Design_an_Initial_Rate_Experiment

October 11, 2021

Written by G. Gygli.

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

 This script uses Python 3

[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 matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

# Michaelis-Menten equation as a function for easy use in the script
def michaelismenten_equation(S0,Km,vmax):
    return vmax*(s/(Km+s)) #v0
# other equations to describe an enzyme reaction can be used by scripting these␣
↪as functions
```

1 Design an initial rate experiment in four steps

This script aims to help you design high-quality initial rate experiments. It uses the Michaelis-Menten equation to simulate data for an enzyme with unknown enzyme reaction parameters. Four steps are needed: #### 1. Estimate the enzyme reaction parameters Input: estimates of K_m and v_max and indicate the enzyme concentration concentration you are planning to use.
If you have no estimates to give, try the experiment with the suggested initial values.

2. "Zero-round" experiment Input: estimates from 1. and **5** broadly spaced substrate concentrations.
Output: fake Michaelis-Menten plot for these parameters to obtain an intial first guess of K_m

3. "First-round" experiment Input: estimates from 1. and **10** better spaced substrate concentrations to obtain an intial first guess of K_m.
Output: fake Michaelis-Menten plot for these parameters to obtain a better guess of K_m

4. "Gold-round" experiment Input: estimates from 1. and **10** perfectly spaced substrate concentrations to obtain an intial first guess of K_m.
Output: fake Michaelis-Menten plot for these parameters to obtain K_m and v_max values

1.1 1. Estimate enzyme reaction parameters

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

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

1.2 2. "Zero-round" experiment

Give the 5 widely spaced concentrations of substrate you want to use for this first-round experiment. 3 Options are given to show you what happens if different S0 are chosen. Uncomment and comment the different lines of code. 1. S0 spaced widely around K_m 2. All S0 spaced below K_m 3. All S0 spaced above K_m

[4]:
```
# 1.
S0 = [0,0.01,1,100,1000,1000000] # units: M, initial values: [0.
↪01,1,100,1000,1000000]

# 2. all S0 below Km
#S0 = [0.01,1,20,50,80] # units: M, initial values: [0.01,1,20,50,80]

# 3. all S0 above Km
#S0 = [300,500,1000,5000,10000] # units: M, initial values:␣
↪[300,500,1000,5000,10000]

S0 = np.round(S0, 2) #round the substrate concentrations to 2 digits
```

Now, run all the code below to see what data you can expect to obtain for the S_0, v_max and K_m you gave.

This is where the data are simulated and plotted. Depending on how the plot looks, you may want to adjust S_0.

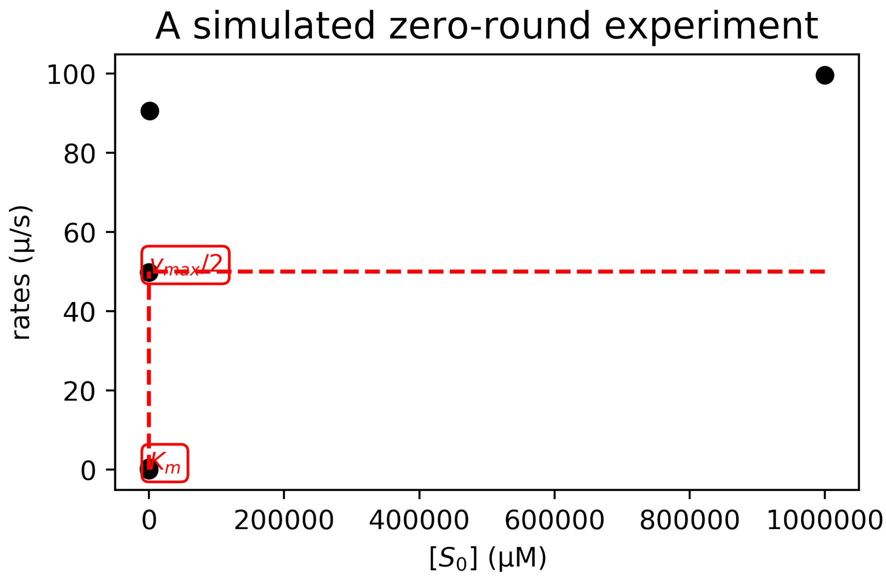
Note that kcat only influences the scale of the y-axis.

1.2.1 Attention:

In this **zero-round experiment**, you will **NOT** get "nice looking" data, but data that helps you to adjust the substrate concentrations S0 so that you can plan the first-round experiment. Try the different options above (Ln 4) to see how they influence the plot you get.

[5]:
```python
# some preparations:
v0 = S0.copy() # initialize the array, the values will be overwritten
# 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,s in enumerate(S0):
    v0[i] = michaelismenten_equation(s,Km,vmax)
    #adding some artificial, random noise
    noise = noiselevelMM*(np.random.random(1)-0.5)
    data_random = v0[i] + noise
    plt.scatter(s,data_random,c="black")

# add some annotations to the plot
plt.vlines(Km, 0, vmax/2, colors="red", linestyles='dashed')
plt.hlines(vmax/2, S0[-1], Km, colors="red", linestyles='dashed')
plt.text(Km,0,"$K_{m}$",color="red",fontsize=9,backgroundcolor="white",
         bbox=dict(facecolor='none', edgecolor='red', boxstyle='round'))
plt.text(0,vmax/2,"$v_{max}$/2",color="red",fontsize=9,
         bbox=dict(facecolor='none', edgecolor='red', boxstyle='round'))
plt.title("A simulated zero-round experiment", fontsize=14)
plt.ylabel('rates (\u03BC/s)')
plt.xlabel('[$S_{0}$] ( M)')
```

[5]:
```
Text(0.5, 0, '[$S_{0}$] ( M)')
```



1.3 3. First-round experiment

Now that we have a better idea of our Km, we can adjust the S0 to enable a better fit of the data with the Michaelis-Menten equation. The lowest S0 should be at least 10x lower than Km and the highest S0 at least 10x higher than Km, with 6 concentrations below and 4 above Km.

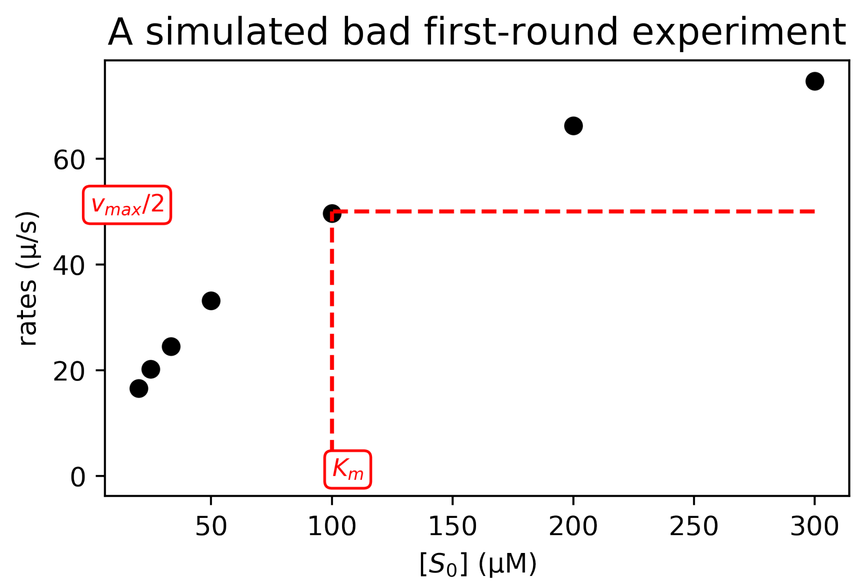
1.3.1 A bad example

Here follows an example of how data look if S0 are badly chosen. The initial values given were chosen too closely together.

[6]:
```
S0 = [Km/5,Km/4,Km/3,Km/2,Km, Km*2,Km*3]
#  M, initial values: [Km/5,Km/4,Km/3,Km/2,Km,Km*2,Km*3]

S0 = np.round(S0, 2) #round the substrate concentrations to 2 digits
v0 = S0.copy() #initialize the array, the values will be overwritten

# 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)

for s in S0:
    v0[i] = michaelismenten_equation(s,Km,vmax)
    #adding some artificial, random noise
    noise = noiselevelMM*(np.random.random(1)-0.5)
```

```
    data_random = v0[i] + noise
    plt.scatter(s,data_random,c="black")

# add some annotations to the plot
plt.vlines(Km, 0, vmax/2, colors="red", linestyles='dashed')
plt.hlines(vmax/2, S0[-1], Km, colors="red", linestyles='dashed')
plt.text(Km,0,"$K_{m}$",color="red",fontsize=9,backgroundcolor="white",
         bbox=dict(facecolor='white', edgecolor='red', boxstyle='round'))
plt.text(0,vmax/2,"$v_{max}$/2",color="red",fontsize=9,
         bbox=dict(facecolor='white', edgecolor='red', boxstyle='round'))
plt.title("A simulated bad first-round experiment", fontsize=14)
plt.ylabel('rates (\u03BC/s)')
plt.xlabel('[$S_{0}$] ( M)')
```

[6]:
```
Text(0.5, 0, '[$S_{0}$] ( M)')
```



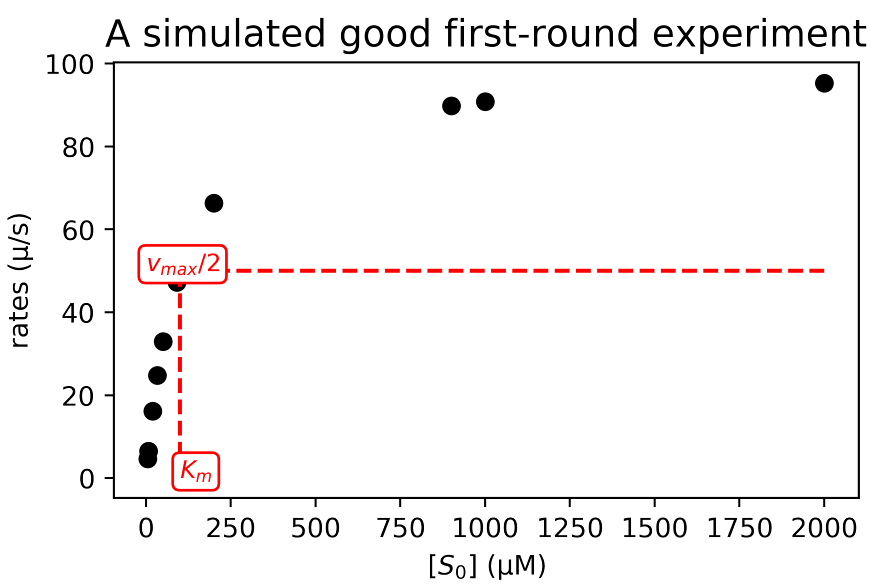
1.3.2 A good example

Here follows an example of how data look if S0 are well chosen. You will see that there is still room for improvement though.

[7]:
```
S0 = [Km/20,Km/15,Km/5,Km/3,Km/2,Km/1.1, Km*2,Km*9,Km*10,Km*20]
#  M, initial values: [Km/20,Km/15,Km/5,Km/3,Km/2,Km/1.1, Km*2,Km*9,Km*10,Km*20]

S0 = np.round(S0, 2) #round the substrate concentrations to 2 digits
v0 = S0.copy() #initialize the array, the values will be overwritten
```

```
# 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)

for s in S0:
    v0[i] = michaelismenten_equation(s,Km,vmax)
    #adding some artificial, random noise
    noise = noiselevelMM*(np.random.random(1)-0.5)
    data_random = v0[i] + noise
    plt.scatter(s,data_random,c="black")

# add some annotations to the plot
plt.vlines(Km, 0, vmax/2, colors="red", linestyles='dashed')
plt.hlines(vmax/2, S0[-1], Km, colors="red", linestyles='dashed')
plt.text(Km,0,"$K_{m}$",color="red",fontsize=9,backgroundcolor="white",
         bbox=dict(facecolor='white', edgecolor='red', boxstyle='round'))
plt.text(0,vmax/2,"$v_{max}$/2",color="red",fontsize=9,
         bbox=dict(facecolor='white', edgecolor='red', boxstyle='round'))
plt.title("A simulated good first-round experiment", fontsize=14)
plt.ylabel('rates (\u03BC/s)')
plt.xlabel('[$S_{0}$] ( M)')
```

```
[7]: Text(0.5, 0, '[$S_{0}$] ( M)')
```

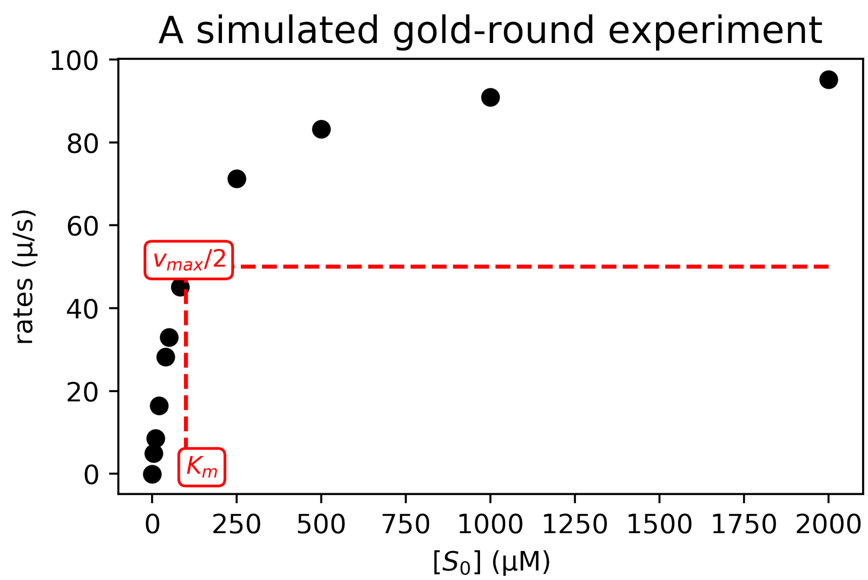


1.4 4. Gold-round experiment

Now that we have a much better idea of our Km and how the data look like, we can adjust the S0 to enable a better fit of the data with the Michaelis-Menten equation. The lowest S0 should be at least 10x lower than Km and the highest S0 at least 10x higher than Km, with 6 concentrations below and 4 above Km.

[8]:
```
S0 = [0,Km/20,Km/10,Km/5,Km/2.5,Km/2,Km/1.2,Km*2.5,Km*5,Km*10,Km*20]
#  M, initial values: [Km/20,Km/10,Km/5,Km/3,Km/2,Km/1.1,Km*2,Km*5,Km*10,Km*20]

S0 = np.round(S0, 2) #round the substrate concentrations to 2 digits
v0 = S0.copy() #initialize the array, the values will be overwritten

# 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)

for s in S0:
    v0[i] = michaelismenten_equation(s,Km,vmax)
    #adding some artificial, random noise
    noise = noiselevelMM*(np.random.random(1)-0.5)
    data_random = v0[i] + noise
    plt.scatter(s,data_random,c="black")

# add some annotations to the plot
plt.vlines(Km, 0, vmax/2, colors="red", linestyles='dashed')
plt.hlines(vmax/2, S0[-1], Km, colors="red", linestyles='dashed')
plt.text(Km,0,"$K_{m}$",color="red",fontsize=9,backgroundcolor="white",
         bbox=dict(facecolor='white', edgecolor='red', boxstyle='round'))
plt.text(0,vmax/2,"$v_{max}$/2",color="red",fontsize=9,
         bbox=dict(facecolor='white', edgecolor='red', boxstyle='round'))
plt.title("A simulated gold-round experiment", fontsize=14)
plt.ylabel('rates (\u03BC/s)')
plt.xlabel('[$S_{0}$] ( M)')
```

[8]:
```
Text(0.5, 0, '[$S_{0}$] ( M)')
```



2 Bonus

Simulated raw data are shown for the gold-round experiment.

[9]:
```
# function for linear fit
def linear(x,a,b):
    return x*a+b
```

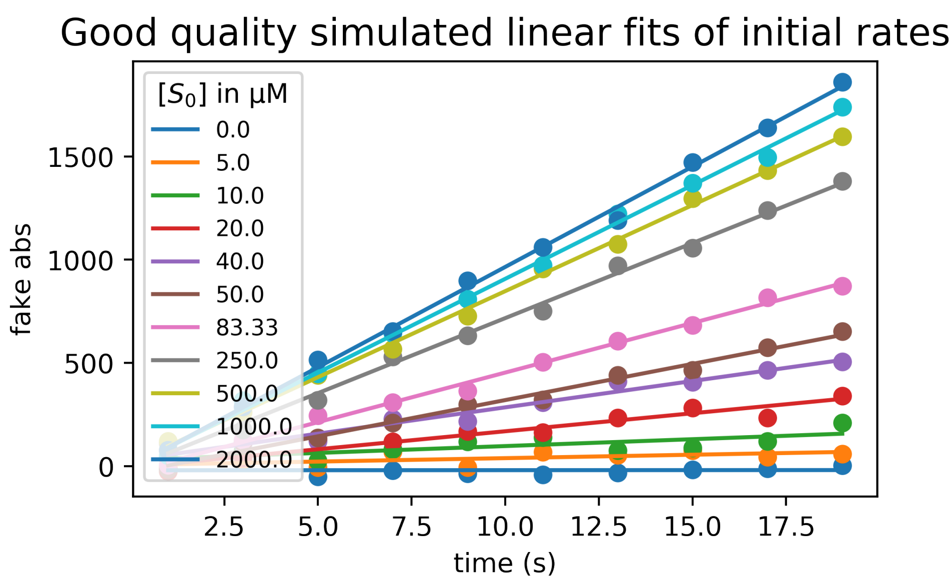
2.1 The noisier your data, the more datapoints you need to get reliable fits!

An example with little noise and 10 datapoints:

[10]:
```
# 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

time = np.arange(1, 20, 2) # initial values: (1, 20, 2): Start = 1, Stop = 20,␣
↪Step Size = 2
# get fake time data, at least 10 datapoints to enable a good fit
# noiselevelL controls how "noisy" your fake data will be.
noiselevelL = 100 #initial value: 100

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

for s in S0:
```

```
    v0 = michaelismenten_equation(s,Km,vmax)
    data = linear(time,v0,0.001)
    #adding some artificial, random noise
    noise = noiselevelL*(np.random.random(len(time))-0.5)
    data_random = data + noise
    plt.scatter(time,data_random)
    popt, pcov = curve_fit(linear, time, data_random)
    rate = popt[0]
    b = popt[1]
    plt.plot(time, linear(time,rate,b))

plt.legend(S0, title="[$S_{0}$] in  M ", loc="upper left", fontsize='small',␣
 ↪fancybox=True)
plt.title("Good quality simulated linear fits of initial rates", fontsize=14)
plt.ylabel('fake abs') # these values are absolutely fake!
plt.xlabel('time (s)')
```

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

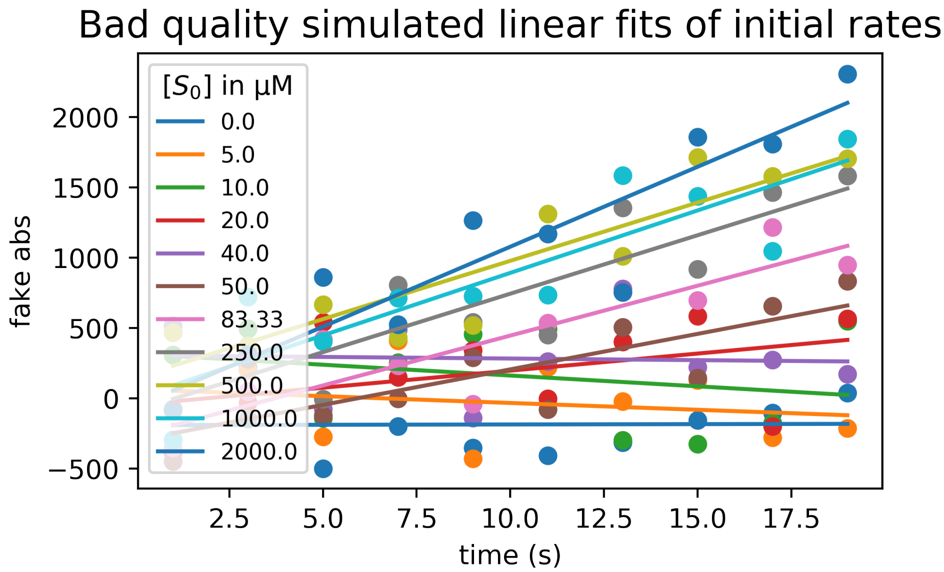


But if the noise increases, the fit with only 10 datapoints looks nasty:

[11]:
```
# 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
```

```
# try this:
time = np.arange(1, 20, 2) # initial values: (1, 20, 2): Start = 1, Stop = 20,␣
↪Step Size = 2
noiselevelL = 1000 #initial value: 1000

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

for s in S0:
    v0 = michaelismenten_equation(s,Km,vmax)
    data = linear(time,v0,0.001)
    #adding some artificial, random noise
    noise = noiselevelL*(np.random.random(len(time))-0.5)
    data_random = data + noise
    plt.scatter(time,data_random)
    popt, pcov = curve_fit(linear, time, data_random)
    rate = popt[0]
    b = popt[1]
    plt.plot(time, linear(time,rate,b))

plt.legend(S0, title="[$S_{0}$] in  M ", loc="upper left", fontsize='small',␣
↪fancybox=True)
plt.title("Bad quality simulated linear fits of initial rates", fontsize=14)
plt.ylabel('fake abs') # these values are absolutely fake!
plt.xlabel('time (s)')
```

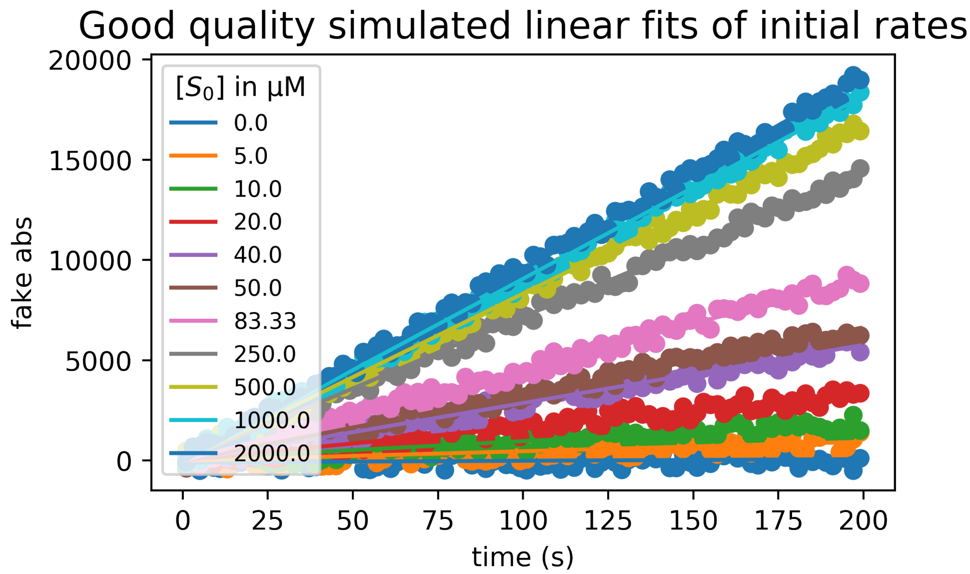
[11]: Text(0.5, 0, 'time (s)')



So it is a good idea to increase the number of datapoints,
i.e. measuring more frequently or increasing the time of the measurement.

[12]:
```
# 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

# .... and this:
time = np.arange(1, 200, 2) # initial values: (1, 200, 2): Start = 1, Stop =␣
↪200, Step Size = 2
noiselevelL = 1000 #initial value: 1000

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

for s in S0:
    v0 = michaelismenten_equation(s,Km,vmax)
    data = linear(time,v0,0.001)
    #adding some artificial, random noise
    noise = noiselevelL*(np.random.random(len(time))-0.5)
    data_random = data + noise
    plt.scatter(time,data_random)
    popt, pcov = curve_fit(linear, time, data_random)
    rate = popt[0]
    b = popt[1]
    plt.plot(time, linear(time,rate,b))

plt.legend(S0, title="[$S_{0}$] in  M ", loc="upper left", fontsize='small',␣
↪fancybox=True)
plt.title("Good quality simulated linear fits of initial rates", fontsize=14)
plt.ylabel('fake abs') # these values are absolutely fake!
plt.xlabel('time (s)')
```

[12]:
```
Text(0.5, 0, 'time (s)')
```


Good quality simulated linear fits of initial rates
fake abs
time (s)
[S_0] in µM
0.0
5.0
10.0
20.0
40.0
50.0
83.33
250.0
500.0
1000.0
2000.0
20000
15000
10000
5000
0
0
25
50
75
100
125
150
175
200

```
[ ]:
```

```
[ ]:
```