1 The Michaelis-Mentil Model

Image from Biostats Book

Where does this come from?

1.1 Some background

In biochemistry, Michaelis-Menten kinetics is one of the simplest and best-known models of enzyme kinetics. It is named after German biochemist Leonor Michaelis and Canadian physician Maud Menten. The reaction can be illustrated as

$$\mathrm{E} + \mathrm{S} \rightleftharpoons \mathrm{ES} \longrightarrow \mathrm{E} + \mathrm{P}$$

where E is the enzime, S is the substrate, ES is the enzyme binding to the substrate, and P is a product.

Question: How can we model this?

1.2 Building a Mathematical Model

$$\mathrm{E} + \mathrm{S} \rightleftharpoons \mathrm{ES} \longrightarrow \mathrm{E} + \mathrm{P}$$

- $\frac{d[E]}{dt}$ depends on ??
- $\frac{d[S]}{dt}$ depends on ??
- $\frac{d[ES]}{dt}$ depends on ??
- $\frac{d[P]}{dt}$ depends on ??

This leads to the following system of differential equations:

$$\mathrm{E} + \mathrm{S} \underset{\mathrm{k_r}}{\overset{\mathrm{k_f}}{\rightleftharpoons}} \mathrm{ES} \xrightarrow{\mathrm{k_{cat}}} \mathrm{E} + \mathrm{P}$$

$$\begin{aligned}
\frac{d[S]}{dt} &= -k_f[E][S] + k_r[ES] && (1)\\
\frac{d[E]}{dt} &= -k_f[E][S] + k_r[ES] + k_{cat}[ES] && (2)\\
\frac{d[ES]}{dt} &= \quad k_f[E][S] - k_r[ES] - k_{cat}[ES] && (3)\\
\frac{d[P]}{dt} &= \qquad\qquad\qquad\qquad\qquad k_{cat}[ES] && (4)
\end{aligned}$$

2 Simplifying the Model

Now we make some simplifications:

- If the enzyme is conserved, then $[E]+[ES] = [E]_0$ is a constant.

- In situations where $\frac{d[P]}{dt}$ is constant (3) and (4) imply that $\frac{d[ES]}{dt} = 0$. This is often used as an approximation when $\frac{d[P]}{dt}$ is nearly constant.

- From this it follows that

$$\begin{aligned}
k_f[E][S] &= (k_r + k_{cat})[ES] \\
k_f([E]_0 - [ES])[S] &= (k_r + k_{cat})[ES] \\
([E]_0 - [ES])[S] &= \frac{k_r + k_{cat}}{k_f}[ES] \\
[E]_0[S] &= \left(\frac{k_r + k_{cat}}{k_f} + [S]\right)[ES] \\
[ES] &= \frac{[E]_0[S]}{\frac{k_r+k_{cat}}{k_f} + [S]} \\
k_{cat}[ES] &= \frac{k_{cat}[E]_0[S]}{\frac{k_r+k_{cat}}{k_f} + [S]} \\
v = \frac{d[P]}{dt} &= \frac{k_{cat}[E]_0[S]}{}
\end{aligned}$$

- Ignoring the meaning of the constants (for the moment) and focusing on the form, we now have the following sort of relationship between v and $[S]$

  $$v = \frac{\alpha[S]}{\beta + [S]} \tag{5}$$

  That is, the "velocity" of the reaction (rate at which the product is produced) is determined by the concentration of the substrate and constants that do not depend on v or $[S]$.

Equation (5) is not in our favorite linear form, but we can use transforamtions to get into a linear form:

$$\frac{1}{v} = \frac{\beta + [S]}{\alpha[S]} = \frac{1}{\alpha} + \frac{\beta}{\alpha}\frac{1}{[S]}$$

3 Using Data to Fit the Model

This now provides an experimental way to estimate the constants α and β (and from them to infer things about k_f, k_r, and k_{cat}.) We can gather data providing values of v and $[S]$ and fit $\frac{1}{v}$ as a linear function of $\frac{1}{[S]}$. Let's do that using some data from a lab conducted by students at Calvin college.

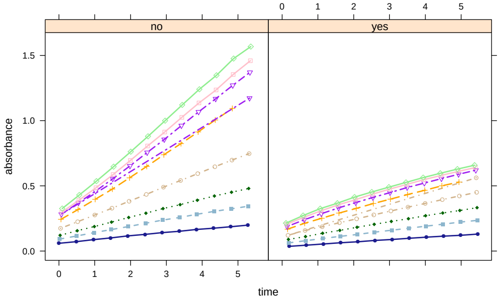
```
mm <-
read.csv("http://www.calvin.edu/~rpruim/data/cal
summary(mm)

       batch              time          absorbance
 SML-1 A1 : 12   Min.   :0.00   Min.   :0.036
 SML-10 B2: 12   1st Qu.:1.37   1st Qu.:0.188
 SML-11 B3: 12   Median :2.74   Median :0.332
 SML-12 B4: 12   Mean   :2.74   Mean   :0.430
 SML-13 B5: 12   3rd Qu.:4.10   3rd Qu.:0.557
 SML-14 B6: 12   Max.   :5.47   Max.   :1.568
 (Other)  :120
```

3.1 Dealing with Indirect Measurements

The measurements are a bit indirect. The amount of product is inferred from the absorbance, and v is inferred by the rate of change in absorbance, which should be a roughly linear function of `time` (in minutes) for each value of $[S]$ (`substrate`, mM peroxide) if our assumptions are true. Notice too that the experiment was run with and without an inhibitor.

Our first step is to estimate the values of v for each level of `substrate` by fitting a simple linear model and determining the slope. We begin by plotting the data to confirm that our linearity assumptions seem appropriate.

```r
xyplot(absorbance ~ time | inhibitor, data =
mm, groups = factor(substrate),
    type = c("p", "l"))
```



Not bad for student-collected data.

Now we need to estimate all those slopes. We can do this all at once with a clever choice of model. For the following analysis, we'll use only the data without the inhibitor.

```
mm$S <- factor(mm$substrate)
mmno <- subset(mm, inhibitor == "no")
slope.model <- lm(absorbance ~ time * S, data
= mmno)
summary(slope.model)



Call:
```

```
lm(formula = absorbance ~ time * S, data = mmno)

Residuals:
     Min       1Q   Median       3Q      Max
-0.14643 -0.00300  0.00011  0.00270  0.04913

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.06214    0.01075    5.78  1.4e-07
time         0.02641    0.00345    7.66  3.7e-11
S0.2         0.03173    0.01523    2.08  0.04041
S0.3         0.05935    0.01526    3.89  0.00021
S0.5         0.10578    0.01528    6.92  1.0e-09
```

```
S1            0.15577    0.01562    9.97  1.1e-15
S1.5          0.20656    0.01516   13.63  < 2e-16
S2            0.19780    0.01535   12.88  < 2e-16
S3            0.22729    0.01538   14.78  < 2e-16
time:S0.2     0.02179    0.00488    4.47  2.6e-05
time:S0.3     0.04235    0.00488    8.68  3.7e-13
time:S0.5     0.08312    0.00488   17.04  < 2e-16
time:S1       0.15245    0.00523   29.14  < 2e-16
time:S1.5     0.17090    0.00468   36.54  < 2e-16
time:S2       0.19665    0.00488   40.31  < 2e-16
time:S3       0.21396    0.00488   43.86  < 2e-16

Residual standard error: 0.0198 on 80 degrees of
```

```
Multiple R-squared: 0.998,Adjusted R-squared: 0.
F-statistic: 2.56e+03 on 15 and 80 DF,  p-value:
```

```r
coef(slope.model)
```

```
(Intercept)        time        S0.2        S0.3
     0.0621      0.0264      0.0317      0.0594
       S1.5          S2          S3   time:S0.2
     0.2066      0.1978      0.2273      0.0218
    time:S1   time:S1.5     time:S2     time:S3
     0.1525      0.1709      0.1966      0.2140
```

```
mmSlopes <- data.frame(S =
as.numeric(levels(mmno$S)), v =
coef(slope.model)["time"] +
    c(`time:S0.1` = 0,
coef(slope.model)[paste("time:S",
levels(mmno$S)[-1],
        sep = "")]))
mmSlopes
```

```
            S      v
time:S0.1 0.1 0.0264
time:S0.2 0.2 0.0482
time:S0.3 0.3 0.0688
time:S0.5 0.5 0.1095
```

```
time:S1   1.0 0.1789
time:S1.5 1.5 0.1973
time:S2   2.0 0.2231
time:S3   3.0 0.2404
```

3.2 Fitting with Ordinary Least Squares

Now we can fit our Michaelis-Menten model:

```r
mmModel <- lm((1/v) ~ I(1/S), mmSlopes)
summary(mmModel)
```

```
Call:
lm(formula = (1/v) ~ I(1/S), data = mmSlopes)

Residuals:
   Min     1Q Median     3Q    Max
-0.544 -0.239  0.119  0.209  0.449

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   2.5915     0.1973    13.1  1.2e-05
I(1/S)        3.5408     0.0468    75.6  3.6e-10

Residual standard error: 0.41 on 6 degrees of fr
```

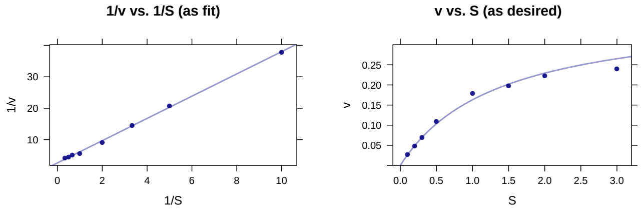
```
Multiple R-squared: 0.999,Adjusted R-squared: 0.
F-statistic: 5.72e+03 on 1 and 6 DF,  p-value: 3
```

```
alpha.hat <- 1/coef(mmModel)[1]
beta.hat <- coef(mmModel)[2] * alpha.hat
c(alpha.hat = alpha.hat, beta.hat = beta.hat)
```

```
alpha.hat.(Intercept)       beta.hat.I(1/S)
                0.386                 1.366
```

```
f <- makeFun(mmModel)
xyplot(1/v ~ 1/S, mmSlopes, main = "1/v vs.
1/S (as fit)")
```

```r
g <- makeFun(f(1/x) ~ x)
plotFun(g(x) ~ x, add = TRUE, col = "navy",
alpha = 0.4)
xyplot(v ~ S, mmSlopes, ylim = c(0, 0.3),
main = "v vs. S (as desired)")
plotFun(1/f(S) ~ S, add = TRUE, col = "navy",
alpha = 0.4)
```



Doing this we see some issues. Although there is a nice tight fit of the transformed data to the least squares regression line, the fit doesn't look nearly as good after back-transforming to the original scales. In particular, errors are much larger for larger values of S. Or thought about the other way around, the way we have fit the model has forced the fit to be extremely good for small values of S at the cost of allowing much poorer fits for larger values of S. This is because the transformations transform the scale for the residuals as well as for the inputs. Note too that the distribution of values of $1/S$ is not nearly as uniform as it is for S.

3.3 Fitting with Nonlinear Least Squares

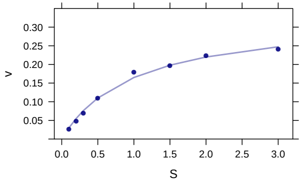
We can do better if we use nonlinear least squares instead of tranforming and using least squares. In nonlinear least squares we fit a parameterized function of arbitrary form by determining (well, estimating anyway) the values of the parameters that minimize the sum of the squares of the residuals

$$SS(\alpha, \beta) = \sum_{i=1}^{n} (v_i - f(\alpha, \beta; S_i))^2$$

This approach will be less forgiving of such large residuals for large values of S. We lose something in the exchange, however. It is not longer the case that simple closed-form formulas exist for the estimates. This means that we will need to rely on numerical estimation. Furthermore the estimators are not guaranteed to be unbiased.

```
# we provide nls() with our estimates from
above as a starting point.
model.nls <- nls(v ~ alpha * S/(beta + S),
data = mmSlopes, start = list(alpha =
alpha.hat,
    beta = beta.hat))
summary(model.nls)


Formula: v ~ alpha * S/(beta + S)

Parameters:
      Estimate Std. Error t value Pr(>|t|)
alpha   0.3297     0.0163   20.24  9.4e-07
```

```
beta    0.9979     0.1191    8.38  0.00016

Residual standard error: 0.00782 on 6 degrees of

Number of iterations to convergence: 5
Achieved convergence tolerance: 6.84e-06


xyplot(v ~ S, mmSlopes, ylim = c(0, 0.35))
ladd(panel.xyplot(mmSlopes$S,
predict(model.nls), type = "l", col = "navy",
    alpha = 0.4))
```



As expected, the fit is much better for larger values of S and we've lost very little for smaller values of S. Now that we have (two sets of) estimates for α and β, we should pause a moment to see what these parameters tell us about the chemistry. Recall Equation

(5)

$$v = \frac{\alpha[S]}{\beta + [S]}$$

As $[S]$ increases,

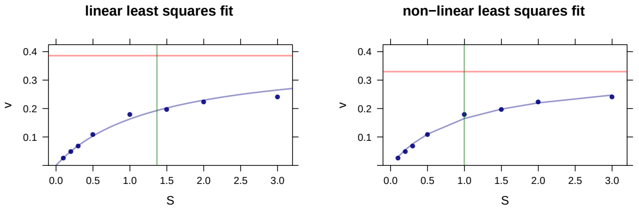
$$\frac{\alpha[S]}{\beta + [S]} \nearrow \alpha$$

so α gives the horizontal asymptote representing the maximum velocity. And if $[S] = \beta$, we get

$$v = \frac{\alpha[S]}{[S] + [S]} = \frac{\alpha}{2} \ ,$$

so β is the value of $[S]$ that gives half the maximum velocity.

```
xyplot(v ~ S, mmSlopes, ylim = c(0, 1.1 *
alpha.hat), main = "linear least squares
fit")
plotFun(1/f(S) ~ S, add = TRUE, col = "navy",
alpha = 0.4)
ladd(panel.abline(h = alpha.hat, col = "red",
alpha = 0.4))
ladd(panel.abline(v = beta.hat, col =
"darkgreen", alpha = 0.4))
xyplot(v ~ S, mmSlopes, ylim = c(0, 1.1 *
alpha.hat), main = "non-linear least squares
fit")
ladd(panel.xyplot(mmSlopes$S,
predict(model.nls), type = "l", col = "navy",
    alpha = 0.4))
ladd(panel.abline(h = coef(model.nls)[1], col
= "red", alpha = 0.4))
ladd(panel.abline(v = coef(model.nls)[2], col
= "darkgreen", alpha = 0.4))
```


linear least squares fit
non−linear least squares fit
v
S
0.1
0.2
0.3
0.4
0.0
0.5
1.0
1.5
2.0
2.5
3.0