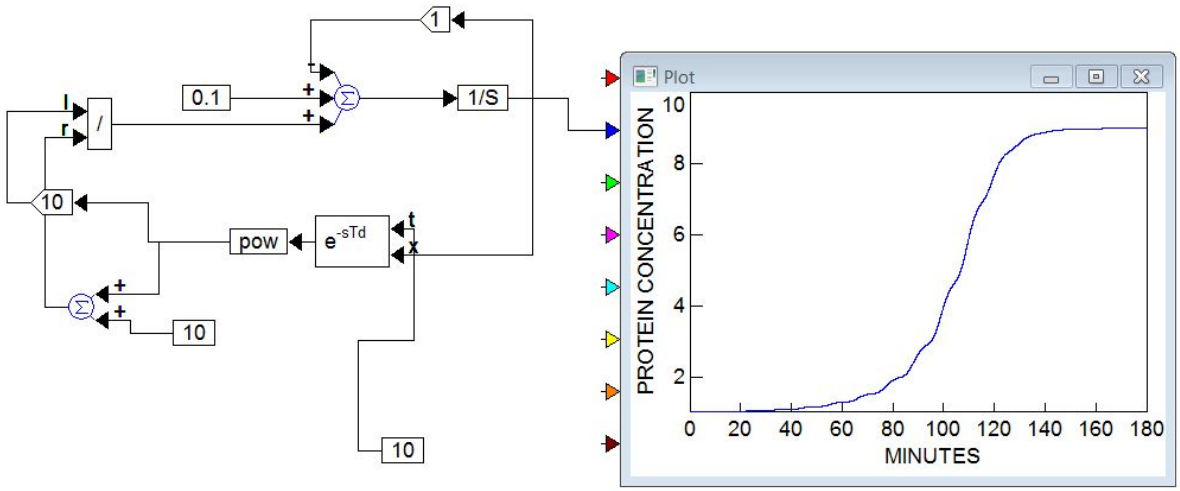
PROGRAMS & CODE FOR CHAPTER EXAMPLES & EXERCISES

CHAPTERS 1 & 2 – none

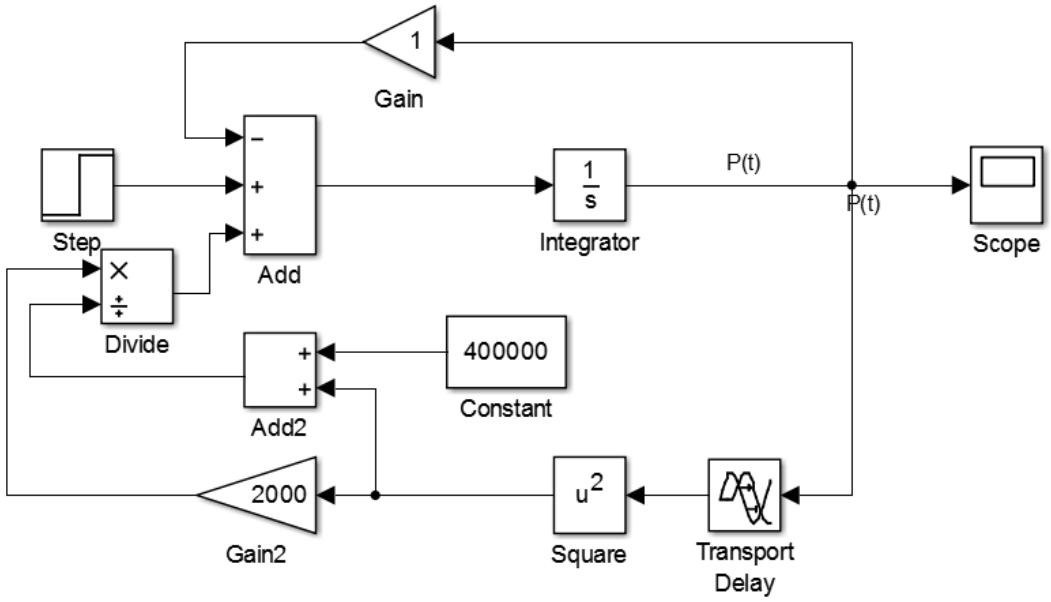
CHAPTER 3

Examples 3.2 – 3.7 *Simulink* and *VisSim* block diagram programs are given in Figures 3.5 – 3.12.

VisSim pgm for ***Exs 3.6*** and ***3.7***:



Simulink pgm for ***Exs 3.6*** and ***3.7***:



***Matlab* code for Fig. 3.8** (for copying and pasting) is:

```
function ddeFbk
%   ddeFbk solutions run with the DDE23 solver
%
%   Simple example of a feedback loop,
%   with an integrator in the forward path and
%   a unit time delay in the negative feedback
%   path. Solutions are stable for small delays
%   and unstable for large delays.
%
%   The ODE for this configuration is simply:
%
%        y'(t) = 1 - y(t-1)
%
%   We solve it over [0, tfinal] with history
%   (IC only) y(t) = 0 for t <= 0. The pure
%   lag is specified as a 1-vector (scalar)
%   here and the delay differential equation
%   is coded in the subfunction DDFbkDE.
%   Because the history is constant,
%   it is supplied as a scalar IC.
%  SETTING THE RUN PARAMETERS...
IC = 0;
delay1 = 0.5;
tfinal = 100;
interval = [0, tfinal];
s1 = dde23(@ddeFbkDE,delay1,IC, interval);
plot(s1.x,s1.y, 'LineWidth',2);

%  REPEATING FOR 2 MORE DELAYS…
delay2 = 1;
s2 = dde23(@ddeFbkDE,delay2,IC, interval);
plot(s1.x,s1.y, s2.x,s2.y, 'LineWidth',2);

%   Note: suppresss s3 and plot for running
%          only first 2 delays
delay3 = 2;
s3 = dde23(@ddeFbkDE,delay3,IC, interval);
plot(s1.x,s1.y, s2.x,s2.y,s3.x,s3.y,…
            'LineWidth',2);
title('SIMPLE TIME-DELAYED FEEDBACK');
xlabel('TIME');
ylabel('OUTPUT SIGNAL');
 ----------------------
% FUNCTION DEFINING THE ODE
function dydt = ddeFbkDE(t,y,Z)
% Differential equation function for ddeFbk.
ylag = Z(:,1);
dydt = 1 - ylag;
```

Example 3.12 The *VisSim* block diagram program is given in Figure 3.14.

Figure 3.22 Program Extras – WhatIf expts

```
% SIMPLE TRANSCRIPTION/TRANSLATION MODEL
%  RUN FILE FOR function TVHJWhatIfFinal.m
% code adapted from: Hayot & Jayaprakash(2008)
  % Time in minutes, Vol = 1
  % k1*DNA is mRNA production rate with D=1(#/t)
  % k3*M is protein production rate (#/t)
  % k2*mRNA degradation rate (#/t)
  % k4*P is Protein degradation rate (#/t)
global b  % pass burst factor b btwn the files
tspan=[0 1000];  % run and plot to  ~10 hrs
yzero=[0;0];      % zero ICs on mRNA and P

b = 2; % set burst factor

[t,y] = ode45(@TVHJWhatIfFinal,tspan,yzero);
plot(t,y,'linewidth', 2);

legend('mRNA # with b = 2', 'PROTEIN # with b = 2', 4);
title('TRANSCRIPTION RATE X 2 FROM 300 to 500 MINS')
xlabel('MINUTES')
ylabel('PROTEIN NUMBER')

Pss = y(length(t),2) % steady state values
Mss = y(length(t),1) %       printed

%  FUNCTION FILE FOR RunTVHJwhatifFinal.m
function dydt = TVHJWhatIfFinal(t,y)
global b % pass variable b btwn the files

% half-life of mRNA = 2 MIN, protein = 60 MIN
k2=log(2)/2; % rate const from half-life
k4=log(2)/60; % rate const from half-life
% b = BURST FACTOR, avg # proteins produced in...
%  mRNA lifetime, b*k2 = k3,in the eqn directly:

dydt = [variablek1(t)-k2.*y(1); b*k2.*y(1)-k4.*y(2)];
```

Program GUI for Figure 3.23 (chaos) is included with the figure.

Example 3.23 *Simulink* program diagram is given in Fig. 3.19 (page 128).
Matlab code for solving same problem (E3.5) and generating digital soln error is:

```
% Simple Call for ODE solver
```

```
[t, x] = ode45('Linear', [0,500], [1 0]);
z = ((1000/999)*exp(-1.*t) - (1/999)*exp(-1.*1000.*t));
delta = x(1)-z;
plot(t, z, t, x(:,1));
title('Solution of Stiff Linear Equation and Solution Error');
xlabel('TIME')
h=legend('z(t)','x(t)');
```

Example 3.27 – *VisSim* program diagrams given in Fig. 3.21 (page 133).

Exercise E3.9 – *VisSim* program is given in Fig. 3.23 (page 138).

Code for Runge-Kutta 4th-order algorithm for solving any ODEs:

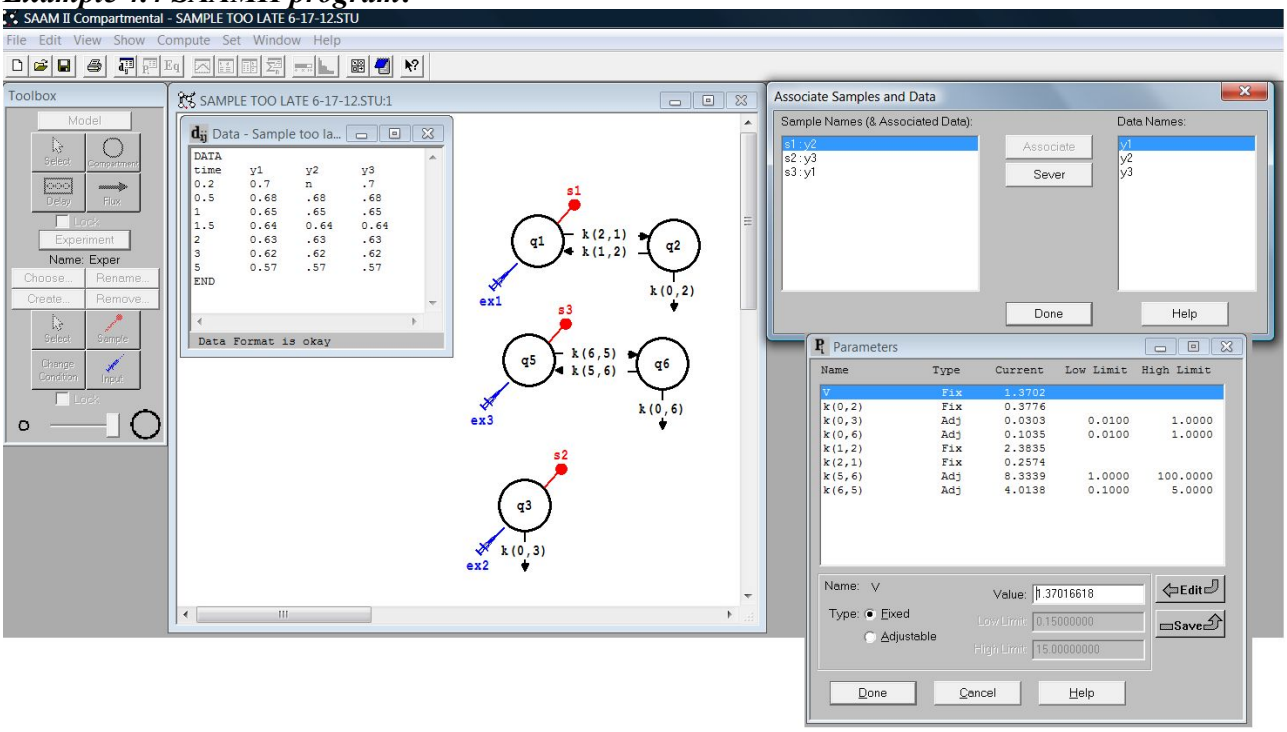
```
function [tp,yp] = rk4sys(dydt,tspan,y0,h,varargin)
% rk4sys: fourth-order Runge-Kutta for a system of ODEs
%   [t,y] = rk4sys(dydt,tspan,y0,h,p1,p2,...): integrates a
%                  system of ODEs with fourth-order RK method
% input:
%   dydt = name of the M-file that evaluates the ODEs
%   tspan = [ti, tf]; initial and final times with output
%                       generated at interval of h, or
%      = [t0 t1 ... tf]; specific times where solution output
%   y0 = initial values of dependent variables
%   h = step size
%   p1,p2,... = additional parameters used by dydt
% output:
%   tp = vector of independent variable
%   yp = vector of solution for dependent variables

if nargin<4,error('at least 4 input arguments required'),end
if any(diff(tspan)<=0),error('tspan not ascending order'), end
n = length(tspan);
ti = tspan(1);tf = tspan(n);
if n == 2
  t = (ti:h:tf)'; n = length(t);
  if t(n)<tf
    t(n+1) = tf;
    n = n+1;
  end
else
  t = tspan;
end
tt = ti; y(1,:) = y0;
np = 1; tp(np) = tt; yp(np,:) = y(1,:);
i=1;
while(1)
  tend = t(np+1);
  hh = t(np+1) - t(np);
  if hh>h,hh = h;end
```

```
  while(1)
    if tt+hh>tend,hh = tend-tt;end
    k1 = dydt(tt,y(i,:),varargin{:})';
    ymid = y(i,:) + k1.*hh./2;
    k2 = dydt(tt+hh/2,ymid,varargin{:})';
    ymid = y(i,:) + k2*hh/2;
    k3 = dydt(tt+hh/2,ymid,varargin{:})';
    yend = y(i,:) + k3*hh;
    k4 = dydt(tt+hh,yend,varargin{:})';
    phi = (k1+2*(k2+k3)+k4)/6;
    y(i+1,:) = y(i,:) + phi*hh;
    tt = tt+hh;
    i=i+1;
    if tt>=tend,break,end
  end
  np = np+1; tp(np) = tt; yp(np,:) = y(i,:);
  if tt>=tf,break,end
end
```

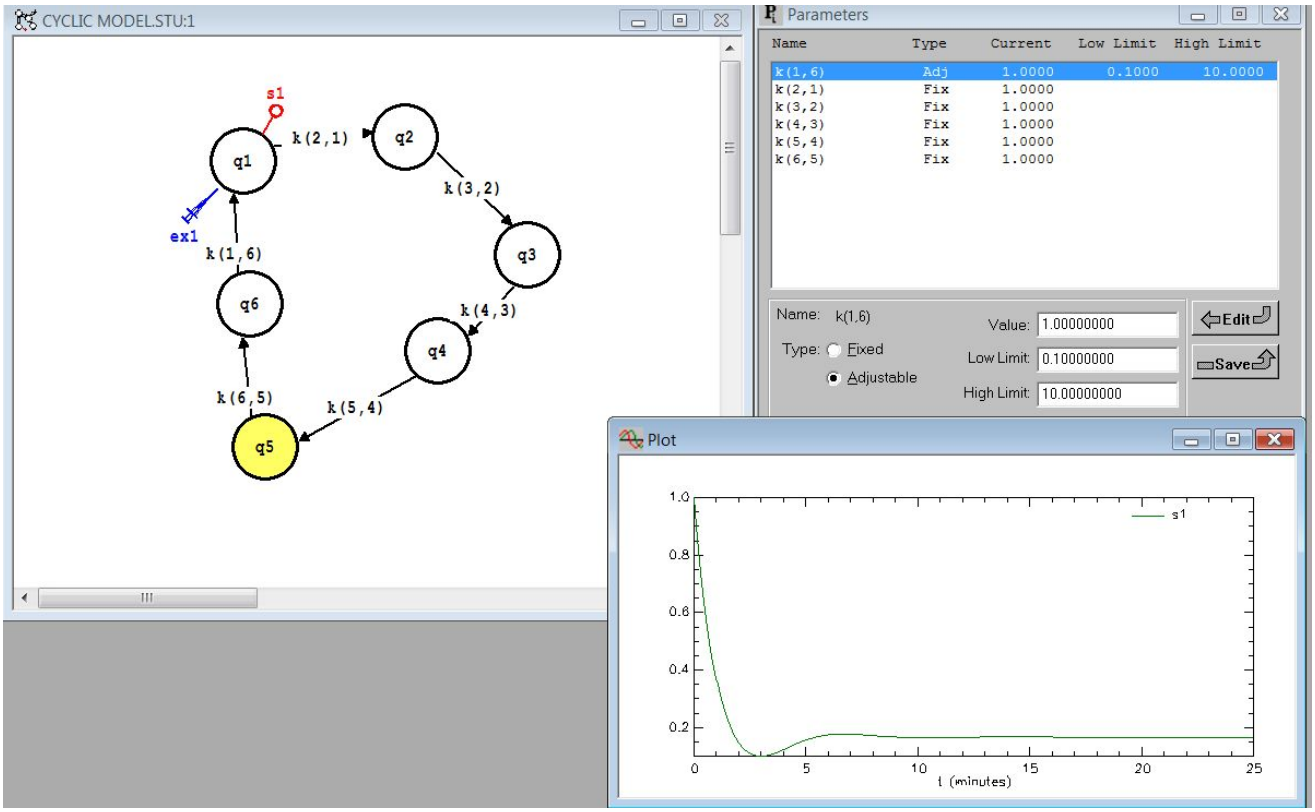
CHAPTER 4

Example 4.4 SAAMII program:



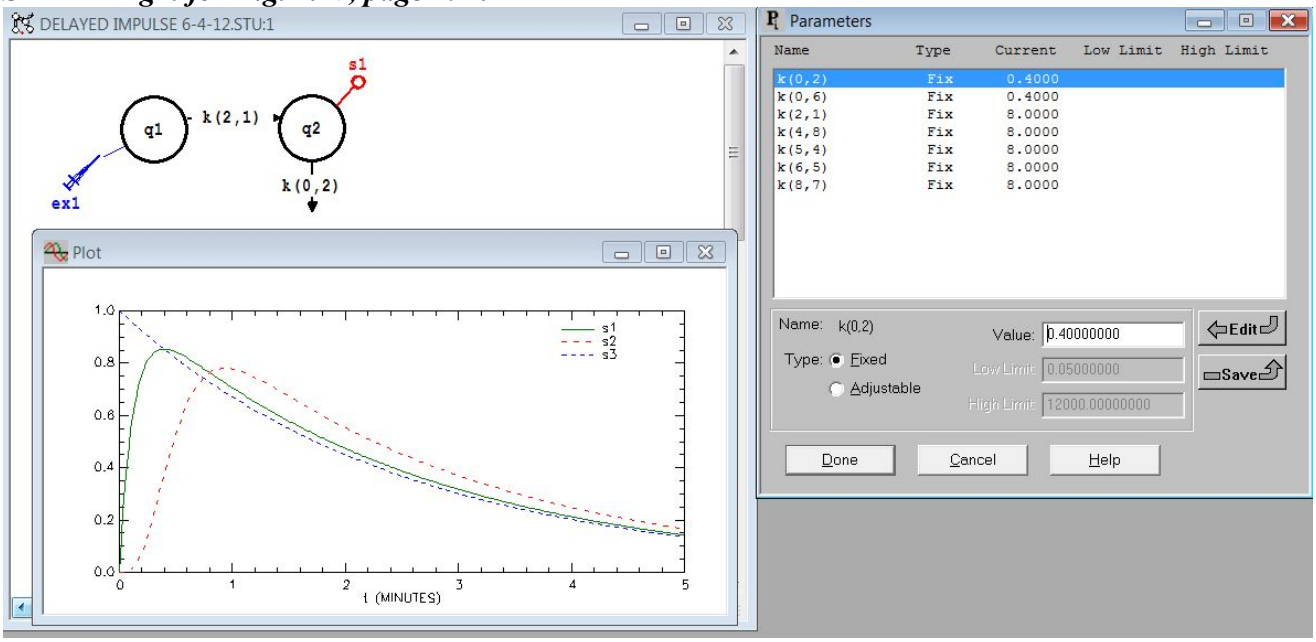
***Example 4.7 – Matlab* code is in Fig. 4.17, page 173.**

Example 4.10 Variant – 6-compartment Cyclic Model (Compare with Fig 4.22)



Example 4.13—Oral Dosing SAAMII pgm is in Fig. 4.30, page 194.

SAAMII Pgm for Fig. 4.27, page 191:



Exercise 4.18 – For 1-Comp NL model, use following code, with param values changed:

```
a = 3; b=1;
dxdt = @(t,x) (-a*x/(b+x)) + 1;
ode45(dxdt,[0,10],0)
```

For 2-comp NL model of Example 4.6 use the code in Fig. 4.17, with param values changed.

CHAPTER 5

Example 5.5 **– Hidden compartment *Matlab* code (for Fig. 5.6 page 220):**

```
t = 0:0.001:5;
k21 = 1;
y1 = exp(-k21*t);
y2a = k21*t.*exp(-k21*t);
k02 = k21/0.9;
y2b = (k21/(k21 - k02))*(exp(-k02*t) - exp(-k21*t));
k02 = k21/1.5;
y2c = (k21/(k21 - k02))*(exp(-k02*t) - exp(-k21*t));

plot(t,y1,'linewidth',2)
hold on
plot(t,y2a,'b','linewidth',2)
hold on
plot (t,y2b,'g','linewidth',2)
hold on
plot (t,y2c,'r','linewidth',2)
```

CHAPTER 6

Example 6.6 **– full code for generating Fig. 6.6 (Fig. 6.7 was abbreviated code):**

```
% FULL M-M ENZYME + QSSA + tQSSA DYNAMICS
% Reactions---R1: S + E -- k1 --> C;
% R2: C--kminus1-->E + S; R3: C--k2-->E + P
%
% Parameters
% the 60 converts k's to per min.
% Set kminus1 = 2/sec or 200/sec
global K1 KMINUS1 K2 e0 s0 Km;
% Rates
K1 = 60*10; KMINUS1 = 60*200; K2 = 60*0.02;
Km = (KMINUS1+K2)/K1;

% Initial Conditions
s0 = 1; e0 = 1; c0 = 0; p0 = 0;
x0 = [s0; e0; c0; p0; s0; p0; s0+c0; p0];
```

```
% Calculate QSSA metrics
tc = 1/(K1*(s0+Km));
ts = (s0+Km)/(e0*K2);
tstotal = (s0+e0+Km)/(e0*K2);
tctotal = 1/(K1*(e0 + s0 + Km));
epsilon = e0/(Km+s0);

disp(sprintf('\n----------Running MM Script----------\n'));
disp(sprintf('Intial substrate: %f\nInitial Enzyme: %f\n',s0,e0));
disp(sprintf('KM: %f\nFast time tc: %f\nSlow time ts: %f\n',Km,tc,ts));
disp(sprintf('tc/ts= %f\n',tc/ts));
disp(sprintf('tc_t/ts_t= %f\n',tctotal/tstotal));
disp(sprintf('k_2/(k_1(s0+e0)= %f\n',K2/(K1*(s0+e0))));
disp(sprintf('k_2/(k_-1(s0+e0)= %f\n',K2/KMINUS1));
disp(sprintf('epsilon=e0/(KM+s0): %f\n',epsilon));

% Solve
tspan = [0 5*ts];

[t,x]=ode15s(@MM_QSSA_tQSSA, tspan, x0);

% Plot QSSA
xmax = tspan(2);
ymax = max(s0,e0);
yoffset = ymax/10;
textFont = 18;
lengendFont = 16;
titleFont = 16;
axisFont = 16;
fig1=figure('PaperPosition',[2 2 8 6.3]);
ax = axes('FontName','cmr','FontSize',12,...
    'Position',[0.1264 0.15 0.7879 0.7643],...
    'Parent',fig1);
h = plot(t, x(:,1:6));
axis([0 xmax 0 ymax]);
set(h,{'LineWidth'},{3;3;3;3;2;2},...
    {'LineStyle'},{'--';':';'.';'-.';'o';'o'},...
    {'Color'},{'b';'k';'r';'m';'b';'m'});
title('MM ENZYME-SUBSTRATE DYNAMICS: QSSA','FontSiz',titleFont);
plottext1 = sprintf('k_-_1 = %d',KMINUS1/60);
plottext2 = sprintf('epsilon  = %.1f',epsilon);
plottext3 = sprintf('t_S  = %.1f' ,ts);
% Display values on plot
text(xmax/3,ymax/2 + yoffset,plottext1,'FontSize',textFont);
text(xmax/3,ymax/2,          plottext2,'FontSize',textFont);
text(xmax/3,ymax/2 - yoffset,plottext3,'FontSize',textFont);
xlabel(ax,'MIN','FontSize',axisFont);
ylabel(ax,'CONCENTRATION','FontSize',axisFont);
legend(ax,{'S','E','ES','P','S_Q_S_S_A','P_Q_S_S_A'},...
    'FontName','MIN','FontSize',lengendFont);
```

```
legend(ax,'boxoff')

plotfile = sprintf('k-1_%d_QSSA',KMINUS1/60);
print('-depsc','-tiff','-r300',plotfile);

% Plot Total QSSA
fig2=figure('PaperPosition',[2 2 8 6.3]);
ax = axes('FontName','cmr','FontSize',12,...
    'Position',[0.1264 0.15 0.7879 0.7643],...
    'Parent',fig2);
h = plot(t,[x(:,1) (x(:,1)+x(:,3)) x(:,4) x(:,7:8)]);
axis([0 xmax 0 ymax]);
set(h,{'LineWidth'},{3;3;3;2;2},...
    {'LineStyle'},{'--';'--';'-.';'d';'d'},...
    {'Color'},{'b';'g';'m';'g';'m'});
title('MM ENZYME-SUBSTRATE DYNAMICS: tQSSA','FontSiz',titleFont);
plottext1 = sprintf('k_-_1 = %d',KMINUS1/60);
plottext2 = sprintf('t_S_t_o_t_a_l  = %.2f',tstotal);
% Display values on plot
text(xmax/3,ymax/2          ,plottext1,'FontSize',textFont);
text(xmax/3,ymax/2 - yoffset,plottext2,'FontSize',textFont);
xlabel(ax,'MIN','FontSize',axisFont);
ylabel(ax,'CONCENTRATION','FontSize',axisFont);
legend(ax,{'S','S+C','P','S_t_Q_S_S_A','P_t_Q_S_S_A'},...
    'FontName','MIN','FontSize',lengendFont);
legend(ax,'boxoff')

plotfile = sprintf('k-1_%d_totQSSA',KMINUS1/60);
print('-depsc','-tiff','-r300',plotfile);
%.....................................
function [dxdt] = MM_QSSA_tQSSA(t,x)
%rate eqns for vanilla michaelis menten solver
global K1 KMINUS1 K2 Km e0;
% Complete set of equations
% sdot=-k1es+kminus1c
dxdt(1)= -K1*x(2)*x(1) +KMINUS1*x(3);
% edot=-k1es+(kminus1+k2)c
dxdt(2)= -K1*x(2)*x(1)+(KMINUS1+K2)*x(3);
% cdot=k1es-(kminus1+k2)c
dxdt(3)= K1*x(2)*x(1) -(KMINUS1+K2)*x(3);
%pdot=k2c
dxdt(4)= K2*x(3);

% QSSA approximations
dxdt(5)= -K2*e0*x(5)/(x(5) + Km);
dxdt(6)= K2*e0*x(5)/(x(5) + Km);

% tQSSA approximations
dxdt(7)= -K2*e0*x(7)/(e0 + Km + x(7));
dxdt(8)= K2*e0*x(7)/(e0 + Km + x(7));
```

```
dxdt=dxdt';
return
```

M-M kinetics with QSSA only approx comparison (Ex. 6.6 extra)

```
% FULL MICHAELIS-MENTEN ENZYME + QSSA DYNAMICS
% Reactions --- R1: S + E--k1 --> C
% R2: C -kminus1-->E + S, R3: ES--k2--> E + P
% pdotQSSA = -sdotQSSA = sVmax/(s+Km)
% Parameters (60 converts k's to per min)
% KMINUS1 = 2/sec or 200/sec (chg & run again)
global K1 KMINUS1 K2 Vmax Km;
K1 = 60*10; KMINUS1 = 60*2;K2 = 60*0.02;
e0 = 1; s0 = 1; Vmax = K2*e0;
Km = (KMINUS1 + K2)/K1;
x0 = [s0; e0; 0; 0; 0];  % ICs
%solve the ODEs
[t,x]=ode15s(@MM_QSSA_final, [0 10], x0);
%Lay out the plots clearly & plot!
fig = figure('PaperPosition',[2 2 8 6.3]);
ax = axes('FontName','cmr','FontSize',12,...
  'Position',[0.1264 0.15 0.7879 0.7643],...
  'Parent',fig);h = plot(t,x);
set(h,{'LineWidth'},{2;2;2;2;1},{'Linestyle'},...
    {'--';':';'-';'--';'*'},{'Color'},...
    {'b';'k';'r';'g';'k'});
title('MM ENZYME-SUBSTRATE DYNAMICS: FULL+QSSA');
legend(ax,'kminus1 = 2/sec');xlabel(ax,'MINUTES');
ylabel(ax,'Concentration (nM)');
legend(ax,{'S','E','ES','P','P-QSSA'},...
'FontName','Times','FontSize',12);legend(ax,'boxoff')
```

CHAPTER 7

Example 7.6 – T-Cell Proliferation – Matlab code for generating Fig. 7.5:

```
% FULL T-CELL PROLIFERATION+QSSA+tQSSA DYNAMICS
% Reactions --- R1: S + E--k1 --> C
% R2: C -kminus1-->E + S, R3: C--k2--> E + 2S
% ds_QSSA/dt = sVmax/(Km+s)= k2*c
% sbar = s + c
% dsbar_tQSSA/dt = sbar*Vmax/(e0+Km+sbar)
% Parameters
global K1 KMINUS1 K2 Vmax Km e0;
K1 = 10; KMINUS1 = 1;K2 = 0.1;
e0 = 100; s0 = 0.1; sbar0 = 0.1; Vmax = K2*e0;
Km = (KMINUS1 + K2)/K1;
x0 = [s0; 0; sbar0; s0; sbar0];  % ICs
%solve the ODEs
[t,x]=ode15s(@TCELL_QSSAMarch2_2012, [0 1000], x0);
%Lay out the plots clearly & plot!
```

```
fig = figure('PaperPosition',[2 2 8 6.3]);
ax = axes('FontName','cmr','FontSize',12,'Position',...
[0.1264 0.15 0.7879 0.7643],'Parent',fig);
h = semilogy(t,x);
set(h,{'LineWidth'},{2;2;3;2;1},{'Linestyle'},...
    {'--';'-';':';'--';'*'},{'Color'},...
    {'b';'y';'r';'g';'k'});
title('T-CELL PROLIFERATION:FULL+QSSA+tQSSA');
xlabel(ax,'TIME');
ylabel(ax,'LOG CONCENTRATION (nM)');
legend('T','C','Tbar=T+C','~T','~Tbar',...
'FontName','Times','FontSize',12,'Location','Best');
legend(ax,'boxoff')

function [dxdt] = TCELL_QSSAMarch2_2012(~,x)
%rate eqns for T-cell replication solver, full + qssa and tqssa
global K1 KMINUS1 K2 Km Vmax e0;
dxdt(1)= -K1*(e0-x(2))*x(1)+(KMINUS1+2*K2)*x(2); %sdot=-k1(e0-
c)s+(kminus1+2K2)c
dxdt(2)= K1*((e0-x(2))*x(1)-Km*x(2)); % x2 is complex C, cdot=k1((e0-
c)s-Kmc)
dxdt(3)= (KMINUS1+2*K2-K1*Km)*x(2); % ODE for sbar = s +c
dxdt(4)=(Vmax * x(4))/(x(4)+ Km); %ds/dt= approx sVmax/(s+Km) qssa
soln
dxdt(5)= Vmax * x(3)/(e0 + Km + x(3)); % approx sbar tqssa soln
dxdt=dxdt';
return
```

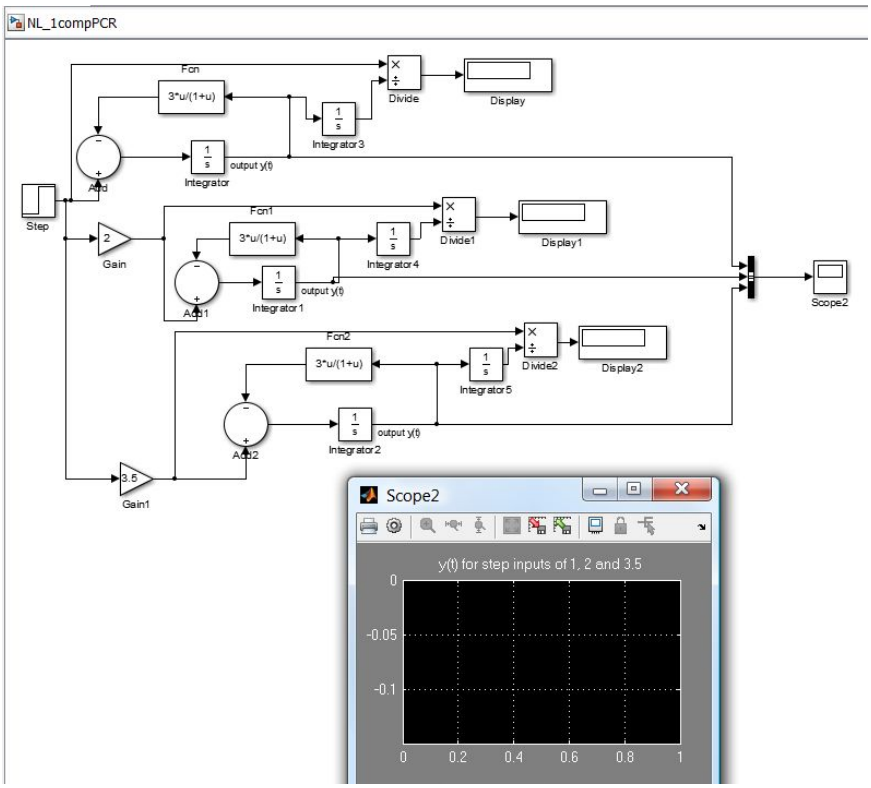
***Example 7.10 – SimBiology* Example**. Code is in Table 7.1 and Figs. 7.7 and accompanying text on page 328.

***Example 7.11 – Copasi* Example.** Code is in Figs. 7.9 and accompanying text on page 328.

***Example 7.15 – Matlab* code for Gillespie algorithm implementation of a single gene model** is given in Fig. 7.12 on page 339.

CHAPTER 8

All needed coded is given in the chapter proper. The following diagram does all AUC and PCR computations for the NL-1-compartment model (pages 377-380).



CHAPTER 9

Code for *Example 9.3*, page 412, and Fig. 9.5:

```
function []=nullcline_bistable()
%
close all; clear all;
%global n K;

%% plot nullclines

x1 = [0:0.05:1];
y2 = [0:0.05:1];
n = 1; K = 0.5;

y1 = x1.^n./(x1.^n + K.^n);

x2 = y2.^n./(K.^n + y2.^n); % redefined y as y2 here so I could plot
them both versus X.

% nodex = [0 0.5 0.8]; for n = 3
% nodey = [0 0.5 0.8]; for n = 3
nodex = [0 0.5]; % for n = 1
nodey = [0 0.5]; % for n = 1
```

```
plot(x1,y1,y2,x1)

h = plot(x1,y1,x2,y2);
set(h,{'LineWidth'},{2.5;2.5}, {'Color'},{'r';'b'});
%set(h,{'LineWidth'},{2.5;2.5}, {'Color'},{'r';'b'}, TextFont =
[16]);
legend({'dx2/dt = 0','dx1/dt = 0'},'Location','Best');

title('NULLCLINES FOR A BISTABLE SYSTEM WITH n = 1');

xlabel('x1'); ylabel('x2');
hold on % retains the current figure to allow additional plotting
plot(nodex,nodey,'LineStyle','.','MarkerSize',25,'Color','k'); %
Highlight fixed points
% plot(nodex,nodey,'LineStyle','o','Color','k'); % Highlight fixed
points

% Find the tangent line
arrowLength = 0.15;
x0 = 0.6;
y0 = x0/(x0+K);
% dy(x0)/dx
slope = (1/(x0+K))-x0/((x0+K)^2);
x1 = x0 +arrowLength;
y1 = slope*(x1-x0)+y0 ;
arrowstart = [x0 y0]; arrowhead = [x1 y1];
%arrow(arrowstart,arrowhead);
hold off
end
```

Code for Selkov model, Fig. 9.9 page 417:

```
close all; clear all; clc;

% -- Initial conditions --
 x0=1; y0=0.25; V0=[x0 y0]; a=0.1; b=0.88;

 % -- Selkov Equations --
 Selkov=@(t,v) [a*v(2) - v(1) + v(1)^2*v(2);...
                b - a*v(2) - v(1)^2*v(2)];

% -- Integration --
tspan=[0 150];
[t v] = ode45(Selkov,tspan,V0);

% -- Nullclines --
xn = [0:0.01:2];
yn1 = xn./(a+xn.^2);
yn2 = b./(a+xn.^2);
```

```
% -- Fixed point --
fp = b/(a+b^2);

% -- Vector Field --
xmin=0; xmax=2; ymin=0; ymax=2.5; step=0.05;
xgrid = linspace(xmin,xmax,1/step);
ygrid = linspace(ymin,ymax,1/step);
[X Y]=meshgrid(xgrid,ygrid);
DX = zeros(size(X));
DY = zeros(size(Y));
for i=1:numel(X)
    grad = Selkov(0,[X(i);Y(i)]);
    DX(i) = grad(1);
    DY(i) = grad(2);
end

% -- Normalize Field Vectors --
lengths = sqrt((DX.*DX) + (DY.*DY));
nDX = DX./lengths;
nDY = DY./lengths;

% -- Plotting --
%%
% Plot Phase Portrait alone
figure;
quiver(X,Y,nDX,nDY,.4,'Color','k','LineWidth',1.4); % Vector Field
hold on
plot(v(:,1),v(:,2),'LineWidth',2.5); % Phase Portrait
plot(xn,yn1,'Color','r','LineStyle','--','LineWidth',2); % Nullcline
plot(xn,yn2,'Color','g','LineStyle','--','LineWidth',2); % Nullcline
plot(b,fp,'LineStyle','o','Color','k','MarkerSize',8,'LineWidth',2); %
Fixed Point
hold off
legend({'VECTOR FIELD','PHASE
PORTRAIT','NULLCLINE1','NULLCLINE2','EQUILIBRIUM PT'})
axis([xmin xmax ymin ymax]);
xlabel('x'); ylabel('y');
title('SELKOV PHASE PLANE PLOTS');

%%
% Plot Phase Portrait with Trajectory
figure; subplot(1,2,1); h = plot(t,v(:,1),t,v(:,2));
set(h,{'LineWidth'},{2;2}, {'Color'},{'r';'b'});
xlim(tspan);
% figure; subplot(1,2,1); plot(t,x(:,1),t,x(:,2));
xlabel('TIME'); ylabel('x1, x2'); legend('x1','x2');

%legend ('b = ','b')

title('TIME RESPONSE SOLUTIONS');
```

```
subplot(1,2,2);
quiver(X,Y,nDX,nDY,.3,'Color','k','LineWidth',1.2); % Vector Field
hold on
plot(v(:,1),v(:,2),'LineWidth',2.5); % Phase Portrait
plot(xn,yn1,'Color','r','LineStyle','--','LineWidth',2); % Nullcline
plot(xn,yn2,'Color','g','LineStyle','--','LineWidth',2); % Nullcline
plot(b,fp,'Marker','o','Color','k','MarkerSize',8,'LineWidth',2); % Fixed
Point
hold off

legend({'VECTOR FIELD','PHASE
PORTRAIT','NULLCLINE1','NULLCLINE2','EQUILIB PT'})

axis([xmin xmax ymin ymax]);
xlabel('x1'); ylabel('x2');
title('SELKOV PHASE PLANE PLOTS');

%% Plotting:
% figure; subplot(1,2,1); plot(t,x(:,1),t,x(:,2));
%%%%%subplot(1,2,2); h = plot(x(:,1),x(:,2));
%set(h,{'LineWidth'},{2});
%xlabel('x1'); ylabel('x2');
%title('BRUSSELATOR PHASE PLANE');
```

Code for Brusselator model, Fig. 9.10, pages 419-421:

```
function[]= brusselator()
% 2-variable ODE model (Brussellator)
% adapted from brux.m by Didier Gonze, 2008
close all; clear all; clc; global a b;

%% Initial conditions:
 x0=0; y0=0; V0=[x0 y0]; a=1; b=3;

%% Integration:
tspan=[0 30];
[t x] = ode45(@brux,tspan,V0);

%% Plotting:
% figure; subplot(1,2,1); plot(t,x(:,1),t,x(:,2));
figure; subplot(1,2,1); h = plot(t,x(:,1),t,x(:,2));
set(h,{'LineWidth'},{2;2}, {'Color'},{'r';'b'});
xlabel('TIME'); ylabel('X, Y'); legend('x1','x2')
title('TIME RESPONSE SOLUTIONS');

subplot(1,2,2); h = plot(x(:,1),x(:,2));
set(h,{'LineWidth'},{2});
xlabel('x1'); ylabel('x2');
```

```
title('BRUSSELATOR PHASE PLANE');

%%~~~~~~~~~~~~~~~~~~~~~~~~~~~ODES~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function dv = brux(~,v)
global a b;

x=v(1); y=v(2);

dv(1) = a-(b+1)*x+x^2*y;     % dx/dt
dv(2) =  b*x-x^2*y;           % dy/dt
dv=dv';
return
```

Code for 1st-order Logistic eqn, Fig. 9.11, pages 422-423:

```
%logistic.m - this MATLAB file solves the
%discrete logistic equation x(i+1)=r*x(i)*(1-x(i))
%and the users is prompted to read in the value
% of r to use as well as the initial value
% x0 which must be in (0,1)
% and the time interval over which to run the
% simulation.
!c:
s=1;
while s>0;
r=input('input growth rate r:     ')
x0=input('input initial population x0:      ')
n=input('end of time interval b:      ')
x=zeros(n+1,1);
t=zeros(n+1,1);
x(1)=x0;
for i=1:n
t(i)=i-1;
x(i+1)=r*x(i)*(1-x(i));
end
t(n+1)=n;
plot(t,x,t,x,'o'),pause
s=input('Do you want to stop - if so enter 0')
end
```

Code for Lorenz model, pages 426-427:

```
% lorenzstart.m
close all;
clear all;
clc;
ti=0;
tf=150;
tspan=[ti, tf];
```

```
x0=[0.3 0.3 0.3]';
[t,x]= ode23(@lorenz,tspan,x0);
figure
subplot(3,1,1), plot(t,x(:,1),'r'),grid on;
title('LORENZ ATTRACTOR'),ylabel('x1(t)');
subplot(3,1,2), plot(t,x(:,2),'b'),grid on;
ylabel('x2(t)');
subplot(3,1,3), plot(t,x(:,3),'g'),grid on;
ylabel('x3(t)');xlabel('TIME')
figure
h = plot3(x(:,1),x(:,2),x(:,3)),grid on;
set(h,{'LineWidth'},{1});
xlabel('x1');ylabel('x2');zlabel('x3')
title('LORENZ ATTRACTOR 3-D');
figure
plot(x(:,1),x(:,2)),grid on;
xlabel('x1');ylabel('x2');
title('LORENZ ATTRACTOR-2-D');


function xdot=lorenz(t,x)
% a=11; b=25; c=8./3;
a=11; b=25; c=8/3;
xdot=[-a*(x(1)-x(2));
b*x(1)-x(2)-x(1)*x(3);
-c*x(3)+x(1)*x(2)];
end
```

CHAPTER 10 – no code needed

CHAPTER 11

Examples 11.3-5* and *7 GUI code in Figs. 11.2-11.4 and 6
Examples 11.8-12 Matlab code given if Figs. 11.7-11

CHAPTER 12

Example 12.2 – SAAMII program for this example is the 1-compartment (top) segment of the program in Fig. 13.8. The data is from Table 13.4, data column z1bc.

CHAPTER 13

SAAMII programs and Matlab code for this chapter are in the figures.

TABLE 13.4 cited on pate 588:

Table 13.4 **- Rabbit Study Data for** ***Example 13.13***

time	z1bc	z1def	z1ghi
0.033	1733.0	n	n
0.0331	1798.7	n	n
0.083	1115.5	n	n
0.0831	1480.5	n	n
0.167	1241.8	n	n
0.1671	1381.6	n	n
0.25	1115.3	n	n
0.2501	1177.6	n	n
0.333	1028.9	n	n
0.3331	971.5	n	n
0.5	954.4	n	n
0.501	979.5	n	n
0.75	705	n	n
0.7501	733.8	n	n
1	800.2	2.5	n
1.001	865.5	4.0	n
1.5	718.4	n	n
1.5001	852.5	n	n
2	746.8	20.6	6.9
2.001	951	47.4	n
2.002	n	58.5	n
3	406	25.1	4.2
3.001	661.8	59.1	8.8
3.002	n	64.2	n
4	455	32.2	10
4.001	583.1	59.8	5.1
4.002	n	60.2	7.2
5	352	45	29.1

5.001	512	53.	76.3
5.002	n	57.6	9.2
6	297	41.4	10.6
6.001	348.1	47.7	15.5
6.002	n	84.1	39.1
7	291.9	31.2	20.4
7.001	329	44.8	35.2
7.002	n	91.1	9.6
8	236.4	111	15.4
8.001	242	39.1	22.6
8.002	n	51	34.2
10	193	46.3	26.4
10.001	211.9	63	32.7
10.002	n	70.2	65.6
12	176	110.9	34.7
12.001	202.6	50.3	36.1
12.002	n	60	76.2
24	53	45.1	37.3
24.001	57.4	50.7	52
24.002	n	53.3	53.1
26	57	36.8	34.1
26.001	60.3	49.8 5	3.4
26.002	n	63.5	54.9
28	50	28.5	34.2
28.001	54.1	40.2	46.6
28.002	n	63.9	52.1
30	47.1	34	27
30.001	64	42.2	44.4
30.002	n	54.1	47
32	45.5	31	22.8
32.001	59.7	34.4	38.3
32.002	n	46.9	42.8
48	24.6	22.9	12.4
48.001	37.5	29.1	19.6
48.002	n	30.7	31.3
72	n	12.1	10.8

72.001	n	12.5	18.6
72.002	n	23.8	5.5
96	n	15	3
96.001	n	6.9	3.4
96.002	n	6.9	8.5
120	n	10.1	1.4
120.001	n	2.3	3.7
120.002	n	n	6.6
144	n	1.6	0.9
144.001	n	1.8	2.3
144.002	n	4.3	3.3
168	n	1.1	1.7
192	n	2	1
END			

Exercise E13.5 (b) – Web app *W3MAMCAT* is currently being updated, so that it will run in current browsers (security issues). Completion is expected by May 2014. Older (e.g. IE6) internet browsers will run *W3MAMCAT*.

CHAPTER 14

Thyroid hormone regulation model – pages 602-610. SBML code for this simulation model is downloadable from: http://biocyb1.cs.ucla.edu/thyrosim/sbml/thyrosim.xml.
The web app for running this model is available at: biocyb1.cs.ucla.edu/thyrosim.

VanSlyke-Cullen Model – *Matlab* code (adapted from flach et al. 2006) for generating the open-loop and closed-loop models of Figs. 14.10-12, pages 614-616 is:

```
function []=VanSlykeCullen_Linearfeedbackflow_full_qssa()
close all; clear all;

global K1 K2 V2 e0;
V2=1; K1 = 1; K2 = 1; e0 = 2;
IC2=[4,0,e0*4/(1+4); 3,0,e0*3/(1+3); 2,0,e0*2/(1+2)]'; %ICs for QSSA
(chosen to start on null surface)

%% full model numerical solution:
figure
subplot(1,2,1);
%plot null surface
ss=0:.4:6; pp=-2:.4:3; [S,P]=meshgrid(ss,pp);
C=e0.*S./(K2/K1+S);   colormap(gray);surfl(S,P,C); alpha(0.4);hold on;
```

```
title('full numerical solution of the Van Slyke-Cullen reaction with
linear flow');
legend('null surface of C','Location','Best');
xlabel('substrate'); ylabel('product'); zlabel('complex');
%plot trajectories
for ic=1:3
    x0=IC2(:,ic);                                % grab column with ICs in
order of s, c, p
    plot3(x0(1),x0(2),x0(3),'r*');               % plut IC as a star on
plot and hold on;
    t=[0 100];                                   % from 0 to 1 sec
    [~,x]=ode15s(@flach_feedback_full, t, x0);   % solve the ODEs
    plot3(x(:,1),x(:,2),x(:,3))                  % plot phase plot
end
%% qssa numerical solution:
subplot(1,2,2);
%plot null surface
ss=0:.4:5; pp=-2:.4:3; [S,P]=meshgrid(ss,pp);
C=e0.*S./(K2/K1+S);  colormap(gray);surfl(S,P,C); alpha(0.4);hold on;
title('QSSA solution for Van Slyke-Cullen reaction with linear flow');
legend('null surface of C','Location','Best');
xlabel('substrate'); ylabel('product'); zlabel('complex');
%plot trajectories
for ic=1:3
    x0=IC2(:,ic);                                % grab column with ICs in
order of s, c, p
    plot3(x0(1),x0(2),x0(3),'r*');               % plut IC as a star on
plot and hold on;
    t=[0 100];                                   % from 0 to 1 sec
    [~,x]=ode15s(@flach_linear_qssa, t, x0);     % solve the ODEs
    x(:,3)=e0*x(:,1)./(K2/K1+x(:,1));            % solve for c as a
function of s (qssa)
    plot3(x(:,1),x(:,2),x(:,3))                  % plot phase plot
end

%% ~~~~~~~~~~~~~~~~~~~~~~ODES~~~~~~~~~~~~~~~~~~~~~~
function [dxdt] = flach_feedback_full(~,x)
global K1 K2 V2 e0;
V1=2 - 3.1*x(2);                                          % V2=2-3.1p
dxdt(1)=V1 - K1*x(1)*(e0 - x(3));                         % linear model
sdot=V1(p)
dxdt(2)=K2*x(3) - V2;                                     % linear model
pdot
dxdt(3)=K1*x(1)*(e0 - x(3)) - K2*(x(3));                  % linear model
cdot
dxdt=dxdt';
return
%  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
function [dxdt] = flach_linear_qssa(~,x)
global K1 K2 V2 e0;
V1=2-3.1*x(2);                                  % V1 linear equation
```

```
dxdt(1)=V1 - e0.*K1*x(1)./(K2/K1+x(1));          % QSSA sdot=V1(p) -
K1se0/(k2/k1 + s))
dxdt(2)=(e0*x(1)./((1/K1)+x(1)))-V2;             % QSSA pdot=se0/(1/k1 + s)-
v2(p)
dxdt(3)=0;                                       % QSSA cdot=0;
dxdt=dxdt';
return
```

Exercise 14.5 – Plant metabolism modeling and analysis, ***Matlab*** **code (for various parts)**:

```
%clear
xdot = zeros(3,1);
tspan =[0:0.5:720];

%vector for initial conditions
xdot0 = [1;0;0];

%Solving the ODE
[t,x] = ode45('kin', tspan, xdot0, []);

A=x(:,1);
B=x(:,2);
C=x(:,3);

KM1=0.8; % uM
KM2=0.1;
KM3=0.004;
vmax1=0.0018; % uM/s
vmax2=0.0018;
vmax3=0.000012;

v1 = vmax1.*A./(A+KM1);
v2 = vmax2.*A./(A+KM2);
v3 = vmax3.*B./(B+KM3);

J2 = C.*(v2+v3);
J3 = B.*(v1-v3);

dJ2dv1 = 0;
dJ2dv2 = C.*A./(A+KM2);
dJ2dv3 = C.*B./(B+KM3);

dJ3dv1 = B.*A./(A+KM1);
dJ3dv2 = zeros(size(dJ3dv1));
dJ3dv3 = B.^2./(B+KM3);

clear A; clear B; clear C; clear v1; clear v2; clear v3; clear J2;
clear J3;
plot(t,dJ2dv1,'b',t,dJ2dv2,'g',t,dJ2dv3,'r',t,dJ3dv1,'--
b',t,dJ3dv2,'--g',t,dJ3dv3,'--r')
```

```
title('Dynamic simulation of changes in metabolite concentrations
over time when Km3=0.1')
xlabel('Time [sec]');
ylabel('Metabolite Concentration [uM]')
-----------
%The kin function :
function xdot = kin(~,x)

%[A]=x(1), [B]=x(2), [C]=x(3)
%Michalis-Menten Kinetic parameters
KM1=0.8; % uM
KM2=0.1;
KM3=0.004;
vmax1=0.0018; % uM/s
vmax2=0.0018;
vmax3=0.000012;

%species equations
xdot = [-vmax1*x(1)/(x(1)+KM1)-vmax2*x(1)/(x(1)+KM2);
vmax1*x(1)/(x(1)+KM1)- vmax3*x(2)/(x(2)+KM3);
vmax3*x(2)/(x(2)+KM3)+vmax2*x(1)/(x(1)+KM2)];
------------------
%The kin function :
function xdot = kin1(~,x)

%[A]=x(1), [B]=x(2), [C]=x(3)
%Michalis-Menten Kinetic parameters
KM1=0.8; % uM
KM2=0.1;
KM3=0.002;
vmax1=0.0018; % uM/s
vmax2=0.0018;
vmax3=0.000012;

%species equations
xdot = [-vmax1*x(1)/(x(1)+KM1)-vmax2*x(1)/(x(1)+KM2);
vmax1*x(1)/(x(1)+KM1)- vmax3*x(2)/(x(2)+KM3);
vmax3*x(2)/(x(2)+KM3)+vmax2*x(1)/(x(1)+KM2)];
----------------------
%clear
xdot = zeros(3,1);
tspan =[0:0.5:720];

%vector for initial conditions
xdot0 = [1;0;0];

%Solving the ODE
[t,x] = ode45('kin', tspan, xdot0, []);
[t,x1] = ode45('kin1', tspan, xdot0, []);

A=x(:,1); % A1=x1(:,1);
```

```
B=x(:,2); % B1=x1(:,2);
C=x(:,3); % C1=x1(:,3);
plot(t,A,'b',t,B,'g',t,C,'r',t,A1,'--b',t,B1,'--g',t,C1,'--r')
legend('A','B','C')
title('Dynamic simulation of changes in metabolite concentrations
over time when Km3=0.1')
xlabel('Time [sec]');
ylabel('Metabolite Concentration [uM]')
```

CHAPTER 15

***COMBOS* code** for structural identifiability of the p53 model beginning on page 661. The equations in *COMBOS* ASCIIMathML cut-and-paste format are:

dx1/dt = p1*x4-p3*x1-p4*((x1^2/(p5+x1))*(1+p6*u1/(p7+u1)))
dx2/dt = p8-p9*x2-p10*(x1*x2/(p11+x2)*(1+p12*u1/(p13+u1)))
dx3/dt = p14-p15*x3-p16*x1*x3*(1-p18*u1)/(p17+x3)
dx4/dt = (p20-p21*(1-p24)*(1-p25)/(p22^4+1))-p20*x4+p21*(x3^4)*(1+p23*u1)*(1-p24*x1)*(1-p25*x2)/(p22^4+x3^4)
y1 = x1
y2 = x2
y3 = x3
y4 = x4

Exercise E15.2 – Rice Wagner Model. See **website Solutions** for the *Simulink* GUI for this model.

CHAPTER 16

OSS Design software – page 697. Here's a weblink to the executable for ***oss1v6,*** compiled on an i7, win 7 64 bit OS:

https://www.wuala.com/long/Documents/?key=139uvlJctSHZ

Email joed@ucla.edu for a copy of the fortran source code.

CHAPTER 17

Example 17.3* –** The ***AMIGO2 code for this example is being updated. It is for simplifying and estimating parameters of the p53 model in this example (pages 714 – 717). A link to it will be published on this site in the near future.
