Problem 1

(a) When the system reaches its equilibrium points, it satisifies:

$$\dot{x}_1 = Ax_1 - Bx_1x_2 = 0$$

$$\dot{x}_2 = -Cx_2 + Dx_1x_2 = 0$$

Since A=1, B=1, C=1, and D=1, the above equations have two solutions, i.e., $x_1 = 0, x_2 = 0$ or $x_1 = C/D = 1, x_2 = A/B = 1$.

(b). The matlab code that uses the Euler method to do the integration is listed below:

```
% Initial condtions
clc, clear;
t_start = 0; dt = 0.1; t_end = 2;
StepCnt = (t_end-t_start)/dt;
t = (t_start:dt:t_end)';

A = 1; B = 1; C = 1; D = 1;

% Integration with ode45, treated as the
reference
options = odeset('reltol',1e-
6,'abstol',1e-6);
[ts,x] = ode45(@prey_predator_model,t,[5
1],options,A,B,C,D);

% Integration with Euler Method
x1_Euler = zeros(StepCnt,1);
x2_Euler = zeros(StepCnt,1);
x1_Euler(1,1) = 5; x2_Euler(1,1) = 1;
for i=1:StepCnt
    x1_Euler(i+1) = x1_Euler(i) + dt *
(A*x1_Euler(i) -
B*x1_Euler(i)*x2_Euler(i));
    x2_Euler(i+1) = x2_Euler(i) + dt * (-
C*x2_Euler(i) +
D*x1_Euler(i)*x2_Euler(i));
end
```

```
% Differential Equations of Prey-Predator
Model
function dx = prey_predator_model(t, x,
A, B, C, D)
dx=[A*x(1)-B*x(1)*x(2);
    -C*x(2)+D*x(1)*x(2)];
end
```

The results are shown in Table 1 and Fig. 1. In Table 1, x1 and x2 are calculated with function ode45 and are treated as the references, while x1_Euler and x2_Euler are obtained by Euler method. The results are also visualized in Fig.1, with error visualized in Fig. 2.

Table 1 Results of Euler Method

t	x1	x2	x1_Euler	x2_Euler	x1 error	x2 error
0.00	5.0000	1.0000	5.0000	1.0000	0.0000	0.0000
0.10	4.8871	1.4864	5.0000	1.4000	0.1129	0.0864
0.20	4.5085	2.1565	4.8000	1.9600	0.2915	0.1965
0.30	3.8589	2.9709	4.3392	2.7048	0.4803	0.2661
0.40	3.0383	3.7980	3.5995	3.6080	0.5612	0.1901
0.50	2.2170	4.4663	2.6607	4.5459	0.4437	0.0796
0.60	1.5328	4.8674	1.7173	5.3008	0.1845	0.4334
0.70	1.0322	4.9994	0.9787	5.6810	0.0535	0.6816
0.80	0.6935	4.9255	0.5206	5.6689	0.1730	0.7434
0.90	0.4728	4.7206	0.2775	5.3971	0.1953	0.6765
1.00	0.3303	4.4441	0.1555	5.0072	0.1748	0.5631
1.10	0.2377	4.1357	0.0932	4.5843	0.1445	0.4486
1.20	0.1765	3.8197	0.0598	4.1686	0.1167	0.3490
1.30	0.1352	3.5100	0.0408	3.7767	0.0944	0.2667
1.40	0.1068	3.2144	0.0295	3.4144	0.0773	0.2001
1.50	0.0868	2.9366	0.0224	3.0831	0.0644	0.1465
1.60	0.0724	2.6783	0.0177	2.7817	0.0547	0.1034
1.70	0.0620	2.4397	0.0146	2.5084	0.0474	0.0687
1.80	0.0543	2.2203	0.0124	2.2612	0.0419	0.0409
1.90	0.0485	2.0194	0.0108	2.0379	0.0377	0.0185
2.00	0.0442	1.8357	0.0097	1.8363	0.0346	0.0006

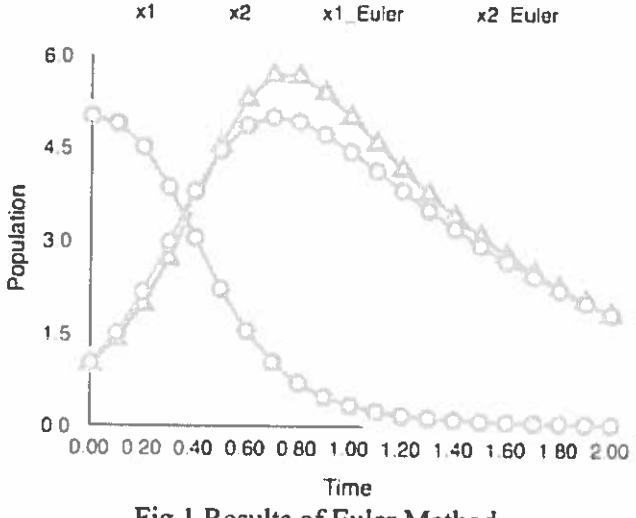

Fig 1 Results of Euler Method

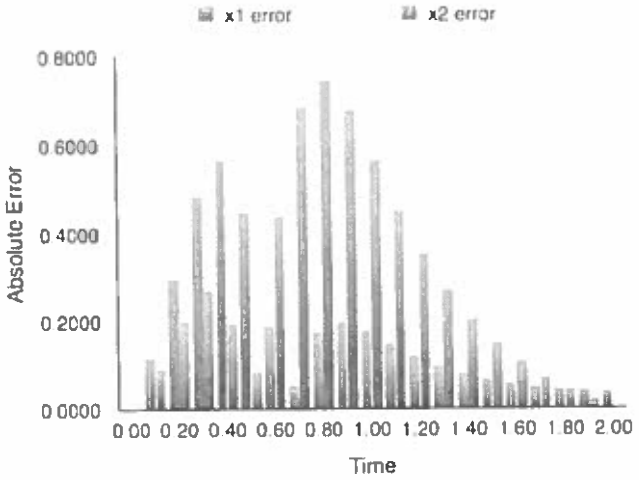

Fig. 2 Absolute Error of Euler Method

If we extend the end time to 20, we can get the following results shown in Fig. 3.

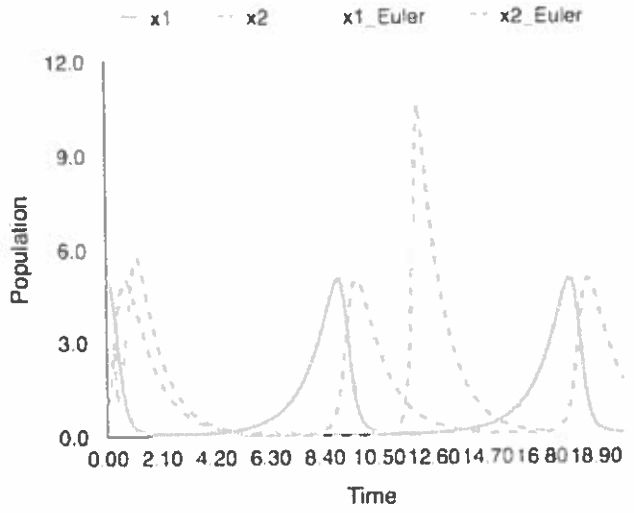

Fig. 3

Problem 2

The matlab code that uses the Runge-Kutta's method to do the integration is listed below:

```
% Integration with Runge-Kutta Method
x1_RK2 = zeros(StepCnt,1);
x2_RK2 = zeros(StepCnt,1);
x1_RK2(1,1) = 5; x2_RK2(1,1) = 1;
for i=1:StepCnt
    k1(1,1) = dt * (A*x1_RK2(i) -
B*x1_RK2(i)*x2_RK2(i));
    k1(2,1) = dt * (-C*x2_RK2(i) +
D*x1_RK2(i)*x2_RK2(i));
    k2 = k1 + dt * k1;
    k = (k1 + k2) / 2;
    x1_RK2(i+1) = x1_RK2(i) + k(1);
    x2_RK2(i+1) = x2_RK2(i) + k(2);
end
```

The results are shown in Table 2. The results are also visualized in Fig.4, with error visualized in Fig. 5. Reference are taken from the results obtained by ode45 in Table 1.

Table 2 Results of Second Order Runge-Kutta Method

t	x1_RK2	x2_RK2	x1_RK2_error	x2_RK2_error
0.00	5.0000	1.0000	0.0000	0.0000
0.10	4.9000	1.4800	0.0129	0.0064
0.20	4.5358	2.1456	0.0273	0.0110
0.30	3.8936	2.9629	0.0348	0.0081
0.40	3.0702	3.7983	0.0319	0.0002
0.50	2.2444	4.4690	0.0274	0.0028
0.60	1.5601	4.8641	0.0273	0.0033
0.70	1.0607	4.9894	0.0285	0.0100
0.80	0.7210	4.9135	0.0274	0.0120
0.90	0.4970	4.7110	0.0242	0.0097
1.00	0.3505	4.4387	0.0202	0.0054
1.10	0.2540	4.1348	0.0163	0.0009
1.20	0.1896	3.8226	0.0131	0.0029
1.30	0.1457	3.5160	0.0105	0.0060
1.40	0.1153	3.2225	0.0085	0.0082
1.50	0.0938	2.9463	0.0070	0.0097
1.60	0.0783	2.6889	0.0059	0.0107
1.70	0.0670	2.4509	0.0050	0.0112
1.80	0.0587	2.2318	0.0044	0.0115
1.90	0.0524	2.0309	0.0039	0.0115
2.00	0.0477	1.8471	0.0035	0.0114

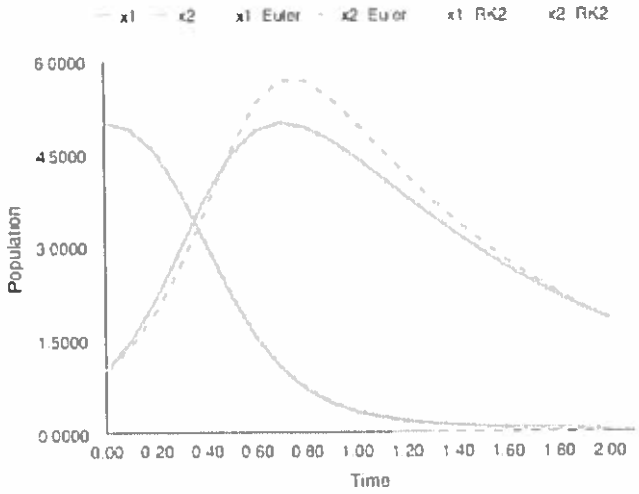

Fig. 4 Results of Second Order Runge-Kutta Method

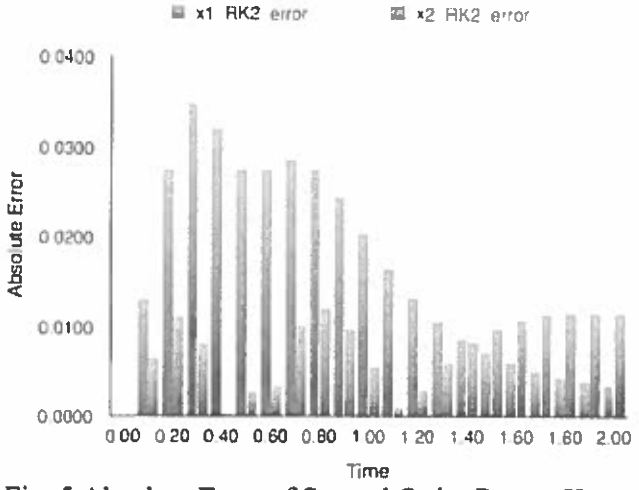

Fig. 5 Absolute Error of Second Order Runge-Kutta Method

The average absolute error of x1 in Euler method is 0.1612, while that of Second Order Runge-Kutta method is 0.0162. The average absolute error of x2 in Euler method is 0.2647, while that of Second Order Runge-Kutta method is 0.0073. Obviously, the Second Order Runge-Kutta method is more accurate than the Euler method in this case.

Problem 3

The matlab code that uses the Second Order Adams-Bashforth method to do the integration is listed below:

```
% Integration with Adams-Bashforth Method
x1_AB2 = zeros(StepCnt,1);
x2_AB2 = zeros(StepCnt,1);
x1_AB2(1,1) = 5; x2_AB2(1,1) = 1;
k1 = zeros(2,1);
k2 = zeros(2,1);
for i=1:1
    k1(1) = dt * (A*x1_AB2(i) -
B*x1_AB2(i)*x2_AB2(i));
    k1(2) = dt * (-C*x2_AB2(i) +
D*x1_AB2(i)*x2_AB2(i));
    tmp = [x1_AB2(i)+k1(1);
x2_AB2(i)+k1(2)];
    k2(1) = dt * (A*tmp(1) -
B*tmp(1)*tmp(2));
    k2(2) = dt * (-C*tmp(2) +
D*tmp(1)*tmp(2));
    x1_AB2(i+1) = x1_AB2(i) + (k1(1) +
k2(1)) / 2;
    x2_AB2(i+1) = x2_AB2(i) + (k1(2) +
k2(2)) / 2;
end
for i=2:StepCnt
    k1(1) = dt * (A*x1_AB2(i) -
B*x1_AB2(i)*x2_AB2(i));
    k1(2) = dt * (-C*x2_AB2(i) +
D*x1_AB2(i)*x2_AB2(i));
    k2(1) = dt * (A*x1_AB2(i-1) -
B*x1_AB2(i-1)*x2_AB2(i-1));
    k2(2) = dt * (-C*x2_AB2(i-1) +
D*x1_AB2(i-1)*x2_AB2(i-1));
    x1_AB2(i+1) = x1_AB2(i) + (3*k1(1) -
k2(1)) / 2;
    x2_AB2(i+1) = x2_AB2(i) + (3*k1(2) -
k2(2)) / 2;
end
```

The results are shown in Table 3.

Table 3 Results of Second Order Adams-Bashforth Method

t	x1_AB2	x2_AB2	x1_AB2_error	x2_AB2_error
0.00	5.0000	1.0000	0.0000	0.0000
0.10	4.9000	1.4800	0.0129	0.0064
0.20	4.5472	2.1458	0.0387	0.0107
0.30	3.8833	2.9989	0.0244	0.0280
0.40	2.9794	3.9154	0.0589	0.1173
0.50	2.0646	4.6456	0.1524	0.1793
0.60	1.3699	4.9999	0.1628	0.1325
0.70	0.9243	5.0301	0.1079	0.0307
0.80	0.6395	4.8805	0.0540	0.0450
0.90	0.4535	4.6356	0.0193	0.0850
1.00	0.3303	4.3436	0.0000	0.1005
1.10	0.2471	4.0339	0.0094	0.1018
1.20	0.1899	3.7238	0.0134	0.0959
1.30	0.1498	3.4231	0.0146	0.0869
1.40	0.1212	3.1374	0.0144	0.0770
1.50	0.1005	2.8693	0.0137	0.0672
1.60	0.0853	2.6201	0.0128	0.0582
1.70	0.0739	2.3896	0.0119	0.0501
1.80	0.0654	2.1775	0.0111	0.0428
1.90	0.0590	1.9829	0.0105	0.0365
2.00	0.0542	1.8048	0.0099	0.0309

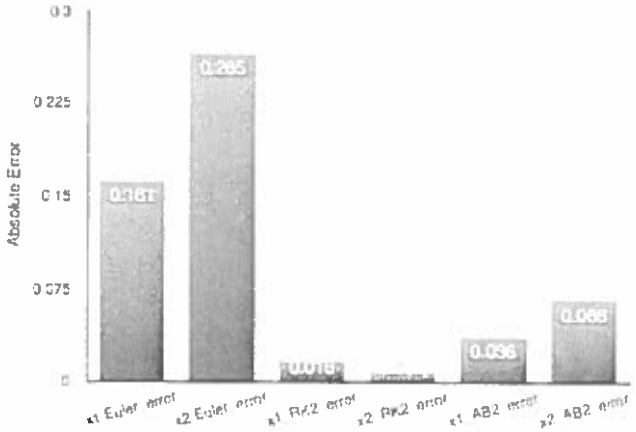

Fig. 6 Comparison of Average Absolute Error from Different Numerical Integration Methods

4. An electro-mechanical system has the following nonlinear dynamic model:

$$0.04\frac{d^2\delta}{dt^2} = 2 - 2\sin\delta - 0.01\frac{d\delta}{dt}$$

a) Write in standard state space form and find all equilibrium state vectors

$$\dot{x_1} = x_2$$
$$\dot{x_2} = 25(2 - 2\sin x_1 - 0.01x_2)$$

$$x_{2e} = 0$$
$$x_{1e} = 90\,\text{deg} = \frac{\pi}{2}$$

b) Given the initial condition $x(0) = [0, 1]$ use the RK2 method to compute $x(\Delta t)$ with step size $\Delta t = 0.01$.

$$k_1 = \Delta t f(x(t))$$
$$= 0.01 f\left(\begin{bmatrix}0\\1\end{bmatrix}\right) = 0.01\begin{bmatrix}1\\49.75\end{bmatrix} = \begin{bmatrix}0.01\\0.4975\end{bmatrix}$$

$$k_2 = \Delta t f(x(t) + k_1)$$
$$= 0.01 f\left(\begin{bmatrix}0\\1\end{bmatrix} + \begin{bmatrix}0.01\\0.4975\end{bmatrix}\right) = 0.01\begin{bmatrix}1.4975\\49.1256\end{bmatrix} = \begin{bmatrix}0.014975\\1.491256\end{bmatrix}$$

$$x(t + \Delta t) = x(t) + \frac{1}{2}(k_1 + k_2)$$
$$= \begin{bmatrix}0\\1\end{bmatrix} + \frac{1}{2}\left(\begin{bmatrix}0.01\\0.4975\end{bmatrix} + \begin{bmatrix}0.014975\\0.491256\end{bmatrix}\right) = \begin{bmatrix}0.0124875\\0.494378\end{bmatrix}$$

5. Using the data for the B7Flat_Eqv case in Lecture 24, manually create an equivalent system eliminating buses 1, 3, and 5. Give the bus admittance matrix for the modified system, and the impedance of the new lines created by equivalencing.

```
B = [1 3 5];
C = [2 4 6 7];
Ybus = [-20.83 16.667  4.167 0     0       0      0;
        16.667 -52.778 5.556 5.556 8.333   16.667 0;
        4.167  5.56    -43.1 33.3  0       0      0;
        0      5.556   33.3  -43.1 4.167   0      0;
        0      8.333   0     4.167 -29.167 0      16.667;
        0      16.667  0     0     0       -25.0  8.333;
        0      0       0     0     16.667  8.333  -25.0];

 Yee = Ybus(B,B)
 Yes = Ybus(B,C)
 Yse = Yes.'
 Yss = Ybus(C,C)

 WardEq = (Yss - Yse*inv(Yee)*Yes)*1i;
 Ybus = WardEq

 % The new lines are 2-4, 4-7, and 2-7
 fprintf('\nThe impedance changes between boundary buses 2, 4, and 7\n')
 z24 = -1/Ybus(2,1)
 z47 = -1/Ybus(4,2)
 z27 = -1/Ybus(4,1)
```

```
Yee =
  -20.8300    4.1670         0
    4.1670  -43.1000         0
         0         0  -29.1670
Yes =
   16.6670         0         0         0
    5.5600   33.3000         0         0
    8.3330    4.1670         0   16.6670
Yse =
   16.6670    5.5600    8.3330
         0   33.3000    4.1670
         0         0         0
         0         0   16.6670
Yss =
  -52.7780    5.5560   16.6670         0
    5.5560  -43.1000         0         0
   16.6670         0  -25.0000    8.3330
         0         0    8.3330  -25.0000
Ybus =
   0.0000 -35.1896i   0.0000 +13.7539i   0.0000 +16.6670i   0.0000 + 4.7618i
   0.0000 +13.7539i   0.0000 -16.2689i   0.0000 + 0.0000i   0.0000 + 2.3812i
   0.0000 +16.6670i   0.0000 + 0.0000i   0.0000 -25.0000i   0.0000 + 8.3330i
   0.0000 + 4.7618i   0.0000 + 2.3812i   0.0000 + 8.3330i   0.0000 -15.4759i

The impedance changes between boundary buses 2, 4, and 7
z24 =
   0.0000 + 0.0727i
z47 =
   0.0000 + 0.4200i
z27 =
   0.0000 + 0.2100i
```