Homework 8

Yash Ganatra

May 1, 2016

Problem 0: Homework checklist

- Collaborators: Zhehuan Xu, Nate Majid Kesto, Emily Zimovan,Woi Sok Oh
- Coding Language: Matlab

Problem 1: (Ascher and Greif, Problem 16.5)

Consider the ODE
$$\frac{d\mathbf{f}(y,t)}{dt} = \mathbf{f}(t,\mathbf{y}), 0 \leq t \leq b$$
with $b \gg 1$.
$$\begin{aligned} t &= \tau\, b \\ d\,t &= b\, d\,\tau \\ \frac{d\,y}{d\,\tau} &= b\,\mathbf{f}(\tau\, b, y) \end{aligned} \tag{1}$$
From Equation 1,y=z To prove that the transformation when applied to Forward Euler satisfies $h_t = b\, h_\tau$, consider Equation2
$$\begin{aligned} &\text{Forward Euler method in } \tau \\ &y_{i+1} = y_i + h_\tau\, b\mathbf{f}(\tau\, b, y_i) \\ &\implies y_{i+1} = y_i + \delta\, t\mathbf{f}(\tau\, b, y_i) \\ &\implies \frac{y_{i+1} - y_i}{\Delta\, t} = \mathbf{f}(\tau\, b, y_i) = \mathbf{f}(t, y_i) \end{aligned} \tag{2}$$

Problem 2: (Ascher and Greif, Example 16.20)

Codes in Appendix 1-3 Converting given DE to a system of ODE by the transformations to give Equation 3
$$\begin{aligned} x_1 &= y_1,\ x_2 = y_1\prime \\ x_3 &= y_2,\ x_4 = y_2\prime \end{aligned}$$

System of equations are

$$
\begin{aligned}
x_1\prime &= x_2 \\
x_2\prime &= x_1 + 2\,x_4 - \hat{\mu}\frac{u_1+\mu}{D_1} - \mu\frac{u_1-\hat{\mu}}{D_2} \\
x_3\prime &= x_4 \\
x_4\prime &= x_3 - 2\,x_2 - \hat{\mu}\frac{u_2}{D_1} - \mu\frac{u_2}{D_2}
\end{aligned}
\tag{3}
$$

This gives a system of 4 ODE's with 4 initial conditions given as

$$
\begin{aligned}
x_1(0) = 0.994,\ x_2(0) = 0 \\
x_3(0) = 0,\ x_4(0) = -2.001\ldots
\end{aligned}
$$

Figures 1-5 shows the plots of $y_1(t)\,\mathrm{vs}\,y_2(t)$ compared with *MATLAB ODE45* solver

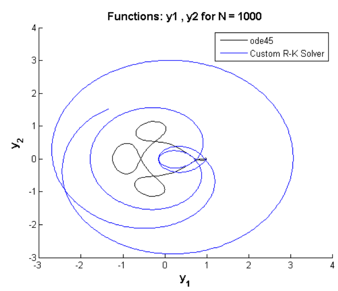

Figure 1: N=1e3

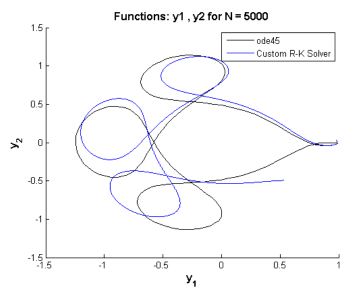

Figure 2: N=5e3

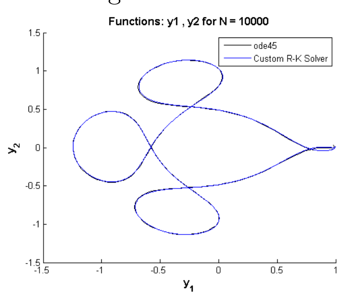

Figure 3: N=1e4

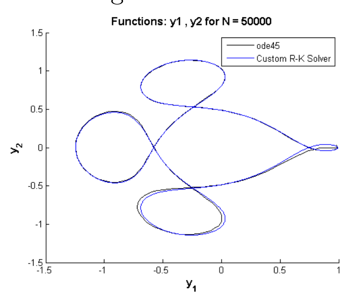

Figure 4: N=5e4

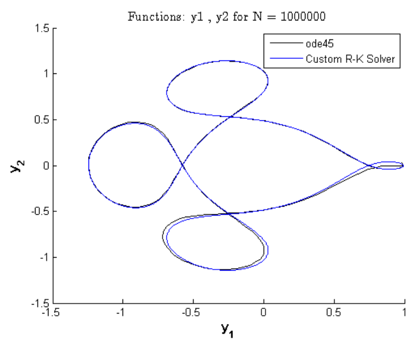

Figure 5: N=1e6

Comments

- Qualitatively sensible results are obtained for $N = 1e4$ for the Runge-Kutta method which is not adaptive.
- MATLAB's ode45 routine - which is adaptive solves the system of equations in $N = 309$ steps. **Having a method that takes local errors for predicting next time step saves computational effort**

Problem 3: (Gautschi Machine Exercise 5.5)

Codes in Appendix 4-8 *Verification of analytical solution*

$$\begin{aligned}
&x(t) = \cos u(t) - \epsilon, \\
&x\prime = \frac{-sin\, u}{1 - \epsilon\, cos\, u} \\
&y\prime = \frac{\sqrt{1-\epsilon^2}\, cos\, u}{1 - \epsilon\, cos\, u} \\
&u\prime = \frac{1}{1 - \epsilon\, cos\, u} \\
&x\prime\prime = \frac{\epsilon - cos\, u}{(1 - \epsilon\, cos\, u)^3} \\
&y\prime\prime = \frac{-\sqrt{1-\epsilon^2}\, sin\, u}{(1 - \epsilon\, cos\, u)^3} \\
&\text{Substituting} \\
&x(t) = \cos u(t) - \epsilon, \\
&y(t) = \sqrt{1-\epsilon^2} \sin u(t)
\end{aligned} \tag{4}$$

Comparing LHS and RHS, the verification is complete Similar to Equation 3, the given DE is converted to a system of ODE's as

$$u_1 = x,\ u_2 = x' \\ u_3 = y,\ x_4 = y'$$

System of equations are shown in Equation 5

$$\begin{aligned} u_1' &= u_2 \\ u_2' &= \frac{-u_1}{r^3} \\ u_3' &= u_4 \\ u_4' &= \frac{-u_3}{r^3} \end{aligned} \tag{5}$$

This gives a system of 4 ODE's with 4 initial conditions given as

$$u_1(0) = 1-\epsilon,\ u_2(0) = 0 \\ u_3(0) = 0,\ u_4(0) = \sqrt{\frac{1+\epsilon}{1-\epsilon}}$$

Figures 6-10 for $\epsilon = 0.3$

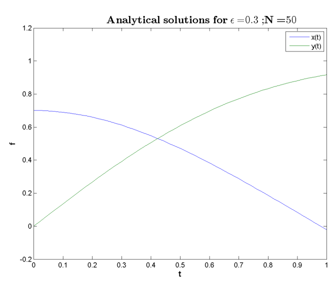

Figure 6: N=50

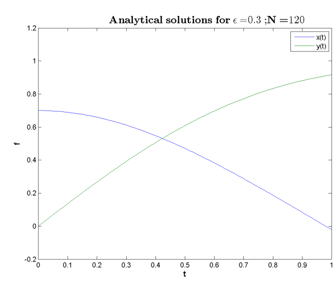

Figure 7: N=120

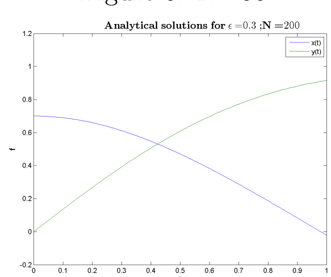

Figure 8: N=200

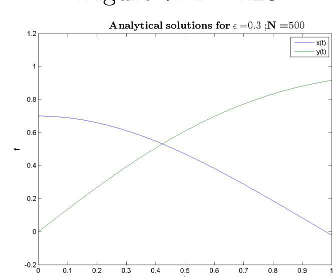

Figure 9: N=500

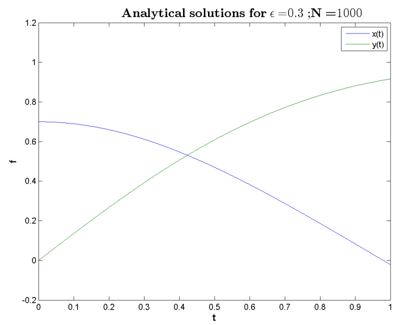

Figure 10: N=1e3

Figures 11-15 for $\epsilon = 0.5$

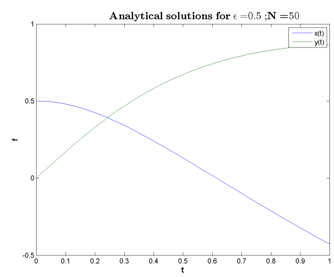

Figure 11: N=50

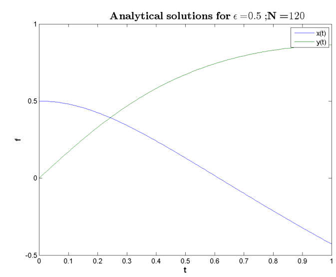

Figure 12: N=120

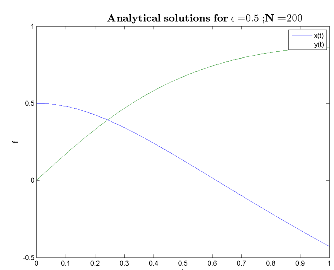

Figure 13: N=200

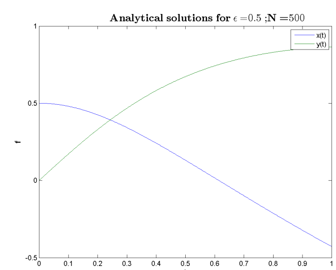

Figure 14: N=500

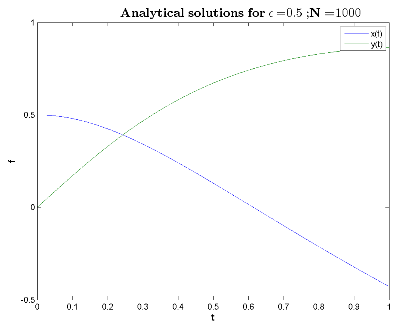

Figure 15: N=1e3

Figures 16-20 for $\epsilon = 0.7$

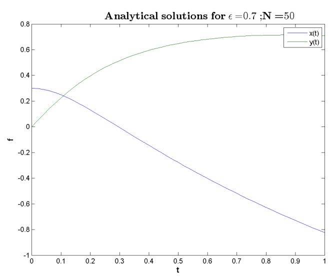

Figure 16: N=50

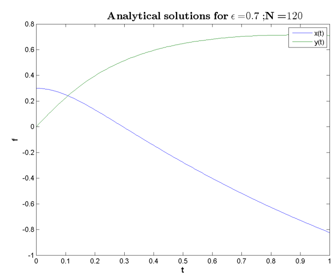

Figure 17: N=120

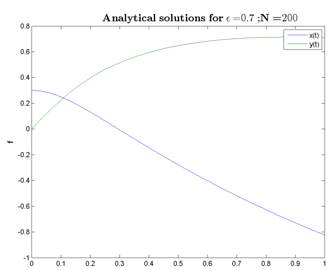

Figure 18: N=200

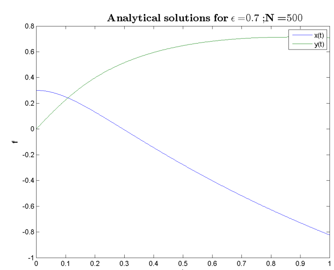

Figure 19: N=500

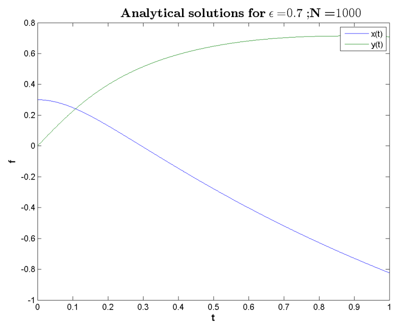

Figure 20: N=1e3

There is minor variation in N for a fixed ϵ. Extending the range to 20 (as given in Gautschi), the differences can be observed. Figures 21-25 show the plots for extended time range and it can be seen that there is a *singularity at* $t \approx 9$

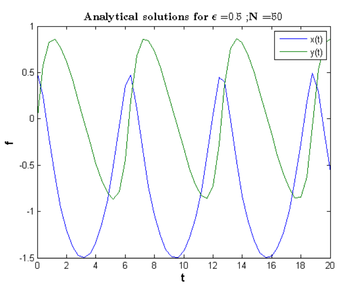

Figure 21: N=50

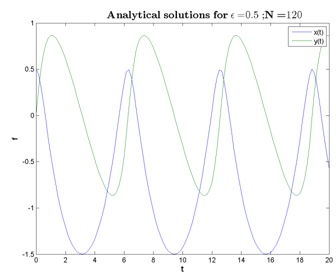

Figure 22: N=120

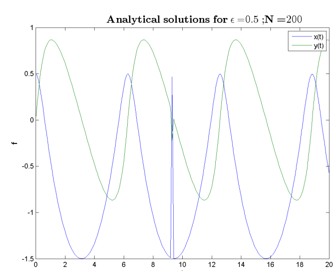

Figure 23: N=200

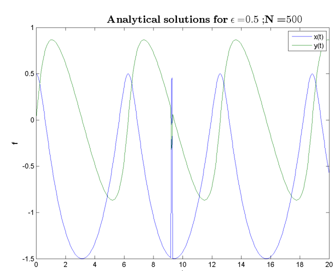

Figure 24: N=500

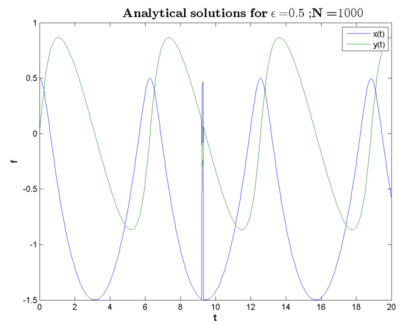

Figure 25: N=1e3

Forward Euler Error Plot

Figure 26 shows the error plot (computed as ∞ norm of absolute error)

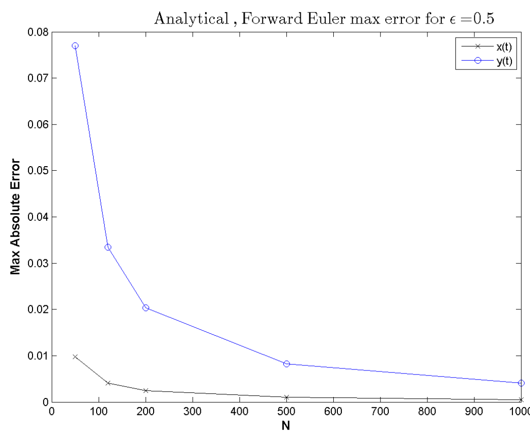

Figure 26: Error plot between Forward Euler and analytical solution for $\epsilon =$ 0.5

Backward Euler Error Plot

Figure 27 shows the error plot (computed as ∞ norm of absolute error)

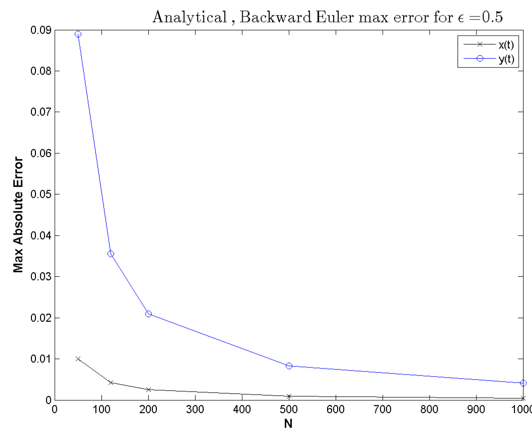

Figure 27: Error plot between Backward Euler and analytical solution for $\epsilon = 0.5$

Comments
The error decreases as expected as N increases

Appendix

Listing 1: Runge-Kutta solver - Main

```
clc;close all;
clear all;
y0 = [0.994,0,0,-2.00158510637908252240537862224]'; %
    initial data
% y0 = [80,30]'; % initial data
% tspan = [0,100]; % integration interval
T = 17.1;%Periodicity
tspan = [0,T]; % integration interval
N =  1e5; % # of steps
%%
%[tout,yout] = rk4_test(@p2_func_test,tspan,y0,N);
[t_fun,y_fun] = rk4_test(@p2_func_test,tspan,y0,N);
%%
[t_m,y_m] = ode45(@func_45,tspan,y0);
figure(3)
plot(y_m(:,1),y_m(:,3),'k-')
%%
% figure(1)
% plot(t_fun,y_fun)
% xlabel('t')
% ylabel('y')
% legend('y_1','y_1_p','y_2','y_2_p')
% figure(2)
% hold on
% plot(y_m(:,1),y_m(:,3),'k-')
% plot(y_fun(1,:),y_fun(3,:),'b-')
% xlabel('y_1','FontSize',12,'FontWeight','bold')
% ylabel('y_2','FontSize',12,'FontWeight','bold')
% legend('ode45','Custom R-K Solver');
% title_name = ( ['Functions: y1 , y2 for N = ' num2str(N
    )]);
% title(title_name,'FontSize',12,'FontWeight','bold')
```

Listing 2: Runge-Kutta solver - function

```
function [t,y] = rk4_test(f,tspan,y0,N)
%%
y0 = y0(:); % make sure y0 is a column vector
m = length(y0); % problem size
%t = tspan(1):h:tspan(2); % output abscissae
%N = length(t)-1; % number of steps
t = linspace(tspan(1),tspan(2),N+1);
h = abs(tspan(2)-tspan(1))/N;
```

```
y = zeros(m,N+1);
%%
y(:,1) = y0; % initialize
% Integrate
for i=1:N
% Calculate the four stages
K1 = feval(f, t(i),y(:,i))';
K2 = feval(f, t(i)+0.5*h, y(:,i)+0.5*h*K1)';
K3 = feval(f, t(i)+.5.*h, y(:,i)+.5.*h.*K2)';
K4 = feval(f, t(i)+h, y(:,i)+h.*K3 )';
% Evaluate approximate solution at next step
y(:,i+1) = y(:,i) + h/6 .*(K1+2*K2+2*K3+K4);
end
```

Listing 3: Runge-Kutta solver - IVP

```
function [f] = p2_func_test(t,y)
mu = 0.012277471; mu_h = 1-mu;
D1 = power((y(1)+mu).^2+y(3).^2,3/2);
D2 = power((y(1)-mu_h).^2+y(3).^2,3/2);
A = 1-(mu_h./D1)-(mu/D2);
C = mu*mu_h*((1./D1)-(1./D2));
f(1) = y(2);
f(2) = A*y(1)+2*y(4)-C;
f(3) = y(4);
f(4) = A*y(3)-2*y(2);
end
```

Listing 4: Euler methods - main

```
clc;close all;
clear all;
%%
tspan = [0 1]; % Interval for solution
N = 20; % # of steps
my_N = [50,120,200,5e2,1e3]; % # of steps
epsilon = 0.3;
%% y0 -> Column Vector of initial conditions
y0 = [1-epsilon,0,0,sqrt((1+epsilon)./(1-epsilon))];
%% Newtons Method setup
error_rel = 1e-5;
error_abs = 1e-5;
%% R-K Code
%[t_rk,y_rk] = rk4_test(@p3_func,tspan,y0,N);
%% Forward Euler
%[t_fe,y_fe] = forw_euler(@p3_func,tspan,y0,N);
%% Backward Euler
```

```
[t_be,y_be] = back_euler(@p3_func,tspan,y0,N,epsilon,
    error_rel,error_abs);
%% From Matlab
[t_m,y_m] = ode45(@p3_func_45,tspan,y0);
figure(2)
plot(t_m,y_m(:,1),t_m,y_m(:,3))
legend('x_ode45','y_ode_45');
%%
figure(1)
hold on
plot(t_be,y_be(1,:),'b-',t_be,y_be(3,:),'b--')
plot(t_m,y_m(:,1),'k-',t_m,y_m(:,3),'k--')
xlabel('t')
ylabel('f')
legend('x(t)','y(t)')
title_name = ( ['x(t),y(t) for $\epsilon$ =' num2str(
    epsilon) ' ;N =  ' num2str(N)]);
title(title_name,'FontSize',12,'FontWeight','bold','
    interpreter','Latex')

%% Matlab + Custom Function plot
% figure(2)
% hold on
% plot(y_m(:,1),y_m(:,3),'k-')
% plot(y_fe(1,:),y_fe(3,:),'b-')
% xlabel('x(t)','FontSize',12,'FontWeight','bold')
% ylabel('y(t)','FontSize',12,'FontWeight','bold')
% legend('ode45','Custom R-K Solver');
% title_name = ( ['ode45, custom Forward Euler function N
    = ' num2str(N)]);
% title(title_name,'FontSize',12,'FontWeight','bold')
```

Listing 5: Euler methods - Forward Euler

```
function [t,y] = forw_euler(f,tspan,y0,N)
%%
y0 = y0(:); % make sure y0 is a column vector
m = length(y0); % problem size
%t = tspan(1):h:tspan(2); % output abscissae
%N = length(t)-1; % number of steps
t = linspace(tspan(1),tspan(2),N+1);
h = abs(tspan(2)-tspan(1))/N;
y = zeros(m,N+1);
%%
y(:,1) = y0; % initialize
% Integrate
```

```
for i=1:N
% Calculate the four stages
K1 = feval(f,y(:,i))';
% Evaluate approximate solution at next step
y(:,i+1) = y(:,i) + h.*K1;
end
```

Listing 6: Newtons method for non-linear equations

```
function [xsol,count,error_new] = p3_newton_func(f,f_der,
    x_0,error_rel,error_abs,...
     N,epsilon)
error = 1;
tol = 0.1;
count =0;
maxiter = 1e2;
error_new = zeros([1,maxiter]);
x = zeros([1,maxiter]);
while(error > tol)
     count = count+1;
     if count==1
     x(count) = x_0 - f(x_0)./f_der(x_0);
     error = abs(x(count)-x_0);
     else
     x(count) = x(count-1) - f(x(count-1))./f_der(x(count
         -1));
     error = abs(x(count)-x(count-1));
     end
     tol =  abs(x(count)).*error_rel + error_abs;
     if count > maxiter
          fprintf('No Conver Reached for e:%d,N:%d\n',
              epsilon,N)
          break
     end
error_new(count)= error;
end
xsol = x(count);
end
```

Listing 7: Euler methods - Backward Euler

```
function [t,y] = back_euler(fun_be,tspan,y0,N,eps,
    error_rel,error_abs)
%%
y0 = y0(:); % y0 is a column vector
m = length(y0); % problem size
%t = tspan(1):h:tspan(2); % output abscissae
```

```
%N = length(t)-1; % number of steps
t = linspace(tspan(1),tspan(2),N+1);
h = abs(tspan(2)-tspan(1))/N;
y = zeros(m,N+1);
%%
y(:,1) = y0; % initialize
% % Integrate
for i=1:N
% Calculate the four stages
K1 = feval(fun_be,y(:,i))';
% Evaluate approximate solution at next step
y_guess = y(:,i) + h.*K1;
%y(:,i+1)-f(y(:,i+1))h - y(:,i) =0;
q = @(v) v' - h.*(fun_be(v)) - y(:,i)';
[x,c_new,error_new] = p3_3_newton_func(q,y_guess,
    error_rel,error_abs,...
     N,eps);
y(:,i+1) = x;
end
```

Listing 8: Euler methods - Backward Euler

```
function [f] = p3_func(u)
r = sqrt(u(1).^2+u(3).^2);
f(1) = u(2);
f(2) = -u(1)./r.^3;
f(3) = u(4);
f(4) = -u(3)./r.^3;
end
```