MATLAB Programs for Modeling and Simulation in Medicine and the Life Sciences and Population Mathematics

Frank Hoppensteadt[1]

February 20, 2008

Contents

Comments

The MATLAB programs in this document were developed over several years, and so they use many versions of MATLAB. All of these programs have been verified to run on MATLAB2007b.

Note that some programs use global variables to provide information to others, and some pass the information as parameter structures. The latter is preferred, since there are situations where global variables can cause confusion.

Most of the solutions to ordinary differential equations use straightforward ode solvers. Some use simple forward Euler quadrature formulas, some use Runge-Kutta based schemes (ode23 and ode45), and some use a stiff differential equation solver (ode15s). Control of the solvers is handled through the 'odeset' command, which specifies the absolute and relative tolerances and the maximum step size allowed. This last is especially helpful in plotting. For the most part, partial differential equations are solved using the method of lines where systems are discretized in spatial coordinates, but treated as being continuous in time.

These programs are designed to execute various calculations arising in our books in mathematical biology[1] They are intended for illustrative purposes only for understanding models that are always based on extensive simplifying assumptions. The user is warned not to regard the statements, equations or programs here as being literal or exact descriptions of biological reality. They are in the nature of metaphors, used as guides toward certain fundamental biological processes and functions. The author assumes no responsibility for the accuracy or efficiency of these programs or their use, or for errors in them.

[1]See F.C. Hoppensteadt, C.S. Peskin, Modeling and Simulation in Medicine and the Life Sciences, Springer-Verlag, New York, 1992, 2001 (2nd. ed.). F.C. Hoppensteadt, An Introduction to the Mathematics of Neurons, Cambridge University Press, Cambridge, 1986, 1997 (2nd. ed).

Chapter 1

Population Dynamics

1.1 Population Numbers

1.1.1 Swedish Population Growth Rate

```
%Swedish Population Data (1000s)
P=[2104 2352 2573 3123 3824]; T=(1:5);%
a=(dot(log(P),T)*5-sum(T)*sum(log(P)))/(5*dot(T,T)-sum(T)^2)%
r=exp(a)
```

1.1.1.1 Swedish data vs Malthus

```
%Initial data
P=[2104 2441 2831 3284 3810]; %
O=[2104 2352 2573 3123 3824 4572 5117 5876 6356 7480];%
Y=1780:20:1960;

%Predicted values computation
for K=5:9
    P(K+1)=(1.16)*P(K);
end;

%Error computation
Error=abs(100*(O-P)./P)


%Plotting
figure(1),clf               %
plot(Y,O,Y,P,'O');          %
title('Swedish Population Data Projections 1780-1960');%
xlabel('Year');             %
ylabel('*1e3-Population');  %
text(1818,3100,'P');        %
text(1833,2700,'O');
```

1.1.2 Logistic Ordinary Differential Equation

```
%verhulst.m Solves Verhulst (logistic) ODE using RK45
%This shows how to solve ODEs using MATLAB packages.
%ode45 (Runge-Kutta 45) and ode15s (stiff ODE solver) are often used.
p.tf=10;p.ht=0.01;                  %Time limits.
p.tt=0:p.ht:p.tf;p.nt=length(p.tt); %Time arrays
%odeset specifies accuracies required of the ode solver
options=odeset('abstol',1e-8,'reltol',1e-8,'maxstep',1e-1);
x0=1;p.r=0.1;p.K=10;                %Initial condition and parameters
tic                                 %Start the clock
[t x]=ode45(@verrhs,p.tt,x0,options,p);%Solve
toc                          %Stop the clock, print elapsed time
figure(1),clf                       %Print
plot(t,x),title(['Logistic Growth, r = ' num2str(p.r)])
```

1.1.2.1 @verrhs.m

```
function xdot=verrhs(t,x,p)
%Specifies ODE for logistic equation
xdot=-x;            % gets dimensions right
xdot=p.r*x*(p.K-x); % x'=rx(K-x)
%Note data are passed using the structure p.
%This is done instead of using global variables
%to keep better control of variable's values.
```

1.1.3 Cube Roots of Unity

```
%This shows how to use imaginary numbers in computations.
%Here i = sqrt(-1)
rho=exp(2*pi*i/3);
figure(1),clf,hold on%
line([2 2],[-2 2])%
plot([rho,rho^2,rho^3],'*')%
axis([-2 3 -2 2]),xlabel('\Re s'),ylabel('\Im s')%
title('Cube Roots of Unity'),hold off
```

1.1.4 Renewal Equation using Trapezoidal Rule

```
%renewal.m, trapezoidal and rectangular rules.
%This shows how to carry out quadratures in the computer.
p.ht=0.01;p.tf=200;p.tt=0:p.ht:p.tf;p.nt=length(p.tt);  %time scale data
p.alf=0.2725;p.bet=0.01;p.r=0.01;                       %model data
s1=(2*p.r)^(1/3)-p.alf                                  %intrinsic growth rate
x=zeros(p.nt,1);m=x; x(1,1)=1.0;m(1,1)=0.0;             %x-births, m-maternity
p.t50=floor(50/p.ht);m1=zeros(length(p.t50),1);         %plot m
for k=1:p.t50
    a=(k-1)*p.ht;
    m1(k,1)=a^2*exp(-p.alf*a);
end

figure(1),clf%
plot(p.ht*(1:p.t50),p.r*m1)%
title('Maternity Function')%
xlabel('Age'), ylabel('Maternity Function')%
drawnow%
rect=zeros(p.nt,1);m=rect';                             %rectangular quadrature
tic                                                     %start the clock
for j=2:p.nt                                            %solve renewal equation
    tma=j*p.ht-(1:(j-1))*p.ht;                          %t-a
    m=(tma.^2).*exp(-p.alf*tma);                        %m(t-a)
%     rect(j)=sum(x(1:(j-1),1).*m');                    %rectangular approx
%     y(j)=exp(-(j-1)*p.bet*p.ht)+p.r*rect(j)*p.ht;
    v=x(1:(j-1),1).*m';                                 %integrand
    y(j)=exp(-(j-1)*p.bet*p.ht)+p.r*trapz(v)*p.ht; %new birthrate
    x(j)=y(j);
end
toc                                                     %print elapsed time
figure(2),clf                                           %plot results
plot(p.tt,x,'r');title('Birth Rate vs Time')%
xlabel('Time, t'), ylabel('Birth Rate')
```

1.1.5 Cobwebbing

```
% cobweb.m. This shows how to draw iterates of a function.
n=20;                        % number of iterates
p=zeros(n,1); q=zeros(n,1); % population sizes
r=20;                        % intrinsic growth rate
p(1,1) = 0.1;                % initial condition
for k=2:n
    p(k,1)=r*p(k-1,1)*exp(-p(k-1,1));% reproduction
    q(k-1,1)=p(k,1);         % offset for plotting
end%
figure(1),clf,hold on%
x=0:0.1:20;%
f=r*x.*exp(-x);              %Ricker's Model
plot(x,f,x,x)                % plot f and 1-to-1 reference
plot(p(1:(n-1),1),q(1:(n-1),1),'*') % data points
for j=1:(n-2)                %Draw the cobweb
    line([p(j,1) q(j,1)],[q(j,1) q(j,1)])
    line([q(j,1) q(j,1)],[q(j,1) q(j+1,1)])
end%
hold off
xlabel(['r = ' num2str(r)]) % label plot
axis([0 10 0 10])%
```

1.1.6 Reproduction Iterate Density

```
%Ricker density
M=1000; N= 100; den=zeros(M,N); %M r-values, N-bins
for mr=1:M
    r=20*mr/M;                  %set r
    r1=r/exp(1);                %calculate scale
    x=1;                        %set initial value for x
    for n=1:1005                %iterate 1005 times for each r
        x1=r*x*exp(-x*r1);      %Ricker's function
        j=floor(x1*N)+1;        %index where x hits
        den(mr,j)=den(mr,j)+1;  %update density matrix bin j
        x=x1;                   %go back for the next x value
    end
end %
den1=den;                       %normalize density matrix%
for m=1:mr
    den1(m,:)=den(m,:)/max(den(m,:));
end %

[R X]=meshgrid((1:M)*(20/M),(1:N)/N);%graphing coordinates
figure(1),clf                   %create a figure and clear it
mesh(R,X,den1')                 %plot the normalized density
view(0,90)                      %Look down from above.
                                %Rotate figure for other views
shading interp                  %affects coloring
xlabel('r'),ylabel('x'),zlabel('Density') %label figure
title('Iteration of Rickers Model')
```

1.1.7 Cicada

```
% The Hoppensteadt-Keller Model of Cicada Synchronization
%
% cicada.m

%Hoppensteadt-Keller cicada model
clear x; clear t; clear P;
%KK=maximum carrying capacity k

%initialize
nmax=100; L=13; KK=10000; alpha=0.95; A=0.042; R=0.95; f=10.0;
t=(1:1:nmax); %time
x=zeros(1,nmax); P=zeros(1,nmax); for i=1:L
x(i)=100; %start with population 100
P(i)=0; %no predation for the first L generations
end;
t=(1:1:nmax); %time

%main loop
for n=L+1:nmax-1
Ksum=KK; %compute the current carrying capacity
for j=1:L-1 Ksum=Ksum-x(n-j)*(alpha^j); end
K(n)=max(Ksum,0); %carrying capacity can't be negative

%update the predator population
P(n+1)=R*P(n)+A*x(n-L)*alpha^L; J=max(f*(x(n-L)*alpha^L-P(n)),0);
x(n)=min(K(n),J); end

P=P(:,1:nmax); %decrease the size of P by one element for plotting
%plot results
figure(1),clf %
stairs(t,x);
hold on; %use to plot x and p on the same graph
stairs(t,P,'r'); %the 'r' makes the predator curve red
hold off; title('Number of nymphs and predators per year');

% Result for L=7, KK=10,000, f=10, alpha=R=0.95, A=.042, nmax=150:
```

```
%approach the balanced solution.
%
% Result using parameters as above, but change L to 13:
%approach the synchronized solution.
```

1.1.8 Chemostat

```
%chemostat.m
% b'= vbs/(s+k) - l b
% s' = l s0 - l s - vbs/(s+k) 1/y

p.S0=20;p.V=.01;p.K=1;p.Yield=0.1;p.f=.1;
p.tf=10;p.ht=0.01;p.tt=0:p.ht:p.tf; y0=ones(2,1);
y0(1,1)=5;y0(2,1)=1;y=y0;
options=odeset('AbsTol',1e-8,'RelTol',1e-8,'maxstep',0.1); %
tic%
[t,y] = ode45(@chemorhs,p.tt,y0,options,p); %
toc%
 figure(1),clf, hold on
    plot(t,y(:,1),'b',t,y(:,2),'r')
    axis([0,p.tf,0,100])
    xlabel('time'); ylabel('bact and substr');
```

1.1.8.1 @chemorhs

```
function zp=chemorhs(t,z,p)

zp=z;

zp(1)=p.V*z(1)*z(2)/(z(2)+p.K);
zp(2)=p.f*(p.S0-z(2))-p.V*z(1)*z(2)/(p.Yield*(z(2)+p.K));
```

1.1.9 SPQ

```
%SPQ

global V K Y

Y=1; V=1; K=1; tf=10; Fs=32; tt=0:(1/Fs):tf; x0=zeros(3,1);x=x0;
x0(1,1)=100; x0(2,1)=5;

[t x]=ode15s(@spqrhs,tt,x0);

figure(1),clf %
plot(x)


%%%%%%%%%%%
```

1.1.9.1 @spqrhs

```
function xp=spqrhs(t,x)

global V K Y

xp=x;%
xp(1)=-(V*x(1)*x(2)/(Y*(K+x(1))));
xp(2)=(V*x(1)*x(2)/(K+x(1))-x(2)/(1+x(1))+x(1)*x(3)/(1+x(1)));
xp(3)=x(2)/(1+x(1))-x(1)*x(3)/(1+x(1));
```

1.1.10 Bacterial Plate

```
%chia.m
global DD M
%data
M=21;tf=1000;Fs=2;tt=0:1/Fs:tf; D=0.01;dx=4/M;
%diffusion matrix - radially symmetric
A=zeros(M); A(1,1)=-2;A(1,2)=2; A(M,M-1)=2;A(M,M)=-2; %
for k=2:(M-1)
   A(k,k)=-2;A(k,k+1)=1;A(k,k-1)=1;
end %
A=(D/dx^2)*A;

B=zeros(M); %
for k=2:(M-1)
   B(k,k+1)=1/k;
   B(k,k)=-1/k;
end %
B=(D/dx^2)*B;

DD=zeros(4*M); DD(1:M,1:M)=zeros(M); DD(M+1:2*M,M+1:2*M)=zeros(M);
DD(2*M+1:3*M,2*M+1:3*M)=0.01*(A+B); DD(3*M+1:4*M,3*M+1:4*M)=(A+B);

%initial data
U=zeros(4*M,1);U0=U; for k=1:M
   U0(k,1)=1;                                          %bacteria
   U0(M+k,1)=0;                                        %quiescent
   U0(2*M+k,1)=200*exp(-k*dx*5);                       %nutrient
   U0(3*M+k,1)=10.0;                                   %buffer
end

options=odeset('AbsTol',1e-8,'RelTol',1e-6); %
tic %
[t U]=ode15s(@chirhs,tt,U0,options); %
toc

figure(1), clf %
subplot(4,1,1), mesh(U(:,1:M)),zlabel('Bacteria')
```

```
subplot(4,1,2), mesh(U(:,M+1:2*M)),zlabel('Cyst') %
subplot(4,1,3), mesh(U(:,2*M+1:3*M)),zlabel('Nutrient')%
subplot(4,1,4), mesh(U(:,3*M+1:4*M)),zlabel('Buffer')
```

1.1.10.1 @chirhs

```
function up=chirhs(t,U) %
global DD M

up=U; V=zeros(4*M,1); %
for k=1:M
   ab=U(k+2*M)*U(k+3*M);
   f=10*ab/(5+ab);
   a12=max(0,8*(1-ab));
   a21=max(tanh(ab-2),0);
   V(k)=f*U(k)-a12*U(k)+a21*U(k+M);
   V(k+M)=a12*U(k)-a21*U(k+M);
   V(k+2*M)=-f*U(k);
   V(k+3*M)=-f*U(k);
end

up=DD*U+V;
```

1.1.11 Graph Transition Matrix

```
G=zeros(100); %
for j=1:100
   xquad=3.98*((j-1)/100)*(1-((j-1)/100));
   k=floor(xquad*100)+1;
   G(j,k)=1;
end %
figure(1),clf %
subplot(2,1,1),spy(G),xlabel('G for r = 3.98')%
subplot(2,1,2),spy(G^10),xlabel('G^{10} for r = 3.98')%
```

1.2 Epidemics

1.2.1 Kermack McKendrick model

```
%Kermack-McKendrick

global r sigma

r=.5; sigma= 1;
N=10; %population size
tf=10; Fs=32; tt=0:(1/Fs):tf;

x0=zeros(3,1);x=x0;

x0(2,1)= 3; %Initial infective population size
x0(1,1)=N-x0(2,1); %Initial susceptible population size
[t x]=ode15s(@kerrhs,tt,x0)

figure(1), plot(t,x(:,1),t,x(:,2),t,x(:,3))

figure(2), plot(x(:,1),x(:,2)) %
axis([0 N 0 N])
```

1.2.1.1 @kerrhs

```
function xp=kerrhs(t,x) %
global r sigma %
xp=x; %
xp(1)=-r*x(1)*x(2);
xp(2)=r*x(1)*x(2)-sigma*x(2); %
xp(3)=sigma*x(2);
```

1.2.2 Schistosomiasis

```
%schisto
N=100; g=zeros(N,2);h=g;f=zeros(N,1); %
figure(1),clf %
for j=1:N
   c=[1 j/10-.5 j/10];
   g(j,:)=roots(c)';
end %
for k=1:N
   if(imag(g(k,1)))
      h(k,1)=0;
   else
      h(k,1)=-g(k,1);
   end
   if(imag(g(k,2)))
      h(k,2)=0;
   else
      h(k,2)=-g(k,2);
   end
end %
del=(1:N)/10; %
figure(2),clf %
plot(del,h(:,1),'.',del,h(:,2),'.',del,f(:),'.')
ylabel('Helminths'),xlabel('\delta')
```

1.2.3 Final Size

```
%final_size.m
Io=1;So=10;%
r=0.1;lambda=0.9; nkappa=100;%
kappa=zeros(2*nkappa,1);
final=zeros(2*nkappa,3); %
figure(1),clf, hold on %
for m=1:3
    So=2*(10^(m-1));
    for j=1:nkappa
        kappa(j)=1*(j-1)/nkappa;%kappa less than 1
        F0=1;F=F0;
        for mm=1:10 %Newton Iteration
            G=exp(-kappa(j)*(1+(Io/So)-F));
            F1=F-(F-G)/(1-kappa(j)*G);
            if(abs(F1-F)<0.001)break;end%test for convergence
            F=F1;
        end
        final(j,m)=F1;
    end
    for j=(nkappa+1):(2*nkappa)%kappa beyond 1
        kappa(j)=1*(j)/nkappa;
        F0=.1;F=F0;
        for mm=1:10
            G=exp(-kappa(j)*(1+(Io/So)-F));
            F1=F-(F-G)/(1-kappa(j)*G);
            if(abs(F1-F)<0.001)break;end
            F=F1;
        end
        final(j,m)=F1;
    end

    plot(kappa,final(:,m),'b.',kappa,ones(length(kappa),1))
    axis([0 kappa(length(kappa)) 0 1.5])
    %
    xlabel('\kappa'),ylabel('F')
end
```

1.2.4 Reed Frost

```
%reed frost

N=5; %Family size
Nt=10; %final time
p=0.5; %contact probability
S=zeros(Nt,1); I=S;
I(1,1)=1; %Initial number of infectives
S(1,1)=N-I(1,1);

for nt=2:Nt
    q1=(1-p)^(I(nt-1,1));
    r=rand;
    q=0;
    for k=0:S(nt-1,1)
        q=q+nchoosek(S(nt-1,1),k)*q1^k*(1-q1)^(S(nt-1,1)-k);
        if(q>r) break; end
    end
    S(nt,1)=k;
    I(nt,1)=S(nt-1,1)-k;
end

[S,I]
```

1.2.5 Ludwig's Simulation

```
%ludwig.m  Don Ludwig's simulation
Ni=100;                          %number of populations
N=40;                           %Population size
Nt=2*N;                         %final time
Nprob=100;                      %number of contact probs per simulation
pr=linspace(0.0, 0.1, Nprob);   %effective contact probabilities

den=zeros(N,Nprob+1); %
tic %
for n=1:Ni
    for kk=1:(Nprob)
        p=pr(kk);
        S=zeros(Nt,1); I=S;
        I(1,1)=1;               %Initial number of infectives
        S(1,1)=N-I(1,1);
        for nt=2:Nt
            q1=(1-p)^(I(nt-1,1));   %prob of avoidance
            r=rand;                 %binomial threshold
            q=0;                    %binomial distribution
            for k=0:S(nt-1,1)
                q=q+nchoosek(S(nt-1,1),k)*q1^k*(1-q1)^(S(nt-1,1)-k);
                if(q>r) break; end
            end
            S(nt,1)=k;
            I(nt,1)=S(nt-1,1)-k;
        end
        den(k+1,kk+1)=den(k+1,kk+1)+1;
    end
end %
toc %
figure(1),clf %
[FAMILY PROB]=meshgrid(0:(N-1),[0.0 pr]);
surf(FAMILY, PROB,den'/Ni) %
caxis(1*[0 0.4]); colormap(hot) %
shading interp%
xlabel('Survivors'), ylabel('Contact prob'), %
```

```
zlabel('Proportion of iters') %
title('Don Ludwig''s Iteration')%
view(0,90) %
colorbar

figure(2),clf                    %Animate survivor distribution as function of p
for kk=1:Nprob
    clf
    bar(den(:,kk))
    title(['Contact probability ' num2str(pr(kk))])
    drawnow
    pause(.1)
end %
figure(3),clf %
bar(den(:,50)/Ni) %
title(['Contact probability ' num2str(pr(50))])
```

1.3 Genetics

1.3.1 Plasmid

```
%Plasmid Markov Chain

%Hypergeometric Markov Chain

global num den1 den2
N=5; %copy number
Nt=100; %number of time steps
g=zeros(Nt,1); %the number of a genes
g(1,1)=1; denom=plabin(2*N,N); N2=2*N; for nt=2:Nt
    i=g(nt-1,1);
    r=rand;
    q=0;
    for j=0:N
        inc=plabin(2*i,j)*plabin(2*(N-i),N-j)/denom;
        q=q+inc;
        if(q >= r) break; end
    end
    g(nt)=j;
end %
%
figure(1),clf %
plot(g) %

function b=plabin(m,n) %
global num den1 den2 num=m; den1=n; den2=m-n;
if(m<n)
    b=0;
elseif (m<(m-n))
    b=0
else
    b=factorial(num)/(factorial(den1)*factorial(den2));
end
```

1.3.2 Difinetti Diagram

```
% triangular coordinates

%draw triangular coordinates
L=2/sqrt(3); a=zeros(4,1);b=a;c=a;
a(1,1)=0;a(2,1)=L;a(3,1)=L/2;a(4,1)=0;
b(1,1)=0;b(2,1)=0;b(3,1)=1;b(4,1)=0; c=b; %
figure(1) %
plot3(a,b,c,'-')
figure(2) %
plot(a,b,'-') %
axis([-.1 1.2 -.1 1.15]) %
hold on


%Draw HW curve
Npts=10;Nfreqs=25; %
x=zeros(Npts,1);y=x;z=x; %
for j=1:Nfreqs
   p=j/(Nfreqs+1);q=1-p;
   a(j)=p^2;
   b(j)=2*p*q;
   c(j)=q^2;
   x(j)=L*c(j)+L*b(j)/2;
   y(j)=b(j);z(j)=p;
end %
figure(1) %
plot3(x,y,z) %
figure(2) %
plot(x,y,'-') %
drawnow
```

1.3.3 Noisy Gene Dynamics

```
function gene
%gene
global r s t eps rbar sbar tbar

sigamp=3;rhoamp=2;tauamp=1.5; eps=0.1;nf=1000;
rho=rhoamp*rand(nf,1); r=ones(nf,1)+eps*rho; rbar=rhoamp*(1+eps/2);
sig=sigamp*rand(nf,1); s=ones(nf,1)+eps*sig; sbar=sigamp*(1+eps/2);
tau=tauamp*rand(nf,1);t=ones(nf,1)+eps*tau;tbar=tauamp*(1+eps/2);

g=zeros(nf,1);g(1,1)=0.2; %
for n=2:nf
   g(n)=genf(g(n-1),n);
end

tf=eps*nf; tt=0:.01:tf; x0=g(1,1);x=x0; %
[t x]=ode15s(@genrhs,tt,x0); figure(1),clf%
plot((1:nf)*eps,g, tt,x,'r')

function xp=genrhs(t1,x) %
global r s t eps rbar sbar tbar %
xp=x;
xp=(rbar.*x.^2+sbar.*x.*(1-x))./...%
  (rbar.*x.^2+2*sbar.*x.*(1-x)+tbar.*(1-x).^2)-x;

function y=genf(x,m) %
global r s t eps rbar sbar tbar%
y=(r(m)*x^2+s(m)*(1-x)*x)/(r(m)*x^2+2*s(m)*x*(1-x)+t(m)*(1-x)^2);
```

1.4 Biogeography

1.4.1 Reaction Diffusion Equations (Continuous Time)

```
% 2d Reaction Diffusion Solver
p.tf=100; p.ht=0.1;p.tt=0:p.ht:p.tf; %
p.xf=2*pi;p.hx=0.4;p.xx=0:p.hx:p.xf;
options=odeset('reltol',1e-8,'abstol',1e-8,'maxstep',1e-1);
p.nt=length(p.tt);p.nx=length(p.xx);
p.alpha=0;p.mu=0;p.beta=1.0;p.w=1.0*ones(p.nx);
p.D=-2*eye(p.nx)+diag(ones(p.nx-1,1),1)+diag(ones(p.nx-1,1),-1);
%Boundary Conditions
%p.D(1,1)=0.0;p.D(1,2)=0.0;p.D(p.nx,p.nx)=0.0;p.D(p.nx,p.nx-1)=0.0;%dirich
%p.D(1,2)=2.0;p.D(p.nx,p.nx-1)=2.0;                  %reflecting
p.D(1,p.nx)=1;p.D(p.nx,1)=1;                         %periodic
p.D=.001*p.D/p.hx^2;                                 %diffusivity
u0=(sin((p.xx'*p.xx)/(2*pi))); v0=zeros(p.nx);       %data on square
u1=reshape(u0,p.nx^2,1);v1=reshape(v0,p.nx^2,1);     %vectorize
z0=[u1;v1];

tic%
[t z]=ode15s(@rdrhs,p.tt,z0,options,p);%solve
toc%

theta=reshape(z(length(t),1:p.nx^2),p.nx,p.nx);      %back to data on square
voltage=z(:,(p.nx^2+1):(2*p.nx^2));
% Set graphics parameters.
fig = figure; set(fig,'color','w') %
p.side=p.nx; x = ((-p.side+1):(p.side))'/p.side;
L=ones(1*length(x)); h = surf(x,x,L); %
[a,e] = view; view(a,90);view(90,90); %
axis([-1 1 -1 1 -2 2]);
caxis(26.9*[-1.5 1]);
colormap(hot); shading flat; axis off%
n=0; while ishandle(fig)
    n=n+1;
    u=reshape(voltage(n,:),p.nx,p.nx);               %plot voltage
    uu=[u u; u u];            vv=uu;    %2x2 patches clarify beh at bndry
```

```
    set(h,'zdata',uu,'cdata',10*vv);
    drawnow
    if(n>=p.nt)break;end
end;
```

1.4.1.1 @rdrhs

```
function zp=rdrhs(t,z,p) %
zp=z; u=reshape(z(1:p.nx^2,1),p.nx,p.nx);
v=reshape(z((p.nx^2+1):(2*p.nx^2)),p.nx,p.nx);
up=v;%
vp=p.w-v-(p.beta*cos(p.mu*t)).*sin(u)...
    -p.alpha*v.*(v.^2-1)+p.D*u+u*p.D;
zp(1:p.nx^2)=reshape(up,p.nx^2,1);
zp((p.nx^2+1):(2*p.nx^2))=reshape(vp,p.nx^2,1);
```

1.4.2 Reaction Diffusion Equations (Discrete Time)

```
% 2d Reaction Diffusion Solver
p.tf=100; p.ht=0.1;p.tt=0:p.ht:p.tf; %
p.xf=2*pi;p.hx=0.4;p.xx=0:p.hx:p.xf;
options=odeset('reltol',1e-8,'abstol',1e-8,'maxstep',1e-1);
p.nt=length(p.tt);p.nx=length(p.xx);
p.alpha=0;p.mu=0;p.beta=1.0;p.w=1.0*ones(p.nx);
p.D=-2*eye(p.nx)+diag(ones(p.nx-1,1),1)+diag(ones(p.nx-1,1),-1);
%Boundary Conditions
%p.D(1,1)=0.0;p.D(1,2)=0.0;p.D(p.nx,p.nx)=0.0;p.D(p.nx,p.nx-1)=0.0;%dirich
%p.D(1,2)=2.0;p.D(p.nx,p.nx-1)=2.0;                  %reflecting
p.D(1,p.nx)=1;p.D(p.nx,1)=1;                         %periodic
p.D=.001*p.D/p.hx^2;                                 %diffusivity
u0=(sin((p.xx'*p.xx)/(2*pi))); v0=zeros(p.nx);       %data on square
% Set graphics parameters.
fig = figure; set(fig,'color','w') %
p.side=p.nx; x = ((-p.side+1):(p.side))'/p.side;
L=ones(1*length(x)); h = surf(x,x,L); %
[a,e] = view; view(a,90);view(90,90); %
axis([-1 1 -1 1 -2 2]); caxis(26.9*[-1.5 1]);
colormap(hot); shading flat; axis off%
n=0; u=u0; v=v0;

while ishandle(fig)
    u=u+p.ht*(v);
    v=v+p.ht*(p.w-v-(p.beta*cos(p.mu*n*p.ht)).*sin(u)...
    -p.alpha*v.*(v.^2-1)+p.D*u+u*p.D);
    n=n+1;
    if(mod(n,2)==0)
        uu=[cos(u) cos(u); cos(u) cos(u)];            vv=uu;    %2x2 patches clari
        set(h,'zdata',uu,'cdata',10*vv);
        drawnow
    end
    if(n>=p.nt)break;end
end;
```

Chapter 2

Neuroscience Simulations

2.1 Atoll Simulation

```
%atoll
%This solves the atoll model and describes
%parabolic bursting

tt=0:.1:300; x0=zeros(2,1); x0(2,1)=pi; x=x0;
options=odeset('AbsTol',1e-8,'RelTol',1e-6,'maxstep',1e-2); %
tic %
[t x] = ode15s(@atorhs,tt,x0,options); %
toc %
[m n]=size(x); %
figure(1),clf
subplot(2,1,1),plot(t,cos(x(:,1))),xlabel('t'),ylabel('cos x')
subplot(2,1,2),plot(t,cos(x(:,2))), xlabel('t'),ylabel('cos y')
```

2.1.1 @atorhs

```
function xp=atorhs(t,x)

xp=x;

xp(1)=5*(1+0.3+cos(x(1))-cos(x(2)));
xp(2)=0.04*(1+cos(x(2))+10*cos(x(1)));
```

2.2 Winfree's time crystal

```
%time crystal - Exercise 6.3

set(gcf,'Renderer','zbuffer')

x=0:.03:(2*pi);a=0:.03:2.1; [X A]=meshgrid(x,a); phase=-pi/2;
f=atan(sin(X+phase)./(A+cos(X+phase)));

figure(1), contour(X,A,f,100) %
xlabel('\phi'),ylabel('A') %
colormap bone%

figure(2), plot3(X,A,f,'.') %
axis %
vis3d %
for i=1:36
    camorbit(10,0,'data',[0 0 1])
    xlabel('\phi'),ylabel('A'),zlabel('\psi')
    drawnow
end
```

2.3 Attention: Winner takes all in the frequency domain.

```
%atn imn2

global N N2 N3 input s A

N=15;N2=2*N;N3=3*N; %
input=zeros(N,1); s=input; input(1)=0.01;

A=zeros(N); A=diag(ones(N-1,1),1)+diag(ones(N-1,1),-1);
A(1,2)=2;A(N-1,N)=2;

x0=0.5*pi*ones(N3,1); x=x0; %
tf=5000; Fs=32; tt=0:1/Fs:tf;%
options=odeset('AbsTol',1e-8,'RelTol',1e-6,'maxstep',1e-2);%
tic %
[t x]=ode15s(@atn2rhs,tt,x0,options); %
figure(1), clf %
hold on %
toc%
for j=1:N3
    %plot(t,max(cos(x(:,j)),0)/2+j);
    plot(t,cos(x(:,j))/2+j);
end %
xlabel('t'),ylabel('0.5*cos x_j(t)')
```

2.3.1 @atn2rhs

```
function xp=atn2rhs(t,x) %
global N N2 N3 input s A

z=cos(x);zp=max(z,0.0);

nnn=2*floor(t/500); %
if(nnn<=12)
```

```
    input(min(nnn+1,N),1)=(0.1+nnn*0.1);
else
    input(min(nnn+1,N),1)=0.0;
end

if(nnn>=8)input(3,1)=0; end %
if(nnn>=10)input(9,1)=0; end
v=ones(1,N);%/N;

xp=x; xp(1:N)=5*(1+z(1:N)-zp(N+1:N2)+input(1:N));
xp(N+1:N2)=0.04*(1+z(N+1:N2)+tanh(2*zp(1:N)-10*v*zp(N+1:N2)));
xp(N2+1:N3)=10*(0.1+z(N2+1:N3)+zp(N+1:N2));
```

2.4 Breathing

```
%Breath

global w wmr alpha m B

w=1.0; wmr=0.01; alpha = 1; %
m = 70.0; B = 50.0;%
Fs=16; tf=500; tt=0.01:1/Fs:tf;%

options=odeset('AbsTol',1e-8,'RelTol',1e-6,'maxstep',1e-2);
x0=ones(5,1);x=x0;

tic; %
[t x]=ode15s(@brerhs,tt,x0,options); %
toc;

figure(1), clf;
plot(t,cos(x(:,1)),t,2+cos(x(:,2)),t,4+cos(x(:,3)),t,6+tanh(x(:,4)));
```

2.4.1 @brerhs

```
function xp=brerhs(t,x)

global w wmr alpha m B%
xp=x;
xp(1)=w+alpha+cos(x(1))-0.3*(1-cos(x(1)))*(1+cos(x(3))); %Inspiratory
xp(2)=w+alpha+cos(x(2))-2*(1-cos(x(2)))*(1+cos(x(1))); %Expiratory
xp(3)=wmr + cos(x(3))+(1-cos(x(3)))*(max(x(4)-10,0.0)); %Baroreceptor
xp(4)=x(5); %Diaphragm displacement
xp(5)=(B*cos(2*pi*t)-x(5)-.5*x(4)+B*(1+cos(x(1)))-(B/2)*(1+cos(x(2))))/m;
```

2.5 Hippocampus pattern

```
%hippo.m
%this is to simulate a hippo segment
p.eps =1/8.0; p.tf=10;p.ht=0.01;p.tt=0:p.ht:p.tf;
p.delphi=0:.1:(2*pi); p.delphi=p.delphi+(pi/3);
p.nn=length(p.delphi); %
x=0*ones(2*p.nn,1);x0=x;
options=odeset('abstol',1e-8,'reltol',1e-8,'maxstep',0.1);
tic%
[t x]=ode45(@hiprhs,p.tt,x0,options, p);
toc%
[m n]=size(x); %
figure(1),clf %
plot((x(m,(p.nn+1):(2*p.nn)))/p.tf)%
xlabel('Cell Number'); ylabel('Output Frequency');
  if(0>1)       %switch off or on following calculation
   figure(2),clf
   axis([0 p.tf 1 nn])
   hold on
   for k=1:nn
    plot(t(:,1),k+zeros(size(t)),'.')
    plot(t(:,1),k+.5*(cos(x(:,p.nn+k-1))),'.')
   end
   xlabel('Time');ylabel('Voltage')
   hold off
  end
figure(3),clf, hold on %
for k=2:p.nn
 g=cos(x(:,p.nn+k-1))+0*randn;
 h=fft(g,512);
 pow=h.*conj(h)/512;
 kk=512*(3:length(p.delphi))/512;
 %semilogy(kk,k+pow(3:length(p.delphi)));
 plot(kk,k+pow(3:length(p.delphi)));
end

xlabel('Frequency');ylabel('Power Spectrum'); hold off
```

2.5.1 @hiprhs

```
function xp=filrhs(t,x,p)

xp=x; %
for i=1:p.nn %
xp(i+p.nn)=5*(1+0.5*p.eps+x(i))/p.eps;
xp(i)=5*(-x(i)+cos(x(i+p.nn))+....
2.5*(cos(x(i+p.nn)))*(cos(t)+cos(t+p.delphi(i))));%full sys
end
```