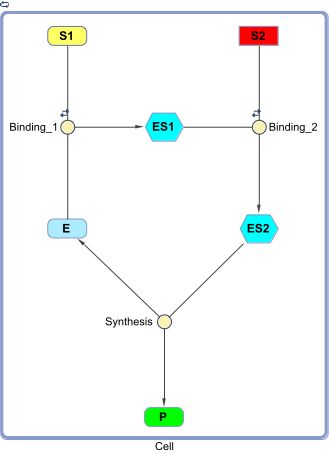
# **Simulation and Analysis of Enzymatic Reactions.**

****The simulation and analysis of enzymatic reactions is based on the same principles and tools used for the simulation of binding reactions. Consider as an example the sequential binding of two substrates to an enzyme to produce a single product: this is defined as a *sequential bi-uni reaction*.

The simulation of this reaction can be carried out by executing the cells in the script:

../TOOLBOXES/ENZYME\_KINETICS/

Bi\_Uni\_Michaelis\_Menten\_Global\_Fit.m

The following are the three reactions involved:

S1 + E <-> ES1

ES2 -> P + E

S2 + ES1 <-> ES2

which correspond to the following ordinary differential equations (ODE):

ODEs:

d(S1)/dt = 1/Cell\*(-ReactionFlux1)

d(E)/dt = 1/Cell\*(-ReactionFlux1 + ReactionFlux2)

d(ES1)/dt = 1/Cell\*(ReactionFlux1 - ReactionFlux3)

d(P)/dt = 1/Cell\*(ReactionFlux2)

d(S2)/dt = 1/Cell\*(-ReactionFlux3)

d(ES2)/dt = 1/Cell\*(-ReactionFlux2 + ReactionFlux3)

Fluxes:

ReactionFlux1 = (kon1\*S1\*E)\*Cell-(koff1\*ES1)\*Cell

ReactionFlux2 = (kcat\*ES2)\*Cell

ReactionFlux3 = (kon2\*S2\*ES1)\*Cell-(koff2\*ES2)\*Cell

Parameter Values:

koff1 = 200 1/second

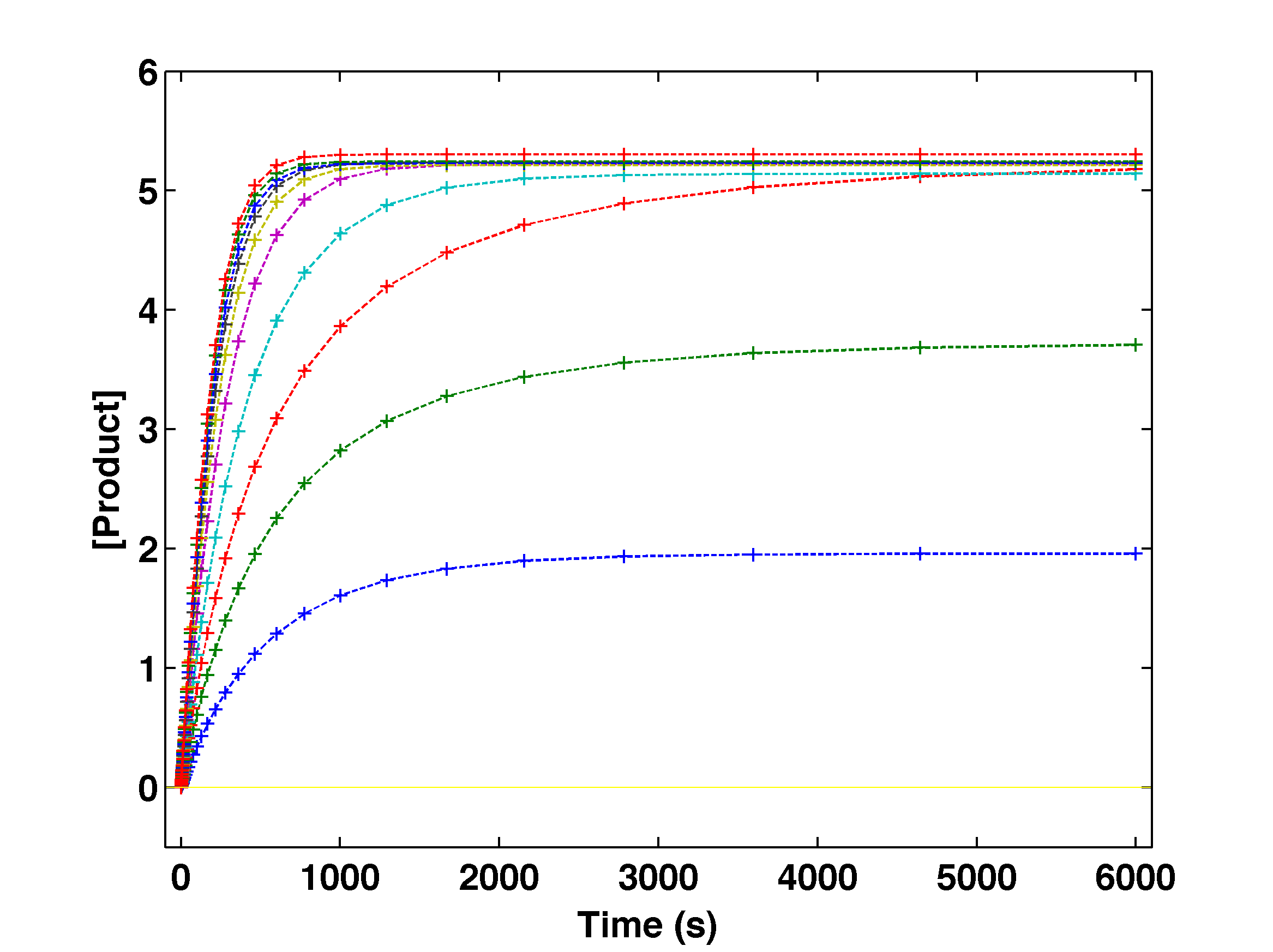
kon1 = 10 1/(micromolarity\*second)

kcat = 0.05 1/second

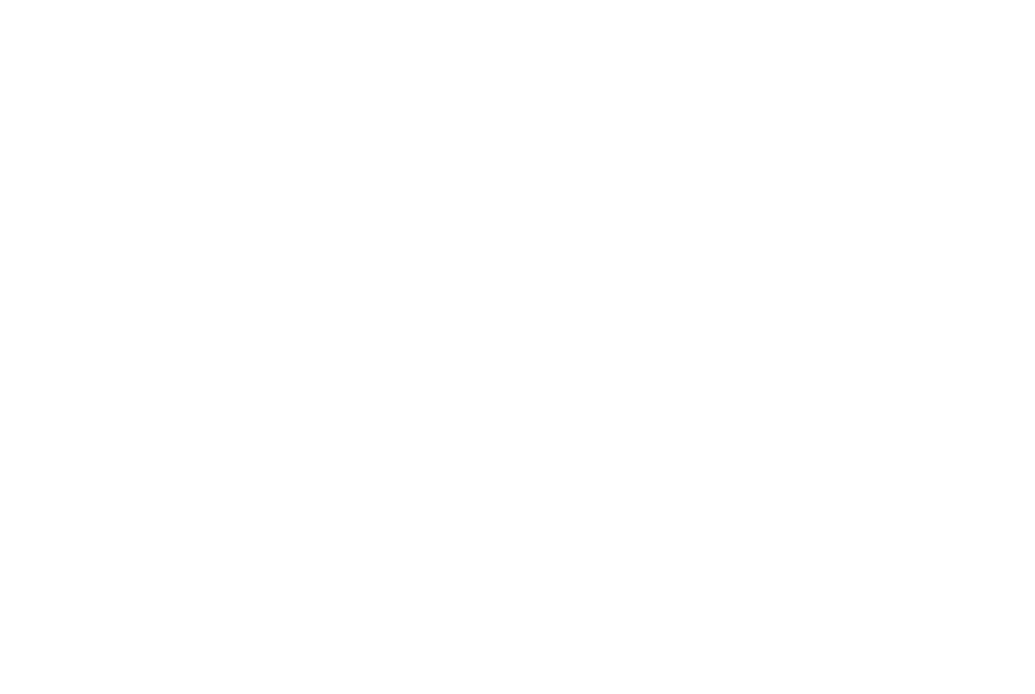
kon2 = 10 1/(micromolarity\*second)

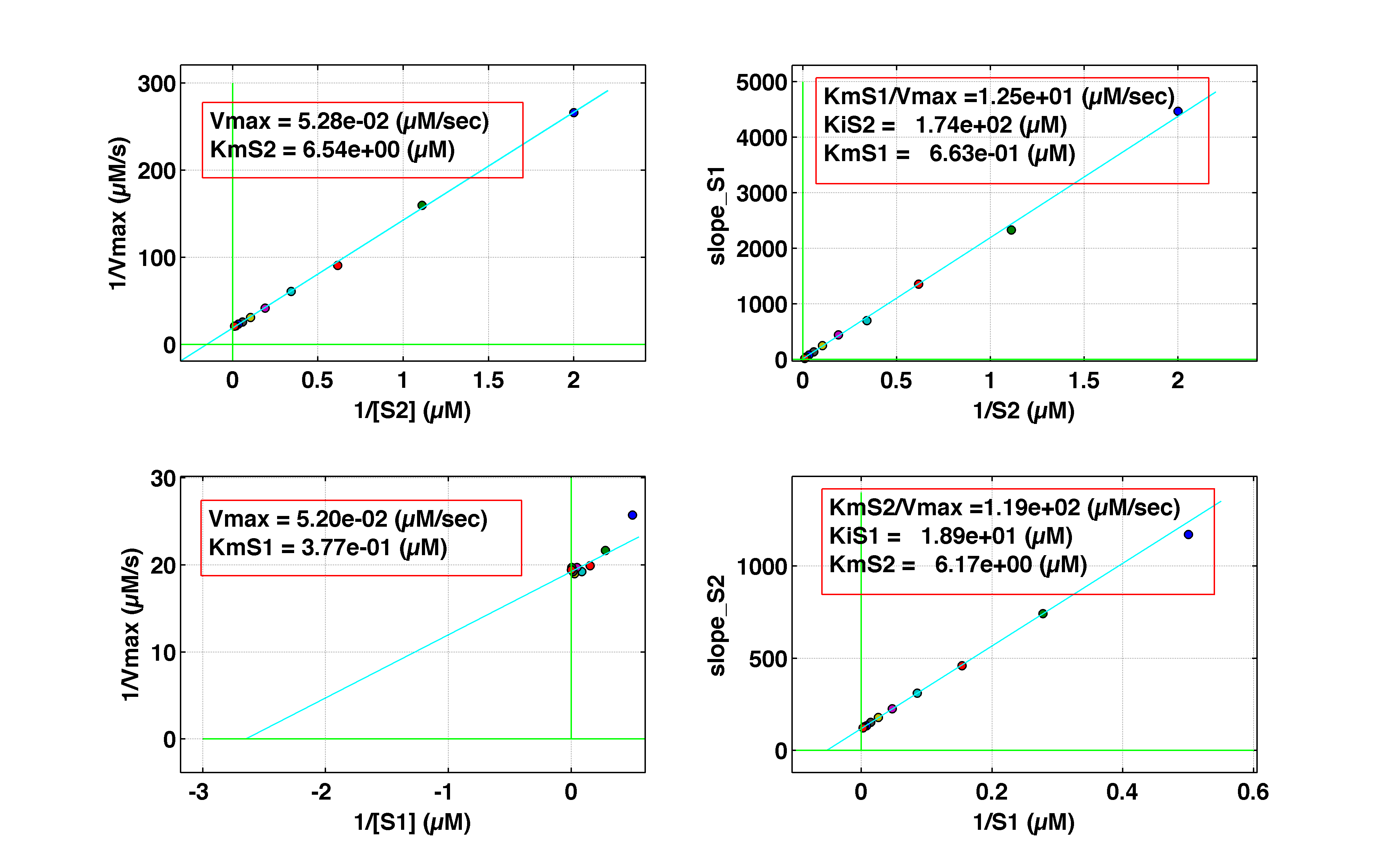
koff2 = 50 1/second

Cell = 1 milliliter

In order to reproduce an experimental kinetic analysis of this reaction we need to simulate progress curves at different combinations of the two substrates. We notice that *K*d1 = *k*off1/*k*on1 = 200/10 = 20 μM, so we can try a concentration range (spaced logarithmically) from 1/10 to 20 times the *K*d (2 to 400 μM). Since *K*d2 = *k*off2/*k*on2 = 50/10 = 5 μM, we can try a concentration range (also spaced logarithmically) from 0.5 to 100 μM. We also choose 35 time points spaced logarithmically between 1 and 6000 seconds with an additional 0 time point as the reference reaction start.

After simulating the reactions using all the different combinations of substrates concentrations we can choose various ways of analyzing the data. We start by plotting the initial velocity of the reaction for each initial concentration of the 1st substrate at the different concentration of the 2nd substrate. We also derive two different Lineweaver-Burke plot in which the reciprocal of the initial velocity is plotted against the reciprocal of the 1st substrate concentration at different concentrations of the 2nd substrate, and against the reciprocal of the 2nd substrate concentration at different concentrations of the 1st substrate.

The *V*max and *K*m's for S1 and S2 can be obtained directly from these two plots, although for better accuracy it is usually recommended to use 4 different replots of the points from the LB plots.



From these replots we can obtain all the kinetic parameters for this enzymatic reaction:

*kcat* = 0.052 s-1

*K*m\_S1 = 0.52 μM

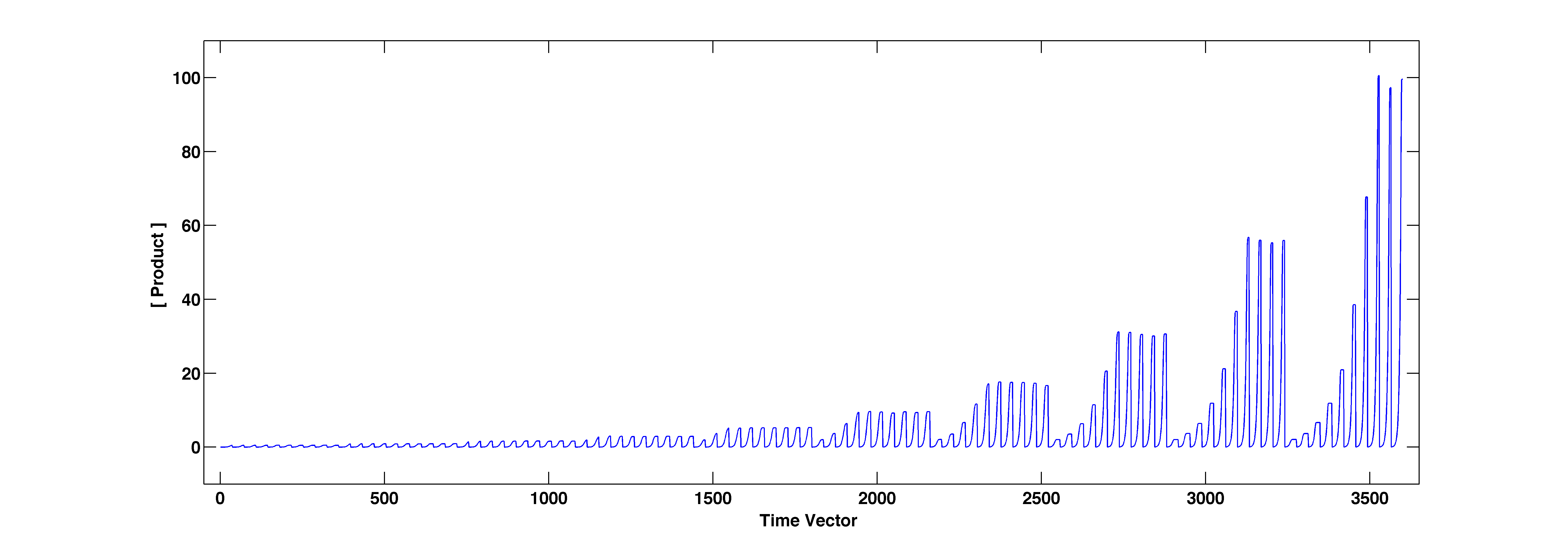
*K*d\_S1 = 18.89 μM

*K*m\_S2 ≈ *K*d\_S2 = 6.35 μM

*k*off\_S1 = 189 s-1

*k*off\_S2 = 63.5 s-1

Alternatively we can fit globally all the progress curves with a single set of parameters. For this purpose we first convert all the progress curves into a single consecutive progress curve.

To understand what this means we plot one set (i.e. set 5) of progress curves at a fixed concentration of S2, and also the vector containing all the concatenated progress curve. The 5th set of curves starts around 1500 in the concatenated vector.

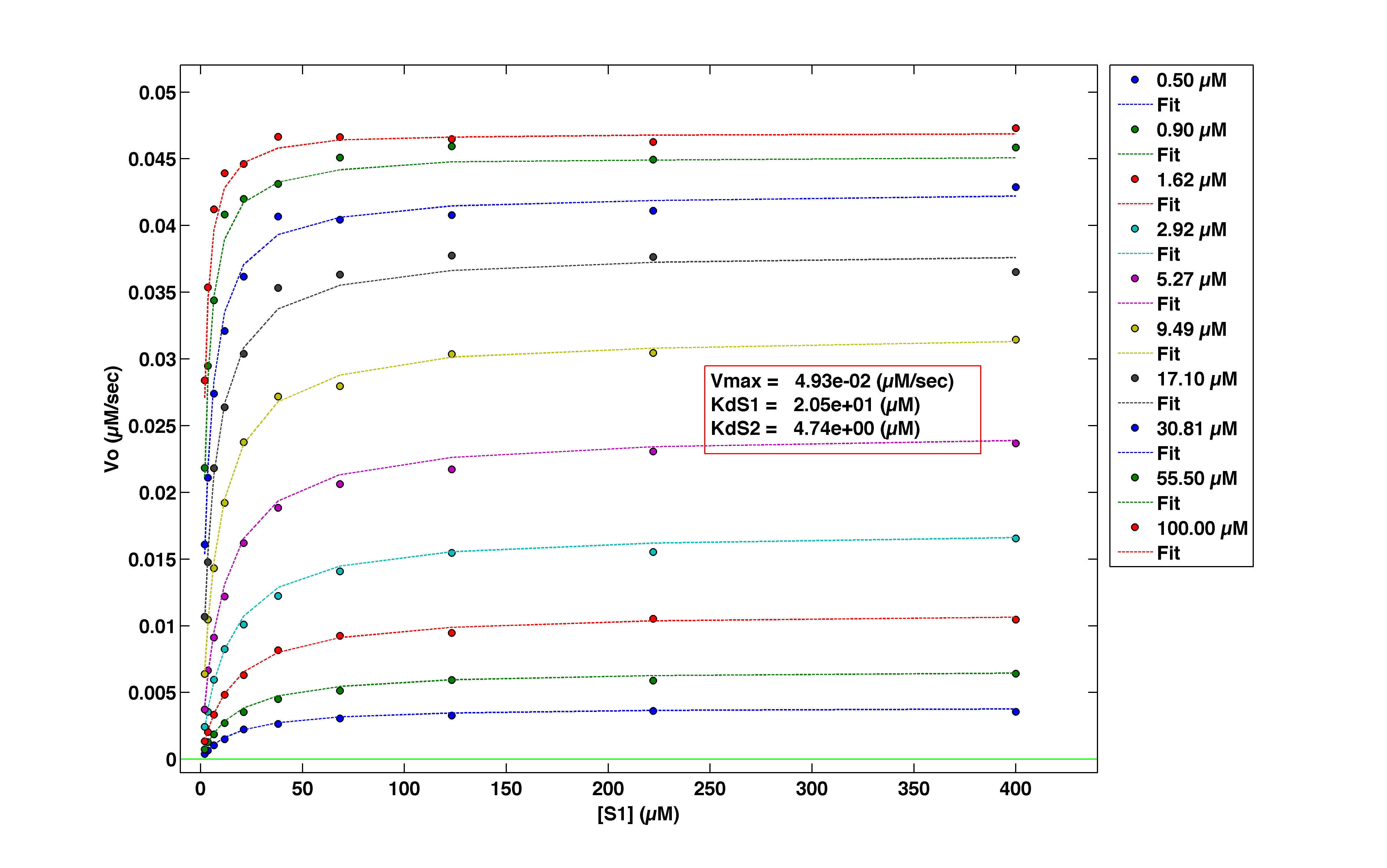
We will refine *k*off\_S1, *k*cat, and *k*off\_S2: the initial values are taken from the previous fit.

Next, we use the *nlinfit* function from the Statistics Toolbox (or *lsqcurvefit* from the Optimization Toolbox) as we did for the global fit of the progress curves originated from the binding of a ligand to two cooperative binding site (see CHAPTER 14). The function called by *nlinfit* at each optimization cycle uses the current value of the parameters to calculate a new concatenated set of progress curves. Refinement of the kinetic parameters is driven by minimization of the residual between the concatenated vector calculated at each refinement cycle and the concatenated vector derived from the observed (or synthetic) progress curves. In this case the refined parameters are:

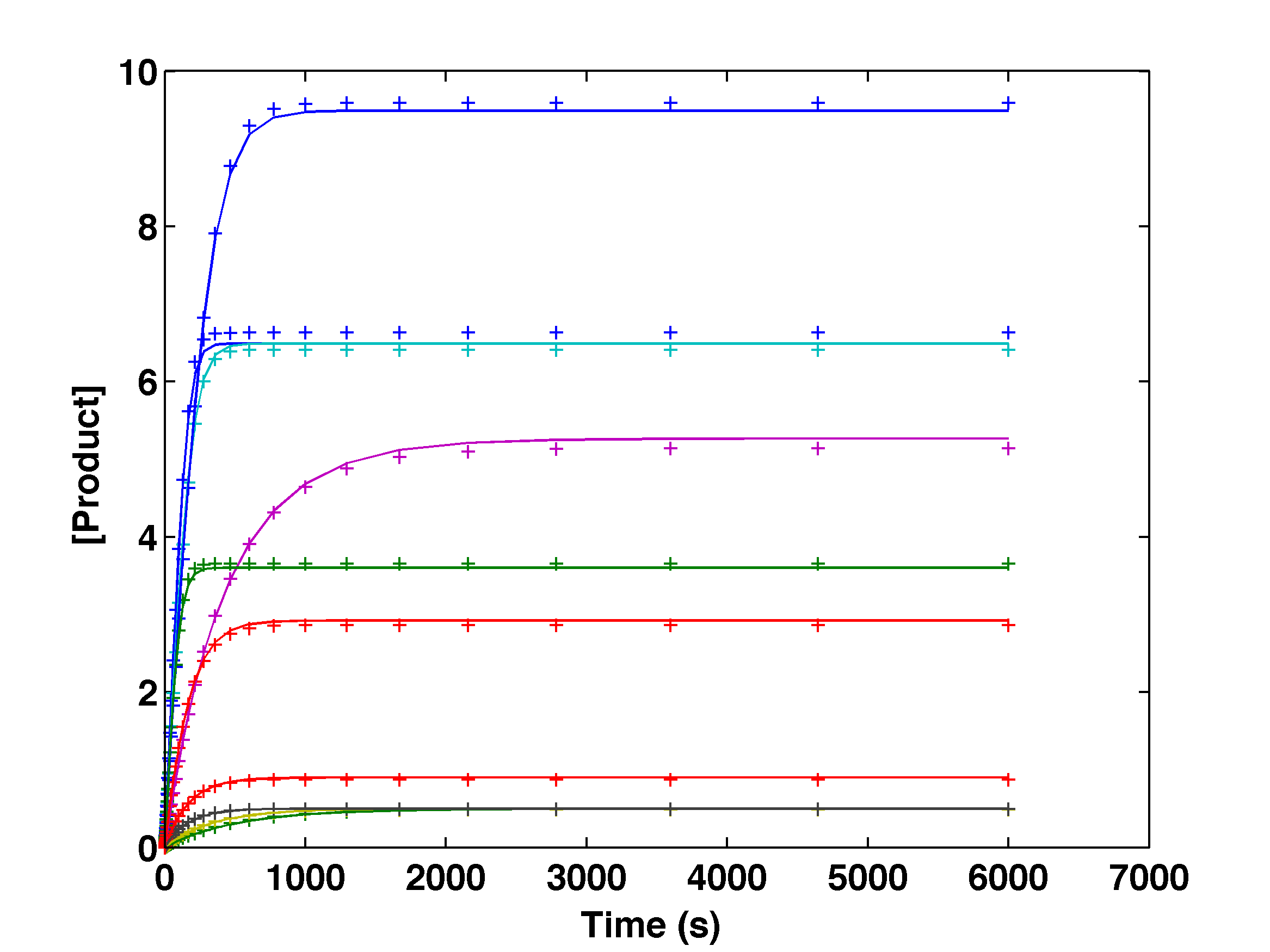
*kcat* = 0.0495 s-1

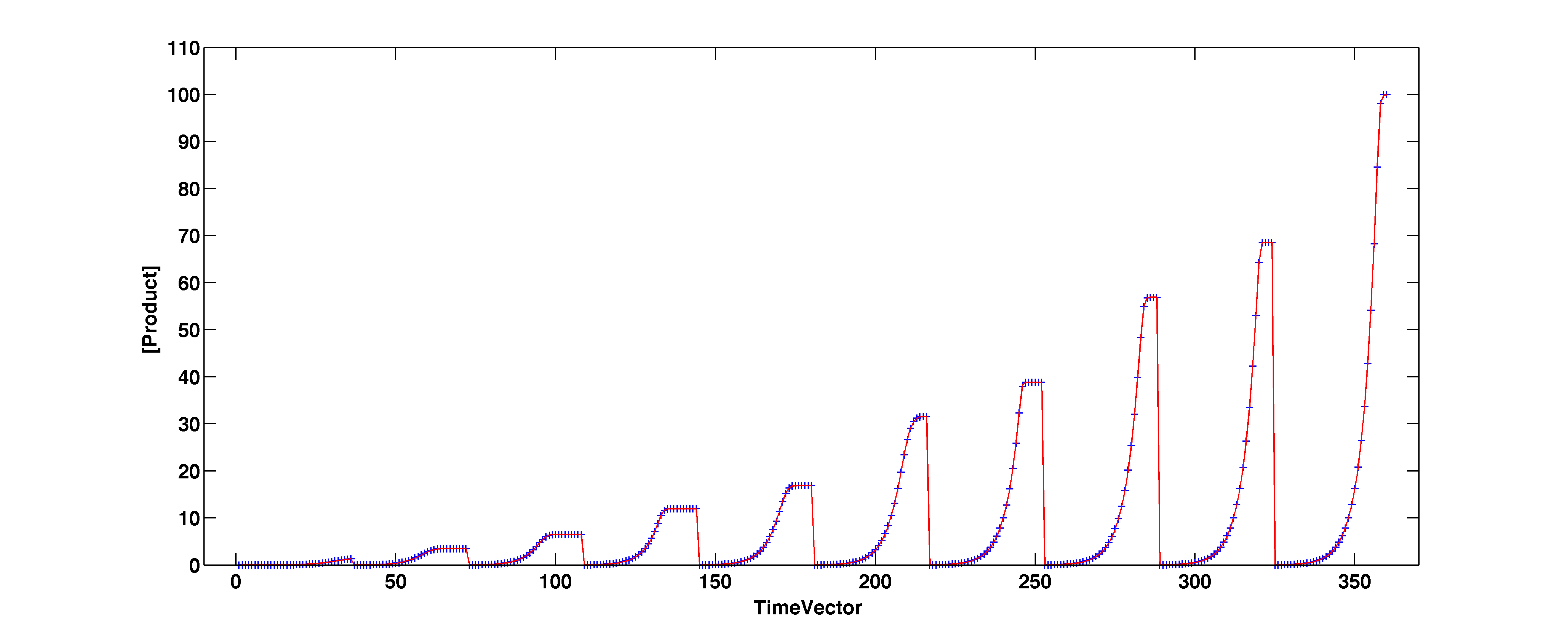
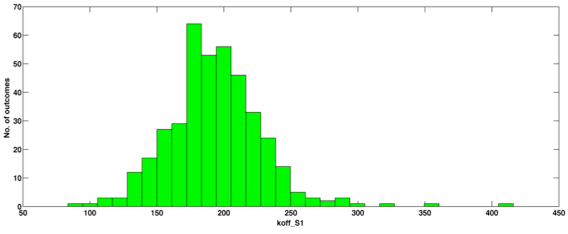
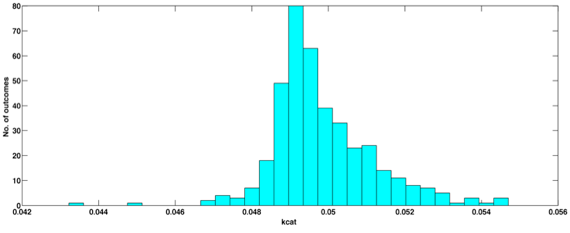
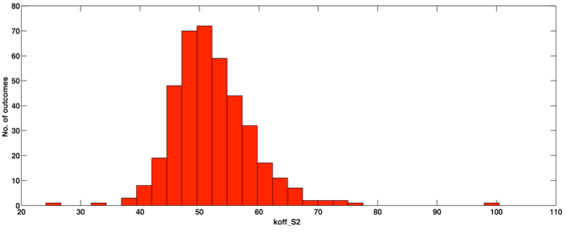
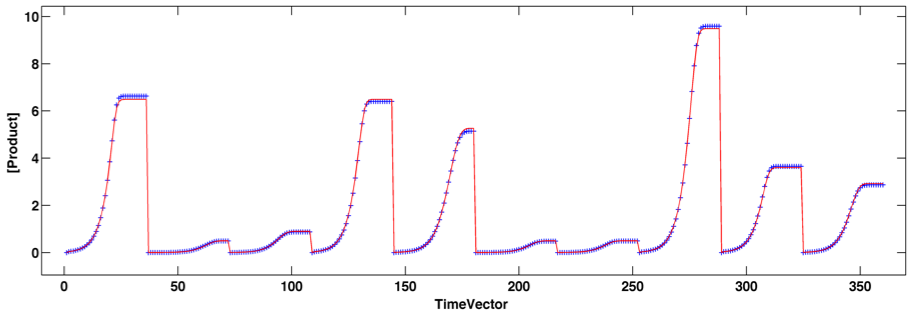
*k*off\_S1 = 191.7 s-1

*k*off\_S2 = 50.7 s-1



At the end of the refinement we can compare the initial velocities of all the progress curves in the synthetic data with those calculated from the refined parameters.

While global refinement of the kinetic parameters against all the progress curves provides more accurate values for these parameters if used as shown, it does not decrease the amount of experimental data that needs to be collected. Fortunately, in most cases even just 10-20 progress curves contain sufficient information to drive the refinement. We can check this by selecting randomly 10 progress curves from different combinations of S1 and S2 concentrations, and by repeating the parameter refinement using only those curves. For the curves shown in the figures above (with the regular or concatenated vector), the refinement yields kinetic parameters values that are within 10% of the true values. However, not any random choice will give acceptable values. This can be easily ascertained by repeating the refinement several hundred times with random sets of 10 curves extracted from the original data. While the distributions of the refined parameters are centered around the true values, the tails of the distributions extend to values that are clearly incorrect. In general, it is advisable to choose combinations of the values of S1 and S2 that provide a good distribution of the amount of product formed over the entire available range. For example, in the concatenated vector shown in the earlier figure only a small range of product concentrations (up to less than 10 μM) was sampled. A much better distribution is shown in the following concatenated vector, which produced improved values (within 5% of the true values) for two of the three parameters.



Since the analysis of only 10 progress curves is much faster, also substrate concentrations can be refined in addition to the kinetic parameters. For example, in the case of the concatenated vector shown above we would refine 3 rate constants and 20 substrates concentrations (2 for each progress curve).

**Conclusion**

Global fit of all the progress curves with a single set of kinetic parameters can be used effectively either to obtain *highly accurate* values for the kinetic parameters of the reaction under study using the same complete set of progress curves employed with traditional initial velocity studies, or to obtain *reasonably accurate* values for the same kinetic parameters, using only a small number of progress curves.**SPECIAL TOPICS**

**Stochastic kinetics.** Traditional chemical kinetics holds that in a thermally equilibrated chemical system, the number of molecules of each chemical species *Si* (*i =1, . . . , N* ) evolves in time according to a system of ordinary differential equations (ODEs) of the form:

RRECME

where the function depends on the specific reactions. This deterministic system of equations, is called the *Reaction Rate Equation* (RRE) (CHAPTER 13).

In such a system, it is usually assumed that molecular populations are many orders of magnitude larger than just 1 molecule. However, if the system volume is very small (i.e., a single cell), and molecular populations are only a few orders of magnitude larger than unity, the RRE does not provide an accurate description of the system. A pertinent example of this kind of situation is offered by the reaction of a rare molecule of a transcription factor to the regulatory region of a gene present in just one copy.

In this case, the binding of the transcription factor is better described using a *probabilistic model*, and traditional chemical kinetics are replaced by *stochastic kinetics*. The following discussion of stochastic kinetics is based on the pioneering work of Daniel Gillespie (see *Annu. Rev. Phys. Chem.* 2007, 58:35–55) and the expository treatment of the subject by Desmond Higham (see *SIAM Review* 2008, 50:347-368).

In developing a probabilistic model of a reaction, we know the numbers of molecules of each species at time *t* = 0, and we want to describe how these numbers evolve with time. We introduce the *state vector*:

where is the number of molecules of species *i* at time *t*. The state vector changes each time one of the *M* types of reaction takes place. For example, consider a Michaelis–Menten type reaction for a system composed of four species:

1. substrate, S
2. enzyme, E
3. enzyme:substrate complex, ES
4. product, P

where *c1*, *c2*, and *c3* are the *rate constants.* If we start with *K* molecules of *S* and *E*, and no molecules of *ES* and *P*, then the state vector can take as possible values (among many more):

We notice that if the 1st, 2nd, or 3rd reaction occurs, then:

, , ,

respectively. The probability of the 1st reaction occurring is proportional to the product of the numbers of *S* and *E* molecules: more precisely, the probability of this reaction taking place in the infinitesimal time interval is given by:

The product reflects the likelihood of two molecules colliding, and the constant is a factor that accounts also for the fact that not all collisions result in a reaction. In general, if we have chemical species *S1*, *S2*, …, *SN* taking part to *M* different reactions, with 1 ≤ *j* ≤ *M*, the *j* th reaction changes the state vector from .

The *propensity function* of the *j* th reaction is defined so that:

is the probability of the *j* th reaction to occur in the infinitesimal time interval .

This definition is the foundation of *stochastic kinetics*. If we know the probability of being in any of the possible states of the system at time *t*, there are only two basic scenarios for time *t* that would allow the system to be in state at time : either the system was already in state at time *t* and no reaction took place over , or for some 1 ≤ *j* ≤ M the system was in state at time *t*, and the *j* th reaction fired over bringing the system into state .

If define *A* as the event that the system is in state at time , *H0* as the event that the system is in state at time *t*, and *Hj* (for 1 ≤ *j* ≤ *M*) as the event that the system is in state at time *t*, then *P*(*A*|*Hj* ) is the probability of the *j* th reaction firing over . From the definition of the propensity function we derive:

Likewise, *P*(*A*|*H0* ) is the probability of no reaction firing over , which must be equal to 1 minus the probability of any reaction firing:

If *A* is the event of interest, and the events *H0,H1,H2, … , HM* are disjoint as only one of them can happen, then we can write the total probability of *A* as:

We now define the quantity as the probability that . Combining this definition with the expressions for and we obtain:

which can be rearranged to:

Taking the limit , the left-hand side becomes a time derivative:

CME

which is known as the *Chemical Master Equation*, CME, in which the *state vector* ranges over the entire set of possible values. According to the CME the evolution of the system depends only on its previous state () and not on its history, a property that is the hallmark of *Markov processes*.

It is important to realize that while both the *Reaction Rate Equation*, RRE, and the *Chemical Master Equation*, CME, are systems of ODE’s:

RRE has *one ODE for each chemical species*.

CME has *one ODE for each possible state of the system*

(that is, every possible combination of the reactant molecules).

As a consequence, numerical integration of the CME, which would give the entire probability distribution of the states of a system, is extremely difficult. However, while in most cases it is impossible to calculate , it is possible to compute simulated trajectories of versus *t* using a different probability function, , defined so that:

is the probability that, given :

1. the next reaction will be the reaction.

2. it will occur in the time interval .

Given a system in state , can be interpreted as the product of two individual probabilities:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prob. of 1 and 2 | = | Prob. that no reaction  occurred during | x | Prob. that *j* reaction  occurred during  ). |

If what happens in the interval is independent of what happens in the interval , we also have:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Prob. of no reaction during |  | = | Prob. no reaction during | x | (1 – sum of prob. of each reaction during  ). |

which using the definition of propensity function can be written as:

rearranging and taking the limit :

Solving the ODE for the initial condition leads to:

Going back to we derive:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Prob. of 2 and 1 | = | Prob. that *j* reaction  occurred during  ). | x | Prob. that no reaction  occurred during |
|  | = |  |  |  |
|  |  | **⇓** |  |  |
|  | = |  |  |  |
|  |  | **⇓** |  |  |
|  | = |  |  |  |
|  |  | **⇓** |  |  |
|  | | | | |

Based on this result it is possible to simulate independently:

1. a *reaction index*: , we pick one of the reactions with the rule that the chance of picking the *j* th reaction is proportional to . Thus, we derive the index *j* from a uniform sampling in the (0,1) interval.

2. a *time to next reaction*: , this is the *density function* for a continuous random variable with an *exponential distribution*. Integrating we get:

If we draw from a uniform distribution in the (0,1) interval, then , and we can rewrite:

Therefore, both *reaction index* and *time to next reaction* can be computed *via* a uniform sampling of the (0,1) interval. This generating method is the basis for the *stochastic simulation algorithm* (SSA) to construct trajectories of :

Initialize the time *t* = *t0* and the system’s state .

⇓

Draw two random numbers, r1 and r2.

⇓

With the system in state at time *t*,

evaluate all the and their sum .

⇓

Set *j* to be the smallest integer satisfying:

while *t*<*T*

⇓

Replace and .

To understand how the algorithm works, consider the simple decay reaction:

null

The propensity function for this 1st order reaction is , and the *state-change vector* is

. The CME for the reaction is:

,

which, since there is only one type of reaction, simplifies to:

The SSA for this reaction is simple: in state at time *t*, we draw a random number *r* in the uniform unit-interval, increase *t* by , decrease by 1, and then repeat.

As an example, let’s derive the numerical results for 100 trajectories, given c =1s-1 and x0 =100:

SSA\_trajectories = figure;

for k = 1:100

x\_0 = 100;

c = 1;

t = zeros(200,1);

x = zeros(200,1);

x(1) = x\_0;

for i = 2:200

r = rand;

tau = (1/(c\*x(i-1))\*log(1/r));

t(i) = t(i-1)+tau;

x(i) = x(i-1)-1;

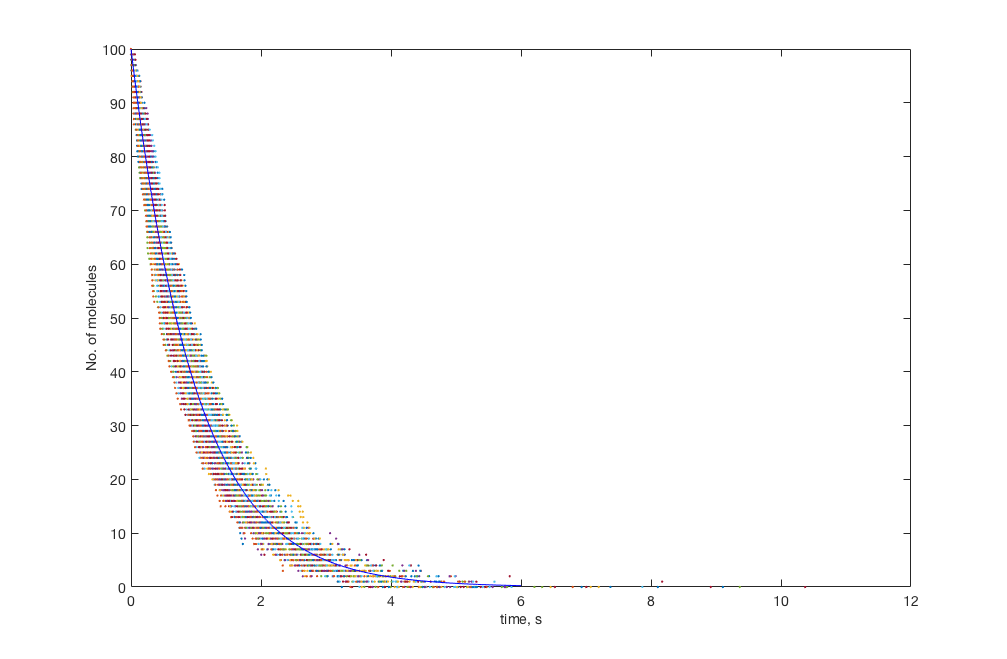
end

plot(t,x,'.','MarkerSize',0.5)

hold on

end

The RRE for the reaction is:

and its solution for the initial condition is:

Here we add to the figure also the RRE deterministic solution:

t\_det = [0:0.01:6];

for i = 1:length(t\_det)

x\_det(i) = x\_0\*exp(-t\_det(i));

end

plot(t\_det,x\_det,'-b')

xlabel('time, s')

ylabel('No. of molecules')

We are now ready to consider a more complex case: for example, we can derive a numerical simulation of an enzymatic reaction. We will use a MATLAB code modified from the original written by Desmond Higham (which can be downloaded from <http://personal.strath.ac.uk/d.j.higham/chem/ssa_mm.m)>, in order to derive 30 different trajectories of the simple enzymatic reaction described at the beginning of this chapter:

with stoichiometric matrix ***V***:

r1 r2 r3

We set the volume, the rate constants and the number of molecules at *t* = 0:

Given Avogadro’s number , for a volume , these number of molecules correspond to very low concentrations:

The columns of the *stoichiometric matrix* ***V*** provide the *state change vector* for each reaction:

V = [-1 1 0; -1 1 1; 1 -1 -1; 0 0 1];

Avog\_Number = 6.0221409e23;

Volume = 1e-14;

MM\_SSA = figure;set(gcf,'Unit','Normalized','Position',[0 0.4 0.5 0.6]);

annotation(MM\_SSA,'textbox',[0.4 0.8 0.06 0.04],'Color','red','String',{'SUBSTRATE'},...

'LineStyle','none','FontWeight','bold','FitBoxToText','off');

annotation(MM\_SSA,'textbox',[0.4 0.78 0.06 0.04],'Color','green','String',{'ENZYME'},...

'LineStyle','none','FontWeight','bold','FitBoxToText','off');

annotation(MM\_SSA,'textbox',[0.4 0.76 0.12 0.04],'Color','magenta',…

'String',{'ES COMPLEX'},'LineStyle','none','FontWeight','bold','FitBoxToText','off');

annotation(MM\_SSA,'textbox',[0.4 0.74 0.06 0.04],'Color','cyan',...

'String',{'PRODUCT'},'LineStyle','none','FontWeight','bold','FitBoxToText','off');

for i = 1:30

    nS = 300;

    nE = 200;

    nES = 0;

    nP = 0;

    k1 = 1e7/(Avog\_Number\*Volume);

    k2 = 1e-3;

    k3 = 5e-2;

    t = 0;

    plot(t,nS,'.r','MarkerSize',0.5);

    xlim([-5 95]);ylim([-20 350]);

    box on;grid on;hold on;

    plot(t,nE,'.g','MarkerSize',0.5);

    plot(t,nES,'.m','MarkerSize',0.5);

    plot(t,nP,'.c','MarkerSize',0.5);

    tfinal = 90;

    while t < tfinal

        a1 = k1\*nS\*nE;

        a2 = k2\*nES;

        a3 = k3\*nES;

        a = [a1 a2 a3];

        asum = a1 + a2 + a3;

        r1 = rand;

        j = min(find(r1<cumsum(a/asum)));

Here, cumsum(a/asum) is a vector whose *k*th component is the cumulative sum of the first *k* components of a/asum, that is:

As the final component of this vector equals 1, it follows that find(rand<cumsum(a/asum)) is a vector giving the indices for those components of cumsum(a/asum) that exceed the random number. So, min(find(rand<cumsum(a/asum))) records the smallest such index.

        r2 = rand;

        tau = log(1/r2)/asum;

        nS = nS + V(1,j);

        nE = nE + V(2,j);

        nES = nES + V(3,j);

        nP = nP + V(4,j);

        t = t + tau;

        plot(t,nS,'.r','MarkerSize',0.5);

        plot(t,nE,'.g','MarkerSize',0.5);

        plot(t,nES,'.m','MarkerSize',0.5);

        plot(t,nP,'.c','MarkerSize',0.5);

    end

end

hline([0 0],'-y');

vline([0 0],'-y');

xlabel('time, s');

ylabel('No. of molecules');

We can superimpose on this result the RRE *eigensolution*:

k1 k3

U1 + U2 ⇔ U3 ⇒ U4 + U2

K2

where U1 is the substrate S, U2 the enzyme E, U3 the complex ES, and U4 the product P.

k1 = 1e7;

k2 = 1e-3;

k3 = 5e-2;

dU1/dt = -k1\*U1\*U2 + 0\*U2 + (k2)\*U3 + 0\*U4

dU2/dt = 0\*U1 -k1\*U1\*U2 + (k2+k3)\*U3 + -0\*U2\*U4

dU3/dt = k1\*U1\*U2 + 0\*U2 -(k2+k3)\*U3 + 0\*U2\*U4

dU4/dt = 0\*U1 + 0\*U2 + k3\*U3 -0\*U2\*U4

We initialize the *reaction rate matrix* assigning the initial concentrations to the species:

nS = 300;

nE = 200;

nES = 0;

nP = 0;

Uo = [nS/(Avog\_Number\*Volume) nE/(Avog\_Number\*Volume) 0.0 0.0]';

U1 = Uo(1);

U2 = Uo(2);

row1 = [ -k1\*U2 0 k2 0 ];

row2 = [ 0 -k1\*U1 (k2+k3) 0 ];

row3 = [ k1\*U2 0 -(k2+k3) 0 ];

row4 = [ 0 0 k3 0 ];

K = [row1;row2;row3;row4];

end\_time = 50.0;

time\_step = 0.01;

time\_vec = [0:time\_step:end\_time];

nsteps = length(time\_vec);

U = zeros(4,nsteps); U(:,1) = Uo;

for i = 2:nsteps

K = [row1;row2;row3;row4];

[S,D] = eig(K);

D = diag(D);

Dt = D\*time\_step;

eDt = diag(exp(Dt));

U(:,i) = (S\*eDt/S)\*U(:,i-1);

U1 = U(1,i);

U2 = U(2,i);

row1 = [ -k1\*U2 0 k2 0 ];

row2 = [ 0 -k1\*U1 (k2+k3) 0 ];

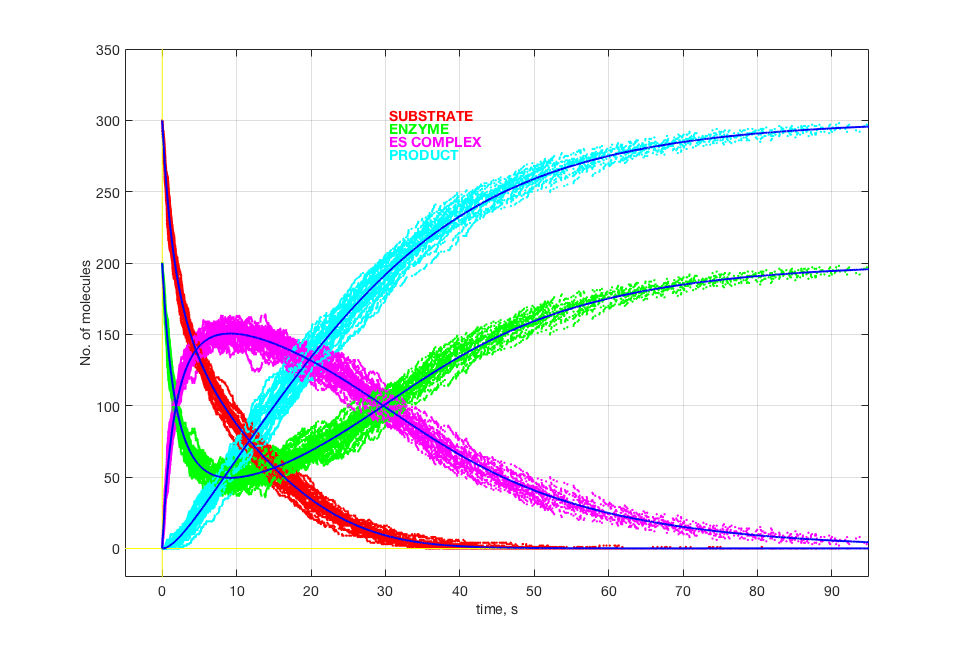
row3 = [ k1\*U2 0 -(k2+k3) 0 ];

row4 = [ 0 0 k3 0 ];

end

Here we superimpose the result on the SSA plot as continuous blue lines:

plot(time\_vec(1:end),U(:,1:end)\*Volume\*Avog\_Number,'-b','LineWidth',2)



Tau leaping and Chemical Langevin Equation.

The SSA reproduces exactly the statistics from the CME. Although straightforward to implement, the SSA becomes very slow to compute when there are too many molecules or some fast reactions, so that it becomes necessary to update the propensity functions very frequently. An approximate way of accelerating the sampling of the CME, is to advance the system through possibly many reaction events at once in each time step , so that updating of the state vector occurs only after many reactions have fired. This gives the *tau-leaping method*. The fundamental premise of this method is the *leap condition*:

*Leap condition*: an accuracy-assuring restriction on tau-leaping that requires to be small enough that no propensity function changes by a significant amount.

If we choose a sufficiently small *τ* interval, so that few reactions fire, the propensity functions remain essentially constant during , because very little updating occurs. Under these conditions, the number of times the *j* reaction channel fires in behaves as a statistically independent *Poisson* random variable with mean (and variance) . Therefore, to the degree that the leap condition is satisfied, we can approximately leap the system ahead by a time by taking:

This is the basic *tau leaping formula*. A trajectory with leap time can be computed as follows:

Initialize the time *t* = *t0* and the system’s state .

⇓

Choose a value of that satisfies the leap condition.

⇓

while t<T

For each reaction draw samples from the Poisson random variable with mean .

⇓

Set and update to .

V = [-1 1 0; -1 1 1; 1 -1 -1; 0 0 1];

Avog\_Number = 6.0221409e23;

Volume = 1e-14;

MM\_TAU = figure;set(gcf,'Unit','Normalized','Position',[0 0.4 0.5 0.6]);

for i = 1:30

nS = 300;

nE = 200;

nES = 0;

nP = 0;

k1 = 1e7/(Avog\_Number\*Volume);

k2 = 1e-3;

k3 = 5e-2;

t = 0;

tau = 1;

plot(t,nS,'.r','MarkerSize',0.5);

xlim([-5 95]);ylim([-20 350]);

box on;grid on;hold on;

plot(t,nE,'.g','MarkerSize',0.5);

plot(t,nES,'.m','MarkerSize',0.5);

plot(t,nP,'.c','MarkerSize',0.5);

tfinal = 90;

    while t < tfinal

        a1 = k1\*nS\*nE;

        a2 = k2\*nES;

        a3 = k3\*nES;

p(1,1) = poissrnd(a1\*tau);

p(2,1) = poissrnd(a2\*tau);

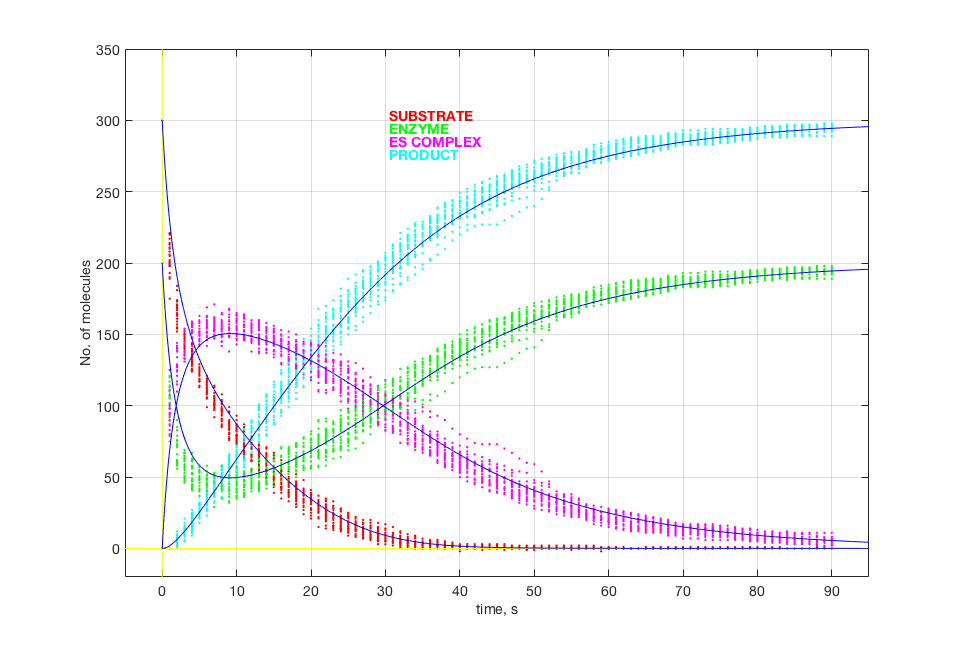
p(3,1) = poissrnd(a3\*tau);

        X = [nS nE nES nP]';

        X = X + V\*p;

        t = t + tau;

        nS = X(1);

        nE = X(2);

        nES = X(3);

        nP = X(4);

        plot(t,nS,'.r','MarkerSize',0.5);

        plot(t,nE,'.g','MarkerSize',0.5);

        plot(t,nES,'.m','MarkerSize',0.5);

        plot(t,nP,'.c','MarkerSize',0.5);

    end

end

hline([0 0],'-y');

vline([0 0],'-y');

xlabel('time, s');

ylabel('No. of molecules');

In practice, in the most current implementations of the algorithm, the leap time is chosen adaptively, based on the current state vector and propensity function values. This *tau-leaping method* is called ***explicit*** because of its similarity to explicit finite difference methods for the solution of initial value problems (CHAPTER 13). We recall here that a standard strategy for solving numerically an ODE of the type:

is to replace the explicit updating formula:

with an implicit formula:

in which the term appears on both sides of the equation. The *tau-leaping* formula:

is clearly an *explicit* updating formula. To make it *implicit* by replacing in the argument of the Poisson random variable with , however, raises some serious questions in the context of Markov process theory, because updates are supposed to be past-forgetting; moreover, even if that replacement could be justified theoretically, there appears to be no way to solve the resulting equation for . However, a partial implicitazion can be obtained with the following formula:

In this formula, the mean of the Poisson random variable is subtracted out and replaced by its value at the later time , but the variance has been left unchanged. Tests of this implicit tau-leaping strategy show that it produces significantly faster simulations than the explicit tau-leaping formula for stiff systems, at the risk however of damping excessively the fluctuations in the fast components of .

In MATLAB both the SSA method and the EXPLICIT and IMPLICIT TAU-LEAPING methods are implemented in the SimBiology Toolbox. The following shows how a single trajectory of the Michaelis Menten reaction can be obtained with all three methods using this toolbox:

Create Model

model = sbiomodel('Michaelis\_Menten');

Avog\_Number = 6.0221409e23; % Avagadro's number

Volume = 1e-14; % volume

Enter Reactions

r1 = addreaction(model, 's1 + s2 <-> s3');

r2 = addreaction(model, 's3 -> s4 + s2');

Set Reactions to be MassAction

kl1 = addkineticlaw(r1, 'MassAction');

kl2 = addkineticlaw(r2, 'MassAction');

Add Rate Constants for Each Reaction

p1f = addparameter(kl1, 'c1f', 'Value', 1e7/(Avog\_Number\*Volume));

p1r = addparameter(kl1, 'c1r', 'Value', 1e-3);

p2 = addparameter(kl2, 'c2', 'Value', 5e-2);

Set the Kinetic Law Constants for Each Kinetic Law.

kl1.ParameterVariableNames = {'c1f','c1r'};

kl2.ParameterVariableNames = {'c2'};

Specify Initial Amounts of Each Species

model.species(1).InitialAmount = 300;    % s1

model.species(2).InitialAmount = 200;    % s2

model.species(3).InitialAmount = 0; % s3

model.species(4).InitialAmount = 0; % s4

Display the Completed Model Objects

model

Display the Reaction Objects

model.Reactions

Display the Species Objects

model.Species

Get the Active Configuration Set for the Model.

cs = getconfigset(model,'active');

Plot the simulations

MM\_SSA\_TAU = figure;set(gcf,'Unit','Normalized','Position',[0 0.1 0.4 0.9]);

Run the SSA Stochastic Solver

tfinal = 90, logging every 10th datapoint.

cs.SolverType = 'ssa';

cs.StopTime = 90;

solver = cs.SolverOptions;

solver.LogDecimation = 10;

cs.CompileOptions.DimensionalAnalysis = false;

[t\_ssa, x\_ssa] = sbiosimulate(model);

h1 = subplot(3,1,1);

plot(h1, t\_ssa, x\_ssa,'.');

grid(h1,'on');

legend(h1, 'S1', 'S2', 'S3', 'S4','location','Best');

ylabel(h1,'No. of molecules');

xlabel(h1,'time, s');

title(h1,'SSA simulation');

Run the Explicit Tau-Leaping Solver

tfinal = 90, logging every 2nd datapoint.

cs.StopTime = 90;

cs.SolverType = 'explTau';

solver = cs.SolverOptions;

solver.LogDecimation = 2;

[t\_etl, x\_etl] = sbiosimulate(model);

h2 = subplot(3,1,2);

plot(h2, t\_etl, x\_etl,'.');

grid(h2,'on');

legend(h2, 'S1', 'S2', 'S3', 'S4','location','Best');

ylabel(h2,'No. of molecules');

xlabel(h2,'time, s');

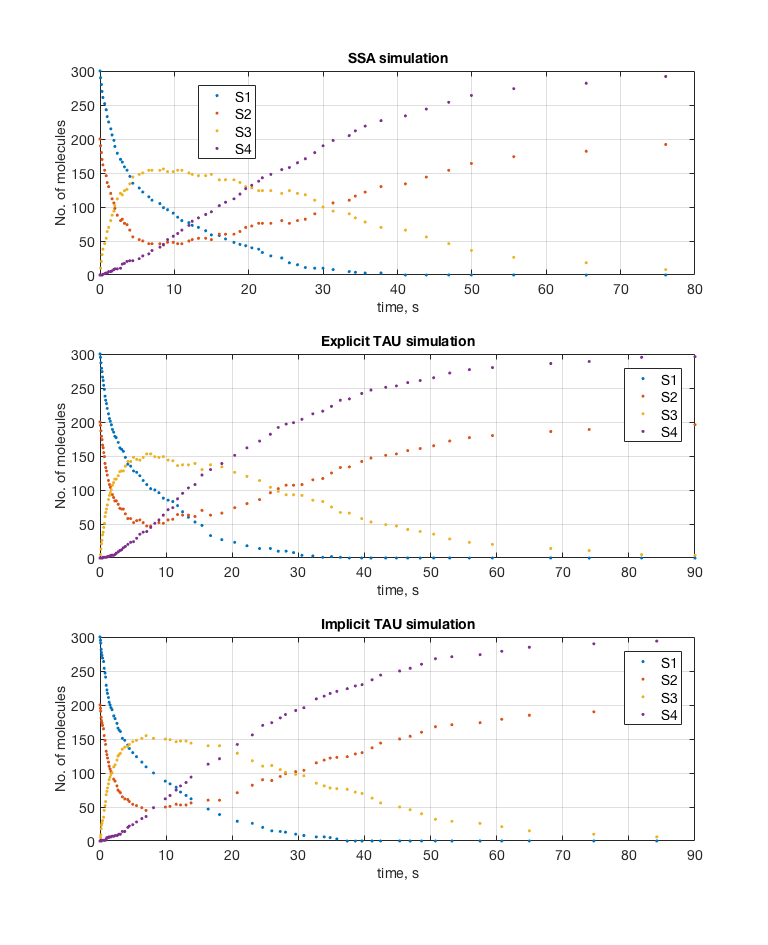
title(h2,'Explicit TAU simulation');

Run the Inplicit Tau-Leaping Solver

tfinal = 90, logging every 2nd datapoint.

cs.StopTime = 90;

cs.SolverType = 'impltau';

solver = cs.SolverOptions;

solver.LogDecimation = 2;

[t\_itl, x\_itl] = sbiosimulate(model);

h3 = subplot(3,1,3);

plot(h3, t\_itl, x\_itl,'.');

grid(h3,'on');

legend(h3, 'S1', 'S2', 'S3',…

'S4','location','Best');

ylabel(h3,'No. of molecules');

xlabel(h3,'time, s');

title(h3,'Implicit TAU simulation');

Compare the number of steps for SSA and Explicit Tau-Leaping Algorithms

fprintf(SSA steps: %d\n', (length(t\_ssa) \* 10));

fprintf(Explicit Tau-Leaping steps: %d\n', ...

(length(t\_etl) \* 10));

fprintf(Implicit Tau-Leaping steps: %d\n', ...

(length(t\_itl) \* 10));

SSA steps: 620

Explicit Tau-Leaping steps: 730

Implicit Tau-Leaping steps: 690

If we suppose that is not only small enough to satisfy the leap condition, but also large enough that the expected number of firings of each reaction channel during is >>1, Then, denoting the normal (Gaussian) random variable with mean *m* and variance *σ2* by *N*(*m*, *σ2*), and invoking the mathematical fact that a Poisson random variable with a mean and variance that is >>1 can be approximated as a normal random variable with that same mean and variance, we can approximate the tau-leaping formula:

as:

Collecting terms, we obtain the *Langevin Leaping Formula*:

Computationally, we can simulate a trajectory of this process by taking steps of the following form:

Initialize the time *t* = *t0* and the system’s state .

⇓

For each reaction draw independent samples

from the normal (0,1) distribution.

while t<T

⇓

Set

⇓

update to .

The following is the implementation for the Michaelis-Menten reaction:

V = [-1 1 0; -1 1 1; 1 -1 -1; 0 0 1];

Avog\_Number = 6.0221409e23;

Volume = 1e-14;

LANGEVIN\_TAU = figure;set(gcf,'Unit','Normalized','Position',[0 0.4 0.5 0.6]);

annotation(LANGEVIN\_TAU,'textbox',...

iter = 0;

for i = 1:30

    iter = iter + 1

    nS = 300;

    nE = 200;

    nES = 0;

    nP = 0;

    k1 = 1e7/(Avog\_Number\*Volume);

    k2 = 1e-3;

    k3 = 5e-2;

    t = 0;

    plot(t,nS,'.r','MarkerSize',0.5);

    xlim([-5 95]);ylim([-20 350]);

    box on;grid on;hold on;

    plot(t,nE,'.g','MarkerSize',0.5);

    plot(t,nES,'.m','MarkerSize',0.5);

    plot(t,nP,'.c','MarkerSize',0.5);

    tfinal = 90;

    L = 300; % total number of iterations

    tau = tfinal/L; % stepsize

    for k = 1:L

        a1 = k1\*nS\*nE;

        a2 = k2\*nES;

        a3 = k3\*nES;

        d(1,1) = tau\*a1 + sqrt(abs(tau\*a1))\*randn;

        d(2,1) = tau\*a2 + sqrt(abs(tau\*a2))\*randn;

        d(3,1) = tau\*a3 + sqrt(abs(tau\*a3))\*randn;

        X = [nS nE nES nP]';

        X = X + V\*d;

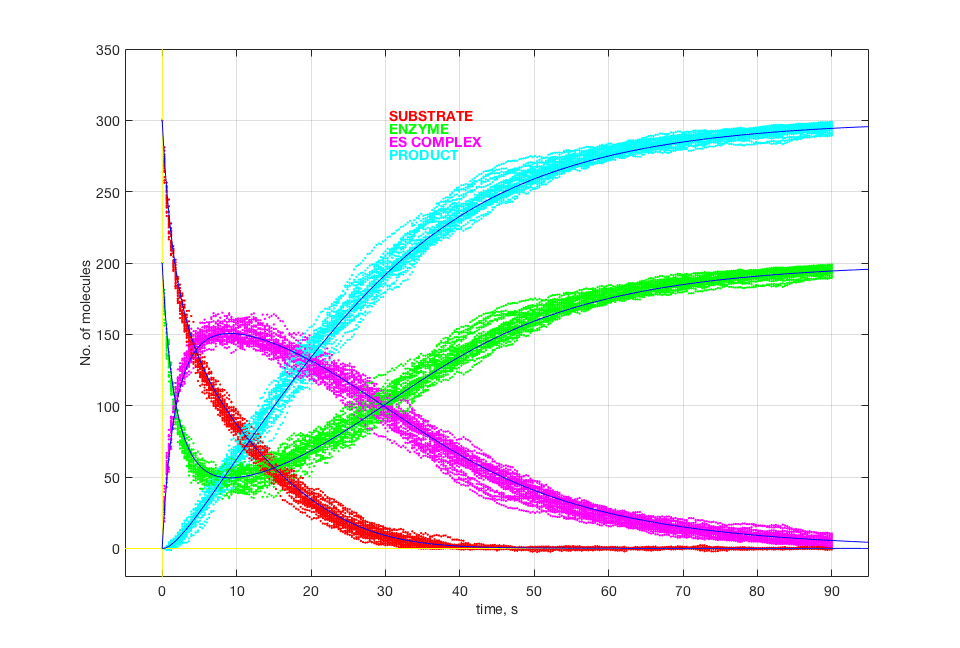
        t = t + tau;

        nS = X(1);

        nE = X(2);

        nES = X(3);

        nP = X(4);

        plot(t,nS,'.r','MarkerSize',0.5);

        plot(t,nE,'.g','MarkerSize',0.5);

        plot(t,nES,'.m','MarkerSize',0.5);

        plot(t,nP,'.c','MarkerSize',0.5);

    end

end

hline([0 0],'-y');

vline([0 0],'-y');

xlabel('time, s');

ylabel('No. of molecules');

To which we can superimpose the RRE eigensolution (shown as a continuous blue line).

We recognize here that what used to be *integer-valued* Poisson random variables in the *Langevin Leaping Formula*:

are now *real valued* normal random variables, and real numbers now describe the amount of each species. Bringing the on the left-hand side we obtain:

where we observe that the increment is expressed as the sum of two terms:

1. a *deterministic term* proportional to
2. a *stochastic term* proportional to

Finally taking the limit we obtain the *stochastic differential equation* (SDE) known as the *Chemical Langevin Equation* (CLE):

where the term is a *Gaussian white noise* representing independent scalar Brownian motions. It is straightforward to notice that if we remove the stochastic part of the CLE we obtain:

which correspond to the traditional deterministic model of chemical kinetics. In fact, we recall here that the propensity functions of different types of reactions are constructed as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| *Reaction type* | Reaction |  | Propensity function |
| *1st order* |  |  |  |
|  |  |  |  |
| *2nd order* |  |  |  |
|  |  |  |

Here we recognize the *propensity functions* as *fluxes* (=velocities, ) expressed in terms of molecule numbers rather than concentrations. In CHAPTER 13 (Special Topic: Topology and dynamics of a network of reactions) we have learned that an equivalent representation of the time derivatives of the concentration vector ***x*** can be derived with a new matrix representing the *linear transformation* of the flux vector ***v*** (that contains the reaction rates):

replaces the traditional representation

where ***S*** is the *stoichiometric matrix* of the reaction (see also CHAPTER 19). Since the *state change* vectors are the columns of the stoichiometric matrix, we derive:

Thus, a simple modification of the CLE code removing the stochastic part allows us to superimpose directly the corresponding RRE solution in terms of molecule numbers.

V = [-1 1 0; -1 1 1; 1 -1 -1; 0 0 1];

Avog\_Number = 6.0221409e23;

Volume = 1e-14;

RRE\_from\_CLE = figure;set(gcf,'Unit','Normalized','Position',[0 0.4 0.5 0.6]);

    nS = 300;

    nE = 200;

    nES = 0;

    nP = 0;

    k1 = 1e7/(Avog\_Number\*Volume);

    k2 = 1e-3;

    k3 = 5e-2;

    t = 0;

    plot(t,nS,'.r','MarkerSize',0.5);

    xlim([-5 95]);ylim([-20 350]);

    box on;grid on;hold on;

    plot(t,nE,'.g','MarkerSize',0.5);

    plot(t,nES,'.m','MarkerSize',0.5);

    plot(t,nP,'.c','MarkerSize',0.5);

    tfinal = 90;

    L = 300; % total number of iterations

    tau = tfinal/L; % stepsize

    for k = 1:L

        a1 = k1\*nS\*nE;

        a2 = k2\*nES;

        a3 = k3\*nES;

        d(1,1) = tau\*a1;

        d(2,1) = tau\*a2;

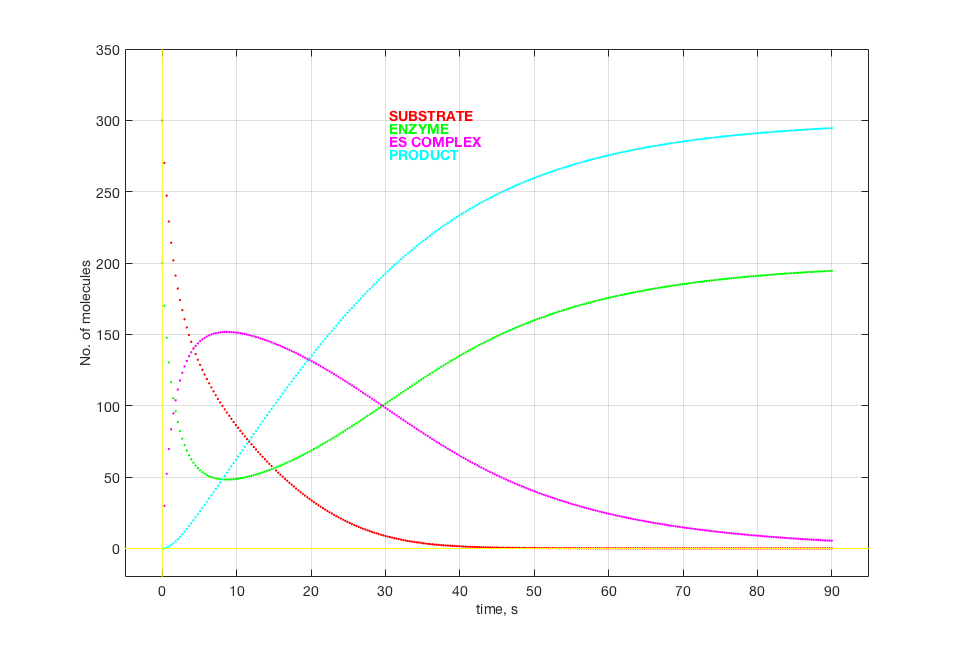
        d(3,1) = tau\*a3;

        X = [nS nE nES nP]';

        X = X + V\*d;

        t = t + tau;

        nS = X(1);

        nE = X(2);

        nES = X(3);

        nP = X(4);

        plot(t,nS,'.r','MarkerSize',0.5);

        plot(t,nE,'.g','MarkerSize',0.5);

        plot(t,nES,'.m','MarkerSize',0.5);

        plot(t,nP,'.c','MarkerSize',0.5);

    end

hline([0 0],'-y');

vline([0 0],'-y');

xlabel('time, s');

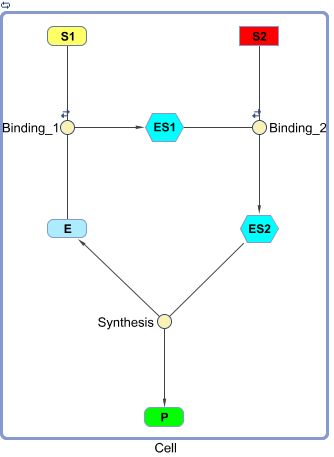
ylabel('No. of molecules');

Notice the logical path from the CME through the CLE to the RRE. The *thermodynamic limit* of a chemical system is defined as the limit in which the species populations and the system volume all approach infinity, but in such a way that the species concentrations stay constant. As this limit is approached, *all propensity functions grow in direct proportion to the size of the system*. Therefore, the term on the left side of the CLE and the first term on the right side both grow like the system size:

whereas the second term on the right grows more slowly as the square root of the system size. In the full thermodynamic limit, the last term becomes negligibly small compared with the other terms, and the CLE reduces to the RRE.

Thus, the important point to stress here is that the RRE solution, in general, does not correspond to the “average” solution from the CLE or CME, but corresponds instead to its *thermodynamic limit*.

**PRACTICE**



Additional MATLAB scripts allowing the simulation of different types of enzyme mechanisms are provided in the directory:

../TOOLBOXES/ENZYME\_KINETICS

Global fit of enzyme reactions progress curves is a type of non-linear least-squares. Practice the generation of synthetic data and its global analysis running the scripts:

Simple\_Michaelis\_Menten\_Global\_Fit

Comp\_Inhib\_Michaelis\_Menten\_Global\_Fit

Bi\_Uni\_Michaelis\_Menten\_Global\_Fit

Using as examples these scripts and the MATLAB program for non-linear least-squares analyzed in CHAPTER 8, write a program for the global fit of experimental kinetic data obtained with an enzyme that catalyzes a reaction of the Bi-Uni type (shown on the side).

For this purpose, from the same directory load:

1. The enzyme model contained in the SimBiology project:

Bi\_Uni\_Michaelis\_Menten

1. The experimental data (Times, Substrate Concentrations, Product Concentrations) in the files:

Bi\_Uni\_Michaelis\_Menten\_Time\_mat.txt

Bi\_Uni\_Michaelis\_Menten\_Substrate\_mat.txt

Bi\_Uni\_Michaelis\_Menten\_Product\_mat.txt

1. Using the same experimental data, show that the global fit is much poorer if you use a model of the enzyme (Bi\_Uni\_Michaelis\_Menten\_Switch\_Substrates) in which the order of binding of the substrates is inverted.