CS341 SYSTEMS BIOLOGY I

LECTURE 4

1. Enzyme Kinetics

Enzymes catalyze reactions. We are studying $S \to P$ in which enzyme E either dramatically speeds up that reaction or is necessary for it to occur.

The reaction is broken down like this:

$$S + E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} SE \xrightarrow{k_2} P + E$$

In class, we are watching a tutorial on enzyme kinetics:

`http://www.wiley.com/college/pratt/0471393878/student/animations/enzyme_kinetics/index.html`

Watch sections 1 and 2 of the tutorial. Answer review question.

The enzyme kinetics are modeled by 4 ODEs. They are not introduced in the tutorial, but here they are:

$$\begin{aligned}
\frac{dS}{dt} &= -k_1 \cdot S \cdot E + k_{-1} \cdot SE \\
\frac{dE}{dt} &= -k_1 S \cdot E + k_{-1} SE + k_2 \cdot SE \\
\frac{dP}{dt} &= k_2 \cdot SE \\
\frac{dSE}{dt} &= k_1 \cdot S \cdot E - k_{-1} \cdot SE - k_2 \cdot SE
\end{aligned}$$

The point of the tutorial is that we can't model it as $S \to P$. So lets see why.

Code up the full model in Python.

Try them with lots of enzyme and little substrate. It will appear to be first order with respect to substrate.

```
import numpy as np
import scipy.integrate

def enzyme_model( t, y, params ):
    S = y[0]
    E = y[1]
    P = y[2]
    SE = y[3]

    k1f = params[0]
    k1b = params[1]
    k2 = params[2]

    dydt = np.zeros( (4,) )
    dydt[0] = -k1f*S*E + k1b*SE # S
    dydt[1] = -k1f*S*E + k1b*SE + k2*SE #E
    dydt[2] = k2*SE #P
    dydt[3] = k1f*S*E - k1b*SE - k2*SE #SE
    return dydt
```

Running it.

```
k1f = 2
k1b = 1
k2 = 3
params = (k1f, k1b, k2)

S0 = 4
E0 = 100
SE0 = 0
P0 = 0
yinit = (S0,E0,P0,SE0)

em = lambda t,y: enzyme_model(t,y,params)
sol = scipy.integrate.solve_ivp( fun=em, t_span=(0,100), y0=yinit,
                                 t_eval=np.arange(0,100,1) )
```

Plotting it

```
import matplotlib.pyplot as plt

plt.plot( sol.t, sol.y.T )
plt.xlabel( "Time")
plt.ylabel( "Concentration" )
plt.legend( ["S","E","P","SE"])
plt.title( "Enzyme-catalyzed reaction");
```

Tutorial section 3 covers what happens when the concentration of the enzyme is small with respect to the concentration of the substrate.

Then try the code with lots of substrate and very little enzyme. It will appear to be first order with respect to enzyme and zeroth order with respect to substrate (i.e. because there are so few enzyme, they are rate-limiting and the amount of substrate has no effect on the rate of product creation).

```
k1f = 2
k1b = 1
k2 = 3
params = (k1f, k1b, k2)

S0 = 100
E0 = 1
SE0 = 0
P0 = 0
yinit = (S0,E0,P0,SE0)

em = lambda t,y: enzyme_model(t,y,params)
sol = scipy.integrate.solve_ivp( fun=em, t_span=(0,100), y0=yinit,
                                 t_eval=np.arange(0,100,1) )
```

Back to the question of the day. What is the relationship between S and dP/dt?

Let's graph it.

Augment the running code so that you draw not just the traces over time, but also the relationship.

```
plt.figure()
plt.plot( sol.t, sol.y.T )
plt.xlabel( "Time")
plt.ylabel( "Concentration" )
plt.legend( ["S","E","P","SE"])
plt.title( "Enzyme-catalyzed reaction");

plt.figure()
P = sol.y[2,:].T
S = sol.y[0,:-1].T
velocity_of_P = np.divide(np.diff(P), (sol.t[1]-sol.t[0]))
plt.plot( S, velocity_of_P );
plt.xlabel( 'S' );
plt.ylabel( 'dP/dt' );
```

The curve is hyperbolic. It asymptotes at a maximal rate of production. If there is very little S in the system, then product can't be made too quickly. Going from a little bit of S to a medium bit of S causes a dramatic increase in the rate of production. But then we get to a point where adding more S gives us no advantage.

[Watch section 4 of the tutorial.]

Using what is called a quasi-steady-state-assumption (that SE and E are both at constant concentrations), it is possible to derive a mathematical expression for $\frac{dP}{dt}$ that is a function of S, but not E or SE (they aren't changing so why model them?). It is

$$v_{max}\frac{S}{K_m + S}$$

where $v_{max} = E(0) \cdot k_2$ and $K_m = \frac{k_{-1}+k_2}{k_1}$.

The kinetics captured by this expression are referred to as Michaelis-Menton kinetics.

Plot the Michaelis-Menton with the velocity_of_P curve we computed from the simulation:

```
Km = (k1b + k2)/k1f
vmax = E0*k2
analytic_dPdt = vmax * np.divide( S, (S+Km) )
plt.plot(S, analytic_dPdt)
plt.plot( S, velocity_of_P)
plt.xlabel( "S")
plt.ylabel("dP/dt")
plt.legend( ["M-M expression","Simulation"]);
```