Practical Kinetics

Exercise 4:

Reaction Progress Analysis of a Catalytic Reaction

Objectives:

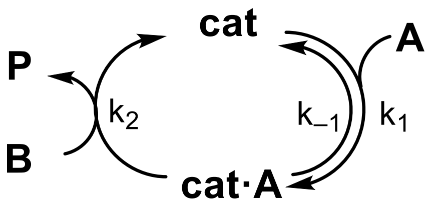
1. Same Excess

2. Different Excess

3. Extracting Rate Constants

Introduction

In this exercise, we will examine a concentration vs. time dataset for a hypothetical catalytic reaction (Michaelis–Menten):



1. What is the order in A and B?

2. What are the various rate constants?

In `dataset3.csv`, you will find 4 experimental runs: P1, P2, P3, P4. Each abbreviation represents the product concentration from its respective run.

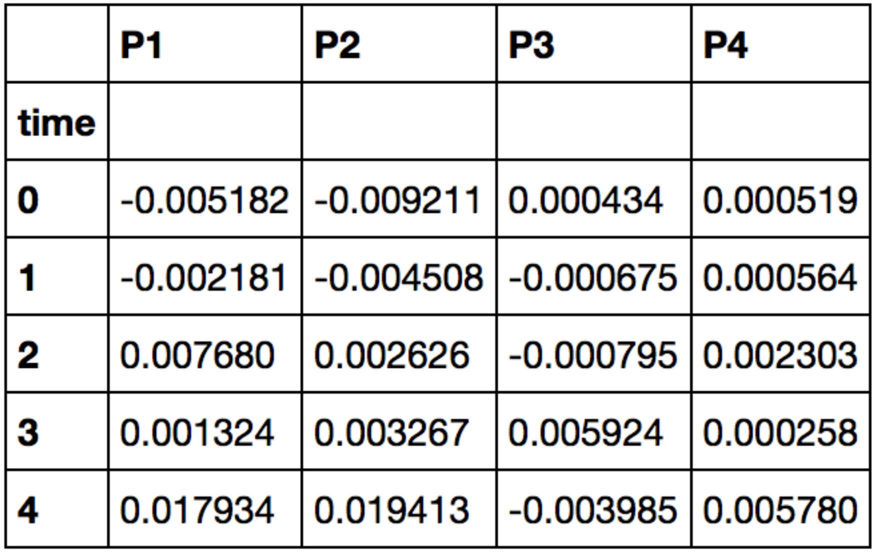
Defining the excess (*e*) as [B]–[A]:

Run #	$[A]_0$	$[B]_0$	Excess
1	1.0	1.2	+0.2
2	1.0	1.0	+0.0
3	1.0	0.8	–0.2
4	0.8	0.8	+0.0

Step 1: Import Data

Use pandas to import the data:

```
df = pd.read_csv("dataset3.csv")
df.set_index("time",inplace=True)
time = df.index
df.head()
```



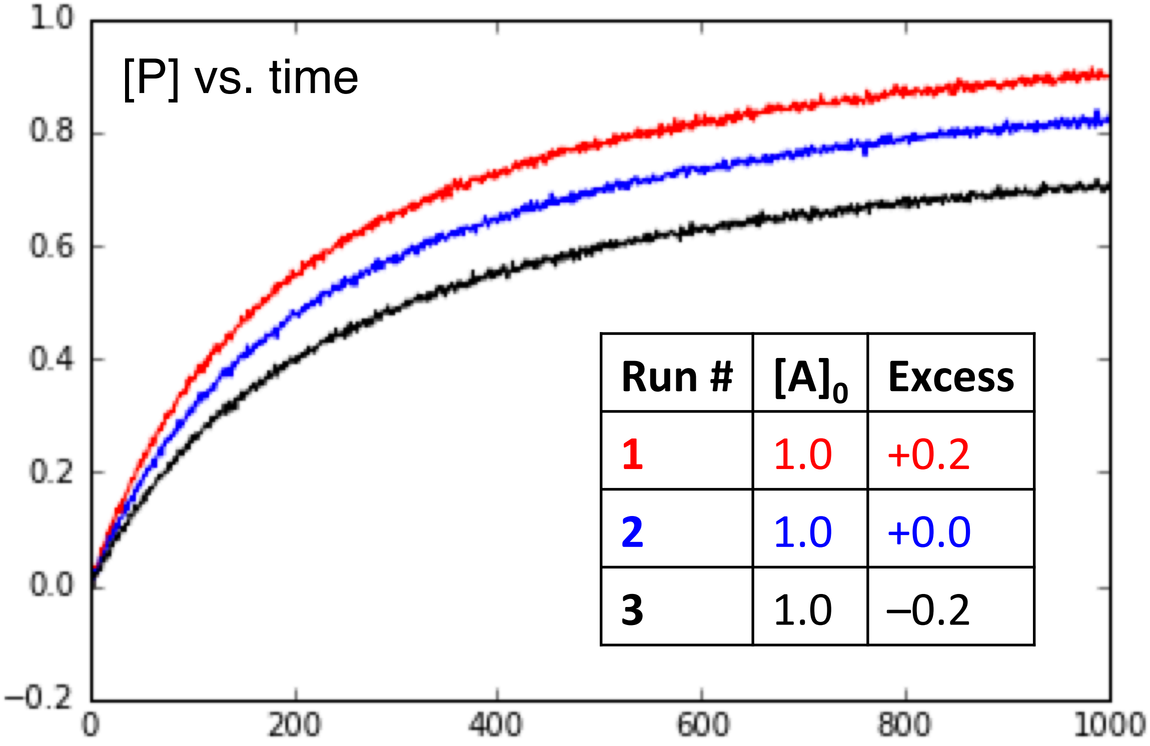
	P1	P2	P3	P4
time				
0	-0.005182	-0.009211	0.000434	0.000519
1	-0.002181	-0.004508	-0.000675	0.000564
2	0.007680	0.002626	-0.000795	0.002303
3	0.001324	0.003267	0.005924	0.000258
4	0.017934	0.019413	-0.003985	0.005780

1000 seconds of data are present.

Plot the first three runs:

```
plt.plot(time, df.P1, "r")
plt.plot(time, df.P2, "b")
plt.plot(time, df.P3, "k")
plt.show()
```

Bigger excesses give faster reactions, but the maximum amount of product is always 1.0 M.



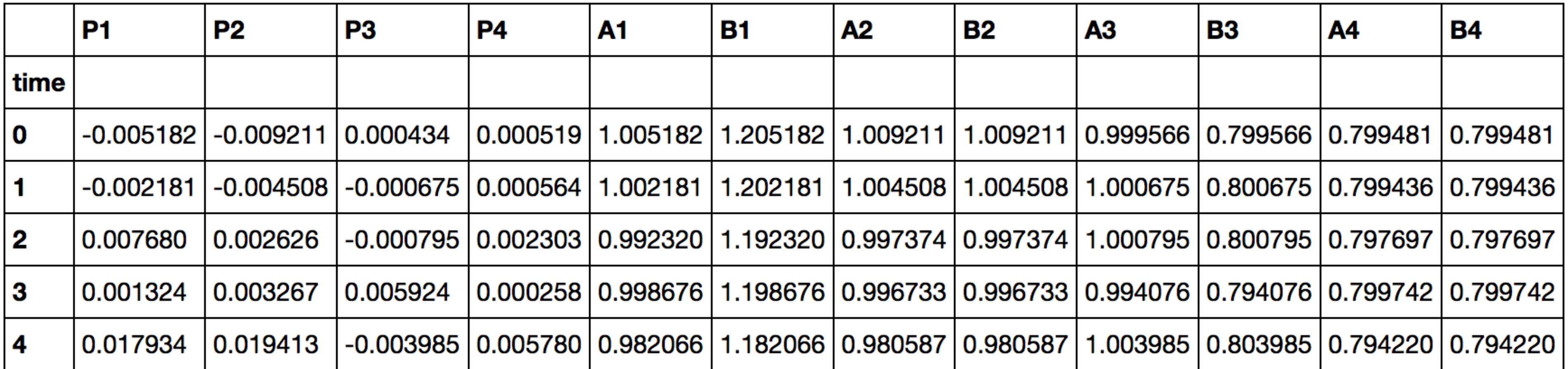
Run #	$[A]_0$	Excess
1	1.0	+0.2
2	1.0	+0.0
3	1.0	–0.2

Step 2: Infer Starting Material Concentrations

Assuming there are no side reactions, we can infer [A] and [B] at all times from [P]. (Alternatively, we could try to measure those quantities directly.)

Make new DataFrame columns for A and B (A1, B1, A2, B2, etc.):

```
def estimate_SM(index, initial_A_concentration, excess):
    P = df["P%d" % index]
    A = initial_A_concentration - P
    B = A + excess
    df["A%d" % index] = Series(A, index = time)
    df["B%d" % index] = Series(B, index = time)

estimate_SM(1, 1.0,  0.2)
estimate_SM(2, 1.0,  0.0)
estimate_SM(3, 1.0, -0.2)
estimate_SM(4, 0.8,  0.0)
df.head()
```



	P1	**P2**	**P3**	**P4**	**A1**	**B1**	**A2**	**B2**	**A3**	**B3**	**A4**	**B4**
time												
0	-0.005182	-0.009211	0.000434	0.000519	1.005182	1.205182	1.009211	1.009211	0.999566	0.799566	0.799481	0.799481
1	-0.002181	-0.004508	-0.000675	0.000564	1.002181	1.202181	1.004508	1.004508	1.000675	0.800675	0.799436	0.799436
2	0.007680	0.002626	-0.000795	0.002303	0.992320	1.192320	0.997374	0.997374	1.000795	0.800795	0.797697	0.797697
3	0.001324	0.003267	0.005924	0.000258	0.998676	1.198676	0.996733	0.996733	0.994076	0.794076	0.799742	0.799742
4	0.017934	0.019413	-0.003985	0.005780	0.982066	1.182066	0.980587	0.980587	1.003985	0.803985	0.794220	0.794220

Step 3: Compute Rate

Now, use polynomial fitting to get at the rate. Put it in columns P1, P2, P3, P4:

```
polynomial_order = 15

def estimate_rate(index):
    concentration = df["P%d" % index]
    poly_coeff = np.polyfit(time, concentration, polynomial_order)
    polynomial = np.poly1d(poly_coeff)
    fitted_concentration = polynomial(time)
    derivative = np.polyder(polynomial)
    rate_vector = derivative(time)
    df["rate%d" % (i+1)]=Series(rate_vector, index=time)
    return rate_vector

for i in range(4):
    rate_vector = estimate_rate(i+1)

df.head()
```

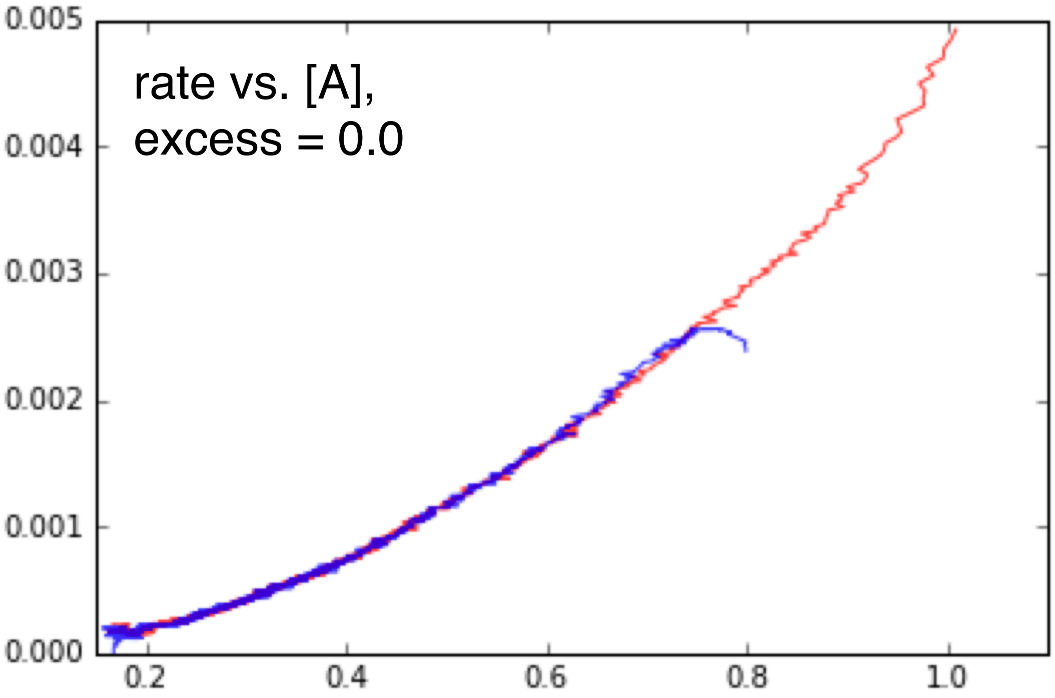
P3	P4	A1	B1	A2	B2	A3	B3	A4	B4	rate1	rate2	rate3	rate4
0.000434	0.000519	1.005182	1.205182	1.009211	1.009211	0.999566	0.799566	0.799481	0.799481	0.005671	0.004920	0.003605	0.002385
-0.000675	0.000564	1.002181	1.202181	1.004508	1.004508	1.000675	0.800675	0.799436	0.799436	0.005590	0.004842	0.003570	0.002411
-0.000795	0.002303	0.992320	1.192320	0.997374	0.997374	1.000795	0.800795	0.797697	0.797697	0.005513	0.004767	0.003536	0.002436
0.005924	0.000258	0.998676	1.198676	0.996733	0.996733	0.994076	0.794076	0.799742	0.799742	0.005438	0.004695	0.003502	0.002457
-0.003985	0.005780	0.982066	1.182066	0.980587	0.980587	1.003985	0.803985	0.794220	0.794220	0.005366	0.004626	0.003470	0.002477

Step 4: Same Excess Analysis

Plot rate vs. concentration for runs 2 and 4, which have the same excess:

```
plt.plot(df.A2, df.rate2, "r", alpha=0.75)
plt.plot(df.A4, df.rate4, "b", alpha=0.75)
plt.xlim(0.15,1.1)
plt.ylim(0.0,0.005)
plt.show()
```

`alpha` sets the transparency of each line.



Again, despite some edge artifacts, the overlay is very good.

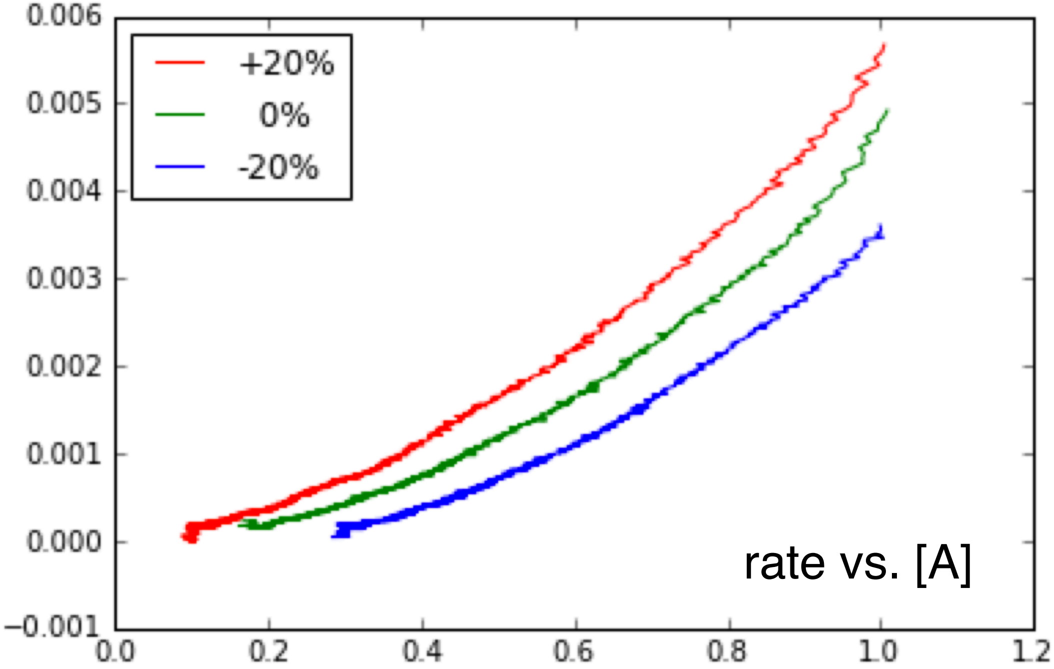
This indicates that there is no catalyst deactivation or product inhibition.

As to our earlier assumption about no side reactions: if there are any, they have the same stoichiometry as the catalytic reaction. We could verify this by measuring A and B independently and examining the mass balance in P.

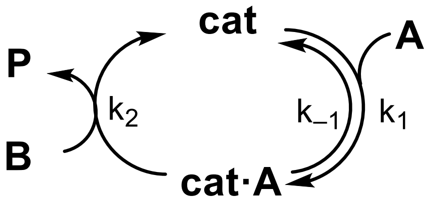
Step 5: Different Excess Analysis

Now, let’s plot the different excess data:

```
plt.plot(df.A1, df.rate1, "r", label="+20%")
plt.plot(df.A2, df.rate2, "g", label="  0%")
plt.plot(df.A3, df.rate3, "b", label="-20%")
plt.legend(loc="best")
plt.show()
```



Larger excesses give bigger rates.



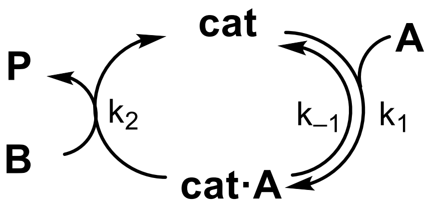
The 1+rate law is:

$$rate = \frac{k_2 K[A][B][cat]_T}{1 + K[A]}$$

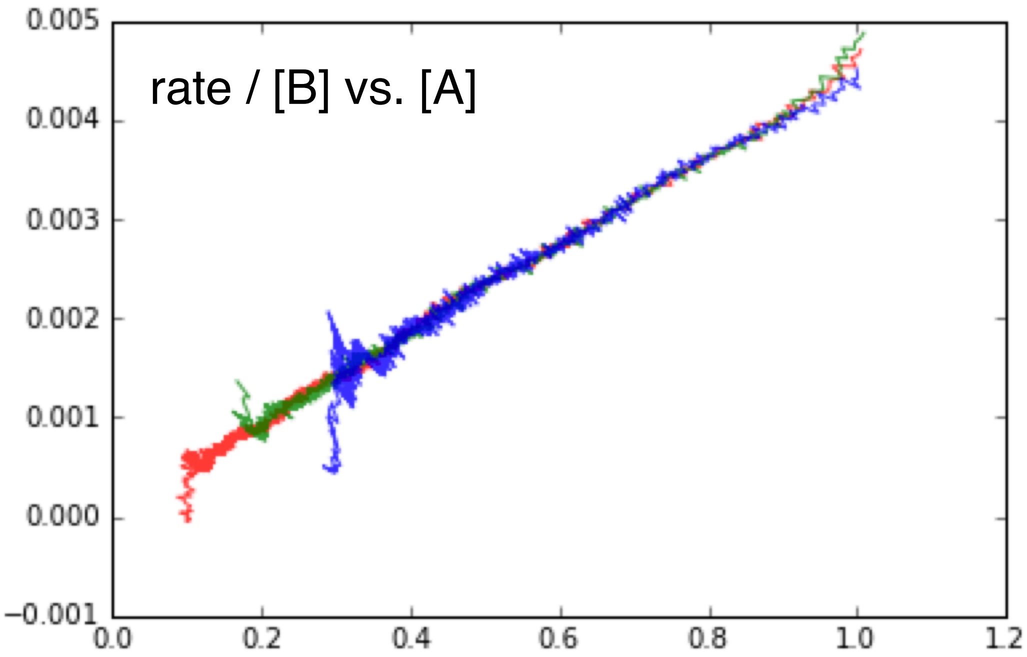
What happens if we plot rate/[B] vs. [A]?

Step 6: Different Excess Analysis

Rate / [B] vs. [A]:



```
plt.plot(df.A1,df.rate1/df.B1,"r", alpha=0.75)
plt.plot(df.A2,df.rate2/df.B2,"g", alpha=0.75)
plt.plot(df.A3,df.rate3/df.B3,"b", alpha=0.75)
plt.show()
```



$$rate = \frac{k_2 K[A][B][cat]_T}{1+K[A]}$$

$$\frac{rate}{[B]} = \frac{k_2 K[A][cat]_T}{1+K[A]}$$

$$\approx \frac{k_2 K[A][cat]_T}{1}$$

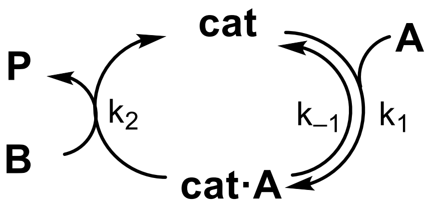
$$= m[A]$$

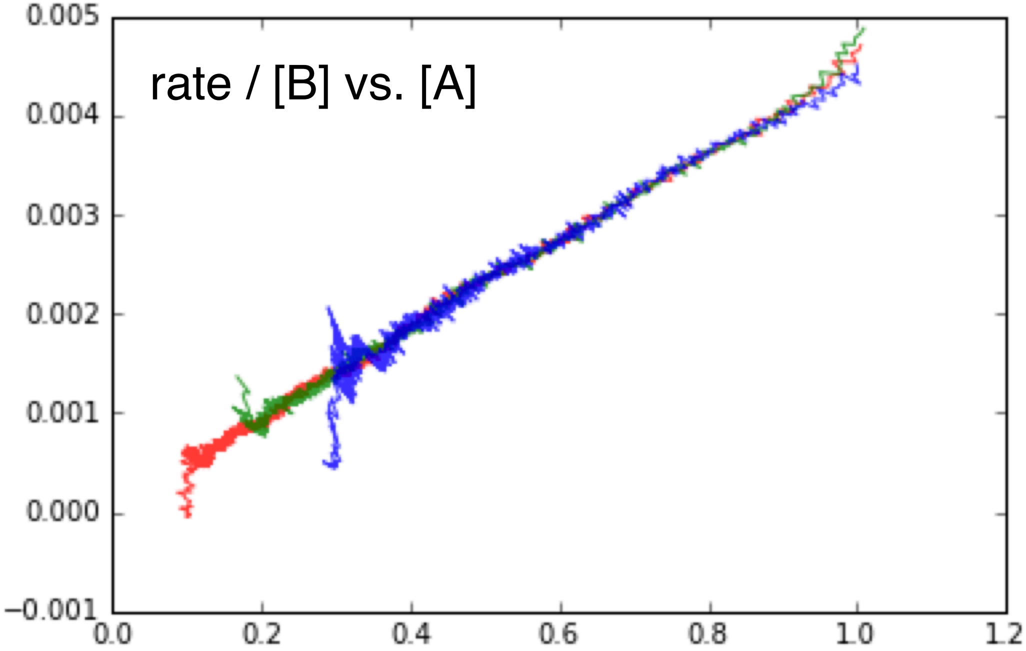
If the resting state is free catalyst ([cat] ≈ $[cat]_T$), the denominator ≈ 1 and we get straight lines whose intercept depends on excess. The slope does not depend on excess, so everything overlays, consistent with the graph.

Step 6: Different Excess Analysis

Rate / [B] vs. [A]:

```
plt.plot(df.A1,df.rate1/df.B1,"r", alpha=0.75)
plt.plot(df.A2,df.rate2/df.B2,"g", alpha=0.75)
plt.plot(df.A3,df.rate3/df.B3,"b", alpha=0.75)
plt.show()
```





$$rate = \frac{k_2 K[A][B][cat]_T}{1+K[A]}$$

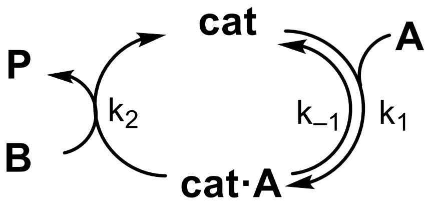
$$\frac{rate}{[B]} = \frac{k_2 K[A][cat]_T}{1+K[A]}$$

$$\approx \frac{k_2 K[A][cat]_T}{K[A]}$$

$$= const$$

If the resting state is bound catalyst ([cat•A] ≈ $[cat]_T$), the denominator ≈ K[A] and we would expect horizontal lines that overlay because the constant is not a function of excess. This is **not** observed here.

Step 5: Different Excess Analysis

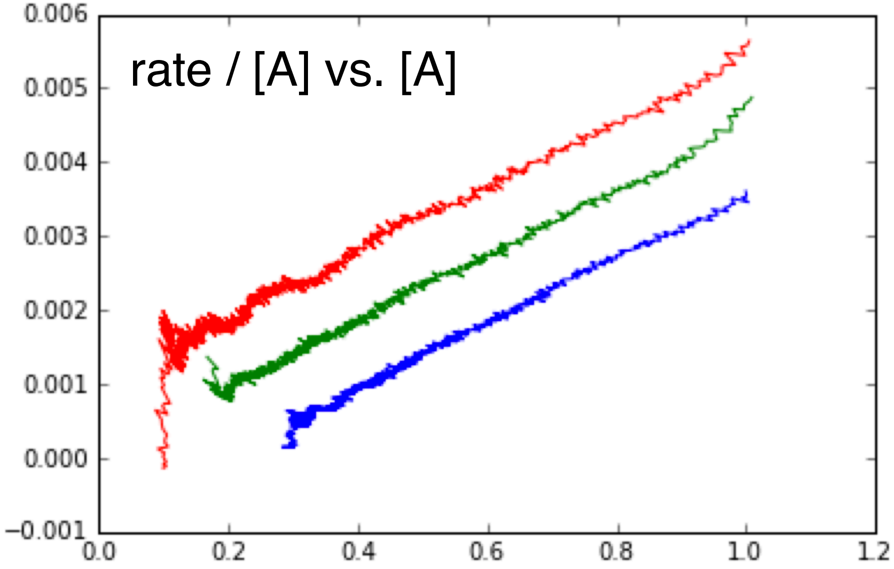


Rate / [A] vs. [A]:

```
plt.plot(df.A1,df.rate1/df.A1,"r")
plt.plot(df.A2,df.rate2/df.A2,"g")
plt.plot(df.A3,df.rate3/df.A3,"b")
plt.show()
```

Recall that [B] – [A] = excess (*e*).

If free catalyst is dominant, the denominator ≈ 1 and we get straight lines whose intercept depends on excess. This **is** what we observe:



$$rate = \frac{k_2K[A][B][cat]_T}{1+K[A]}$$

$$\frac{rate}{[A]} = \frac{k_2K[B][cat]_T}{1+K[A]}$$

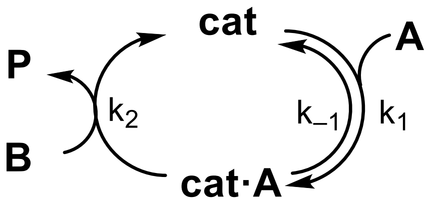
$$= \frac{k_2K\left([A]+e\right)[cat]_T}{1+K[A]}$$

$$= \frac{k_2K[cat]_T[A]+k_2K[cat]_Te}{1+K[A]}$$

$$= \frac{k_2K[cat]_T[A]+k_2K[cat]_Te}{1}$$

$$= m[A]+b(e)$$

Step 5: Different Excess Analysis



Rate / [A] vs. [A]:

```
plt.plot(df.A1,df.rate1/df.A1,"r")
plt.plot(df.A2,df.rate2/df.A2,"g")
plt.plot(df.A3,df.rate3/df.A3,"b")
plt.show()
```

$$rate = \frac{k_2 K[A][B][cat]_T}{1+K[A]}$$

$$\frac{rate}{[A]} = \frac{k_2 K[B][cat]_T}{1+K[A]}$$

$$= \frac{k_2 K\left([A]+e\right)[cat]_T}{1+K[A]}$$

$$= \frac{k_2 K[cat]_T[A]+k_2 K[cat]_T e}{1+K[A]}$$

$$= \frac{k_2 K[cat]_T[A]+k_2 K[cat]_T e}{K[A]}$$

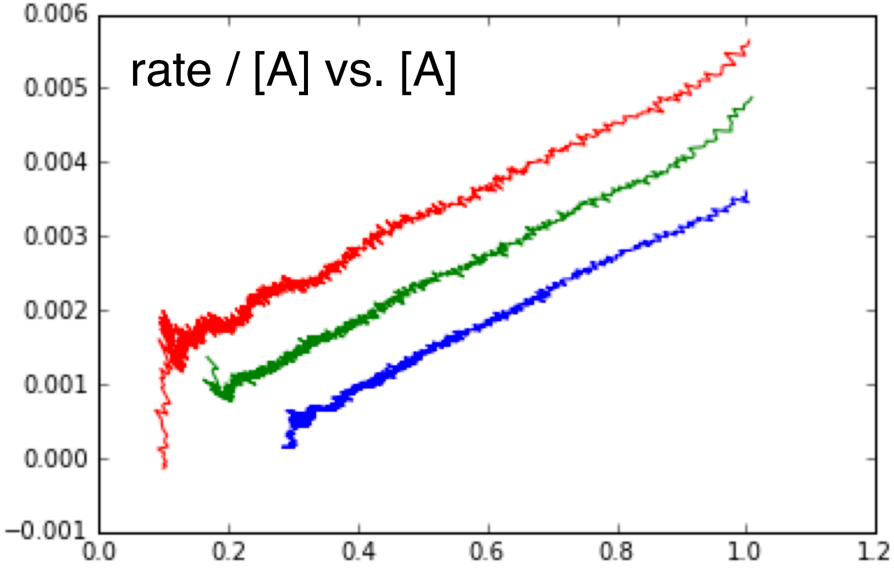
$$= k_2[cat]_T + \frac{k_2[cat]_T e}{[A]}$$

$$= b + \frac{c(e)}{[A]}$$

Recall that [B] – [A] = excess (*e*).

If bound catalyst is dominant, the denominator ≈ K[A]. We would expect saturation behavior in [A], with the curvature depending on excess.

This is **not** observed here.

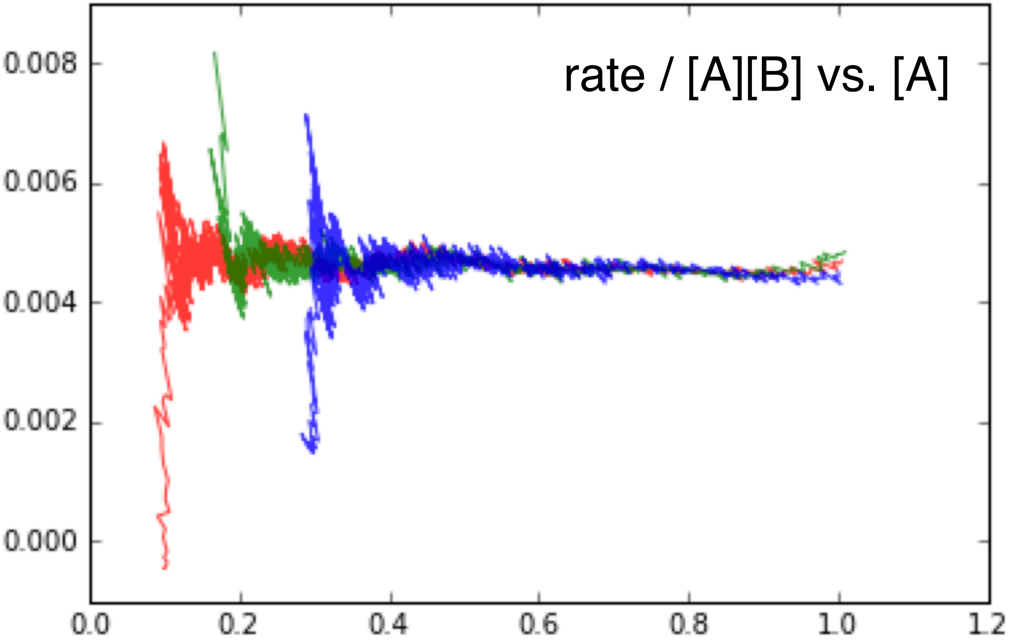


Step 5: Different Excess Analysis

Plot rate / [A][B] vs. [A]:

```
plt.plot(df.A1,df.rate1/df.A1/df.B1,"r", alpha=0.75)
plt.plot(df.A2,df.rate2/df.A2/df.B2,"g", alpha=0.75)
plt.plot(df.A3,df.rate3/df.A3/df.B3,"b", alpha=0.75)
plt.show()
```

Now we get horizontal, overlaying lines:



$$rate = \frac{k_2 K[A][B][cat]_T}{1+K[A]}$$

$$\frac{rate}{[A][B]} = \frac{k_2 K[cat]_T}{1+K[A]}$$

When the resting state is free catalyst ([cat] ≈ $[cat]_T$), we expect this to be a constant that does not depend on excess.

This is what we observe here.

Overall, the reaction is essentially first order in [A] and [B].

Step 6: Three Parameter Fit

To extract the rate constants, we can fit to the steady state rate law:

$$rate = \frac{k_1 k_2 [A][B][cat]_T}{k_{-1} + k_2[B] + k_1[A]}$$

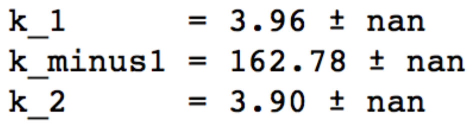
```
total_catalyst = 0.05

def make_rate_law(excess):
    def rate_law(A, k_1, k_minus1, k_2):
        B = A + excess
        numerator = k_1 * k_2 * A * B * total_catalyst
        denominator = k_minus1 + k_2*B + k_1*A
        return numerator / denominator
    return rate_law

rate_law = make_rate_law(0.20)
popt,pcov = curve_fit(rate_law, df.A1, df.rate1)
errors = np.sqrt(np.diag(pcov))
print "k_1      = %.2f ± %.2f" % (popt[0], errors[0])
print "k_minus1 = %.2f ± %.2f" % (popt[1], errors[1])
print "k_2      = %.2f ± %.2f" % (popt[2], errors[2])
fitted_rate = rate_law(df.A1, popt[0], popt[1], popt[2])
plt.plot(df.A1,df.rate1,"b")
plt.plot(df.A1,fitted_rate,"r")
plt.show()
```

This is the same curve fitting strategy we used in previous exercises.

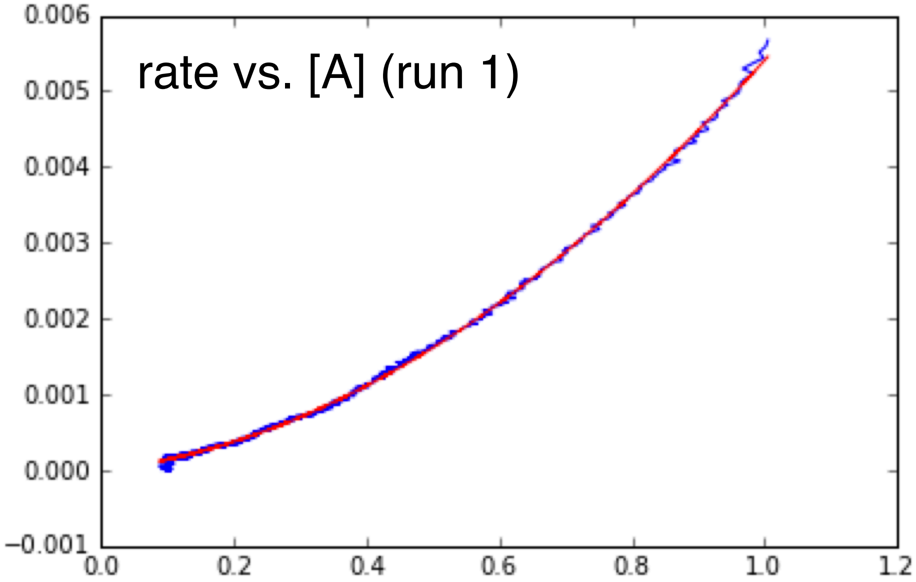
Technical Note: We nest two functions to prevent the excess from being treated as a parameter for optimization. This strategy is called creating a “closure” or “lambda.”

Step 6: Three Parameter Fit

The fit is very good, and the extracted parameters agree with the actual values of $k_1 = 10$, $k_{-1} = 100$, and $k_2 = 1$:



```
k_1        = 3.96 ± nan
k_minus1 = 162.78 ± nan
k_2        = 3.90 ± nan
```



$$rate = \frac{k_1 k_2 [A][B][cat]_T}{k_{-1} + k_2[B] + k_1[A]}$$

However, the error bars are "not a number"! This suggests that the data might be nearly two-dimensional, even though the fit is three-dimensional.

This is consistent with the fact that this is a pre-equilibrium scenario, but we are fitting to a steady state rate law. When the actual k_{-1} is so large, fitting adjustments in k_1 are indistinguishable from inverse adjustments in k_{-1}.

Step 8: Fitting Each Dataset

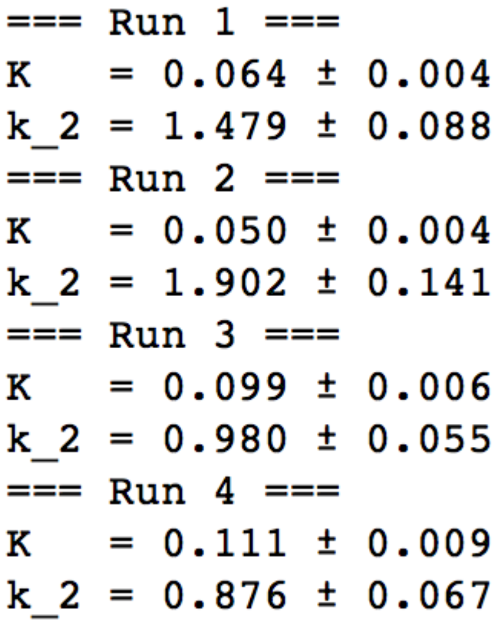
Since we know the rate data are poor near the edges of each run, we can cut off five points from the beginning and end of each dataset and then fit:

```
cutoff = 5

def fit_constants(index, excess):
    rate_law = make_rate_law(excess)
    x = np.array(df["A%d" % index])[cutoff:-cutoff]
    y = np.array(df["rate%d" % index])[cutoff:-cutoff]
    popt,pcov = curve_fit(rate_law, x, y)
    errors = np.sqrt(np.diag(pcov))
    print "=== Run %d ===" % index
    print "K   = %.3f ± %.3f" % (popt[0], errors[0])
    print "k_2 = %.3f ± %.3f" % (popt[1], errors[1])
    fitted_rate = rate_law(x, popt[0], popt[1])
    #plt.plot(x, y, "b")
    #plt.plot(x, fitted_rate, "r")
    #plt.show()

fit_constants(1, 0.2)
fit_constants(2, 0.0)
fit_constants(3, -0.2)
fit_constants(4, 0.0)
```

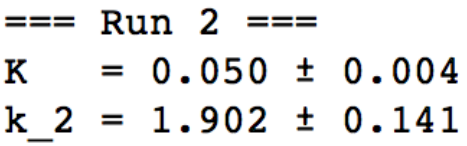
To view each fit, uncomment the plotting lines above by removing the # signs.

Step 7: Two Parameter Fit

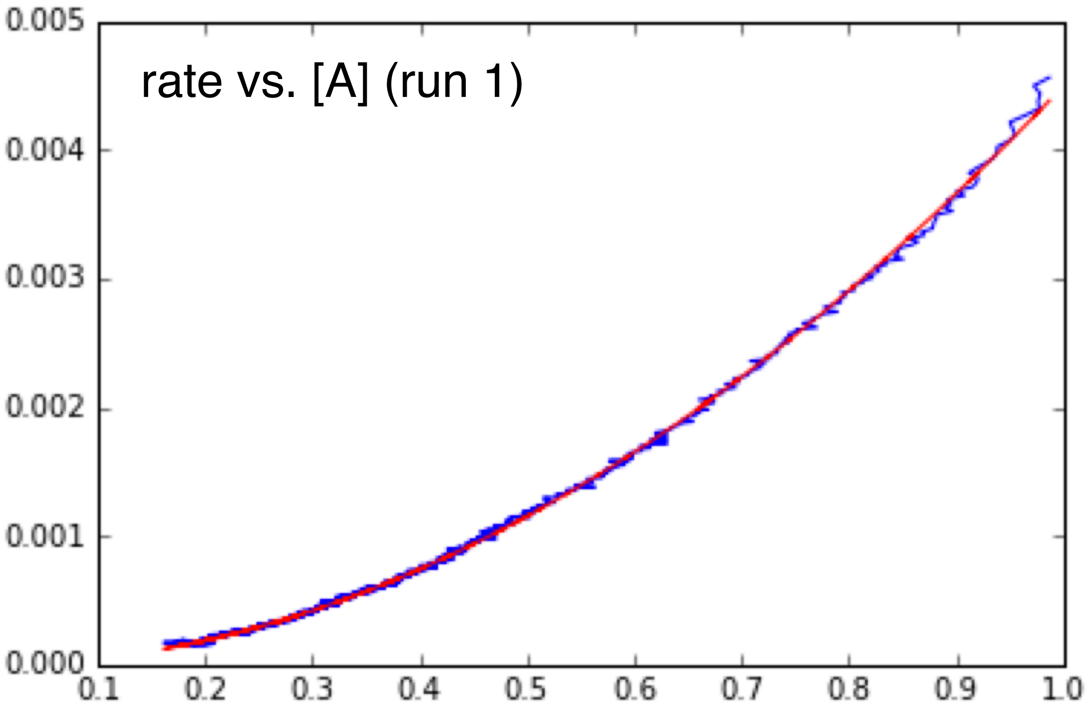
The results are in good agreement with the actual parameters:



```
=== Run 1 ===
K   = 0.064 ± 0.004
k_2 = 1.479 ± 0.088
=== Run 2 ===
K   = 0.050 ± 0.004
k_2 = 1.902 ± 0.141
=== Run 3 ===
K   = 0.099 ± 0.006
k_2 = 0.980 ± 0.055
=== Run 4 ===
K   = 0.111 ± 0.009
k_2 = 0.876 ± 0.067
```



```
=== Run 2 ===
K   = 0.050 ± 0.004
k_2 = 1.902 ± 0.141
```



(To fit all of the data at once, we would need to use a different fitting package that supports multiple indepedent variables, since rate is a function of both concentration and excess.)

Summary

(1) Read a CSV:

```
import pandas as pd
df = pd.read_csv("dataset3.csv")
df.set_index("time",inplace=True)
time = df.index
df.head()
```

(2) Infer starting material concentrations from excess:

```
def estimate_SM(index, initial_A_concentration, excess):
    P = df["P%d" % index]
    A = initial_A_concentration - P
    B = A + excess
    df["A%d" % index] = Series(A, index = time)
    df["B%d" % index] = Series(B, index = time)

estimate_SM(1, 1.0,  0.2)
estimate_SM(2, 1.0,  0.0)
estimate_SM(3, 1.0, -0.2)
estimate_SM(4, 0.8,  0.0)
df.head()
```

Summary

(3) Fit to a polynomial and take the derivative to get rate:

```
polynomial_order = 15

def estimate_rate(index):
    concentration = df["P%d" % index]
    poly_coeff = np.polyfit(time, concentration, polynomial_order)
    polynomial = np.poly1d(poly_coeff)
    fitted_concentration = polynomial(time)
    derivative = np.polyder(polynomial)
    rate_vector = derivative(time)
    df["rate%d" % (i+1)]=Series(rate_vector, index=time)
    return rate_vector

for i in range(4):
    rate_vector = estimate_rate(i+1)
```

The order of the polynomial should be the smallest possible that gives consistent rates across all the experiments.

(4) Same/Different Excess Plots:

```
plt.plot(df.A2, df.rate2/df.B2, "r", alpha=0.75)
plt.plot(df.A4, df.rate4/df.B4, "b", alpha=0.75)
plt.xlim(0.15,1.1)
plt.ylim(0.0,0.005)
plt.show()
```

Summary

(5) Fit runs to a steady state rate law:

```
cutoff = 5

def fit_constants(index, excess):
    rate_law = make_rate_law(excess)
    x = np.array(df["A%d" % index])[cutoff:-cutoff]
    y = np.array(df["rate%d" % index])[cutoff:-cutoff]
    popt,pcov = curve_fit(rate_law, x, y)
    errors = np.sqrt(np.diag(pcov))
    print "=== Run %d ===" % index
    print "K   = %.3f ± %.3f" % (popt[0], errors[0])
    print "k_2 = %.3f ± %.3f" % (popt[1], errors[1])
    fitted_rate = rate_law(x, popt[0], popt[1])
    #plt.plot(x, y, "b")
    #plt.plot(x, fitted_rate, "r")
    #plt.show()

fit_constants(1, 0.2)
fit_constants(2, 0.0)
fit_constants(3, -0.2)
fit_constants(4, 0.0)
```