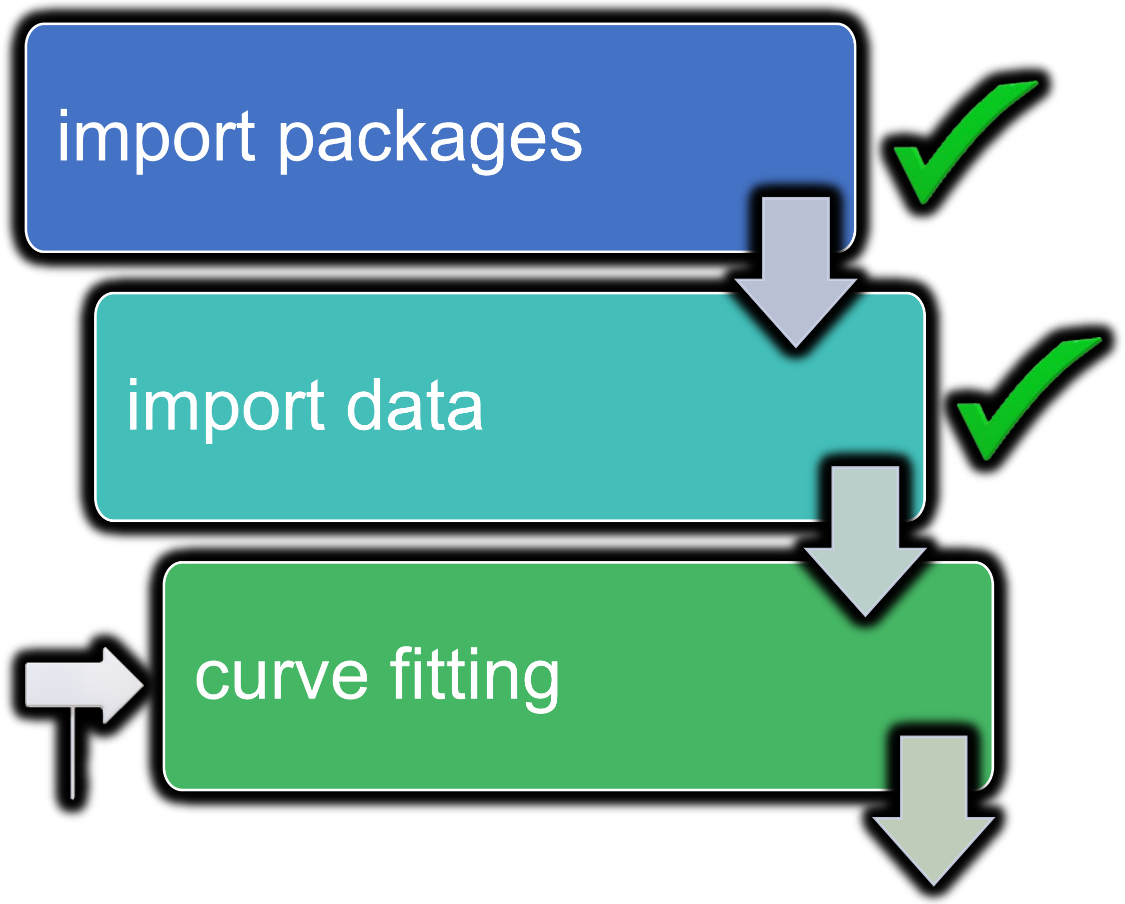
import packages

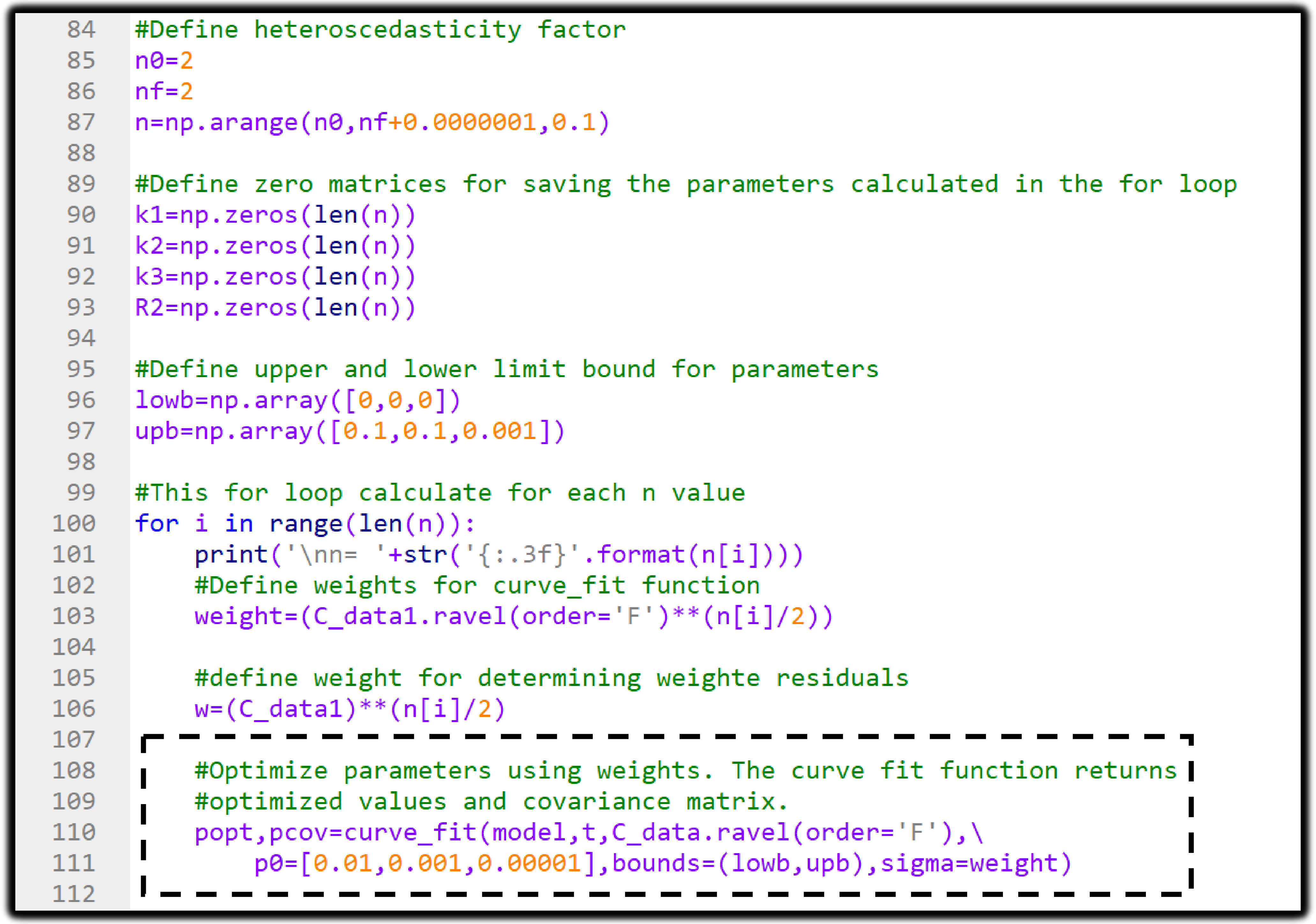
import data



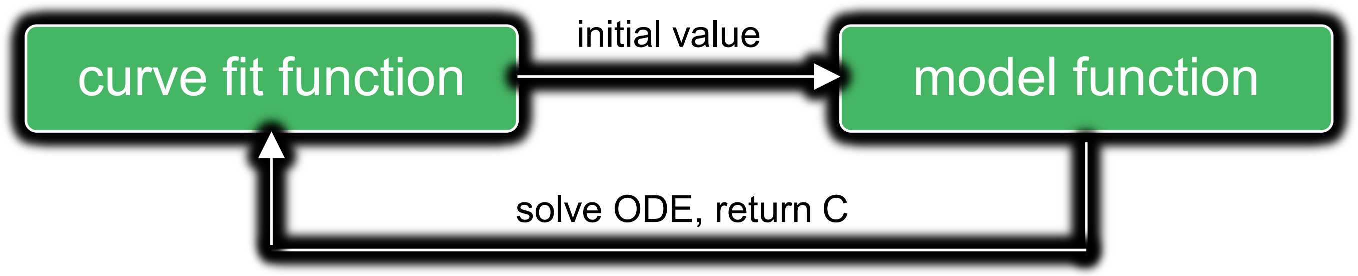
```
#Read data from excel
data_file = pd.read_excel('Data.xlsx')

t=data_file['Time']
data_file1=data_file.loc[:,'Ca1':'Cd2']
C_data=pd.DataFrame(data_file1).to_numpy()
```

part1: batch reactor + isothermal data (code)

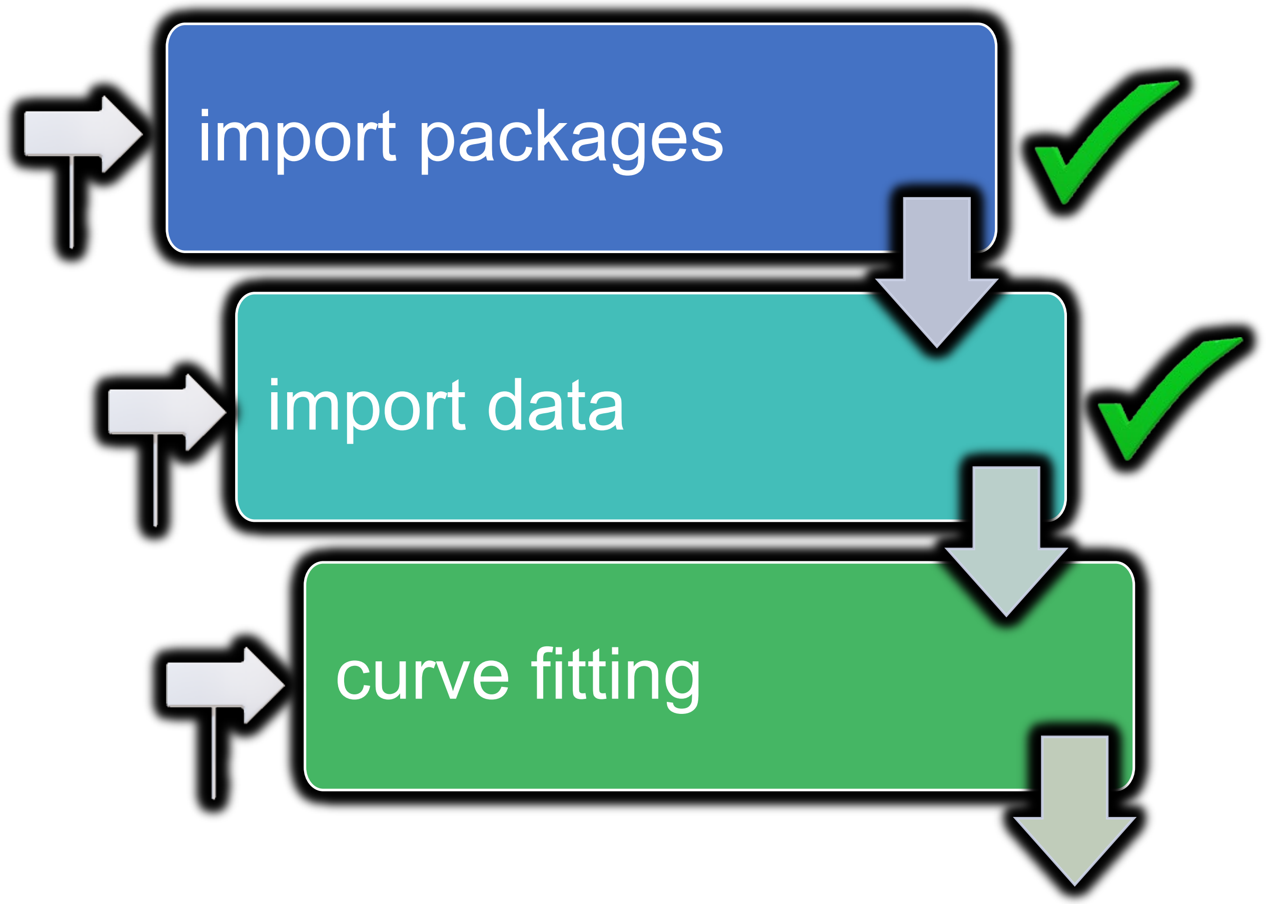




```python
#Define heteroscedasticity factor
n0=2
nf=2
n=np.arange(n0,nf+0.0000001,0.1)

#Define zero matrices for saving the parameters calculated in the for loop
k1=np.zeros(len(n))
k2=np.zeros(len(n))
k3=np.zeros(len(n))
R2=np.zeros(len(n))

#Define upper and lower limit bound for parameters
lowb=np.array([0,0,0])
upb=np.array([0.1,0.1,0.001])

#This for loop calculate for each n value
for i in range(len(n)):
    print('\nn= '+str('{:.3f}'.format(n[i])))
    #Define weights for curve_fit function
    weight=(C_data1.ravel(order='F')**(n[i]/2))

    #define weight for determining weighte residuals
    w=(C_data1)**(n[i]/2)

    #Optimize parameters using weights. The curve fit function returns
    #optimized values and covariance matrix.
    popt,pcov=curve_fit(model,t,C_data.ravel(order='F'),\
        p0=[0.01,0.001,0.00001],bounds=(lowb,upb),sigma=weight)

```



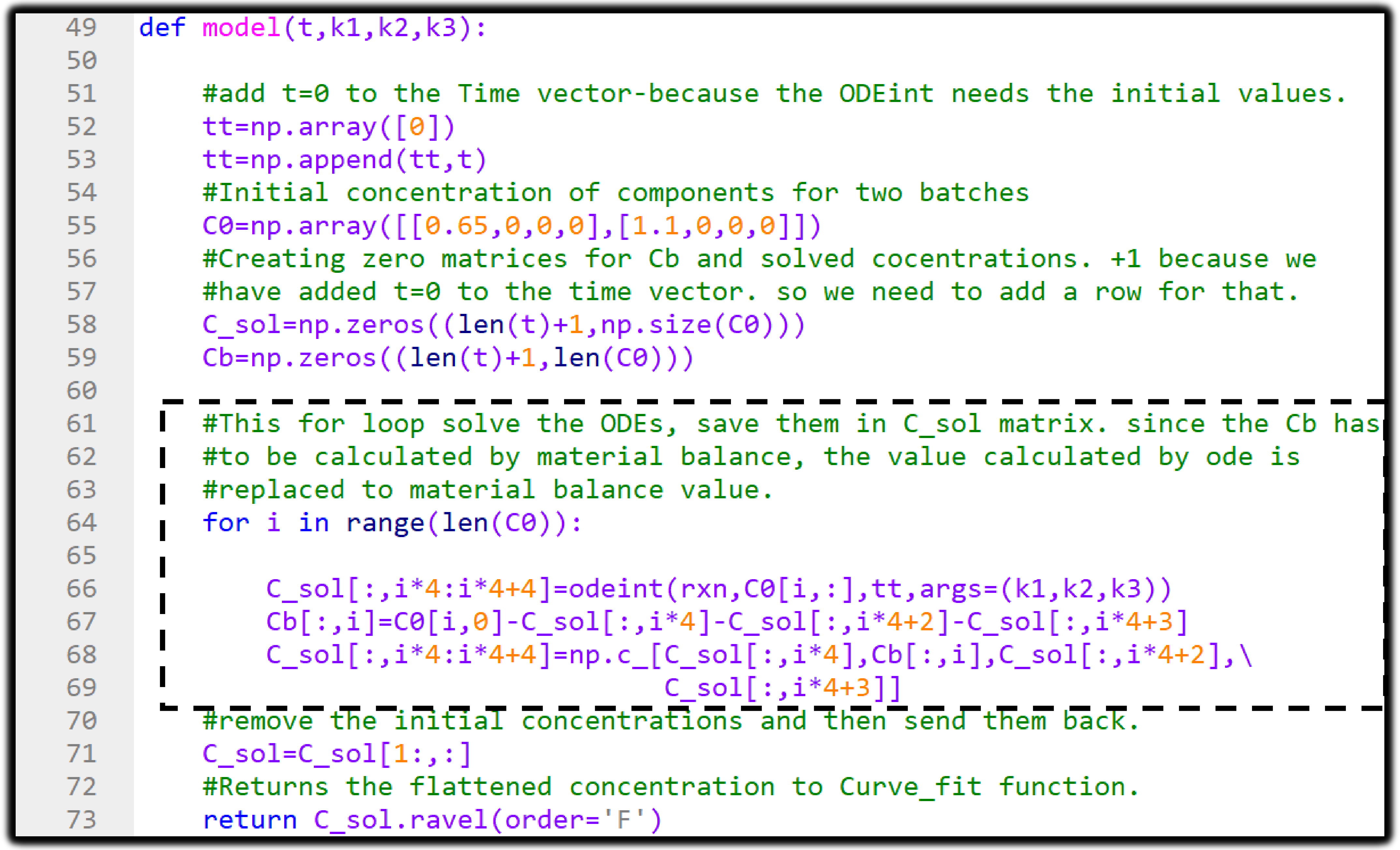
part1: batch reactor + isothermal data (code)



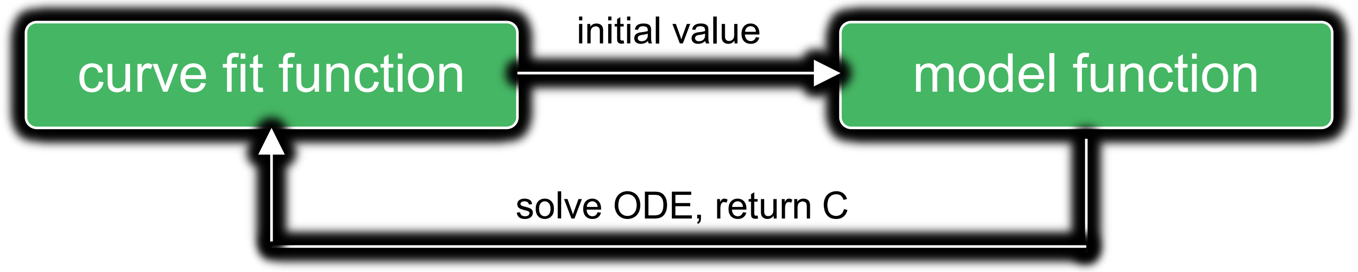
import packages

import data

curve fitting



```python
def model(t,k1,k2,k3):

    #add t=0 to the Time vector-because the ODEint needs the initial values.
    tt=np.array([0])
    tt=np.append(tt,t)
    #Initial concentration of components for two batches
    C0=np.array([[0.65,0,0,0],[1.1,0,0,0]])
    #Creating zero matrices for Cb and solved cocentrations. +1 because we
    #have added t=0 to the time vector. so we need to add a row for that.
    C_sol=np.zeros((len(t)+1,np.size(C0)))
    Cb=np.zeros((len(t)+1,len(C0)))

    #This for loop solve the ODEs, save them in C_sol matrix. since the Cb has
    #to be calculated by material balance, the value calculated by ode is
    #replaced to material balance value.
    for i in range(len(C0)):

        C_sol[:,i*4:i*4+4]=odeint(rxn,C0[i,:],tt,args=(k1,k2,k3))
        Cb[:,i]=C0[i,0]-C_sol[:,i*4]-C_sol[:,i*4+2]-C_sol[:,i*4+3]
        C_sol[:,i*4:i*4+4]=np.c_[C_sol[:,i*4],Cb[:,i],C_sol[:,i*4+2],\
                                  C_sol[:,i*4+3]]
    #remove the initial concentrations and then send them back.
    C_sol=C_sol[1:,:]
    #Returns the flattened concentration to Curve_fit function.
    return C_sol.ravel(order='F')
```



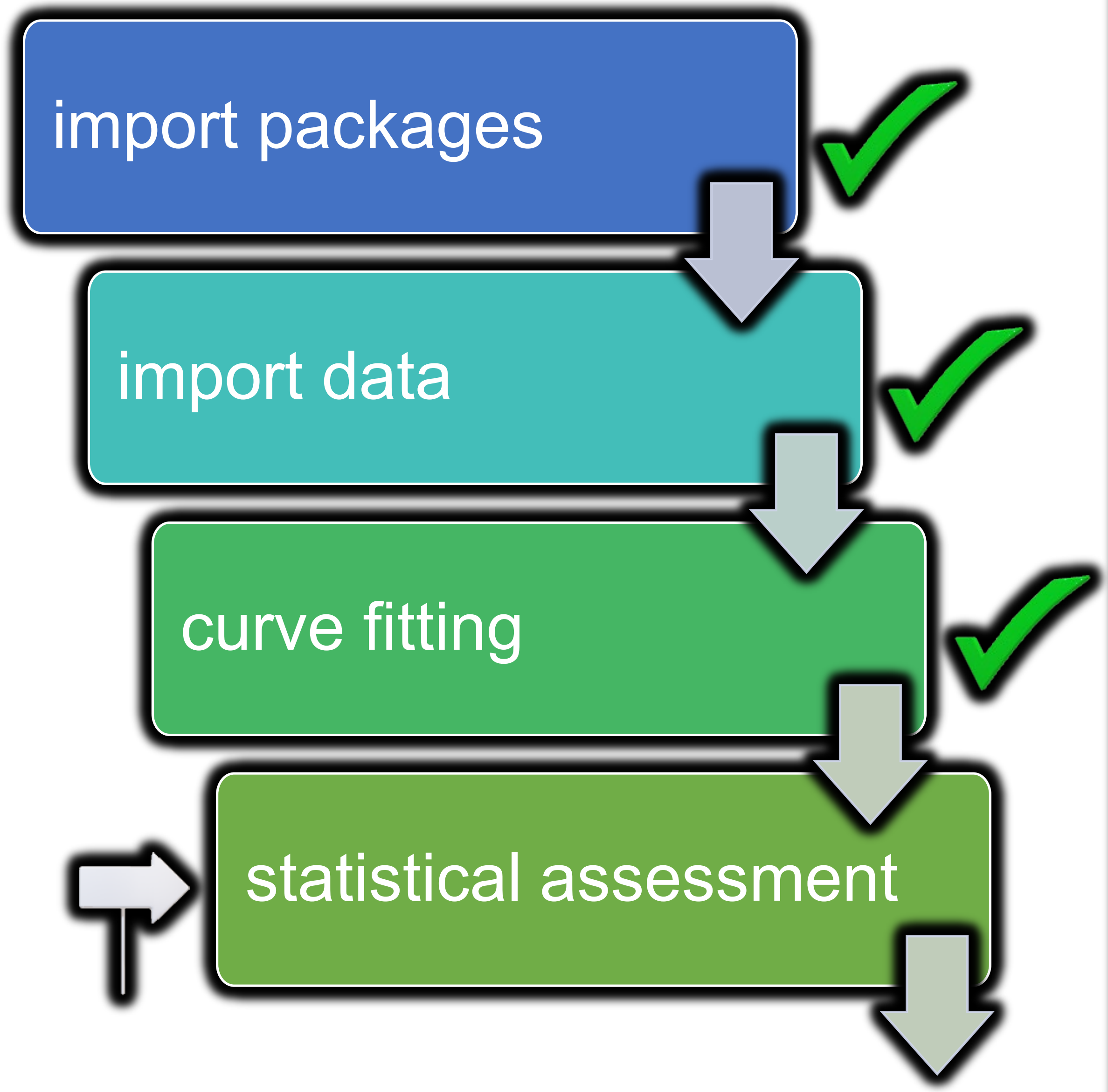
curve fit function

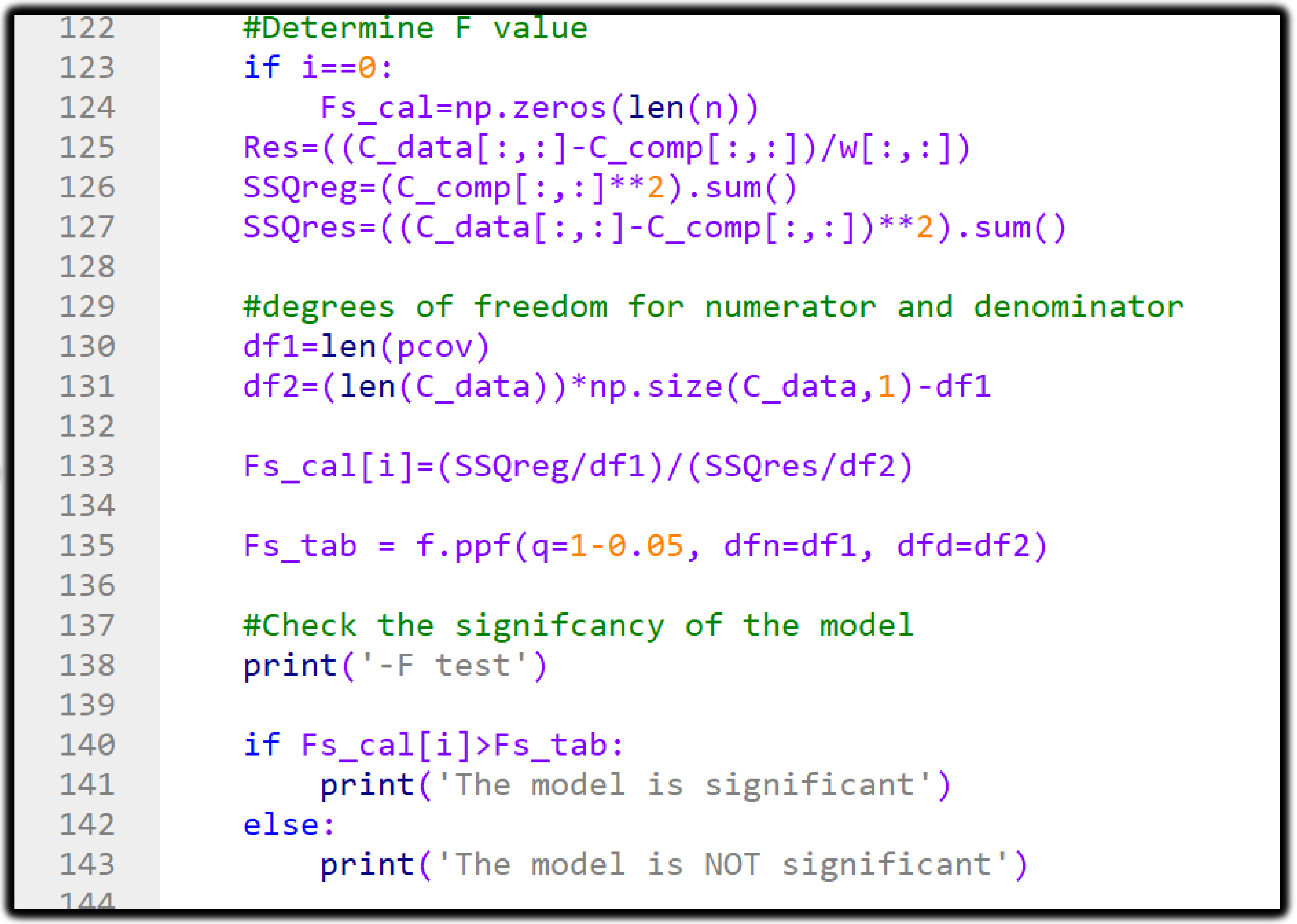
initial value

model function

solve ODE, return C

part1: batch reactor + isothermal data (procedure)

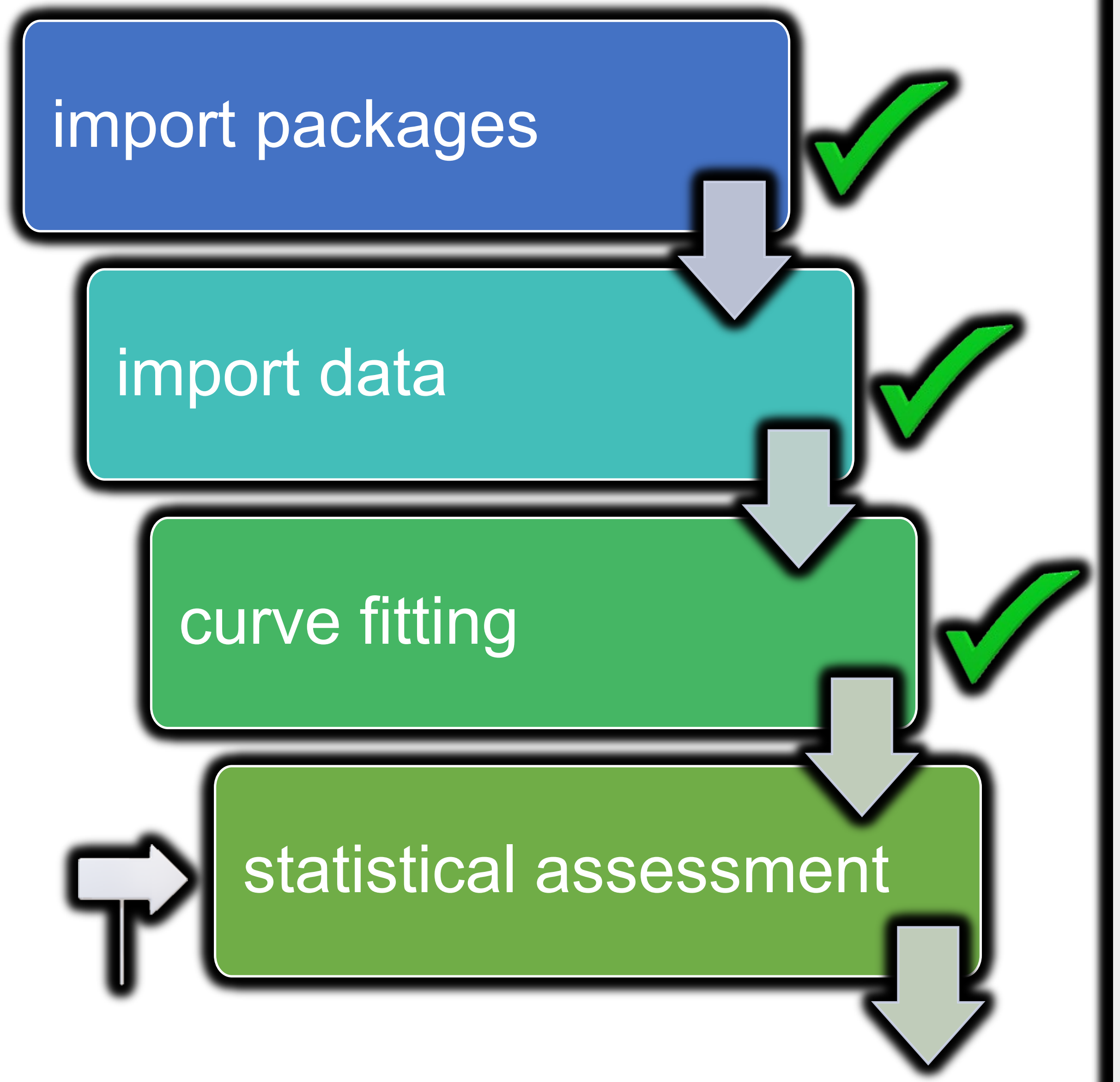


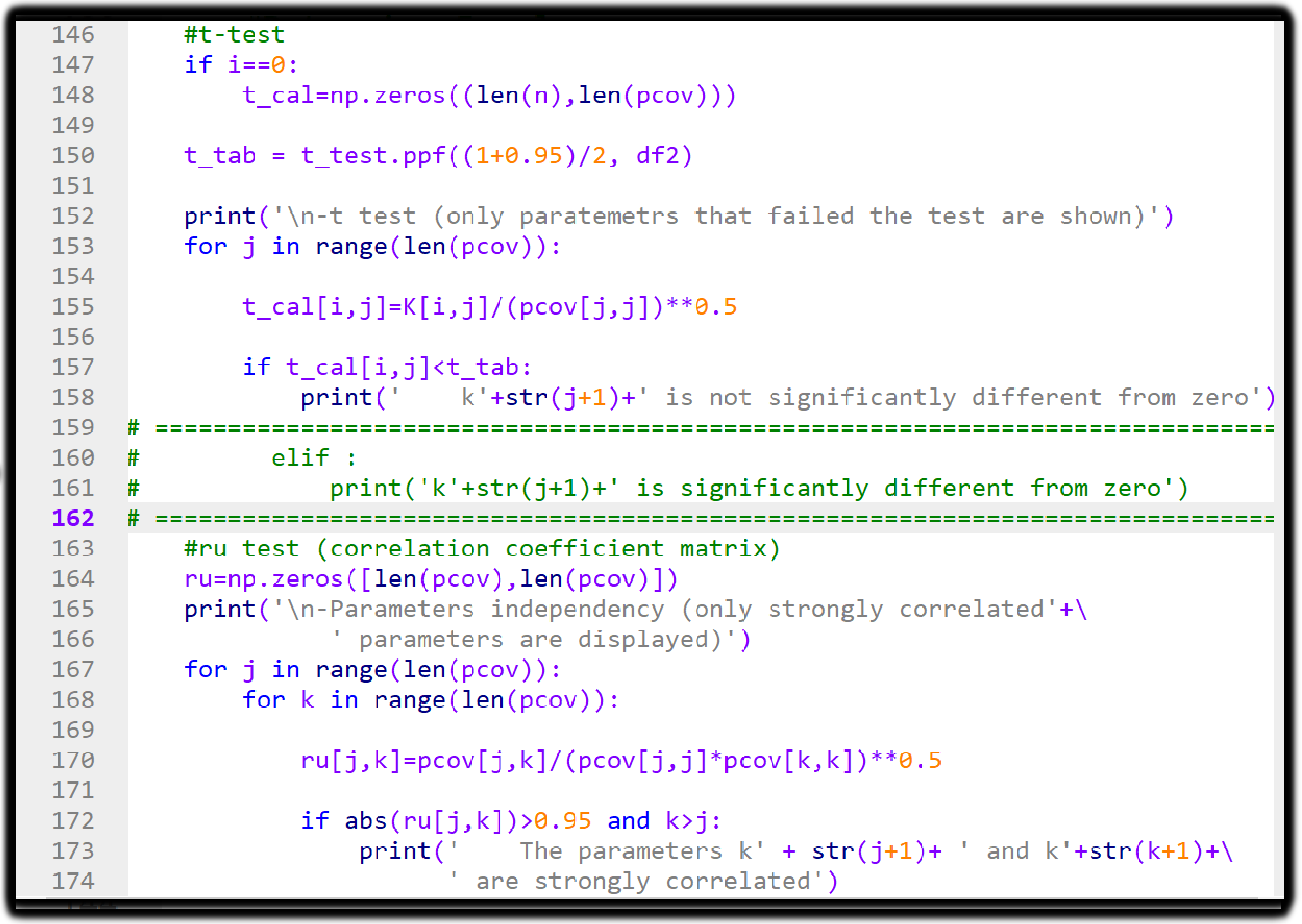


```
#Determine F value
if i==0:
    Fs_cal=np.zeros(len(n))
Res=((C_data[:,:]-C_comp[:,:])/w[:,:])
SSQreg=(C_comp[:,:]**2).sum()
SSQres=((C_data[:,:]-C_comp[:,:])**2).sum()

#degrees of freedom for numerator and denominator
df1=len(pcov)
df2=(len(C_data))*np.size(C_data,1)-df1

Fs_cal[i]=(SSQreg/df1)/(SSQres/df2)

Fs_tab = f.ppf(q=1-0.05, dfn=df1, dfd=df2)

#Check the signifcancy of the model
print('-F test')

if Fs_cal[i]>Fs_tab:
    print('The model is significant')
else:
    print('The model is NOT significant')
```

f test

part1: batch reactor + isothermal data (procedure)





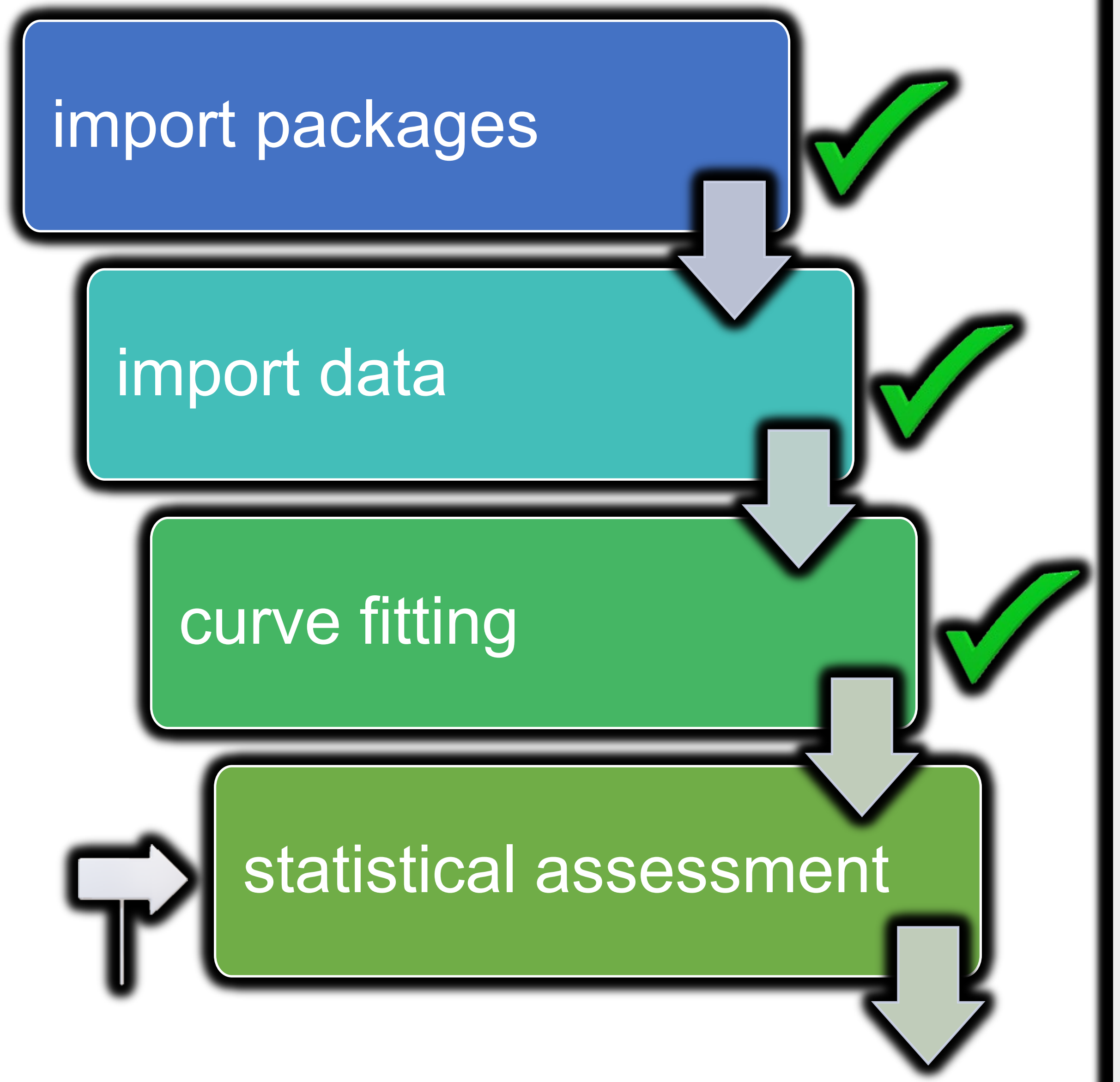
```
    #t-test
    if i==0:
        t_cal=np.zeros((len(n),len(pcov)))

    t_tab = t_test.ppf((1+0.95)/2, df2)

    print('\n-t test (only paratemetrs that failed the test are shown)')
    for j in range(len(pcov)):

        t_cal[i,j]=K[i,j]/(pcov[j,j])**0.5

        if t_cal[i,j]<t_tab:
            print('    k'+str(j+1)+' is not significantly different from zero')
# ============================================================================
#         elif :
#             print('k'+str(j+1)+' is significantly different from zero')
# ============================================================================
    #ru test (correlation coefficient matrix)
    ru=np.zeros([len(pcov),len(pcov)])
    print('\n-Parameters independency (only strongly correlated'+\
              ' parameters are displayed)')
    for j in range(len(pcov)):
        for k in range(len(pcov)):

            ru[j,k]=pcov[j,k]/(pcov[j,j]*pcov[k,k])**0.5

            if abs(ru[j,k])>0.95 and k>j:
                print('    The parameters k' + str(j+1)+ ' and k'+str(k+1)+\
                      ' are strongly correlated')
```

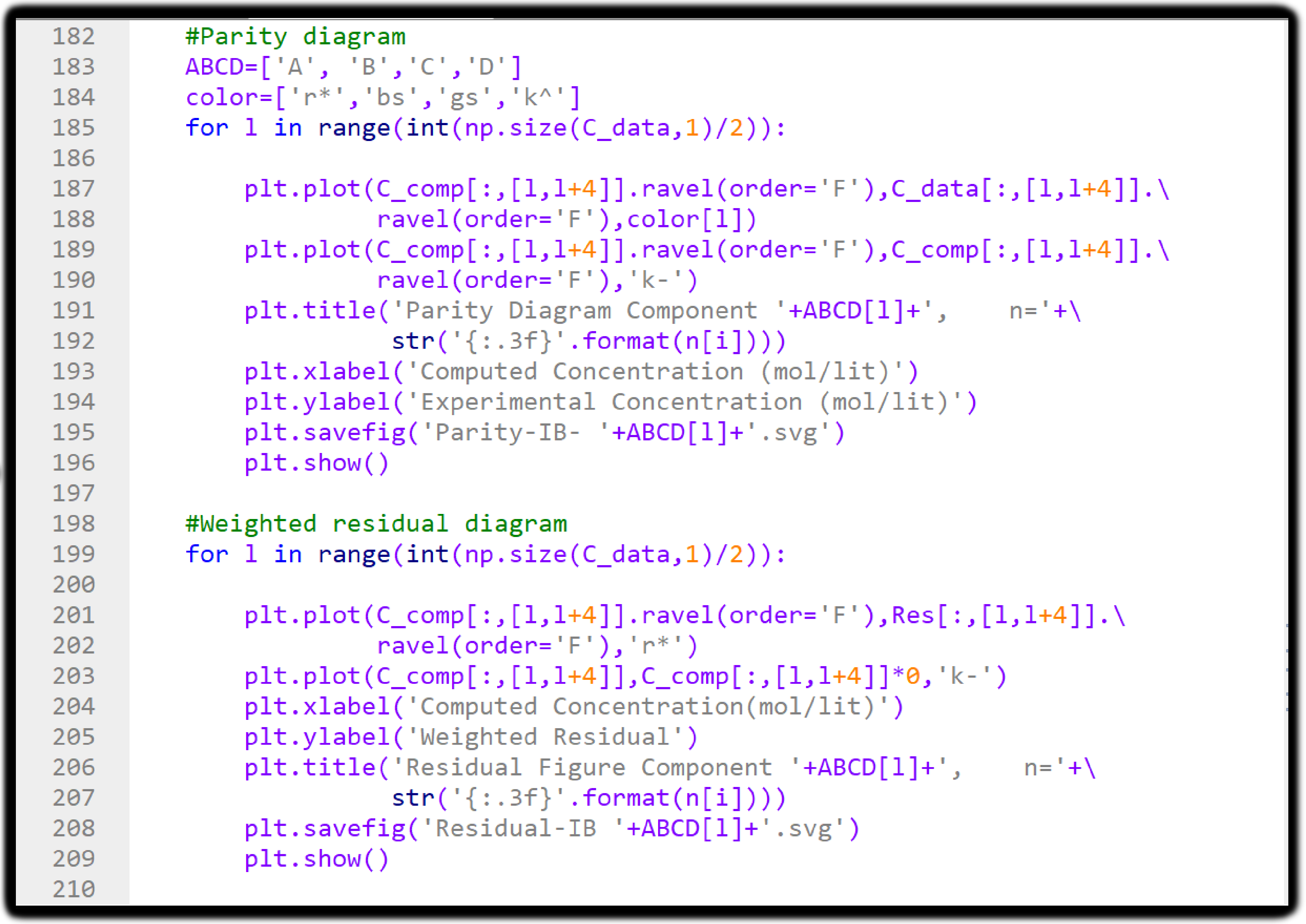
f test

t test

independency test

part1: batch reactor + isothermal data (procedure)





```
#Parity diagram
ABCD=['A', 'B','C','D']
color=['r*','bs','gs','k^']
for l in range(int(np.size(C_data,1)/2)):

    plt.plot(C_comp[:,[l,l+4]].ravel(order='F'),C_data[:,[l,l+4]].\
             ravel(order='F'),color[l])
    plt.plot(C_comp[:,[l,l+4]].ravel(order='F'),C_comp[:,[l,l+4]].\
             ravel(order='F'),'k-')
    plt.title('Parity Diagram Component '+ABCD[l]+',    n='+\
              str('{:.3f}'.format(n[i])))
    plt.xlabel('Computed Concentration (mol/lit)')
    plt.ylabel('Experimental Concentration (mol/lit)')
    plt.savefig('Parity-IB- '+ABCD[l]+'.svg')
    plt.show()

#Weighted residual diagram
for l in range(int(np.size(C_data,1)/2)):

    plt.plot(C_comp[:,[l,l+4]].ravel(order='F'),Res[:,[l,l+4]].\
             ravel(order='F'),'r*')
    plt.plot(C_comp[:,[l,l+4]],C_comp[:,[l,l+4]]*0,'k-')
    plt.xlabel('Computed Concentration(mol/lit)')
    plt.ylabel('Weighted Residual')
    plt.title('Residual Figure Component '+ABCD[l]+',    n='+\
              str('{:.3f}'.format(n[i])))
    plt.savefig('Residual-IB '+ABCD[l]+'.svg')
    plt.show()

```

f test

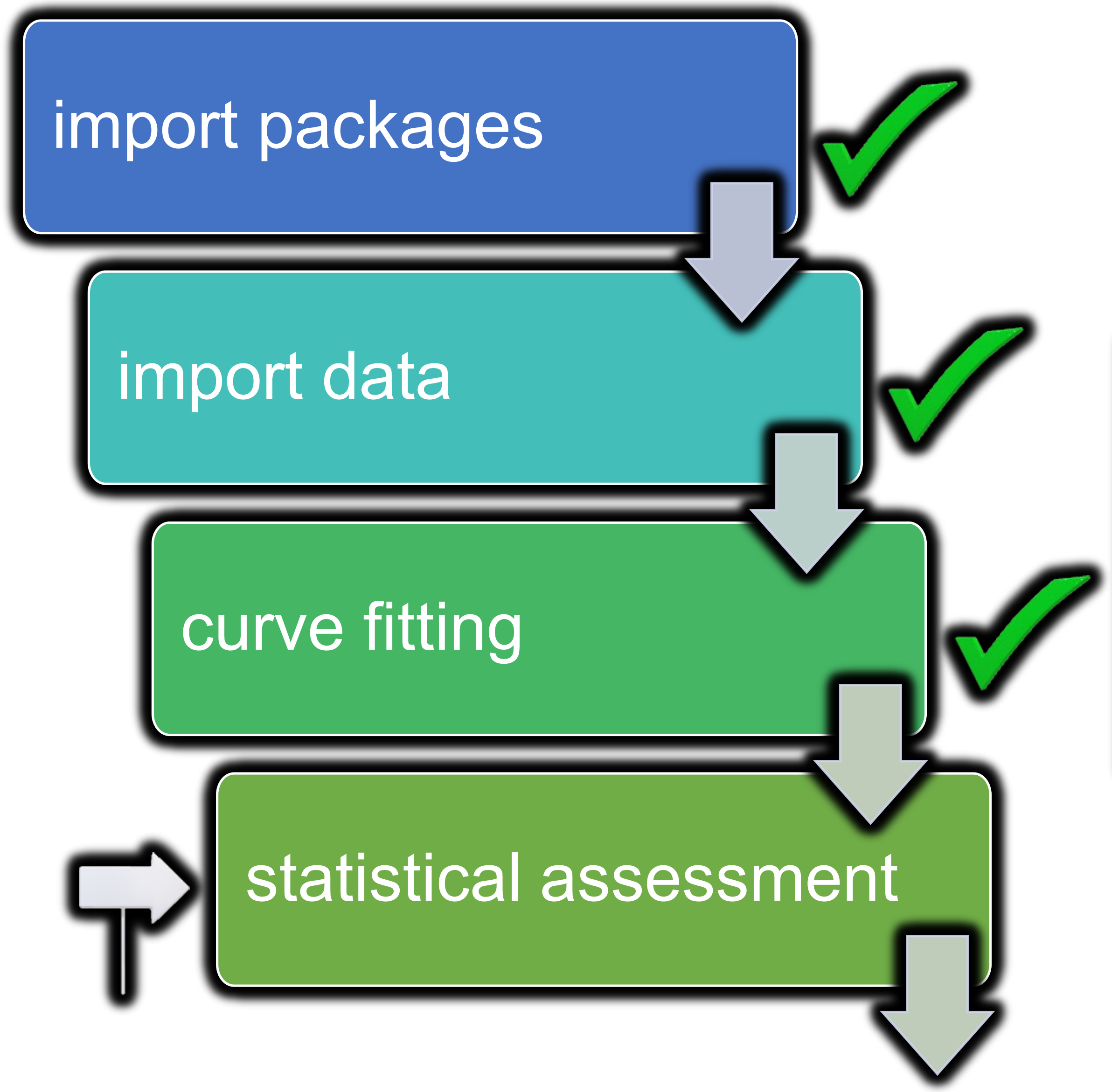
t test

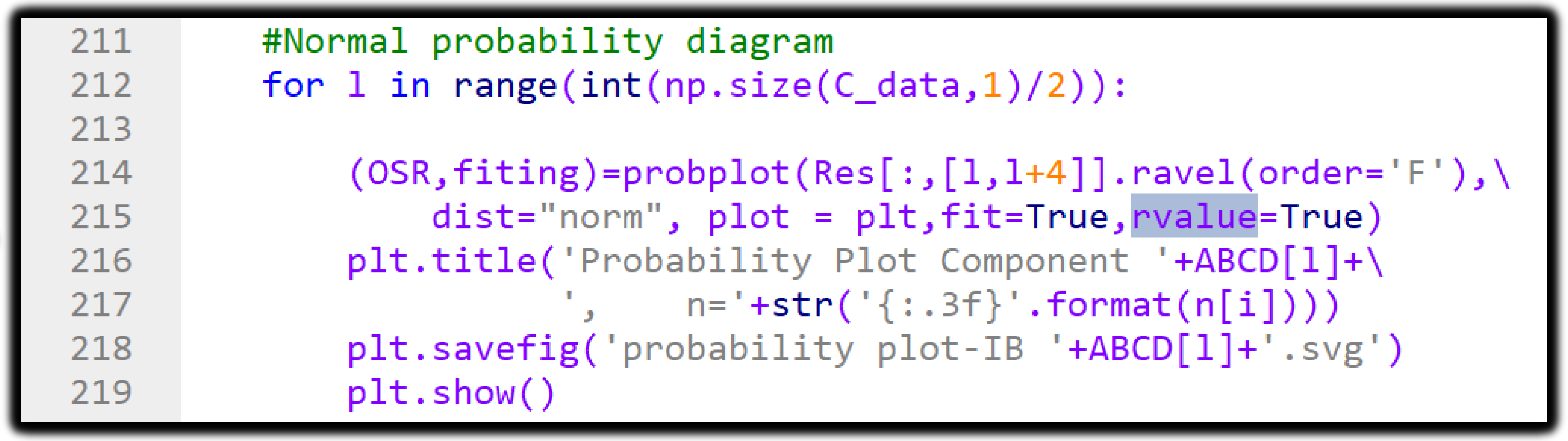
independency test

parity diagram

residual diagram

part1: batch reactor + isothermal data (procedure)





```
    #Normal probability diagram
    for l in range(int(np.size(C_data,1)/2)):

        (OSR,fiting)=probplot(Res[:,[l,l+4]].ravel(order='F'),\
            dist="norm", plot = plt,fit=True,rvalue=True)
        plt.title('Probability Plot Component '+ABCD[l]+\
                  ',    n='+str('{:.3f}'.format(n[i])))
        plt.savefig('probability plot-IB '+ABCD[l]+'.svg')
        plt.show()
```

f test

t test

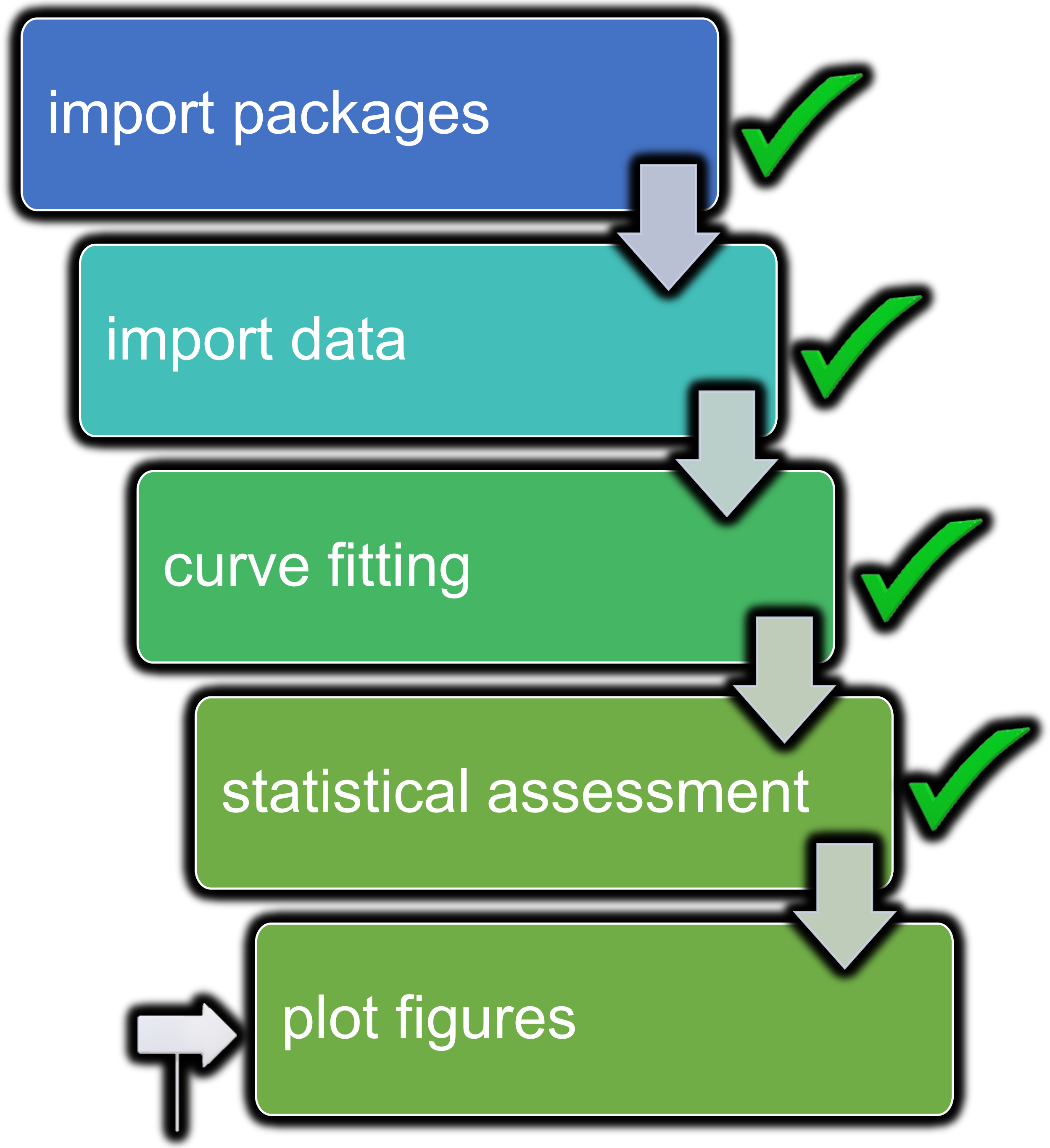
independency test

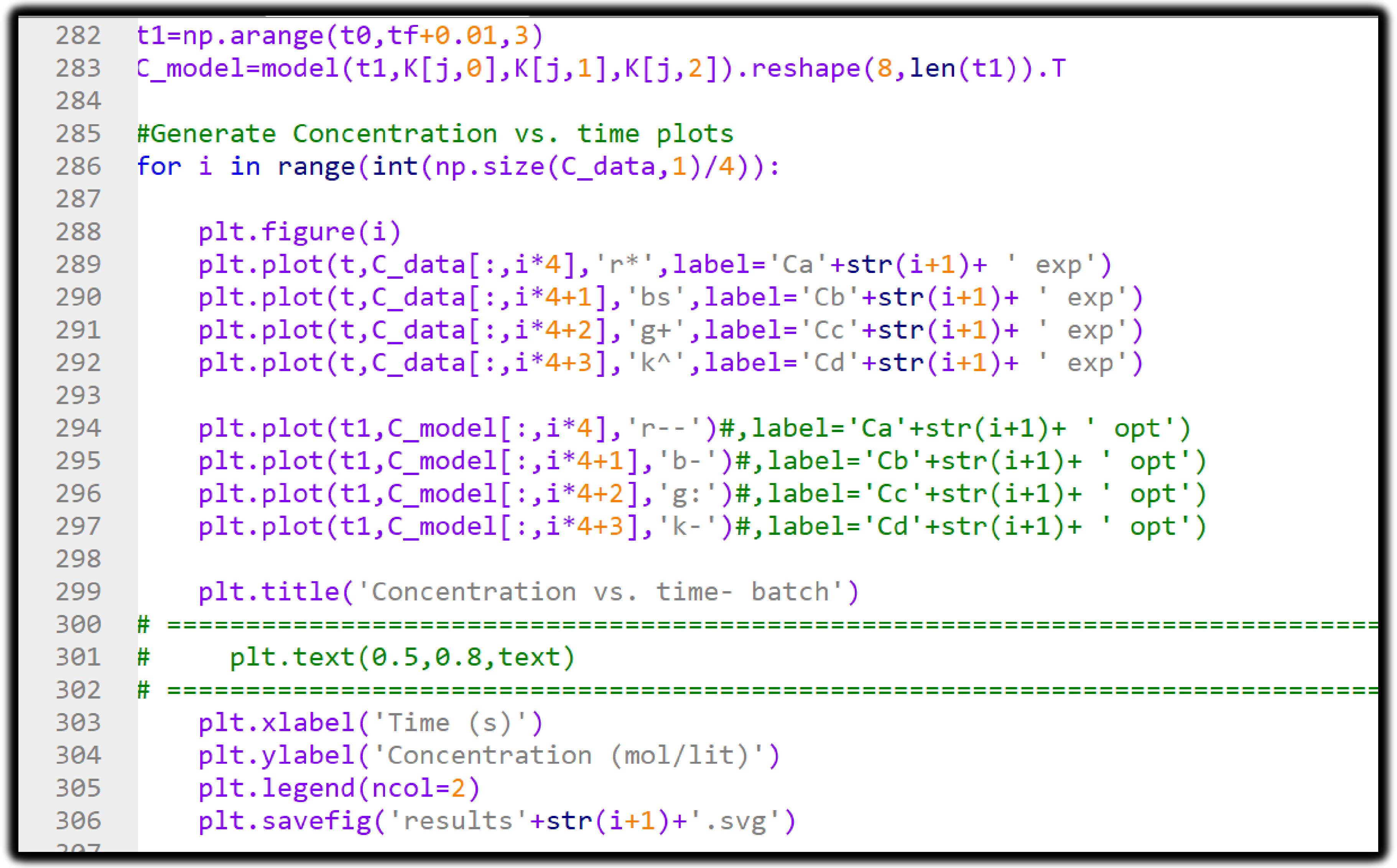
parity diagram

residual diagram

normal probability plot

part1: batch reactor + isothermal data (procedure)



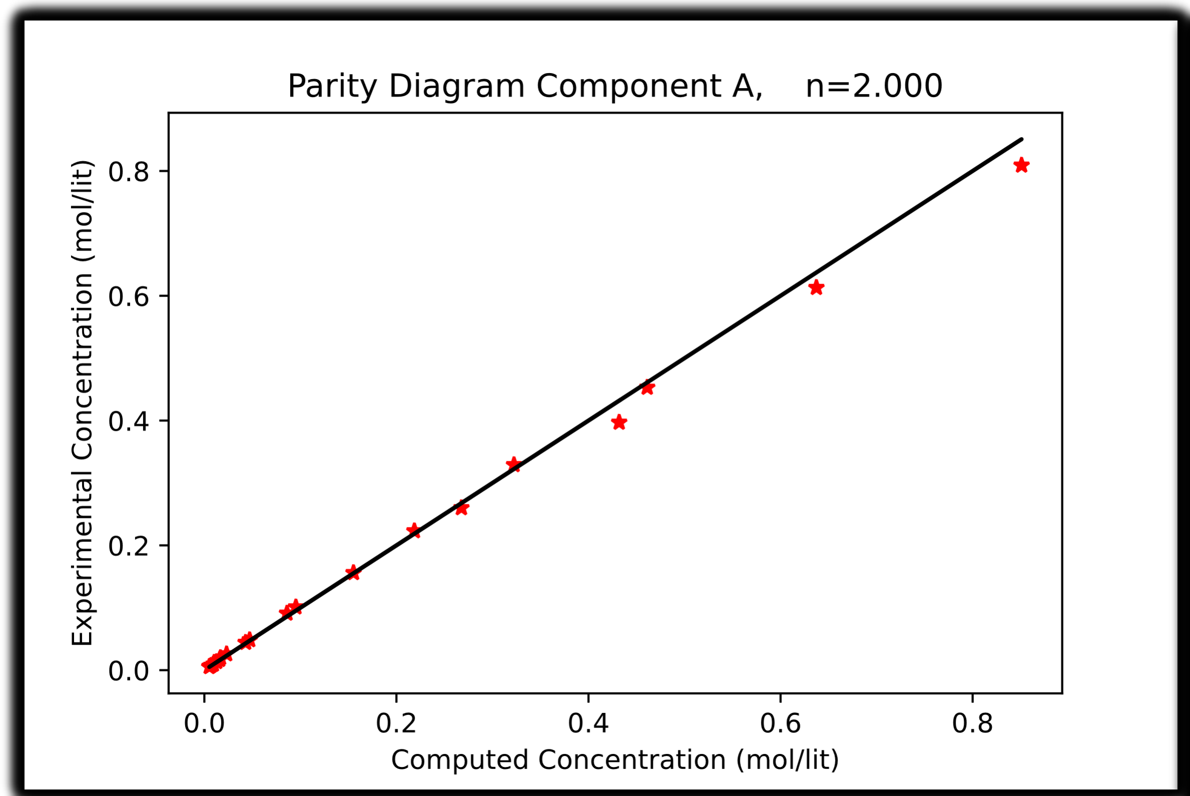


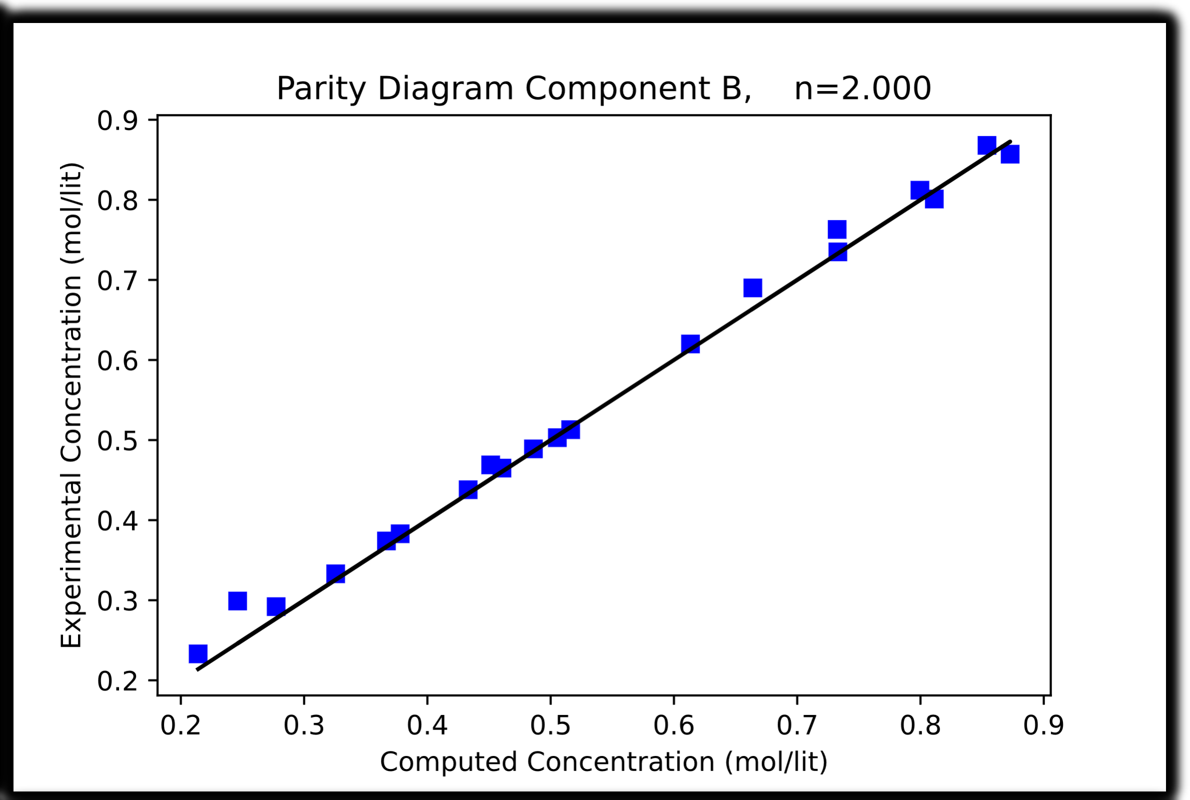
```
t1=np.arange(t0,tf+0.01,3)
C_model=model(t1,K[j,0],K[j,1],K[j,2]).reshape(8,len(t1)).T

#Generate Concentration vs. time plots
for i in range(int(np.size(C_data,1)/4)):

    plt.figure(i)
    plt.plot(t,C_data[:,i*4],'r*',label='Ca'+str(i+1)+ ' exp')
    plt.plot(t,C_data[:,i*4+1],'bs',label='Cb'+str(i+1)+ ' exp')
    plt.plot(t,C_data[:,i*4+2],'g+',label='Cc'+str(i+1)+ ' exp')
    plt.plot(t,C_data[:,i*4+3],'k^',label='Cd'+str(i+1)+ ' exp')

    plt.plot(t1,C_model[:,i*4],'r--')#,label='Ca'+str(i+1)+ ' opt')
    plt.plot(t1,C_model[:,i*4+1],'b-')#,label='Cb'+str(i+1)+ ' opt')
    plt.plot(t1,C_model[:,i*4+2],'g:')#,label='Cc'+str(i+1)+ ' opt')
    plt.plot(t1,C_model[:,i*4+3],'k-')#,label='Cd'+str(i+1)+ ' opt')

    plt.title('Concentration vs. time- batch')
# ===============================================================================
#     plt.text(0.5,0.8,text)
# ===============================================================================
    plt.xlabel('Time (s)')
    plt.ylabel('Concentration (mol/lit)')
    plt.legend(ncol=2)
    plt.savefig('results'+str(i+1)+'.svg')
```

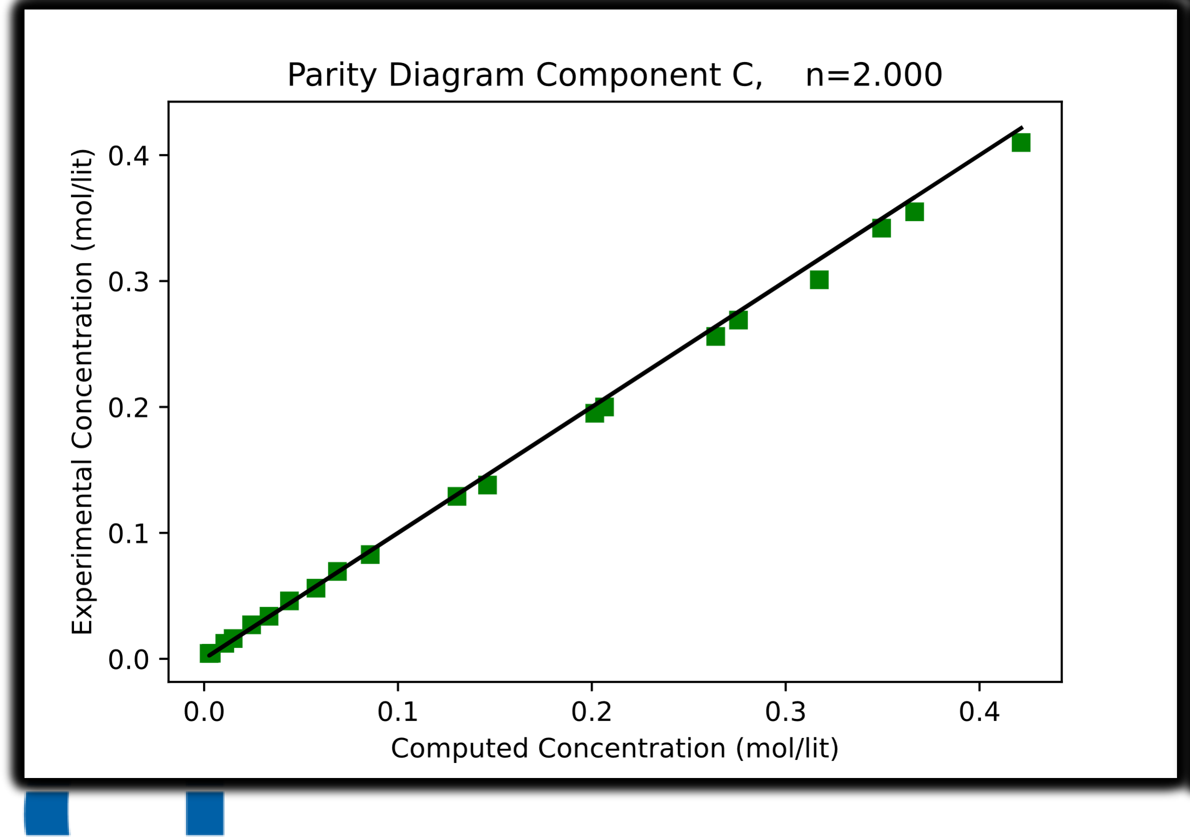
part1: batch reactor + isothermal data (result)

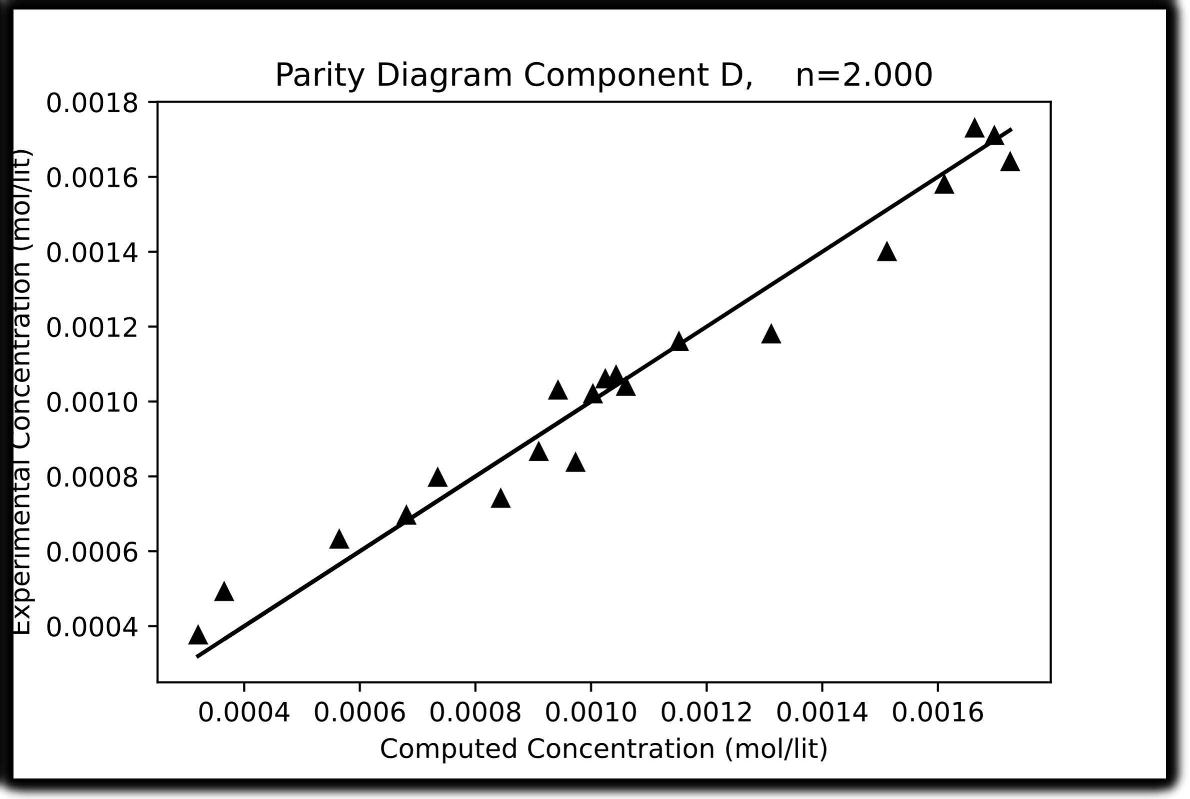
parameter/diagram	outputs		
k values	k1=0.01016	k2=0.00195	$k3=1.49*10^{-5}$
F test	$2.1*10^{4} > 2.72$ ✓		
T test	91 > 2 ✓	50 > 2 ✓	50 > 2 ✓
Independent parameters?	✓	✓	✓
parity diagram	✓		

part1: batch reactor + isothermal data (result)





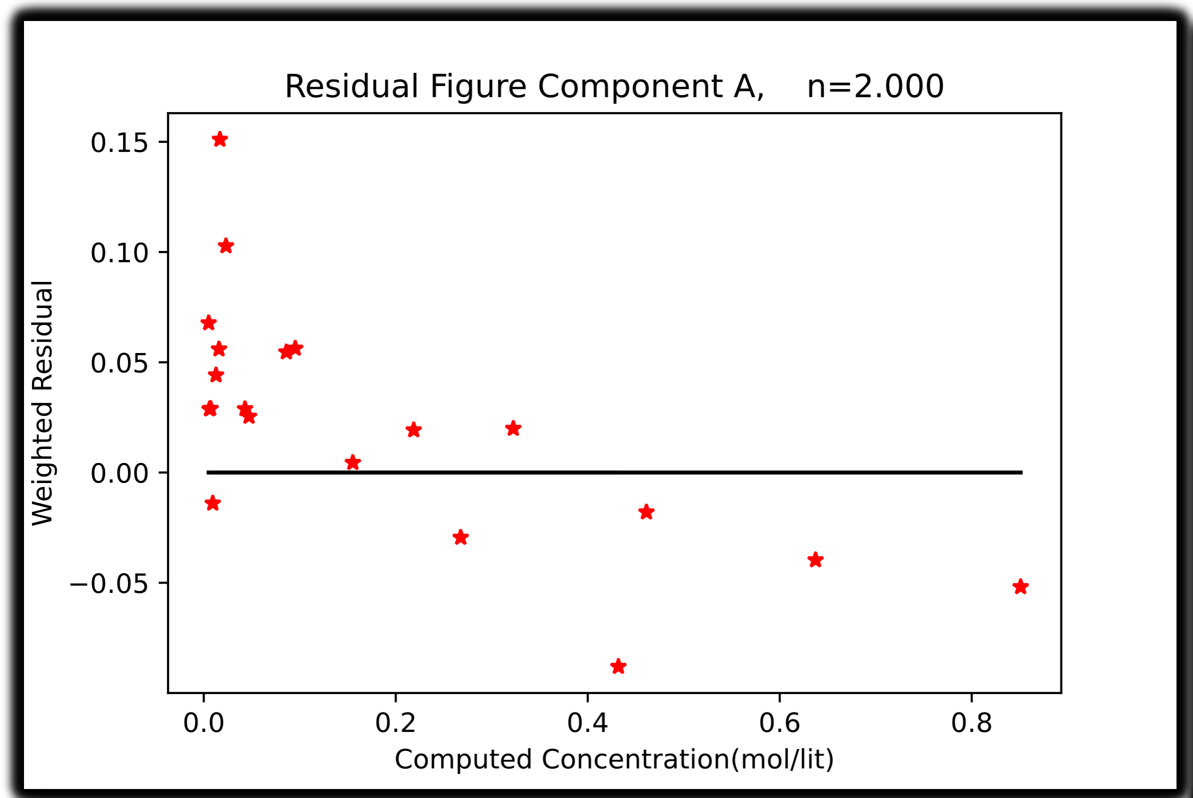


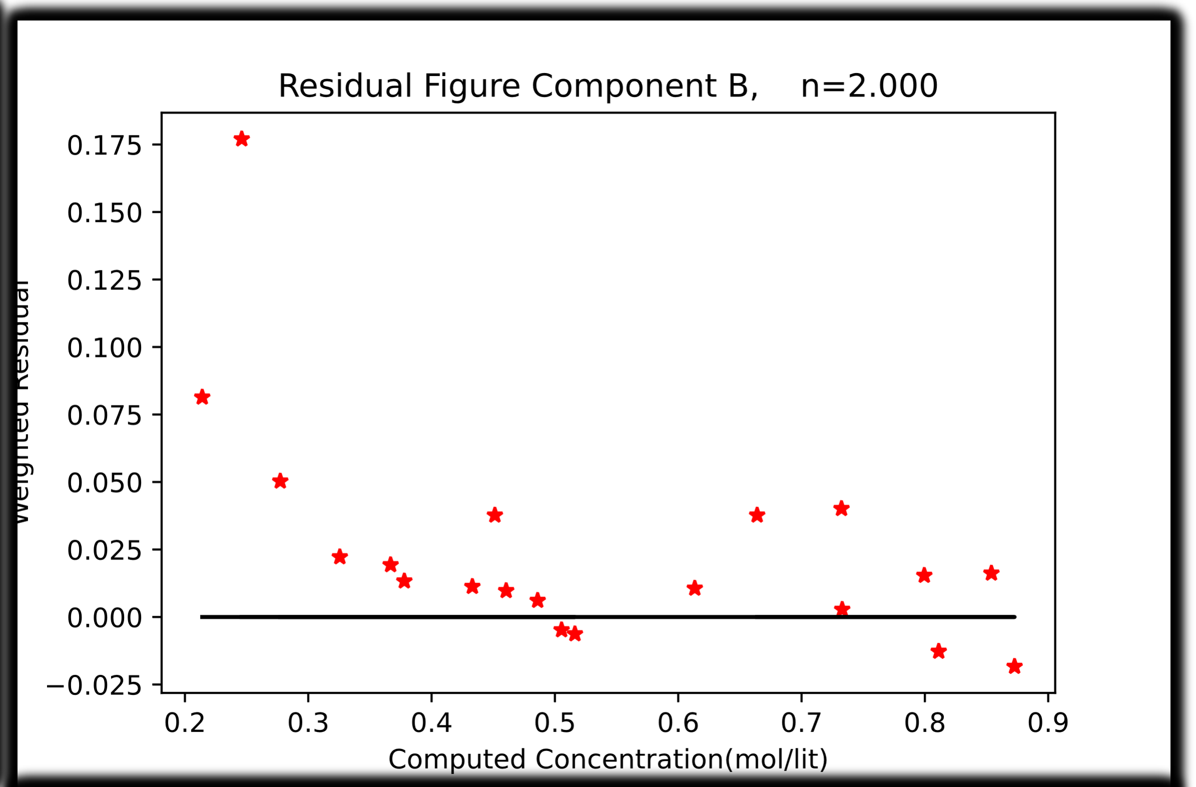


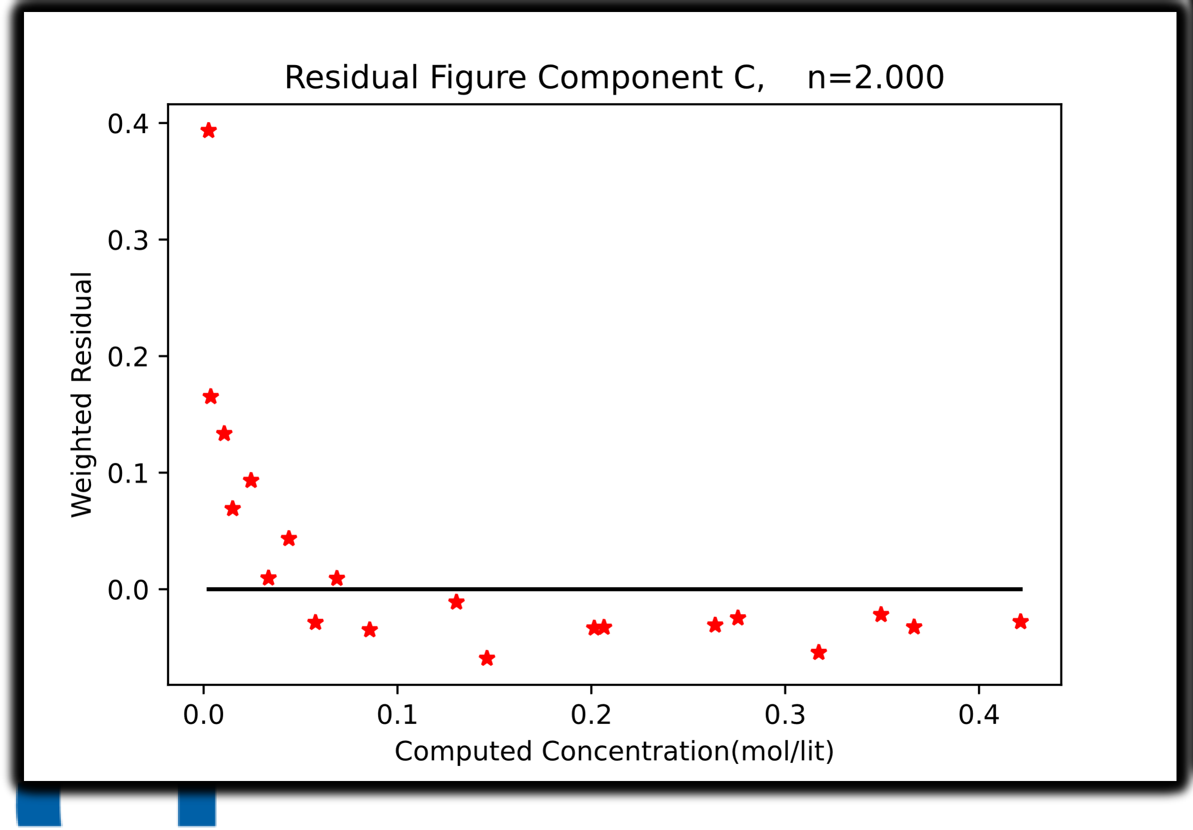
part1: batch reactor + isothermal data (result)

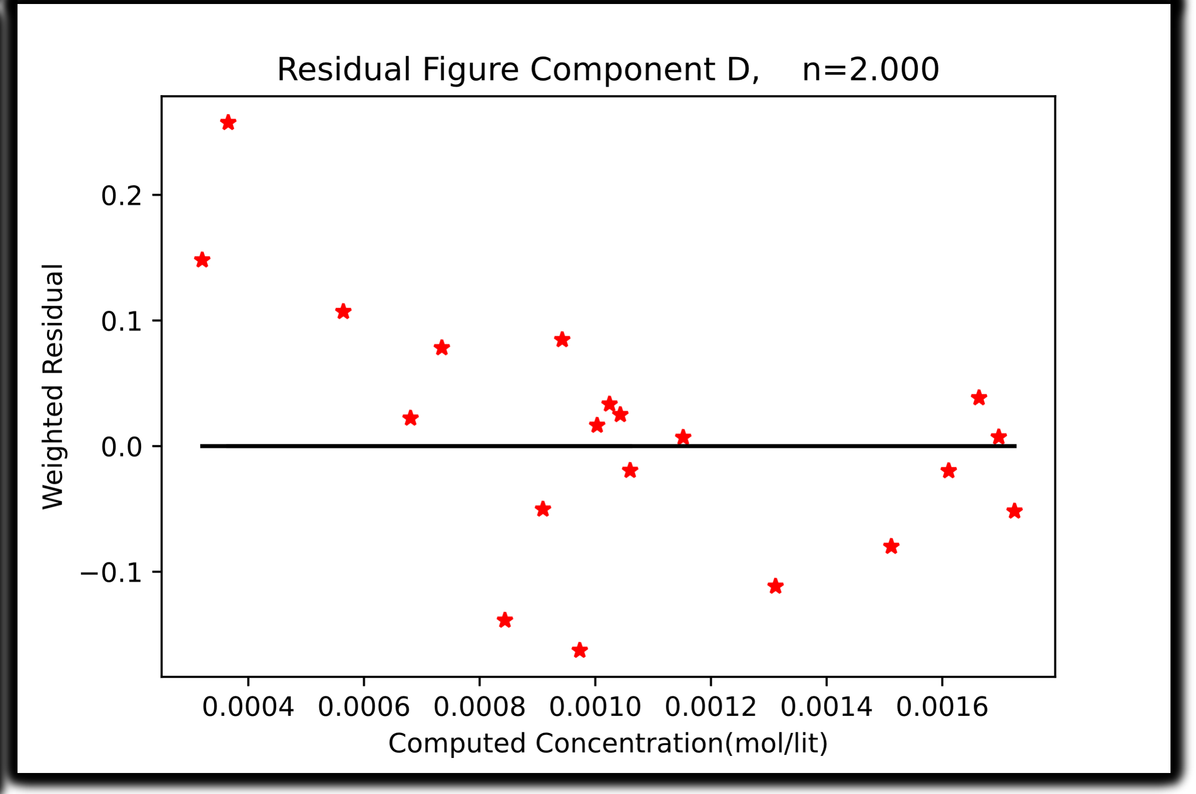
parameter/diagram	outputs		
k values	k1=0.01016	k2=0.00195	$k3=1.49*10^{-5}$
F test	$2.1*10^4 > 2.72$ ✓		
T test	91 > 2 ✓	50 > 2 ✓	50 > 2 ✓
Independent parameters?	✓	✓	✓
parity diagram	✓		
residual diagram	✓		

part1: batch reactor + isothermal data (result)





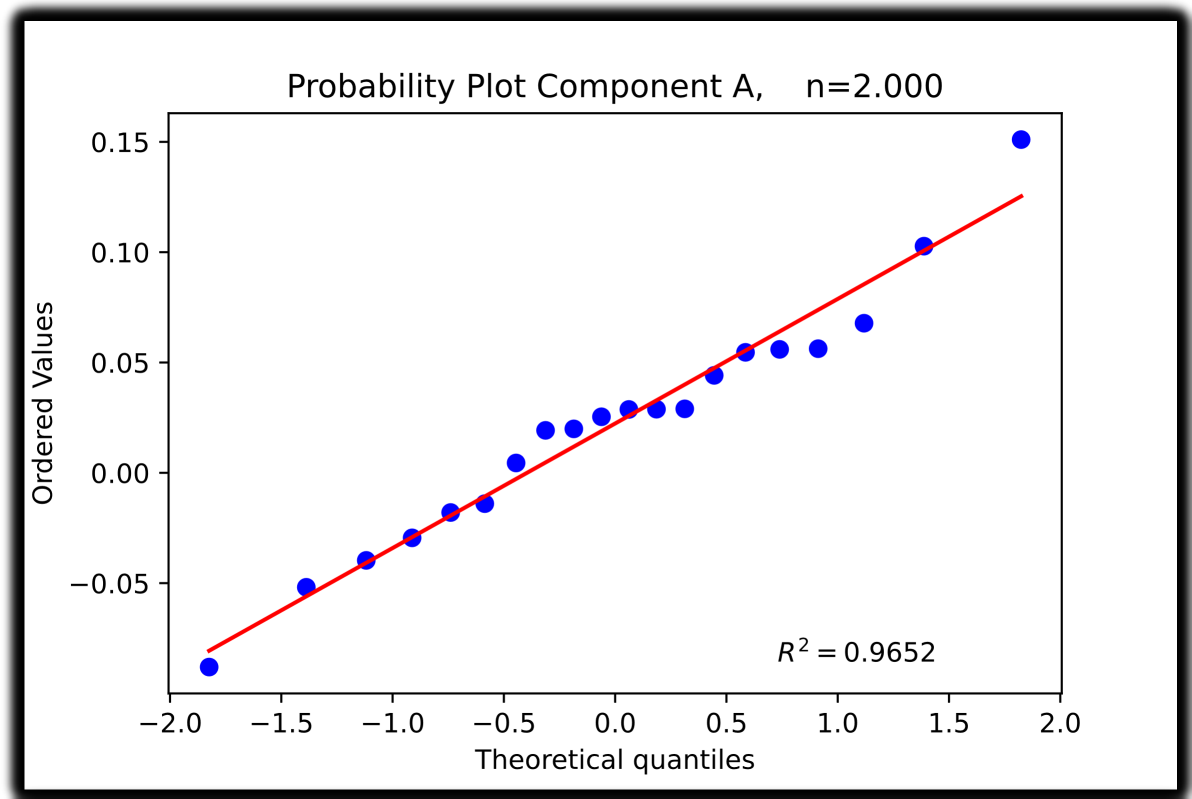




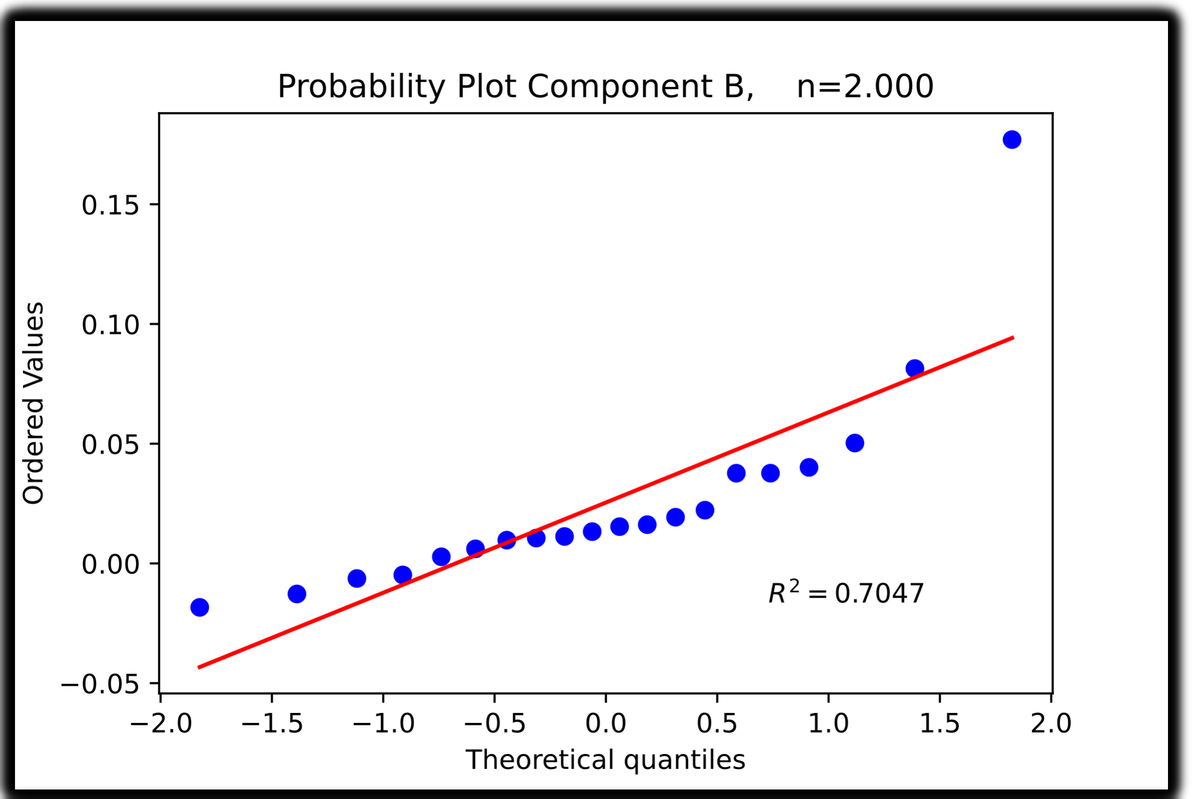
part1: batch reactor + isothermal data (result)

parameter/diagram	outputs		
k values	k1=0.01016	k2=0.00195	k3=1.49*10^{-5}
F test	2.1*10^{4} > 2.72 ✓		
T test	91 > 2 ✓	50 > 2 ✓	50 > 2 ✓
Independent parameters?	✓	✓	✓
parity diagram	✓		
residual diagram	✓		
Normal probablity diagram	✓		

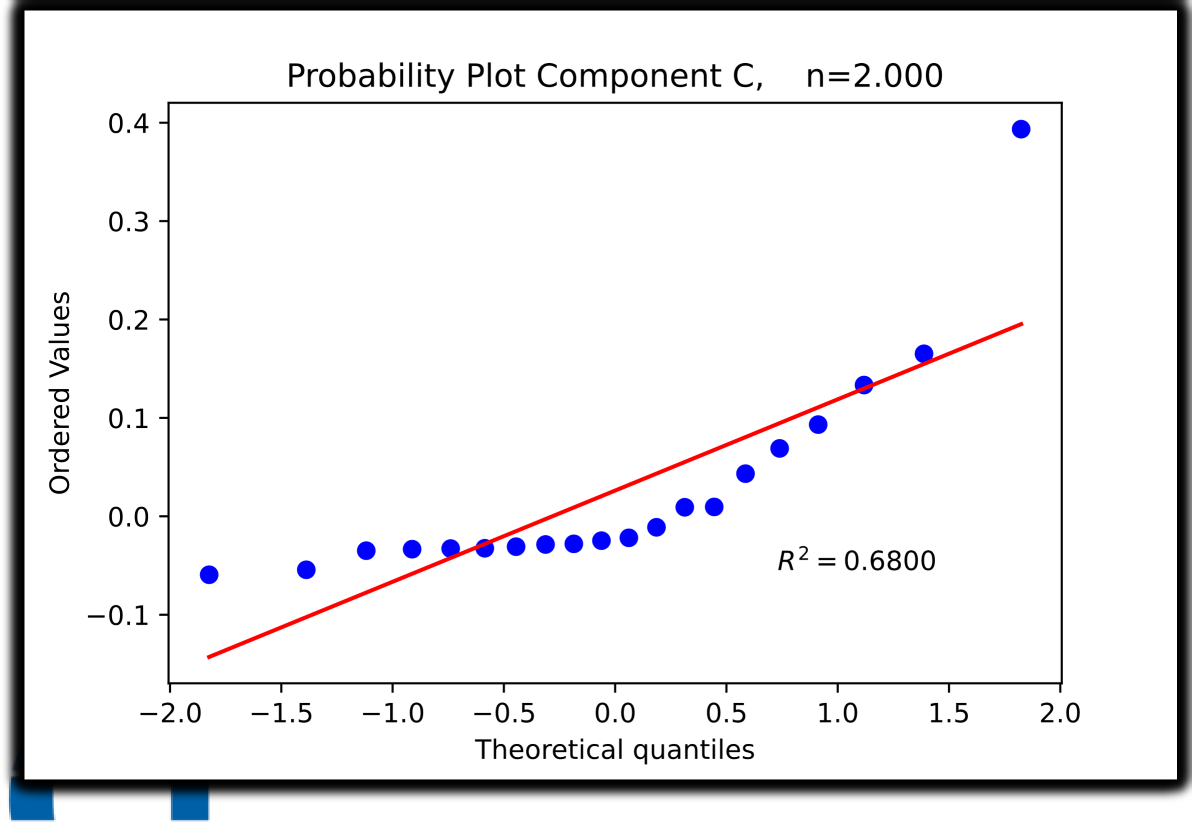
part1: batch reactor + isothermal data (result)



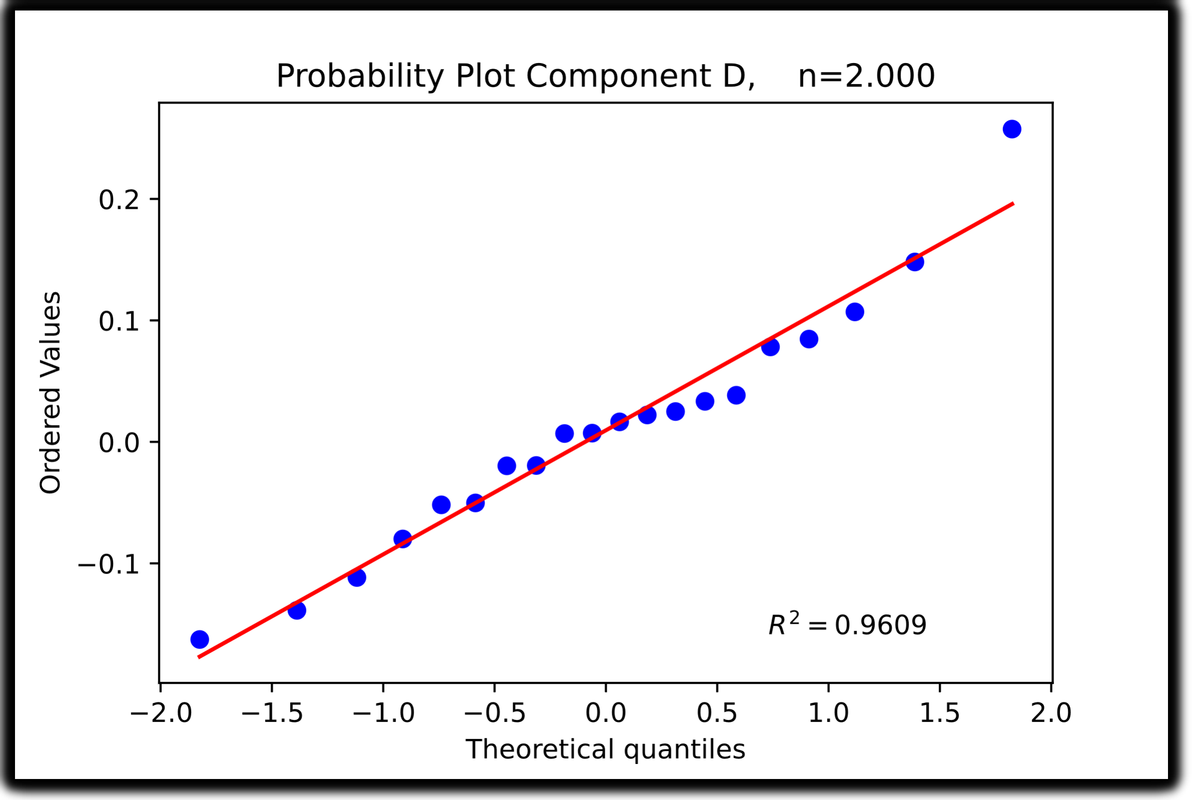
Probability Plot Component A, n=2.000



Probability Plot Component B, n=2.000



Probability Plot Component C, n=2.000



Probability Plot Component D, n=2.000

GHENT UNIVERSITY

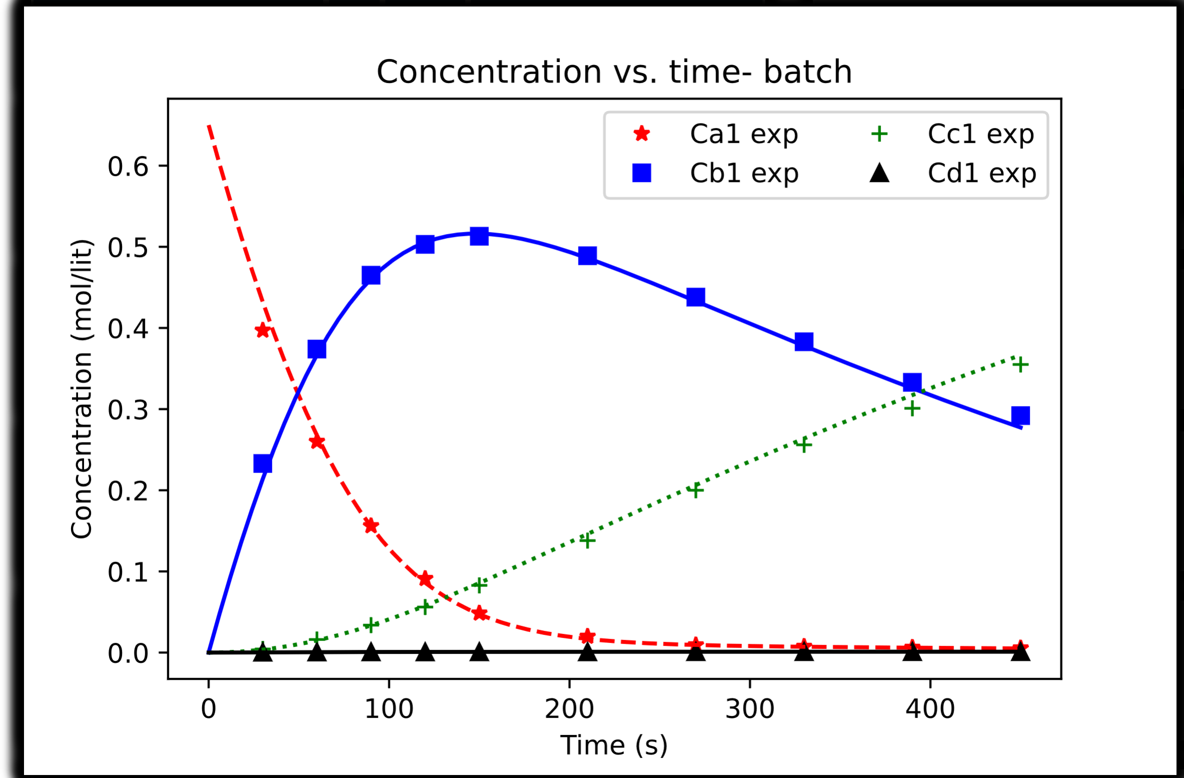
DRIVING CHEMICAL TECHNOLOGY

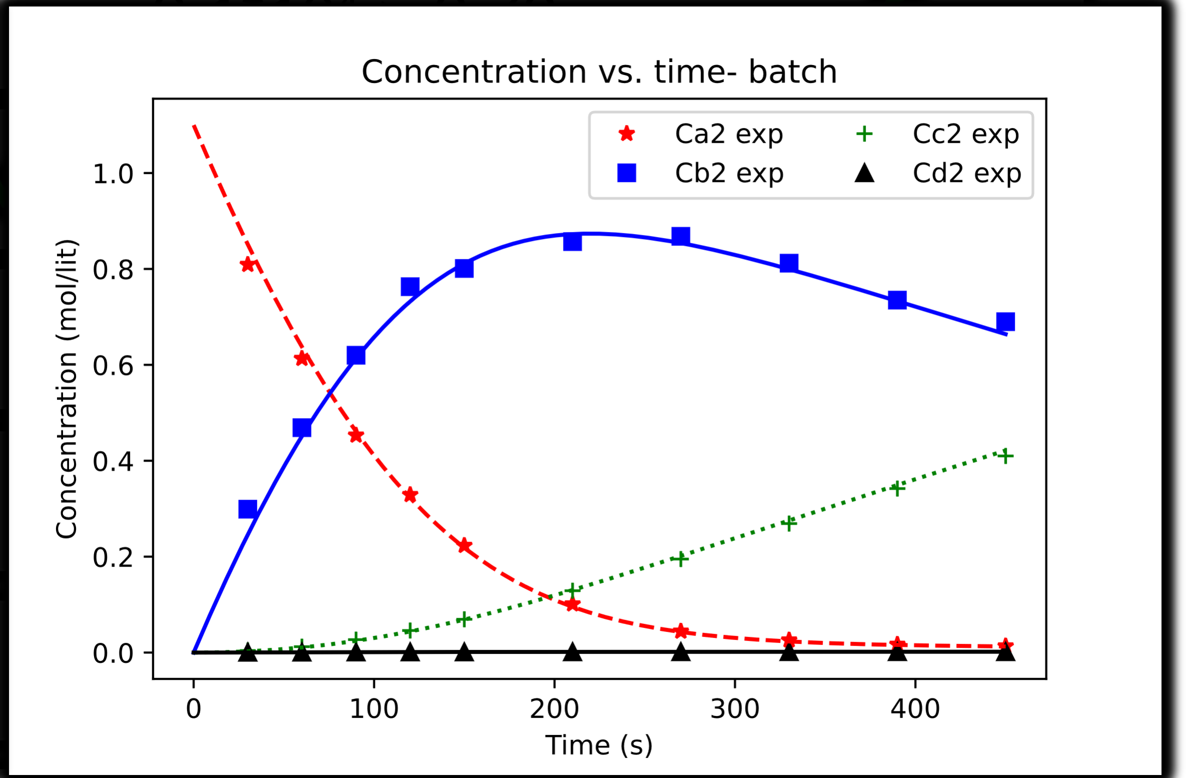
part1: batch reactor + isothermal data (result)

parameter/diagram	outputs		
k values	k1=0.01016	k2=0.00195	k3=$1.49*10^{-5}$
F test	$2.1*10^4 > 2.72$ ✓		
T test	91 > 2 ✓	50 > 2 ✓	50 > 2 ✓
Independent parameters?	✓	✓	✓
parity diagram	✓		
residual diagram	✓		
Normal probablity diagram	✓		
conc. vs. time	✓		

part1: batch reactor + isothermal data (result)

parameter/diagram	outputs		
k values	k1=0.01016	k2=0.00195	k3=1.49*10^{-5}
[illegible]	[illegible]		
Normal probablity diagram	✓		
conc. vs. time	✓		



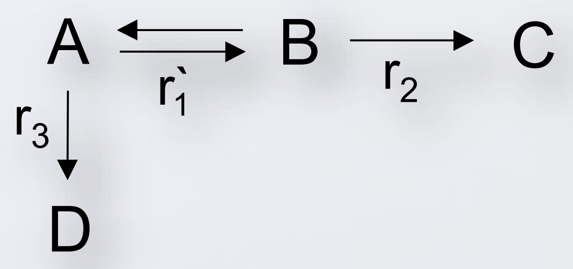


part2: batch reactor + nonisothermal data

$$\frac{dC_A}{dt} = \frac{-k_1 K_A\left(C_A - \frac{C_B}{K_{eq1}}\right) - k_3 K_A C_A}{1 + K_A C_A + K_B C_B}$$

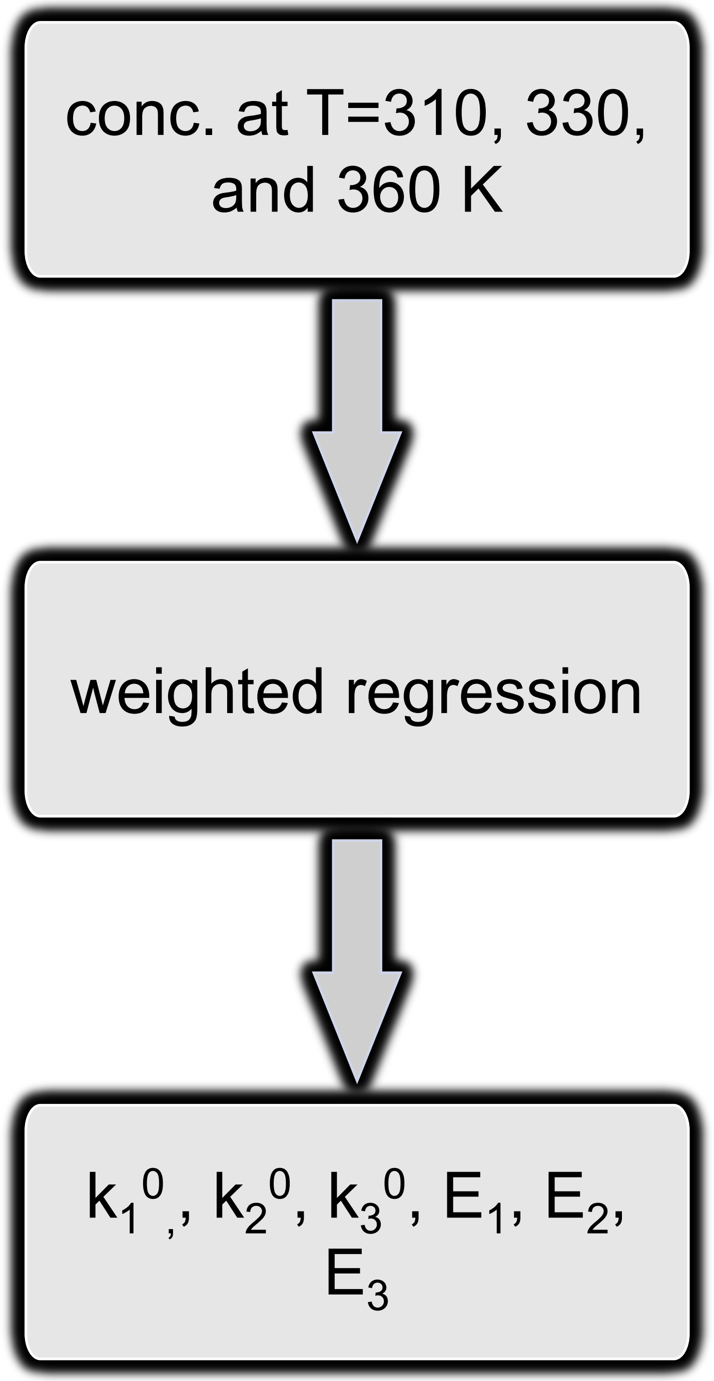
$$\frac{dC_C}{dt} = \frac{k_2 K_B C_B}{1 + K_A C_A + K_B C_B}$$

$$\frac{dC_D}{dt} = \frac{k_3 K_A C_A}{1 + K_A C_A + K_B C_B}$$



$$C_B = C_{A0} - C_A - C_C - C_D$$

$$K_i = k_i^0 exp\left(\frac{-E_i}{RT}\right) \qquad k_i = k_{avg} exp\left[\frac{-E_i}{R}\left(\frac{1}{T} - \frac{1}{T_{avg}}\right)\right]$$

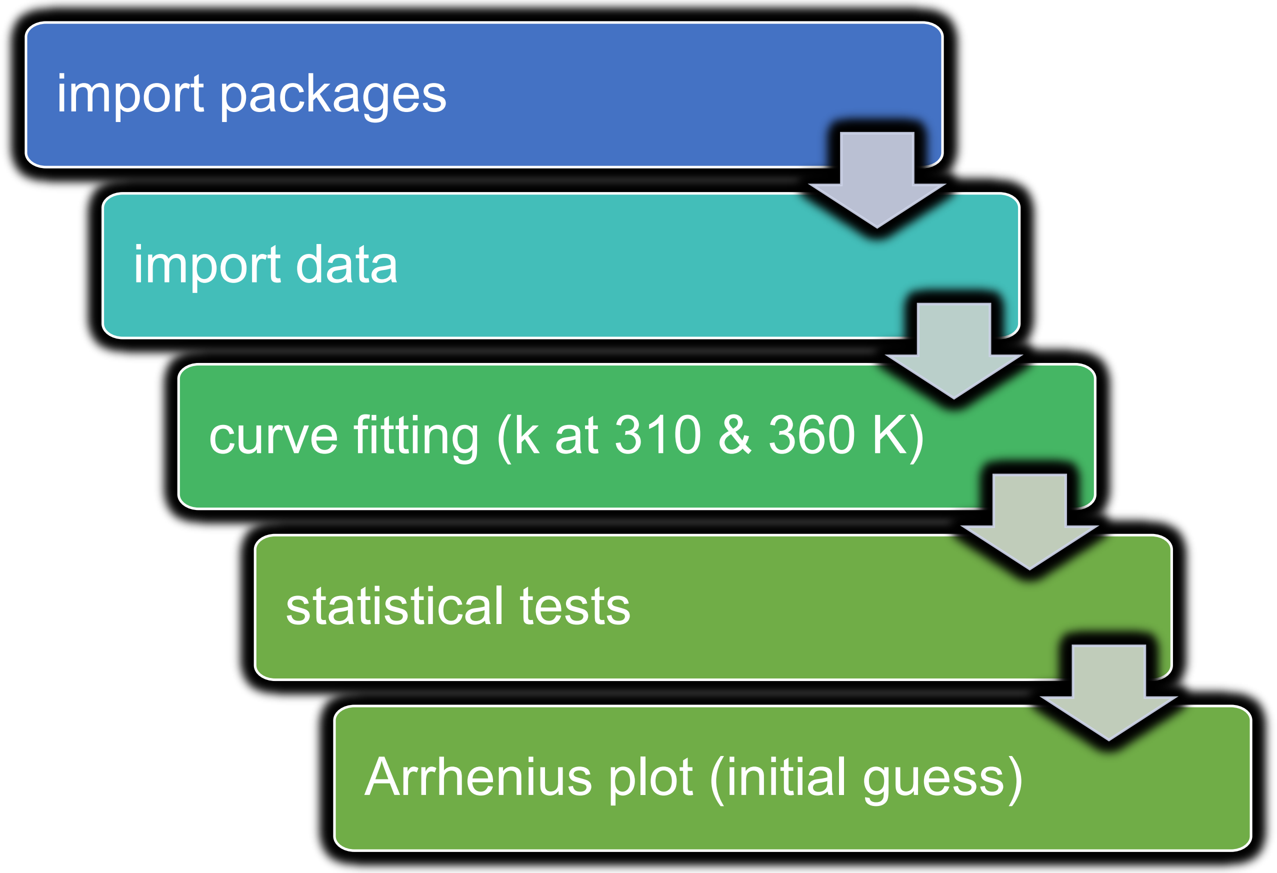


conc. at T=310, 330, and 360 K

weighted regression

k_1^0, k_2^0, k_3^0, E_1, E_2, E_3

part2: batch reactor + nonisothermal data (code)

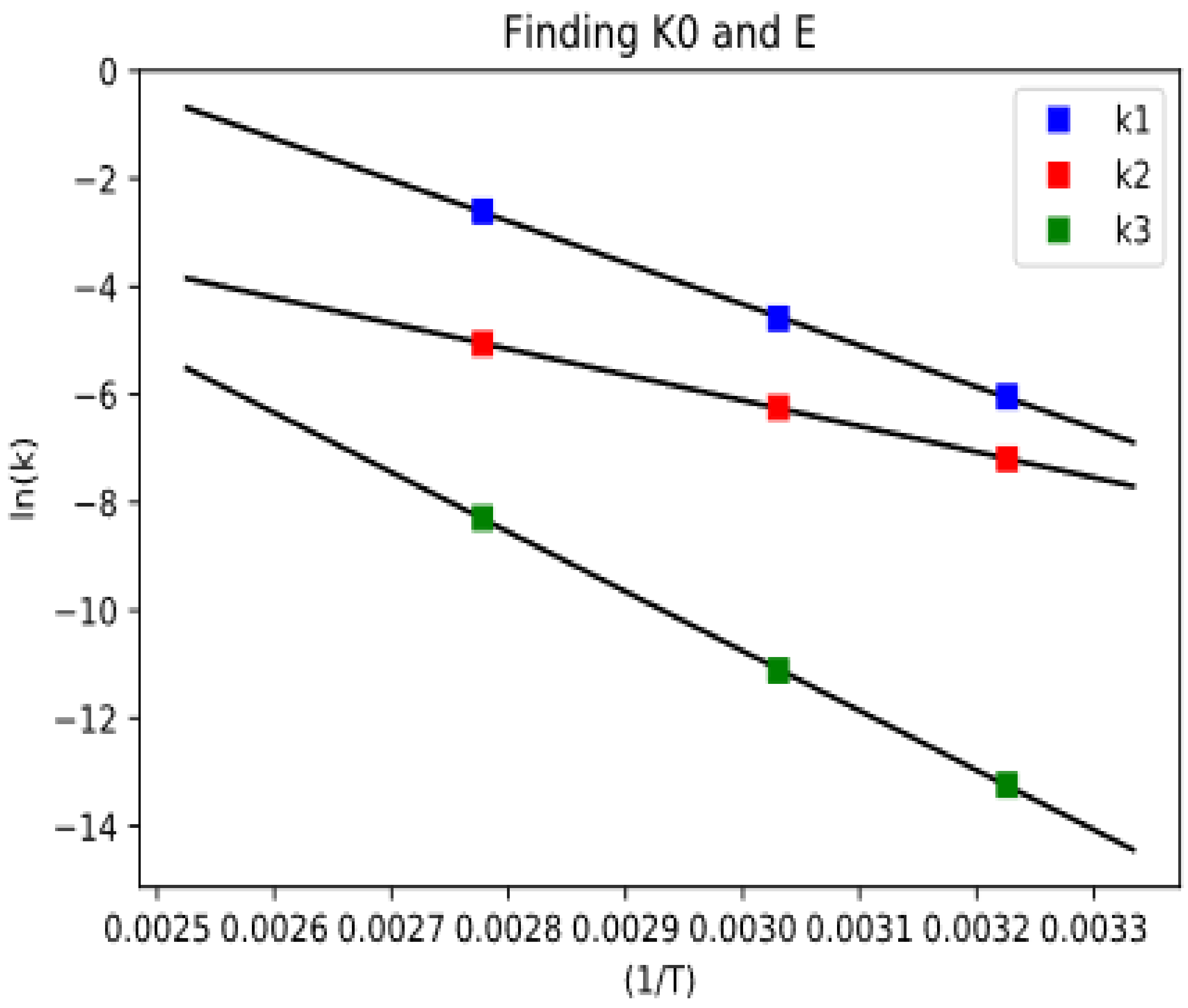

import packages

import data

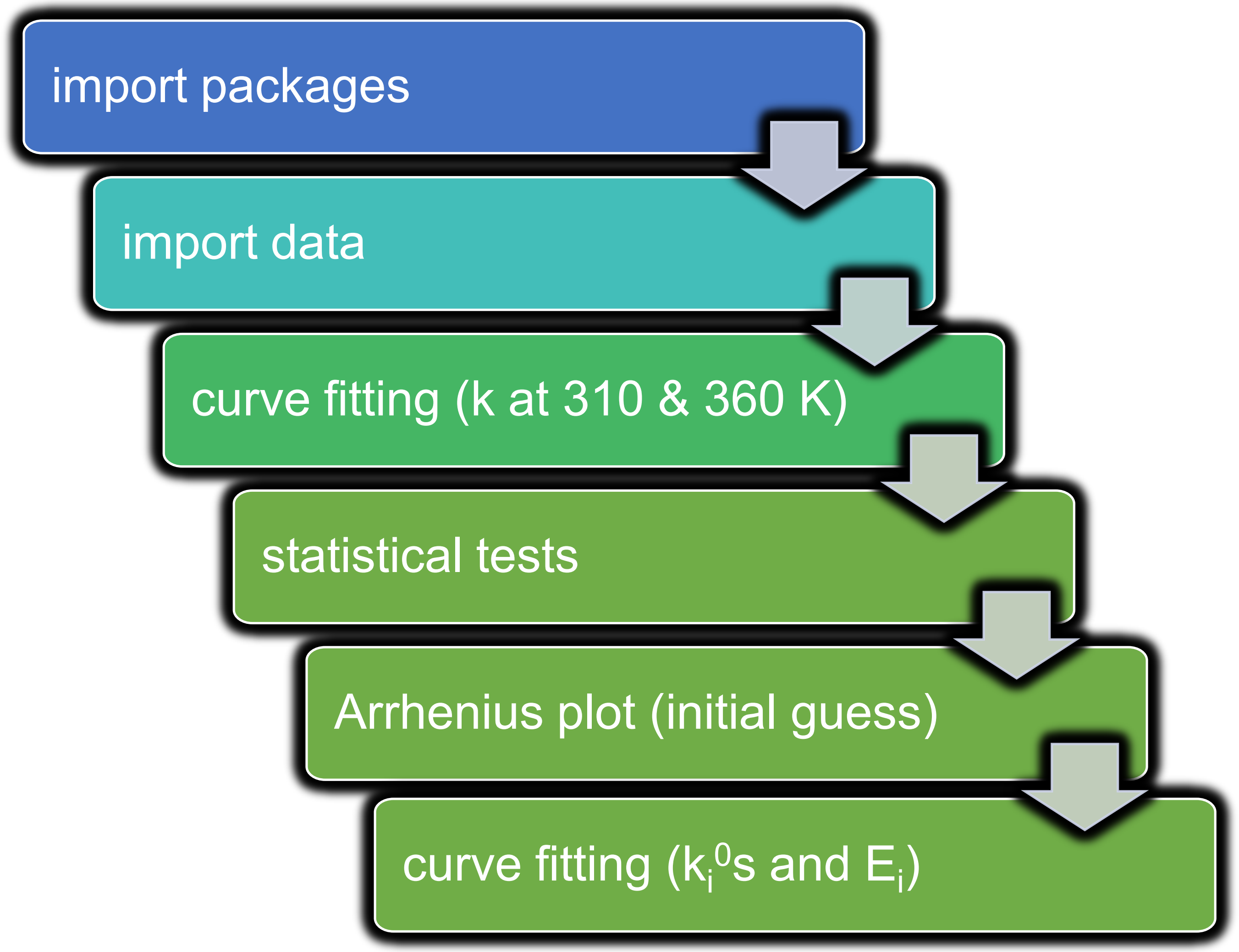
curve fitting (k at 310 & 360 K)

statistical tests

Arrhenius plot (initial guess)



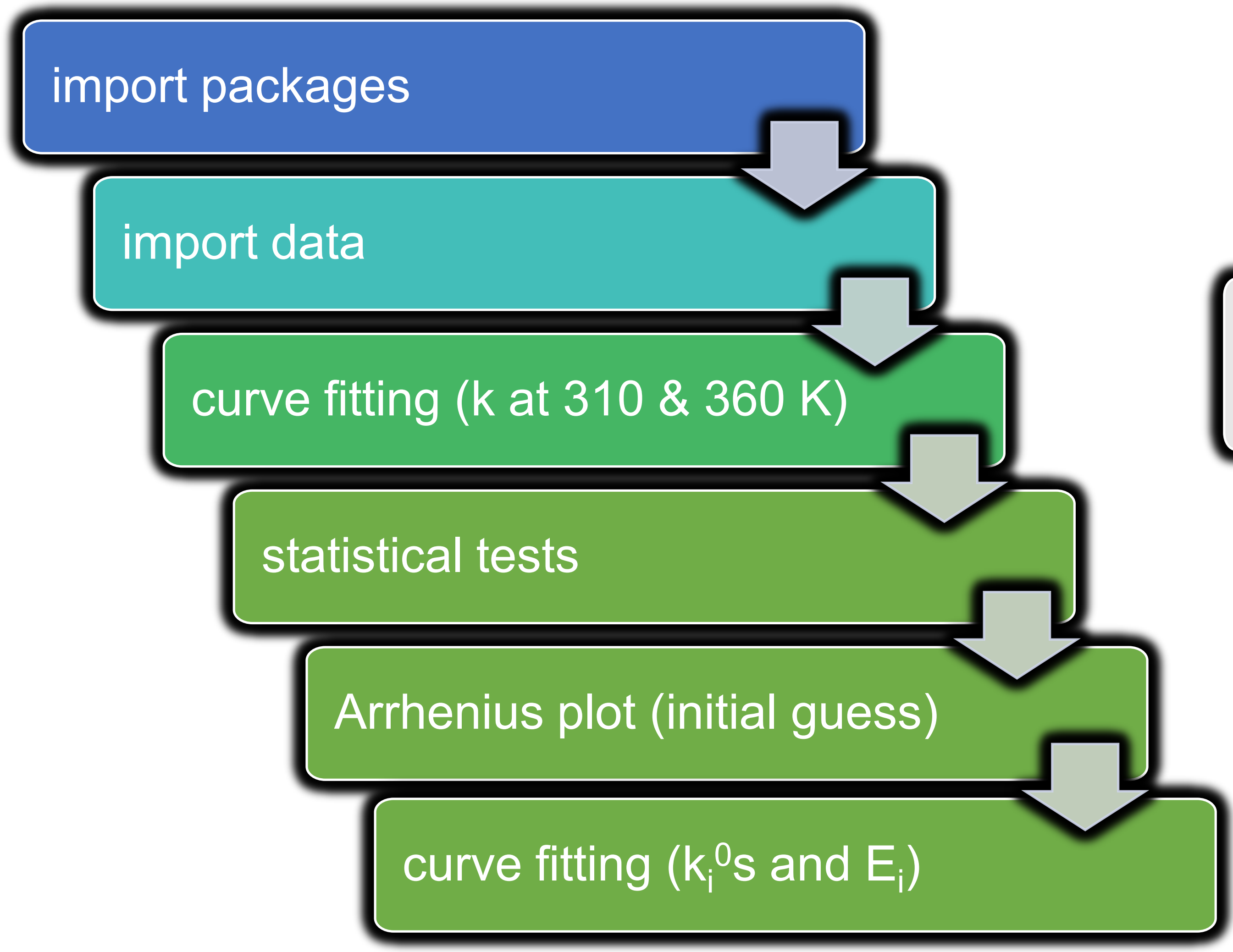
part2: batch reactor + nonisothermal data (code)


import packages
import data
curve fitting (k at 310 & 360 K)
statistical tests
Arrhenius plot (initial guess)
curve fitting (k_i^0s and E_i)

correlation coefficient matrix

	k_1^0	K_2^0	K_3^0	E_1	E_2	E_3
k_1^0	1.000	-0.416	0.396	0.999	-0.414	0.386
K_2^0	-0.416	1.000	-0.114	-0.429	0.999	-0.115
K_3^0	0.396	-0.114	1.000	0.392	-0.112	0.999
E_1	0.999	-0.429	0.392	1.000	-0.427	0.383
E_2	-0.414	0.999	-0.112	-0.427	1.000	-0.114
E_3	0.386	-0.115	0.999	0.383	-0.114	1.000

part2: batch reactor + nonisothermal data (code)



import packages

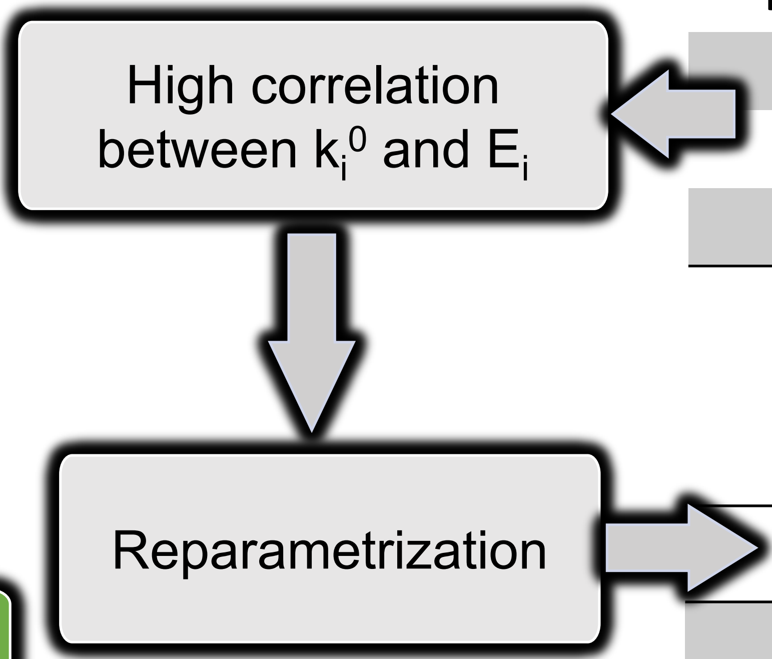
import data

curve fitting (k at 310 & 360 K)

statistical tests

Arrhenius plot (initial guess)

curve fitting (k_i^0s and E_i)



High correlation between k_i^0 and E_i

Reparametrization

correlation coefficient matrix

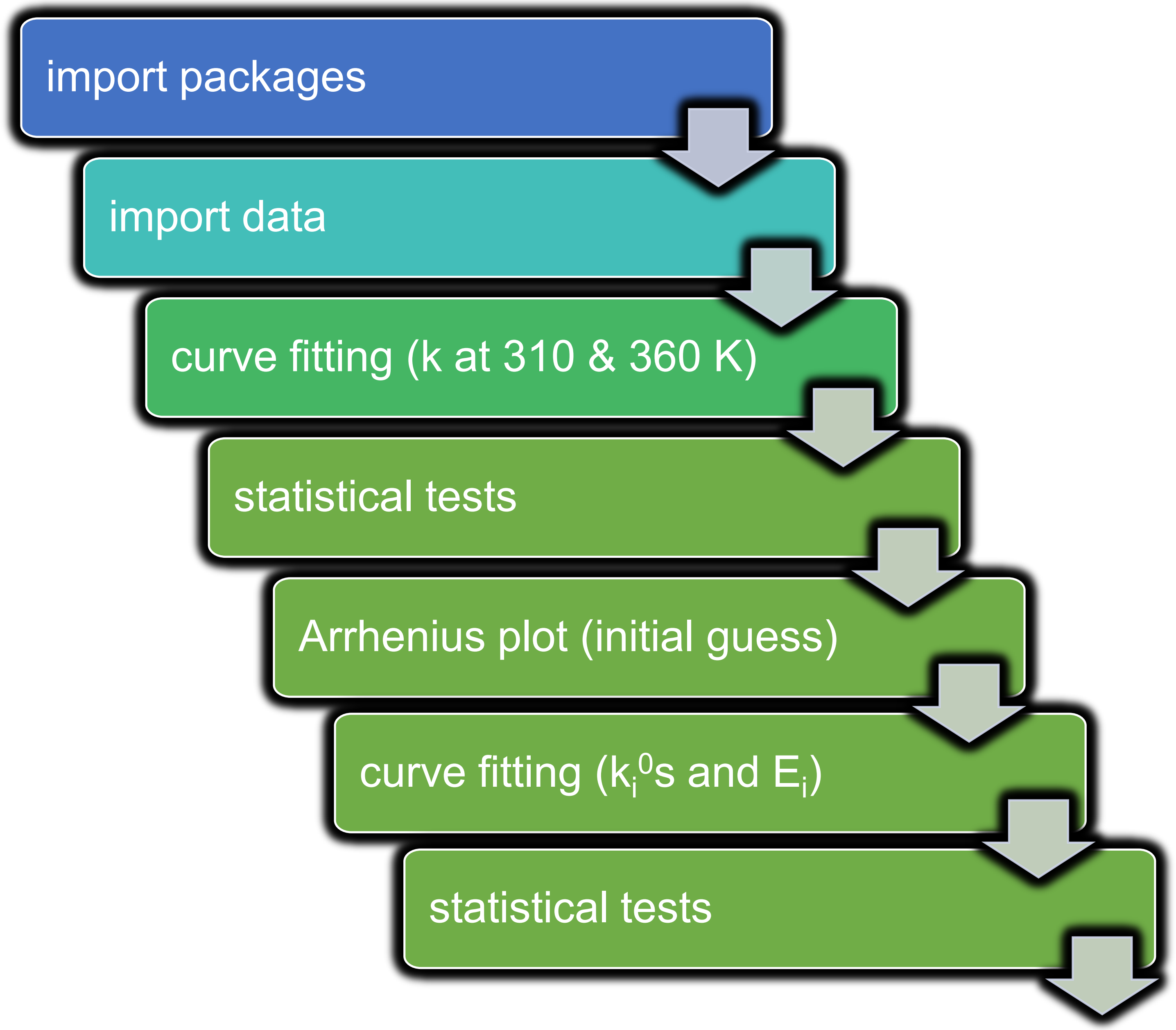
	k_1^0	K_2^0	K_3^0	E_1	E_2	E_3
k_1^0	1.000	-0.416	0.396	0.999	-0.414	0.386
K_2^0	-0.416	1.000	-0.114	-0.429	0.999	-0.115
K_3^0	0.396	-0.114	1.000	0.392	-0.112	0.999
E_1	0.999	-0.429	0.392	1.000	-0.427	0.383
E_2	-0.414	0.999	-0.112	-0.427	1.000	-0.114
E_3	0.386	-0.115	0.999	0.383	-0.114	1.000

$$k_i = k_{avg} exp\left[\frac{-E_i}{R}\left(\frac{1}{T} - \frac{1}{T_{avg}}\right)\right]$$

correlation coefficient matrix

	k_1^0	K_2^0	K_3^0	E_1	E_2	E_3
k_1^0	1.000	-0.356	0.408	0.401	0.135	0.239
K_2^0	-0.356	1.000	-0.099	0.225	-0.609	0.045
K_3^0	0.408	-0.099	1.000	0.263	0.007	0.200
E_1	0.401	0.225	0.263	1.000	-0.427	0.377
E_2	0.135	-0.609	0.007	-0.427	1.000	-0.120
E_3	0.239	0.045	0.200	0.377	-0.120	1.000

part2: batch reactor + nonisothermal data (code)



import packages

import data

curve fitting (k at 310 & 360 K)

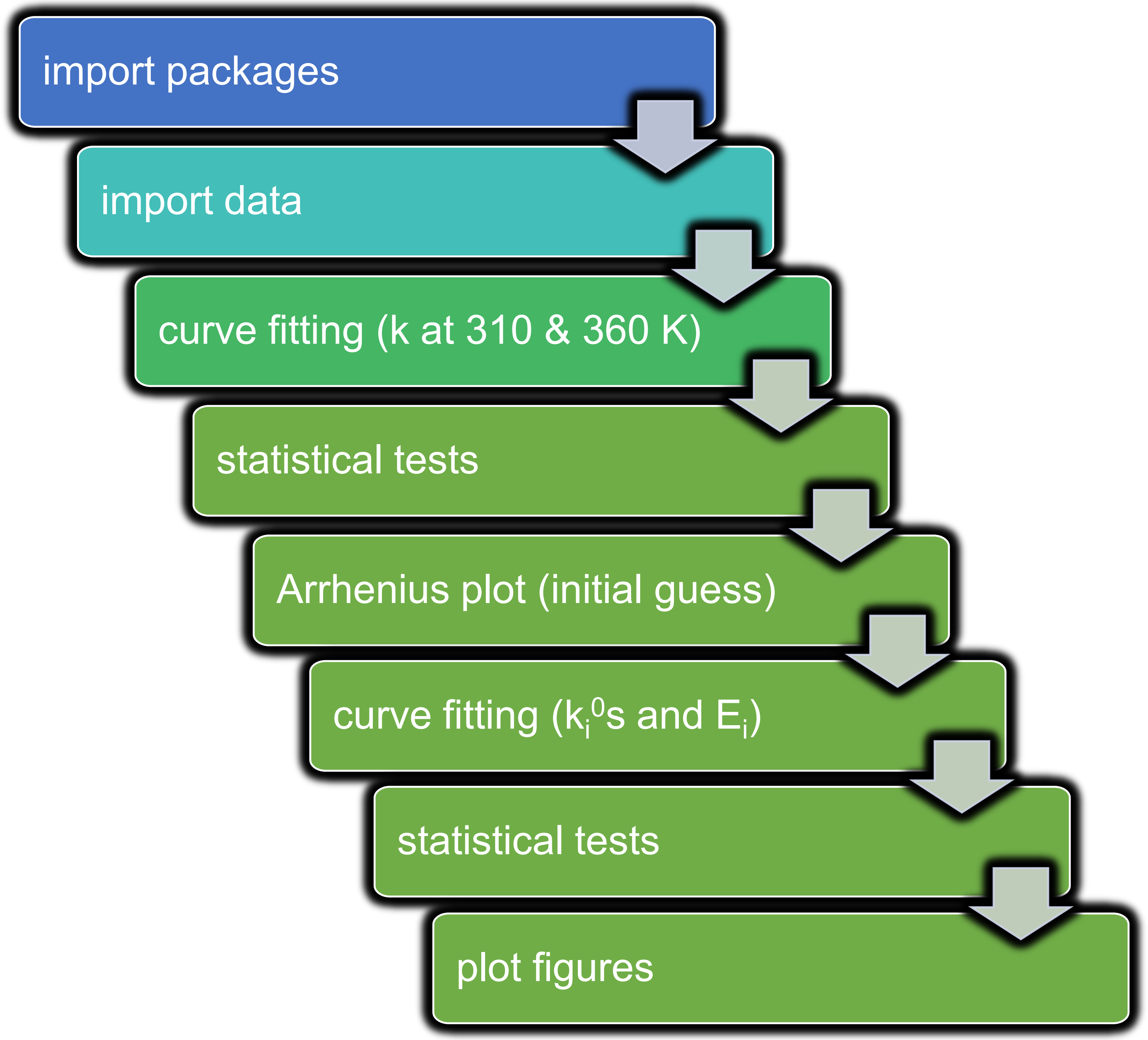
statistical tests

Arrhenius plot (initial guess)

curve fitting (k_i^0s and E_i)

statistical tests

part2: batch reactor + nonisothermal data (code)



import packages

import data

curve fitting (k at 310 & 360 K)

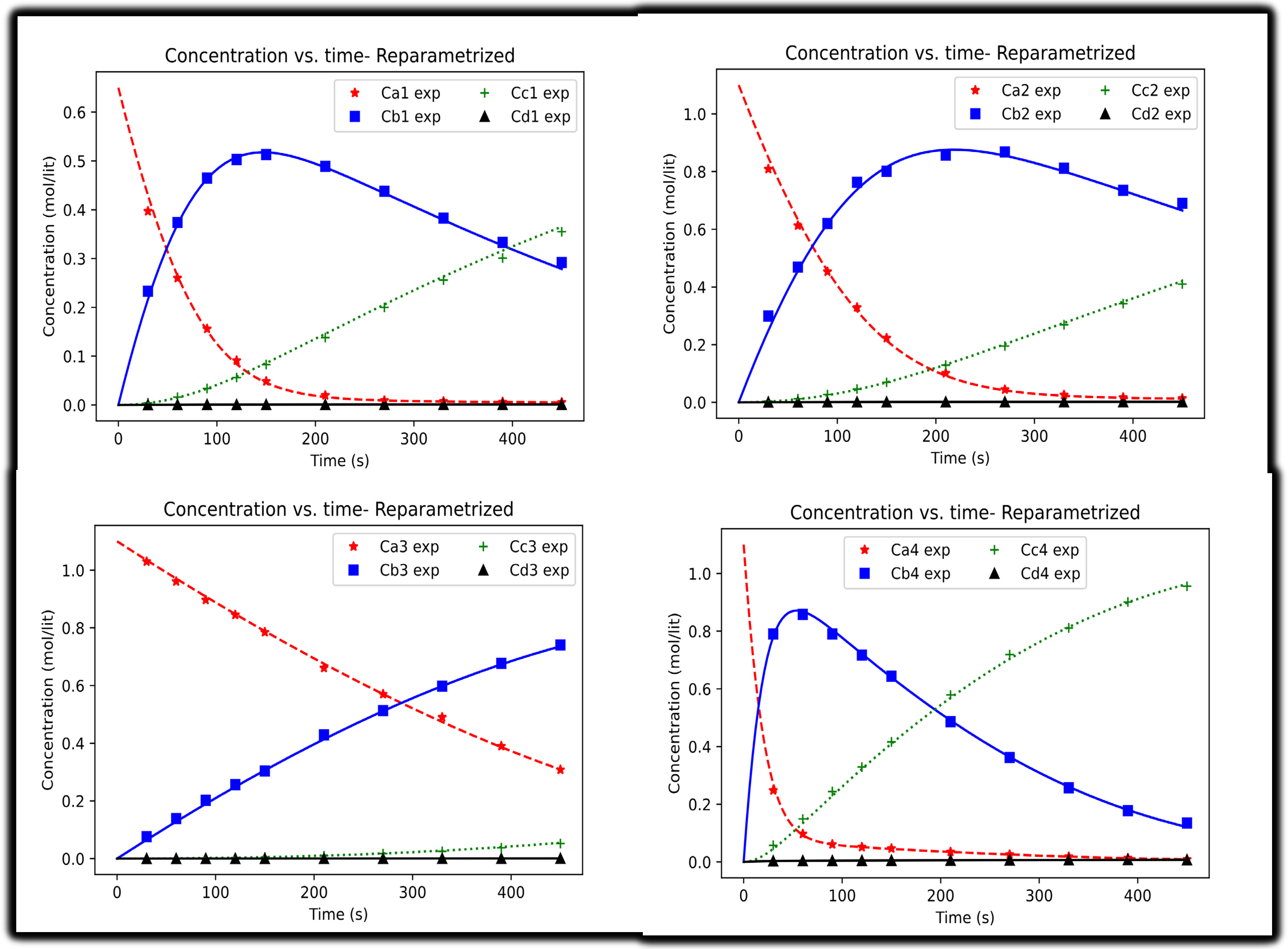
statistical tests

Arrhenius plot (initial guess)

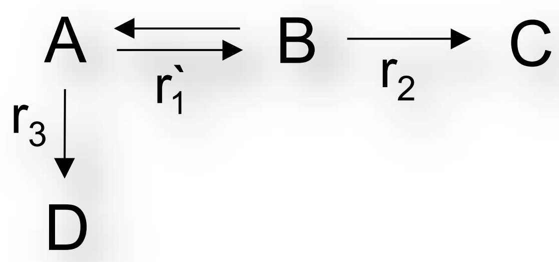
curve fitting (k_i^0s and E_i)

statistical tests

plot figures



part3: CSTR isothermal data+ all data

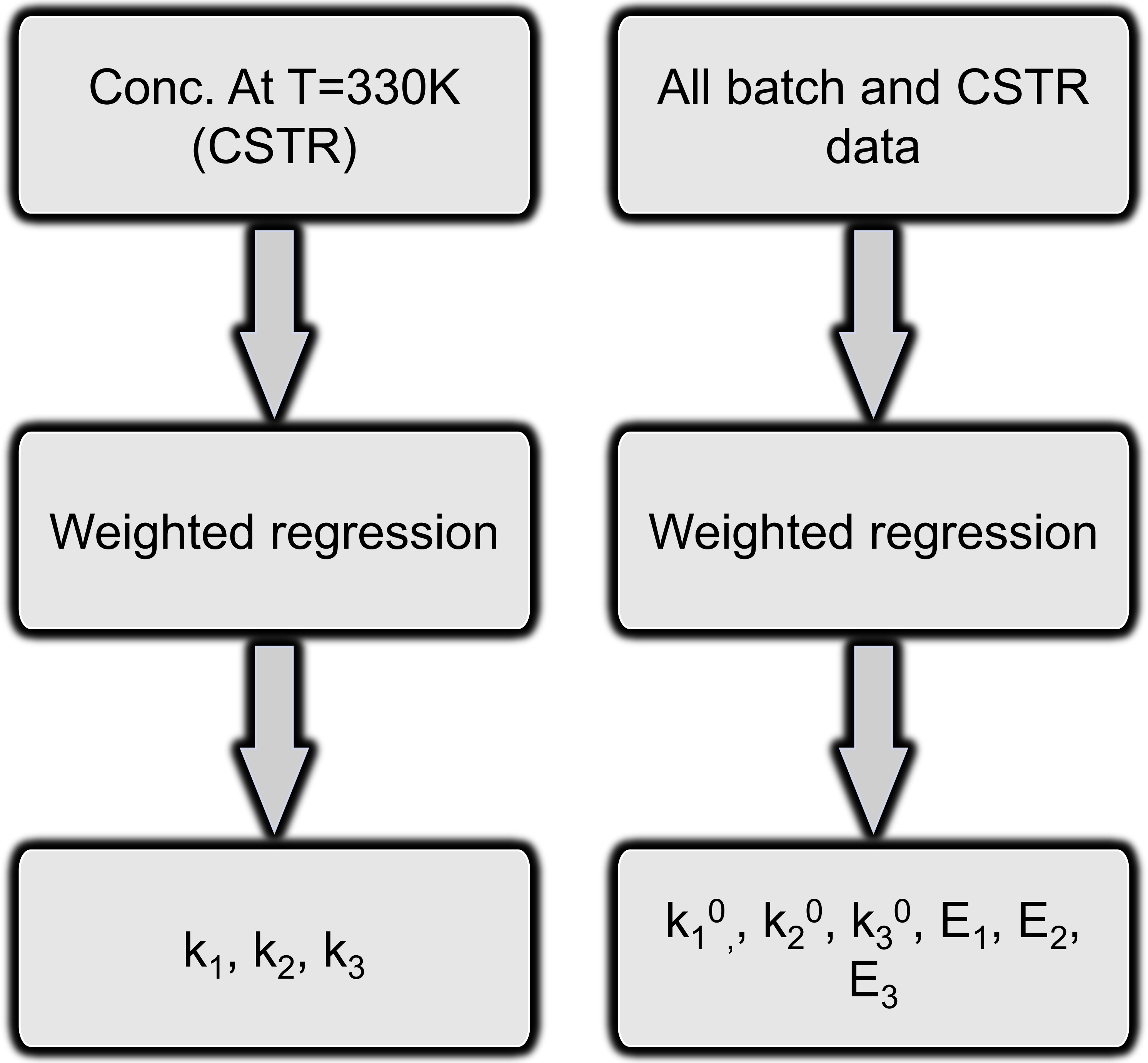


$$C_A = C_{A0}\frac{\left[-k_1K_A\left(C_A-\frac{C_B}{K_{eq1}}\right)-k_3K_AC_A\right]\tau}{1+K_AC_A+K_BC_B}$$

$$C_C = \frac{(k_2K_BC_B)\tau}{1+K_AC_A+K_BC_B}$$

$$C_D = \frac{(k_3K_AC_A)\tau}{1+K_AC_A+K_BC_B}$$

$$C_B = C_{A0}-C_A-C_C-C_D$$


Conc. At T=330K (CSTR)

Weighted regression

k_1, k_2, k_3

All batch and CSTR data

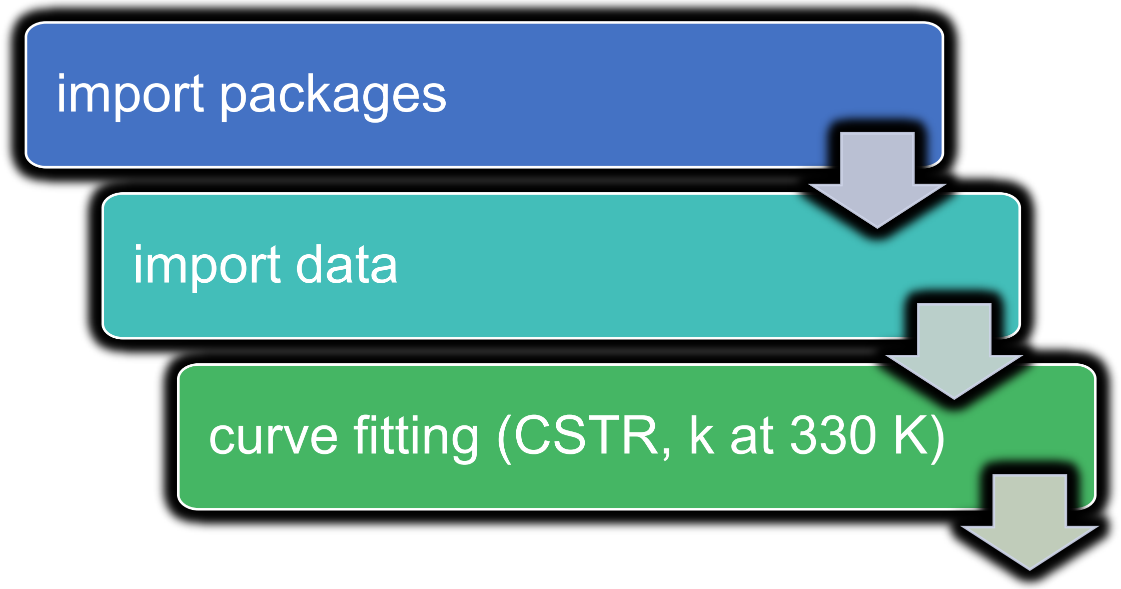
Weighted regression

k_1^0, k_2^0, k_3^0, E_1, E_2, E_3

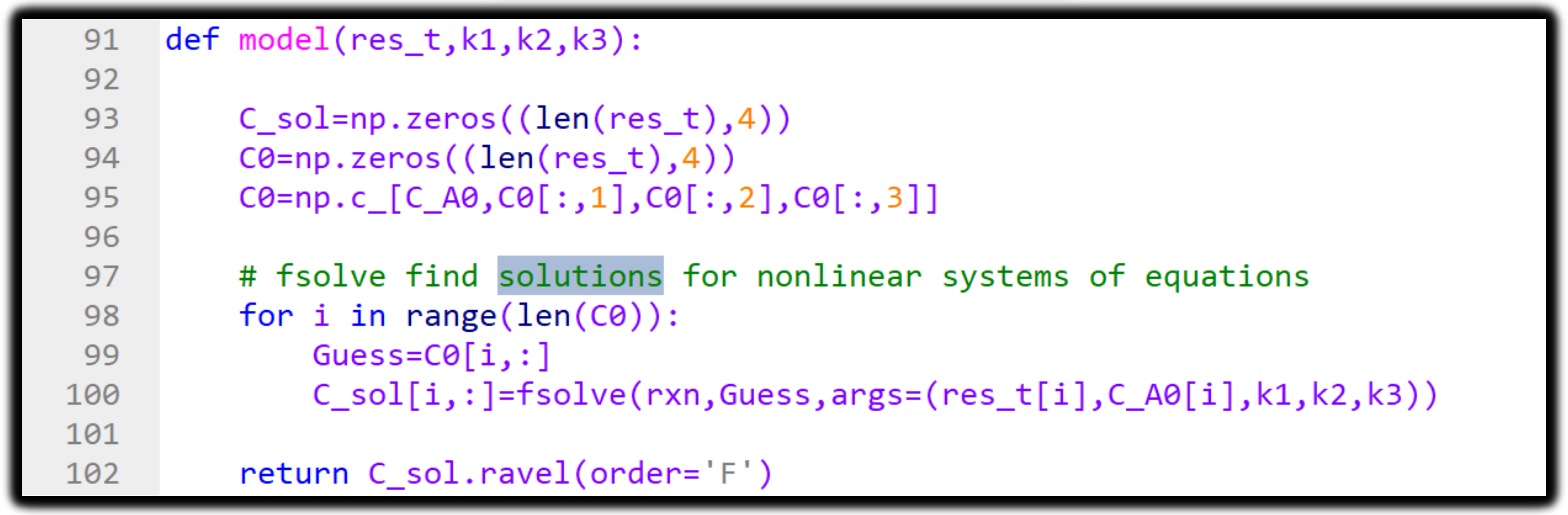
GHENT UNIVERSITY

LCT DRIVING CHEMICAL TECHNOLOGY

part3: CSTR isothermal data+ all data (code)

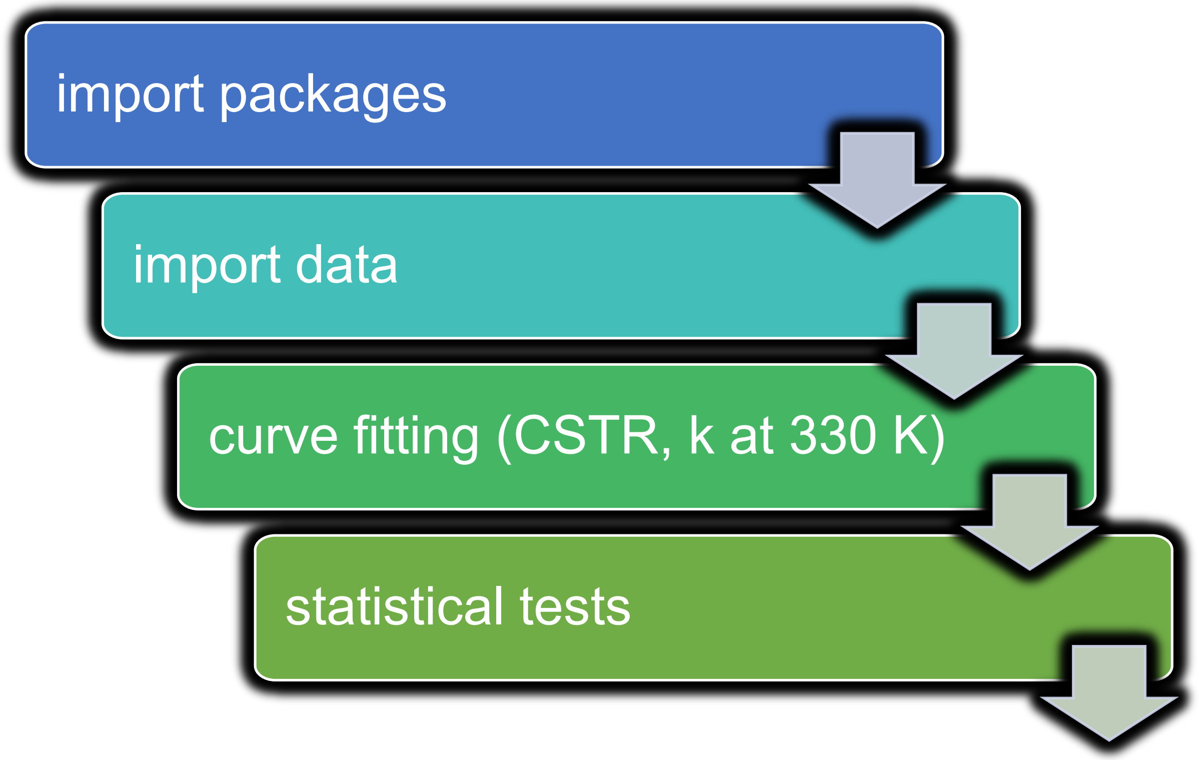


Systems of algebraic equations



```python
def model(res_t,k1,k2,k3):

    C_sol=np.zeros((len(res_t),4))
    C0=np.zeros((len(res_t),4))
    C0=np.c_[C_A0,C0[:,1],C0[:,2],C0[:,3]]

    # fsolve find solutions for nonlinear systems of equations
    for i in range(len(C0)):
        Guess=C0[i,:]
        C_sol[i,:]=fsolve(rxn,Guess,args=(res_t[i],C_A0[i],k1,k2,k3))

    return C_sol.ravel(order='F')
```

part3: CSTR isothermal data+ all data (code)



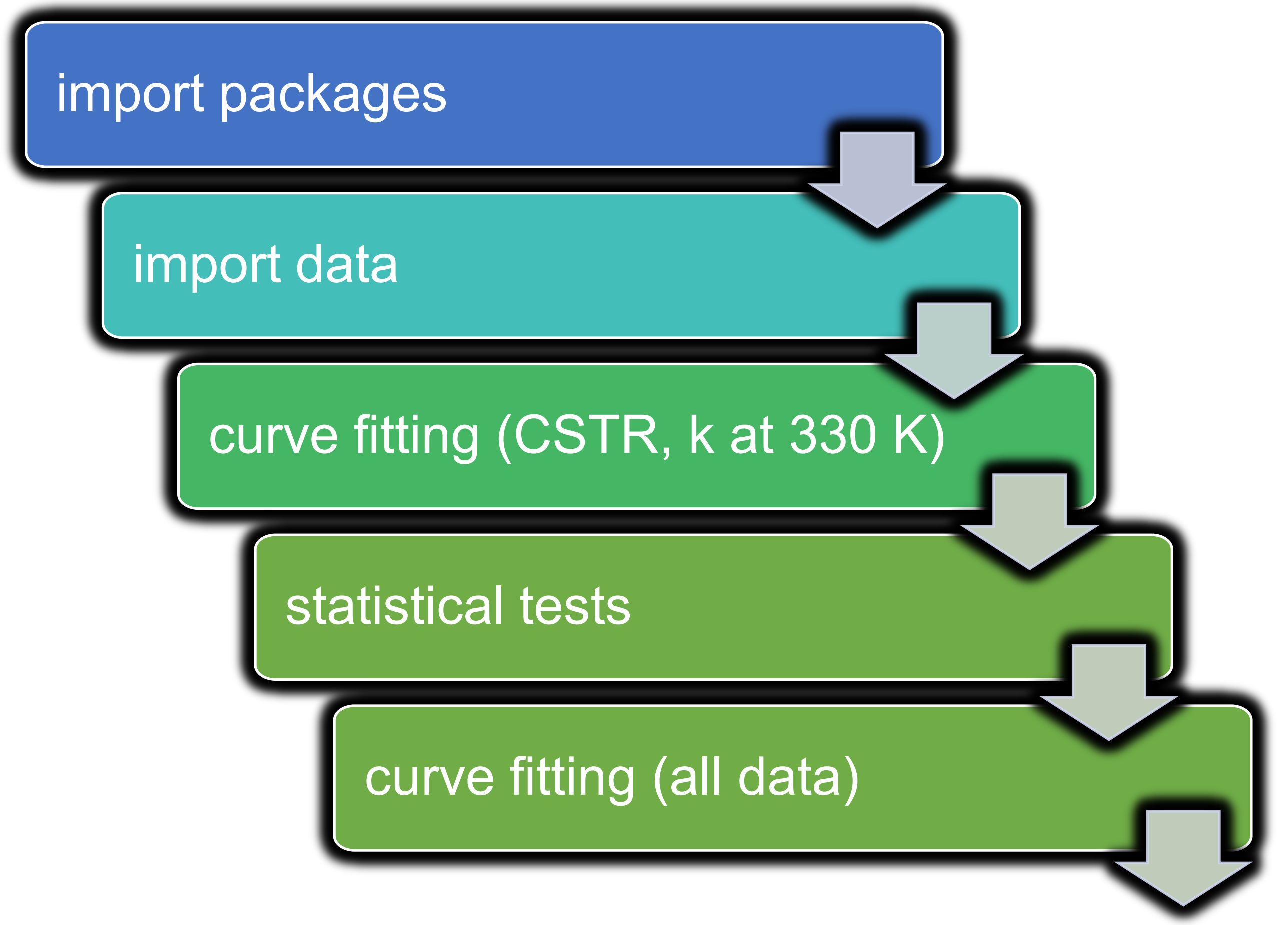
import packages

import data

curve fitting (CSTR, k at 330 K)

statistical tests

part3: CSTR isothermal data+ all data (code)



import packages

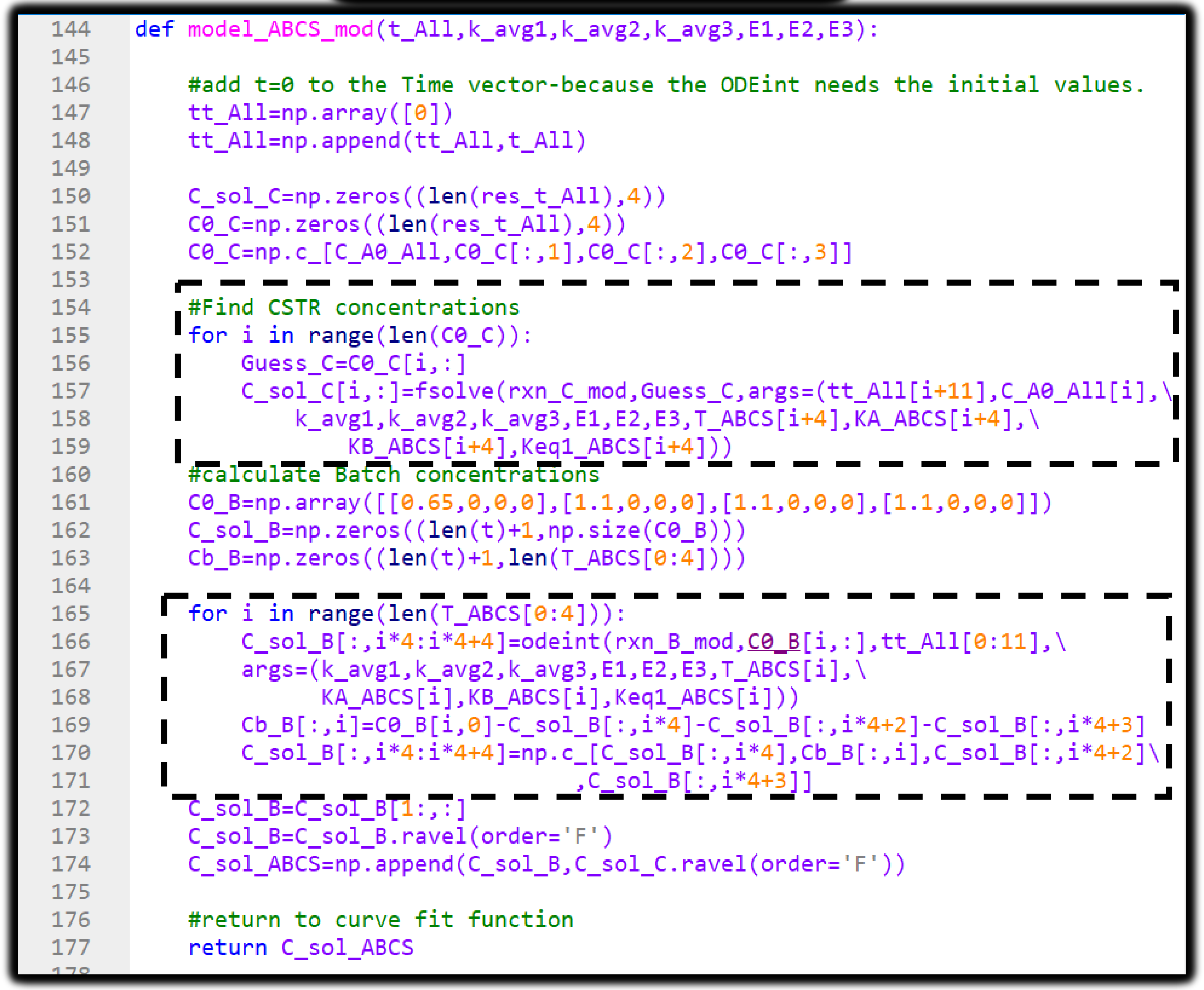
import data

curve fitting (CSTR, k at 330 K)

statistical tests

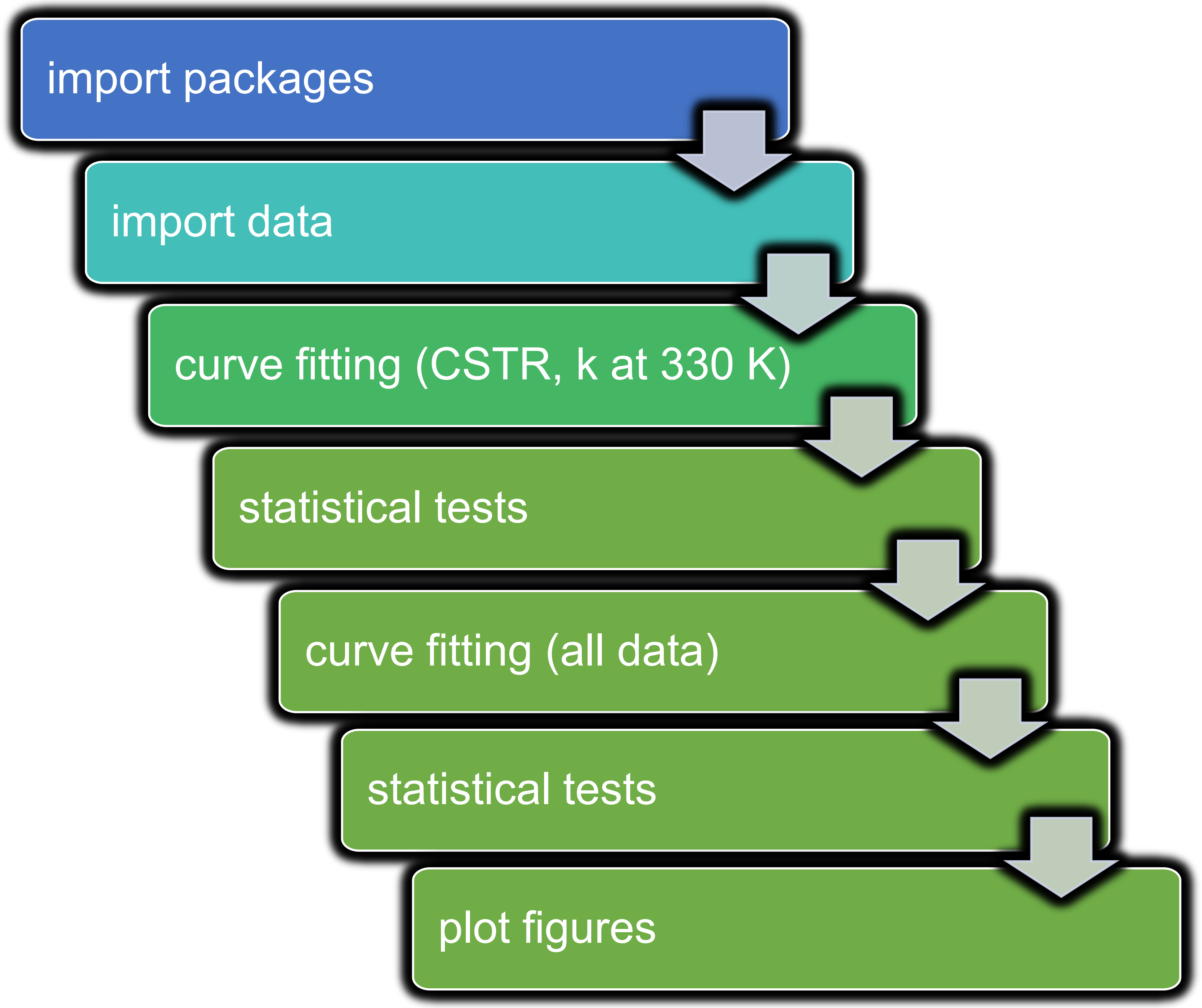
curve fitting (all data)

ODE + Algebraic equations



```python
def model_ABCS_mod(t_All,k_avg1,k_avg2,k_avg3,E1,E2,E3):

    #add t=0 to the Time vector-because the ODEint needs the initial values.
    tt_All=np.array([0])
    tt_All=np.append(tt_All,t_All)

    C_sol_C=np.zeros((len(res_t_All),4))
    C0_C=np.zeros((len(res_t_All),4))
    C0_C=np.c_[C_A0_All,C0_C[:,1],C0_C[:,2],C0_C[:,3]]

    #Find CSTR concentrations
    for i in range(len(C0_C)):
        Guess_C=C0_C[i,:]
        C_sol_C[i,:]=fsolve(rxn_C_mod,Guess_C,args=(tt_All[i+11],C_A0_All[i],\
            k_avg1,k_avg2,k_avg3,E1,E2,E3,T_ABCS[i+4],KA_ABCS[i+4],\
                KB_ABCS[i+4],Keq1_ABCS[i+4]))
    #calculate Batch concentrations
    C0_B=np.array([[0.65,0,0,0],[1.1,0,0,0],[1.1,0,0,0],[1.1,0,0,0]])
    C_sol_B=np.zeros((len(t)+1,np.size(C0_B)))
    Cb_B=np.zeros((len(t)+1,len(T_ABCS[0:4])))

    for i in range(len(T_ABCS[0:4])):
        C_sol_B[:,i*4:i*4+4]=odeint(rxn_B_mod,C0_B[i,:],tt_All[0:11],\
        args=(k_avg1,k_avg2,k_avg3,E1,E2,E3,T_ABCS[i],\
              KA_ABCS[i],KB_ABCS[i],Keq1_ABCS[i]))
        Cb_B[:,i]=C0_B[i,0]-C_sol_B[:,i*4]-C_sol_B[:,i*4+2]-C_sol_B[:,i*4+3]
        C_sol_B[:,i*4:i*4+4]=np.c_[C_sol_B[:,i*4],Cb_B[:,i],C_sol_B[:,i*4+2]\
                                 ,C_sol_B[:,i*4+3]]
    C_sol_B=C_sol_B[1:,:]
    C_sol_B=C_sol_B.ravel(order='F')
    C_sol_ABCS=np.append(C_sol_B,C_sol_C.ravel(order='F'))

    #return to curve fit function
    return C_sol_ABCS
```

part3: CSTR isothermal data+ all data (code)



import packages

import data

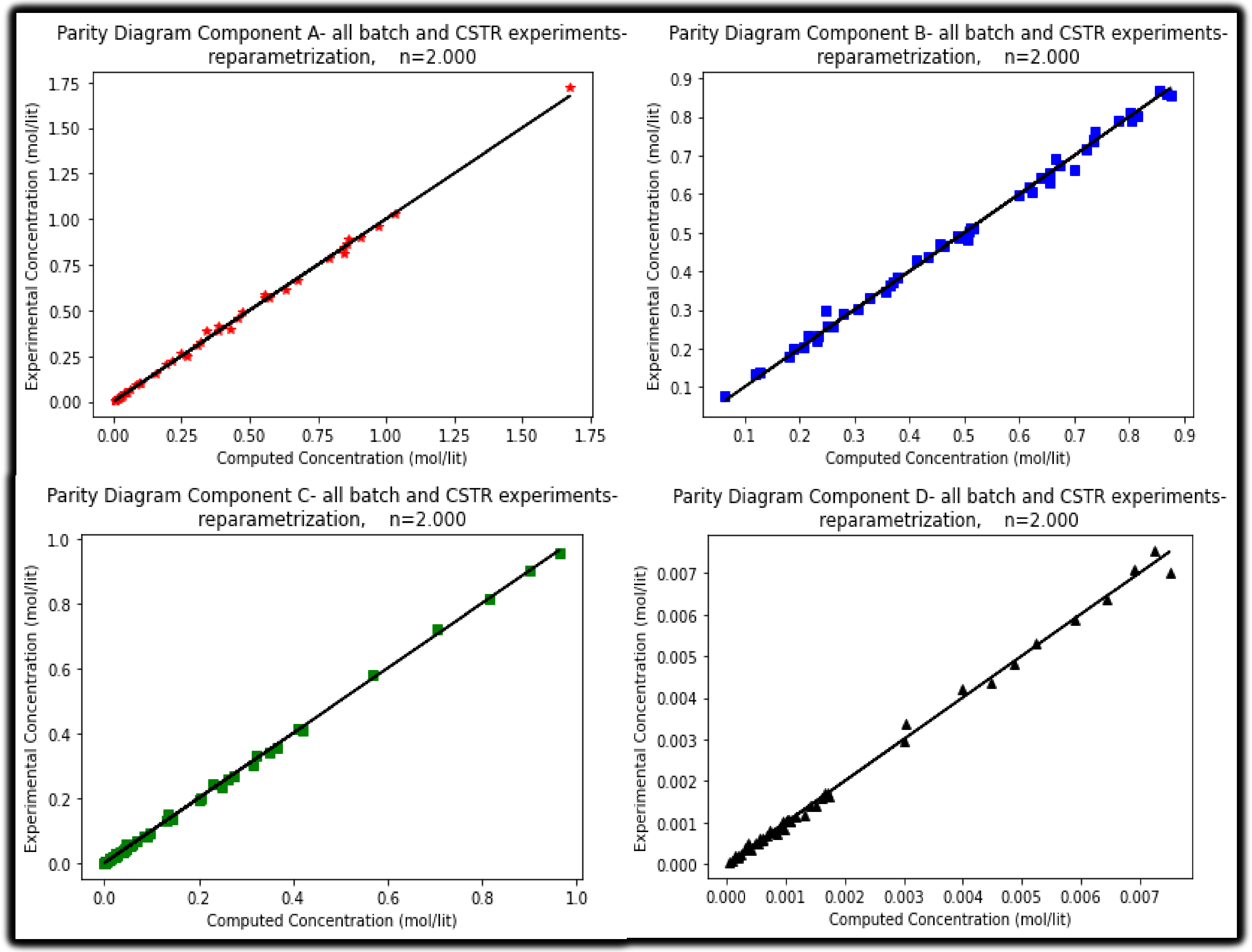
curve fitting (CSTR, k at 330 K)

statistical tests

curve fitting (all data)

statistical tests

plot figures



Parity Diagram Component A- all batch and CSTR experiments- reparametrization, n=2.000

Parity Diagram Component B- all batch and CSTR experiments- reparametrization, n=2.000

Parity Diagram Component C- all batch and CSTR experiments- reparametrization, n=2.000

Parity Diagram Component D- all batch and CSTR experiments- reparametrization, n=2.000

overview

- ✓ introduction to Python
- ✓ How to work with Python?
- ✓ Python for numerical programming
- ✓ EUROKIN parameter estimation-case 1
- ➡ complex system
- ○ summary

complex system

mass transport equations

heat transport equations

momentum transfer equations

reaction

FEniCS project

SfePy

overview

- ✓ introduction to Python
- ✓ How to work with Python?
- ✓ Python for numerical programming
- ✓ EUROKIN parameter estimation-case 1
- ✓ complex system
- ➡ summary

summary

advantages	free	open source	beginner-friendly	easy to import export, & visualize
operating system	Windows	Linux	MacOS	
help	powerful	inside the software and online		
solution to equations	AE	ODE	PDE	
optimization	available in Scipy package	Scipy.optimize		
statistical calculation	available in Scipy package	Scipy.stats		

LABORATORY FOR CHEMICAL TECHNOLOGY

Technologiepark 125, 9052 Ghent, Belgium

E info.lct@ugent.be
T 003293311757

https://www.lct.ugent.be

GHENT
UNIVERSITY

LCT
DRIVING CHEMICAL TECHNOLOGY