# Chemical Reaction Engineering - Part 8 - more data analysis

*Richard K. Herz, rherz@ucsd.edu, ReactorLab.net*

Gases reacting - using change in pressure or volume

When we have gases reacting, we may be able to use measurements of total pressure or volume to determine conversion. How? By writing a stoichiometric table and the ideal gas law (or nonideal equation of state if required). Then we can use conversion vs. time data with the integral or differential methods. Be sure to include any inert gases in the stoichiometric table.

Method of excess - estimating orders for multiple components

Consider the reaction A+B → products where we suspect that both components A and B are involved in the rate equation.

$$r_A = -k C_A^{\alpha} C_B^{\beta}$$

We can isolate the effect of one component by starting with a mixture where all other components are present in much greater concentrations.

For example, start with a mixture where $C_{B0} >> C_{A0}$. The percent change in $C_B$ will be much smaller than the percent change in $C_A$, so $C_B$ will be approximately constant at $C_{B0}$.

$$r_A \approx -[k C_{B0}^{\beta}] C_A^{\alpha} = -k' C_A^{\alpha}$$

Then use one of the techniques we learned earlier to estimate the order in A and the rate constant. Then do experiments with A in excess.

$$r_A \approx -[k C_{A0}^{\alpha}] C_B^{\beta} = -k'' C_B^{\beta}$$

Remember that product components may also appear in rate equations, as well as catalyst concentrations and concentrations of other components.

Method of initial rates

By making a measurement at low conversion, we can estimate the initial reaction rate.

$$\frac{dN_A}{dt} = r_A V$$

$$\left[\frac{dN_A}{dt}\right]_{t=0} \approx \frac{N_A - N_{A0}}{t-0}$$

$$\left[\frac{dC_A}{dt}\right]_{t=0} \approx \frac{C_A - C_{A0}}{t-0}$$ since $V$ will be approximately constant at low conversion

By starting at different values of $C_{A0}$ and plotting the estimate of the initial rate vs. $C_{A0}$, we can estimate the order of the reaction in A.

The method of initial rates can be combined with the method of excess when there are multiple components in the rate equation.

The method of initial rates may be useful in "reversible" reactions where the equilibrium conversion is not close to 100%. For example, for A + B = C + D, you can do the method of initial rates starting with only A and B in combination with the method of excess. Then you can wait until the reaction approaches equilibrium. You may also be able to repeat starting with only C and D. Remember that the forward and back rate equations have to be consistent with the equilibrium relationship ($K_{eq,c}$ = ...) in order for them to apply over the entire range of conversion.

Method of half life

Consider a 1st-order, essentially irreversible reaction in a constant volume reactor.

Record the "half life," which is the time $t_{1/2}$ at which the reactant concentration has dropped to one-half of it's original value.

$$C_A = \frac{C_{A0}}{2} = C_{A0} e^{-k t_{1/2}}$$

Solve for the rate constant

$$k = \frac{-\ln(1/2)}{t_{1/2}} = \frac{0.693}{t_{1/2}}$$

You may see this method used in a chemistry lab with a reaction in the cuvette of a spectrophotometer and a person with a stopwatch looking at the meter reading for absorbance or transmitted intensity.

Note that you don't have to use half life. You could also use one-third life. Usually you hear about the half life method.

You should check the assumption of 1st-order reaction by doing experiments at different values of $C_{A0}$. The value estimated for $k$ will change as $C_{A0}$ is changed if the reaction is not 1st-order.

When you don't have a 1st-order reaction. Try this,

$$r_A = -k C_A^{\alpha} \quad \text{for } \alpha \neq 1$$

$$t_{1/2} = \frac{0.5^{1-\alpha} - 1}{k(\alpha - 1)} C_{A0}^{1-\alpha} \qquad \text{for constant } V$$

$$\ln\left(t_{1/2}\right)=\ln\left(\frac{0.5^{1-\alpha}-1}{k(\alpha-1)}\right)+(1-\alpha)\ln\left(C_{A0}\right)$$

A plot of ln($t_{1/2}$) vs. ln($C_{A0}$) gives a straight line of slope (1 - α ), for α ≠ 1, if the original rate law that was assumed applies.

Methods of differentiating data

In the preceding set of notes, we differentiated data simply by getting the difference between each pair of points.

This should work on exams or in the "real world" where you only have a few data points.

Real data, however, will have some amount of random error or scatter or "noise" in the measurements. This leads to noise in the differentiated data, i.e., the estimates of reaction rate.

In the presence of significant scatter in the data, and where you have a relatively large number of data points, you should try this, for example, in a Matlab script:

- for *x* and *y* data "points" (x,y pair = a point)
- choose a number *N* substantially less that the number of data points
- choose a polynomial order *y* = f(*x*), usually of order 1 to 3
- starting with the first group of *N* points,
- repeat the following
  - fit a polynomial of the chosen order through the first group of points
  - calculate the value of the analytical derivative of the polynomial (*dy/dx*) at the midpoint *x*
  - record this value and the midpoint x value
  - drop the first data point in the group and add the next data point in the complete data set

The order of the polynomial should be low, 1 to 3, to prevent "threading a line" through each point in a group. The number of points *N* must be at least one more than the order of the polynomial.

The best values of order and *N* is the subject of a course in signal processing. Here we will just give guidelines. For this course, write a Matlab script, change the values, and see what happens.

For points that are equally spaced in x, you can use one of Simpson's rules to do the differentiation without having to fit a polynomial to each group. For equally spaced data, Simpson did the fitting and differentiation for you. Fitting a polynomial with Matlab's `polyfit()` function is just as easy, however. The `polyfit()` function does not need equally spaced data.

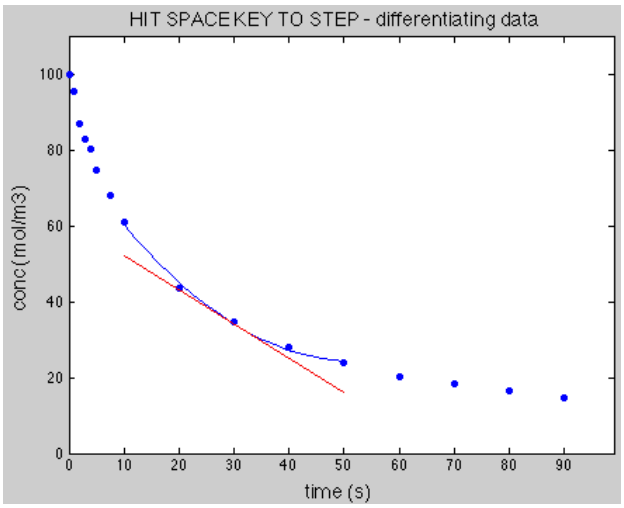
Example of differential method

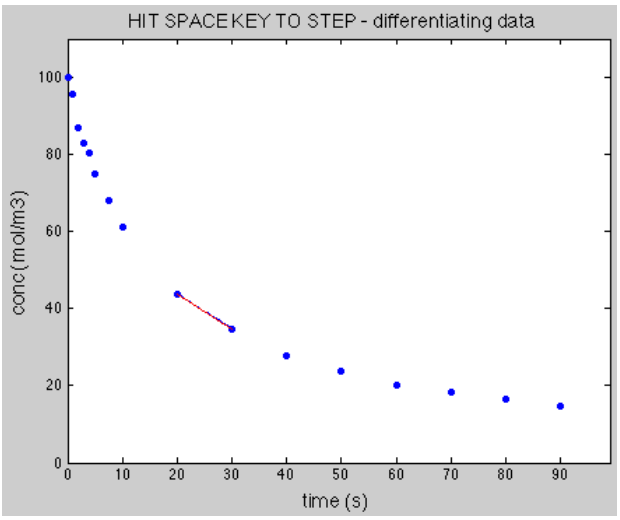
Data were taken in ReactorLab in Division 1, Lab 1, Batch. This is an isothermal, constant volume batch reactor. The reaction is A → B. A Matlab script was used to process the data. The Matlab script is listed below.

PLOTS ON THE LEFT side below had groups of 5 concentration-time points fit to a 2nd-order polynomial. Then the polynomial coefficients were used to compute the derivative $dC_A/dt$ for each group of 5 points. Each new group was formed by dropping the first point and adding an additional point at the end.

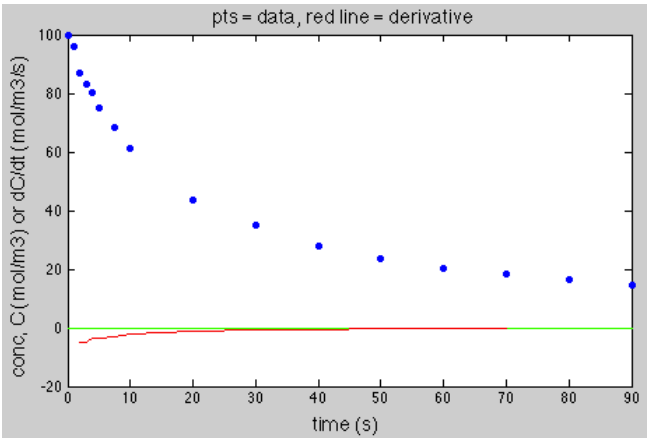
PLOTS ON THE RIGHT side below had groups of 2 points fit to a 1st-order polynomial (straight line). That is, the difference between neighboring points was used to estimate the derivative.

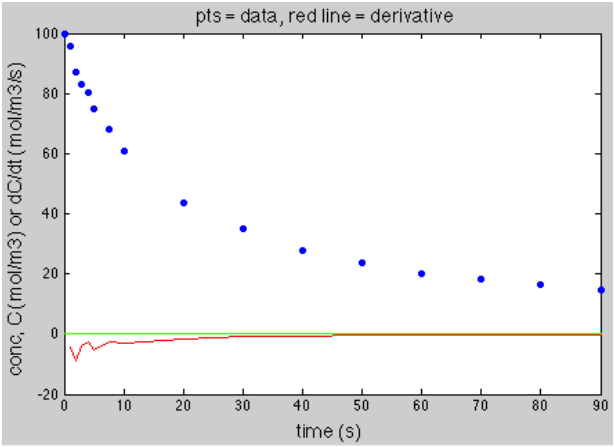
Here is a plot made during this process showing the polynomial fit and a straight line showing the slope $dC_A/dt$ at the midpoint of the group.



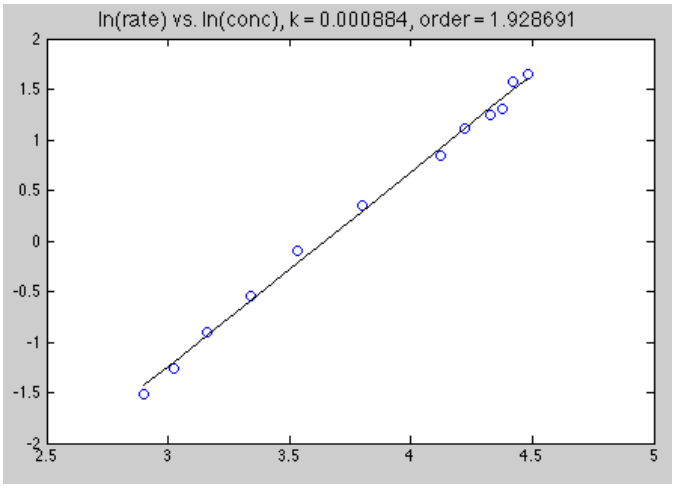



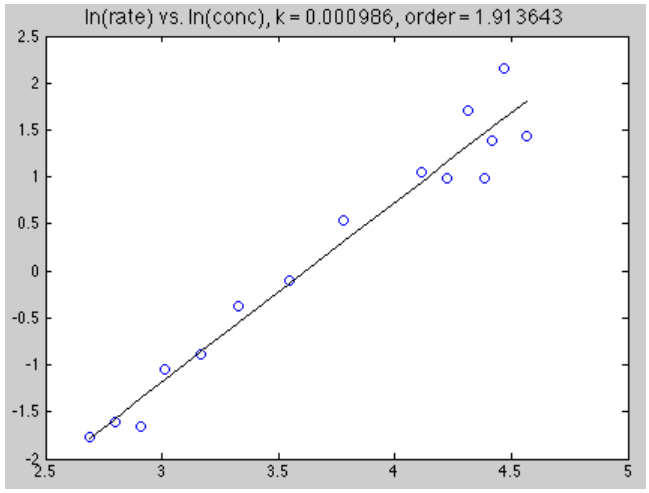

Here is a plot of the data points and a curve plotting the $dC_A/dt$ values. Note the greater "noise" in the derivative on the right. There is not much random error present in these data. The more error in your data, the more points you want to take and the more points you want in the groups you use to estimate the derivatives.



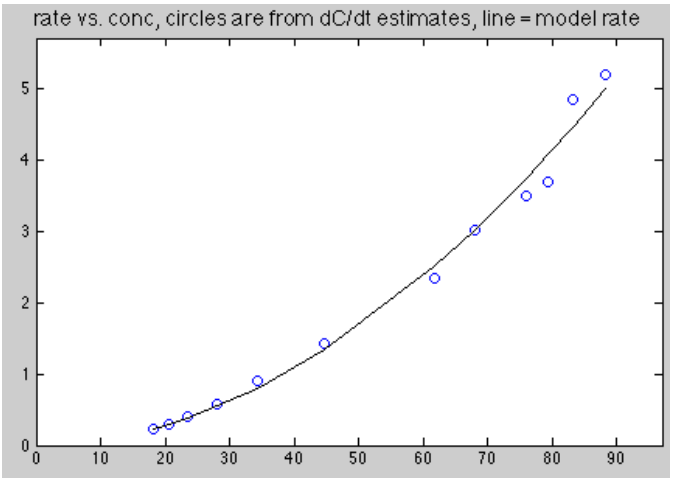



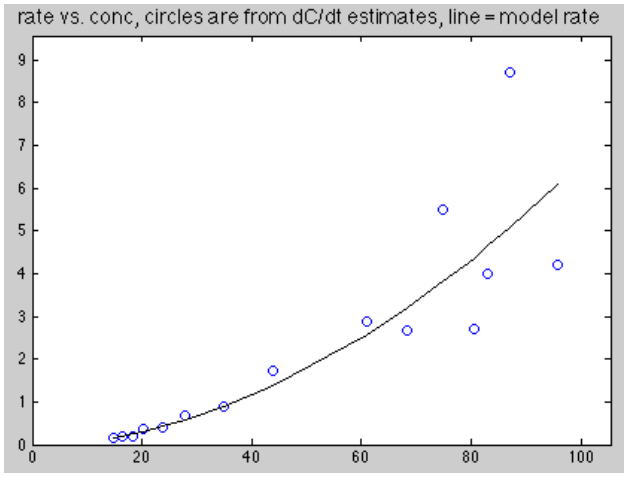
Here is plot of -ln(-$dC_A/dt$) vs ln($C_A$). The slope is an estimate of the order in $C_A$ as explained above, and the intercept gives an estimate of $k$.




Here is a plot of rate vs. concentration (at constant $V$, $dC_A/dt = r_A$). The points are from the estimates of -$dC_A/dt$ and the curve is the rate predicted from $-r_A = kC_A^n$ using the values of $k$ and order from the data analysis.




Here is the Matlab script

```
clear
fprintf('--------------------------\n') % run separator in command window
% D1L1 Batch, Data Set 2b
%
% k*_2b remained constant at    1.00E-4  (value at 300 K)
% Ea_2b remained constant at       60.0  kJ/mol
% n_2b remained constant at        2.00  (dimensionless)
% T_2b remained constant at       325.0  K
% Cin_2b remained constant at      100.0  mol/m3
% vol_2b remained constant at       1.00  m3
% time_2b has units of s
% tau_2b has units of s
% Cout_2b has units of mol/m3
% conv_2b has units of (dimensionless)
```

```
% time_2b   tau_2b  Cout_2b conv_2b
d = [0.00   0.00    100.0   0.00
1.00    1.00    95.8    4.21E-2
2.00    2.00    87.1    0.129
3.00    3.00    83.1    0.169
4.00    4.00    80.4    0.196
5.00    5.00    74.9    0.251
7.50    7.50    68.2    0.318
10.0    10.0    61.0    0.391
20.0    20.0    43.8    0.562
30.0    30.0    34.8    0.652
40.0    40.0    27.9    0.721
50.0    50.0    23.8    0.762
60.0    60.0    20.3    0.797
70.0    70.0    18.4    0.816
80.0    80.0    16.4    0.836
90.0    90.0    14.7    0.853];

% extract time and concentration vectors
t = d(:,1);
c = d(:,3);

% plot
plot(t,c,'.')
title('HIT SPACE KEY TO STEP - differentiating data')
ylabel('conc (mol/m3)')
xlabel('time (s)')

[rows cols] = size(c); % size is std matlab function
npts = rows;

%----- DIFFERENTIATE WITH GENERAL POLYNOMIAL FIT ----------

% We want to look at the rate of change of the concentration vs. t
% by differentiating these data.
% The simplest way is just to get the slope between each
% pair of points using Matlab's diff function.
% But this will show a very "noisy" derivative with these data.
% We can also fit a polynomial to groups of points
% to get some "smoothing" of the data.

% The following will also work with pts that are
% NOT equally spaced in time

% Vary n and m and see how the derivative on the plot changes
n = 2; % order of polynomial, reasonable numbers are 1-4
m = 5; % number of points in group to fit, m >= n+1
if m < n+1
    fprintf('you must increase m in the derivative polynomial fit! \n\n')
end

fprintf('see plot - hit return key to step \n')

% k = number of groups of m pts we can fit
% with dropping one pt and adding one pt for each group
% Note this grouping is different than for quadrature
k = npts-m+1;

for j = 1:k
    i = j;
    % use Matlab std function polyfit to fit n-th
    % order polynomial to this group of pts
    % with polynomial coefficients returned in array a
    % with highest-order term first, lowest last
```

```
    a = polyfit(t(i:i+m-1),c(i:i+m-1),n);

    % differentiate polynomial analytically over interval
    % and evaluate at approx. mid-point time t(i+floor(m/2))
    sumd = 0; % mol/m3/s
    for jj = 1:n
        sumd = sumd + ...
            a(jj)*(n-jj+1)*t(i+floor(m/2))^(n-jj);
    end

    % save derivative, time, and concentration values in arrays
    diffValue(j) = sumd; % mol/m3/s
    diffTime(j) = t(i+floor(m/2)); % s
    diffConc(j) = polyval(a,t(i+floor(m/2)));

    % OPTIONAL - SHOW PROCESS ON PLOT
    if i < 10 % just do this for first few groups
        % add the polynomial and the slope to the plot
        % get intercept b of the tangent line of the polynomial at the mid-pt
        b = polyval(a,t(i+floor(m/2))) - sumd*t(i+floor(m/2));
        % get start and end times
        tp(1) = t(i);
        tp(2) = t(i+m-1);
        % get y values of the tangent line at the start and end times
        yp = sumd*tp + b;
        % get conc values for the polynomial at each time
        tpp = linspace(t(i),t(i+m-1),20); % get more pts so smooth on plot
        cp = polyval(a,tpp);
        plot(t,c,'.')
        axis([0 1.1*max(t) 0 1.1*max(c)])
        title('HIT SPACE KEY TO STEP - differentiating data')
        ylabel('conc (mol/m3)')
        xlabel('time (s)')
        hold on
        % plot(t(i:i+m-1),cp,'b',tp,yp,'r')
            plot(tpp,cp,'b',tp,yp,'r')
       hold off
        pause
        % YOU NEED TO HIT ANY KEY TO CONTINUE STEPPING
        % SEE THE POLYNOMIALS FIT ON THE PLOT
    end
    % END OF OPTIONAL - SHOW PROCESS ON PLOT
end

% generate pts for a line at zero
zv = zeros(2);
zt = [t(1) t(npts)];

% make final plot
plot(t,c,'.',zt,zv,'g',diffTime,diffValue,'r')
title('pts = data, red line = derivative')
ylabel('conc, C (mol/m3) or dC/dt (mol/m3/s)')
xlabel('time (s)')

pause

rate = -diffValue;
figure(1)
plot(log(diffConc), log(rate), 'bo')

% fit straight line through points in log-log plot
coef = polyfit(log(diffConc), log(rate),1);
% the coefficients of 1st order polynomial are the order and ln(k)
nfit = coef(1); % order
k325fit = exp(coef(2)); % k at 325
```

```
hold on
lnr = polyval(coef,log(diffConc)); % compute points for line through points
plot(log(diffConc),lnr,'k')
s = sprintf('ln(rate) vs. ln(conc), k = %4f, order = %4f',k325fit, nfit);
title(s)
hold off

% now show the model rate vs. the experimental rate estimates
figure(2)
ratefit = k325fit * diffConc .^ nfit;
plot(diffConc, rate, 'bo', diffConc, ratefit,'k')
title('rate vs. conc, circles are from dC/dt estimates, line = model rate')
axis([0 1.1*max(diffConc) 0 1.1*max(rate)])

fprintf('DONE - see final plot \n')

% note the random error or scatter in these data
% which means we don't get the "true" values computed by the lab
%
% question for you: what does "true" mean in real world?
%
% order = 1.93 vs. "true" order = 2 in lab
%
% k325 = 8.84e-04 vs. "true" k325 = 6.36e-04 in lab
%
% k300fit =
%
%    1.3901e-04
%
% vs. k300 = 1.0e-04 "true" rate coefficient in lab for Ea = 60 kJ/mo
```

Nonlinear least squares fitting

In the examples above, we were able to put equations into the form of a straight line in order to estimate parameter values. This is not always possible.

Consider the "reversible" reaction A = B in an isothermal, constant volume batch reactor. Start with pure A. Use the integral method of analysis.

$$\frac{dN_A}{dt} = r_A V = (-k_f C_A + k_b C_B) V$$

$$\frac{dC_A}{dt} = -k_f C_A + k_b C_B \qquad \text{for constant } V$$

$$\frac{dX_A}{dt} = k_f (1 - X_A) - k_b X_A \qquad \text{starting with pure A}$$

$$\frac{dX}{dt} = k_f - (k_f + k_b) X$$

$$X_{eq} = \frac{k_f}{k_f + k_b} \qquad \text{at equilibrium}$$

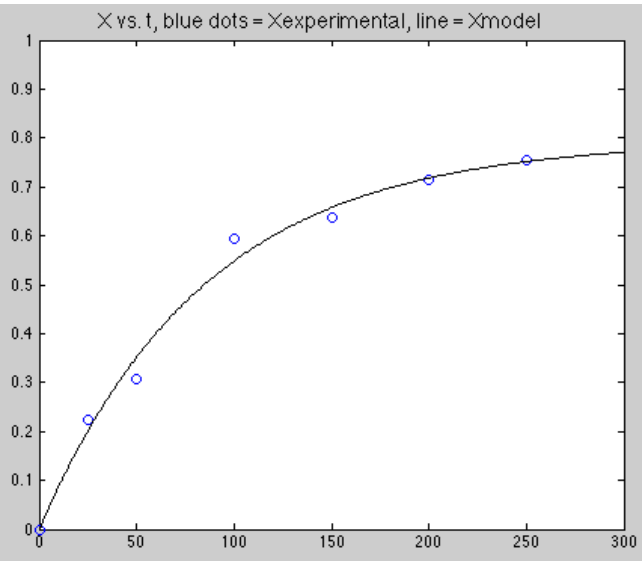
$$-\ln\left(1-X/X_{eq}\right)=\left(k_f+k_b\right)t$$

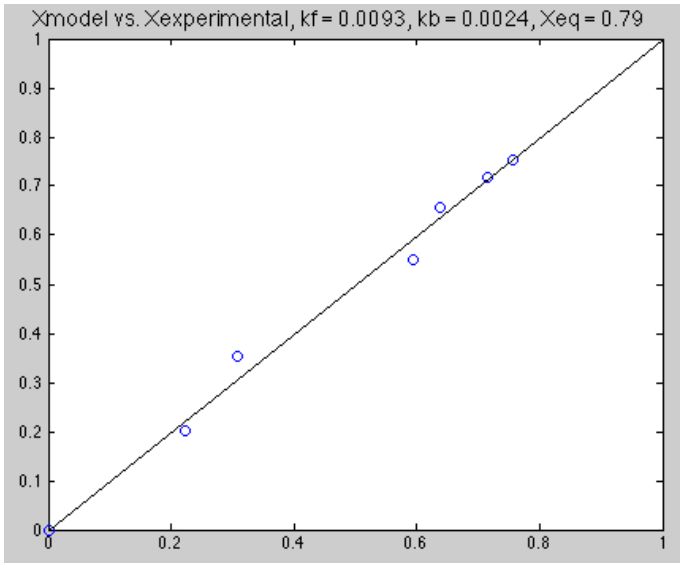
$$X=X_{eq}\left(1-e^{-\left(k_f+k_b\right)t}\right)$$

The only way to get a straight-line fit is to assume a value for $X_{eq}$ and then use a straight-line fit to get ($k_f$+$k_b$), then iterate, trying other guesses of $X_{eq}$.

The solution is to use an algorithm designed to do multi-parameter, nonlinear fits. An example is the Nelder-Mead algorithm. Those algorithms vary the unknown parameters in order to minimize the sum of the squared errors between the model values (model *X* here) and the experimental values.

Here we use a Matlab script and the standard Matlab function `lsqcurvefit()` to fit some sample data. That function uses two different algorithms: trust region reflective and Levenberg-Marquardt




You always want to do plots similar to this to compare model and experimental estimates. Do not rely solely on comparing values of the sum of the squared errors or the $R^2$ value (coefficient of determination). You need to make sure the trends in the model predictions match the trends in the data. A number won't do that - you need a plot.

Another reason to make plots is that nonlinear curve fitting can be very sensitive to the initial guesses used. The fit might be converging to a local minimum in the sum of squared errors and not to the global minimum. If the final plots show a bad fit, try other sets of initial guesses.

Here we used the integral method of analysis. The differential method of analysis for this problem will end up requiring only a straight-line fit but requires the work of differentiating the data.

Matlab listing

```
% multiparameter nonlinear fit of
% concentration vs. time data for A = B reversible reaction
% main file uses function file xmodelF.m
clear
fprintf('--------------------------- \n') % run separator

d = [
0.00 100.0
25.0 77.6
50.0 69.3
100.0 40.6
150.0 36.3
200.0 28.6
250.0 24.4
];

% extract columns
t = d(:,1);
c = d(:,2); % CA

xexp = (c(1) - c)/c(1); % experimental conversion of A

% from integration of balance equation on A
% dCA/dt = -kf*CA + kb*CB
% we get: xmodel = a*(1 - exp(-b*t))
% where a = xeq = kf/(kf+kb) and b = (kf+kb)

% lsqcurvefit() is standard Matlab function for nonlinear least squares
% use to find the values of a and b to minimize sum of squared
% errors (xmodel - xexp)^2
xdata = t;
ydata = xexp;
p0 = [0.5;0.5]; % initial guesses of a = xeq, b = (kf+kb)

% WARNING: nonlinear fitting can be very sensitive to initial guesses
% try different values if final plots look bad - a reason to always plot!

[p, resnorm] = lsqcurvefit('xmodelF',p0,xdata,ydata)

fprintf('row 1 of p is a value (xeq), row 2 is b value (kf+kb), \n')
fprintf('resnorm = sum of squared errors (squared 2-norm of residual) \n\n')

xeq = p(1)
% xeq = kf/(kf+kb)
% kf = xeq*(kf+kb) = p(1)*p(2)
kf = p(1)*p(2)
kb = p(2) - kf

xmodel = xmodelF(p,xdata);
plot(xexp,xmodel,'bo',[0,1],[0,1],'k')
axis([0 1 0 1])
tt1 = 'Xmodel vs. Xexperimental, ';
tt2 = sprintf('kf = %4.4f, kb = %4.4f, Xeq = %4.2f',kf,kb,xeq)
tt = [tt1  tt2]
title(tt)

tp = 0:1:max(t);
xp = xmodelF(p,tp);
figure(2), plot(t,xexp,'bo',tp,xp,'k')
title('X vs. t, blue dots = Xexperimental, line = Xmodel')
```

Listing of function file xmodelF.m

```
function x = xmodelF(p,xdata)
    a = p(1);
    b = p(2);
    % conversion = function(t)
    % from integrating balance equation
    % here, x = function(xdata)
    x = a*(1 - exp(-b*xdata));
```

Additional resources

Schmidt's book - on library reserve & online via `http://roger.ucsd.edu/record=b7179459~S9`
Chapter 2 - sections 18 and 19

Levenspiel's book - on library reserve
Chapter 3

Fogler's *Essentials* book - on library reserve
Chapter 7

Also see Fogler's web resources at `http://www.umich.edu/~essen/html/07chap/frames.htm`