DS 290: Modelling and Simulation

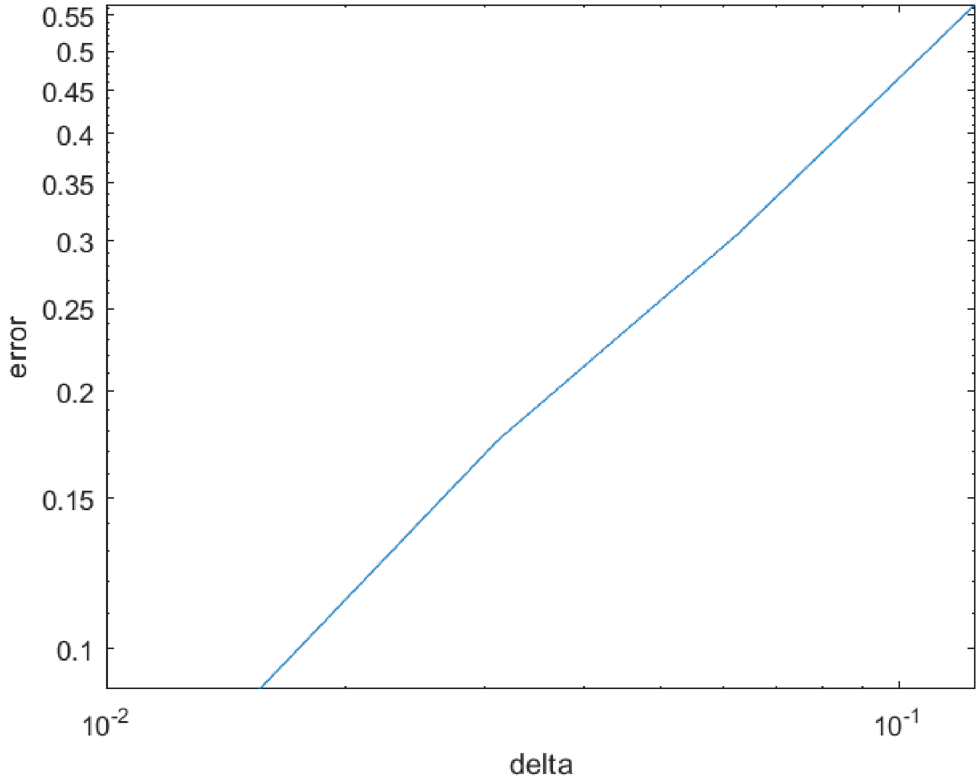
Weak convergence of linear sdes

considering the sde $\mathrm{dX}_t = a\, X_t\, \mathrm{dt} + b\, X_t\, \mathrm{dW}_t$

```
% a = 1.5, b = 1, X0 = Y0 = 1, T =1
% delta = 2^(-3), 2^(-4), 2^(-5), 2^(-6)
%===============================================================
M = 20; % denote the number of batches
N = 100; % denotes the number of sample paths
Del = [2^(-3),2^(-4),2^(-5),2^(-6)];
errorestimate = zeros(size(Del,2),1);
c = 1;
x0 = 1.0;
b = 1.0;
a = 1.5;
for delta = Del
    delta;
    su = 0;
    for m = 1:1:M
        t = 0:delta:1;
        wt = zeros(size(t,2),1);
        for p=1:1:N
            delw = sqrt(delta)*randn(size(t,2)-1,1);
            for i=2:1:size(t,2)
                wt(i) = wt(i-1) + delw(i-1);
            end
            Y0 = 1;
            Y = zeros(size(t,2),1);
            Y(1) = 1.0;
            for j=2:1:size(t,2)
                Y(j) = Y(j-1) + a*Y(j-1)*delta + b*Y(j-1)*delw(j-1);
            end
            su = su + Y(end) -exp(a-0.5*b^(2) + b*wt(end));
        end
    end
    errorestimate(c) = abs(su)/(M*N);
    c = c + 1;
end
errorestimate
```

```
errorestimate = 4×1
    0.5666
    0.3047
    0.1755
    0.0897
```

```
figure(1)
loglog(Del,errorestimate)
ylabel('error')
xlabel('delta')
hold off
```



Compute the slope from the plot for estimating the convergence

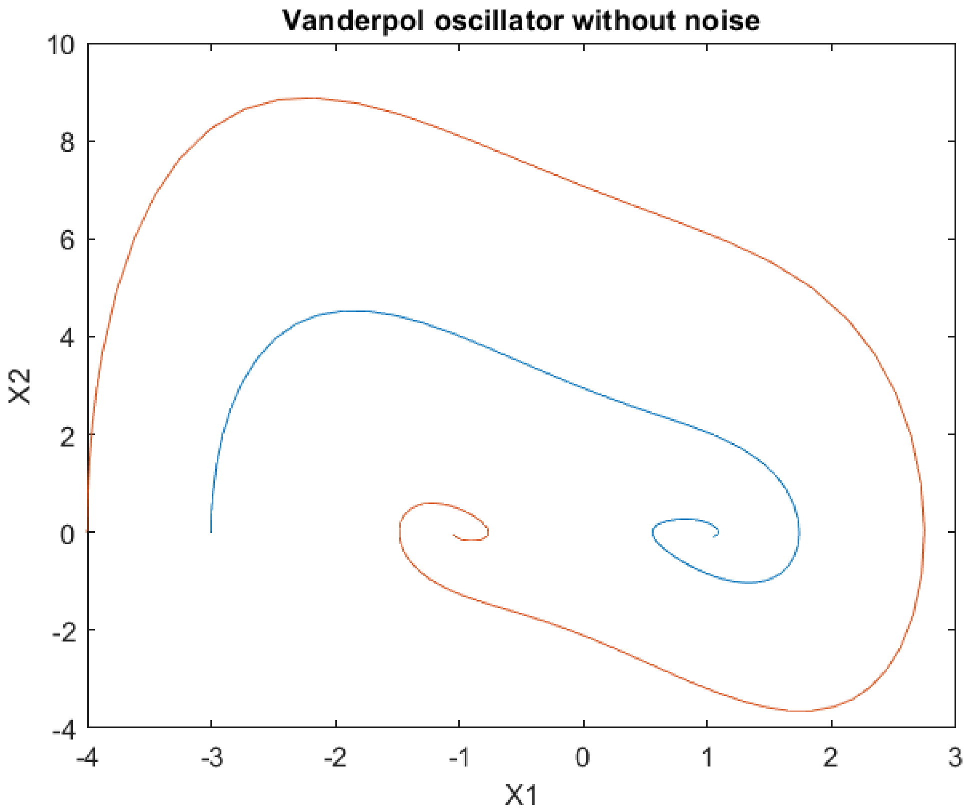
Demonstration of strong schemes using the Noisy Duffing Vanderpol Oscillator

$$\mathrm{dX}_t^1 = X_t^2 \mathrm{dt}$$

$$\mathrm{dX}_t^2 = \left\{X_t^1\left(\alpha - \left(X_t^2\right)^2\right) - X_t^2\right\}\mathrm{dt} + \sigma X_t^1 \mathrm{dW}_t$$

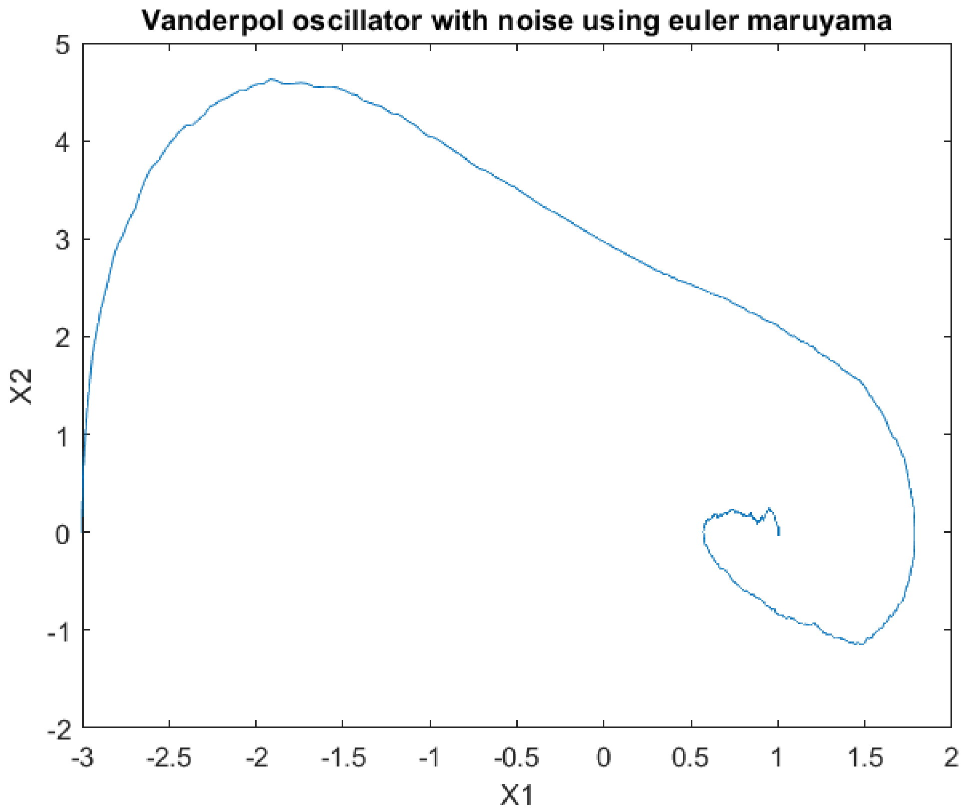
a. without noise

```
% assuming alpha = 1
clear all;
alpha = 1;
X0 = [-3;0];
[t,y] = ode45(@oscillator,[0,8],X0);
[t,y1] = ode45(@oscillator,[0,8],[-4,0]);
figure(2)
plot(y(:,1),y(:,2)),hold on;plot(y1(:,1),y1(:,2))
title('Vanderpol oscillator without noise')
xlabel('X1')
ylabel('X2')
hold off
```



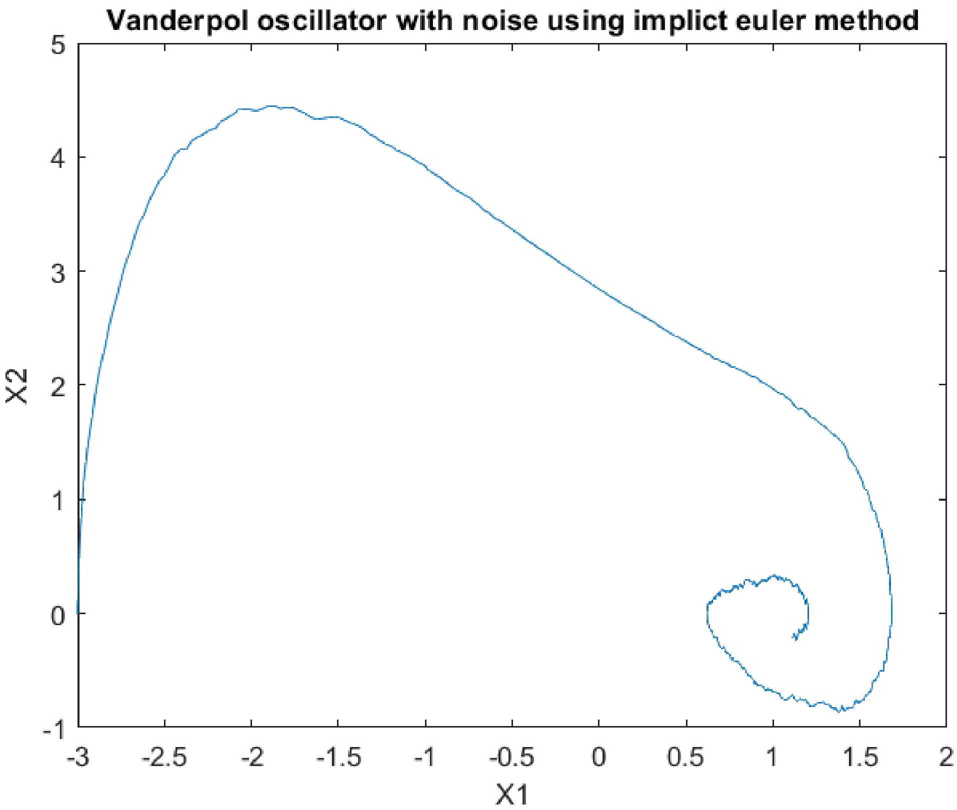
b. with noise using euler maruyama scheme

```
clear all;
T = 8; delta = 2^(-7);
t = 0:delta:T;
alpha = 1;
X0 = [-3;0];
X = zeros(size(t,2),2);
X(1,:) = X0;
sigma = 0.1 ;
delw = sqrt(delta)*randn(size(t,2)-1,1);
for j=2:1:size(t,2)
    X(j,1) = X(j-1,1) + X(j-1,2)*delta;
    X(j,2) = X(j-1,2) + (X(j-1,1)*(alpha -X(j-1,1)^(2))- X(j-1,2))*delta + sigma*X(j-1,1)*delw(
end
figure(3)
plot(X(:,1),X(:,2))
title('Vanderpol oscillator with noise using euler maruyama')
xlabel('X1')
ylabel('X2')
hold off
```



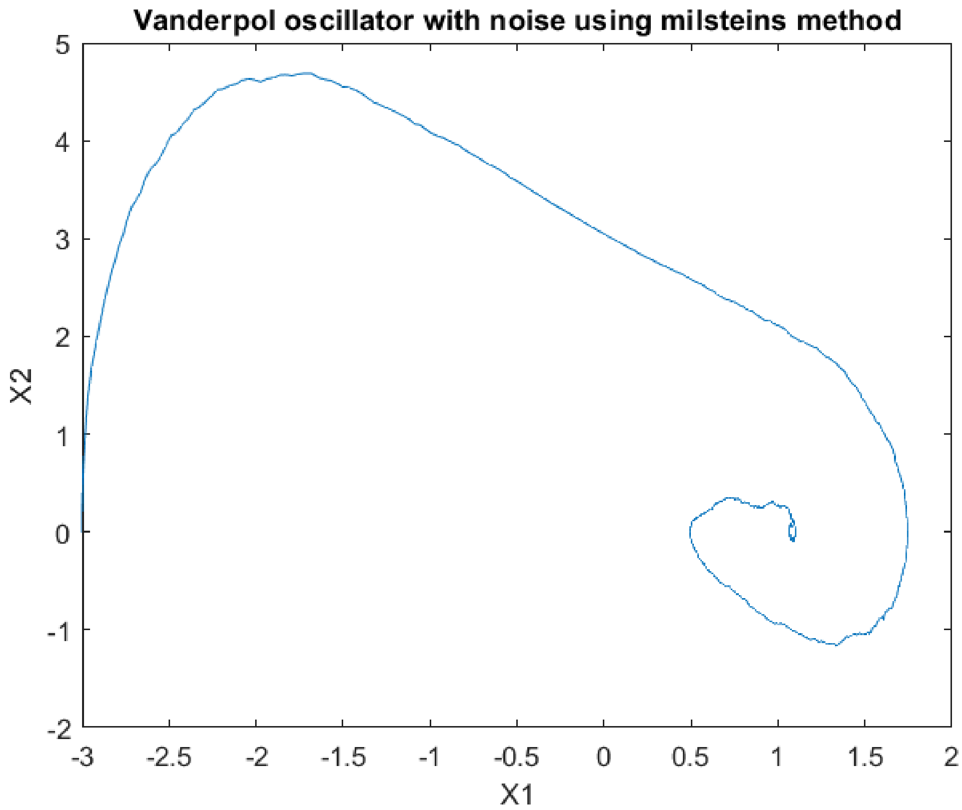
c. with noise using implicit euler scheme

```
clear all;
alpha = 1;
T = 8; delta = 2^(-7);
t = 0:delta:T;
X0 = [-3;0];
X = zeros(size(t,2),2);
X(1,:) = X0;
sigma = 0.1 ;
delw = sqrt(delta)*randn(size(t,2)-1,1);
options = optimoptions('fsolve','Display','off');
for j=2:1:size(t,2)
    F = @(x) [X(j-1,1) + x(2)*delta - x(1);
         X(j-1,2) + (x(1)*(alpha -x(1)^(2))- x(2))*delta + sigma*X(j-1,1)*delw(j-1) - x(2)];
    x0 = [X(j-1,1);X(j-1,2)];
    x = fsolve(F,x0,options);
    X(j,1) = x(1);
    X(j,2) = x(2);
end
figure(4)
plot(X(:,1),X(:,2))
title('Vanderpol oscillator with noise using implict euler method')
xlabel('X1')
ylabel('X2')
hold off
```



with noise using Milsteins scheme:

```
T = 8; delta = 2^(-7);
t = 0:delta:T;
X0 = [-3;0];
X = zeros(size(t,2),2);
X(1,:) = X0;
sigma = 0.1 ;
delw = sqrt(delta)*randn(size(t,2)-1,1);
for j=2:1:size(t,2)
    X(j,1) = X(j-1,1) + X(j-1,2)*delta;
    X(j,2) = X(j-1,2) + (X(j-1,1)*(alpha -X(j-1,1)^(2))- X(j-1,2))*delta + sigma*X(j-1,1)*delw(
end
figure(5)
plot(X(:,1),X(:,2))
title('Vanderpol oscillator with noise using milsteins method')
xlabel('X1')
ylabel('X2')
hold off
```



d. comparison of their execution times

```
% compare the execution times of all the above methods and observe
```

Stiff sytems:

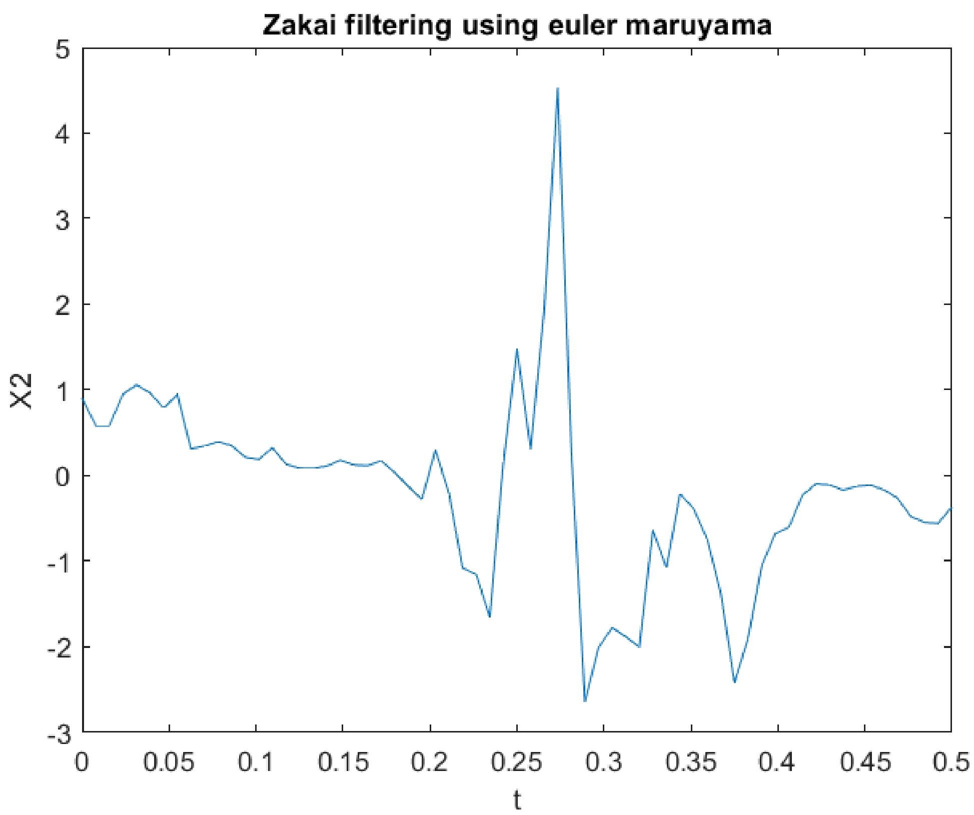
Zakai Filtering example:

$$d\left(\begin{bmatrix} X_t^1 \\ X_t^2 \end{bmatrix}\right) = \begin{bmatrix} -50 & 50 \\ 50 & -50 \end{bmatrix}\left(\begin{bmatrix} X_t^1 \\ X_t^2 \end{bmatrix}\right)\mathrm{dt} + \begin{bmatrix} 15 & 0 \\ 0 & 0 \end{bmatrix}\left(\begin{bmatrix} X_t^1 \\ X_t^2 \end{bmatrix}\right)\mathrm{dW}_t$$

Euler maruyama scheme:

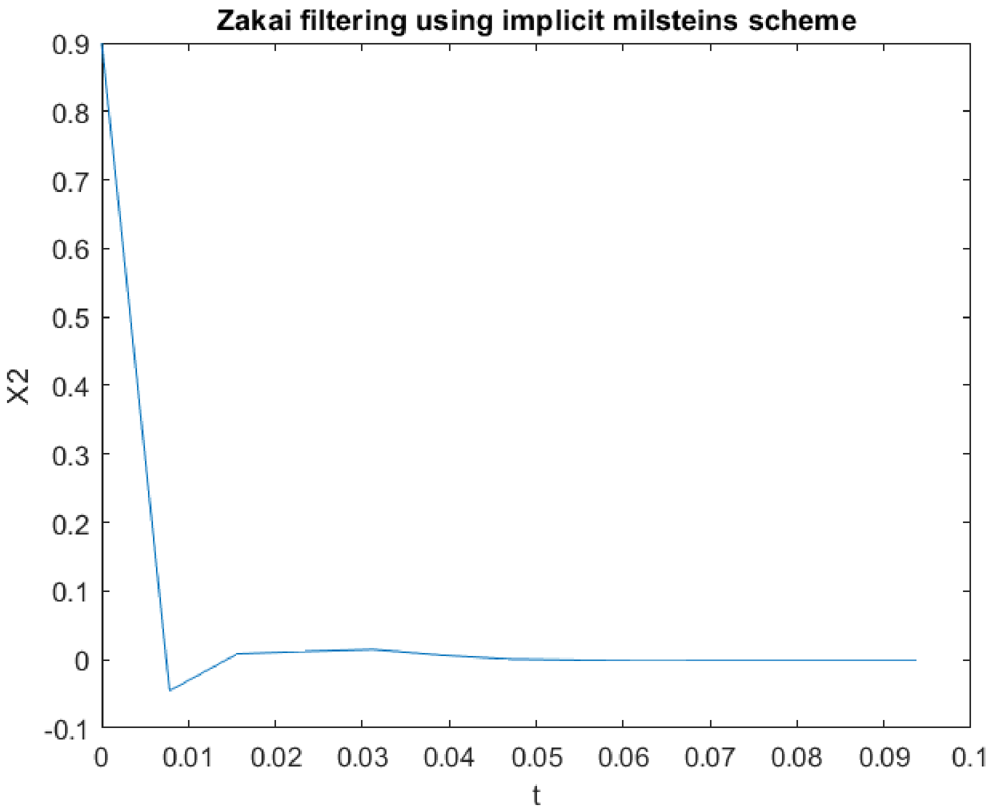
```
T = 0.5; delta = 2^(-7);
t = 0:delta:T;
X0 = [0.1;0.9];
X = zeros(size(t,2),2);
X(1,:) = X0;
delw = sqrt(delta)*randn(size(t,2)-1,1);
for j=2:1:size(t,2)
    X(j,1) = X(j-1,1) -50*X(j-1,1)*delta + 50*X(j-1,2)*delta + 15*X(j-1,1)*delw(j-1) ;
    X(j,2) = X(j-1,2) + 50*X(j-1,1)*delta -50*X(j-1,2)*delta ;
end
figure(6)
plot(t,X(:,2))
```

```
xlabel('t')
ylabel('X2')
title('Zakai filtering using euler maruyama')
```



Implicit Milstein scheme:

```
warning('off','all')
clear all;
T = 0.1; delta = 2^(-7);
t = 0:delta:T;
X0 = [0.1;0.9];
X = zeros(size(t,2),2);
X(1,:) = X0;
delw = sqrt(delta)*randn(size(t,2)-1,1);
options = optimoptions('fsolve','Display','off');
for j=2:1:size(t,2)
     F = @(x) [X(j-1,1) -50*x(1)*delta + 50*x(2)*delta + 15*X(j-1,1)*delw(j-1) + 0.5*(delw(j-1)
          X(j-1,2) + 50*x(1)*delta -50*x(2)*delta - x(2)];
     x0 = [X(j-1,1);X(j-1,2)];
     x = fsolve(F,x0,options);
     X(j,1) = x(1);
     X(j,2) = x(2);
end
figure(7)
plot(t,X(:,2))
xlabel('t')
ylabel('X2')
title('Zakai filtering using implicit milsteins scheme')
```



Comparison of execution times

```
% compare the execution times of the abovs two methods. Comment
```

Bifurcation : For understanding bifuractions please refer to Differential Equations and Dynamical systems by lawrence perkko.

An example of bifurcations demonstrated in the Lecture is simulated below, The code involves computation of the Lyapunov exponent of the linearized sde[page no: 58].

Linearized system of noisy brusselator equations

Linearized system (Ito)

$$\mathrm{dX}_t^1 = \left\{(\alpha - 1)X_t^1 + X_t^2\right\}\mathrm{dt} + \sigma X_t^1 \mathrm{dW}_t$$

$$\mathrm{dX}_t^2 = \left\{-\alpha X_t^1 - X_t^2\right\}\mathrm{dt} - \sigma X_t^1 \mathrm{dW}_t$$

Stochastic Bifurcation :

Using implicit Euler scheme

```
warning('off','all')
clear all;
alpha = 1;
sigma = 0.2;
T = 1000; delta = 2^(-7);
t = 0:delta:T;
```

```
X0 = [0.25,0.25];
X = zeros(size(t,2),2);
X(1,:) = X0;
delw = sqrt(delta)*randn(size(t,2)-1,1);
options = optimoptions('fsolve','Display','off');
for j=2:1:size(t,2)
     F = @(x) [X(j-1,1) + ((alpha -1)*x(1) + x(2))*delta + sigma*X(j-1,1)*delw(j-1) - x(1);
         X(j-1,2) + (-alpha*x(1) - x(2))*delta - sigma*X(j-1,1) *delw(j-1)  - x(2)];
     x0 = [X(j-1,1);X(j-1,2)];
     x = fsolve(F,x0,options);
     X(j,1) = x(1);
     X(j,2) = x(2);
end
%==================================================
% For lyapunov exponent
nt = size(t,2)-1;
expvalue = (1/(nt*delta));
su = 0;
for i=1:1:nt
    su = su + log(norm(X(i+1))/norm(X(i)));
end
expvalue = expvalue *su
```

```
expvalue = -0.0125
```

Note: Here the Lyapunov exponent depends on the value of sigma chosen

```
% The sign of lyapunov exponent changes when alpha is 3,
% so bifurcation occurs for sigma = 0.2 at alpha = 3.
```

```
function dydt = oscillator(t,y)
    dydt = [y(2); (1-y(1)^2)*y(1)-y(2)];
end
```