Elin Nyman
Bayesian Learning project
2013-01-15

Bayesian Learning and Systems Biology

Introduction

My idea for this project is to apply my knowledge from the course Bayesian Learning to my own research within systems biology. In systems biology mathematical tools such as modeling and simulation are used to analyze biological data (1). A common approach within systems biology is to create ordinary differential equation (ODE) models to study the dynamics of the system and to be able to analyze data from many proteins/molecules/cells/organs at the same time. The parameters in such models cannot be measured directly in most biological systems – instead optimization routines are used to fit the model simulations to the available data from the system. An alternative method to the optimization algorithms is to use some kind of sampling within a Bayesian framework. I will here create a Metropolis-Hastings sampling algorithm and use it to analyze a simple systems biology problem.

In my research I use mathematical modeling to analyze data from the insulin receptor signaling pathway (2-4). The data is obtained from fat cells treated with insulin and I study the dynamics of the insulin receptor activation as well as the dynamic activation of downstream signaling intermediates. In this project I will use data from the activated form of the insulin receptor, and a simple model of the insulin receptor including the receptor in an inactive form (A1), in an active form (A2), and in an internalized form (A3).

Theory and Methods

The model I use in this project is a simple system of ordinary differential equations (ODEs) (Matlab ODE model in Appendix A):

d/dt(A1) = k3*A3 - k1*A1
d/dt(A2) = k1*A1 - k2*A2
d/dt(A3) = k2*A2 – k3*A3

where A1-A3 are states corresponding to different forms of the insulin receptor A, and θ = k1-k3 are the model parameters that determines the rate of change going from A1 to A2 and A2 to A3, respectively. The initial conditions of the system, which sum to 100 %, are:

A1(0) = 90
A2(0) = 1
A3(0) = 9

The simple ODE model is illustrated in Figure 1.

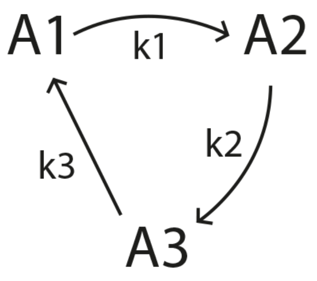

Figure 1: The model of the insulin receptor where A1 is the inactive state, A2 the active state, and A3 the internalized state. The model parameters k1-k3 determines the rate of the reactions.

The activated form of the insulin receptor (A2) can be measured, i.e. the available data (y) is given by:

y = A2

We have stimulated the cells with insulin and made 14 measurements over 6 minutes repeated 6 times to catch the initial dynamics of the receptor signaling (Figure 2). The activation response is transient with a maximal value at about half a minute and the steady state is reached after 1-3 minutes.

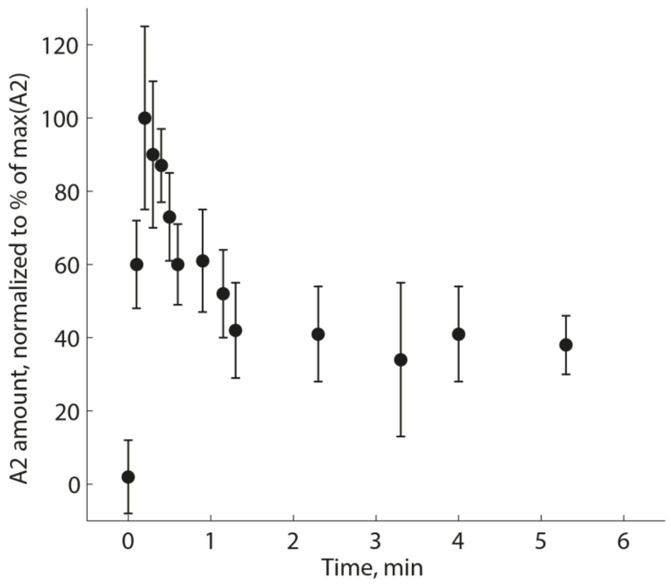

Figure 2: The activated form of the receptor (A2) measured over 6 minutes with mean values and standard deviation indicated (n = 6). The data is normalized to the maximal response.

I chose to work in log-space since the model parameters (θ) are bounded to be positive. The log-likelihood for the model (assuming a Gaussian error model with parameters $\hat{y}$ and σ) is:

$$\log(p(y \mid \theta)) = \Sigma(\log(1/(\sqrt{(2*\pi*\sigma\ )}))-(1/2*((y-\hat{y})/\ \sigma)\hat{}2))$$

The model parameters are not possible to measure, i.e. the values are unknown. Also, the parameters are probably not uniquely identifiable using the limited available data, i.e. there will be many different combinations of parameters that will give the same behavior of the model and about the same value of the likelihood function. To constrain the parameters to somewhat realistic values within a chosen range I select a bounded log uniform prior on the model parameters:

$$\log(p(\theta)) = 1/(\log(\theta_{max})-\log(\theta_{min}))$$

As realistic boundaries for the parameters I chose θ_{max} =20 and θ_{max} =0.01 for all parameters. The log posterior is the sum of the log-likelihood and the log-prior:

$$\log(p(\theta \mid y)) = \log(p(y \mid \theta)) + \log(p(\theta))$$

To be able to sample from the log posterior I use a Metropolis-Hastings algorithm that I have implemented in Matlab (Appendix B). Metropolis-Hastings is a Markov Chain Monte Carlo (MCMC) sampling method to use when the posterior probability distribution is unknown. The Metropolis-Hastings algorithm will provide a sequence (a chain) of random samples of the parameters, θ. I check the chain for convergence, i.e. that enough random samples are drawn, both by studying the jumps of the individual parameters, and using a function `geweke` in Matlab that calculates the means of the first 10 % and the last 50 % of the chain and test for equality. I included the code to this function in Appendix C.

My Metropolis-Hastings algorithm has the following inputs: number of samples to draw, starting guess for the draws, data, low and high parameters boundaries (θ_{max} and θ_{max}), and a function `logPost` that calculates the log posterior probability (Appendix D). The outputs are the draws, the posterior probability at the draws, and the rate of acceptance. The algorithm proposes a draw from a normal distribution that is centered on the last accepted draw (first time on the starting guess) and truncated between θ_{max} and θ_{max}. The variance in the truncated normal distribution is chosen so that approximately 20 % of the draws are accepted. The posterior probability of the draw is calculated and compared with the previous accepted draw and accepted with a certainty (for details, see Appendix B).

Results

The script used to create all plots in the Result part is found in Appendix E. The first step was to use my Metropolis-Hastings algorithm (Appendix B) to come up with a good starting guess of the unknown parameters. I simulated 10000 draws starting with θ = (k1=1, k2=1, k3=1) and calculated an approximation of the marginal density for each of the parameters using the function `ksdensity` in Matlab (Figure 3).

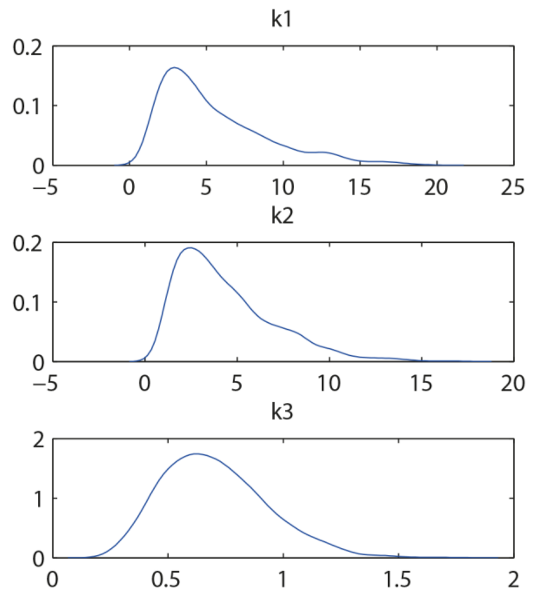

Figure 3: The estimated marginal density of the parameters k1-k3.

The maximal marginal density for the parameters are approximately θ = (k1=3, k2=2.2, k3=0.6), see Figure 3, and I use these values for my main sampling. I also look at the correlations between the parameters using the function `mcmcplot` in Matlab. The parameters k1 and k2 are clearly (non-linearly) correlated (Figure 4, top), and there are also correlations between the other pairs of parameters, although not as clear (Figure 4, bottom). I expected correlations among all parameters since I do not think the parameters are uniquely identifiable using the limited available data.

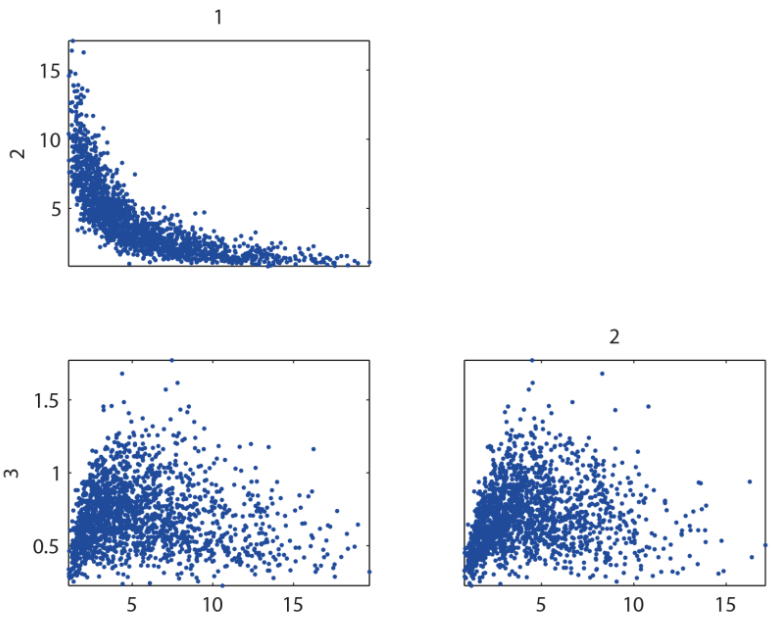

Figure 4: The correlations between the parameters k1 and k2 (top), k1 and k3 (bottom left), and k2 and k3 (bottom right).

I created 10000 draws in the main simulation run, and with an acceptance rate of approximately 0.17 there was 1732 unique parameter values (see Appendix E for the results script). The different draws of k1-k3 are illustrated in Figure 5, with the starting value (obtained from the maximal marginal density, see Figure 3) marked in bold.

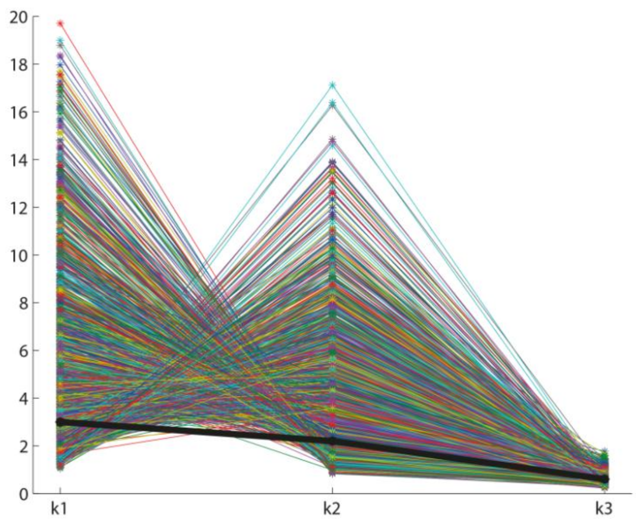

Figure 5: The different simulated draws of values of the parameters k1, k2, and k3 in the main run with the starting guess marked in bold.

To be able to decide if 10000 draws were enough for convergence of my Metropolis-Hastings algorithm I studied the convergence of each parameter, as illustrated in Figure 6. It is not possible to determine the convergence in this plot, and I also used the function `geweke` in Matlab that returned p = [0.8069 0.7941 0.9332]. This function calculates a measurement of the equality between the first 10 % and the last 50 % of the chain – the chain can be considered converged if $p > 0.7$ for all sub-dimensions of the chain. Taken this together I consider 10000 iterations to be enough to reach convergence with my Metropolis-Hastings algorithm.

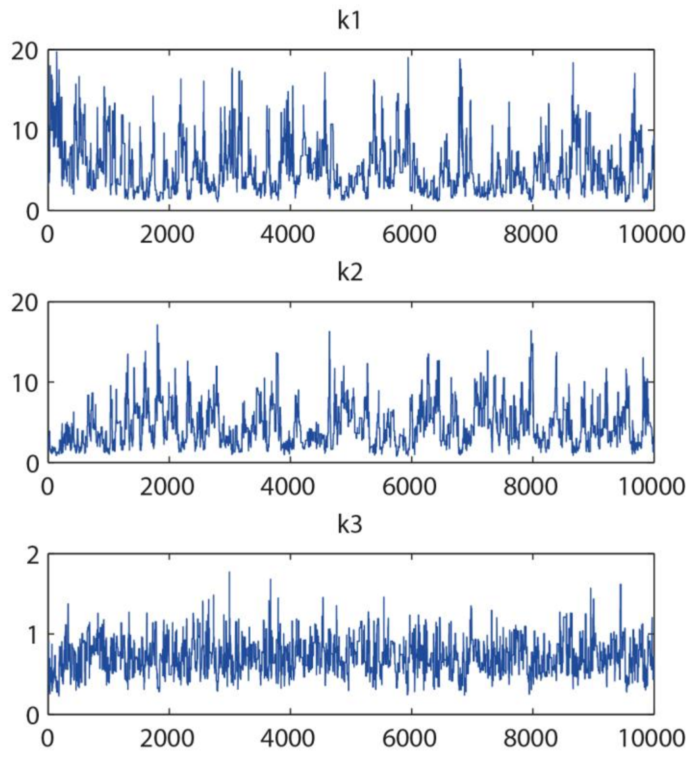

Figure 6: The 10000 iterations in the main run resulted in these draws of k1-k3.

The parameter draws can now be used to simulate the ODE model and thus to study the model dynamics. The fit between the A2 simulations using the draws of the parameters together with the data is illustrated in Figure 7. The starting guess is marked in bold. There are good agreements between simulations and data and there is a wide spread in the found solutions, which is good since I want to make predictions about non-measured states. To be able to make valid predictions we need not only the best parameters, but a good approximation of all the parameters that together with the ODE model possibly could have created the data.

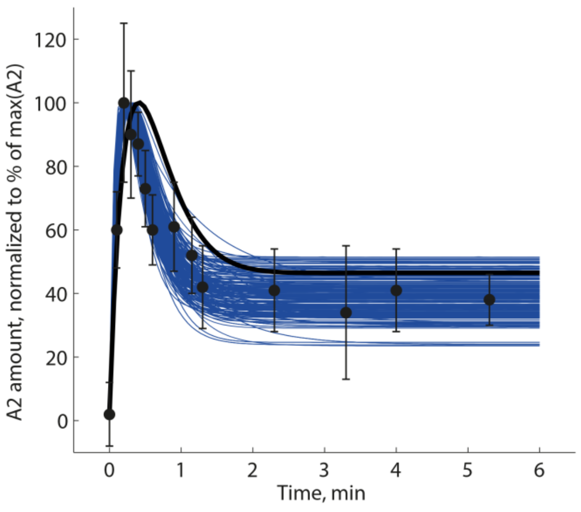

Figure 7: The agreement between the data and the simulated response of A2 using the ODE model and the 10000 draws of the parameters (every 10th of the unique accepted draws are simulated). The starting guess is marked in bold

An interesting prediction to study in this project is the dynamics of the internalized state of the receptor, A3 (Figure 8). The receptor internalization is believed to be important in the formation the transient response of the insulin receptor (5). For all the different parameter draws the simulated A3 reaches a steady-state where 60-85 % of the insulin receptors are internalized after 1-5 minutes (Figure 8). Such a prediction is possible to test experimentally.

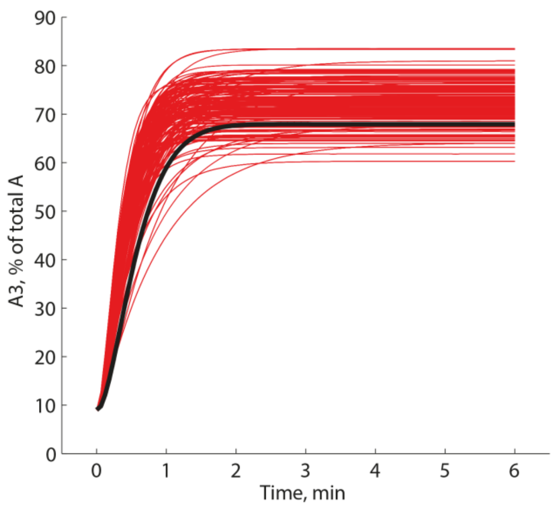

Figure 8: The predicted dynamics of A3 using the ODE model and the 10000 draws of the parameters (every 10th of the unique accepted draws are simulated). The starting guess is marked in bold.

Discussion

I have applied Bayesian methods to my own research within systems biology. It enjoyed this small project and I will try to go on with more advanced models with more states and more measurements to evaluate the usefulness of Bayesian methods in my research. An important issue to solve to be able to scale up the problem is to speed up the simulations of the ODE model. This is possible using MEX (Matlab executable) files for the simulations, and since I already use these files for simulation when doing optimizations, this should be straight forward. When comparing optimization routines and sampling methods I find the sampling methods to be better in finding a broad range of possible solutions for the parameters. In my research I use optimization methods that are specialized to find not a single global optimum, but many different local optimums that give a fair agreement between model and data. The reason for this choice of optimization methods are that I want to make predictions that will hold for the model structure independent of the chosen parameters. I see a strong potential in the Bayesian sampling methods in achieving such predictions.

References

1. Kitano, H. (2002) Systems biology: a brief overview. *Science* **295**, 1662-1664
2. Nyman, E., Brännmark, C., Palmér, R., Brugård, J., Nyström, F. H., Strålfors, P., and Cedersund, G. (2011) A hierarchical whole-body modeling approach elucidates the link between in Vitro insulin signaling and in Vivo glucose homeostasis. *J Biol Chem* **286**, 26028-26041
3. Nyman, E., Fagerholm, S., Jullesson, D., Strålfors, P., and Cedersund, G. (2012) Mechanistic explanations for counter-intuitive phosphorylation dynamics of the insulin receptor and insulin receptor substrate-1 in response to insulin in murine adipocytes. *FEBS J*
4. Nyman, E., Cedersund, G., and Stralfors, P. (2012) Insulin signaling - mathematical modeling comes of age. *Trends Endocrinol Metab* **23**, 107-115
5. Brännmark, C., Palmér, R., Glad, S. T., Cedersund, G., and Strålfors, P. (2010) Mass and information feedbacks through receptor endocytosis govern insulin signaling as revealed using a parameter-free modeling framework. *J Biol Chem* **285**, 20171-20179

Appendix A – the ODE model

```
function ydot = odeModel(t,y,param)
A1=y(1); A2=y(2); A3=y(3);
ydot = [
    - param(1)*A1                     + param(3)*A3;
    + param(1)*A1    - param(2)*A2;
                     + param(2)*A2    - param(3)*A3;
       ];
```

Appendix B – Metropolis-Hastings algorithm

```
function [draws postProb acceptanceRate] =
metropolis_hastings(N,param,data,low,high,FUN)

draws = zeros(N+1,length(param));
postProb = zeros(N,1);
draws(1,:) = param;
currentProb = feval(FUN,param,data,low,high);
acceptCounter = 0;
variance = 0.5;
propDraw = [0 0 0];

for i = 2:N+1

    for j = 1:3
        propDraw(j) = truncnormrnd(1,draws(i-1,j),variance,low,high);
    end

    newProb = feval(FUN,propDraw,data,low,high);
    postProb(i-1) = newProb;
    acceptProb = min([exp(newProb-currentProb) 1]);
    accept = rand;

    if accept < acceptProb
        draws(i,:) = propDraw;
        currentProb = newProb;
        acceptCounter = acceptCounter+1;
    else
        draws(i,:) = draws(i-1,:);
    end

end
        acceptanceRate = acceptCounter/N;
end
```

Appendix C – Geweke's MCMC convergence diagnostics

```
function [z,p]=geweke(chain,a,b)
%GEWEKE Geweke's MCMC convergence diagnostic
% [z,p] = geweke(chain,a,b)
% Test for equality of the means of the first a% (default 10%) and
% last b% (50%) of a Markov chain.
% See:
% Stephen P. Brooks and Gareth O. Roberts.
% Assessing convergence of Markov chain Monte Carlo algorithms.
% Statistics and Computing, 8:319--335, 1998.

% ML, 2002
% $Revision: 1.3 $  $Date: 2003/05/07 12:22:19 $

[nsimu,npar]=size(chain);

if nargin<3
  a = 0.1;
  b = 0.5;
end

na = floor(a*nsimu);
nb = nsimu-floor(b*nsimu)+1;

if (na+nb)/nsimu >= 1
  error('Error with na and nb');
end

m1 = mean(chain(1:na,:));
m2 = mean(chain(nb:end,:));

%%% Spectral estimates for variance
sa = spectrum0(chain(1:na,:));
sb = spectrum0(chain(nb:end,:));

z = (m1-m2)./(sqrt(sa/na+sb/(nsimu-nb+1)));
p = 2*(1-nordf(abs(z)));
```

Appendix D – logPost

```
function output = logPost(param,data,high,low)

time = data.ydata(:,1);
response = data.ydata(:,2);
sigma = data.ydata(:,3);
initalValues = data.initial;
[t,y] = ode45(@odeModel,time,initalValues,[],exp(param));

% normalization of model simulation
simResponse = y(:,2)./max(y(:,2)).*100;

% log-likelihood
logLik = sum(log(1./(sqrt(2*pi*sigma)))-(1/2*((response-
simResponse)./sigma).^2));

if(abs(logLik) == Inf)
    logLik = -20000;
end

% distribution of log uniform prior
logPrior = 1/(high-low);

output = logLik + logPrior;
```

Appendix E – all results script

```
data.ydata = [
0       2       10
0.1     60      12
0.2     100     25
0.3     90      20
0.4     87      10
0.5     73      12
0.6     60      11
0.9     61      14
1.15    52      12
1.3     42      13
2.3     41      13
3.3     34      21
4       41      13
5.3     38      8
    ];

A10 = 90; A20 = 1; A30 = 9;
data.initial = [A10;A20;A30];

N=10000;

low = log(0.01);
high = log(20);

% startGuess = [1;1;1]; % first run
startGuess = [3;2.2;0.6]; % main run
param = log(startGuess);

[draws postProb acceptanceRate] =
metropolis_hastings(N,param,data,low,high,@logPost);

acceptanceRate;
acceptedDraws = acceptanceRate*N;

% find all unique draws
j=find(draws(2:N+1,1)>draws(1:N,1));
k=find(draws(2:N+1,1)<draws(1:N,1));
params=[exp(draws(1,:));exp(draws(j,:));exp(draws(k,:));exp(draws(end,:))];

% estimate the marginal densities
[p_param1 param1]=ksdensity(params(10:end,1));
[p_param2 param2]=ksdensity(params(10:end,2));
[p_param3 param3]=ksdensity(params(10:end,3));

% plot the marginal densities (Figure 3)
figure()
subplot(3,1,1),plot(param1,p_param1)
title('k1')
subplot(3,1,2),plot(param2,p_param2)
title('k2')
subplot(3,1,3),plot(param3,p_param3)
title('k3')
```

```
% plot the parameter correlations (Figure 4)
figure()
mcmcplot(params,1:3,[],'pairs')

% plot the draws from the main run (Figure 5)'
figure()
hold on
plot([1 2 3],[params(2:end,1) params(2:end,2) params(2:end,3)],'*-')
plot([1 2 3],[params(1,1) params(1,2) params(1,3)],'k*-','LineWidth',5)
axis([0.9 3.1 0 20])
set(gca,'XTick',1:3);
set(gca,'XTickLabel',['k1';'k2';'k3']);

% plot the 10000 draws of the parameters to see convergence (Figure 6)
figure()
subplot(3,1,1),plot(1:N,exp(draws(1:end-1,1)))
title('k1')
subplot(3,1,2),plot(1:N,exp(draws(1:end-1,2)))
title('k2')
subplot(3,1,3),plot(1:N,exp(draws(1:end-1,3)))
title('k3')

% chose every 10 unique parameter
params = params(1:10:end,:);

resolution = 1000;

t = zeros(resolution,size(params,1),1);
y = zeros(resolution,size(params,1),3);

for i = 1:size(params,1)
    [t(:,i) y(:,i,:)] =
ode45(@odeModel,linspace(0,6,resolution),data.initial,[],params(i,:));
end

% plot the simulations of A2 and the data (Figure 7)
figure()
hold on
for i = 2:size(params,1)
    plot(t,y(:,i,2)./max(y(:,i,2)).*100,'b-')
end
plot(t,y(:,1,2)./max(y(:,1,2)).*100,'b:','LineWidth',3)
errorbar(data.ydata(:,1),data.ydata(:,2),data.ydata(:,3),'ko')
axis([-0.5 6.5 -10 130])
xlabel('Time, min')
ylabel('A2 amount, normalized to % of max(A2)')

% plot the simulations of A3 (Figure 8)
figure()
hold on
plot(t,y(:,1,3),'r:','LineWidth',3)
for i = 2:size(params,1)
    plot(t,y(:,i,3),'r-')
end
axis([-0.5 6.5 0 90])
xlabel('Time, min')
ylabel('A3, % of total A')
```