Derek Frank
dmfrank@ucsc.edu
AMS 147
2/24/10

Homework #6

1.

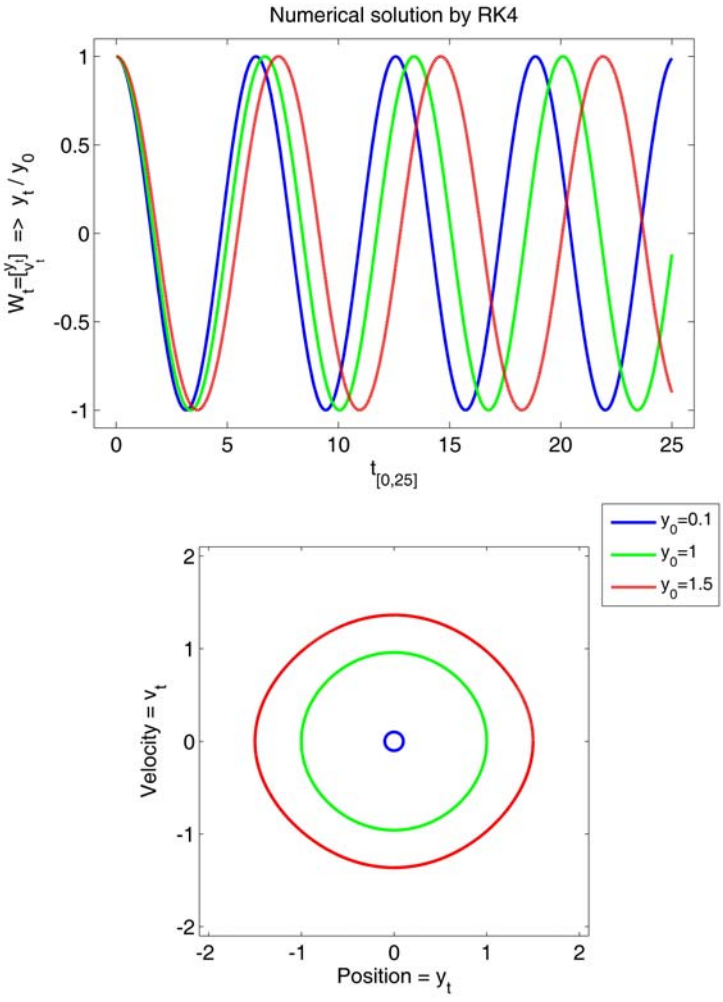
a. In this first problem, I am to use the classic fourth order Runge-Kutta method, with spatial time step, $h=0.05$, and time, $t \in [0,25]$, to solve and plot the second order differential equation $y``+siny=0$, which describes the motion of a frictionless pendulum, for three separate initial conditions:

$y_0=0.1 \Rightarrow y_0`=0$
$y_0=1.0 \Rightarrow y_0`=0$
$y_0=1.5 \Rightarrow y_0`=0$

y: displacement, $y`$: velocity

b. Using Matlab, I am going to implement the Runge-Kutta method and solve the given system for each initial condition. To plot each system with the same amplitude, I will divide each value for y_t, a vector, by y_0, a scalar. I will then proceed to plot this y_t/y_0 with respect to t.

c.



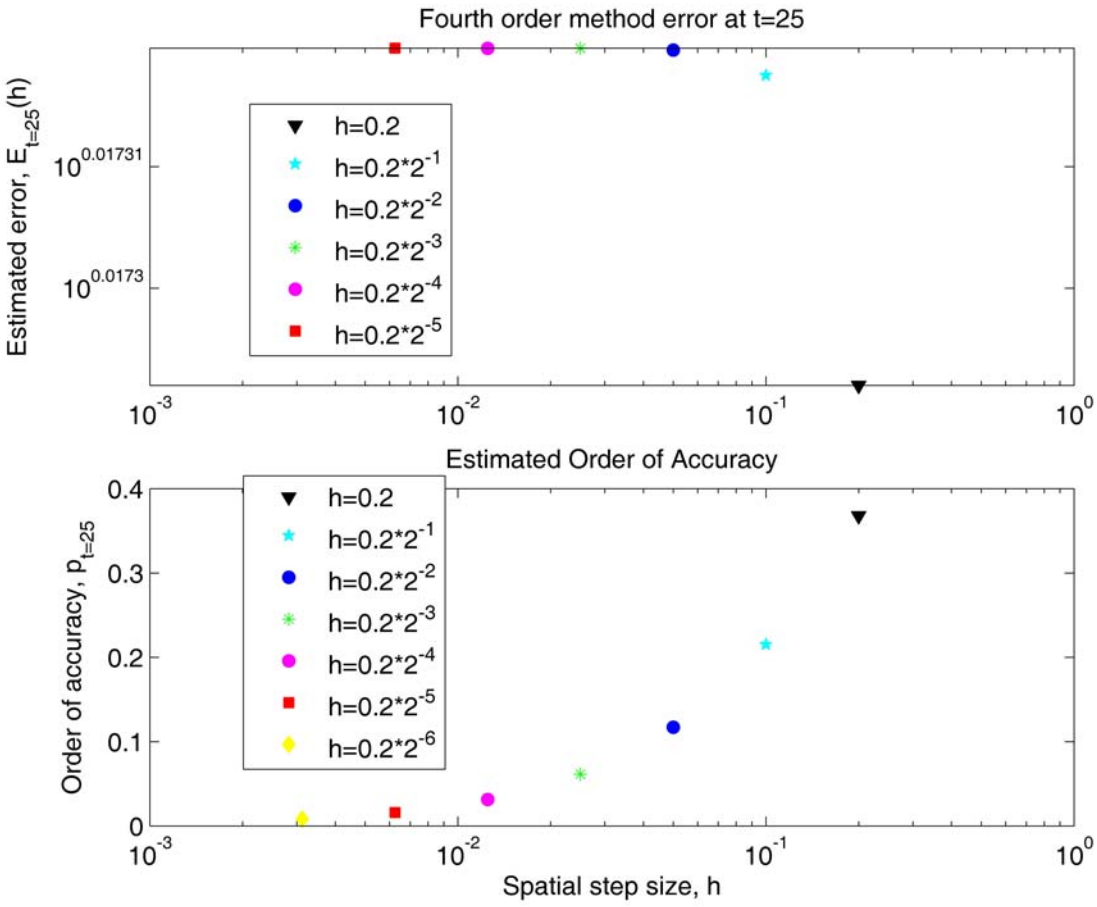
d. The results produced show the solved system for three different initial conditions. The frequencies of each differ. The solution for $y_0=0.1$ has the highest frequency, followed by $y_0=1.0$, then $y_0=1.5$ has the smallest frequency.

2.

a. In this next problem, I am to estimate and plot both the error and accuracy of the Runge-Kutta method with the initital condition $y_0=1.5$and $y_0`=0$, for $t=25$. Again, the spatial time step is $h=0.05$.

b. Using Matlab, I will implement both the numerical error estimation, $E_n(h)=(y_n(h)-y_{2n}(h/2)) / (1-1/2^p)$, and the order of accuracy estimation, p.

c.



d. The results first show that the error for the selected spatial time steps decreases as the time step becomes bigger. The error appears to be greater than one for all time steps of $h=[\ 0.2, 0.2*2^{-1}, 0.2*2^{-2}, 0.2*2^{-3}, 0.2*2^{-4}, 0.2*2^{-5}]$. Additionally, the results show an increase in accuracy as the spatial step size, h, gets smaller. Estimated order of accuracy for $t=25$:

$h=0.2 \quad \Rightarrow \ p=0.36764$
$h=0.2*2^{-1} \ \Rightarrow \ p=0.2151$
$h=0.2*2^{-2} \ \Rightarrow \ p=0.11709$
$h=0.2*2^{-3} \ \Rightarrow \ p=0.061224$
$h=0.2*2^{-4} \ \Rightarrow \ p=0.031326$
$h=0.2*2^{-5} \ \Rightarrow \ p=0.015847$

$h=0.2*2^{-6} \Rightarrow p=0.0079705$

3.

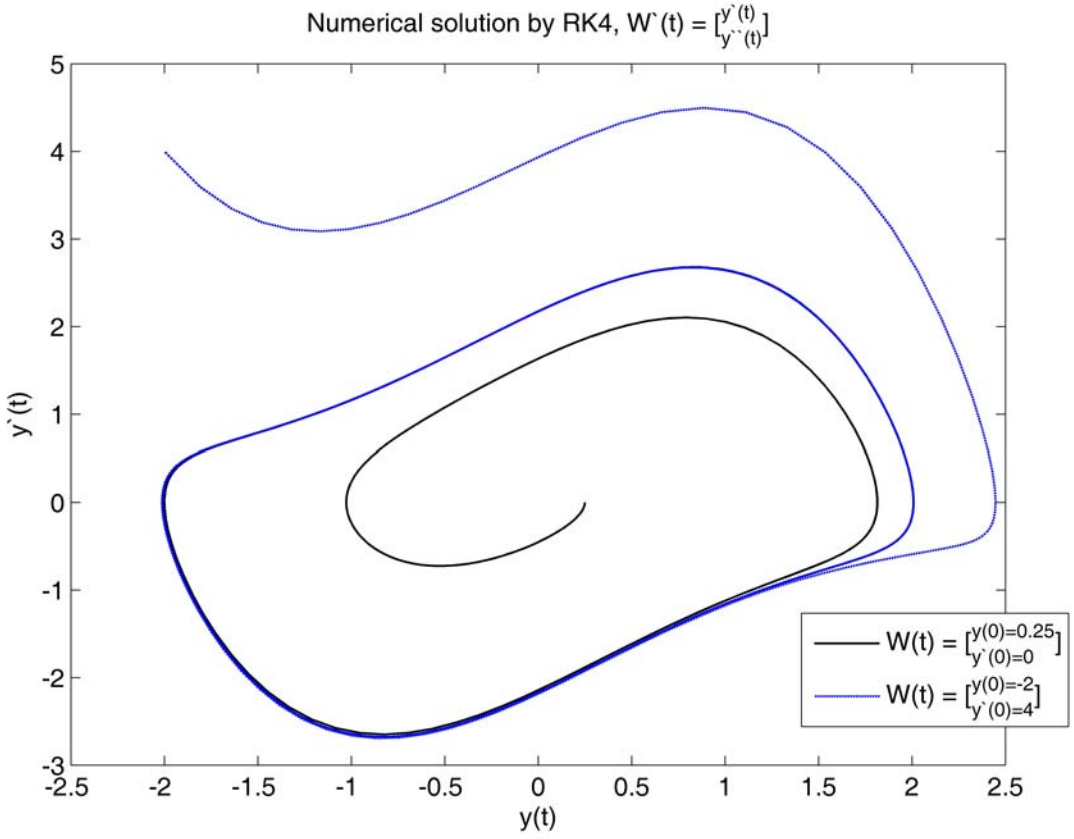
a. In this final problem, I am to solve the van der Pol equation, $y``-(1-y^2)y`+y=0$, with spatial time step, $h=0.05$, and time, $t \in [0,25]$, using the Runge-Kutta method for initial conditions:

$y_0=0.25 \Rightarrow y_0`=0$

$y_0=-2.0 \Rightarrow y_0`=4$

b. To solve this system, I implement the Runge-Kutta method using Matlab and solve once for each initial condition. I then plot the results for $y`(t)$ against $y(t)$.

c.



d. The results show the solution for each initial problem. Both curves go around the origin. The initial condition with $y(t)=0.25$ *and* $y`(t)=4$ differs from the one with $y(t)=-2$ and $y`(t)=0$ in that it begins to stray further from the origin, while the other appears to stay close and initially starts closer. Also, the curves appear to overlap for a period of time.

Appendix:

1. "f_sys.m"

```
function [z]=f_sys(w,t)
% This function calculates f_sys(w,t)
```

```
%
z=zeros(1,2);
theta=w(1);
v=w(2);
z(1)=v;
z(2)=-sin(theta);
```

“calc_sRK4a.m”

```
% This code implements the classical four stage fourth order
% Runge Kutta method to solve an ODE system. After the
% calculation,  it saves the workspace to a data file.
%
clear
%
m=2;
w0a=[.1, 0];
h=0.05;
nstep=25/h;
%
wa=zeros(nstep+1,m);
t=zeros(nstep+1,1);
t(1)=0;
wa(1,1:m)=w0a;
%
p=4;
d=[0,   1/2, 1/2, 1  ];
c=[0,   0,   0,   0  ;
   1/2, 0,   0,   0  ;
   0,   1/2, 0,   0  ;
   0,   0,   1,   0  ];
b=[1/6, 1/3, 1/3, 1/6];
k=zeros(p,m);
%
for j=1:nstep,
  for i=1:p,
    k(i,1:m)=h*f_sys(wa(j,1:m)+c(i,1:i-1)*k(1:i-1,1:m),
t(j)+d(i)*h);
  end
  wa(j+1,1:m)=wa(j,1:m)+b*k;
  t(j+1)=t(j)+h;
end
%
save data_sRK4a
```

“calc_sRK4b.m”

```
% This code implements the classical four stage fourth order
% Runge Kutta method to solve an ODE system. After the
% calculation, it saves the workspace to a data file.
%
clear
%
m=2;
w0b=[1, 0];
h=0.05;
nstep=25/h;
%
wb=zeros(nstep+1,m);
t=zeros(nstep+1,1);
```

```
t(1)=0;
wb(1,1:m)=w0b;
%
p=4;
d=[0,   1/2, 1/2, 1  ];
c=[0,   0,   0,   0  ;
   1/2, 0,   0,   0  ;
   0,   1/2, 0,   0  ;
   0,   0,   1,   0  ];
b=[1/6, 1/3, 1/3, 1/6];
k=zeros(p,m);
%
for j=1:nstep,
  for i=1:p,
    k(i,1:m)=h*f_sys(wb(j,1:m)+c(i,1:i-1)*k(1:i-1,1:m),
t(j)+d(i)*h);
  end
  wb(j+1,1:m)=wb(j,1:m)+b*k;
  t(j+1)=t(j)+h;
end
%
save data_sRK4b
```

“calc_sRK4c.m”

```
% This code implements the classical four stage fourth order
% Runge Kutta method to solve an ODE system. After the
% calculation, it saves the workspace to a data file.
%
clear
%
m=2;
w0c=[1.5, 0];
h=0.05;
nstep=25/h;
%
wc=zeros(nstep+1,m);
t=zeros(nstep+1,1);
t(1)=0;
wc(1,1:m)=w0c;
%
p=4;
d=[0,   1/2, 1/2, 1  ];
c=[0,   0,   0,   0  ;
   1/2, 0,   0,   0  ;
   0,   1/2, 0,   0  ;
   0,   0,   1,   0  ];
b=[1/6, 1/3, 1/3, 1/6];
k=zeros(p,m);
%
for j=1:nstep,
  for i=1:p,
    k(i,1:m)=h*f_sys(wc(j,1:m)+c(i,1:i-1)*k(1:i-1,1:m),
t(j)+d(i)*h);
  end
  wc(j+1,1:m)=wc(j,1:m)+b*k;
  t(j+1)=t(j)+h;
end
%
save data_sRK4c
```

"plot_sRK4.m"

```
% This code reads in the data file generated by calc_sRK4.m.
% Then it plots the two components of the numerical solution:
% position and velocity.
%
clear
figure(2);
clf reset
%
set(gcf,'position',[100,50,500,600])
set(gcf,'paperposition',[0.5,0.5,7.5,10.0])
%
load data_sRK4a
load data_sRK4b
load data_sRK4c
%
ya=wa(1:nstep+1,1)/w0a(1);
yb=wb(1:nstep+1,1)/w0b(1);
yc=wc(1:nstep+1,1)/w0c(1);
axes('position',[0.18,0.56,0.74,0.36])
%
plot(t,ya(1:nstep+1,1),'b-','linewidth',2.0)
hold on
plot(t,yb(1:nstep+1,1),'g-','linewidth', 2.0)
hold on
plot(t,yc(1:nstep+1,1),'r--', 'linewidth', 2.0)
hold on
%
set(gca,'fontsize',14)
axis([-1,26,-1.1,1.1])
set(gca,'xtick',[0:5:25])
set(gca,'ytick',[-1:0.5:1])
xlabel('t_{[0,25]}')
ylabel('W_t=[^{y_t}_{v_t}]  =>  y_t / y_0')
title('Numerical solution by RK4')
h1=legend('y_0=0.1','y_0=1','y_0=1.5');
set(h1,'fontsize',12)
%
axes('position',[0.18,0.09,0.74,0.36])
plot(wa(1:nstep+1,1),wa(1:nstep+1,2),'b-', 'linewidth', 2)
hold on
plot(wb(1:nstep+1,1),wb(1:nstep+1,2),'g-', 'linewidth', 2)
hold on
plot(wc(1:nstep+1,1),wc(1:nstep+1,2),'r--', 'linewidth', 2)
%
set(gca,'fontsize',14)
axis equal
axis([-2.1,2.1,-2.1,2.1])
set(gca,'xtick',[-2:1:2])
set(gca,'ytick',[-2:1:2])
xlabel('Position = y_t')
ylabel('Velocity = v_t')
```

2. "f_sys.m"

```
function [z]=f_sys(w,t)
% This function calculates f_sys(w,t)
%
```

```
z=zeros(1,2);
theta=w(1);
v=w(2);
z(1)=v;
z(2)=-sin(theta);
```

“calc_sRK4.m”

```
% This code implements the classical four stage fourth order
% Runge Kutta method to solve an ODE system. After the
% calculation, it saves the workspace to a data file.
%
clear
%
m=2;
w0=[1.5, 0];
%
h1=0.2; h2=0.2*2.^(-1); h3=0.2*2.^(-2); h4=0.2*2.^(-3);
h5=0.2*2.^(-4); h6=0.2*2.^(-5); h7=0.2*2.^(-6);
%
nstep1=25/h1; nstep2=25/h2; nstep3=25/h3; nstep4=25/h4;
nstep5=25/h5; nstep6=25/h6; nstep7=25/h7;
%
t1=zeros(nstep7+1,1); t2=zeros(nstep7+1,1); t3=zeros(nstep7+1,1);
t4=zeros(nstep7+1,1); t5=zeros(nstep7+1,1);
t6=zeros(nstep7+1,1); t7=zeros(nstep7+1,1);
t1(1)=0; t2(1)=0; t3(1)=0; t4(1)=0; t5(1)=0; t6(1)=0; t7(1)=0;
%
w1=zeros(nstep1+1,m); w2=zeros(nstep2+1,m); w3=zeros(nstep3+1,m);
w4=zeros(nstep4+1,m); w5=zeros(nstep5+1,m);
w6=zeros(nstep6+1,m); w7=zeros(nstep6+1,m);
w1(1,1:m)=w0; w2(1,1:m)=w0; w3(1,1:m)=w0; w4(1,1:m)=w0;
w5(1,1:m)=w0; w6(1,1:m)=w0; w7(1,1:m)=w0;
%
p=4;
d=[0,   1/2, 1/2, 1  ];
c=[0,   0,   0,   0  ;
   1/2, 0,   0,   0  ;
   0,   1/2, 0,   0  ;
   0,   0,   1,   0  ];
b=[1/6, 1/3, 1/3, 1/6];
k1=zeros(p,m); k2=zeros(p,m); k3=zeros(p,m);
k4=zeros(p,m); k5=zeros(p,m);
k6=zeros(p,m); k7=zeros(p,m);
% calculate RK4
for j=1:nstep1,
  for i=1:p,
    k1(i,1:m)=h1*f_sys(w1(j,1:m)+c(i,1:i-1)*k1(1:i-1,1:m),
t1(j)+d(i)*h1);
  end
  w1(j+1,1:m)=w1(j,1:m)+b*k1;
  t1(j+1)=t1(j)+h1;
end
for j=1:nstep2,
  for i=1:p,
    k2(i,1:m)=h2*f_sys(w2(j,1:m)+c(i,1:i-1)*k2(1:i-1,1:m),
t2(j)+d(i)*h2);
  end
  w2(j+1,1:m)=w2(j,1:m)+b*k2;
  t2(j+1)=t2(j)+h2;
```

```
end
for j=1:nstep3,
  for i=1:p,
    k3(i,1:m)=h3*f_sys(w3(j,1:m)+c(i,1:i-1)*k3(1:i-1,1:m),
t3(j)+d(i)*h3);
  end
  w3(j+1,1:m)=w3(j,1:m)+b*k3;
  t3(j+1)=t3(j)+h3;
end
for j=1:nstep4,
  for i=1:p,
    k4(i,1:m)=h4*f_sys(w4(j,1:m)+c(i,1:i-1)*k4(1:i-1,1:m),
t4(j)+d(i)*h4);
  end
  w4(j+1,1:m)=w4(j,1:m)+b*k4;
  t4(j+1)=t4(j)+h4;
end
for j=1:nstep5,
  for i=1:p,
    k5(i,1:m)=h5*f_sys(w5(j,1:m)+c(i,1:i-1)*k5(1:i-1,1:m),
t5(j)+d(i)*h5);
  end
  w5(j+1,1:m)=w5(j,1:m)+b*k5;
  t5(j+1)=t5(j)+h5;
end
for j=1:nstep6,
  for i=1:p,
    k6(i,1:m)=h6*f_sys(w6(j,1:m)+c(i,1:i-1)*k6(1:i-1,1:m),
t6(j)+d(i)*h6);
  end
  w6(j+1,1:m)=w6(j,1:m)+b*k6;
  t6(j+1)=t6(j)+h6;
end
for j=1:nstep7,
  for i=1:p,
    k7(i,1:m)=h7*f_sys(w7(j,1:m)+c(i,1:i-1)*k7(1:i-1,1:m),
t7(j)+d(i)*h7);
  end
  w7(j+1,1:m)=w7(j,1:m)+b*k7;
  t7(j+1)=t7(j)+h7;
end

save data_sRK4
```

“calc_sRK4_err.m”

```
% This code estimates the error of the fourth order
% numerical differentiation method and plots it as
% a function of the spatial step.  As well, the order
% of accuracy is estimated at t=25 and plotted.
%
clear
figure(3)
clf reset

load data_sRK4
%
err1=abs(w1(nstep1+1,1)-w2(nstep1+1,1))/(1-0.5^4)+1.0e-16;
err2=abs(w2(nstep2+1,1)-w3(nstep2+1,1))/(1-0.5^4)+1.0e-16;
```

```
err3=abs(w3(nstep3+1,1)-w4(nstep3+1,1))/(1-0.5^4)+1.0e-16;
err4=abs(w4(nstep4+1,1)-w5(nstep4+1,1))/(1-0.5^4)+1.0e-16;
err5=abs(w5(nstep5+1,1)-w6(nstep5+1,1))/(1-0.5^4)+1.0e-16;
err6=abs(w6(nstep6+1,1)-w7(nstep6+1,1))/(1-0.5^4)+1.0e-16;
%
axes('position',[0.18,0.56,0.74,0.36])
loglog(h1, err1,'kv','markerfacecolor','k')
hold on
loglog(h2, err2,'cp','Markerfacecolor','c')
hold on
loglog(h3, err3,'bo','Markerfacecolor','b')
hold on
loglog(h4, err4,'g*','Markerfacecolor','g')
hold on
loglog(h5, err5,'mo','Markerfacecolor','m')
hold on
loglog(h6, err6,'rs','Markerfacecolor','r')
hold on
%
%axis([10*e-3,10,10*e(0.01),10*e(0.02)])
set(gca,'fontsize',12)
%set(gca,'xtick',10.^[-5:-1])
%set(gca,'ytick',10.^[-13:2:-3])
%xlabel('Spatial step size, h')
ylabel('Estimated error, E_{t=25}(h)')
title('Fourth order method error at t=25')
legend('h=0.2','h=0.2*2^{-1}','h=0.2*2^{-2}','h=0.2*2^{-
3}','h=0.2*2^{-4}','h=0.2*2^{-5}')
% estimate order of accuracy
p11=abs(w1(nstep1-1,1)-w1(nstep1,1));
p12=abs(w1(nstep1,1)-w1(nstep1+1,1));
p1=log2(p11/p12);
p21=abs(w2(nstep2-1,1)-w2(nstep2,1));
p22=abs(w2(nstep2,1)-w2(nstep2+1,1));
p2=log2(p21/p22);
p31=abs(w3(nstep3-1,1)-w3(nstep3,1));
p32=abs(w3(nstep3,1)-w3(nstep3+1,1));
p3=log2(p31/p32);
p41=abs(w4(nstep4-1,1)-w4(nstep4,1));
p42=abs(w4(nstep4,1)-w4(nstep4+1,1));
p4=log2(p41/p42);
p51=abs(w5(nstep5-1,1)-w5(nstep5,1));
p52=abs(w5(nstep5,1)-w5(nstep5+1,1));
p5=log2(p51/p52);
p61=abs(w6(nstep6-1,1)-w6(nstep6,1));
p62=abs(w6(nstep6,1)-w6(nstep6+1,1));
p6=log2(p61/p62);
p71=abs(w7(nstep7-1,1)-w7(nstep7,1));
p72=abs(w7(nstep7,1)-w7(nstep7+1,1));
p7=log2(p71/p72);
%
axes('position',[0.18,0.09,0.74,0.36])
semilogx(h1,p1,'kv','markerfacecolor','k')
hold on
semilogx(h2,p2,'cp','Markerfacecolor','c')
hold on
semilogx(h3,p3,'bo','Markerfacecolor','b')
hold on
semilogx(h4,p4,'g*','Markerfacecolor','g')
```

```
hold on
semilogx(h5,p5,'mo','Markerfacecolor','m')
hold on
semilogx(h6,p6,'rs','Markerfacecolor','r')
hold on
semilogx(h7,p7,'yd','Markerfacecolor','y')
set(gca,'fontsize',12)
xlabel('Spatial step size, h')
ylabel('Order of accuracy, p_{t=25}')
title('Estimated Order of Accuracy')
legend('h=0.2','h=0.2*2^{-1}','h=0.2*2^{-2}','h=0.2*2^{-
3}','h=0.2*2^{-4}','h=0.2*2^{-5}','h=0.2*2^{-6}')
%
disp(['p = [',num2str(p1),']'])
disp(['p = [',num2str(p2),']'])
disp(['p = [',num2str(p3),']'])
disp(['p = [',num2str(p4),']'])
disp(['p = [',num2str(p5),']'])
disp(['p = [',num2str(p6),']'])
disp(['p = [',num2str(p7),']'])
```

3. "f_sys.m"

```
function [z]=f_sys(w,t)
% This function calculates f_sys(w,t)
%
z=zeros(1,2);
theta=w(1);
v=w(2);
z(1)=v;
z(2)=(1-w(1)^2)*w(2)-w(1);
```

"calc_sRK4.a.m"

```
% This code implements the classical four stage fourth order
% Runge Kutta method to solve an ODE system. After the
% calculation,it saves the workspace to a data file.
%
clear
%
m=2;
w0a=[0.25, 0];
h=0.05;
nstep=25/h;
%
wa=zeros(nstep+1,m);
ta=zeros(nstep+1,1);
ta(1)=0;
wa(1,1:m)=w0a;
%
p=4;
d=[0,   1/2, 1/2, 1  ];
c=[0,   0,   0,   0  ;
   1/2, 0,   0,   0  ;
   0,   1/2, 0,   0  ;
   0,   0,   1,   0  ];
b=[1/6, 1/3, 1/3, 1/6];
k=zeros(p,m);
%
```

```
for j=1:nstep,
  for i=1:p,
    k(i,1:m)=h*f_sys(wa(j,1:m)+c(i,1:i-1)*k(1:i-1,1:m),
ta(j)+d(i)*h);
  end
  wa(j+1,1:m)=wa(j,1:m)+b*k;
  ta(j+1)=ta(j)+h;
end
%
save data_sRK4a
```

“calc_sRK4.b.m”

```
% This code implements the classical four stage fourth order
% Runge Kutta method to solve an ODE system. After the
% calculation, it saves the workspace to a data file.
%
clear
%
m=2;
w0b=[-2, 4];
h=0.05;
nstep=25/h;
%
wb=zeros(nstep+1,m);
tb=zeros(nstep+1,1);
tb(1)=0;
wb(1,1:m)=w0b;
%
p=4;
d=[0,   1/2, 1/2, 1  ];
c=[0,   0,   0,   0  ;
   1/2, 0,   0,   0  ;
   0,   1/2, 0,   0  ;
   0,   0,   1,   0  ];
b=[1/6, 1/3, 1/3, 1/6];
k=zeros(p,m);
%
for j=1:nstep,
  for i=1:p,
    k(i,1:m)=h*f_sys(wb(j,1:m)+c(i,1:i-1)*k(1:i-1,1:m),
tb(j)+d(i)*h);
  end
  wb(j+1,1:m)=wb(j,1:m)+b*k;
  tb(j+1)=tb(j)+h;
end
%
save data_sRK4b
```

“plot_sRK4.m”

```
% This code reads in the data file generated by calc_sRK4.m. Then
% it plots the two components of the numerical solution: position
% and velocity.
%
clear
figure(4);
clf reset
axes('position',[0.15,0.15,0.75,0.75])
%
```

```
load data_sRK4a
load data_sRK4b
%
plot(wa(1:nstep+1,1),wa(1:nstep+1,2),'k-','linewidth',1.0)
hold on
plot(wb(1:nstep+1,1),wb(1:nstep+1,2),'b--','linewidth',1.0)
set(gca,'fontsize',12)
%axis([0,40,-1.1,1.1])
%set(gca,'xtick',[0:10:40])
%set(gca,'ytick',[-1:0.5:1])
xlabel('y(t)')
ylabel('y`(t)')
title('Numerical solution by RK4, W`(t) = [_{y``(t)}^{y`(t)}]')
h1=legend('W(t) = [_{y`(0)=0}^{y(0)=0.25}]','W(t) =
[_{y`(0)=4}^{y(0)=-2}]');
set(h1,'fontsize',12)
```