ProgressCurve_TimeCourse

February 1, 2022

Written by G. Gygli.

Contact gudrun.gygli@kit.edu in case of questions.

This script uses Python 3

1 A time-course experiment

This script aims to help you understand how to optimize the design of a time-course experiments for ***single-substrate, single-enzyme catalyzed reactions***.

Input: estimates of K_m and v_max and indicate the enzyme concentration concentration (E_0) and substrate concentration (s_0) you are planning to use.
If you have no estimates to give, try the experiment with the suggested initial values.

This notebook is heavily based on the work by Stroberg and Schnell, and their recommendations for the design of time course experiments: https://doi.org/10.1016/j.bpc.2016.09.004

In order for both K_m and v_max to be derived from substrate progress curve measurements: 1. s_0 must be within approximately an order of magnitude of K_m.
a. If $s_0 >> K_\mathrm{m}$, a linear fit to the initial velocity will yield v_max, but provide no information about K_m.
b. If $s_0 << K_\mathrm{m}$, the ratio of v_max to K_m, but neither parameter independently can be determined. 2. E_0 must be smaller than the Michaelis constant, that is $\frac{E_0}{K_\mathrm{m}} << 1$ 3. Data points should be collected around the time point where the time course curvature is at it highest. 4. E_0 must be smaller than s_0. 5. E_0 should be between 0.25 and 25 K_m.

Finally, note that the equations used to simulate data are:

$\frac{s(t)}{s_0} = (\frac{s_0}{K_\mathrm{m}})^{-1} W[\frac{s_0}{K_\mathrm{m}} e^{(\frac{s_0}{K_\mathrm{m}} - \frac{V}{K_\mathrm{m}} t)}]$
(which does not take into account E_0).

And the the approximation taking into account E_0:

$\frac{s(t)}{s_0} \approx e^{-\frac{k_\mathrm{cat} e_0 t}{K_\mathrm{m}} (1 - \frac{s_0}{k_\mathrm{cat} e_0} t)}$

We will perform steps with this notebook:

Step 1: Import packages and define functions

Step 2: Provide input parameters - Enzyme reaction parameters - Experimental conditions parameters

Step 3: Simulate and plot data

Step 4: Now it is up to you

NOTE that your enzyme might behave completely different than in this example due to a more complex reaction mechanism!

1.0.1 Step 1: Import packages and define functions

[1]:
```
# Here, we import all the python packages we need to run this script.
# in the unlikely case they are not installed on your computer, this might help:
 ↪
# https://jakevdp.github.io/blog/2017/12/05/
 ↪installing-python-packages-from-jupyter/
# accessed 04.10.21
import pandas
from scipy.special import lambertw
from scipy.optimize import curve_fit
import numpy as np
import scipy
import warnings
from scipy.optimize import OptimizeWarning
import matplotlib.pyplot as plt

# function to model
def Srt_schnell(t, Km: float, Vmax: float):
#     print("t,Km,Vmax",t,Km,Vmax)
    E = np.exp((s0/Km)-(Vmax/Km)*t)
#     L = np.abs(lambertw((s0/Km)*E))  # this apparently can be complex which␣
 ↪can cause Errors
    # - the initial valuesgiven in the example below do not suffer form this␣
 ↪problem, in  case you
    # run into this problems, try using the line below instead,
    # where the output of the lambertw function is converted into an absolute␣
 ↪value (using np.abs() )
    L = lambertw((s0/Km)*E)  # this apparently can be complex which can cause␣
 ↪Errors
    y = pow((s0/Km),-1)*L
    return y

def Srt_enzymeconcentrationdependent(t,E0, Km: float, Vmax: float):
    kcat=Vmax/E0
    y = np.exp((-(kcat*t*E0)/Km)*(1 - (s0/(kcat*E0*t))))
    return y
```

1.1 Step 2. Provide input parameters

1.1.1 2.1: Enzyme reaction parameters

Here, you can give an estimate of the enzyme reaction parameters K_m and v_max for a ***single-substrate, single-enzyme catalyzed reaction***. These parameters are needed to model data using the Michaelis-Menten equation, and can be adjusted as you progress with your experiments.

[2]:
```
vmax = 10 # units:  M/s, initial value: 10
Km = 100 # units:  M, initial value: 100
```

[3]:
```
# we will be using a randon noise generator to include some variation in our␣
 ↪simulated data
# fixing the seed for the random noise generation to always get the same result.
 ↪
# comment this line if you want to always get different data.
np.random.seed(1) # initial value: 1
# noiselevelMM controls how "noisy" your fake data will be, but it has not real␣
 ↪meaning for your actual experiment.
noiselevelMM = 1 # initial value: 1
```

1.1.2 2.2 Experimental conditions parameters

Choose how long you want to follow the reaction, i.e. how long your experiment should take by setting the length of t (units in the example are minutes, this obviously has consequences on the units of v_max, and the x-axis in the plot). Units must be manually verified and adjusted if changes are made…

Note that besides t you also **MUST** give a starting concentration of substrate (s_0) and a starting concentration of enzyme (E_0).

[4]:
```
t = list(range(0, 100,5)) # units: seconds, initial value: 100
# note: this line creates a list (array), filled with values starting from 0␣
 ↪and ending with 100, with a step of 5:
# 0,5,10,...,100
s0 = 200 # units:  M, initial value: 200
E0 = 25 # units:  M, initial value: 25
```

1.2 Step 3: Simulate and plot data

Now, run all the code below to see what data you can expect to obtain for the input parameters you gave.

This is where the data are simulated and plotted. Depending on how the plot looks, you may want to adjust t , s_0 and E_0.

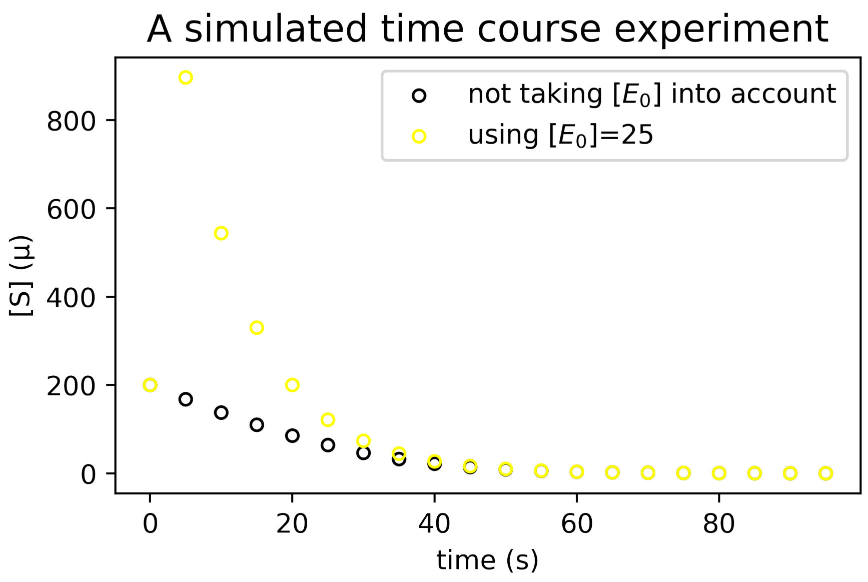
Note that v_max influences how quickly the reaction is over. Because v_max is a property of the enzyme, we need to shorten t for high v_max, and lengthen t for low v_max.

The approximation used to simulate data taking into account E_0 leads to unrealistic behaviour for small t s: in the beginning of your simulated experiment, you can observe substrate concentrations that are $>> s_0$ (uncomment line 31 in the code below to fully see the effect). This is certainly not realistic, but shows you that the curvature of your data can change in this case.

```
[5]: # some preparations:
v0 = t.copy() # initialize the array, the values will be overwritten later
v1 = t.copy() # initialize the array, the values will be overwritten later

# fixing the seed for the random noise generation to always get the same result.
 ↪
# comment this line if you want to always get different data.
np.random.seed(1) # initial value: 1

#create a figure with satisfactory dimensions and resolution:
fig = plt.figure(figsize=[5,3], dpi=500)

# This code is repeated again and again below.
# It is not put into a function to enable beginners in Python programming to␣
 ↪understand what is happening.
for i, value in enumerate(t):
#     print(i)
    if i==0:
        v0[i]=s0
    else:
        v0[i] = s0*Srt_schnell(value,Km,vmax)
        noise = noiselevelMM*(np.random.random(1)-0.5)
        data_random = v0[i] + noise

for i, value in enumerate(t):
    if i==0:
        v1[i]=s0
    else:
        #using your E0 given above as input
        v1[i] = s0*Srt_enzymeconcentrationdependent(value,E0,Km,vmax)
        noise = noiselevelMM*(np.random.random(1)-0.5)
        data_random = v1[i] + noise

# plt.ylim(ymax = s0+(s0/10), ymin = 0-(s0/10))
# fixing the axis so that in case of very bad initial parameter choices we do␣
 ↪not get confused by [S]>>[S0]

plt.scatter(t,v0,label="not taking [$E_0$] into account", facecolors='none',␣
 ↪edgecolors='black',s=20)
plt.scatter(t,v1,label='using [$E_0$]={}'.format(E0), facecolors='none',␣
 ↪edgecolors='yellow',s=20)

plt.title("A simulated time course experiment", fontsize=14)
plt.legend(loc='upper right')
plt.ylabel('[S] (\u03BC)')
plt.xlabel('time (s)')
```

```
C:\Users\Gudrun\anaconda3\lib\site-packages\numpy\core\_asarray.py:138:
ComplexWarning: Casting complex values to real discards the imaginary part
  return array(a, dtype, copy=False, order=order, subok=True)
```

```
[5]: Text(0.5, 0, 'time (s)')
```



If we now check if we fulfill the criteria by Stroberg and Schnell (https://doi.org/10.1016/j.bpc.2016.09.004), we see that there are some issues to deal with:

1. The s_0 must be within approximately an order of magnitude of the Michaelis constant. TRUE: $s_0 = 200$ M, $K_\mathrm{m} = 100$ M
2. E_0/K_m << 1 TRUE: E_0=25 -> 25/100 << 1
3. Data points should be collected around the time point where the time course curvature is at it highest. FALSE: we need to sample better by reducing the stepsize for our list called "time".
4. E_0 must be smaller than s_0. TRUE $E_0 = 25$ M $< s_0 = 200$ M
5. E_0 should be between 0.25 and 25 K_m. TRUE

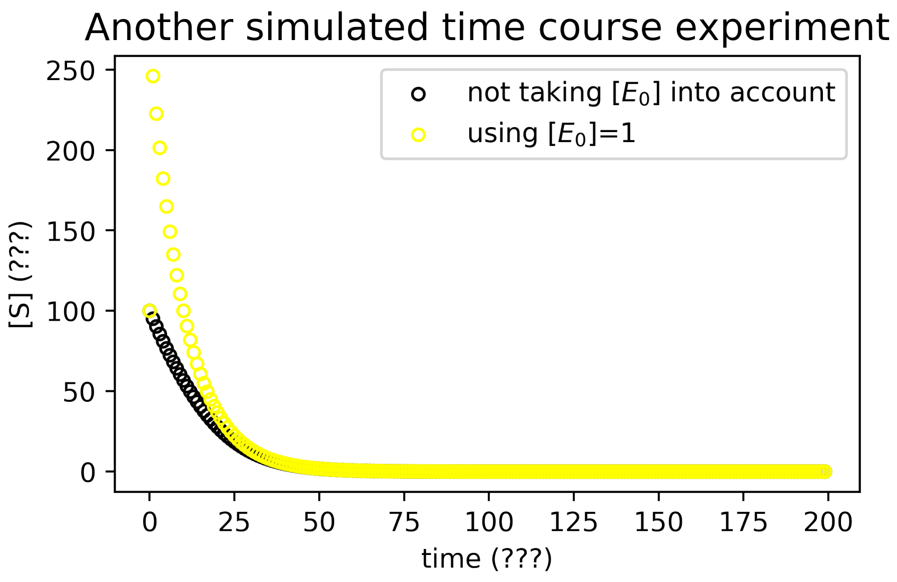
1.3 Step 4: Now it is up to you

you can try what happens if you adjust E_0**,** s_0 **and** t in the code below. Make sure to adjust the units and info in the plot axes and legend.

[6]:
```
t = list(range(0, 200)) # units: seconds (?)
# note: this line creates a list (array) filled with 100 elements, starting␣
↪from 0 and ending with 100
s0 = 100 # units:  M (?)
E0 = 1 # units:  M (?)

# some preparations:
v0 = t.copy() # initialize the array, the values will be overwritten later
v1 = t.copy() # initialize the array, the values will be overwritten later

# fixing the seed for the random noise generation to always get the same result.
↪
# comment this line if you want to always get different data.
np.random.seed(1) # initial value: 1

#create a figure with satisfactory dimensions and resolution:
fig = plt.figure(figsize=[5,3], dpi=500)

# This code is repeated again and again below.
# It is not put into a function to enable beginners in Python programming to␣
↪understand what is happening.
for i, value in enumerate(t):
#     print(i)
    if i==0:
        v0[i]=s0
    else:
        v0[i] = s0*Srt_schnell(value,Km,vmax)
        noise = noiselevelMM*(np.random.random(1)-0.5)
        data_random = v0[i] + noise

for i, value in enumerate(t):
    if i==0:
        v1[i]=s0
    else:
        #using your E0 given above as input
        v1[i] = s0*Srt_enzymeconcentrationdependent(value,E0,Km,vmax)
        noise = noiselevelMM*(np.random.random(1)-0.5)
        data_random = v1[i] + noise

# plt.ylim(ymax = s0+(s0/10), ymin = 0-(s0/10))
#fixing the axis so that in case of very bad initial parameter choices we do␣
↪not get confused

plt.scatter(t,v0,label="not taking [$E_0$] into account", facecolors='none',␣
↪edgecolors='black',s=20)
plt.scatter(t,v1,label='using [$E_0$]={}'.format(E0), facecolors='none',␣
↪edgecolors='yellow',s=20)
```

```
plt.title("Another simulated time course experiment", fontsize=14)
plt.legend(loc='upper right')

plt.ylabel('[S] (???)')
plt.xlabel('time (???)')
```

```
[6]: Text(0.5, 0, 'time (???)')
```



```
[ ]:
```