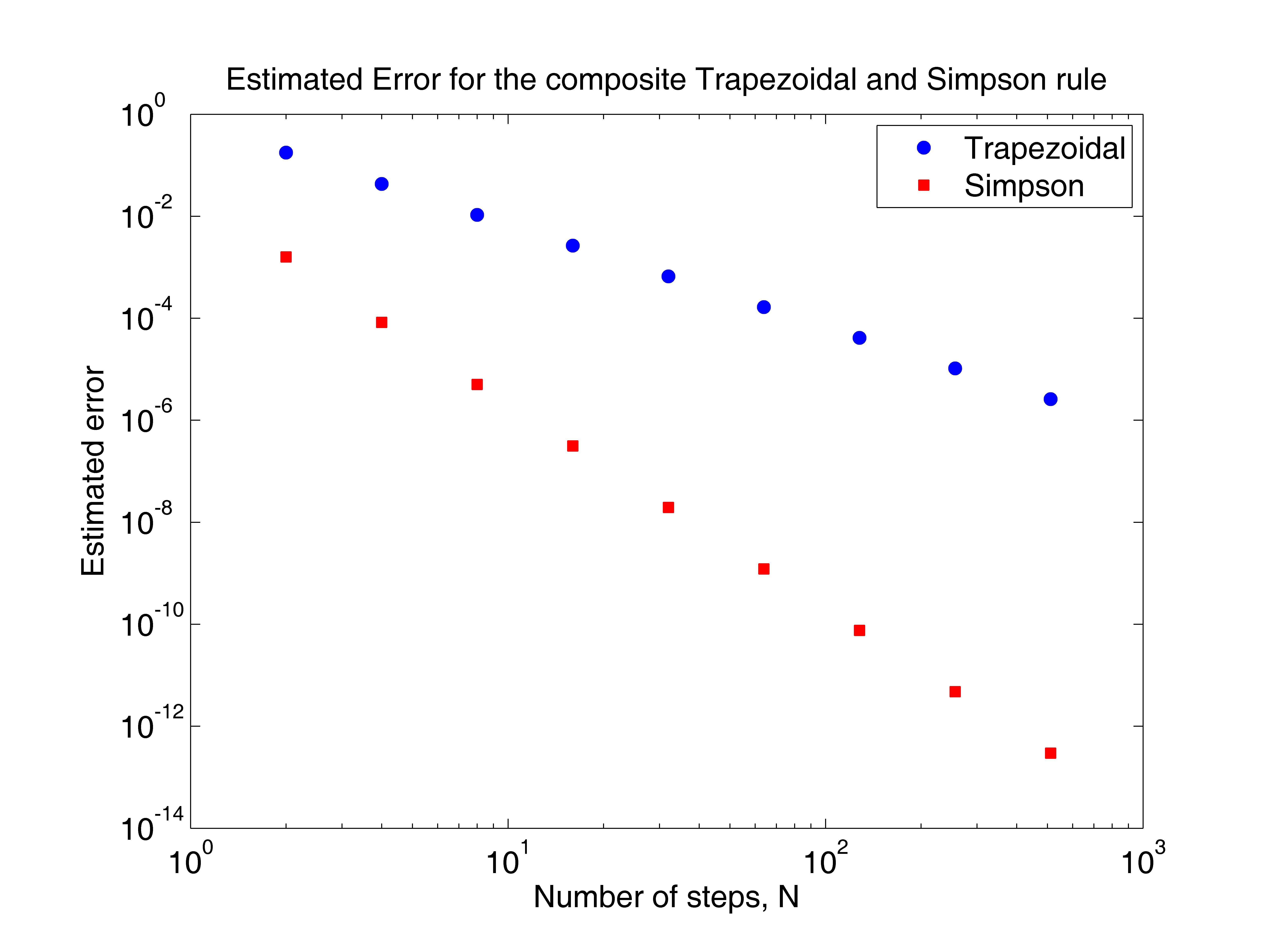
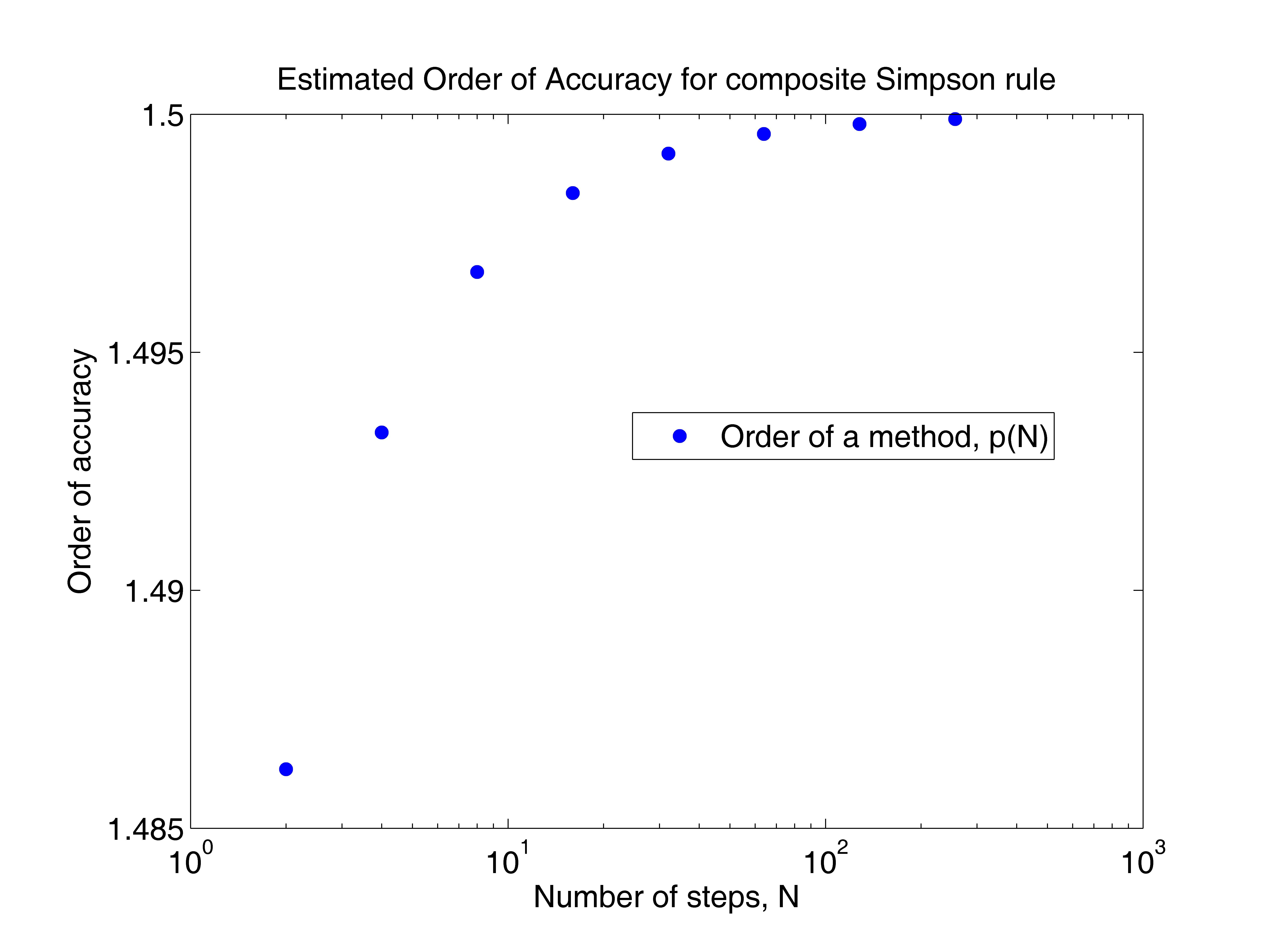
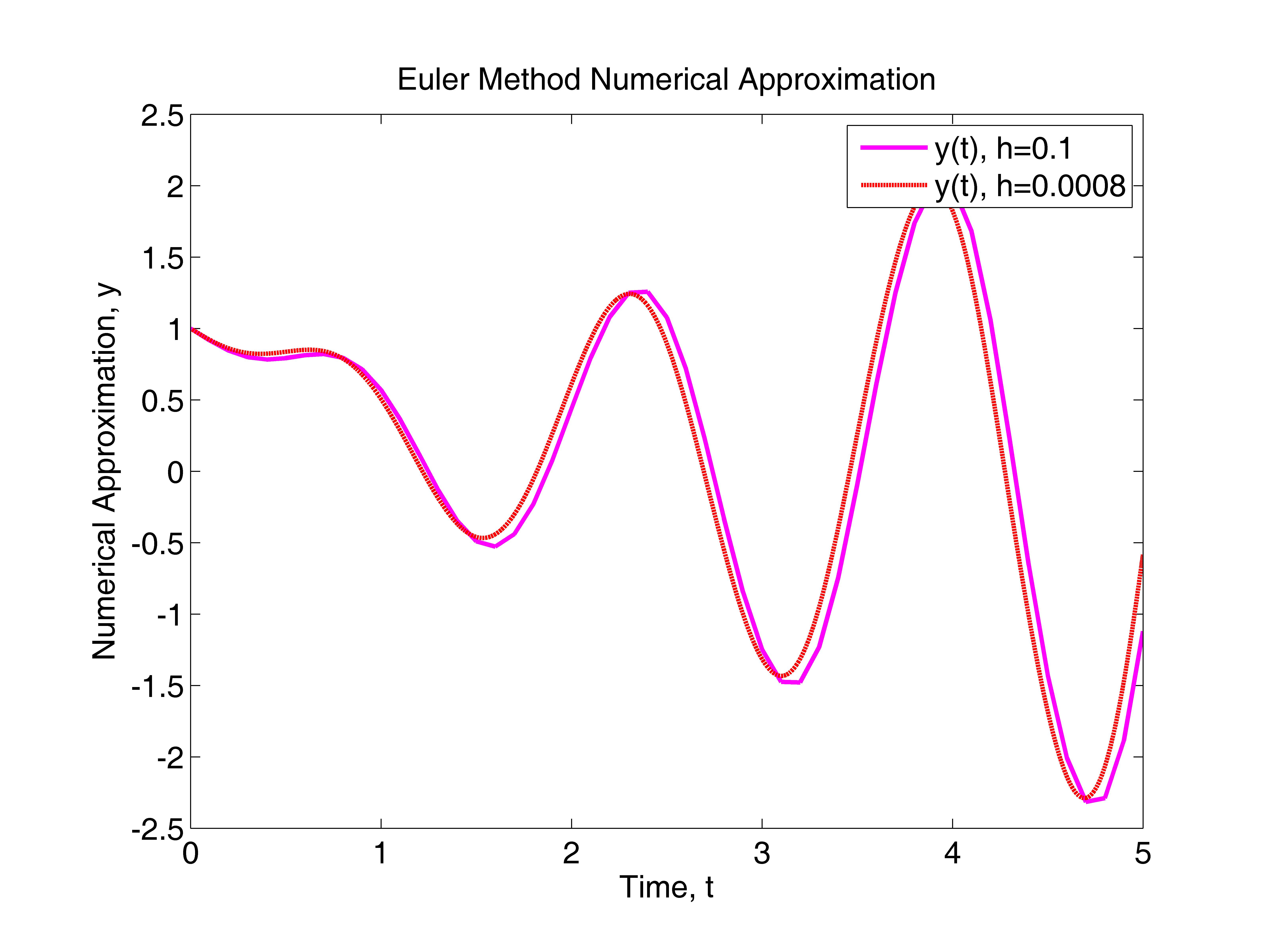
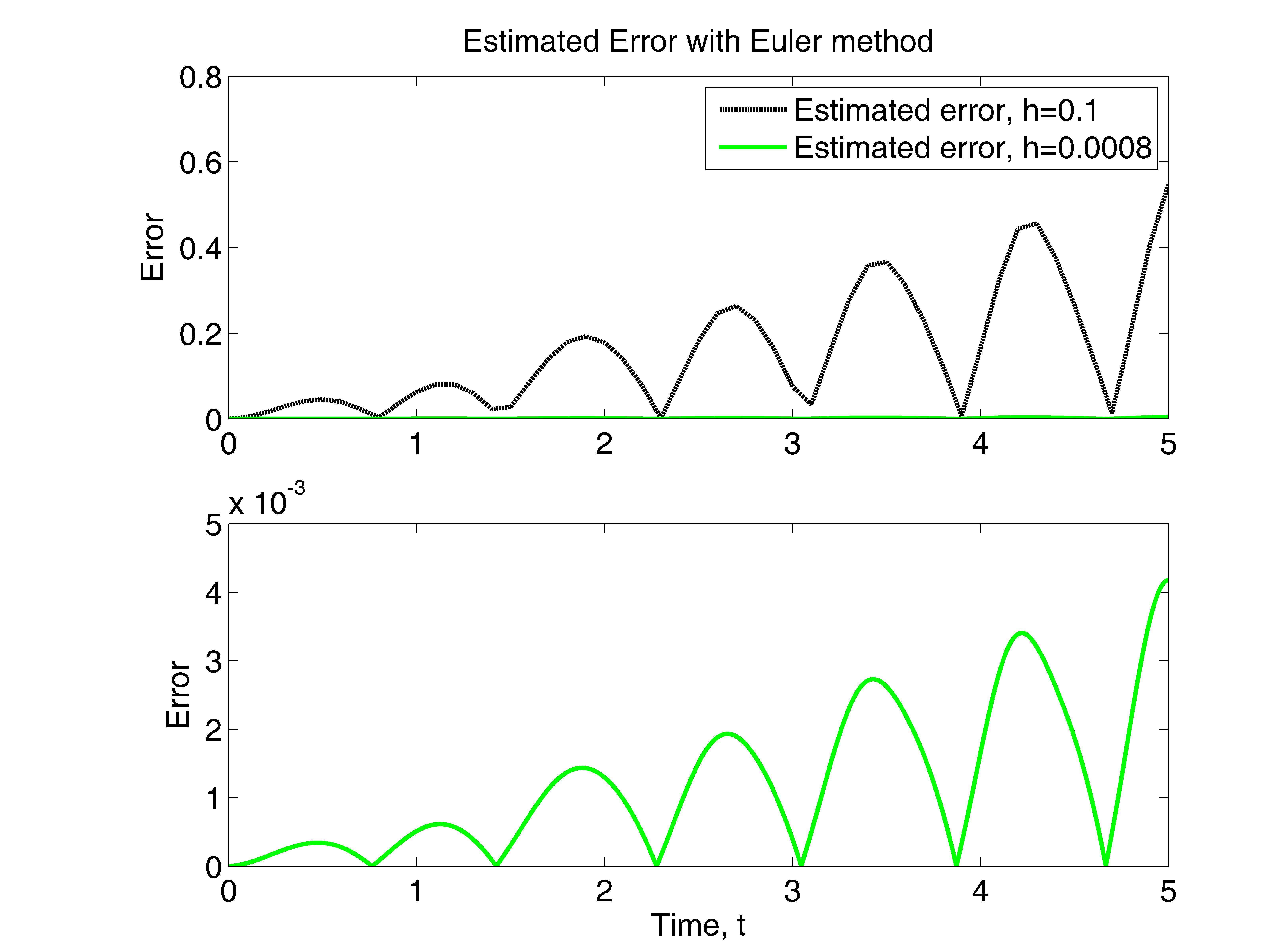
Derek Frank

dmfrank@ucsc.edu

AMS 147

2/17/10

Homework #5

* 1. The first problem I am to solve is to estimate the error caused by the numerical estimation of *20 esinxx* using both the composite Trapezoidal rule and the composite Simpson rule. The spatial step size, *h*, depends on *N=2[1:1:10]*.
  2. To solve this problem, I implement both methods in Matlab as functions to determine the numerical estimation of each rule. The rules are treated as functions so that I can vary the value of *h* and call on them multiple times. Now I implement the numerical estimation, *E(h)=(T(h)-T(h/2))/(1-(1/2)p)*, in Matlab. I find the numerical values for each *N* and *h* and solve for the error. The composite Trapezoidal rule is a second order method, while the composite Simpson rule is a fourth order method.
  3. Both the composite Simpson rule, the fourth order method, and the composite Trapezoidal rule, the second order method, appear linear on the log-scale. Nevertheless, the Simpson rule attains more accuracy than the Trapezoidal rule on this plot. Both methods become more accurate as N gets larger, however, the Simpson rule becomes accurate more quickly than the Trapezoidal rule.
  4. In this problem I am to estimate the order of accuracy of the numerical estimation of *20 e-xx* using the composite Simpson rule. The spatial step size, *h*, depends on *N=2[1:1:10]*.
  5. Using Matlab, I first implement the Simpson rule, then the estimation of the order of accuracy. The order of accuracy, *p*, can be estimated by calculating *p=log2[(T(h)-T(h/2))/ (T(h/2)-T(h/4))]*.
  6. The results show the order of accuracy, *p*, approaching 1.5 as *N* gets bigger.
  7. In the final problem, I am to solve *y`=-sin(y)+2(t)sin(4t)* for *t=[0,5]* given *y(0)=1*.
  8. Using Matlab, I implement Euler’s method, *yn+1=yn+hF(yn,tn)*, to solve for *y(t)* on the interval for *t=0* to *t=5* with a time step *h=0.1*. I then implement the error estimation on Eulers method by solving for *y(t)*, with *h* and *h/2*, and calculate *E(h)=(T(h)-T(h/2))/(1-(1/2))*. I can now plug in any value of *h* to plot the error.
  9. 
  10. The results show, in particular, that the more time steps that occur, the more accurate the numerical approximation. After noticing the error approximated, with *h=0.1* and *t=5*, was *err(5)=.546756721*, I quickly found that either *h=.0008* or *h=.0005* produce an *error<0.005*. I used *h=0.0008* to attain the *err(5)=0.00417846983* and indeed the error is less than 0.005.

Appendix:

1. “err\_est.m”

% This code estimates the error of the numerical integration

% method and plots it as a function of the number of steps, N,

% per spatial step, h.

%

clear

figure(1)

clf reset

axes('position',[0.15,0.13,0.75,0.75])

%

a=0.0; b=2.0;

N=2.^([1:1:10]);

Nsize=10;

h=(b-a)./N;

%

T=zeros(Nsize); S=zeros(Nsize);

for i=1:Nsize,

[T(i)]=trap\_num\_est(h(i),N(i));

[S(i)]=simp\_num\_est(h(i),N(i));

end

% second order

errT=abs(T(1:Nsize-1)-T(2:Nsize))/(1-0.5^2)+1.0e-16;

% fourth order

errS=abs(S(1:Nsize-1)-S(2:Nsize))/(1-0.5^4)+1.0e-16;

%

loglog(N(1:Nsize-1), errT,'bo', 'markerfacecolor', 'b')

hold on

loglog(N(1:Nsize-1), errS,'rs', 'markerfacecolor', 'r')

hold on

%

set(gca,'fontsize',14)

xlabel('Number of steps, N')

ylabel('Estimated error')

title('Estimated Error for the composite Trapezoidal and Simpson rule')

legend('Trapezoidal', 'Simpson')

“trap\_est.m”

function [T]=trap\_num\_est(h,N)

% This code uses the composite Trapezoidal rule to calculate

% int\_{a}^{b} f(x) dx.

%

a=0.0; b=2.0;

nsize=10;

%

x=a+[0:N]\*h;

y=f(x);

T=(y(1)+y(N+1)+2\*sum(y(2:N)))\*h/2;

“simp\_est.m”

function [T]=simp\_num\_est(h, N)

% This code uses the composite Simpson rule to calculate

% int\_{a}^{b} f(x) dx.

%

a=0.0; b=2.0;

%

x=a+[0:N]\*h;

y=f(x);

x2=a+[0:N-1]\*h+h/2;

y2=f(x2);

T=(y(1)+y(N+1)+2\*sum(y(2:N))+4\*sum(y2))\*h/6;

“f.m”

function [y]=f(x)

% This function calculates f(x).

%

y=exp(sin(x));

1. “simp\_ord\_acc.m”

% This code determines the order of accuracy for the

% int\_a^b f(x)=exp(-sqrt(x)) dx

% using the composite Simpson rule.

%

clear

figure(2)

clf reset

axes('position',[0.15,0.13,0.75,0.75])

%

a=0.0; b=2.0;

Nsize=10;

N=2.^[1:1:Nsize];

h=(b-a)./N;

S=zeros(Nsize);

for i=1:Nsize,

[S(i)]=simp\_num\_est(h(i),N(i));

end

%

err1=abs(S(1:Nsize-1)-S(2:Nsize))+1.0e-16;

p=log2((err(1:Nsize-2))./(err(2:Nsize-1)));

%

semilogx(N(1:Nsize-2),p,'bo','markerfacecolor','b')

set(gca,'fontsize',14)

xlabel('Number of steps, N')

ylabel('Order of accuracy')

title('Estimated Order of Accuracy for composite Simpson rule')

legend('Order of a method, p(N)')

%

disp(['p = [',num2str(p),']'])

for i=1:Nsize,

disp(['S = [',num2str(S(i)),']'])

end

“simp\_num\_est.m”

function [T]=simp\_num\_est(h, N)

% This code uses the composite Simpson rule to calculate

% int\_{a}^{b} f(x) dx.

%

a=0.0; b=2.0;

%

x=a+[0:N]\*h;

y=f(x);

x2=a+[0:N-1]\*h+h/2;

y2=f(x2);

T=(y(1)+y(N+1)+2\*sum(y(2:N))+4\*sum(y2))\*h/6;

“f.m”

function [y]=f(x)

% This function calculates f(x) for a given x.

%

y=exp(-sqrt(x));

1. “run\_euler.m”

% Calculate and plot the error estimation for h=.1 and h=.0008

%

clear

figure(4)

clf reset

[y t1]=est\_err(.1);

[y2 t2]=est\_err(.0008);

%

axes('position',[0.18,0.56,0.74,0.36])

plot(t1,y,'k--','linewidth', 2.0)

hold on

plot(t2,y2,'g-','linewidth',2.0)

legend('Estimated error, h=0.1','Estimated error, h=0.0008')

set(gca,'fontsize',14)

ylabel('Error')

title('Estimated Error with Euler method')

%

axes('position',[0.18,0.09,0.74,0.36])

plot(t2,y2,'g-','linewidth',2.0)

set(gca,'fontsize',14)

xlabel('Time, t')

ylabel('Error')

“est\_err.m”

function [err\_est, t1]=est\_err(h)

% This code uses the Euler method to solve y'=-sin(y)+2\*t\*sin(4\*t)

% with time step h=0.1 and h=0.05. Then it estimates the error

% and plots the estimated error as a function of time.

%

y0=1;

%h=.1;

%

% Run with h

%

n=5/h;

t=[0:n]\*h;

y=zeros(1,n+1);

y(1)=y0;

for j=1:n,

y(j+1)=y(j)+h\*(-sin(y(j))+2\*t(j)\*sin(4\*t(j)));

end

h1=h;

n1=n;

t1=t;

y1=y;

%

% Run with h/2

%

h=h/2;

n=5/h;

t=[0:n]\*h;

y=zeros(1,n+1);

y(1)=y0;

for j=1:n,

y(j+1)=y(j)+h\*(-sin(y(j))+2\*t(j)\*sin(4\*t(j)));

end

h2=h;

n2=n;

t2=t;

y2=y;

%

% Error at t=5

err\_t5=abs(y1(n1+1)-y2(n2+1))/(1-0.5);

disp(' ')

disp([' The estimated error for h = ',num2str(h1),' at t = 5 is'])

disp([' Error = ',num2str(err\_t5,'%16.8e'),'.'])

disp(' ')

%

% Error as a function of time

err\_est=abs(y1-y2(1:2:n2+1))/(1-0.5);

“Euler.m”

% This code uses the Euler method to solve y'=-sin(y)+2\*t\*sin(4\*t)

% from t=0 to t=5. Then it plots the numerical solution

%

clear

figure(3)

clf reset

axes('position',[0.15,0.13,0.75,0.75])

%

y0=1;

h=0.1;

%

n=5/h;

t=[0:n]\*h;

y=zeros(1,n+1);

y(1)=y0;

%

for j=1:n,

y(j+1)=y(j)+h\*(-sin(y(j))+2\*t(j)\*sin(4\*t(j)));

end

% h=.0008

h2=0.0008;

%

n2=5/h2;

t2=[0:n2]\*h2;

y2=zeros(1,n2+1);

y2(1)=y0;

%

for j=1:n2,

y2(j+1)=y2(j)+h2\*(-sin(y2(j))+2\*t2(j)\*sin(4\*t2(j)));

end

%

plot(t,y,'m-','linewidth',2.0)

hold on

plot(t2,y2,'r--','linewidth',2.0)

%

set(gca,'fontsize',14)

xlabel('Time, t')

ylabel('Numerical Approximation, y')

title('Euler Method Numerical Approximation')

legend('y(t), h=0.1','y(t), h=0.0008')