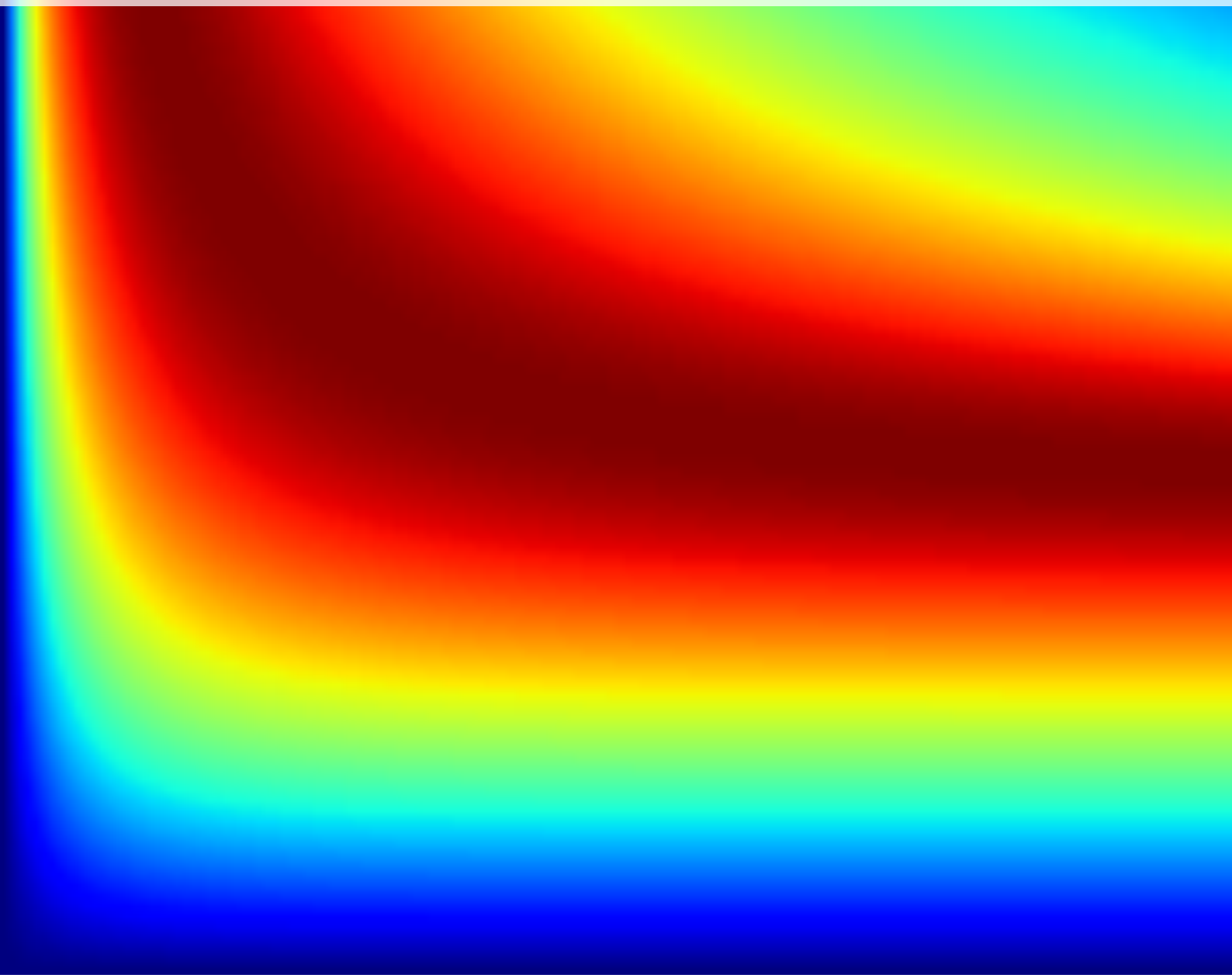
06

Modeling in Chemical Technology

Examples of Numerical Calculations with Python

Szczepan Bednarz

Cracow University of Technology, Poland



Politechnika Krakowska im. Tadeusza Kościuszki

InnoChem

Erasmus+

Erasmus+ Project - InnoChem 2014-1-PL01-KA203-003415

Modeling in Chemical Technology

Examples of Numerical Calculations with Python

16th October 2016

v1.1

The repository of the course: https://github.com/sbednarz/O6

Szczepan Bednarz sbednarz@pk.edu.pl.

Funded by the
Erasmus+ Programme
of the European Union

The European Commission support for the production of this publication does not constitute endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Contents

1 Introduction

Overview

This is a collection of solved problems and exercises in chemical technology. The examples show how to use numerical calculations with `Python` to create simple models of not so simple chemical systems and processes. The collection is a part of "Modeling in chemical technology (O6)" computational course.

Technical issues

All of the examples in this book were created and tested with the Anaconda Python v2.7. However, the `Python` code could be relatively easy ported to other computational platforms including `Octave`, `Matlab`, `MathCad`, `Maple` or `Mathematica`. Additionally, translation of `Polymath` scripts (very popular in the USA universities) into `Python` code seems to be fairly easy (see examples 39 and 44).

To play with `Python` calculations just visit the site: http://tmpnb.org or install any `Python` distribution on your personal computer.

Acknowledgements

Development of this course was funded by the Erasmus+ Programme of the European Union, grant agreement number: 2014-1-PL01-KA203-003415 "Improvement of innovative teaching methods in the fields of Technology and Chemical Engineering according to the best standards of the Bologna Process".

2 Computational Methods

Example 1. Quick Introduction to Python

● ○ ○

The aim of this example is just a quick very simplified introduction to Python. Please study also complete python tutorials. The most recommended are [1][2][3].

Run the program:

```
# File 1-1.py [View]
# Example 1. Quick Introduction to Python

# A line fragment started with hash sign (#) is a comment
# You can explain your code and write here your notes
# Anything after the # is ignored by python.
# You can also use a comment to "disable" or comment out a piece of code:
# print "This won't run."

print "This will run."  # this is comment, too
```

The program should print:

```
%run 1-1.py
This will run.
```

As you can see, to print a message, you can use in Python script `print` function.

Python as true programming language, has variables. Variables are names of containers for e.g. numbers. Run the script:

```
# File 1-2.py [View]
# Example 1. Quick Introduction to Python

# Variables definition:
a = 4
b = 4.56
c = 3.4e-1 # =3.4x10^-1 = 0.34

# To show variables content you can use 'print' function:
print(a)
print(b)
print(c)

# You can also write:
d = 45.23
print( d )

# Spaces in the middle of a line code are not important for python, but
# could make the code more readable by human. Use them.
# Please note! Spaces at THE BEGINNING of a line are VERY IMPORTANT for python
# This issue will be explained later.
```

You should see:

```
%run 1-2.py
4
4.56
0.34
45.23
```

Arithmetic operations are fairly simple in Python. Run some calculations:

```
# File 1-3.py [View]
# Example 1. Quick Introduction to Python

# Arithmetic calculations
a = 2
b = 3
print(a+b)

c = a + b
print(c)

d = a + b*a
print(d)

d = 2**8 # 2 power 8
print(d)

d = (a + 1.3*a)/a - 1
print(d)
# Please don't forget about multiply sign (*) and
# put braces into your expressions if necessary.
```

You should see:

```
%run 1-3.py
5
5
8
256
1.3
```

To make your programs more verbose, you can print informations:

```
# File 1-4.py [View]
# Example 1. Quick Introduction to Python

# Variables can contain also text strings
a = 'I like'
b = ' Python'
print(a)
print(b)

# To join strings just put +
c = a+b
print(c)

# You can print text messages:
print('Wow!')
print("Wow "+b+"!")
# Please note quotation marks ' "
```

You should see:

```
%run 1-4.py
I like
 Python
I like Python
Wow!
Wow  Python!
```

Finally, here is a complete script. It does one trivial think - calculate molar concentration based on mole number and volume of a solution:

```
# File 1-5.py [View]
# Example 1. Quick Introduction to Python

# More complete script to calculate molar concentration (c)
# calculations first
n = 0.4 # mol     <--- a comment as an unit name, just for code clarity
V = 1.6 # L
c = n/V # mol/L
# print a report
print('n=')
print(n)
print('V=')
print(V)
print('Molar concentration=')
print(c)
```

The output (report) is:

```
%run 1-5.py
n=
0.4
V=
1.6
Molar concentration=
0.25
```


[1] M. Scott Shell. *An introduction to Python for scientific computing*. URL: http://www.engr.ucsb.edu/~shell/che210d/python.pdf.

[2] Wikibooks. *Python Programming*. URL: https://en.wikibooks.org/wiki/Python_Programming.

[3] Zed A. Shaw. *Learn Python the Hard Way*. URL: http://learnpythonthehardway.org/book/index.html.

Example 2. Simple Calculations with Python

● ○ ○

From the Example 1 you have learned how to use one of basic Python function: `print`. To perform mathematical calculations you need more specific functions. In Python, specialized functions are grouped together in a package - a module. To use them you should import whole module or import a function from a module:

```
# File 2-1.py [View]
# Example 2. Simple Calculations with Python

# Importing of whole module and giving alias 'np'
# The alias could be any (short) name you wish
import numpy as np

# or importing one function
from numpy import sin

# Now you can use mathematical functions:
a = sin(1)     # function
b = np.cos(1)  # function call is: the alias - dot - the name
print(a)
print(b)
```

Module `numpy` is a huge package and it provides a bunch of useful mathematics and matrix operations [1], just as example:

```
# File 2-2.py [View]
# Example 2. Simple Calculations with Python

import numpy as np

# Most important math functions living in numpy module are:
print( np.log10(10) )       # logarithm with base 10
print( np.log(10) )         # natural logarithm
print( np.sqrt(2) )         # square
print( np.sin( np.pi ) )    # sin & pi constant
```

and the results:

```
%run 2-2.py
1.0
2.30258509299
1.41421356237
1.22464679915e-16
```

Please note that result of `sin(pi)` expression is a very very small value of 1.22e-16 - in numerical calculations we called the value 0 (zero)

[1] *NumPy Documentation*. URL: https://docs.scipy.org/doc/numpy/reference/routines.html.

Example 3. Calculations with Python

● ○ ○

List is a Python object (variable) to store multiple content. Each of element in a list has individual index. Python counts from zero. Here is how to create lists and get/set a value in a list.

```
# File 3-4.py [View]
# Example 3. Calculations with Python
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

a = []               # create empty list
b = [0, 21]          # create two-elements list
c = [1.3, 1.3e-3, 3] # create three-element list
print a
print b
print c

e1 = c[0]            # unpack elements; approach 1
e2 = c[1]
e3 = c[2]
print e1
print e2
print e3

ee1, ee2, ee3 = c    # unpack list; approach 2
print ee1, ee2, ee3  # print variables

%run 3-4.py
[]
[0, 21]
[1.3, 0.0013, 3]
1.3
0.0013
3
1.3 0.0013 3
```

A sequence of lines that have the same indentation form blocks in Python. In other word, a few spaces indentation indicate a 'block' of code. The indented block is ended by the first line that follows it with no indentation. Code blocks are necessary for creation of Python functions, loops and to control flow. Here are a few examples of creation and use of functions:

```
# File 3-1.py [View]
# Example 3. Calculations with Python
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

def eq1(x):
    return x**2

def eq2(x,y):
    z = x+y
    return z

def eq3(x):
    return x**p
```

```
x = 4
result = eq1(x)
print result      # 16

x=1
y=9
print eq2(x,y) # 10

p = 3
x = 2
print eq3(x) # 8

%run 3-1.py
16
10
8

# File 3-2.py [View]
# Example 3. Calculations with Python
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

def eq1(x):
    print x[0]
    print x[1]
    return x[0]+x[1]

a0 = 3
b0 = 7
result = eq1([a0,b0])
print result

c0 = [a0, b0]
result = eq1(c0)
print result

%run 3-2.py
3
7
10
3
7
10

# File 3-3.py [View]
# Example 3. Calculations with Python
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

def eq1(x):
    a = x[0]
    b = x[1]
    r1 = a+b
    r2 = a-b
    return [r1, r2]

a0 = 3
b0 = 7
result = eq1([a0,b0])
```

```
print result
print result[0]
print result[1]

%run 3-3.py
[10, -4]
10
-4
```

In this course `Python` code is organized according to following structure:

1. Import modules and functions
2. Define helper functions (if any)
3. Define main model function:
    (a) Algebraic equation(s)
    (b) Ordinary differential equation(s)
4. Define parameters and initial conditions (if necessary)
5. Run calculations: call `fsolve()` or `odeint()`
6. Post-processing
    (a) Additional calculations (if any)
    (b) Printing results
    (c) Plotting graphs
7. Results analysis and verification (by User)

Example 4. Arrays in Python

● ○ ○

So far, you use in calculations scalar values, but the real power of numerical calculations is in using of sets of data packed in arrays. Numpy arrays are more advanced data type than Python lists.

To start, run:

```
# File 4-1.py [View]
# Example 4. Arrays in Python

import numpy as np

# Lets create a small one-dimensional array - vector:
v1 = np.array( [2,4,6] )
# Please remember about module_alias-dot-function_name notation

print(v1)                       # 3 elements: 2,4,6
print( v1[0] )                  # first element (index = 0)
print( v1[1] )                  # first element (index = 1)
print( v1[2] )                  # first element (index = 2)

print('sum of elements')
print( v1[0] + v1[1] + v1[2] )  # sum of the vector elements

# You can do also some arithmetic operations (element-by-element mode):
print( v1+2 )
print( v1**2 )

v2 = 3*v1
print( v2 )
```

you should see :

```
%run 4-1.py
[2 4 6]
2
4
6
sum of elements
12
[4 6 8]
[ 4 16 36]
[ 6 12 18]
```

So, `array()` function could be used for creation of arrays. The function requires list of elements as an argument.

More examples:

```
# File 4-2.py [View]
# Example 4. Arrays in Python

import numpy as np

v1 = np.array( [2,4,6] )

# You can apply math functions from NumPy module,
# simply pass name of the array (vector) to a function:

print( np.log10(v1) ) # log10 of each v1 element
```

```
# Please note v1 vector is unchanged
print(v1)

# To save the results you can create new vector:
v2 = np.log10(v1)
print(v2)
```

you should see:

```
%run 4-2.py
[ 0.30103     0.60205999  0.77815125]
[2 4 6]
[ 0.30103     0.60205999  0.77815125]
```

Sometimes is useful to create a series of values, to do so, you can use `linspace()` or `arange()` functions:

```
# File 4-3.py [View]
# Example 4. Arrays in Python

import numpy as np

v1 =  np.linspace(0, 10, 5)
# start = 0
# end = 10
# number of equally spaced values (N) = 5
print(v1)

v2 =  np.linspace(-1, 1, 5)
# start = 0 end = 10 N = 5
print(v2)                       # -1 -0.5 0 0.5 1

v3 = np.linspace(-1,1)          # default N=50
print(v3)                       # 50 equally spaced values are generated

v4 = np.arange(0, 10, 5)        # 0 5
# start = 0 step = 5 end 10 (not included)
print(v4)

v5 = np.arange(0, 1, 0.25)      # 0 0.25 0.5 0.75
# start = 0 step = 0.25 end 1 (not included)
print(v5)
```

you should see values reside in five created vectors `v1, v2, v3, v4` and `v5` (inside square brackets `[]`):

```
%run 4-3.py
[  0.    2.5   5.    7.5  10. ]
[-1.  -0.5  0.   0.5  1. ]
[-1.         -0.95918367 -0.91836735 -0.87755102 -0.83673469 -0.79591837
 -0.75510204 -0.71428571 -0.67346939 -0.63265306 -0.59183673 -0.55102041
 -0.51020408 -0.46938776 -0.42857143 -0.3877551  -0.34693878 -0.30612245
 -0.26530612 -0.2244898  -0.18367347 -0.14285714 -0.10204082 -0.06122449
 -0.02040816  0.02040816  0.06122449  0.10204082  0.14285714  0.18367347
  0.2244898   0.26530612  0.30612245  0.34693878  0.3877551   0.42857143
  0.46938776  0.51020408  0.55102041  0.59183673  0.63265306  0.67346939
  0.71428571  0.75510204  0.79591837  0.83673469  0.87755102  0.91836735
  0.95918367  1.        ]
[0 5]
[ 0.    0.25  0.5   0.75]
```

Arrays are a bit complicated in use than vectors:

```
# File 4-4.py [View]
# Example 4. Arrays in Python

import numpy as np

# Creation of array (matrix) 2x3 (2 columns, 3rows):
m1 = np.array([[10,20],[30,40],[50,60]])
print(m1)

# Addressing one element [row_index][column_index]:
print( m1[0][0] )      # 10 - a first element - first column & row
print( m1[1][1] )      # 40 - a second column & row

# Addressing selected column (colon means all elements):
print('columns')
print( m1[:,0] )      # a first column (vector!)
print( m1[:,1] )      # a second column (vector)

# and selected rows:
print('rows')
print( m1[0,:] )      # a first row (vector)
print( m1[1,:] )      # a second row (vector)
print( m1[-1,:] )     # the last one (3rd) (also vector)
```

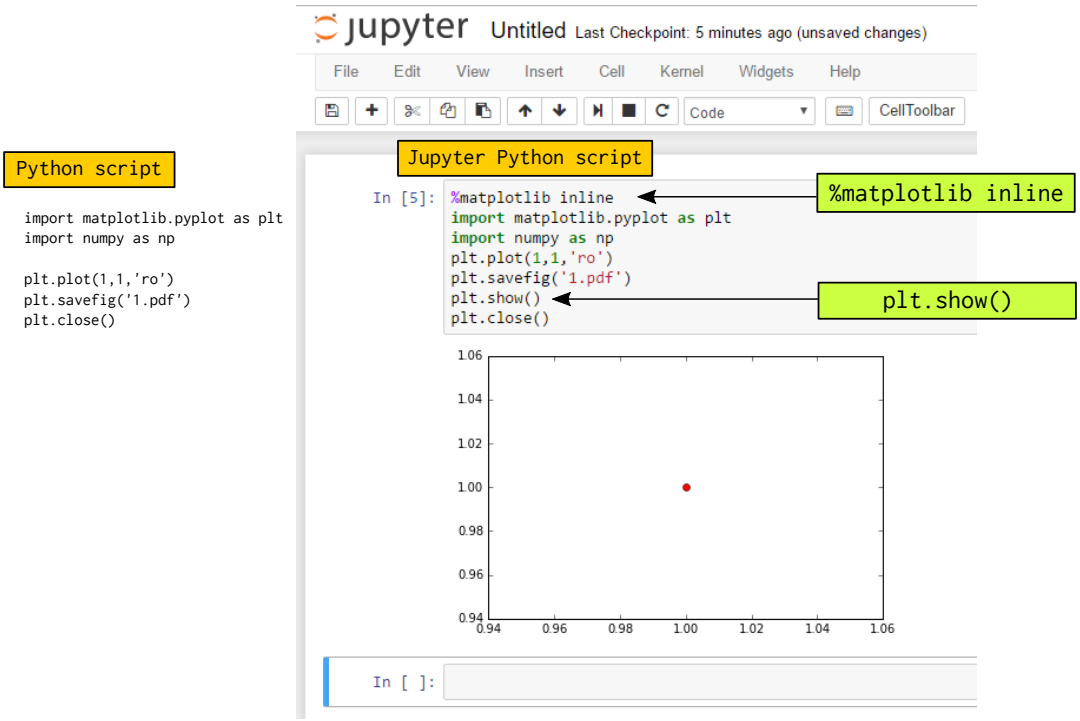
The script produces following numbers:

```
%run 4-4.py
[[10 20]
 [30 40]
 [50 60]]
10
40
columns
[10 30 50]
[20 40 60]
rows
[10 20]
[30 40]
[50 60]
```

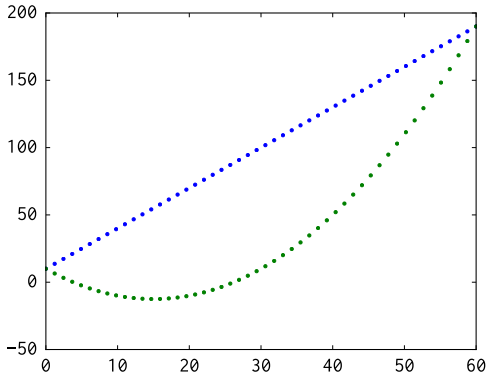
Example 5. Plots

● ○ ○

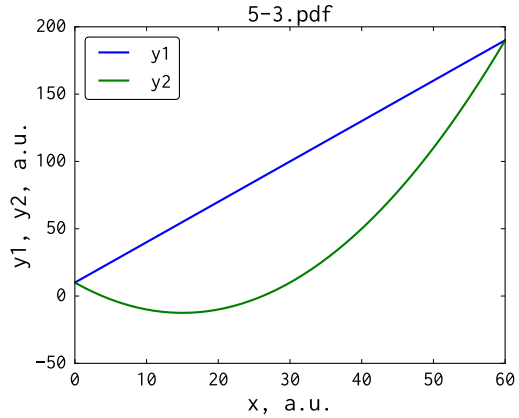
```
# File 5-1.py [View]
# Example 5. Plots
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt   # matplotlib pyplot submodule
import numpy as np                # numpy

plt.plot(1,1,'ro')                # draw a red bullet at (1,1)
plt.savefig('5-1.pdf')              # save the plot to file '5-1.pdf'
plt.close()                       # finish plotting
```




```
# File 5-2.py [View]
# Example 5. Plots
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,60)
y1 = 3*x+10
y2 = 0.1*x**2 - 3*x + 10

plt.plot(x,y1,'b.')
plt.plot(x,y2,'g.')
plt.savefig('5-2.pdf')
plt.close()
```

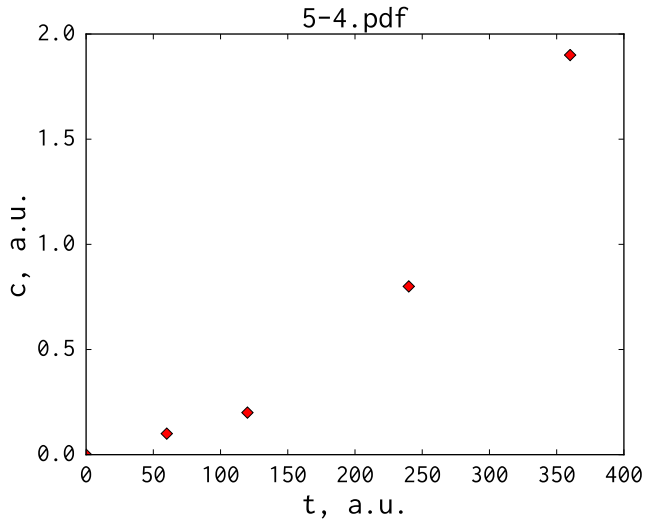



```
# File 5-3.py [View]
# Example 5. Plots
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0,60)
y1 = 3*x+10
y2 = 0.1*x**2 - 3*x + 10

plt.plot(x,y1,'b-', label='y1')
plt.plot(x,y2,'g-', label='y2')
plt.title('5-3.pdf')
plt.xlabel('x, a.u.')
plt.ylabel('y1, y2, a.u.')
plt.legend(loc='upper left')      # loc = location
plt.savefig('5-3.pdf')
plt.close()
```



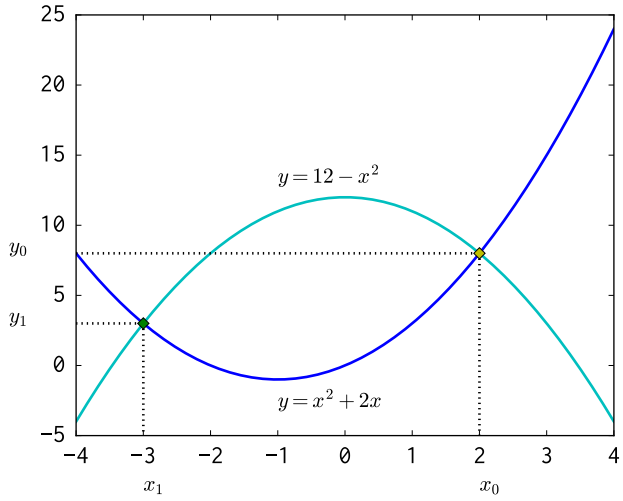

```
# File 5-4.py [View]
# Example 5. Plots
```

```
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np

# the data
t = np.array([0, 60, 120, 240, 360])
c = np.array([0, 0.1, 0.2, 0.8, 1.9])
# ploting
plt.plot(t, c, 'rD')
plt.title('5-4.pdf')
plt.xlabel('t, a.u.')
plt.ylabel('c, a.u.')
plt.savefig('5-4.pdf')
plt.close()
```




```
# File 5-5.py [View]
# Example 5. Plots
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

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

# calculations
def system( unknowns ):
    x = unknowns[0]       # unpack x
    y = unknowns[1]       # unpack y
    eq1 = y - x**2 - 2*x  # calculate eq1 for given x,y
    eq2 = y - 12 + x**2   # calculate eq2 for given x,y
    return [eq1, eq2]     # return the results (0?)

guess = [1, 1]
x0, y0 = fsolve(system, guess)
guess = [-10, -10]
```

```
x1, y1 = fsolve(system, guess)
# and reporting ...
x = np.linspace(-4, 4)            # x = -4 ... 4
plt.plot(x, x**2 + 2*x, 'b-')     # plot x^2+2x
plt.plot(x, 12-x**2, 'c-')        # plot 12-x^2

plt.plot((-4,x0), (y0,y0),'k:')   # plot line from -4,y0 to x0,y0
plt.plot((x0,x0), (y0,-5),'k:')   #

plt.plot((-4,x1), (y1,y1),'k:')
plt.plot((x1,x1), (y1,-5),'k:')

plt.plot(x0,y0,'yD')              # plot a marker at x0,y0
plt.plot(x1,y1,'gD')

plt.text(-1,-3,'$y=x^2+2x$')      # add some text -> LaTeX syntax (!)
plt.text(-1,13,'$y=12-x^2$')

plt.text(-5,y0,'$y_0$')
plt.text(-5,y1,'$y_1$')
plt.text(x0,-9,'$x_0$')
plt.text(x1,-9,'$x_1$')


plt.savefig('5-5.pdf')
plt.close()
```



Example 6. Solving algebraic equation(s)

● ○ ○

Lets solve a simple nonlinear equation:

$$\sqrt{x} = 2.5 \tag{1}$$

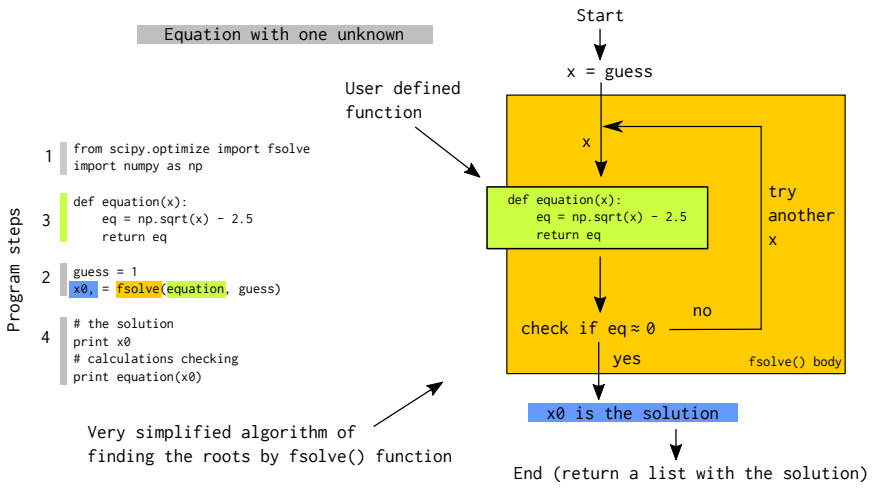
To find the roots of a function `fsolve` function can be used [1]. First, we have to re-arrange eq. 1 to get 0 on right-hand side:

$$\sqrt{x} - 2.5 = 0 \tag{2}$$

Now we are ready for numerical calculations:

```
# File 6-1.py [View]
# Example 6. Solving algebraic equation(s)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
import numpy as np

def equation(x):
    eq = np.sqrt(x) - 2.5
    return eq

guess = 1
x0, = fsolve(equation, guess)
print x0              # the solution
print equation(x0)    # the result checking, 0?

%run 6-1.py
6.25
0.0
```

Here is how the program works:



Lets solve (nonlinear) equation with one unknown:

$$x^3 - 2x^2 = 9x - 10 \tag{3}$$

Try guess values ranged from -10 to 10. How many roots you can find? Make plot of function $f(x) = x^3 - 2x^2 - 9x + 10$.

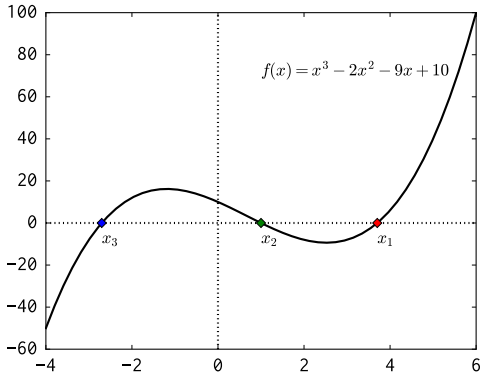
Calculations

Calculations procedure is as above. Here is the script:

```
# File 6-2.py [View]
# Example 6. Solving algebraic equation(s)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
import numpy as np

def equation(x):
    eq =  x**3 - 2*x**2 - 9*x + 10
    return eq

guess = 6
x1, = fsolve(equation, guess)
print "x1={}".format(x1)              # the solution no.1
guess = 0
x2, = fsolve(equation, guess)
print "x2={}".format(x2)              # the solution no.2
guess = -10
x3, = fsolve(equation, guess)
print "x3={}".format(x3)              # the solution no.3
```

Results

```
%run 6-2.py
x1=3.70156211872
x2=1.0
x3=-2.70156211872
```



Lets study a system of nonlinear equations. For example solve the following system numerically, and for comparison graphically:

$$\begin{aligned} y &= x^2 + 2x \\ y &= 12 - x^2 \end{aligned} \tag{4}$$

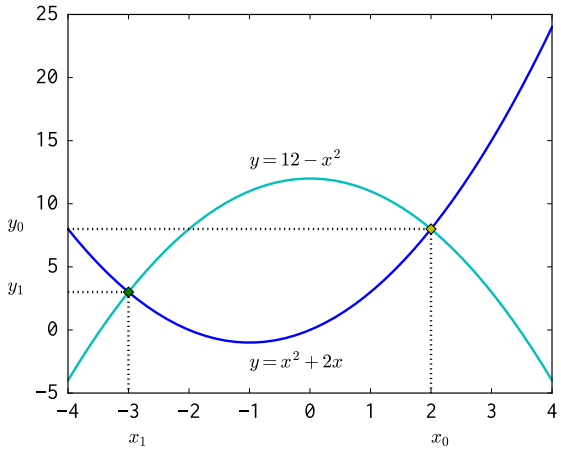
Calculations

First, we have to re-arrange eq. 4 to get 0 on right-hand side and then the equations can be inserted into the script:

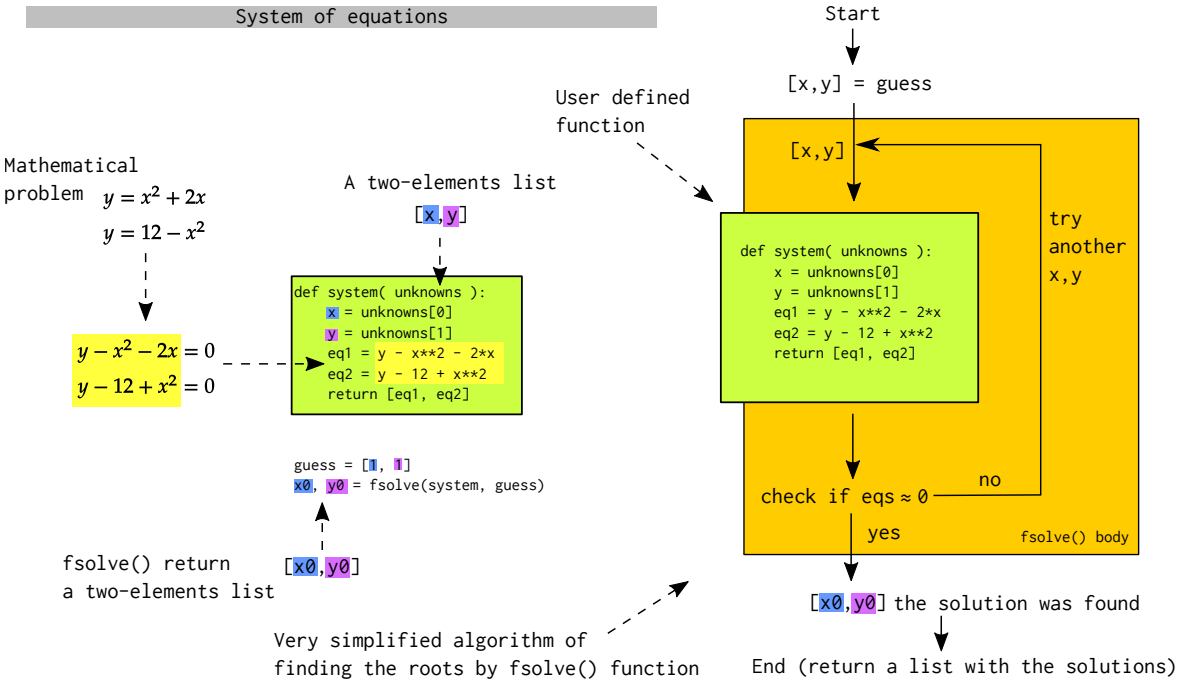
```
# File 6-4.py [View]
# Example 6. Solving algebraic equation(s)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
import numpy as np

def system( unknowns ):
    x = unknowns[0]       # unpack x
    y = unknowns[1]       # unpack y
    eq1 = y - x**2 - 2*x  # calculate eq1 for given x,y
    eq2 = y - 12 + x**2   # calculate eq2 for given x,y
    return [eq1, eq2]     # return the results (0?)

guess = [1, 1]                          # two unknowns = two guess; in order: x, y
x0, y0 = fsolve(system, guess)
print "x0, y0 = {} {}".format(x0, y0)
guess = [-10, -10]                      # different pair of guess values
x1, y1 = fsolve(system, guess)
print "x1, y1 = {} {}".format(x1, y1)
```

Results

```
%run 6-4.py
x0, y0 = 2.0 8.0
x1, y1 = -3.0 3.0
```



Note we have two unknowns, so we need pass to `fsolve` two guess values (as a two-elements list). Additionally, `fsolve` must know a name of a user-defined function which is able to calculate and return values of each equation for a given pair `x,y`. Here is how the program works:



For more details how `fsolve()` works see manual [1].

[1] *SciPy Documentation, fsolve() API*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.fsolve.html.

Example 7. Solving ODE(s)

● ○ ○

Consider an initial value problem (ODE with initial conditions):

$$\frac{dy}{dt} = -ky, \quad t_0 = 0, y(t_0) = 2 \tag{5}$$

Find $y(t = 18)$.

Calculations

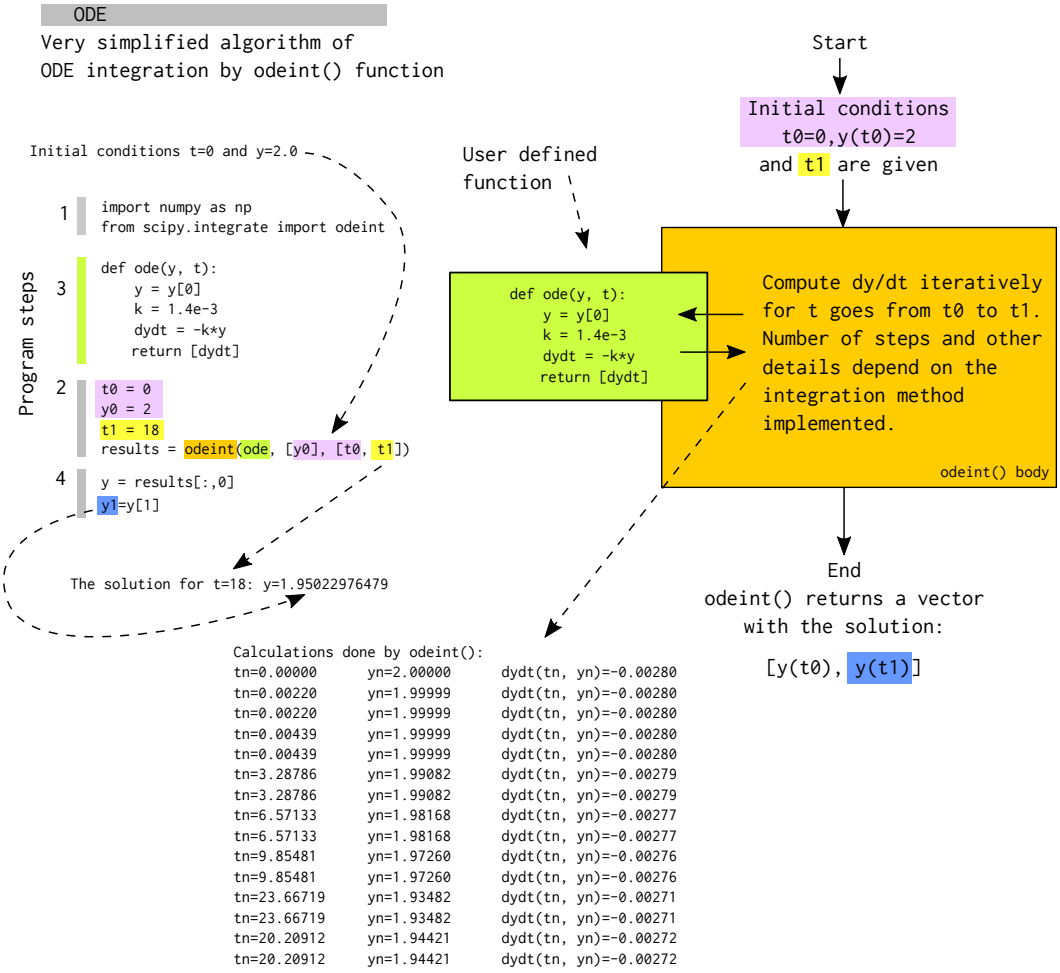
```
# File 7-1.py [View]
# Example 7. Solving ODE(s)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import numpy as np
from scipy.integrate import odeint

def ode(y, t):
    y = y[0]            # unpack variable, it is more convenient write y instead y[0]
    k = 1.4e-3          # a parameter (just for example)
    dydt = -k*y         # calculate derivative (dy/dt)

    # just to show calculations progress
    print "tn={:.5f}\tyn={:.5f}\tdydt(tn, yn)={:.5f}".format(t,y,dydt)

    return [dydt]       # return the derivative value


t0 = 0                  # initial time
y0 = 2                  # initial conditions: y(t0) = 2
t1 = 18                 # we are interested in the value of y at t1 = 18

print "Calculations done by odeint():"
results = odeint(ode, [y0], [t0, t1])  # integrate the ODE
y = results[:,0]                       # results is two-elements vector
print "Raw results: ", y
print "Initial conditions t=0 and y={}".format(y[0]) # element '0'
print "The solution for t=18: y={}".format(y[1])   # element '1'
```

Results

```
%run 7-1.py
Calculations done by odeint():
tn=0.00000      yn=2.00000      dydt(tn, yn)=-0.00280
tn=0.00220      yn=1.99999      dydt(tn, yn)=-0.00280
tn=0.00220      yn=1.99999      dydt(tn, yn)=-0.00280
tn=0.00439      yn=1.99999      dydt(tn, yn)=-0.00280
tn=0.00439      yn=1.99999      dydt(tn, yn)=-0.00280
tn=3.28786      yn=1.99082      dydt(tn, yn)=-0.00279
tn=3.28786      yn=1.99082      dydt(tn, yn)=-0.00279
tn=6.57133      yn=1.98168      dydt(tn, yn)=-0.00277
tn=6.57133      yn=1.98168      dydt(tn, yn)=-0.00277
tn=9.85481      yn=1.97260      dydt(tn, yn)=-0.00276
tn=9.85481      yn=1.97260      dydt(tn, yn)=-0.00276
tn=23.66719     yn=1.93482      dydt(tn, yn)=-0.00271
tn=23.66719     yn=1.93482      dydt(tn, yn)=-0.00271
```

```
tn=20.20912     yn=1.94421      dydt(tn, yn)=-0.00272
tn=20.20912     yn=1.94421      dydt(tn, yn)=-0.00272
Raw results:  [ 2.          1.95022976]
Initial conditions t=0 and y=2.0
The solution for t=18: y=1.95022976479
```

Here is how the program works:



Solve IVP (eq.5) for t =200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000. Print value of $y(t = 600)$.

Calculations

```
# File 7-2.py [View]
# Example 7. Solving ODE(s)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import numpy as np
from scipy.integrate import odeint

def ode(y, t):
    y = y[0]              # unpack variable, it is more convenient write y instead y[0]
    k = 1.4e-3            # a parameter (just for example)
    dydt = -k*y           # calculate derivative (dy/dt)
    return [dydt]         # return the derivative
```

```
y0 = 2                    # initial conditions y(t0) = 2

t = np.linspace(0,2000,11)            # t goes from 0 to 2000 in steps of 200: (2000-0)/(11-1)
results = odeint(ode, [y0], t)        # do calculations
y = results[:,0]
print t                               # time ticks
print y                               # and instantaneous y

print t[3]
print y[3]
```

Results

```
%run 7-2.py
[    0.   200.   400.   600.   800.  1000.  1200.  1400.  1600.  1800.
  2000.]
[ 2.          1.51156746  1.14241815  0.86342108  0.65255961  0.49319394
  0.37274796  0.28171684  0.212917    0.1609192   0.12162011]
600.0
0.863421078399
```

Here is how the script works:

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

def ode(y, t):
    y = y[0]
    k = 1.4e-3
    dydt = -k*y
    return [dydt]

y0 = 2  <- - - -  Initial conditions t[0], y0

t = np.linspace(0,2000,11)
results = odeint(ode, [y0], t)
y = results[:,0]
print t
print y

print t[3]
print y[3]
```

Index	Vector t	Vector y
0	0.0	2.0
1	200.0	1.51156746222
2	400.0	1.14241815479
3	600.0	0.863421078399
4	800.0	0.652559609924
5	1000.0	0.493193939903
6	1200.0	0.3727479595
7	1400.0	0.281716839817
8	1600.0	0.212917000706
9	1800.0	0.160919202512
10	2000.0	0.12162011277

Solutions: y(t)

Integrate system of ODEs for t =200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800 and 2000:

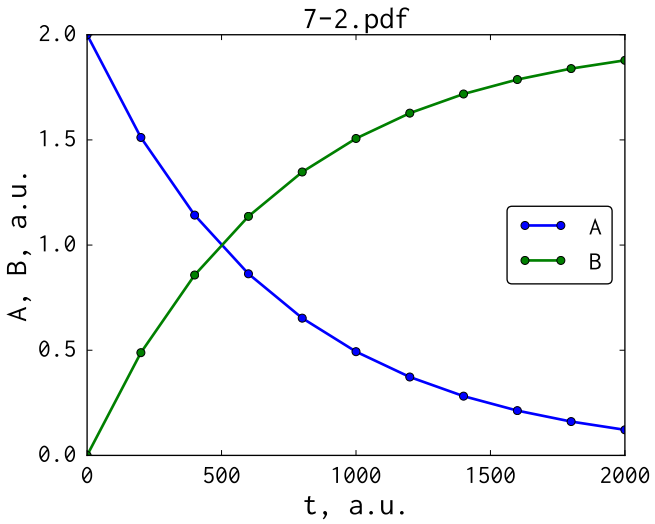
$$\frac{dA}{dt} = -kA$$
$$\frac{dB}{dt} = kA \tag{6}$$

assuming that $k = 1.4 \times 10^{-3}$ if at $t_0 = 0$ $A_0 = 2$ and $B_0 = 0$. Print the results as a raw output and formated table as well as make a plot.

```
# File 7-3.py [View]
# Example 7. Solving ODE(s)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
```

```
def model(y, t):
    A = y[0]             # unpack variables
    B = y[1]
    dAdt = -k*A          # calculate derivatives
    dBdt = k*A
    return [dAdt, dBdt]  # return derivatives

# Parameters
# They can be inside a ODEs function or outside
k = 1.4e-3
A0 = 2
B0 = 0

# Time span
t = np.linspace(0, 2000, 11) # 0 - 2000 step 200
# Initial conditions
ic = [A0, B0]

results = odeint(model, ic, t)
A = results[:,0]
B = results[:,1]

# print the results
print t
print A
print B
print
# or in a more clear form:
print "idx\tt\tA\tB"
for i in range(t.size):
    print "{}\t{}\t\t{:.3f}\t\t{:.3f}".format(i,t[i], A[i], B[i])

# and plotting the results
plt.plot(t, A, 'bo-', label = 'A')
plt.plot(t, B, 'go-', label = 'B')
plt.ylabel('A, B, a.u.')
plt.xlabel('t, a.u.')
plt.legend(loc='best')
plt.title('7-2.pdf')
plt.savefig('7-2.pdf')
plt.close()

%run 7-3.py
[    0.   200.   400.   600.   800.  1000.  1200.  1400.  1600.  1800.
  2000.]
[ 2.          1.5115675   1.14241815  0.86342105  0.65255958  0.49319392
  0.37274793  0.28171682  0.21291699  0.1609192   0.12162012]
[ 0.          0.4884325   0.85758185  1.13657895  1.34744042  1.50680608
  1.62725207  1.71828318  1.78708301  1.8390808   1.87837988]

idx     t               A               B
0       0.0             2.000           0.000
1       200.0           1.512           0.488
2       400.0           1.142           0.858
3       600.0           0.863           1.137
4       800.0           0.653           1.347
5       1000.0          0.493           1.507
6       1200.0          0.373           1.627
```

```
7       1400.0          0.282           1.718
8       1600.0          0.213           1.787
9       1800.0          0.161           1.839
10      2000.0          0.122           1.878
```

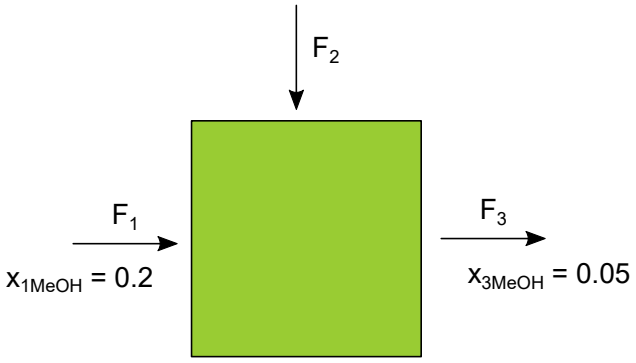


3 Mass Balance

Example 8. Mass balance (1)

● ○ ○

A 20% w/w solution of MeOH (F_1) is mixed with pure water (F_2) to get 5% w/w MeOH (F_3). Calculate flow rate of water (F_2) and the diluted solution (F_3), knowing that F_1=1234 kg h^{-1} and the process is under stady-state.

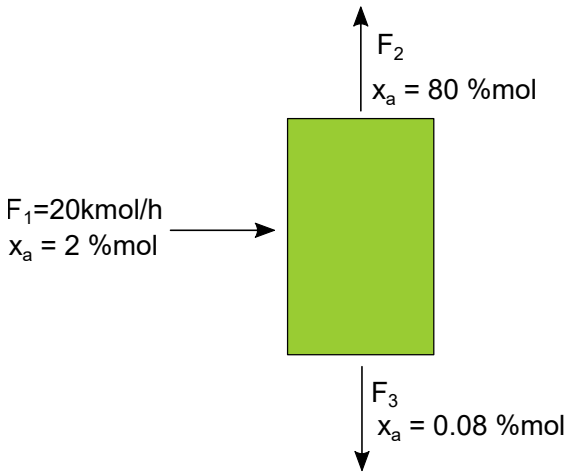



```
# File 8-1.py [View]
# Example 8. Mass balance (1)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

F1 = 1234 #kg/h
x1MeOH = 0.2
x2MeOH = 0
x2H2O = 1
x3MeOH = 0.05

def model(X):
    x1H2O, F2, F3, x3H2O = X
    eq1 = F1 + F2 - F3                              # material balance of the system
    eq2 = x1MeOH * F1 + x2MeOH * F2 - x3MeOH * F3   # MeOH balance
    eq3 = x1MeOH + x1H2O -1                         # mass fraction constraint
    eq4 = x3MeOH + x3H2O -1                         # mass fraction constraint
    return [eq1, eq2, eq3, eq4]

guess = [0.1, 100, 100, 0.1]
x1H2O, F2, F3, x3H2O = fsolve(model, guess)
print "x1H2O = {:.2f}".format(x1H2O)
print "F2 = {:.2f} kg/h".format(F2)
print "F3 = {:.2f} kg/h, x3H2O = {:.2f}".format(F3, x3H2O)

%run 8-1.py
x1H2O = 0.80
F2 = 3702.00 kg/h
F3 = 4936.00 kg/h, x3H2O = 0.95
```

Example 9. Mass balance (2)

● ○ ○

Consider a continous steady-state process of acetone-water distilation. Calculate flows of a distilate (F_2) and a bottom fraction (F_3). Molar fractions of acetone for each stream are shown on the scheme.




```
# File 9-1.py [View]
# Example 9. Mass balance (2)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

F1 = 20 #kmol/h

# acetone mass fractions
x1a = 2.0/100
x2a = 80.0/100
x3a = 0.08/100

def model(X):
    F2, F3 = X
    eq1 = F1 - F2 - F3
    eq2 = x1a * F1 - x2a * F2 - x3a * F3
    return [eq1, eq2]

guess = [1, 1] # F2, F3
F2, F3 = fsolve(model, guess)
print "Distilate F2 = {:.2f} kmol/h".format(F2)
print "Bottom fraction F3 = {:.2f} kmol/h.".format(F3)
```

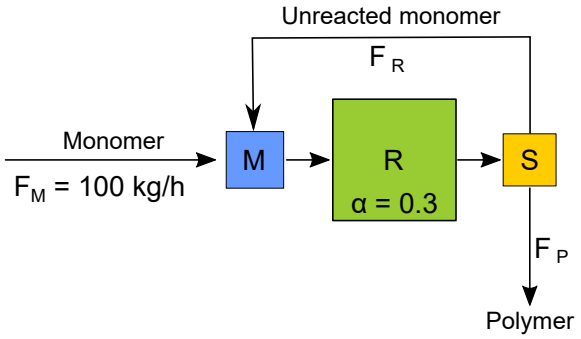
The results:

```
%run 9-1.py
Distilate F2 = 0.48 kmol/h
Bottom fraction F3 = 19.52 kmol/h.
```

Example 10. Mass balance with recycle (1)

● ○ ○

Consider a continuous steady-state process of polymerization with recycle of an unreacted monomer (scheme). Conversion α of the monomer into the polymer in the reactor (R) is 30% per pass. Calculate flow of the recycled stream assuming ideal separation of the polymer in the separator (S).




```
# File 10-1.py [View]
# Example 10. Mass balance with recycle (1)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

FM = 100 #kg/h
alfa = 0.3

def model(X):
    FP, FR = X                    # equivalent syntax: FP=X[0] FR=X[1]
    eq1 = FM - FP                 # total mass balance
    eq2 = alfa*(1*FM + 1*FR) - FP  # monomer balance
    return [eq1, eq2]

guess = [100, 100] # FP FR
FP, FR = fsolve(model, guess)
print 'FP={:.2f} kg/h'.format(FP)
print 'FR={:.2f} kg/h'.format(FR)
```

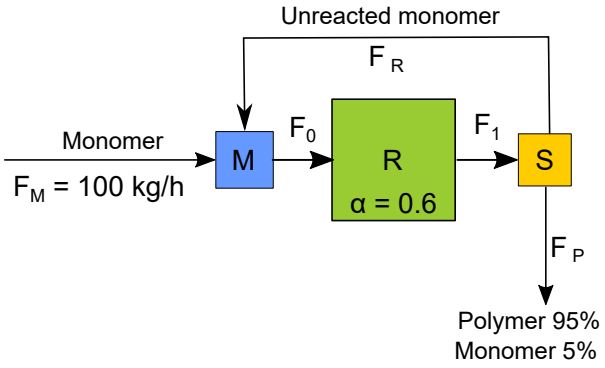
Results:

```
%run 10-1.py
FP=100.00 kg/h
FR=233.33 kg/h
```

Example 11. Mass balance with recycle (2)

● ○ ○

Consider a continuous steady-state process of polymerization with recycle of an unreacted monomer (scheme). Conversion of the monomer to polymer α in the reactor (R) is 60% per pass. Calculate flow of the recycled stream if the polymer separated in the separator (S) contains 5% of the monomer.




```
# File 11-1.py [View]
# Example 11. Mass balance with recycle (2)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

from scipy.optimize import fsolve

FM = 100 #kg/h
alpha = 0.6

def model(X):
    FP, FR = X
    eq1 = (1-alpha)*(FM + FR) - FR - 0.05*FP # monomer balance
    eq2 = alpha*(FM + FR) - 0.95*FP          # polymer balance
    return [eq1, eq2]

guess = [100,100] # FP, FR
FP, FR = fsolve(model, guess)
print 'FP={:.2f} kg/h'.format(FP)
print 'FR={:.2f} kg/h'.format(FR)
```

Results:

```
%run 11-1.py
FP=100.00 kg/h
FR=58.33 kg/h
```

Note that $F_M = F_P$.

Example 12. Recycle - sequential method

●●○

This example illustrates sequential method for mass balance with recycle stream [1][2]. Compare algebraic approach (examples 10 and 11) with this iterative technique.

Calculations

The algorithm require solving mass balance for each unit in a loop. We start calculation with the mixer (assuming in first iteration recycle stream rate of zero), next we continue computations for the reactor and finally for the separator. The procedure is repeated until calculated stream flows do not change with iteration. The number of iterations is chosen by through trial and error.

Results

Here is solution of example 10 by sequential method:

```
# File 12-1.py [View]
# Example 12. Recycle - sequential method
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

FM = 100 # kg/h
alpha = 0.3

i = 1 # i = iteration
i_max = 35

print "i\tF0\t\tF1\t\tFR"

# monomer streams
while i < i_max:
    if i == 1:
        FR = 0
    F0 = FM + FR        # mixer (monomer)
    F1 = F0 - alpha*F0  # reactor (monomer)
    FR = F1             # separator
    print "{}\t{:.3f}\t\t{:.3f}\t\t{:.3f}".format(i,F0, F1, FR)
    i = i + 1           # next iteration

%run 12-1.py
i       F0              F1              FR
1       100.000         70.000          70.000
2       170.000         119.000         119.000
3       219.000         153.300         153.300
4       253.300         177.310         177.310
5       277.310         194.117         194.117
6       294.117         205.882         205.882
7       305.882         214.117         214.117
8       314.117         219.882         219.882
9       319.882         223.917         223.917
10      323.917         226.742         226.742
11      326.742         228.720         228.720
12      328.720         230.104         230.104
13      330.104         231.073         231.073
14      331.073         231.751         231.751
15      331.751         232.226         232.226
```

```
16      332.226         232.558         232.558
17      332.558         232.791         232.791
18      332.791         232.953         232.953
19      332.953         233.067         233.067
20      333.067         233.147         233.147
21      333.147         233.203         233.203
22      333.203         233.242         233.242
23      333.242         233.269         233.269
24      333.269         233.289         233.289
25      333.289         233.302         233.302
26      333.302         233.311         233.311
27      333.311         233.318         233.318
28      333.318         233.323         233.323
29      333.323         233.326         233.326
30      333.326         233.328         233.328
31      333.328         233.330         233.330
32      333.330         233.331         233.331
33      333.331         233.332         233.332
34      333.332         233.332         233.332
```

After 34 iterations the solution is found to be $F_R = 233.33\,\mathrm{kg\,h^{-1}}$.

Here is solution of example 11 by sequential method:

```
# File 12-2.py [View]
# Example 12. Recycle - sequential method
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

FM = 100 # kg/h
alpha = 0.6
xm = 0.05

i = 1 # i = iteration
i_max = 15

print "i\tF0\t\tFR"

# streams
while i < i_max:
    if i == 1:
        FR = 0
    F0 = FM + FR            # mixer (monomer)
    F1P = alpha*F0          # reactor (polymer)
    F1M = F0 - F1P          # reactor (monomer)
    FP = F1P + xm*F1P       # output stream (FP = polymer + monomer)
    FR = F1M - FP*xm        # recycle (monomer balance)
    print "{}\t{:.3f}\t\t{:.3f}".format(i,F0, FR)
    i = i + 1               # next iteration

%run 12-2.py
i       F0              FR
1       100.000         36.850
2       136.850         50.429
3       150.429         55.433
4       155.433         57.277
5       157.277         57.957
6       157.957         58.207
7       158.207         58.299
8       158.299         58.333
```

```
9       158.333         58.346
10      158.346         58.350
11      158.350         58.352
12      158.352         58.353
13      158.353         58.353
14      158.353         58.353
```

In this case after 14 iterations the solution is found to be $F_R = 58.35\,\mathrm{kg\,h^{-1}}$.

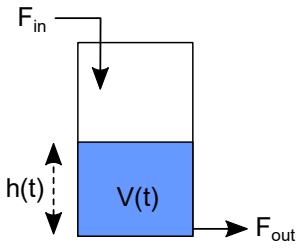
[1] B.A. Finlayson. *Introduction to Chemical Engineering Computing*. Wiley, 2014. ISBN: 9781118888377. URL: https://books.google.pl/books?id=5awEAwAAQBAJ.

[2] R. Rousseau R. Felder. *Elementary Principles of Chemical Processes*. 3rd ed. Wiley, 2005.

Example 13. Differential mass balance

● ○ ○

A tank is filled at a flow rate of $F_{in} = 10\,\mathrm{L\,min^{-1}}$. Volumetric flow rate from the tank is $F_{out} = 1\,\mathrm{L\,min^{-1}}$. Plot volume of a liquid in the tank versus time from 0 to 60 min assuming that densities (ρ) of inlet and outlet flows are the same and initially the tank is empty.



Calculations

The following equation is a mass balance that can be applied to any system without chemical reaction:

$$\text{Accumulation} = \text{In} - \text{Out} \tag{7}$$

Thus for our system we have:

$$\frac{dm}{dt} = \frac{d(\rho V)}{dt} = \rho F_{in} - \rho F_{out} \tag{8}$$

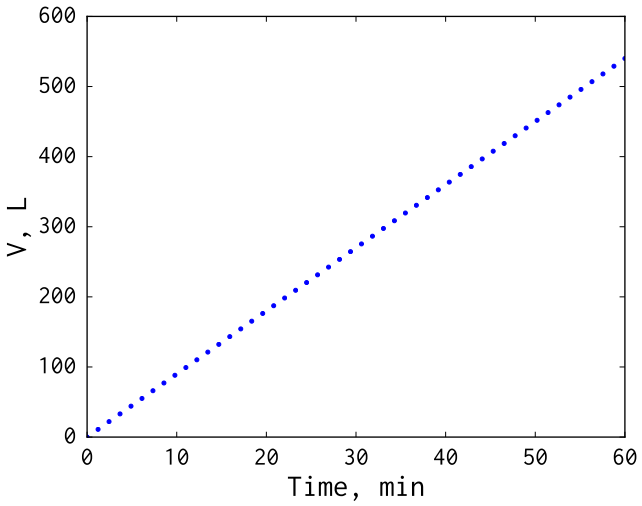
Because the density is constant:

$$\frac{dV}{dt} = F_{in} - F_{out} \tag{9}$$

Note that the system is in unsteady state (accumulation term is not equal to zero). In the script we are going to solve the ordinary differential equation 9 with the initial conditions $V(t = 0) = 0$ numerically.

```
# File 13-1.py [View]
# Example 13. Differential mass balance
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# model = mass balance = ODE (or ODEs)
def model(y, t):
    V = y[0]
    dVdt = Fin - Fout
    return [dVdt]

Fin = 10 #L/min
Fout = 1 #L/min
t = np.linspace(0,60) # 0 - 60 min

ic = [0] # initial conditions: V at t=0 -> V0=0

results = odeint(model, ic, t)
```

```
V = results[:,0]
print "Time, min"
print t
print "V(t), L"
print V # V(t)

plt.plot(t,V, 'b.')
plt.xlabel('Time, min')
plt.ylabel('V, L')
plt.savefig('13-1.pdf')
plt.close()
```



Exercises

1) Check units in eq. 8 and eq. 9. Calculate derivative $\frac{d(\rho V)}{dt}$ if ρ varies in time and if ρ is time independent.

2) Model a case when the tank contains initially 400 L of a liquid, and $F_{in} = 1\,\mathrm{L\,min^{-1}}$ and $F_{out} = 10\,\mathrm{L\,min^{-1}}$.

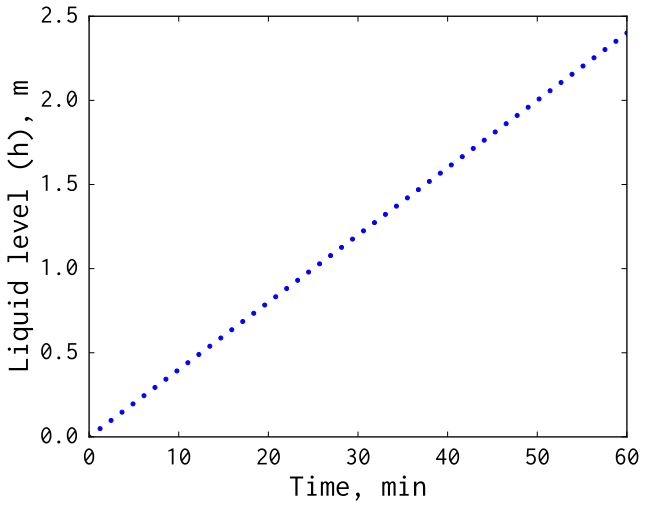
3) Develop a model for a cylindrical tank to calculate liquid level as a function of time: $h(t)$, assuming cross-section area of $5.0\,\mathrm{m^3}$, $F_{in} = 10\,\mathrm{L\,min^{-1}}$ and $F_{out} = 1\,\mathrm{L\,min^{-1}}$. The tank is initially empty.

Calculations

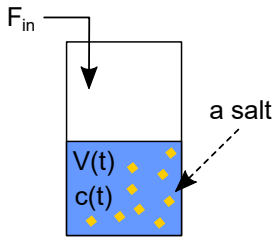
```
# File 13-2.py [View]
# Example 13. Differential mass balance
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# model = mass balance = ODE (or ODEs)
def model(y, t):
    h = y[0]
    dhdt = Fin/A - Fout/A
    return [dhdt]
```

```
Fin = 1 #m3/min
Fout = 0.8 #m3/min
A = 5.0 # m3
t = np.linspace(0,60) # 0 - 60 min

ic = [0] # initial conditions: h at t=0 -> h0=0

results = odeint(model, ic, t)
h = results[:,0]
print "Time, min"
print t
print "h(t), m"
print h # h(t)

plt.plot(t,h, 'b.')
plt.xlabel('Time, min')
plt.ylabel('Liquid level (h), m')
plt.savefig('13-2.pdf')
plt.close()
```

Results



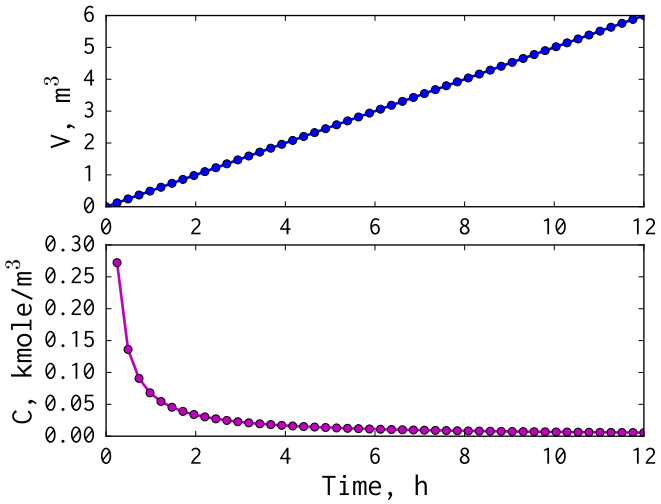
4) Consider a dissolution-dilution process in a tank in which initially there is a salt. Mass of the salt is 5 kg and molar weight of the substance is 150 kg kmole^{-1}. Plot the solution volume and molar concentration of the salt versus filling time $0 < t < 12$ h if inlet flow rate of the solvent is $F_{in} = 0.5\,\mathrm{m^3\,h^{-1}}$.



Calculations

```
# File 13-3.py [View]
# Example 13. Differential mass balance
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# model = mass balance = ODE
def model(y, t):
    V = y[0]
    dVdt = Fin
    return [dVdt]

m = 5.0                 # kg
M = 150.0               # kg/kmol
n = m/M                 # kmol
Fin = 0.5               # m3/h
t = np.linspace(0, 12)  # 0 - 12h

ic = [0]                # initial conditions: V at t=0 -> V0=0
results = odeint(model, ic, t)
V = results[:,0]        # instant V
C = n/V                 # instant C
print "Time, h"
print t
print "V(t), m3"
print V
print "C(t), kmol/m3"
print C

plt.subplot(211)
plt.plot(t,V, 'bo-')
plt.xlabel('Time, h')
plt.ylabel('V, m$^3$')
plt.subplot(212)
plt.plot(t,C, 'mo-')
plt.xlabel('Time, h')
plt.ylabel('C, kmole/m$^3$')
plt.savefig('13-3.pdf')
plt.close()
```

Results

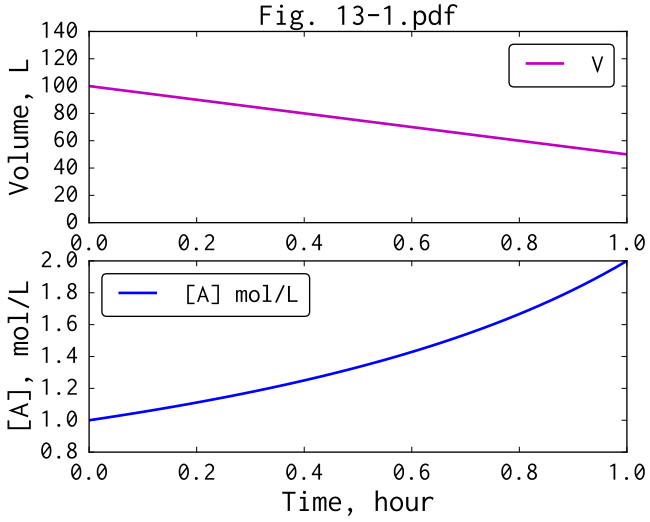


Exercises

An evaporator is initially filled by 100 L of a solution of substance A at a concentration of 1 mol L^{-1}. Assuming a solvent evaporation rate of 50 L h^{-1} plot molar concentration of A versus evaporation time from 0 to 1 h.

Results

Your plot should look like this:



4 Chemical Equilibrium

Example 14. Esterification (1)

● ○ ○

Calculate concentrations of reagents at equilibrium in a system containing initially ethanol 16 mol L^{-1} and acetic acid 3 mol L^{-1} if equilibrium constant is 4.9.

Calculations

```
# File 14-1.py [View]
# Example 14. Esterification (1)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
# a - alcohol, e - ester, w - water
def model(X):
    a, acid, e, w = X
    K = 4.9                         # equilibrium constant
    c_a = 16.0                      # mol/L EtOH
    c_acid = 3.0                    # mol/L, CH3COOH
    eq1 = (e*w)/(a*acid)-K          # equilibrium constant
    eq2 = a + e - c_a               # the alcohol balance
    eq3 = acid + e - c_acid         # the acid balance
    eq4 = e - w                     # number of ester bonds is equivalent to
                                    # amount of water produced
    return [eq1, eq2, eq3, eq4]

a = 1.0
acid = 1.0
e = 2.0
w = 2.0
guess = [a, acid, e, w]
a, acid, e, w = fsolve(model, guess)
print "Equilibrium concentrations:"
print "[EtOH]={:.2f} mol/L".format(a)
print "[CH3COOH]={:.2f} mol/L".format(acid)
print "[CH3COOEt]={:.2f} mol/L".format(e)
print "[H2O]={:.2f} mol/L".format(w)
```

Results

```
%run 14-1.py
Equilibrium concentrations:
[EtOH]=13.13 mol/L
[CH3COOH]=0.13 mol/L
[CH3COOEt]=2.87 mol/L
[H2O]=2.87 mol/L
```

Exercises

Experiment with varying ratios of the alcohol to the acid. Try to modify the program to model system containing initially the alcohol, the acid and little amounts of water.

Example 15. Esterification (2)

●●○

Consider esterification process of mixture of ethanol and 1-propanol by acetic acid:

$$\begin{aligned} EtOH + CH_3COOH &\overset{K_1}{\rightleftharpoons} CH_3COOEt + H_2O \\ PrOH + CH_3COOH &\overset{K_2}{\rightleftharpoons} CH_3COOPr + H_2O \end{aligned} \tag{10}$$

Calculate equilibrium concentrations of reagents if $K_1 = 4.9$ and $K_2 = 4.2$ and initial concentrations of substrates are: $c_{CH_3COOH} = 3\,\mathrm{mol\,L^{-1}}$, $c_{EtOH} = 7\,\mathrm{mol\,L^{-1}}$ and $c_{PrOH} = 6\,\mathrm{mol\,L^{-1}}$.

Calculations

```
# File 15-1.py [View]
# Example 15. Esterification (2)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

def model(X):
    ch3cooh, etoh, proh, ch3cooet, ch3coopr, water = X

    # equilibrium constants
    K1 = 4.9          # ethyl ester
    K2 = 4.2          # propyl ester

    # initial concentrations
    c_ch3cooh = 3
    c_proh = 8.0
    c_etoh = 7.0

    # model
    eq1 = (ch3cooet * water)/(etoh * ch3cooh) - K1    # equilibrium constant
    eq2 = (ch3coopr * water)/(proh * ch3cooh) - K2    # equilibrium constant
    eq3 = etoh + ch3cooet - c_etoh                     # etoh balance
    eq4 = proh + ch3coopr - c_proh                     # proh balance
    eq5 = ch3cooet + ch3coopr + ch3cooh - c_ch3cooh    # ch3cooh balance
    eq6 = ch3cooet + ch3coopr - water                  # esters bonds balance
    return [eq1, eq2, eq3, eq4, eq5, eq6]


# ch3cooh, etoh, proh, ch3cooet, ch3coopr, water
guess = [0.1, 1, 1, 1, 1, 1]  # try different guess values
results = fsolve(model, guess)
ch3cooh, etoh, proh, ch3cooet, ch3coopr, water = results

# report
print "Equilibrium concentrations:"
print "[CH3COOH] = {:.3f} mol/L".format(ch3cooh)
print "[EtOH] = {:.3f} mol/L".format(etoh)
print "[PrOH] = {:.3f} mol/L".format(proh)
print "[CH3COOEt] = {:.3f} mol/L".format(ch3cooet)
print "[CH3COOPr] = {:.3f} mol/L".format(ch3coopr)
print "[H2O] = {:.3f} mol/L".format(water)
```

Results

```
%run 15-1.py
Equilibrium concentrations:
[CH3COOH] = 0.148 mol/L
[EtOH] = 5.580 mol/L
[PrOH] = 6.568 mol/L
[CH3COOEt] = 1.420 mol/L
[CH3COOPr] = 1.432 mol/L
[H2O] = 2.852 mol/L
```

Exercises

Try to varying initial substrates concentrations to obtain esters contaminated with minimal amount of the acid. Verify if your parameters have a chemical sense.

Example 16. Equilibrium composition - gas phase reactions

●●○

Chloromethane reacts with steam at 600K:

$$\begin{aligned} CH_3Cl + H_2O &\overset{K_{p1}}{\rightleftharpoons} CH_3OH + HCl \\ 2CH_3OH &\overset{K_{p2}}{\rightleftharpoons} (CH_3)_2O + H_2O \end{aligned} \tag{11}$$

Calculate equilibrium composition assuming an initially equimolar mixture of chroromethane and steam and ideal gas behaviour. Verify results by elements balance.
The problem was adopted from [1].

Calculations

Note that amount of gaseous products is equal to the amount of gaseous reactants, therefore, $K_y = K_p$. So, we can use in calculations mole fractions.

```
# File 16-1.py [View]
# Example 16. Equilibrium composition - gas phase reactions
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

def model(vars):
        x = vars[0]    # equilibrium number of moles of CH3Cl
        y = vars[1]    # equilibrium number of moles of the ether (ch3_2o)
        Kp1 = 0.00154
        Kp2 = 10.6
        eq1 = ((x-2*y)/n0 * x/n0)/((1-x)/n0 * (1-x+y)/n0) - Kp1
        eq2 = (y/n0 * (1-x+y)/n0)/((x-2*y)/n0)**2 - Kp2
        return [eq1, eq2]


# we assume total mole number of 2 (molar ratio 1:1)
n0 = 2
# try different guess values ranged from 1e-5 to 1e-2
ini = [1e-3,1e-3]
# main calculations
x,y = fsolve(model, ini)
# report ....
print('x={:.2e} y={:.2e}'.format(x, y))

# Validation of the solution
# Calculations of number of moles
n_ch3cl = 1-x
n_h2o = 1-x+y
n_ch3oh = x-2*y     # this expression could be a bit difficult to understand
n_hcl = x
n_ch3_2o = y
# Molar fractions
y_ch3cl = (1-x)/n0
y_h2o = (1-x+y)/n0
y_ch3oh = (x-2*y)/n0
y_hcl = x/n0
```

```
y_ch3_2o = y/n0
# reporting
print("y_ch3cl = {:.3e}".format(y_ch3cl))
print("y_h2o = {:.3e}".format(y_h2o))
print("y_ch3oh = {:.3e}".format(y_ch3oh))
print("y_hcl = {:.3e}".format(y_hcl))
print("y_ether = {:.3e}".format(y_ch3_2o))

# Element balance
print("Chlorine balance (1mol)")
print n_ch3cl + n_hcl
print("Carbon balance (1mol)")
print n_ch3cl + n_ch3oh + 2*n_ch3_2o
print("Oxygen balance (1mol)")
print n_h2o + n_ch3oh + n_ch3_2o
print("Hydrogen balance (5mol)")
print 3*n_ch3cl + 2*n_h2o + 4*n_ch3oh + 1*n_hcl + 6*n_ch3_2o
# final checking
Kp1calc = (y_ch3oh*y_hcl)/(y_ch3cl*y_h2o)
Kp2calc = (y_ch3_2o*y_h2o)/y_ch3oh**2
print("Calculated Kp1 = {:.2e}".format(Kp1calc))
print("Calculated Kp2 = {:.2e}".format(Kp2calc))
```

Results

```
%run 16-1.py
x=4.82e-02 y=9.44e-03
y_ch3cl = 4.759e-01
y_h2o = 4.806e-01
y_ch3oh = 1.463e-02
y_hcl = 2.408e-02
y_ether = 4.722e-03
Chlorine balance (1mol)
1.0
Carbon balance (1mol)
1.0
Oxygen balance (1mol)
1.0
Hydrogen balance (5mol)
5.0
Calculated Kp1 = 1.54e-03
Calculated Kp2 = 1.06e+01
```

Exercises

Check if at equilibrium total number of moles is equal to the initial one.

[1] Stanley M. Walas. *Chemical Reaction Engineering Handbook of Solved Problems*. CRC Press, 1995.

Example 17. Esterification (3)

•••

In example 14 we developed model for simple esterification reaction. Modify the script to plot equilibrium concentration of the ester versus initial concentration of the acid in the range $1 < c_{CH_3COOH} < 3\,\mathrm{mol\,L^{-1}}$. Tabulate the calculation results.

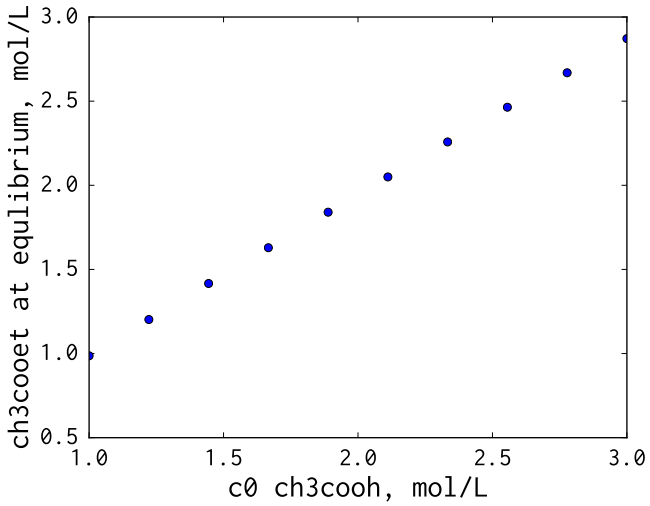
Calculations

```
# File 17-1.py [View]
# Example 17. Esterification (3)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

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

# a - alcohol, e - ester, w - water
def model(X, C_ACID):               # <--- note model has 2 arguments
    a, acid, e, w = X
    K = 4.9
    c_a = 16.0
    c_acid = C_ACID                 # Python is case-sensitive

    eq1 = (e*w)/(a*acid)-K          # equilibrium constant
    eq2 = a + e - c_a               # the alcohol balance
    eq3 = acid + e - c_acid         # the acid balance

    eq4 = e - w                     # number of ester bonds
                                    # is equivalent to
                                    # amount of water produced
    return [eq1, eq2, eq3, eq4]

# warming-up example
# alcohol, acid, ester, water
guess = [1, 1, 2, 2]                # try different guess values
a, acid, e, w = fsolve(model, guess, 3) # we pass to fsolve additional parameter
print a, acid, e, w
# end of warming-up example

print "Report, concentrations in mol/L"
print "ch3cooh_0\tetoh\t\tch3cooh\t\tch3cooet\twater"
# initial acid concentration = 1-3 mol/L
concentrations = np.linspace(1,3,10)
# array prepared to save the results
ester_concentrations = np.array([])

# calculations loop
for C_ACID in concentrations:
    # alcohol, acid, ester, water
    guess = [10, 0.1, 2, 2]       # put here most realistic guess values
    a, acid, e, w = fsolve(model, guess, C_ACID)
    print "{:.2f}\t\t{:.2f}\t\t{:.2f}\t\t{:.2f}\t\t{:.2f}".format(C_ACID, a, acid, e, w)
    ester_concentrations = np.append(ester_concentrations, e)

# and graphical report
```

```
plt.plot(concentrations, ester_concentrations, 'bo') # excess of EtOH => linear dependence
plt.xlabel('c0 ch3cooh, mol/L')
plt.ylabel('ch3cooet at equlibrium, mol/L')
plt.savefig('17-1.pdf')
plt.close()
```

Results

```
%run 17-1.py
13.1282051282 0.128205128205 2.87179487179 2.87179487179
Report, concentrations in mol/L
ch3cooh_0       etoh            ch3cooh         ch3cooet        water
1.00            15.01           0.01            0.99            0.99
1.22            14.80           0.02            1.20            1.20
1.44            14.58           0.03            1.42            1.42
1.67            14.37           0.04            1.63            1.63
1.89            14.16           0.05            1.84            1.84
2.11            13.95           0.06            2.05            2.05
2.33            13.74           0.08            2.26            2.26
2.56            13.54           0.09            2.46            2.46
2.78            13.33           0.11            2.67            2.67
3.00            13.13           0.13            2.87            2.87
```



Exercises

Plot equilibrium conversion of acetic acid versus initial concentration of the acid. Propose optimal conditions of the ester synthesis. Discuss you proposition in group.

Example 18. Effect of inert gases on the equilibrium composition

●●○

Ethylbenzene undergoes dehydrogenation into styrene at 600-650^{o}C:

$$EB \overset{K_p}{\rightleftharpoons} S + H_2 \tag{12}$$

1) Calculate equilibrium composition if the initial feed is ethylbenzene, process undergoes under 0.1 MPa and K_p=0.099. 2) Repeat calculations if initial feed is 1:10 molar ratio mixure of ethylbenzene and steam.

Calculations

Here are scripts to solve the both problems. Note that in the programs two calculation approaches are implemented: Model1 - one equation with one unknown (equilibrium conversion); and Model2 - three equations with three unknowns (numbers of moles of reagents).

1) Pure ethylbenzene as the feed:

```
# File 18-1.py [View]
# Example 18. Effect of inert gases on the equilibrium composition
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

# one unknown one equation
# EB <=> S + H2
def model1(x):
    xe = x              # equilibrium conversion
    Kp = 0.099
    n0EB = 1
    P = 0.1             # MPa
    P0 = 0.1            # MPa

    # at equilibrium
    nEB = n0EB - xe
    nS = xe
    nH2 = xe
    nTotal = nEB + nS + nH2 # = 1+xe

    yEB = nEB/nTotal
    yS = nS/nTotal
    yH2 = nH2/nTotal

    eq1 = (yS*yH2/yEB) * (P/P0) - Kp
    return eq1

guess = 0.5
print "Model1 100%EB"
xe, = fsolve(model1, guess)
print "Equilibrium conversion: {:.2f}".format(xe)

# three unknowns three equations
# EB <=> S + H2
def model2(X):
    nEB, nS, nH2 = X # number of moles
```

```
    Kp = 0.099
    n0EB = 1
    P = 0.1 # MPa
    P0 = 0.1 # MPa

    # at equilibrium
    nTotal = nEB + nS + nH2
    yEB = nEB/nTotal
    yS = nS/nTotal
    yH2 = nH2/nTotal
    eq1 = (yS*yH2/yEB) * (P/P0) - Kp
    eq2 = n0EB - nEB - nS
    eq3 = nS - nH2
    return [eq1, eq2, eq3]

guess = [0.5,0.5,0.5]
print "Model2 100%EB"
yEB, yS, yH2 = fsolve(model2, guess)
print "Equilibrium mole fractions:"
print "yEB={:.2f}".format(yEB)
print "yS={:.2f}".format(yS)
print "yH2={:.2f}".format(yH2)
```

Results

```
%run 18-1.py
Model1 100%EB
Equilibrium conversion: 0.30
Model2 100%EB
Equilibrium mole fractions:
yEB=0.70
yS=0.30
yH2=0.30
```

2) Ethylbenzene - steam as the feed:

```
# File 18-2.py [View]
# Example 18. Effect of inert gases on the equilibrium composition
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve

# one unknown one equation
# EB <=> S + H2
def model1(x):
    xe = x                  #equilibrium conversion
    Kp = 0.099
    n0EB = 1
    n0H2O = 10              # steam (inert)
    P = 0.1                 # MPa
    P0 = 0.1                # MPa

    # at equilibrium
    nEB = n0EB - xe
    nS = xe
    nH2 = xe
    nTotal = nEB + nS + nH2 + n0H2O # = 11+xe
```

```
    yEB = nEB/nTotal
    yS = nS/nTotal
    yH2 = nH2/nTotal

    eq1 = (yS*yH2/yEB) * (P/P0) - Kp
    return eq1


guess = 0.5
xe, = fsolve(model1, guess)
print "Model1 EB+steam 1:10 molar ratio"
print "Equilibrium conversion: {:.2f}".format(xe)
```

Results

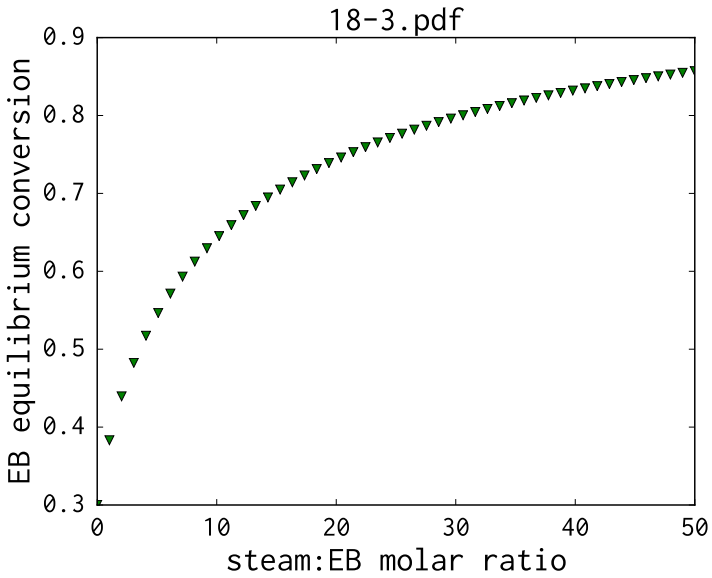
```
%run 18-2.py
Model1 EB+steam 1:10 molar ratio
Equilibrium conversion: 0.64
```

Exercises

1) Write Model2 for EB-steam system.

2) Plot equilibrium conversion of EB versus molar ratio steam:EB ranged from 0 (pure EB) to 50. You can use Model1 or Model2.

Results



Calculations

```
# File 18-3.py [View]
# Example 18. Effect of inert gases on the equilibrium composition
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License
```

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

# one unknown one equation
# EB <=> S + H2
def model1(x, steam):
    xe = x                  #equilibrium conversion
    Kp = 0.099
    n0EB = 1
    n0H2O = steam           # steam (inert)
    P = 0.1                 # MPa
    P0 = 0.1                # MPa

    # at equilibrium
    nEB = n0EB - xe
    nS = xe
    nH2 = xe
    nTotal = nEB + nS + nH2 + n0H2O # = 11+xe

    yEB = nEB/nTotal
    yS = nS/nTotal
    yH2 = nH2/nTotal

    eq1 = (yS*yH2/yEB) * (P/P0) - Kp
    return eq1


steam_amounts = np.linspace(0,50)
results = np.array([])             # to save calculated xe

for steam in steam_amounts:
    guess = 0.5
    xe, = fsolve(model1, guess, steam)
    results = np.append( results, xe )

plt.plot(steam_amounts, results, 'gv')
plt.xlabel('steam:EB molar ratio')
plt.ylabel('EB equilibrium conversion')
plt.title('18-3.pdf')
plt.savefig('18-3.pdf')
plt.close()
```

Example 19. Strong electrolyte equilibrium

● ○ ○

Calculate equilibrium concentrations of H^+, OH^-, Cl^- in a $1 \times 10^{-1}\,\mathrm{mol\,L^{-1}}$ solution of HCl. Repeat calculations for $1 \times 10^{-7}\,\mathrm{mol\,L^{-1}}$ solution.

Calculations

In the HCl solution we have three kind of species, namely H^+, OH^- and Cl^-. To calculate equilibrium concentrations, we need to create and solve an equation system containing three equations for three unknowns i.e.: 1) expression for ion product of water, 2) the electroneutrality balance and 3) chlorine balance:

```
# File 19-1.py [View]
# Example 19. Strong electrolyte equilibrium
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
from numpy import log10

def model(X, C_HCL):
    h, oh, cl = X
    c_hcl = C_HCL
    Kw = 1e-14

    eq1 = h*oh - Kw              # ion product of water
    eq2 = h - cl - oh            # the electroneutrality constraint
    eq3 = c_hcl - cl             # chlorine balance
    return [eq1, eq2, eq3]

c_hcl = 1e-1 # mol/L
guess = [1e-5, 1e-5, 1e-5]       # h, oh, cl
h, oh, cl = fsolve(model, guess, c_hcl)
pH = -log10(h)

print "Equilibrium concentrations (cHCL=0.1mol/L):"
print "[H+]={:.2e} mol/L".format(h)
print "[Cl-]={:.2e} mol/L".format(cl)
print "[OH-]={:.2e} mol/L".format(oh)
print "pH={:.2f}".format(pH)


c_hcl = 1e-7 # mol/L (only theoretically possible)
guess = [1e-5, 1e-5, 1e-5]        # h, oh, cl
h, oh, cl = fsolve(model, guess, c_hcl)
pH = -log10(h)

print "Equilibrium concentrations (cHCL=1e-7mol/L):"
print "[H+]={:.2e} mol/L".format(h)
print "[Cl-]={:.2e} mol/L".format(cl)
print "[OH-]={:.2e} mol/L".format(oh)
print "pH={:.2f}".format(pH)
```

Results

```
%run 19-1.py
Equilibrium concentrations (cHCL=0.1mol/L):
```

```
[H+]=1.00e-01 mol/L
[Cl-]=1.00e-01 mol/L
[OH-]=1.00e-13 mol/L
pH=1.00
Equilibrium concentrations (cHCL=1e-7mol/L):
[H+]=1.62e-07 mol/L
[Cl-]=1.00e-07 mol/L
[OH-]=6.18e-08 mol/L
pH=6.79
```

Exercises

Try different sets of guess values and repeat calculations. What do you observe? Write a program to calculate pH of solution of a strong base.

Example 20. Weak electrolyte equilibrium

● ○ ○

Estimate pH and concentrations of species presented in a $0.5\,\mathrm{mol\,L^{-1}}$ solution of acetic acid, $K_a = 1.8 \times 10^{-5}$. Estimate pH and concentrations of ionic and neutral species formed in a $0.1\,\mathrm{mol\,L^{-1}}$ solution of diprotic acid (e.g. itaconic acid), $pK_{a1} = 3.85$, $pK_{a2} = 5.45$.

Calculations

See example 19. Please note bass balance expressions and taking into account dissociation constants.

```
# File 20-1.py [View]
# Example 20. Weak electrolyte equilibrium
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
from numpy import log10

def model(X, C_HA):
    h, oh, ch3cooh, ch3coo = X
    c_ch3cooh = C_HA
    Kw = 1e-14
    Ka = 1.8e-5

    eq1 = h*oh - Kw                    # water ionization constant
    eq2 = h - oh - ch3coo              # the electroneutrality constraint
    eq3 = h*ch3coo/ch3cooh - Ka        # dissociation process
    eq4 = c_ch3cooh - ch3cooh - ch3coo # acetic acid balance
    return [eq1, eq2, eq3, eq4]

C_HA = 0.5 # mol/L
# h, oh, ch3cooh, ch3coo
guess = [1e-2, 1e-5, 1e-2, 1e-2]       # experiment with different, but realistic values
h, oh, ch3cooh, ch3coo = fsolve(model, guess, C_HA)
pH = -log10(h)
print "Equilibrium concentrations (cCH3COOH=0.5mol/L):"
print "[H+]={:.2e} mol/L".format(h)
print "[OH-]={:.2e} mol/L".format(oh)
print "[CH3COOH]={:.2e} mol/L".format(ch3cooh)
print "[CH3COO-]={:.2e} mol/L".format(ch3coo)
print "pH={:.2f}".format(pH)
```

Results

```
%run 20-1.py
Equilibrium concentrations (cCH3COOH=0.5mol/L):
[H+]=2.99e-03 mol/L
[OH-]=3.85e-12 mol/L
[CH3COOH]=4.97e-01 mol/L
[CH3COO-]=2.99e-03 mol/L
pH=2.52
```

Calculations

```
# File 20-2.py [View]
# Example 20. Weak electrolyte equilibrium
```

```
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from scipy.optimize import fsolve
from numpy import log10

# H2A <=> HA- + H+
# HA- <=> A2- + H+
def model(X, C_H2A):
    h, oh, h2a, ha, a = X
    c_h2a = C_H2A
    Kw = 1e-14
    # dissociation constants for itaconic acid
    Ka1 = 10**-3.85
    Ka2 = 10**-5.45

    eq1 = h*oh - Kw                    # water ionization constant
    eq2 = h - oh - ha - 2*a            # the electroneutrality constraint
    eq3 = h*ha/h2a - Ka1               # dissociation process
    eq4 = h*a/ha - Ka2                 # dissociation process
    eq5 = c_h2a - h2a - ha - a         # the acid balance
    return [eq1, eq2, eq3, eq4, eq5]

C_H2A = 0.1 # mol/L

# h, oh, h2a, ha, a
guess = [1e-2, 1e-5, 1e-2, 1e-5, 1e-5]          # experiment with realistic values
h, oh, h2a, ha, a = fsolve(model, guess, C_H2A) # check sense of the results (>0?)
pH = -log10(h)

print "Equilibrium concentrations (cH2A=0.1mol/L):"
print "[H+]={:.2e} mol/L".format(h)
print "[OH-]={:.2e} mol/L".format(oh)
print "[H2A]={:.2e} mol/L".format(h2a)
print "[HA-]={:.2e} mol/L".format(ha)
print "[A2-]={:.2e} mol/L".format(a)
print "pH={:.2f}".format(pH)
```

Results

```
%run 20-2.py
Equilibrium concentrations (cH2A=0.1mol/L):
[H+]=3.69e-03 mol/L
[OH-]=3.82e-12 mol/L
[H2A]=9.63e-02 mol/L
[HA-]=3.68e-03 mol/L
[A2-]=3.54e-06 mol/L
pH=2.43
```

Exercises

Repeat calculations for 0.001, 0.01 and 0.1 $\mathrm{mol\,L^{-1}}$ concentrations of the acids. Change guess values if necessarily. Always verify physical/chemical meaning of the results.

Example 21. Hydrolysis of a salt

●●○

Consider 0.5 mol L^{-1} solution of sodium acrylate. As a salt of weak acid (pK_a=4.25) and strong base it undergoes hydrolysis. Calculate pH and concentration of species formed at equilibrium.

Calculations

First we need to point out ionic and non-ionic species presented in the system at equilibrium. Next create and solve a non-linear algebraic equations system consisted of mass balances, chemical equilibrium equations and electroneutrality principle.

```
# File 21-1.py [View]
# Example 21. Hydrolysis of a salt
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import numpy as np
from scipy.optimize import fsolve

# pH 0.5mol/L sodium acrylate
# NaA => Na+ + A-; A- <=> HA + OH-
def model(X):
    Na, A, HA, H, OH = X
    Kw = 1e-14
    Ka = 10**-4.25
    c = 0.5

    eq1 = A*H/HA - Ka                 # dissociation constant
    #eq1 = A*H - HA*K  <-- this form of Ka expression is recommended for more complex systems
    eq2 = H*OH - Kw                   # water ion product
    eq3 = c - A - HA                  # the acid balance
    eq4 = Na + H - A - OH             # charge balance
    eq5 = Na - c                      # it is obvious
    return [eq1, eq2, eq3, eq4, eq5]

Na, A, HA, H, OH = fsolve(model, [0.5, 1e-4, 1e-4, 1e-4, 1e-4])
pH = -np.log10(H)
print '0.5 mo/L sodium acrylate'
print '[Na+]={:.4e} mol/L'.format(Na)
print '[A-]={:.4e} mol/L'.format(A)
print '[HA]={:.4e} mol/L'.format(HA)
print '[OH-]={:.4e} mol/L'.format(OH)
print '[H+]={:.4e} mol/L'.format(H)
print 'pH={:.2f}'.format(pH)
```

Results

```
%run 21-1.py
0.5 mo/L sodium acrylate
[Na+]=5.0000e-01 mol/L
[A-]=4.9959e-01 mol/L
[HA]=4.1108e-04 mol/L
[OH-]=4.1113e-04 mol/L
[H+]=4.6272e-08 mol/L
pH=7.33
```

Exercises

Modify the script to calculate pH of a $0.1\,\mathrm{mol\,L^{-1}}$ solution a salt of a strong acid and a weak base, e.g. for ammonium chloride.

Example 22. Amino acid solution

●●○

Amino acids are organic compounds having both acidic (carboxylic) and basic (amine) groups. Calculate pH of a 0.1 mol L^{-1} solution of alanine, $pK_{a1} = 2.3$, $pK_{a2} = 9.8$.

Calculations

The calculation procedure is the same as in examples 19-21.

```
# File 22-1.py [View]
# Example 22. Amino acid solution
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import numpy as np
from scipy.optimize import fsolve

# low pH ----------------------------> high pH
# +H3N-R-COOH <=> +H3N-R-COO- <=> H2N-R-COO-
# R1 <=> R2 <=> R3
def model(X):
    R1, R2, R3, H, OH = X
    Kw = 1e-14
    Ka1 = 10**-2.3
    Ka2 = 10**-9.8
    c = 0.1                         # alanine concentration, mol/L
    eq1 = R2*H/R1 - Ka1
    eq2 = R3*H/R2 - Ka2
    eq3 = H*OH - Kw
    eq4 = H + R1 - R3 - OH
    eq5 = c - R1 - R2 - R3
    return [eq1, eq2, eq3, eq4, eq5]

R1, R2, R3, H, OH = fsolve(model, [1e-4, 1e-2, 1e-4, 1e-5, 1e-6])
pH = -np.log10(H)
print "Alanine 0.1 mol/L"
print "[cationic form]={:.3e} mol/L".format(R1)
print "[zwitterion form]={:.3e} mol/L".format(R2)
print "[anionic form]={:.3e} mol/L".format(R3)
print "pH={:.2f}".format(pH)
```

Results

```
%run 22-1.py
Alanine 0.1 mol/L
[cationic form]=1.735e-05 mol/L
[zwitterion form]=9.996e-02 mol/L
[anionic form]=1.821e-05 mol/L
pH=6.06
```

Exercises

Modify script to calculate concentrations of alanine forms $^{+}H_3N-R-COOH$, $^{+}H_3N-R-COO^{-}$ and $H_2N-R-COO^{-}$ in a buffered solution at pH=3 and pH=9.

Example 23. Hydrolysis of triprotic acid salt

●●○

Alkali metal phosphates are often use as buffers components. Calculate pH of a $0.01\,\mathrm{mol\,L^{-1}}$ solution of Na_2HPO_4, pK_{a1}=2.15, pK_{a2}=7.12 and pK_{a3}=12.35.

Calculations

Note for such complex system is better to use chemical equilibrium equations in a product form (eliminate division). It makes numerical procedures more stable and the solution can be found more easily. The second remark deals with guess values. In this example we have even seven unknowns, and to get the solution it is important to pass to `fsolve()` function most realistic guess values (close to the solution).

See also examples 19, 20, 21 and 22.

```
# File 23-1.py [View]
# Example 23. Hydrolysis of triprotic acid salt
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import numpy as np
from scipy.optimize import fsolve

# pH Na2HPO4 0.01mol/L
# HPO4 <=> PO4
# H2PO4 <=> HPO4
# H3PO4 <=> H2PO4
def model(X):
    H3A, H2A, HA, A, Na, H, OH = X
    Kw = 1e-14
    Ka1 = 10**-2.15
    Ka2 = 10**-7.12
    Ka3 = 10**-12.35
    c = 0.01 # mol/L

    eq1 = H2A*H - H3A*Ka1                       # dissociation constant in a product form
    eq2 = HA*H - H2A*Ka2                        # it makes numeric calculations more easily
    eq3 = A*H - HA*Ka3                          #
    eq4 = H*OH - Kw
    eq5 = H + Na - 3*A - 2*HA - H2A - OH
    eq6 = c - A - HA - H2A - H3A
    eq7 = Na - 2*c                               # <---
    return [eq1, eq2, eq3, eq4, eq5, eq6, eq7]

H3PO4, H2PO4, HPO4, PO4, Na, H, OH = fsolve(model, [1e-6, 1e-6, 1e-6, 1e-6, 0.01, 1e-6, 1e-6])
print H3PO4              #
print H2PO4              #
print HPO4               #
print PO4                #
print Na                 #
print H                  #
print OH                 # see if values (results) have chemical sense

pH = -np.log10(H)
print "pH={:.2f}".format(pH)
```

Results

```
%run 23-1.py
2.03988972735e-12
4.35075376845e-05
0.00994311172024
1.33807400378e-05
0.02
3.31926698209e-10
3.01271336532e-05
pH=9.48
```

Exercises

Modify script to calculate pH of NaH_2PO_4 and Na_3PO_4. Hint: see equation 7. Answers: pH=4.75 (NaH_2PO_4), pH=11.87 (Na_3PO_4).

Example 24. Mixture of a weak acid and a strong base

●●○

Assume we have initially mixture of 0.1 mol acetic acid and 0.05 mol potassium hydroxide in a 1 L of water. Calculate pH of the solution.

Calculations

```
# File 24-1.py [View]
# Example 24. Mixture of a weak acid and a strong base
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import numpy as np
from scipy.optimize import fsolve

# initially:
# 0.1mol CH3COOH + 0.05mol KOH in 1L
# pH=?
def model(X):
    HA, A, K, H, OH = X
    Kw = 1e-14
    Ka = 10**-4.756                     # pKa = 4.756
    c1 = 0.1 # mol/L CH3COOH
    c2 = 0.05 # mol/L KOH

    eq1 = A*H - HA*Ka                   # dissociation constant in a product form
    eq2 = H*OH - Kw                     # water ionization product
    eq3 = H + K - A - OH                # charge balance
    eq4 = c1 - A - HA                   # the acid balance
    eq5 = K - c2                        # the potassium cation balance
    return [eq1, eq2, eq3, eq4, eq5]

HA, A, K, H, OH = fsolve(model, [1e-5, 1e-5, 0.05, 1e-5, 1e-5])
print HA, A, K, H, OH  # see if values (results) have chemical sense
pH = -np.log10(H)
print "pH={:.2f}".format(pH)
```

Results

```
%run 24-1.py
0.0499824740566 0.0500175259434 0.05 1.75265139625e-05 5.70563825202e-10
pH=4.76
```

Exercises

Calculate pH for different molar ratios of the acid to the base. Rewrite the script to compute pH of a mixture of hydrochloric acid with tris(hydroxymethyl)aminomethane (Tris).

5 Chemical Kinetics

Example 25. Concentration profiles

● ○ ○

Assume the first order reaction: $A \rightarrow B$. Plot concentration of A versus reaction time if $k = 1.4 \times 10^{-3}\,\mathrm{s}^{-1}$. Compare analytical solution with numeric one.

Calculations

The rate equation for a first-order reaction $A \rightarrow$ products

$$-\frac{d[A]}{dt} = k[A] \tag{13}$$

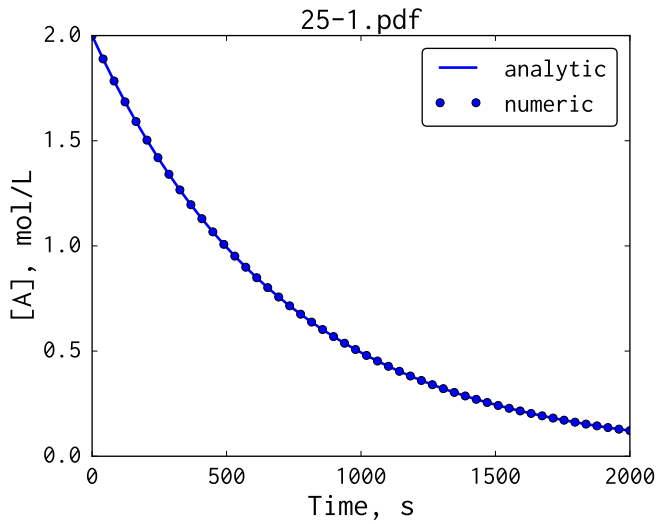
Rearrangement and integration of eq. 13 give:

$$[A] = [A]_0 \exp(-kt) \tag{14}$$

where k is a rate constant and $[A]_0$ is initial A concentration. In the following script we are going to numerically integrate eq. 13 and for comparison to use simply analytical expression eq. 14. Run the script:

```
# File 25-1.py [View]
# Example 25. Concentration profiles
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
from scipy.misc import derivative

# The reaction: A -> B, k

# Approach 1
# Analytic solution
def analytic(t):
    return A0*np.exp(-k*t)

# Approach 2
# Numerical integration of ODEs
def model(y, t):
    A = y[0]             # unpack variables
    B = y[1]
    dAdt = -k*A          # calculate derivatives
    dBdt = k*A
    return [dAdt, dBdt]  # return derivatives

# Parameters
k = 1.4e-3               # 1/s
A0 = 2                   # mol/L
B0 = 0

# Time span
t = np.linspace(0, 2000) # 0 - 2000s
# Initial conditions
ic = [A0, B0]
```

```
# Calculate concentration vs. time profiles
# analytically
Aa = analytic(t)
# numerically
results = odeint(model, ic, t)
An = results[:,0]

# Compare the results graphically
plt.plot(t, Aa, 'b-', label = 'analytic')
plt.plot(t, An, 'bo', label = 'numeric')
plt.ylabel('[A], mol/L')
plt.xlabel('Time, s')
plt.legend()
plt.title('25-1.pdf')
plt.savefig('25-1.pdf')
plt.close()
```



Exercises

Plot concentration of B vs. reaction time.

Example 26. Rate of reaction

● ○ ○

Please have a look at example 25. In this example we calculate the reaction rate. Suppose an elementary reaction $A \rightarrow B$. Compute reaction rate at time ranged from 0 to 2000 s if $k = 1.4 \times 10^{-3}\,\mathrm{s}^{-1}$.

Calculations

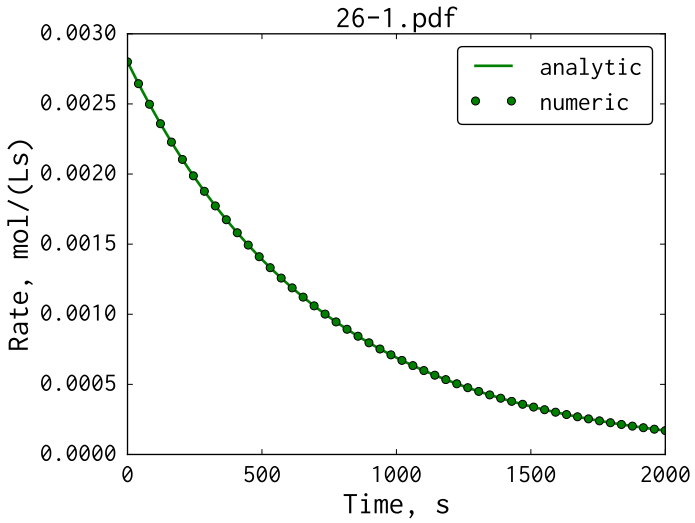
Please remember that the rate (instantaneous rate) is a derivative, and can be computed at different reaction times. In other words, the instantaneous rate of a reaction is the reaction rate at any given point in time.

Have a look at the script:

```
# File 26-1.py [View]
# Example 26. Rate of reaction
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint  # to solve ODE numerically
from scipy.misc import derivative   # to calculate derivative

# The reaction: A -> B, k

# Approach 1
# Analytic solution (possible only for relatively simple mechanisms)
def analytic(t):
    return A0*np.exp(-k*t)

# Approach 2
# Numerical integration of ODEs
def model(y, t):
    A = y[0]            # unpack variables
    B = y[1]
    dAdt = -k*A         # calculate derivatives
    dBdt = k*B
    return [dAdt, dBdt]  # return derivatives

# Parameters
k = 1.4e-3              # 1/s
A0 = 2                  # mol/L
B0 = 0

# Time span
t = np.linspace(0, 2000) # 0 - 2000s
# Initial conditions
ic = [A0, B0]

# Calculate concentration vs. time profiles
# analytically
Aa = analytic(t)
# numerically
results = odeint(model, ic, t)
An = results[:,0]

# Let's calculate the reaction rates
# r= -dA/dt = k*A so
```

```
rdef = k*Aa # or k*An
# or calculate derivatives numerically
rderiv = -derivative(analytic, t)

# We can calculate the rate at different reaction time
# special meaning have the initial rate (at t=0)
# Let's calculate - numerically
r0 = -derivative(analytic, 0)
print("Initial rate (derivative):")
print(r0)
# or from the rate definition
# note that at t=0 A=A0
r00 = k*A0
print("Initial rate (def):")
print(r00)
# or based on derivative approximation
print("Initial rate (approximation):")
r0aprox = -(Aa[1] - Aa[0])/(t[1] - t[0])
print(r0aprox)

# Compare the results
plt.plot(t, rdef, 'g-', label = 'analytic')
plt.plot(t, rderiv, 'go', label = 'numeric')
plt.ylabel('Rate, mol/(Ls)')
plt.xlabel('Time, s')
plt.legend()
plt.title('26-1.pdf')
plt.subplots_adjust(left=0.2,right=0.9) # to make plot more clear
plt.savefig('26-1.pdf')
plt.close()
```

After running the program you should see plot of reaction rate vs. time:



and calculated initial rates:

```
%run 26-1.py
Initial rate (derivative):
0.00280000091467
```

```
Initial rate (def):
0.0028
Initial rate (approximation):
0.00272150228725
```

Exercises

Print values of arrays `Aa` and `t`. What is meaning of the code: `Aa[1]-Aa[0]` and `t[1]-t[0]`? Compare calculated initial rates of the reaction with the plot. Try to estimate the reaction rate at 1000 s by graphical method and by calculations.

Example 27. Second order kinetics

● ○ ○

Consider second-order reaction: $A + B \rightarrow C$. Plot concentrations $[A]$, $[B]$ and $[C]$ versus time from 0 to 1200 s if $k = 3 \times 10^{-3}\,\mathrm{L\,mol^{-1}\,s^{-1}}$.

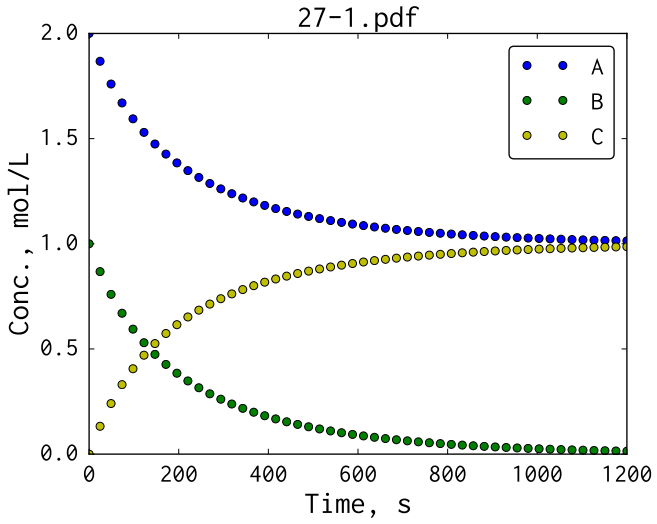
Calculations

```
# File 27-1.py [View]
# Example 27. Second order kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint  # to solve ODE numerically
from scipy.misc import derivative   # to calculate derivative

# The reaction: A + B -> C, k

def model(y, t):
    A = y[0]                    # unpack variables
    B = y[1]
    C = y[2]
    dAdt = -k*A*B               # calculate derivatives
    dBdt = -k*A*B
    dCdt = k*A*B
    return [dAdt, dBdt, dCdt]  # return derivatives

# Parameters
k = 3e-3                        # L/(mol s)
A0 = 2                          # mol/L
B0 = 1                          # mol/L
C0 = 0                          # mol/L

# Time span
t = np.linspace(0, 1200) # 0 - 1200s
# Initial conditions
ic = [A0, B0, C0]

# Integrate numerically ODEs
results = odeint(model, ic, t)
# Unpack concentrations
A = results[:,0]
B = results[:,1]
C = results[:,2]

# checking the results
# moles balance:
print C+A # should be A0
print C+B # should be B0


# Plot the results
plt.plot(t, A, 'bo', label = 'A')
plt.plot(t, B, 'go', label = 'B')
plt.plot(t, C, 'yo', label = 'C')
plt.ylabel('Conc., mol/L')
```

```
plt.xlabel('Time, s')
plt.legend()
plt.title('27-1.pdf')
plt.savefig('27-1.pdf')
plt.close()
```

and here is the output - moles balance at different reaction time $A(t) + C(t) = A_0$ and $B(t) + C(t) = B_0$:

```
%run 27-1.py
[ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
[ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```



Exercises

Repeat calculations for:

1) $[A]_0$=0 $[B]_0$=1 mol L^{-1} k=3 × 10^{-3} L mol^{-1} s^{-1}

2) $[A]_0$=2 mol L^{-1} $[B]_0$=1 mol L^{-1} k=3 × 10^{-4} L mol^{-1} s^{-1}

3) $[A]_0$=2 mol L^{-1} $[B]_0$=2 mol L^{-1} k=3 × 10^{-3} L mol^{-1} s^{-1}

Example 28. First order equilibrium kinetics

● ○ ○

Suppose a first order equilibrium reaction:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \tag{15}$$

where $k_1 = 0.06\,\mathrm{min}^{-1}$ and $k_2 = 0.012\,\mathrm{min}^{-1}$. Plot concentrations of A and B versus reaction time (0–60 min).

Calculations

Note that scheme eq. 15 can be written in an equivalent form as:

$$\begin{aligned} A &\xrightarrow{k_1} B \\ B &\xrightarrow{k_2} A \end{aligned} \tag{16}$$

thus the system of ordinary differential equations (ODE) that describes the concentrations of A and B is given by:

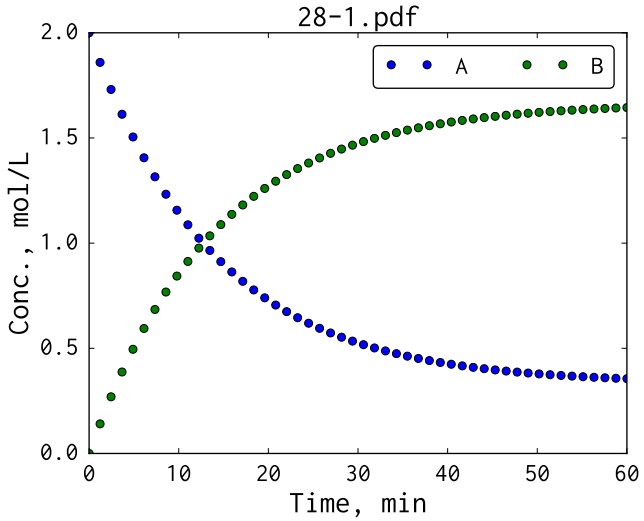
$$\begin{aligned} \frac{dA}{dt} &= -k_1 A + k_2 B \\ \frac{dB}{dt} &= k_1 A - k_2 B \end{aligned} \tag{17}$$

In the script we are going to solve the system eq. 17 numerically:

```
# File 28-1.py [View]
# Example 28. First order equilibrium kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint  # to solve ODE numerically
from scipy.misc import derivative   # to calculate derivative

# The reaction:
# A -> B, k1
# B -> A, k2
def model(y, t):
    A = y[0]                    # unpack variables
    B = y[1]
    dAdt = -k1*A + k2*B         # calculate derivatives
    dBdt = k1*A - k2*B
    return [dAdt, dBdt]  # return derivatives

# Parameters
k1 = 1e-3*60                    # 1/min
k2 = 2e-4*60                    # 1/min
A0 = 2                          # mol/L
B0 = 0                          # mol/L
# Time span
t = np.linspace(0, 60) # 0 - 60min
# Initial conditions
ic = [A0, B0]
# Integrate numerically ODEs
results = odeint(model, ic, t)
# Unpack concentrations
A = results[:,0]
```

```
B = results[:,1]
# checking the results
# moles balance:
print "Moles balance: B+A:"
print B+A # should be A0+B0

# Plot the results
plt.plot(t, A, 'bo', label = 'A')
plt.plot(t, B, 'go', label = 'B')
plt.ylabel('Conc., mol/L')
plt.xlabel('Time, min')
plt.legend(ncol=2)              # to make the plot nicer
plt.title('28-1.pdf')
plt.savefig('28-1.pdf')
plt.close()

# print final concentrations (at equilibrium)
print "A_eq: {:.2f} mol/L".format(A[-1])
print "B_eq: {:.2f} mol/L".format(B[-1])
```

You should see the plot



and results of moles balance checking $A(t) + B(t) = A_0 + B_0$ at different reaction time:

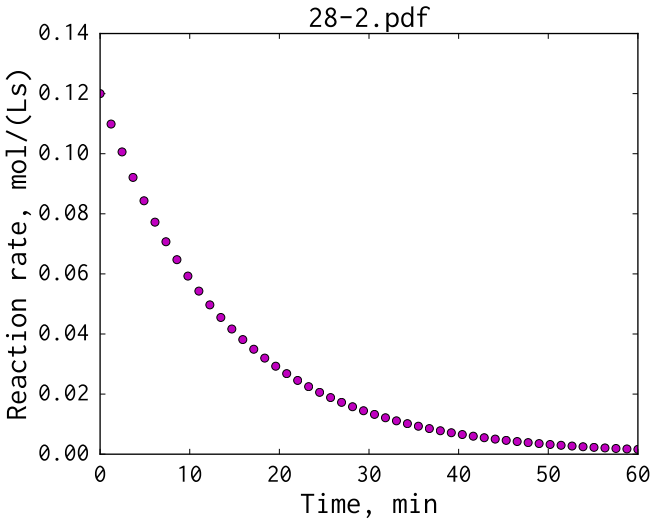
```
%run 28-1.py
Moles balance: B+A:
[ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
A_eq: 0.36 mol/L
B_eq: 1.64 mol/L
```

Exercises

1) Calculate equilibrium constant. 2) Plot reaction rate versus time.

Results

Your plot should look like this:



Please note reaction rate at equilibrium.

Example 29. Second order equilibrium kinetics

● ○ ○

Suppose second order equilibrium reaction:

$$2A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} B \tag{18}$$

where $k_1 = 2 \times 10^{-3}\,\mathrm{s}^{-1}$ and $k_2 = 3 \times 10^{-4}\,\mathrm{s}^{-1}$. Plot concentrations of A and B versus reaction time (0–6000 s) if $[A]_0$ =2 mol L^{-1}.

Calculations

Note that scheme eq. 18 can be written in equivalent form as:

$$\begin{aligned} A + A &\xrightarrow{k_1} B \\ B &\xrightarrow{k_2} A + A \end{aligned} \tag{19}$$

thus the system of ODEs that describes the concentrations of A and B is given by:

$$\begin{aligned} \frac{dA}{dt} &= -k_1 AA - k_1 AA + k_2 B + k_2 B = -2k_1 A^2 + 2k_2 B \\ \frac{dB}{dt} &= k_1 AA - k_2 B = k_1 A^2 - k_2 B \end{aligned} \tag{20}$$

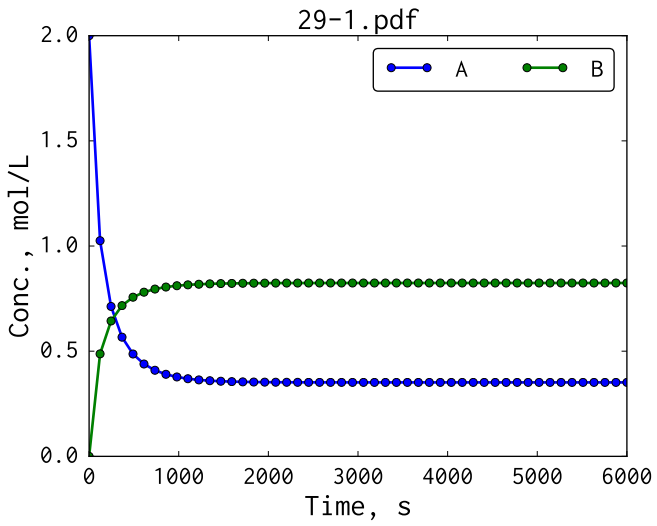
Note that based on mole balance at each reaction time the following expression must be true: $A(t) + 2B(t) = A_0 + B_0$.

In the script we are going to solve the system eq. 20 numerically:

```
# File 29-1.py [View]
# Example 29. Second order equilibrium kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint  # to solve ODE numerically
from scipy.misc import derivative   # to calculate derivative

# The reaction:
# 2A -> B, k1
# B -> 2A, k2
def model(y, t):
    A = y[0]                    # unpack variables
    B = y[1]
    dAdt = -2*k1*A*A + 2*k2*B    # calculate derivatives
    dBdt = k1*A*A - k2*B
    return [dAdt, dBdt]  # return derivatives

# Parameters
k1 = 2e-3                       # L/mols
k2 = 3e-4                       # 1/s
A0 = 2                          # mol/L
B0 = 0                          # mol/L
# Time span
t = np.linspace(0, 6000) # 0 - 6000s
# Initial conditions
ic = [A0, B0]
# Integrate numerically ODEs
```

```
results = odeint(model, ic, t)
# Unpack concentrations
A = results[:,0]
B = results[:,1]
# checking the results
# moles balance:
print 2*B+A # should be A0+B0

# Plot the results
plt.plot(t, A, 'bo-', label = 'A')
plt.plot(t, B, 'go-', label = 'B')
plt.ylabel('Conc., mol/L')
plt.xlabel('Time, s')
plt.legend(ncol=2)              # to make the plot nicer
plt.title('29-1.pdf')
plt.savefig('29-1.pdf')
plt.close()
```

You should see the plot



and results of moles balance checking at different reaction time:

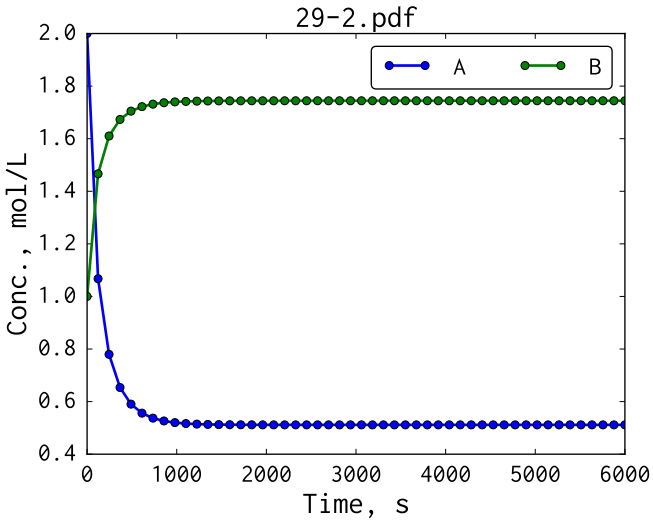
```
%run 29-1.py
[ 2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.
  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.  2.]
```

Exercises

1) Calculate equilibrium constant. 2) Repeat simulation for $[A]_0$ =2 mol L^{-1} and $[B]_0$ =1 mol L^{-1}.

Results

1) Add to the script a line of code: `print B[-1]/A[-1]**2`, then you should see in the output the value of equilibrium constant of `6.666`. Note that $K = \frac{k_1}{k_2} = 2 \times 10^{-3} / 3 \times 10^{-4} = 6.666$.
2) Your plot should look like this:


29-2.pdf
A
B
Conc., mol/L
Time, s
2.0
1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
1000
2000
3000
4000
5000
6000

Example 30. Autocatalytic reaction

●●○

Autocatalytic reactions are those in which at least one of the products is a reactant. In other word, a reaction is said to be autocatalytic if a product of the reaction causes it to go faster. Consider the simplest example of autocatalytic reaction: $A+B \xrightarrow{k} 2B$. Suppose $k = 2.16\,\mathrm{L\,mol^{-1}\,h^{-1}}$, $[A]_0 = 2\,\mathrm{mol\,L^{-1}}$ and $[B]_0 = 0.0001\,\mathrm{mol\,L^{-1}}$ plot $[A]$ and $[B]$ versus time (0-5 h). Additionally, calculate reaction rate and plot dependence of the rate as function of time.

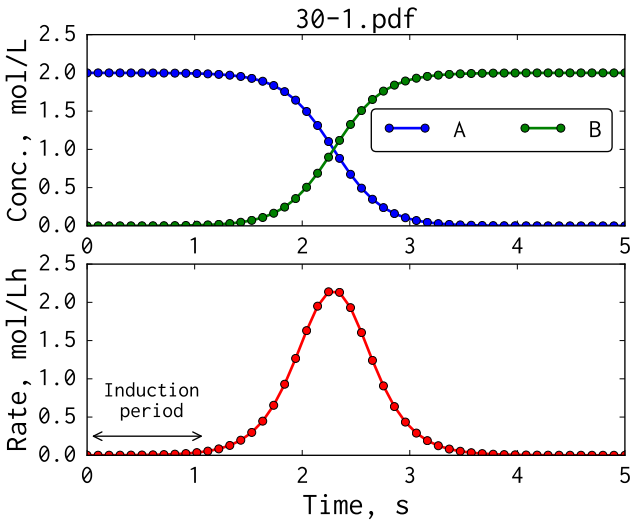
Calculations

To create kinetic model study previous examples, next run the script:

```
# File 30-1.py [View]
# Example 30. Autocatalytic reaction
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint  # to solve ODE numerically
from scipy.misc import derivative   # to calculate derivative

# The reaction:
# A + B -> 2B, k1
def model(y, t):
    A = y[0]                    # unpack variables
    B = y[1]
    dAdt = -k1*A*B              # calculate derivatives
    dBdt = k1*A*B
    return [dAdt, dBdt]         # return derivatives

k1 = 2.16                      # L/molh
A0 = 2                         # mol/L
B0 = 0.0001                    # mol/L, traces of B
t = np.linspace(0, 5) # 0 - 5h
ic = [A0, B0]
results = odeint(model, ic, t)
A = results[:,0]
B = results[:,1]
# calculate reaction rate
rate = k1*A*B
# checking the results - moles balance:
print B+A # should be A0+B0
# Plot the results - 2 plots per graph
fig = plt.figure()
ax1 = plt.subplot(211)
ax1.plot(t, A, 'bo-', label = 'A')
ax1.plot(t, B, 'go-', label = 'B')
ax1.set_ylabel('Conc., mol/L')
ax1.set_xlabel('Time, s')
ax1.legend(ncol=2, loc='center right')  # to make the plot nicer
ax1.set_title('30-1.pdf')
ax2 = plt.subplot(212)
ax2.plot(t, rate, 'ro-')
ax2.set_ylabel('Rate, mol/Lh')
ax2.set_xlabel('Time, s')
```

```
ax2.annotate("", xy=(0.02,0.25), xytext=(1.1,0.25), arrowprops=dict(arrowstyle='<->'))
ax2.text(0.6, 0.5, 'Induction\nperiod', horizontalalignment='center')
plt.savefig('30-1.pdf')
plt.close()
```

Please note unusual shape of the rate curve:



Results of checking of the mole balance should look like this:

```
%run 30-1.py
[ 2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001
  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001
  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001
  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001
  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001  2.0001
  2.0001  2.0001  2.0001  2.0001  2.0001]
```

Exercises

Study dependence of autocatalytic reaction rate on time. What is characteristic for such type of reactions?

Study effect of initial concentration of B on the reaction kinetic. Assume $[B]_0$ from 0 to 0.1 mol L^{-1}.

See also example 45.

Example 31. Reversible monomolecular system

●●○

The isomerization of butenes (1-butene (A), *cis*-2-butene (B), and *trans*-2-butene (C)) over alumina catalyst at 230°C has been studied[1][2]. It was found that kinetics of the system can be described by three-component reversible system:

$$\begin{aligned} A &\underset{k_{21}}{\overset{k_{12}}{\rightleftharpoons}} B \\ A &\underset{k_{31}}{\overset{k_{13}}{\rightleftharpoons}} C \\ B &\underset{k_{32}}{\overset{k_{23}}{\rightleftharpoons}} C \end{aligned} \tag{21}$$

where relative rate constants are: $k_{12} = 10.344$, $k_{21} = 4.623$, $k_{13} = 3.724$, $k_{31} = 1.000$, $k_{23} = 5.616$, and $k_{32} = 3.371$. Calculate equilibrium concentrations of hydrocarbons (mole fractions) if feed gas contains:

1. Pure 1-butene,
2. Pure *cis*-2-butene, or
3. Pure *trans*-2-butene.

In calculations to use normalized time (from 0 to 1).

Calculations

The system of differential equations representing the behavior of the system (21) looks like this:

$$\begin{aligned} \frac{dA}{dt} &= -k_{12}A + k_{21}B - k_{13}A + k_{31}C \\ \frac{dB}{dt} &= k_{12}A - k_{21}B + k_{32}C - k_{23}B \\ \frac{dC}{dt} &= k_{13}A - k_{31}C + k_{23}B - k_{32}C \end{aligned} \tag{22}$$

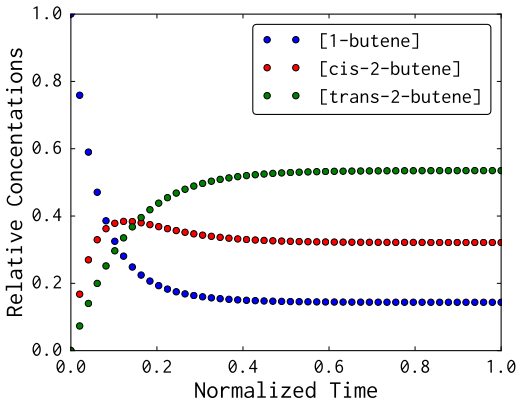
Numerical integration of the system (22) for initial conditions of A_0=1 or B_0=1 or C_0=1 gives the answer:

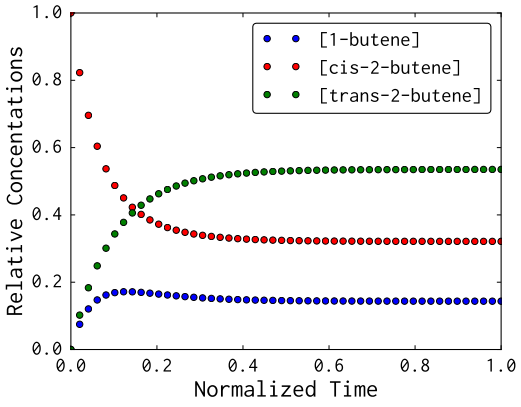
```
# File 31-1.py [View]
# Example 31. Reversible monomolecular system
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy
from scipy.integrate import odeint

# a = 1-butene, b = cis-2-butene, c = trans-2-butene
def model(y, t):
    a = y[0]
    b = y[1]
    c = y[2]
    #relative rate constants
    k12 = 10.344
    k21 = 4.623
    k31 = 1
    k13 = 3.724
```

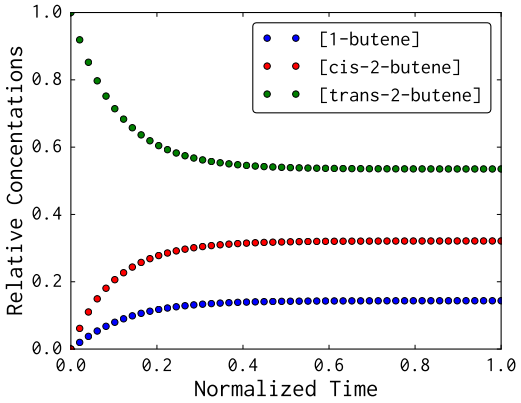
```
    k23 = 5.616
    k32 = 3.371
    # ODEs
    dadt = -k12*a + k21*b -k13*a + k31*c
    dbdt = k12*a - k21*b + k32*c - k23*b
    dcdt = k13*a - k31*c + k23*b - k32*c
    return [dadt, dbdt, dcdt]

t = numpy.linspace(0, 1) # normalized time
# case 1: pure 1-butene
a0=1
b0=0
c0=0
initial = [a0, b0, c0]
results = odeint( model, initial, t )
# Reagents mole fractions:
a = results[:,0]
b = results[:,1]
c = results[:,2]

fig, ax = plt.subplots()
ax.plot(t, a, 'bo', label='[1-butene]')
ax.plot(t, b, 'ro', label='[cis-2-butene]')
ax.plot(t, c, 'go', label='[trans-2-butene]')
ax.legend()
ax.set_xlabel('Normalized Time')
ax.set_ylabel('Relative Concentations')
fig.savefig('31-1a.pdf')

print('Final (at equilibrium) mole fractions:')
print('A={:.3f} B={:.3f} C={:.3f}'.format(a[-1], b[-1], c[-1]) )
```

The equilibrium values are:

```
%run 31-1.py
Final (at equilibrium) mole fractions:
A=0.144 B=0.321 C=0.535
```







[1] Werner O Haag and Herman Pines. “The Kinetics of Carbanion-catalyzed Isomerization of Butenes and 1-Pentene1-3”. In: *Journal of the American Chemical Society* 82.2 (1960), pp. 387–391.

[2] James Wei and Charles D Prater. “The structure and analysis of complex reaction systems”. In: *Advances in Catalysis* 13 (1962), pp. 203–392.

Example 32. Fractional order reactions

●●○

Free-radical polymerization of a polar monomers (M) initiated by persulfates (I) often follows complex mechanism, simplified to: $M + I \rightarrow$ Polymer. Assume that empirical rate equations have the following forms:

$$\begin{aligned} \frac{dM}{dt} &= -k_1[I]^{a_1}[M]^{b_1} \\ \frac{dI}{dt} &= -k_2[I]^{a_2}[M]^{b_2} \end{aligned} \tag{23}$$

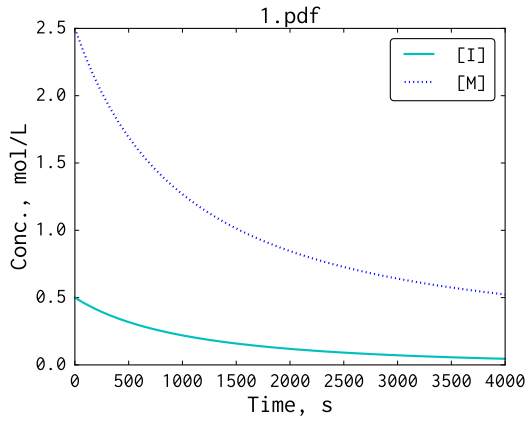
and the rate constants and the exponents are k_1=5 × 10^{-4}, k_2=1 × 10^{-3}, a_1=0.9, b_1=0.7, a_2=0.5 and b_2=1.3. Plot concentrations of the monomer and the initiator versus time from 0 to 4000 s. Plot reaction rates versus polymerization time.

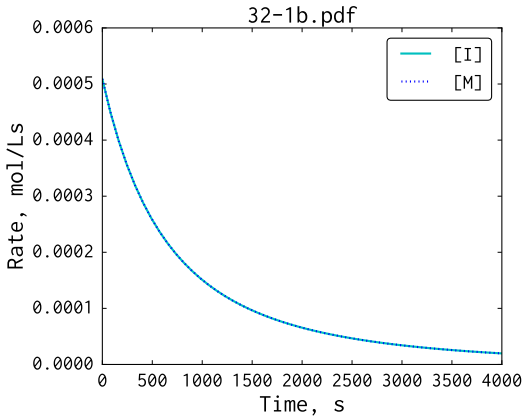
Calculations

```
# File 32-1.py [View]
# Example 32. Fractional order reactions
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

#empirical rate expression 1
def rI(I,M):
    k1 = 5e-4
    a1 = 0.9
    b1 = 0.7
    return -k1 * I**a1 * M**b1

#empirical rate expression 2
def rM(I,M):
    k2 = 1e-3
    a2 = 0.5
    b2 = 1.3
    return -k2 * I**a2 * M**b2

# ODEs
def model(y, t):
    I = y[0]
    M = y[1]
    dIdt = rI(I,M)
    dMdt = rM(I,M)
    return [dIdt, dMdt]

# initial conditions:
I0 = 0.5 # mol/L
M0 = 2.5 # mol/L

t = np.linspace(0,4000)
results = odeint(model, [I0, M0], t)
I = results[:,0]
M = results[:,1]
dIdt = rI(I,M)
dMdt = rI(I,M)
```

```
plt.plot(t, I, 'c-', label='[I]')
plt.plot(t, M, 'b:', label='[M]')
plt.legend(loc='best')
plt.xlabel('Time, s')
plt.ylabel('Conc., mol/L')
plt.title('1.pdf')
plt.savefig('32-1a.pdf')
plt.close()
plt.subplots_adjust(left=0.25, right=0.9)
plt.plot(t, -dIdt, 'c-', label='[I]')      # to have positive rate
plt.plot(t, -dMdt, 'b:', label='[M]')
plt.legend(loc='upper right')
plt.xlabel('Time, s')
plt.ylabel('Rate, mol/Ls')
plt.title('32-1b.pdf')
plt.savefig('32-1b.pdf')
```

Results





Exercises

Check instant mass balance for this system. Try varying the exponents to see what happens.

Example 33. Kinetic model for oxidation process (1)

●●○

Suppose an oxidation process follows the mechanism:

$$A + B \xrightarrow{k_1} C \tag{24}$$

$$C + B \xrightarrow{k_2} D \tag{25}$$

where A is a substrate, B is an oxidant, C and D are desired and undesired products, respectively. Assume that initial concentration of the substrate is held at $1\,\mathrm{mol\,L^{-1}}$ and rate constant are $k_1 = 3.6\,\mathrm{L\,mol^{-1}\,h^{-1}}$ and $k_2 = 0.36\,\mathrm{L\,mol^{-1}\,h^{-1}}$. Let initial concentration of the oxidant in the range from 0 to $3\,\mathrm{mol\,L^{-1}}$.
Plot concentrations of reagents as a function of time from 0 to 5 h. Compare conversion degree of A with yield and selectivity of C.

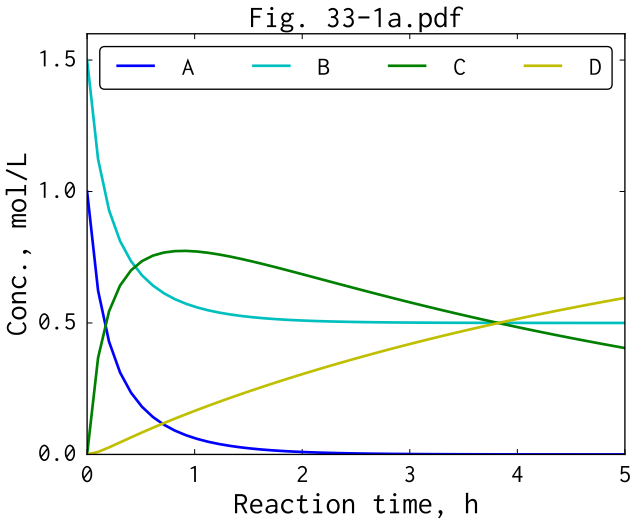
Calculations

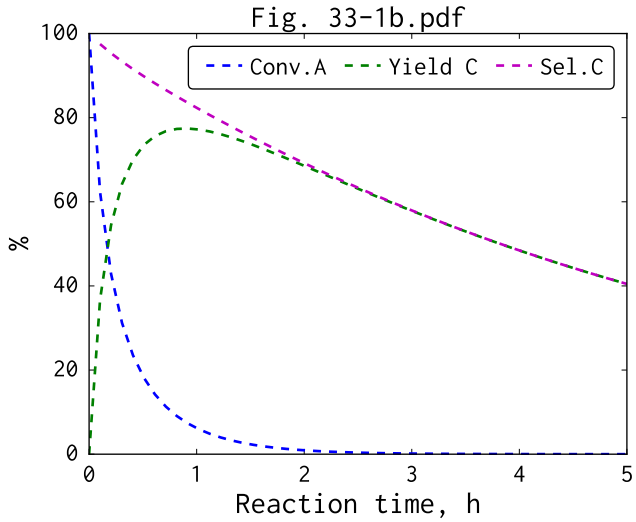
The following script can be used to solve the problem. Study how the program works:

```
# File 33-1.py [View]
# Example 33. Kinetic model for oxidation process (1)

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# A + B -> C, k1
# B + C -> D, k2
# A - a substrate, B - an oxidant, C - desired, D - undesired product

def model(y, t):
    A = y[0]
    B = y[1]
    C = y[2]
    D = y[3]
    #rate constants
    k1 = 1e-3*3600
    k2 = 1e-4*3600            # <--- change the value, see the effect (Y & S)
    # ODEs
    dAdt = -k1*A*B
    dBdt = -k1*A*B
    dCdt = k1*A*B - k2*B*C
    dDdt = k2*B*C
    return [dAdt, dBdt, dCdt, dDdt]

A0=1.0
B0=1.5
C0=0
D0=0
ic = [A0, B0, C0, D0]          # initial conditions
t = np.linspace(0,5)
results = odeint( model, ic, t)
A = results[:,0]
B = results[:,1]
C = results[:,2]
```

```
D = results[:,3]

alpha = 100*A/A0              # calculate the substrate conversion
Y = 100*C/A0                  # calculate the product yield
S = 100*C/(C+D)               # calculate process selectivity

fig,ax1 = plt.subplots()
ax1.set_title('Fig. 33-1a.pdf')
ax1.plot(t ,A, label='A')
ax1.plot(t ,B, label='B')
ax1.plot(t ,C, label='C')
ax1.plot(t ,D, label='D')
ax1.legend(loc='upper right', ncol=4)
ax1.set_yticks([0,0.5,1.0,1.5])
ax1.set_ylabel('Conc., mol/L')
ax1.set_xlabel('Reaction time, h')
plt.savefig('33-1a.pdf')
plt.close()

fig,ax1 = plt.subplots()
ax1.set_title('Fig. 33-1b.pdf')
ax1.plot(t , alpha, 'b--', label='Conv.A')
ax1.plot(t ,Y, 'g--',  label='Yield C')
ax1.plot(t ,S, 'm--',  label='Sel.C')
ax1.set_xlabel('Reaction time, h')
ax1.set_ylabel('%')
ax1.legend(loc='upper right', ncol=3, handletextpad=0,
        columnspacing=0.4)
plt.savefig('33-1b.pdf')
plt.close()
```

Here are the plots:



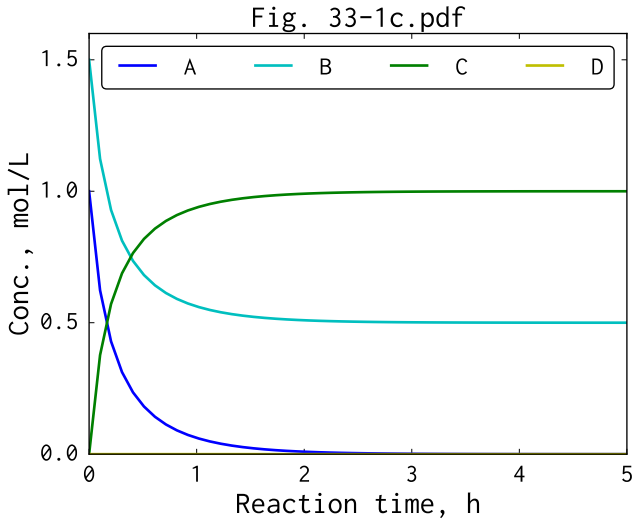


Exercises

Study influence of rate of the side reaction (eq. eq:kinox1:2) on concentrations and conversion-yield-selectivity relations.

Results

Here are results for B_0=1.5 mol L^{-1} and k_2=0. Note absence of byproduct in the reaction mixture.



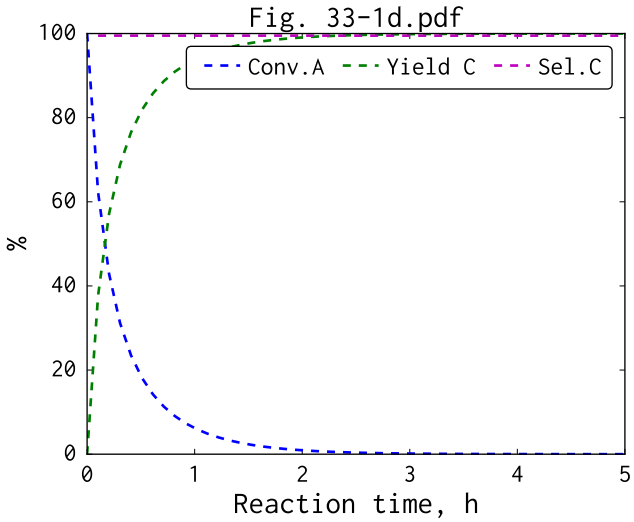


Fig. 33-1d.pdf
100
80
60
40
20
0
%
0
1
2
3
4
5
Reaction time, h
Conv.A
Yield C
Sel.C

Example 34. Kinetic model for oxidation process (2)

•••

Suppose an oxidation process follows the mechanism:

$$\begin{aligned} A + B &\xrightarrow{k_1} C \\ C + B &\xrightarrow{k_2} D \end{aligned} \tag{26}$$

where A is a substrate, B is an oxidant, C and D are desired and undesired products, respectively. Assume that initial concentration of the substrate is held at $1\,\mathrm{mol\,L^{-1}}$ and rate constant are $k_1 = 3.6\,\mathrm{L\,mol^{-1}\,h^{-1}}$ and $k_2 = 0.36\,\mathrm{L\,mol^{-1}\,h^{-1}}$. Let initial concentration of the oxidant in the range from 0 to $3\,\mathrm{mol\,L^{-1}}$ and simulation time from 0 to 5 h.

Plot concentration of desired product as a function of time and initial concentration of the oxidant. Make 2D surface plot.

This example is continuation of the example 33.

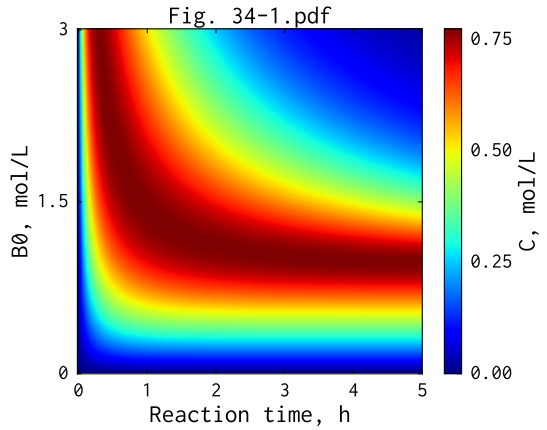
Calculations

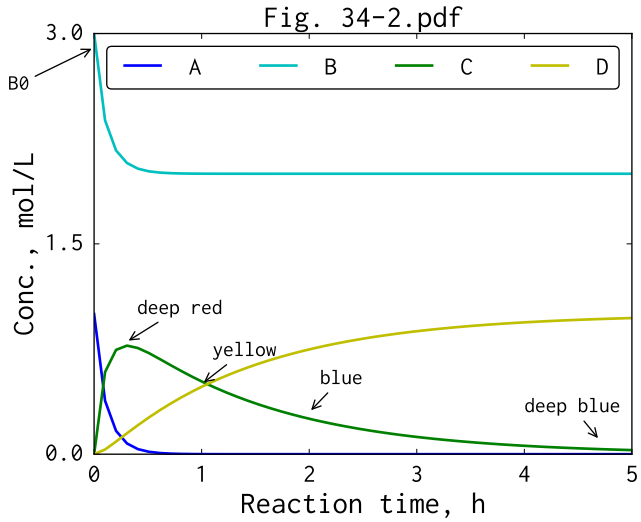
The following script can be used to solve the problem. Study how the program works:

```
# File 34-1.py [View]
# Example 34. Kinetic model for oxidation process (2)
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
from scipy.integrate import odeint

# A + B -> C, k1
# B + C -> D, k2
# A - a substrate, B - an oxidant, C - desired, D - undesired product
def model(y, t):
    A = y[0]
    B = y[1]
    C = y[2]
    D = y[3]
    #rate constants
    k1 = 3.60
    k2 = 0.36
    # ODEs
    dAdt = -k1*A*B
    dBdt = -k1*A*B
    dCdt = k1*A*B - k2*B*C
    dDdt = k2*B*C
    return [dAdt, dBdt, dCdt, dDdt]

def run_simulation(b0):                    # simulate the reaction
    A0=1.0                                 # for given B0
    B0=b0
    C0=0
    D0=0
```

```
    ic = [A0, B0, C0, D0]                 # initial conditions
    t = np.linspace(0, t_max,N)           # time span
    results = odeint( model, ic, t)       # results - concentrations(t)
    C = results[:,2]
    D = results[:,3]
    return C

N = 150                                   # number of steps
t_max = 5                                 # reaction time in hour
B0_min = 0                                # B0 start
B0_max = 3                                # B0 end

data = np.empty([0,N])                    # prepare matrix for results: D(B0,t)
b0_range = np.linspace(B0_min, B0_max, N)
for b0 in b0_range:
    C =run_simulation(b0)
    data = np.insert(data, 0, C, axis=0)  # insert results (rows) in reversed order

# make a 2D surface plot
fig, ax = plt.subplots()
imgplot = ax.imshow(data, interpolation='none', cmap=cm.jet)
cbar = fig.colorbar(imgplot, ticks=[0,0.25, 0.5, 0.75])
cbar.set_label('C, mol/L')
ax.set_xticks([0, N/5, 2*N/5, 3*N/5, 4*N/5, N])
ax.set_xticklabels([0, t_max/5, 2*t_max/5, 3*t_max/5, 4*t_max/5, t_max])
ax.set_xlabel('Reaction time, h')
ax.set_yticks([0, N/2.0, N])
ax.set_yticklabels([B0_max, (B0_max-B0_min)/2.0, B0_min])
ax.set_ylabel('B0, mol/L')
ax.set_title('Fig. 34-1.pdf')
plt.savefig('34-1.pdf')
plt.close()
```

The results are shown on Fig. 34-1.pdf. Fig. 34-2.pdf could help understanding structure of the color map (Fig 34-1.pdf).



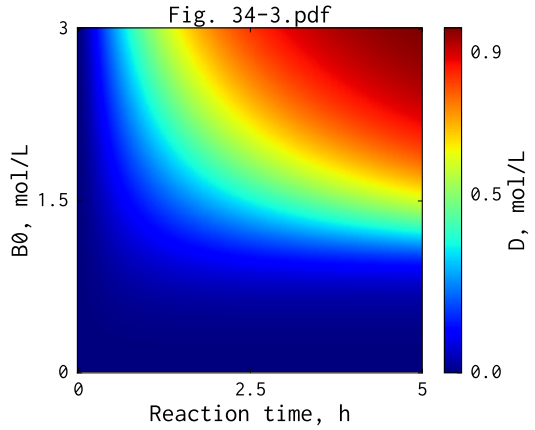


Exercises

Try to explain changes in the product concentrations vs. reaction time and vs. B_0. What is maximal concentration of C? Try to figure out optimal parameters of the process (B_0 and the reaction time).
Modify script 34-1.py and make a 2D plot showing influence of initial oxidant concentration and the process time on concentration of undesired product D.

Results

Your plot should look like this:



6 Ideal Reactors

Example 35. Filling a batch reactor

●●○

Initially a reactor contains $V_0 = 2\,\mathrm{m}^3$ of a solvent. The reactor is filled by solution of substrate A at a concentration $C_{A0} = 2\,\mathrm{kmol\,m}^{-3}$ at a flow rate of $F_{in} = 0.06\,\mathrm{m}^3\,\mathrm{min}^{-1}$ for 30 min. The rate equation is $r_a = kC_A$, where $k = 0.25\,\mathrm{min}^{-1}$. Plot concentration of A versus filling time.
The example was adopted from [1]. See also example 13.

Calculations

This example demonstrates how to manage with kinetics problems at non constant volume. In the script two approaches to solve the problem are implemented: Model1 – one ODE with one unknown (eq. 30) and Model2 – system of ODE's with $V(t)$ and $n_A(t)$ as unknowns (see the script below).
Here is a mass balance for the system:

$$\begin{gathered}\text{Accumulation} = \text{In} - \text{Out} - \text{Rate of consumption} \\ \frac{dn_A}{dt} = \frac{d(C_A V)}{dt} = F_{in}C_{A0} - 0 - kC_A V\end{gathered} \tag{27}$$

After differentiation we get:

$$V\frac{dC_A}{dt} + C_A\frac{dV}{dt} = F_{in}C_{A0} - kC_A V \tag{28}$$

We know that:

$$\begin{gathered}V(t) = V_0 + F_{in}t \quad \text{and} \\ \frac{dV(t)}{dt} = F_{in}\end{gathered} \tag{29}$$

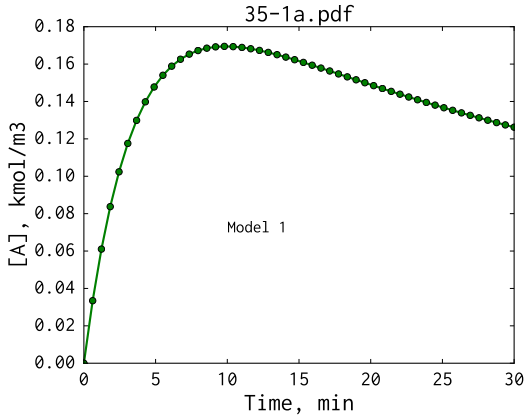
After substitution of eq. 29 into eq. 28, rearranging and substitution by numeric values we get:

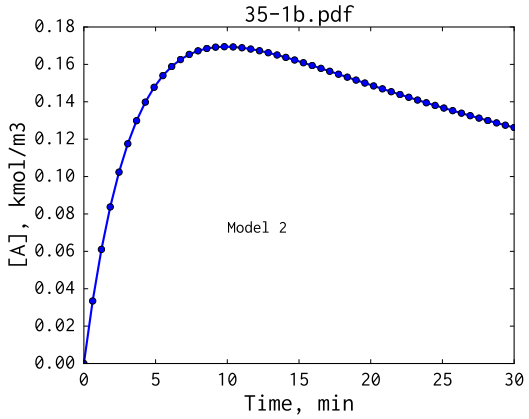
$$\frac{dC_A}{dt} = \frac{0.12 - (0.56 + 0.015t)C_A}{2 + 0.06t} \tag{30}$$

```
# File 35-1.py [View]
# Example 35. Filling a batch reactor
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

def model1(y,t):
        A = y[0]
        dAdt = (0.12-(0.56 + 0.015*t)*A)/(2+0.06*t)
        return [dAdt]

A0 = 0
t = np.linspace(0,30)
results = odeint(model1, [A0], t)
A = results[:,0]
plt.plot(t,A,'go-')
plt.title('35-1a.pdf')
plt.xlabel('Time, min')
```

```
plt.ylabel('[A], kmol/m3')
plt.text(10, 0.07, 'Model 1')
plt.savefig('35-1a.pdf')
plt.close()


def model2(y,t):
    V = y[0]                            # instant total volume
    nA = y[1]                           # instant mole numbers
    cA = nA/V                           # instant concentration, kmol/m3
    dVdt = 0.06                         # rate of change of the volume
    dnAdt = 0.06*2 - k*cA*V             # mole balance: flow_in - consumed_in_rxn
    return [dVdt, dnAdt]                # note the rate definition: V dN/dt = r

Fin = 0.06                              # m3/min
k = 0.25                                # 1/min
nA0 = 0                                 #
V0 = 2                                  # m3
t = np.linspace(0,30) #
results = odeint(model2, [V0, nA0], t)
V = results[:,0]                        # V(t)
nA = results[:,1]                       # nA(t)
cA = nA/V                               # calculation of the concentration (!)
plt.plot(t,cA,'bo-')
plt.title('35-1b.pdf')
plt.xlabel('Time, min')
plt.ylabel('[A], kmol/m3')
plt.text(10, 0.07, 'Model 2')
plt.savefig('35-1b.pdf')
plt.close()
```

Results





Exercises

Plot n_A versus filling time. Compare the created plot with the concentration curves above.

Try varying rate of feeding flow to see what happens.

Simulate the system if the rate equation is: 1) $r_A = kC_A^{0.5}$, 2) $r_A = kC_A^2$.

[1] Stanley M. Walas. *Chemical Reaction Engineering Handbook of Solved Problems*. CRC Press, 1995.

Example 36. Batch polymerization

●●○

Consider a bulk (no solvent) free-radical polymerization process of a monomer (M) to a polymer (P):

$$M \xrightarrow{k} P$$

Suppose that the rate equation is:

$$\frac{d[M]}{dt} = k[M]^{1.3}$$

and densities of monomer and polymer are $\rho_M = 0.9\,\mathrm{g\,mL^{-1}}$ and $\rho_P = 1.0\,\mathrm{g\,mL^{-1}}$, respectively. Plot the monomer concentration and total volume versus time from 0 to 3 h assuming that density of reaction mixture is a linear function of the monomer conversion α:

$$\rho = \rho_M + \alpha(\rho_P - \rho_M)$$

$k = 1.44\,\mathrm{L^{0.3}mol^{-0.3}h^{-1}}$, initial amount of the monomer is $n_{M0} = 8\,\mathrm{mol}$ and its molar weight is $M_M = 104\,\mathrm{mol\,L^{-1}}$.

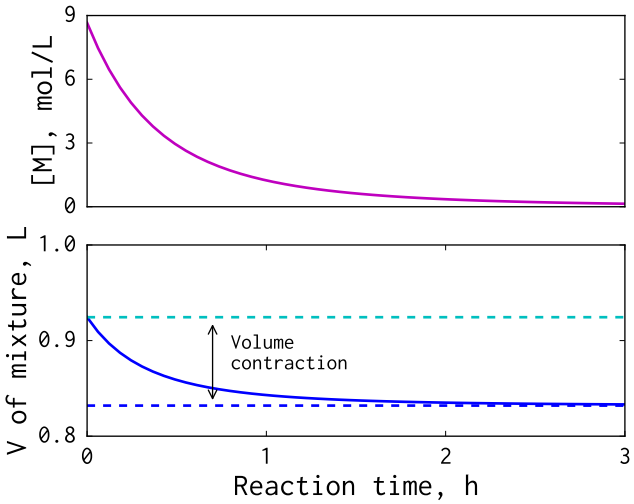
Calculations

This example shows how to model kinetics at non constant density. As in example 35, we are going to use in a model number of moles instead of a molar concentration. The calculations are as follows: at each integration step (`odeint()` call) compute instantaneous values of: conversion → density → volume of the reaction mixture → monomer concentration → and finally the derivative. For clarity calculations of the monomer conversion and the volume were implemented as separated `Python` functions.

```
# File 36-1.py [View]
# Example 36. Batch polymerization
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Stoichiometry:
# M -> P, k
# Empirical rate expresssion:
# -dM/dt = k*M**1.3
# rho = f (alpha): ro = 0.9 + alpha*0.1

# calculate conversion of the monomer (M)
def calcAlpha(nM):
    return (nM0-nM)/nM0

# calculate total volume of reaction mixture,
# based on initial mass and conversion of the monomer (M)
# and 'empirical' dependence of solution density
# as a function of the solution composition rho=f(alpha)
def calcV(nM):
    alpha = calcAlpha(nM)
    ro = roM + (roP-roM)*alpha
    V = m0 / ro
    return V/1000.0             # to have volume in L
```

```
def model(y,t):                 # kinetic model
    nM = y[0]
                                # calculate instant values
    V = calcV(nM)               # volume of the mixture
    M = nM/V                    # molar concentration of M (substrate)

    dnMdt = -k*M**1.3*V         # calculate & return derivatives
    return [dnMdt]

# The model parameters
k = 4e-4*3600                   # polymerization rate coefficient
roM = 0.9                       # density of pure M
roP = 1.0                       # density of pure P
nM0 = 8                         # initial amount of M (mole)
nP0 = 0                         # initial amount of P (mole)
MM = 104                        # molecular weight of M g/mol
m0 = nM0 * MM                   # initial mass of the monomer (g)

ic = [nM0]                      # initial conditions
t = np.linspace(0,3)
results = odeint(model, ic, t)

# post-processing of results
nM = results[:,0]               # nM(t)
V = calcV(nM)                   # V(t)
M = nM/V                        # calculate molar concentrations of M
alpha = calcAlpha(nM)
Vlim100 = calcV(0)              # V at 100% conversion, just to plot on graph
Vlim0 = calcV(8)                # V at 0% conversion, just to plot on graph

ax1 = plt.subplot(211)
ax1.set_xlim(0,3)
ax1.set_ylim(0,9)
ax1.set_xticks([])
ax1.set_yticks([0.0,3.0,6.0,9.0])
ax1.set_ylabel('[M], mol/L')
ax1.plot(t,M,'m-')
# plot V(t)
ax2 = plt.subplot(212)
ax2.set_xlim(0,3)
ax2.set_ylim(0.8,1)
ax2.set_xticks([0,1,2,3])
ax2.set_xlabel('Reaction time, h')
ax2.set_yticks([0.8, 0.9 , 1.0])
ax2.set_ylabel('V of mixture, L')
ax2.plot(t,V,'b-')
ax2.plot((0,3),(Vlim100, Vlim100),'b--') # just to make the graph nicer
ax2.plot((0,3),(Vlim0, Vlim0),'c--')     #
ax2.annotate("", xy=(0.7,0.835), xytext=(0.7,0.92), arrowprops=dict(arrowstyle='<->'))
ax2.text(0.8, 0.87, 'Volume\ncontraction')
plt.savefig('36-1.pdf')
plt.close()
```

Results



Exercises

Plot density of the system as function of the conversion from 0 to 1.

Modify script to simulate simply constant density system. Compare the results

Example 37. Reaching steady state in CSTR

●●○

Consider a second-order, liquid-phase reaction in an 10 L isothermal CSTR:

$$2A \xrightarrow{k} B$$

The rate equation is: $r = 2k[A]^2$, $k = 0.1\,\mathrm{L\,mol^{-1}\,min^{-1}}$ and the flow rate $F = 0.1\,\mathrm{L\,min^{-1}}$. Plot the concentration of A versus time from 0 to 30 min for constant feed concentration $[A]_f = 2\,\mathrm{mol\,L^{-1}}$ if:

1) the reactor is initially filled with a solvent, thus $[A]_0 = 0\,\mathrm{mol\,L^{-1}}$,

2) the reactor is initially filled with feed, so $[A]_0 = 2\,\mathrm{mol\,L^{-1}}$.

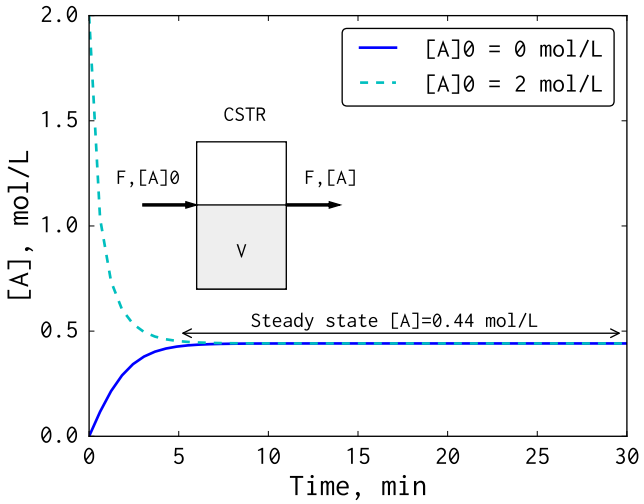
This example was adopted from [1]

Calculations

```
# File 37-1.py [View]
# Example 37. Reaching steady state in CSTR
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

from __future__ import division
import matplotlib.pyplot as plt
import matplotlib.patches as p
import numpy as np
from scipy.integrate import odeint

# Reaction: 2A -> B, k
# Reaching steady state in a CSTR
def model(y, t):
    A = y[0]
    r = 2*k*A**2
    dAdt = F*Af - F*A - r*V
    return [dAdt]

V = 10    # L
F = 0.1   # L/min
k = 0.04   # 1/min
Af = 2    # mol/L

# simulation time from 0 to 30 min
t = np.linspace(0, 30)

# case 1. Initially reactor filled with an inert
A0 = 0
results = odeint(model, [A0], t)
A1 = results[:,0]

# case 2. Initially reactor filled with feed
A0 = 2 # mol/L
results = odeint(model,[A0], t)
A2 = results[:,0]
Afinal = A1[-1]
txt = "Steady state [A]={:.2f} mol/L".format(Afinal)

# Plot
ax = plt.subplot()
```

```
ax.plot(t, A1, 'b-', label='[A]0 = 0 mol/L')
ax.plot(t, A2, 'c--', label='[A]0 = 2 mol/L')
ax.set_xlabel('Time, min')
ax.set_ylabel('[A], mol/L')
ax.annotate('', xy=(5,0.49), xytext=(29.8,0.49), arrowprops=dict(arrowstyle='<->'))
ax.text(9, 0.52,txt)
ax.legend()

# Reactor scheme
ax.add_patch(p.Arrow(3,1.1,3,0, width=0.05, facecolor='black'))
ax.add_patch(p.Arrow(11,1.1,3,0, width=0.05, facecolor='black'))
ax.add_patch(p.Rectangle((6,0.7),5,0.7,fill=False))
ax.add_patch(p.Rectangle((6,0.7),5,0.4,facecolor='#eeeeee'))
ax.text(7.5,1.5,'CSTR')
ax.text(8.2,0.85,'V')
ax.text(1.5,1.2,'F,[A]0')
ax.text(12,1.2,'F,[A]')
plt.savefig('37-1.pdf')
plt.close()
```

Results



Exercises

Plot concentration of B versus time. Investigate influence of the reactor volume and the rate constant on the equilibrating time.

[1] J.B. Rawlings and J. G. Ekerdt. *Chemical Reactor Analysis and Design Fundamentals*. Nob Hill Publishing, 2002.

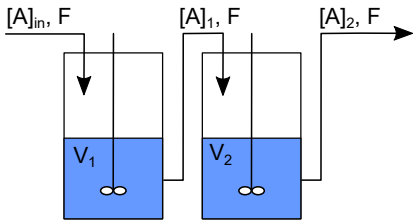
Example 38. CSTR in series

●●○

Consider a first-order reaction in a cascade of two CSTR's of equal volume:

$$A \xrightarrow{k} 2B$$

Pure A enters at a flow rate of $0.1\,\mathrm{L\,min^{-1}}$ and at concentration of $2\,\mathrm{mol\,L^{-1}}$. Assuming that $V_1 = V_2 = 3\,\mathrm{L}$, create a model of the system start-up if reactors are filled initially with the feed and the rate constant k is $0.1\,\mathrm{min^{-1}}$. Plot concentration of A versus process time in each of reactors. What is concentration of A at steady-state?



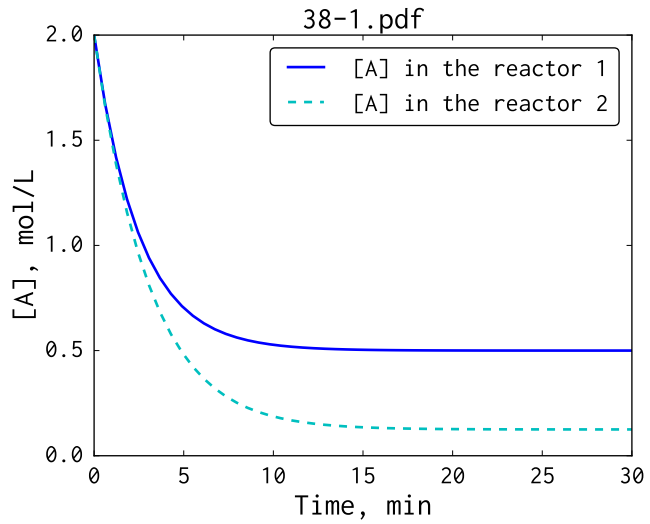
Calculations

Have a look into example 37. Create a differential mass balance for each of reactors.

```
# File 38-1.py [View]
# Example 38. CSTR in series
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Reaction: A -> B, k
# IN -> CSTR1 -> CSTR2 -> OUT
def model(y, t):
    A1 = y[0]
    A2 = y[1]
    dA1dt = F*Ain - F*A1 - V*k*A1
    dA2dt = F*A1 - F*A2 - V*k*A2
    return [dA1dt, dA2dt]

V = 3    # L
F = 0.1   # L/min
k = 0.1   # 1/min
Ain = 2    # mol/L

# simulation time from 0 to 30 min
t = np.linspace(0, 30)
results = odeint(model,[Ain, Ain], t)
A1 = results[:,0]
A2 = results[:,1]
plt.plot(t, A1, 'b-', label='[A] in the reactor 1')
plt.plot(t, A2, 'c--', label='[A] in the reactor 2')
plt.xlabel('Time, min')
plt.ylabel('[A], mol/L')
```

```
plt.legend()
plt.title('38-1.pdf')
plt.savefig('38-1.pdf')
plt.close()
```

Results



Exercises

1) Create model of the system if:

$V_1 = 9\,\mathrm{L}$ and $V_2 = 1\,\mathrm{L}$;

$V_1 = 1\,\mathrm{L}$ and $V_2 = 9\,\mathrm{L}$;

2) Write a script to simulate series of three CSTR reactors

3) Based on kinetic model from example 27, plot concentrations of A, B and C versus start-up time.

Example 39. Isothermal semi-batch reactor

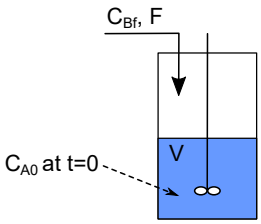
●●○

A catalytic reaction takes place in the liquid phase in an isothermal semibatch reactor:

$$A \xrightarrow{B} C$$

The rate of consumption of the substrate $r_A = k[A][B]$, $k = 0.25\,\mathrm{L\,mol^{-1}\,min^{-1}}$. Initially the reactor contains 2700 L of a solution of A at concentration of 20 mol $\mathrm{L^{-1}}$. The concentration of catalyst B in the reactor initially is zero. Starting at time = 0, a solution containing 0.05 mol $\mathrm{L^{-1}}$ of B is feed to the reactor at flow rate of 12.5 L $\mathrm{min^{-1}}$. Plot concentrations of A, B and C as well as total volume versus time from 0 to 200 min. How many moles of C are in the reactor after 200 min?

The problem was taken from LearnChemE material: https://www.youtube.com/watch?v=KzMRcACcvsA



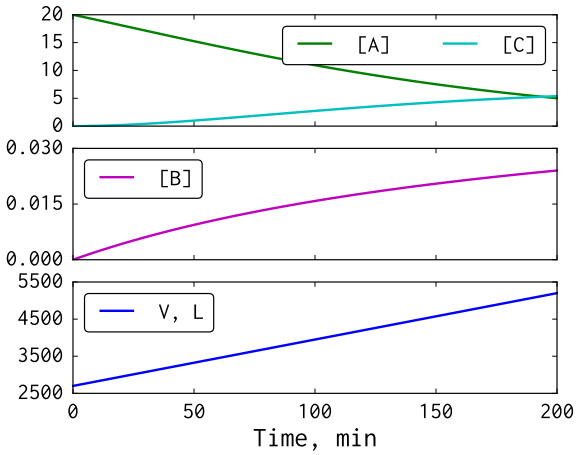
Calculations

Based on this example you can see how easy is to translate a `Polymath` code into an equivalent `Python` script.

```
# File 39-1.py [View]
# Example 39. Isothermal semi-batch reactor
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

# This code is based on LearnChemE example:
# https://www.youtube.com/watch?v=KzMRcACcvsA

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# A -> C rA = -k*CA*CB
def model(y,t):
    # unpack instantaneous values
    nA = y[0]
    nB = y[1]
    nC = y[2]
    V = y[3]
    # calculate instant concentrations
    A=nA/V
    B=nB/V
    C=nC/V
    # derivative mole balances
    dnAdt = -k*A*B*V
    dnBdt = FB                          # molar flow rate (mol/min) of B
    dnCdt = k*A*B*V                     #
    dVdt = v                            # flow rate of the catalyst addition
    return [dnAdt, dnBdt, dnCdt, dVdt]
```

```
nA0 = 54000                             # moles
nB0 = 0
nC0 = 0
V0 = 2700                               # L
k=0.25                                  # L/(mol min)
FB= 0.05 * 12.5                         # mol/min
v=12.5                                  # L/min

ic = [nA0, nB0, nC0, V0]                # initial conditions
t = np.linspace(0,200)                  # 0 - 200 min
results = odeint(model, ic, t)

# results post-processing
nA = results[:,0]
nB = results[:,1]
nC = results[:,2]
V = results[:,3]
A = nA/V
B = nB/V
C = nC/V
print "Final number of moles of C (after 200min) = {:.0f} mole".format(nC[-1])
# -1 means the last element == 200min
print "Final concentration of C (after 200min) = {:.0f} mol/L".format(C[-1])

plt.subplot(311)
plt.plot(t,A, 'g-', label='[A]')
plt.plot(t,C, 'c-', label='[C]')
plt.xticks([0, 50, 100, 150, 200],[])
plt.legend(loc='upper right', ncol=2)

plt.subplot(312)
plt.plot(t,B, 'm-', label='[B]')
plt.xticks([0, 50, 100, 150, 200],[])
plt.yticks([0, 0.015, 0.030])
plt.legend(loc='upper left')

plt.subplot(313)
plt.plot(t,V, 'b-', label='V, L')
plt.xlabel('Time, min')
plt.yticks([2500, 3500, 4500, 5500])
plt.legend(loc='upper left')
plt.savefig('39-1.pdf')
plt.close()
```

Results




```
%run 39-1.py
Final number of moles of C (after 200min) = 27987 mole
Final concentration of C (after 200min) = 5 mol/L
```

Exercises

1) Print in a table $A(t)$, $B(t)$, $C(t)$ and $V(t)$.

2) How many moles of the catalyst B is required to carry out the process?

3) Calculate instantaneous mass balance in term of concentrations $[A] + [C]$ and numbers of moles $n_A + n_B$. What do you observe? Why?

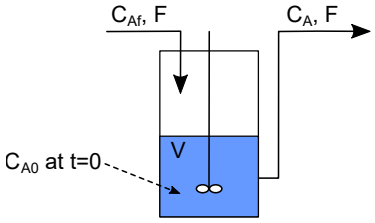
Example 40. Multiple steady states

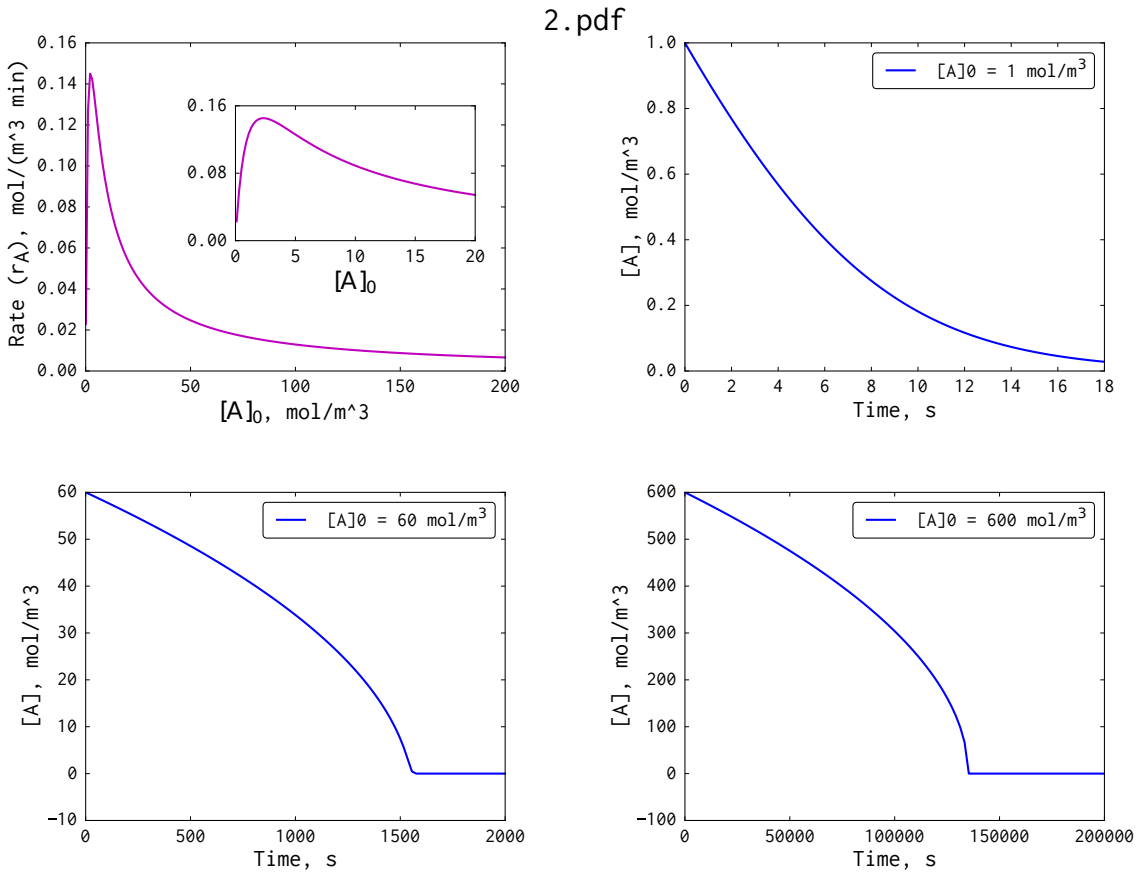
●●○

Consider a complex reaction: A → Product in a CSTR with constant V and F. The rate equation for this reaction is:

$$r_A = \frac{k_1 C_A}{(1 + k_2 C_A)^2} \tag{31}$$

$k_1 = 0.25\,\mathrm{min}^{-1}$, $k_2 = 0.43\,\mathrm{m}^3\,\mathrm{mol}^{-1}$. Simulate reaching a steady-state assuming that $V = 60\,\mathrm{m}^3$, $F = 0.07\,\mathrm{m}^3\,\mathrm{min}^{-1}$ and concentration of A in the feed is $69\,\mathrm{mol\,m}^{-3}$. The reactor is filled initially with the solution of A at concentration of A_0. Run simulation for various values of A_0 from 0 to $70\,\mathrm{mol\,m}^{-3}$. This example was adopted from [1].





Calculations

Rate equation of the form eq. 31 can be seen with some reactions catalyzed by solid catalysts. Additionally, eq. 31 can be used to model inhibition by substrate in many biochemical systems (enzymatic reactions, biomass growth).

Analyze shape of the rate curve (fig. 2.pdf). Note that for higher concentrations of the substrate A, the rate is smaller than for more diluted solutions of A.

Before you begin analysis of dynamics of the system see example 37. Please remember that mass balance for component A at steady-state can be written as:

$$FC_{Af} - FC_A - r_A V = 0 \tag{32}$$

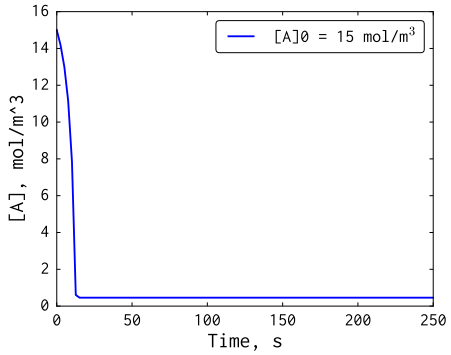
and it is an algebraic equation. For the considered reaction kinetics (eq. 31) and for particular values of F and V the equation 32 has three mathematical and physical solutions.

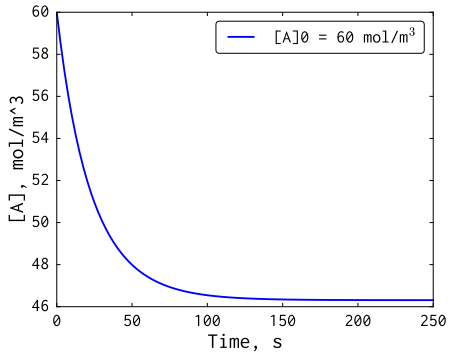
```
# File 40-1.py [View]
# Example 40. Multiple steady states
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve

# rate of substrate A consumption
def rA(A):
    k1 = 0.25 # 1/min
    k2 = 0.43 # m^3/mol
    return k1*A/(1 + k2*A)**2

# mass balance: in - out
def net(A):
    return F*(Af - A)

def steadystate(y):                      # mole balance for a steady-state
    A = y[0]
    eq1 = net(A) - rA(A)*V               # mole balance of A
    return [eq1]

def model(y, t):                         # mole balance of A for an unsteady-state
    A = y[0]
    dAdt = net(A) - rA(A)*V
    return [dAdt]

V = 60 # m^3
F = 70e-3 # m^3/min
# [A] in the feed
Af = 69 # mol/m^3


x1, = fsolve(steadystate,[1])            #guess values: 1, 20, 50
x2, = fsolve(steadystate,[20])
x3, = fsolve(steadystate,[50])
print x1, x2, x3

t = np.linspace(0,250,100)

# [A] in the reactor at t=0
```

```
A0 = 15 # mol/m^3
results = odeint(model, [A0], t)
plt.plot(t, results[:,0], label='[A]0 = 15 mol/m$^3$')
plt.xlabel('Time, s')
plt.ylabel('[A], mol/m^3')
plt.legend()
plt.savefig('40-1a.pdf')
plt.close()

# [A] in the reactor at t=0
A0 = 60 # mol/m^3
results = odeint(model, [A0], t)
plt.plot(t, results[:,0], label='[A]0 = 60 mol/m$^3$')
plt.xlabel('Time, s')
plt.ylabel('[A], mol/m^3')
plt.legend()
plt.savefig('40-1b.pdf')
plt.close()

A = np.linspace(0,70,200)
plt.plot(A, net(A), 'g-', label='net flow rate A')
plt.plot(A, rA(A)*V, 'y-', label='A reacted')
plt.xlabel('[A], mol/m^3')
plt.ylabel('Rate, mol/(min m^3)')
plt.ylim(0,10)
plt.plot(x1,0,'rD')
plt.plot(x2,0,'rD')
plt.plot(x3,0,'rD')
plt.legend()
plt.savefig('40-1c.pdf')
plt.close()
```

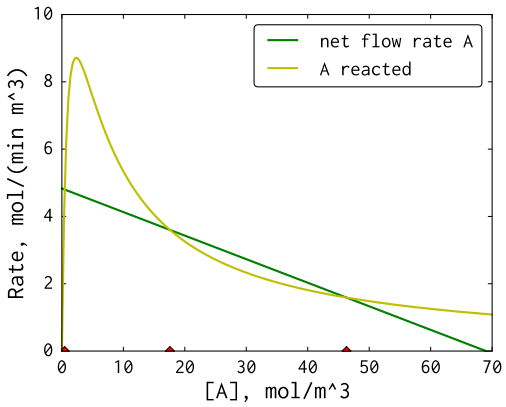
Results

Here are results of time evaluation of the substrate concentration for various initial concentration of A in the reactor (A_0).





Graphical solution of the steady-state equation (eq. 31:



Concentrations of A under multiple steady-states are:

```
%run 40-1.py
0.458378152275 17.5793679715 46.3110910856
```

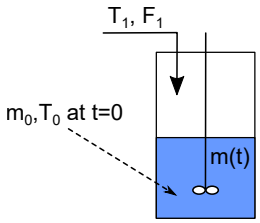
[1] R. K. Herz. *Chemical Reaction Engineering*. URL: http://www.reactorlab.net.

7 Thermal effects

Example 41. Heat balance - filling a tank

●●○

A tank is initially filled with 20 kg of water at 20 °C. Starting at time = 0 a stream of water at 60 °C is feed to the tank at flow rate of 0.1 $m^3\,h^{-1}$. Plot mass of water in the tank and temperature of the liquid versus filling time from 0 to 1 h assuming adiabatic conditions and constant density of water of 1000 $kg\,m^{-3}$.



Calculations

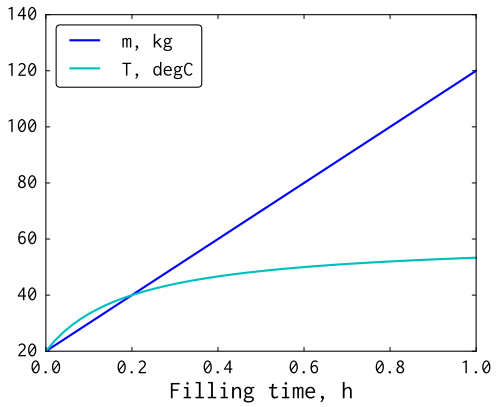
As an introduction please watch: https://www.youtube.com/watch?v=WThXhZ4mBz8 and see example 13. In this example we are going to solve mass and heat balance in one ODE system.

```
# File 41-1.py [View]
# Example 41. Heat balance - filling a tank
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

def model(y, t):
    m = y[0]                    # mass at instant time
    T = y[1]                    # temperature at instant time
    dmdt = F1*rho               # mass balance

                                # See Newton's law of cooling
    dTdt = F1*rho*(T1-T)/m      # T1-T is time-dependent thermal gradient
                                # between hot stream and water inside the tank.
                                # Heat capacity of the inlet steam is equal
                                # to h.c. of the mixture in the tank, therefore is shortened
    return [dmdt, dTdt]

# hot stream
F1 = 0.1        # m3/h
rho = 1000      # kg/m3
T1 = 60         #degC
# initial conditions
m0 = 20 # kg
T0 = 20 # degC

t = np.linspace(0, 1) # 0-1h
results = odeint(model, [m0, T0], t)

m = results[:,0]
```

```
T = results[:,1]
plt.plot(t,m,label='m, kg')
plt.plot(t,T,label='T, degC')
plt.xlabel('Filling time, h')
plt.legend(loc='best')
plt.savefig('41-1.pdf')
plt.close()

# checking the calculations
cp = 4.18    # kJ/(kg K), average mass-specific heat capacity of water
Te = T[-1]   # final temperature of water in the tank
me = m[-1]   # final mass of water in the tank
m1 = me-m0   # mass of water feeded
print "Checking ..."
print "Q initial (J) + Q introduced (J)"
print cp*m0*(T0) + cp*m1*(T1)
print "Q total (J)"
print cp*me*Te
```

Results

```
%run 41-1.py
Checking ...
Q initial (J) + Q introduced (J)
26752.0
Q total (J)
26752.0007807
```



Exercises

1) Compute $T(t)$ and $m(t)$ when the temperature of the inlet stream is 4 °C.

2) Assume that a second flow of water is fed to the tank ($F_2 = 0.05\,\mathrm{m^3\,h^{-1}}, T_2 = 90\,°\mathrm{C}$). Write a program to model the system.

Example 42. Heat generation

●●○

Consider a first-order exothermic reaction $A \rightarrow B$. The rate equation is $r_a = kA$, $k = 0.65\,\mathrm{min}^{-1}$, and enthalpy of the reaction is $125.574\,\mathrm{kJ\,mol}^{-1}$.

Plot conversion of A and rate of A consumption versus reaction time if initial concentration of A is $2\,\mathrm{mol\,L}^{-1}$. Plot total heat released and instantaneous heat release rate versus time from 0 to 5 min knowing that total volume of the reaction mixture is 10 L and assuming adiabatic conditions of the process.

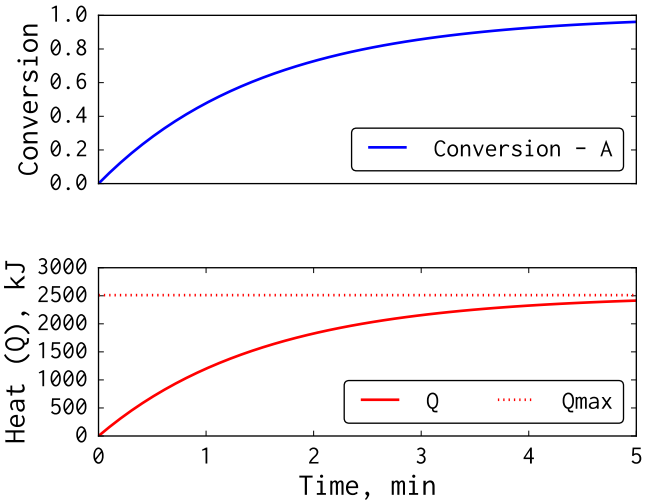
At what time does the heat release reach maximum? What is the total amount of heat released? Do you see any link between reaction rate and rate of heat release/consumption?

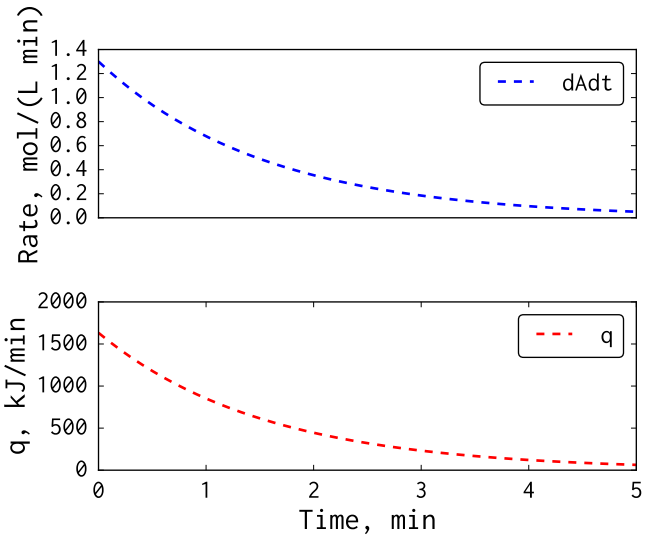
Calculations

```
# File 42-1.py [View]
# Example 42. Heat generation
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# kinetic model
def rA(A):
    k = 0.65 # 1/min
    return -k*A

def model(y, t):
    A = y[0]
    Q = y[1]
    dAdt = rA(A)           # component A balance
    dQdt = dAdt*V0*H       # instantaneous heat release rate
    return [dAdt, dQdt]

A0 = 2.0 # mol/L
H = -125.574 #kJ/mol
V0 = 10 # L

ic = [A0, 0]               # initial conditions: A0, Q0
Qmax = -A0*V0*H            # <---
t = np.linspace(0,5)       # 0 - 5 min
results = odeint(model, ic, t)
A = results[:,0]
Q = results[:,1]

alpha = (A0-A)/A0          # calculate conversion of A
dAdt = rA(A)               # rate of reaction
q = dAdt*V0*H              # instantaneous heat release rate (kJ/min)

plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9)
plt.subplot(211)
plt.plot(t, alpha, label='Conversion - A')
plt.legend(loc='best')
plt.xticks([])
plt.ylabel('Conversion')
plt.subplot(212)
```

```
plt.plot(t, Q, 'r-', label='Q')
plt.plot((0,5),(Qmax, Qmax), 'r:', label='Qmax')
plt.legend(loc='best', ncol=2)
plt.xlabel('Time, min')
plt.ylabel('Heat (Q), kJ')
plt.savefig('42-1a.pdf')
plt.close()

plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9)
plt.subplot(211)
plt.plot(t, -dAdt, 'b--', label='dAdt')
plt.legend(loc='best')
plt.xticks([])
plt.ylabel('Rate, mol/(L min)')
plt.subplot(212)
plt.plot(t, q, 'r--', label='q')
plt.legend(loc='best')
plt.xlabel('Time, min')
plt.ylabel('q, kJ/min')
plt.ylim(0,2000)
plt.savefig('42-1b.pdf')
plt.close()
```

Results





Exercises

Investigate influence of the reaction kinetics (mechanism) on the dynamics of heat generation. Keeping $[A]_0 = 2\,\mathrm{mol\,L^{-1}}$ and $k = 0.65\,\mathrm{min^{-1}}$, repeat simulations for:

1) $r_A = k[A]^{1.5}$

2) $r_A = k[A]^2$

3) $r_A = k$

Example 43. Non isothermal kinetics

●●○

Consider a first-order reaction $A \rightarrow B$. The rate equation is $r_a = kA$, $k = k_0 \exp(-E_a/RT)$ $k_0 = 1 \times 10^{11}\,\mathrm{L\,mol^{-1}\,s^{-1}}$, $E_a = 78\,\mathrm{kJ\,mol^{-1}}$. Plot conversion of A versus reaction time from 0 to 300 s if:

1) the process is isothermal $T = 310\,\mathrm{K}$.

2) reaction takes place under constant heating rate $r_T = 0.066\,\mathrm{K\,s^{-1}}$, initially $T = 310\,\mathrm{K}$.

Calculations

The following set of two ODEs forms model of the non-isothermal process:

$$\begin{aligned} \frac{dT}{dt} &= r_T \\ \frac{dA}{dt} &= k_0 \exp(-E_a/RT)A \end{aligned} \tag{33}$$

with initial conditions:

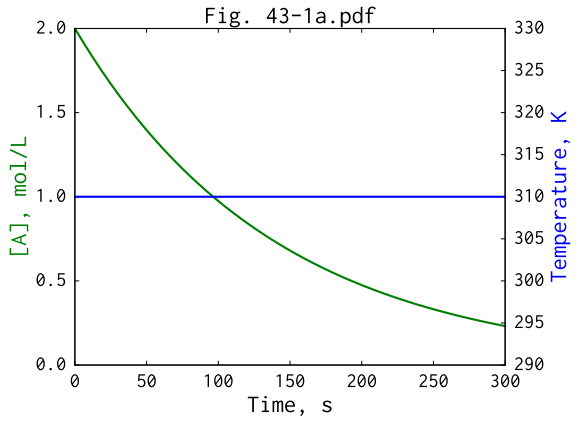
$$\mathrm{T(t=0)=310,\ A(t=0)=2} \tag{34}$$

In the script we re going to solve the initial value problem (eq. 33 and eq. 34) numerically.

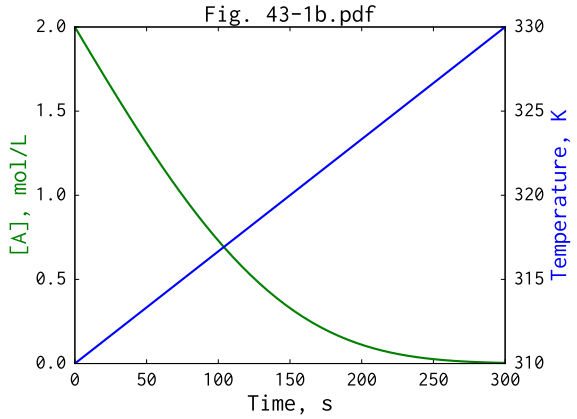
```
# File 43-1.py [View]
# Example 43. Non isothermal kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

def model(y, t):
   A = y[0]
   T = y[1]
   k = k0*np.exp(-Ea/(R*T))  # calculate instant k at T
   dAdt = -k*A               # A -> B ; da/dt = ka; kinetic model
   dTdt = rT                 # heating/cooling rate
   return [dAdt, dTdt]

# The model parameters
R = 8.314e-3                 # kJ/mol*K
# Activation energy
Ea = 78                      # kJ/mol
# Preexponential factor
k0 = 10**11.0                # L/mol*s
# Calculation time: 0 - 300s
t_range = np.linspace(0, 300)

# Initial conditions
A0 = 2          # mol/L
T0 = 310        # K
ic = [A0, T0]   # initial conditions

# Case 1
# dT/dt = 0 isothermal
rT = 0
results = odeint(model, ic, t_range)
A1 = results[:,0]
```

```
T1 = results[:,1]

# Case 2
# dT/dt = 0.066 deg/s linear heating
rT = 20/300.0
results = odeint(model, ic, t_range)
A2 = results[:,0]
T2 = results[:,1]

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(t_range, A1, 'g-')
ax2.plot(t_range, T1, 'b-')
ax1.set_ylim(0,2)
ax1.set_xlabel('Time, s')
ax1.set_ylabel('[A], mol/L', color='g')
ax2.set_ylabel('Temperature, K', color='b')
plt.title('Fig. 43-1a.pdf')
plt.savefig('43-1a.pdf')
plt.close()

fig, ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot(t_range, A2, 'g-')
ax2.plot(t_range, T2, 'b-')
ax1.set_xlabel('Time, s')
ax1.set_ylabel('[A], mol/L', color='g')
ax2.set_ylabel('Temperature, K', color='b')
plt.title('Fig. 43-1b.pdf')
plt.savefig('43-1b.pdf')
plt.close()
```

Here is result of simulation of the isothermal process:



Here is result of simulation of the non-isothermal process:

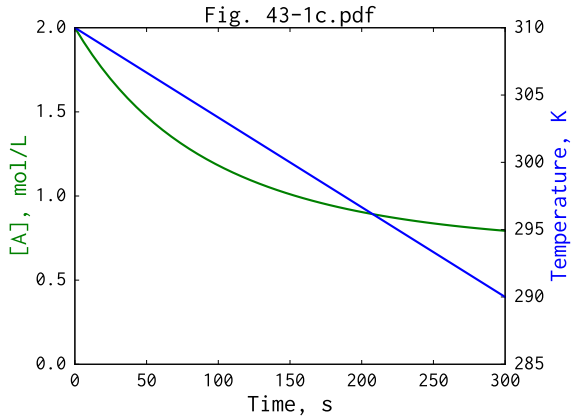


Exercises

Calculate $A(t)$ and $T(t)$ under constant cooling rate of $0.066\,\mathrm{K\,s^{-1}}$.

Results

Your plot should look like this:



Example 44. Adiabatic batch reactor

●●○

The liquid phase reaction $A \rightarrow B$ takes place in an 100-L volume adiabatic batch reactor. Initially the reactor contains only A ($C_{A0} = 5.0\,\mathrm{mol\,L^{-1}}$). The heat of the reaction is $-14.6\,\mathrm{kJ\,mol}$ and is independent of temperature. Heat capacity of reaction mixture is $83.6\,\mathrm{J\,mol^{-1}\,K^{-1}}$ and does not depend on the conversion. The reaction rate has form:

$$r_A = -kC_A^{2.5}, \quad k = 0.0025 \exp\left(-1760/T\right)$$

Plot conversion of A and reactor temperature versus reaction time from 0 to 4000 s assuming initial temperature of 373 K. What is the reactor temperature when 60% of A has reacted? How long does it take to reach 60% conversion of A?

This problem was taken from LearnChemE material: http://www.youtube.com/watch?v=--cZwa2TXhY. The example is Python implementation of the Polymath code.

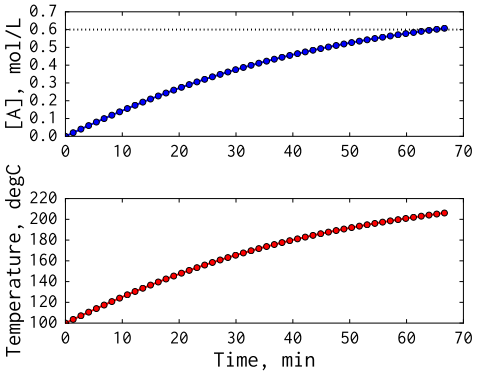
Calculations

Compare analytic solution to the numerical method.

```
# File 44-1.py [View]
# Example 44. Adiabatic batch reactor
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# calculate the rate at an instant of concentration (A) and temperature (T)
def rA(A, T):
    k = 0.0025*np.exp(-1760/T)
    return k*A**2.5

# ODEs
def model(y, t):
    A = y[0]                          # instant concentration of A
    T = y[1]                          # instant temperature
    dAdt = -rA(A, T)                  # calculate the rate at time t and at temperature T
    dTdt = -rA(A, T)*delH / (A0*cpA)  # rate of change of temperature (from heat balance)
    return [dAdt, dTdt]

delH = -14600  # kJ/mol
cpA = 83.6     # J/(mol K)
A0 = 5.0       # mol/L
T0 = 373       # K

t = np.linspace(0, 4000)              # 0-4000s
results = odeint(model, [A0, T0], t)
A = results[:,0]                      # A(t)
T = results[:,1] - 273                # extract temperature & convert it from K to degC
X = (A0-A)/A0                         # calculate conversion from A(t)

plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9) # just to make the plot more clear
plt.subplot(211)
plt.plot(t/60.0, X, 'bo-')            # t/60.0 => convert time unit from s to min
plt.plot((0, 70), (0.6, 0.6),'k:')
```

```
plt.ylabel('[A], mol/L')
plt.subplot(212)
plt.plot(t/60.0, T, 'ro-')
plt.xlabel('Time, min')
plt.ylabel('Temperature, degC')
plt.savefig('44-1.pdf')
plt.close()
```

Results

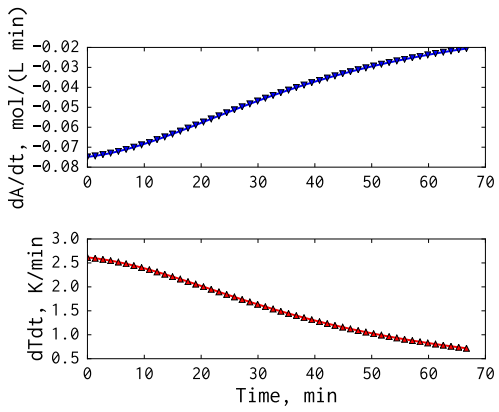


Exercises

Try varying reaction enthalpy or initial temperature to see what happens.

Modify script to plot instant values: dA/dt and dT/dt versus reaction time. Try to interpret the shape of both curves.

Results



8 Biochemical reaction systems

Example 45. Exponential and logistic growth

● ○ ○

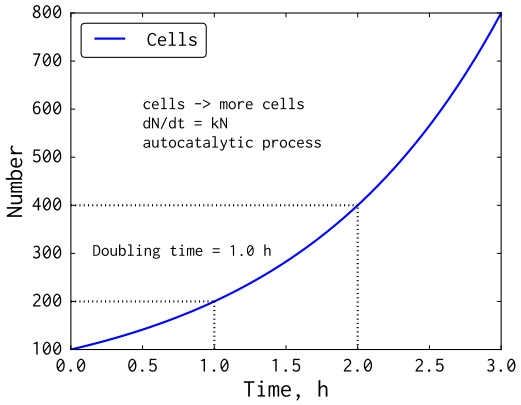
Exponential (Malthusian) Growth

The simplest mathematical model of population (P) growth is obtained by assuming that the rate of increase of the population at any time is proportional to the size of the population at that time.

$$dP/dt = kP$$

Let k to be 0.6931 and initial population of cells is 100. Plot number of cells versus time of population growth.

```
# File 45-1.py [View]
# Example 45. Exponential and logistic growth
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Exponential growth
def model(y, t):
    N = y[0]
    dNdt = k*N
    return [dNdt]

k = 0.6931 #ln(2)
N0 = 100
t = np.linspace(0, 3)
results = odeint( model, [N0], t)
N = results[:,0]
# doubling time
t2 = np.log(2)/k
str = 'Doubling time = {:.1f} h'.format(t2)

plt.plot(t,N, label='Cells')
plt.xlabel('Time, h')
plt.ylabel('Number')
plt.legend(loc='best')
plt.text(0.5,600,'cells -> more cells')
plt.text(0.5,560,'dN/dt = kN')
plt.text(0.5,520,'autocatalytic process')
plt.text(0.15,295, str)
plt.plot((0,1*t2),(2*N0,2*N0),'k:')
plt.plot((1*t2,1*t2),(N0,2*N0),'k:')
plt.plot((0,2*t2),(4*N0,4*N0),'k:')
plt.plot((2*t2,2*t2),(N0,4*N0),'k:')
plt.savefig('45-1.pdf')
plt.close()
```



Logistic Population Model

The model assume that the death rate ($D(t)$) per individual is directly proportional to the instantaneous population ($P(t)$), and that the birth ($B(t)$) rate per individual remains constant:

$$dP/dt = (B(t) - D(t))P$$

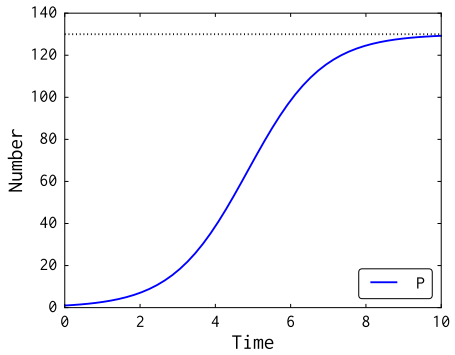
The model can be written in other form:

$$dP/dt = rP(1 - P/C)$$

where: $r = B_0$, C is carrying capacity equal to B_0/D_0. Assuming a growth rate $r = 1$, carrying capacity $C = 130$ and initial population $P_0 = 1$ simulate a population growth from 0 to 10 (undimensional time).

```
# File 45-2.py [View]
# Example 45. Exponential and logistic growth
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Logistic model
def model(y, t):
    P = y[0]
    dPdt = r*P * (1-P/C)
    return [dPdt]

# growth rate
r = 1
# carrying capacity
C = 130
# initial population
P0 = 1
# simulation time
t_min = 0
t_max = 10

t = np.linspace(t_min, t_max)
results = odeint( model, [P0], t)
P = results[:,0]
```

```
plt.plot(t,P, label='P')
plt.xlabel('Time')
plt.ylabel('Number')
plt.legend(loc='best')
plt.plot((0,10),(C,C), 'k:')
plt.savefig('45-2.pdf')
plt.close()
```

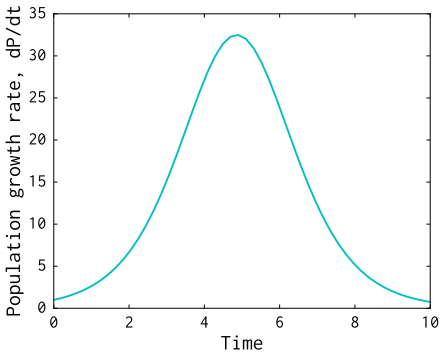


Exercises

Try varying model parameters (r and C) to see what happens.

Modify the script to plot rate of population growth versus time.

Results



See also example 30.

Example 46. Monod growth

● ○ ○

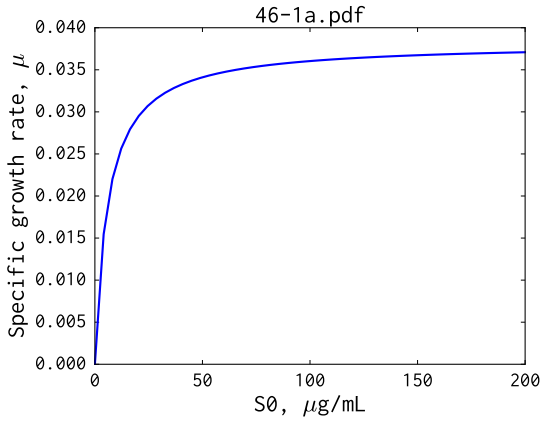
Consider Monod's bacterial growth model. Here are parameters of the model: the maximum specific growth rate of the cells $\mu_{max} = 0.0382$; the saturation constant $K_M = 6$; and the yield coefficient $Y = 0.139 \times 10^{-3}$.
Plot rate of the bacteria growth versus limiting substrate concentration in a range from 0 to 200 $\mu g\,mL^{-1}$.
Plot cells and the substrate concentration versus time from 0 to 100 h if initial concentration of bacteria X_0 is 0.002, and the initial substrate concentration S_0 is 200 $\mu g\,mL^{-1}$.
The model parameters were taken from [1].

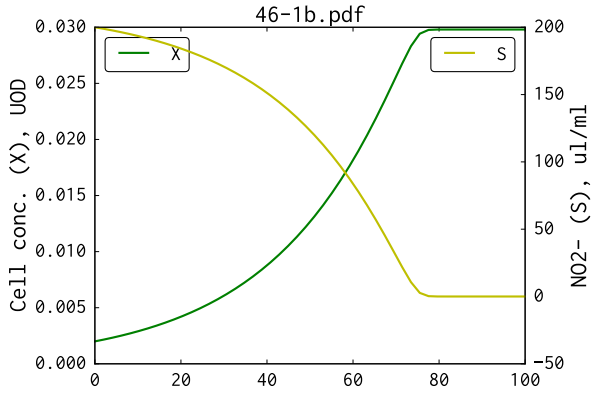
Calculations

```
# File 46-1.py [View]
# Example 46. Monod growth
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Monod growth
def model(y, t):
    X = y[0]
    S = y[1]
    mi = mi_max*S/(KM+S)     # the specific growth rate
    dXdt = X * mi            # biomass growth (e.g. nitrobacter cells)
    dSdt = -1/Y * X * mi     # limiting substrate uptake
    return [dXdt, dSdt]

mi_max = 0.0382 # the maximum specific growth rate of the cells
KM = 6 # the saturation constant
Y = 0.139e-3 # The yield coefficients

# Initial conditions
X0 = 0.002  # O.D.
S0 = 200    # nitrite ug/ml

# specific growth rate
S = np.linspace(0,200) # substrate concentration
mi = mi_max*S/(KM+S)   # the rate
plt.plot(S, mi)
plt.xlabel('S0, $\mu$g/mL')
plt.ylabel('Specific growth rate, $\mu$')
plt.title('46-1a.pdf')
plt.savefig('46-1a.pdf')
plt.close()

# Fermentation simulation
t = np.linspace(0, 100)
results = odeint( model, [X0, S0], t)
X = results[:,0]
S = results[:,1]
ax1 = plt.subplot(111)
ax1.plot(t,X, 'g', label='X')
ax1.set_ylabel('Cell conc. (X), UOD')
```

```
ax1.legend(loc='upper left')
ax2 = ax1.twinx()
ax2.plot(t,S, 'y', label='S')
ax2.set_ylabel('NO2- (S), ul/ml')
ax2.set_xlabel('Time, h')
ax2.legend(loc='upper right')
plt.title('46-1b.pdf')
plt.savefig('46-1b.pdf')
plt.close()
```

Results





Exercises

Plot rate of the cells growth (dX/dt) and the substrate uptake (dS/dt) versus time. Additionally, calculate the ratio:

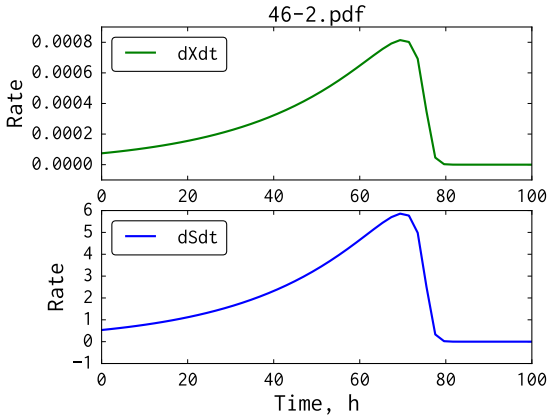
$$\frac{dX/dt}{dS/dt}$$

Compare the ratios with yield coefficient (Y) value.

Calculations

```
# File 46-2.py [View]
# Example 46. Monod growth
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Monod growth
def model(y, t):
    X = y[0]
    S = y[1]
    mi = mi_max*S/(KM+S)     # the specific growth rate
    dXdt = X * mi            # biomass growth rate
    dSdt = -1/Y * X * mi     # limiting substrate consumption rate
    return [dXdt, dSdt]

mi_max = 0.0382 # the maximum specific growth rate of the cells
KM = 6 # the saturation constant
Y = 0.139e-3 # The yield coefficients

# Initial conditions
X0 = 0.002  # O.D.
S0 = 200    # nitrite ug/ml

# Fermentation simulation
t = np.linspace(0, 100)
results = odeint( model, [X0, S0], t)
X = results[:,0]
S = results[:,1]
# Calculation the rates
dXdt = X * mi_max*S/(KM+S)
dSdt = 1/Y * X * mi_max*S/(KM+S)

ax = plt.subplot(211)
ax.set_title('46-2.pdf')
ax.plot(t, dXdt, 'g-', label='dXdt')
ax.set_xlabel('Time, h')
ax.set_ylabel('Rate')
ax.legend(loc='upper left')
ax = plt.subplot(212)
ax.plot(t, dSdt, 'b-',label='dSdt')
ax.set_xlabel('Time, h')
ax.set_ylabel('Rate')
ax.legend(loc='upper left')
plt.subplots_adjust(left=0.2, right=0.9)
plt.savefig('46-2.pdf')
plt.close()

print 'dXdt/dSdt = '
print dXdt/dSdt
print('Y = {}'.format(Y))
```

Results




```
%run 46-2.py
dXdt/dSdt =
[ 0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139  0.000139
  0.000139]
Y = 0.000139
```

[1] A Corman and A Pave. "On parameter estimation of Monod's bacterial growth model from batch culture data." In: *The Journal of General and Applied Microbiology* 29 (1983), pp. 91–101.

Example 47. Fermentation - Monod model

●●○

Develop a simple kinetic model of batch fermenter including: biomass (X), limiting substrate (S) and a product balance (P). Assume the Monod model:

$$\begin{aligned} \mu &= \mu_{max}\frac{S}{K_M + S} \\ \frac{dX}{dt} &= \mu X \\ \frac{dS}{dt} &= -\frac{1}{Y_{XS}}\mu X \\ \frac{dP}{dt} &= Y_{PX}\mu X \end{aligned} \tag{35}$$

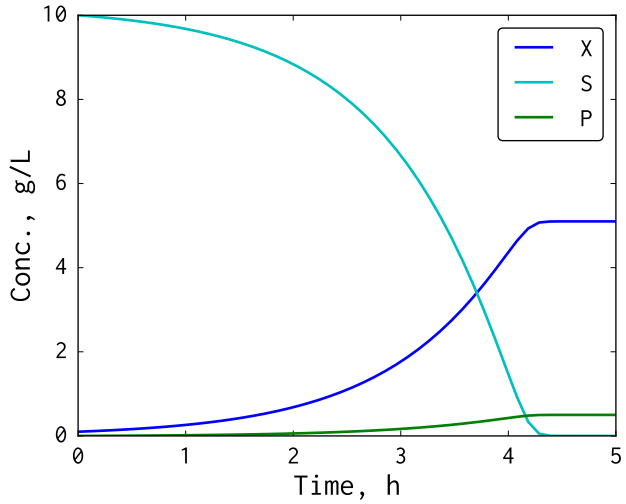
The model parameters are: $K_M = 0.4\,\mathrm{g\,L^{-1}}$, $Y_{XS} = 0.5$, $Y_{PX} = 0.1$ $\mu_{max} = 1\,\mathrm{h^{-1}}$. Plot concentrations of biomass, the substrate and the product versus fermentation time from 0 to 5 h if initially: $X_0 = 0.1\,\mathrm{g\,L^{-1}}$, $S_0 = 10\,\mathrm{g\,L^{-1}}$ and $P_0 = 0$.

Calculations

```
# File 47-1.py [View]
# Example 47. Fermentation - Monod model
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Fermentation (Monod growth kinetics)
def model(y, t):
    X = y[0]
    S = y[1]
    P = y[2]
    mi = mi_max * S/(KM + S)
    dXdt = X * mi
    dSdt = -1/Y_XS * X * mi
    dPdt = Y_PX * X * mi
    return [dXdt, dSdt, dPdt]

KM = 0.4 # g/L
Y_XS = 0.5
Y_PX = 0.1
mi_max = 1 # 1/h


# Experiment with different values:
X0 = 0.1 # g/L
S0 = 10  # g/L
P0 = 0   # g/L

t = np.linspace(0, 5) # 0-5h
results = odeint( model, [X0, S0, P0], t)
X = results[:,0]
S = results[:,1]
```

```
P = results[:,2]
plt.plot(t,X, label='X')
plt.plot(t,S, label='S')
plt.plot(t,P, label='P')
plt.xlabel('Time, h')
plt.ylabel('Conc., g/L')
plt.legend()
plt.savefig('47-1.pdf')
plt.close()
```

Results



Exercises

1) Try varying initial concentrations of biomass and the substrate to see what happens.

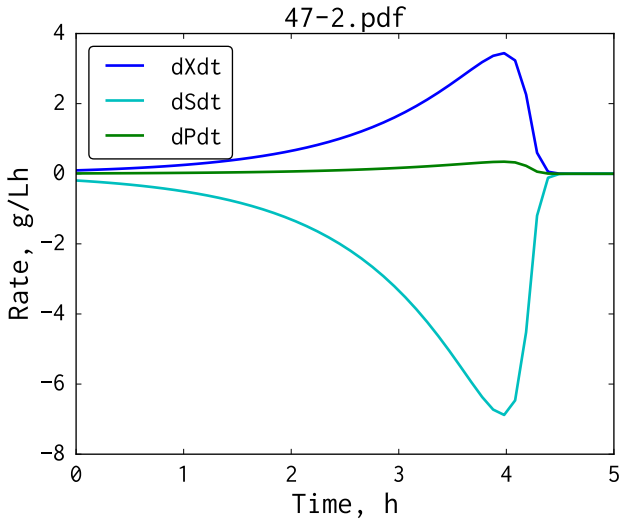
2) Modify script to plot rates ($dX/dt, dS/dt$ and $dPdt$) versus fermentation time. Calculate instant yield coefficients and mass balance.

Calculations

```
# File 47-2.py [View]
# Example 47. Fermentation - Monod model
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Fermentation (Monod growth kinetics)
def model(y, t):
    X = y[0]
    S = y[1]
    P = y[2]
    mi = mi_max * S/(KM + S)
    dXdt = X * mi
```

```
    dSdt = -1/Y_XS * X * mi
    dPdt = Y_PX * X * mi
    return [dXdt, dSdt, dPdt]

KM = 0.4 # g/L
Y_XS = 0.5
Y_PX = 0.1
mi_max = 1 # 1/h

# initial conditions
X0 = 0.1 # g/L
S0 = 10  # g/L
P0 = 0   # g/L

t = np.linspace(0, 5) # 0-5h
results = odeint( model, [X0, S0, P0], t)
X = results[:,0]
S = results[:,1]
P = results[:,2]

# calculate rates
dXdt = mi_max * X * S/(KM + S)
dSdt = -1/Y_XS * mi_max * X * S/(KM + S) # minus = uptake
dPdt = Y_PX * mi_max * X * S/(KM + S)

plt.plot(t,dXdt, label='dXdt')
plt.plot(t,dSdt, label='dSdt')
plt.plot(t,dPdt, label='dPdt')
plt.xlabel('Time, h')
plt.ylabel('Rate, g/Lh')
plt.legend(loc='best')
plt.title('47-2.pdf')
plt.savefig('47-2.pdf')
plt.close()

# yield coefficients
print 'dXdt/-dSdt'
print dXdt/-dSdt
print 'dPdt/dXdt'
print dPdt/dXdt

# Mass balances
print S + X + P
```

Results




```
%run 47-2.py
dXdt/-dSdt
[ 0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5
  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5
  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5
  0.5  0.5  0.5  0.5  0.5]
dPdt/dXdt
[ 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1]
[ 10.1         10.0907222   10.08048887  10.06920183  10.05675279
  10.04302243  10.02787927  10.01117835   9.99275999   9.97244827
   9.9500494    9.92534991   9.89811484   9.86808549   9.83497707
   9.79847634   9.75823875   9.71388552   9.66500043   9.61112637
   9.55176168   9.48635625   9.41430745   9.33495584   9.24758092
   9.15139699   9.0455491    8.92910992   8.80107759   8.66037564
   8.50585617   8.33630875   8.15047853   7.9471003    7.72496067
   7.48301239   7.2205904    6.93784386   6.63667364   6.32301048
   6.01317522   5.7524179    5.62405315   5.6020254    5.60015176
   5.60001126   5.60000083   5.60000006   5.6          5.6       ]
```

Exercises

Why mass balance is not conserved? Please note that concentrations of X, S and P are in $\mathrm{g\,L^{-1}}$.

Example 48. Batch Alcoholic Fermentation

●●○

Batch alcoholic fermentation with *Saccharomyces cerevisiae* at different temperatures and initial sugar concentrations was modeled [1]:

$$\begin{aligned} \frac{dX}{dt} &= f_1(X,S,P) \\ \frac{dP}{dt} &= f_2(X,S,P) \\ \frac{dS}{dt} &= f_3\left(\frac{dX}{dt}, \frac{dP}{dt}\right) \end{aligned} \tag{36}$$

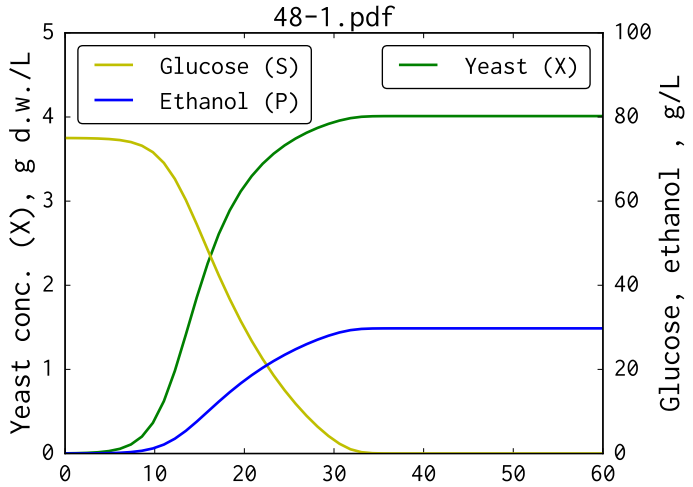
Two inhibition models by the product were tested: Aiba model (Model A) and Novak model (Model B). Implement in Python Model A. Simulate fermentation process assuming that initially: $X_0 = 0.002\,\mathrm{g\,L^{-1}}$, $P_0 = 0$, $S_0 = 75.0\,\mathrm{g\,L^{-1}}$ and the total fermentation time is 60 h. Other needed parameters take from the paper.

Calculations

```
# File 48-1.py [View]
# Example 48. Batch Alcoholic Fermentation
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Model A, see:
# F. Gbdia, C. Casas and C. Sola
# Chem. Tech. Biorechnol. 1988,41, 155-165

def modelA(y, t):
    X = y[0]
    P = y[1]
    S = y[2]
    dXdt = mi_max * X * np.exp(-KP*P) * S/(KS+S)
    dPdt = ni_max * X * np.exp(-KPP*P) * S/(KSS+S)
    dSdt = -1/RSX * dXdt - 1/RSP*dPdt
    return [dXdt, dPdt, dSdt]

mi_max = 0.556 # 1/h
KS = 1.03 # g/L
KP = 0.139 # g/L
ni_max = 1.79 # 1/h
KSS = 1.68 # g/L
KPP = 0.07 # g/L
RSX = 0.607
RSP = 0.435

# initial conditions
X0 = 0.002 # g d.w./L (d.w. = dry weight)
P0 = 0 #
S0 = 75 # g/L

t = np.linspace(0, 60) # 0 - 60 h
results = odeint( modelA, [X0, P0, S0], t)
```

```
X = results[:,0]
P = results[:,1]
S = results[:,2]

ax1 = plt.subplot(111)
ax1.plot(t,X, 'g', label='Yeast (X)')
ax1.set_ylabel('Yeast conc. (X), g d.w./L')
ax1.legend(loc='best')
ax1.set_ylim(0,5)

ax2 = ax1.twinx()
ax2.plot(t,S, 'y', label='Glucose (S)')
ax2.plot(t,P, 'b', label='Ethanol (P)')
ax2.set_ylabel('Glucose, ethanol , g/L')
ax2.set_xlabel('Time, h')
ax2.legend(loc='upper left')
ax2.set_ylim(0,100)
plt.title('48-1.pdf')
plt.savefig('48-1.pdf')
plt.close()
```

Results

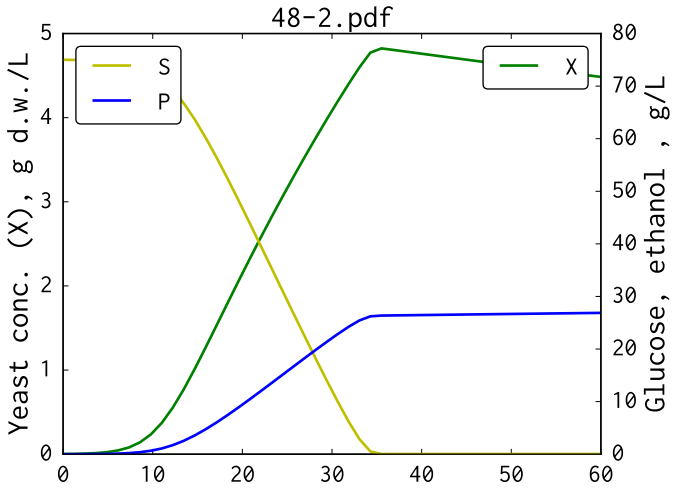


Exercises

Implement model B (see [1]). Simulate fermentation process assuming that initially: $X_0 = 0.002\,\mathrm{g\,L^{-1}}$, $P_0 = 0$, $S_0 = 75.0\,\mathrm{g\,L^{-1}}$, and the total fermentation time is 60 h.

Results

Your plot should look like this:



[1] F Godia, C Casas, and C Sola. “Batch alcoholic fermentation modeling by simultaneous integration of growth and fermentation equations”. In: *Journal of Chemical Technology and Biotechnology* 41 (1988), pp. 155–165.

Example 49. Enzymatic reaction kinetics

●●○

Develop a kinetic model for simple enzymatic reaction:

$$S \xrightarrow{E} P \tag{37}$$

Plot $S(t)$, $P(t)$, $E(t)$, $ES(t)$ for time ranged from 0 to

1) 1 s (pre-equilibrium stage),

2) 6000 s,

assuming $k_1 = 500$, $k_2 = 6$, $k_3 = 1.5$ if initially $S_0 = 0.1\,\mathrm{mol\,L^{-1}}$ and $E_0 = 0.000\,01\,\mathrm{mol\,L^{-1}}$.

Based on quasi-steady-state assumption calculate K_M and V_{max}. Calculate instantaneous substrate and enzyme balances.

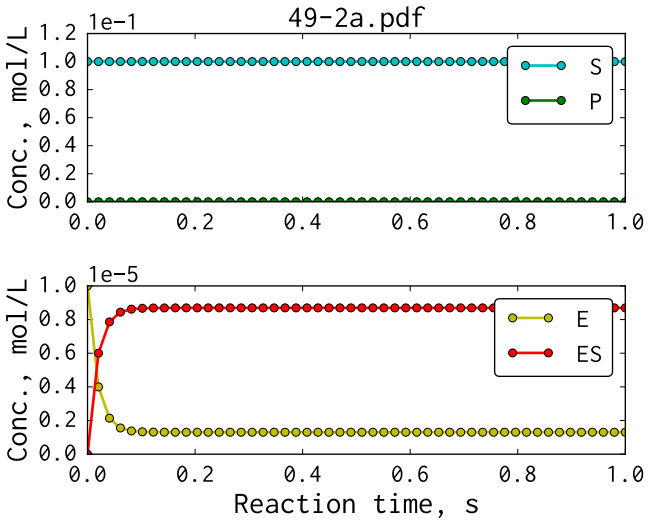
Calculations

```
# File 49-2.py [View]
# Example 49. Enzymatic reaction kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Simple enzymatic reaction
# E + S <-> ES, k1, k2
# ES -> E + P, k3

def model(y, t):
    E = y[0]                    # unpack variables
    S = y[1]
    ES = y[2]
    P = y[3]
    dEdt = -k1*E*S + k2*ES + k3*ES
    dSdt = -k1*E*S + k2*ES
    dESdt = k1*E*S - k2*ES - k3*ES
    dPdt = k3*ES
    return [dEdt, dSdt, dESdt, dPdt]  # return derivatives

k1 = 500
k2 = 6
k3 = 1.5
S0 = 0.1
E0 = 0.00001

ic = [E0, S0, 0, 0]
t = np.linspace(0,1)
results = odeint( model, ic, t)
E = results[:,0]
S = results[:,1]
ES = results[:,2]
P = results[:,3]

print("KM = {}".format((k2+k3)/k1))
print("Vmax = {}".format(k3*E0))
```

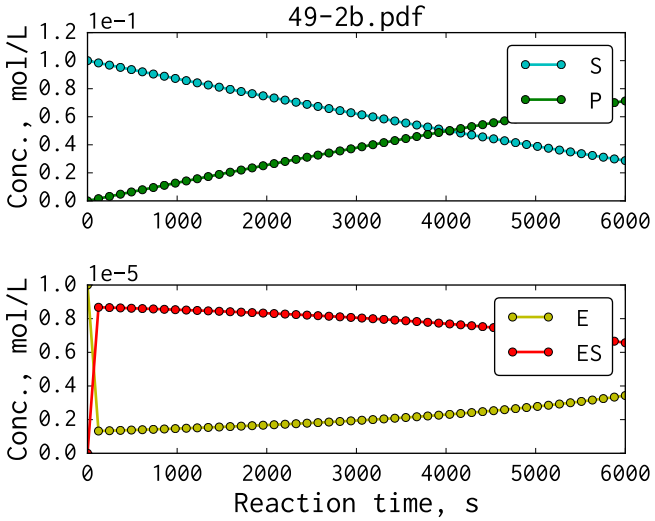
```
print "Substrate balance"
print S + ES + P
print "Enzyme balance"
print E + ES


# first second of the reaction ...
plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9)
ax1 = plt.subplot(211)
ax1.set_title('49-2a.pdf')
ax1.plot(t, S, 'co-', label='S')
ax1.plot(t, P, 'go-', label='P')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax1.legend()
ax1.set_ylabel('Conc., mol/L')
ax2 = plt.subplot(212)
ax2.plot(t, E, 'yo-', label='E')
ax2.plot(t, ES, 'ro-', label='ES')
ax2.legend()
ax2.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax2.set_xlabel('Reaction time, s')
ax2.set_ylabel('Conc., mol/L')
plt.savefig('49-2a.pdf')
plt.close()


# wider time window
t = np.linspace(0,6000)
results = odeint( model, ic, t)
E = results[:,0]
S = results[:,1]
ES = results[:,2]
P = results[:,3]

plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9)
ax1 = plt.subplot(211)
ax1.set_title('49-2b.pdf')
ax1.plot(t, S, 'co-', label='S')
ax1.plot(t, P, 'go-', label='P')
ax1.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax1.legend()
ax1.set_ylabel('Conc., mol/L')
ax2 = plt.subplot(212)
ax2.plot(t, E, 'yo-', label='E')
ax2.plot(t, ES, 'ro-', label='ES')
ax2.legend()
ax2.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax2.set_xlabel('Reaction time, s')
ax2.set_ylabel('Conc., mol/L')
plt.savefig('49-2b.pdf')
plt.close()
```

Results

Kinetic curves for time from 0 to 1 s (!):



Kinetic curves for time from 0 to 6000 s:




```
%run 49-2.py
KM = 0.015
Vmax = 1.5e-05
Substrate balance
[ 0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
  0.1  0.1  0.1  0.1  0.1]
Enzyme balance
[  1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
```

```
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05   1.00000000e-05   1.00000000e-05
   1.00000000e-05   1.00000000e-05]
```

Exercises

Compare exact solution of the ODEs model to Michaelis-Menten approximation (for steady-state).

1) Compare the substrate uptake rates

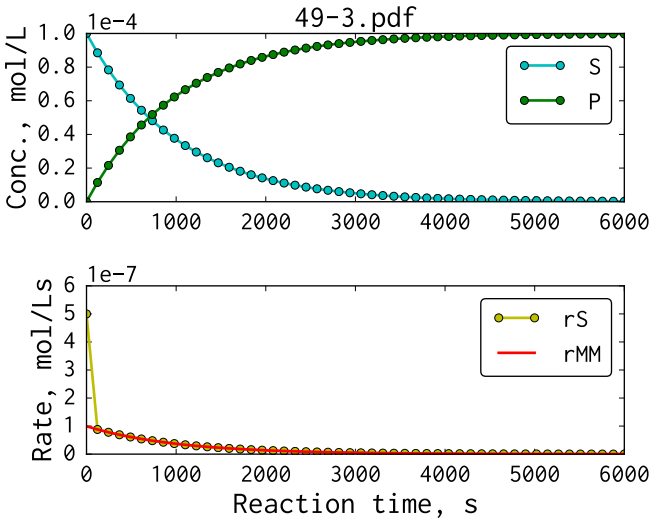
2) Compare the substrate concentrations

Results

The rates

```
# File 49-3.py [View]
# Example 49. Enzymatic reaction kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Simple enzymatic reaction
# E + S <-> ES, k1, k2
# ES -> E + P, k3
def model(y, t):
    E = y[0]                    # unpack variables
    S = y[1]
    ES = y[2]
    P = y[3]
    dEdt = -k1*E*S + k2*ES + k3*ES
    dSdt = -k1*E*S + k2*ES
    dESdt = k1*E*S - k2*ES - k3*ES
    dPdt = k3*ES
    return [dEdt, dSdt, dESdt, dPdt]  # return derivatives

k1 = 500
k2 = 6
k3 = 1.5


S0 = 0.0001           # experiment with different values
E0 = 0.00001

ic = [E0, S0, 0, 0]
t = np.linspace(0,6000)
results = odeint( model, ic, t)
E = results[:,0]
S = results[:,1]
ES = results[:,2]
P = results[:,3]

KM = (k2+k3)/k1
```

```
Vmax = k3*E0

# exact rate of substrate uptake
rS = -k1*E*S + k2*ES
# M-M approximation (for steady state)
rMM = Vmax*S/(KM+S)

plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9)
ax1 = plt.subplot(211)
ax1.set_title('49-3.pdf')
ax1.plot(t, S, 'co-', label='S')
ax1.plot(t, P, 'go-', label='P')
ax1.legend()
ax1.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax1.set_ylabel('Conc., mol/L')
ax2 = plt.subplot(212)
ax2.plot(t, -rS, 'yo-', label='rS')
ax2.plot(t, rMM, 'r-', label='rMM')
ax2.legend()
ax2.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax2.set_xlabel('Reaction time, s')
ax2.set_ylabel('Rate, mol/Ls')
plt.savefig('49-3.pdf')
plt.close()
```

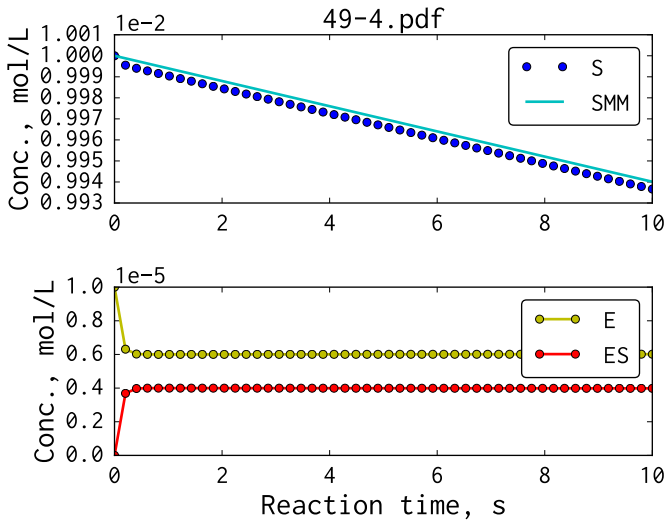


The concentrations

```
# File 49-4.py [View]
# Example 49. Enzymatic reaction kinetics
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

# Simple enzymatic reaction
```

```
# E + S <-> ES, k1, k2
# ES -> E + P, k3
def model(y, t):
    E = y[0]                    # unpack variables
    S = y[1]
    ES = y[2]
    P = y[3]
    dEdt = -k1*E*S + k2*ES + k3*ES
    dSdt = -k1*E*S + k2*ES
    dESdt = k1*E*S - k2*ES - k3*ES
    dPdt = k3*ES
    return [dEdt, dSdt, dESdt, dPdt]  # return derivatives

# Michaelis-Menten model
def modelMM(y, t):
    S = y[0]
    dSdt = -Vmax*S/(KM+S)
    dPdt = -dSdt
    return [dSdt, dPdt]


k1 = 500
k2 = 6
k3 = 1.5

S0 = 0.01           # experiment with different values
E0 = 0.00001

KM = (k2+k3)/k1
Vmax = k3*E0


t = np.linspace(0,10)

results = odeint( model, [E0, S0, 0, 0], t)
E = results[:,0]
S = results[:,1]
ES = results[:,2]
P = results[:,3]

results = odeint( modelMM, [S0, 0], t)
SMM = results[:,0]
PMM = results[:,1]

plt.subplots_adjust(hspace = 0.5, left=0.2, right=0.9)
ax1 = plt.subplot(211)
ax1.set_title('49-4.pdf')
ax1.plot(t, S, 'bo', label='S')
ax1.plot(t, SMM, 'c-', label='SMM')
#ax1.plot(t, P, 'go', label='P')
#ax1.plot(t, PMM, 'y-', label='PMM')
ax1.legend()
ax1.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
ax1.set_ylabel('Conc., mol/L')
ax2 = plt.subplot(212)
ax2.plot(t, E, 'yo-', label='E')
ax2.plot(t, ES, 'ro-', label='ES')
ax2.legend()
ax2.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
```

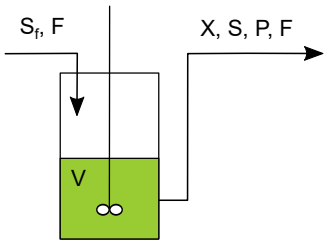
```
ax2.set_xlabel('Reaction time, s')
ax2.set_ylabel('Conc., mol/L')
plt.savefig('49-4.pdf')
plt.close()
```



Example 50. Chemostat

● ○ ○

Consider a fermentation process in a CSTR, assuming Monod-type biomass growth. Here is a Python implementation of Chemostat model.

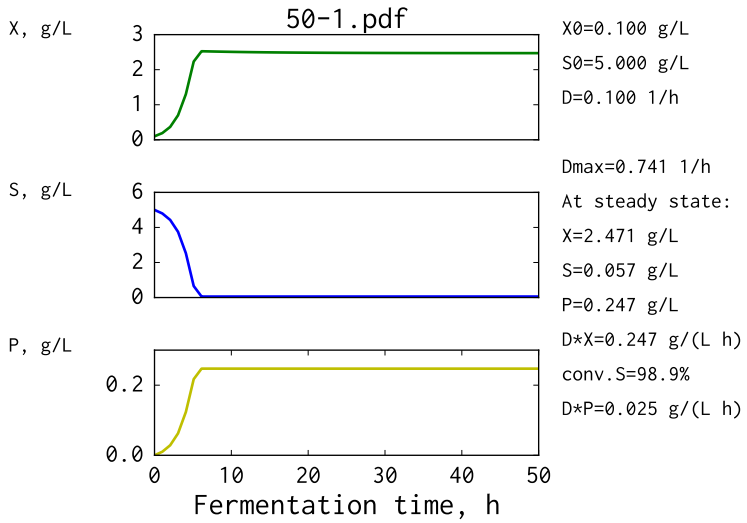


Try varying fermentation parameters to see what happens. Note dilution rate is defined as medium flow rate over culture volume: $D = F/V$. For the explanation of the other symbols see example 46.

```
# File 50-1.py [View]
# Example 50. Chemostat
# Copyright (C) 2016, Szczepan Bednarz
# Released under the GNU General Public License

import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint
from scipy.optimize import fsolve

# algebraic equations
def steadystate(y):
    X = y[0]
    S = y[1]
    P = y[2]
    mi = mi_max * S/(KM + S)
    rX = X * mi
    rS = -1/Y_XS * X * mi
    rP = Y_PX * X * mi
    eq1 = rX - D*X
    eq2 = D*Sf - D*S + rS
    eq3 = rP - D*P
    return [eq1, eq2, eq3]

def Dmax(S0):
     return mi_max * S0/(KM + S0)

# ODEs
def model(y, t):
    X = y[0]
    S = y[1]
    P = y[2]
    mi = mi_max * S/(KM + S)
    rX = X * mi
    rS = -1/Y_XS * X * mi       # note negative value
    rP = Y_PX * X * mi
    # non-steady state
    dXdt = rX - D*X             # sterile feed
    dSdt = D*Sf - D*S + rS      # +rS because rS is negative, see above
```

```
    dPdt = rP - D*P
    return [dXdt, dSdt, dPdt]

Sf = 5 # g/L
D = 0.1 # 1/h     <----------------- try values from 0.01 to 1
KM = 0.4 # g/L
Y_XS = 0.5
Y_PX = 0.1
mi_max = 0.8 # 1/h


X0 = 0.1 # g/L
S0 = Sf  # g/L
P0 = 0   # g/L


tmax = 50 # h
t = np.linspace(0, tmax)
results = odeint( model, [X0, S0, P0], t)
X = results[:,0]
S = results[:,1]
P = results[:,2]

# calculation of steady state concentrations
guess = [2,0,0.1]
ss = fsolve(steadystate, guess)
Xss = ss[0]
Sss = ss[1]
Pss = ss[2]

#print Dmax(S0)
#print Xss, Sss, Pss

fig = plt.figure()
fig.subplots_adjust(hspace = 0.5, left=0.2, right=0.7)
ax = fig.add_subplot(311)
ax.set_title('50-1.pdf')
ax.plot(t, X, 'g-')
ax.set_xticks([])
ax.locator_params(axis = 'y',tight=False, nbins=3)
ax = fig.add_subplot(312)
ax.plot(t, S, 'b')
ax.set_xticks([])
ax.locator_params(axis = 'y',tight=False, nbins=3)
ax = fig.add_subplot(313)
ax.plot(t, P, 'y')
ax.locator_params(axis = 'y',tight=False, nbins=3)
ax.set_xlabel('Fermentation time, h')

ax.text(0.01,0.9,'X, g/L', transform=fig.transFigure)
ax.text(0.01,0.62,'S, g/L', transform=fig.transFigure)
ax.text(0.01,0.35,'P, g/L', transform=fig.transFigure)

ax.text(0.73,0.9, 'X0={:.3f} g/L'.format(X0) , transform=fig.transFigure)
ax.text(0.73,0.84, 'S0={:.3f} g/L'.format(S0) , transform=fig.transFigure)
ax.text(0.73,0.78, 'D={:.3f} 1/h'.format(D) , transform=fig.transFigure)
ax.text(0.73,0.66, 'Dmax={:.3f} 1/h'.format(Dmax(S0)) , transform=fig.transFigure)

if D < Dmax(S0):
```

```
    ax.text(0.73,0.60, 'At steady state:' , transform=fig.transFigure)
else:
    ax.text(0.73,0.60, '! Washout:' , transform=fig.transFigure)

ax.text(0.73,0.54, 'X={:.3f} g/L'.format(Xss) , transform=fig.transFigure)
ax.text(0.73,0.48, 'S={:.3f} g/L'.format(Sss) , transform=fig.transFigure)
ax.text(0.73,0.42, 'P={:.3f} g/L'.format(Pss) , transform=fig.transFigure)
ax.text(0.73,0.36, 'D*X={:.3f} g/(L h)'.format(D*Xss) , transform=fig.transFigure)
ax.text(0.73,0.30, 'conv.S={:.1f}%'.format(100*(S0-Sss)/S0) , transform=fig.transFigure)
ax.text(0.73,0.24, 'D*P={:.3f} g/(L h)'.format(D*Pss) , transform=fig.transFigure)
plt.savefig('50-1.pdf')
plt.close()
```



9 Bibliography

Books

[1] John Ingham, Irving J. Dunn, Elmar Heinzle, Jiri E. Prenosil, and Jonathan B. Snape. *Chemical Engineering Dynamics: An Introduction to Modelling and Computer Simulation*. 3rd ed. Wiley-VCH, 2007.

[2] Irving J. Dunn, Elmar Heinzle, John Ingham, and Jiri E. Prenosil. *Biological Reaction Engineering : Dynamic Modelling Fundamentals with Simulation Examples*. 3rd ed. Wiley-VCH, 2016.

[3] Stanley M. Walas. *Chemical Reaction Engineering Handbook of Solved Problems*. CRC Press, 1995.

[4] O. Levenspiel. *Chemical Reaction Engineering*. John Wiley & Sons, 1999.

[5] H.S. Fogler. *Elements of Chemical Reaction Engineering*. Prentice-Hall, 1999.

[6] David M. Himmelblau and James B. Riggs. *Basic Principles and Calculations in Chemical Engineering*. 7th ed. Prentice Hall, 2003.

[7] J.B. Rawlings and J. G. Ekerdt. *Chemical Reactor Analysis and Design Fundamentals*. Nob Hill Publishing, 2002.

[8] Erich Steiner. *The Chemistry Maths Book, Second Edition*. 2nd. Oxford University Press, USA, 2008.

[9] B. Tabiś and W. Żukowski. *Przykłady i zadania z zakresu inżynierii reaktorów chemicznych*. Politechnika Krakowska, 2000.

[10] B. Tabiś. *Zasady inżynierii reaktorów chemicznych*. Wydawnictwo Naukowo-Techniczne, 2000.

[11] J. Handzlik and J. Ogonowski. *Ćwiczenia tablicowe z technologii organicznej*. Politechnika Krakowska, 1995.

[12] J. Bałdyga, M. Henczka, and W. Podgórska. *Obliczenia w inżynierii bioreaktorów*. 2nd ed. Politechnika Warszawska, 2012.

Internet resources

[1] R. K. Herz. *Chemical Reaction Engineering*. URL: http://www.reactorlab.net.

[2] John R. Kitchin. *pycse: First release*. June 2015. DOI: 10.5281/zenodo.19111. URL: http://dx.doi.org/10.5281/zenodo.19111.

[3] Hans Fangohr. *Introduction to Python for Computational Science and Engineering*. URL: http://www.southampton.ac.uk/~fangohr/training/python/pdfs/Python-for-Computational-Science-and-Engineering.pdf.

[4] Zed A. Shaw. *Learn Python the Hard Way*. URL: http://learnpythonthehardway.org/book/index.html.

[5] M. Scott Shell. *An introduction to Python for scientific computing*. URL: http://www.engr.ucsb.edu/~shell/che210d/python.pdf.

[6] M. Scott Shell. *An introduction to Numpy and Scipy*. URL: http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf.