Math 4NA3 - Assignment 3

February 22, 2021

Question 1:

We will approximate $\int_0^{\pi} \sin^3(x)dx$ using Romberg integration. Note that the exact value is 4/3.

Here is the Matlab code that I modified from avenue.

```
Rnum = Rombergg(@(x)(sin(x)).^3,0,pi,10,10^-12)
exact = 4/3
n = length(Rnum)
for i = 1:n
    error(i) = abs(exact - Rnum(i));
end

loglog(1:n,error)
title('Q1');
xlabel('Iterations');ylabel('Error');
grid on


function Rnum = Rombergg(f,a,b,maxRomb,tol)
% Computes recursive Romberg integrations starting with
% composite trapezoid rule (CTR)
% Input:
% f       = function to be integrated
% a, b    = lower and upper limits of integration
% maxRomb = the number of recursive Romberg integrations
%           (determines finest grid spacing as (b-a)^/2^(maxRomb-1))
% tol     = stops iterations if difference between approximations are
%            less than tol
% Output:
% Rnum    = vector of Romberg results for integral at each order

R = ones(maxRomb,maxRomb);
hmin = (b-a)/2^(maxRomb-1); % finest grid spacing
for k = 1 : maxRomb         % CTR on the coarser grids
    h = 2^(k-1)*hmin;
    x = a : h : b;
    y = feval(f,x);
    lenY = length(y);
    R(k,1) = 0.5*h*(y(1) + 2*sum(y(2:lenY-1)) + y(lenY));
end

% Romberg integrations
% (Richard extrapolation for trunction errors of CTR: h^2, h^4, h^6,
```

```
    ...)
for k = 2 : maxRomb
    for kk = 1 : (maxRomb-k+1)
        R(kk,k) = R(kk,k-1)+(R(kk,k-1) - R(kk+1,k-1))/(4^(k-1)-1);
    end
    if abs(R(1,k)-R(1,k-1))<tol % breaks out if not enough improvement
        z = maxRomb - k;
        break
    end
    z = 0;
end
Rnum = R(1,1:maxRomb-z);
end
```

From the plot, we achieve an error near 10^{-15} in 4 iterations before the improvements are less than 10^{-12}.

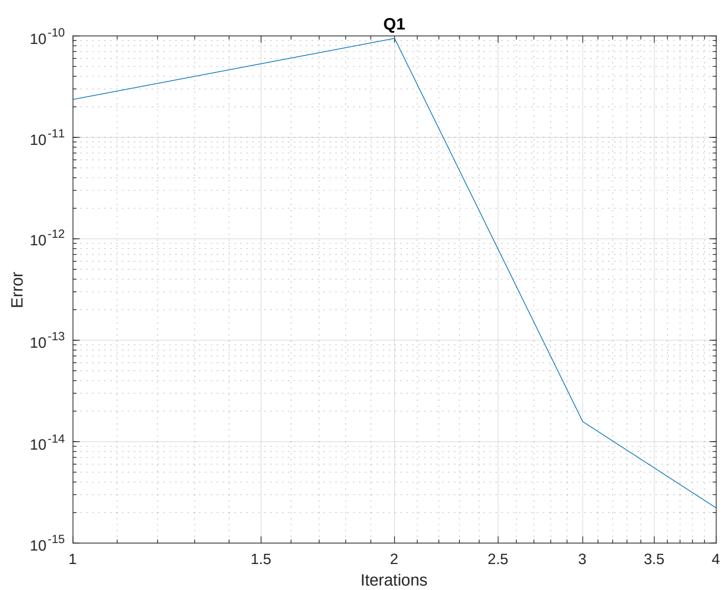

Figure 1: Absolute error per iteration

Question 2:

Here is the code I used to implement the second-order predictor-corrector on this particular Lotka-Volterra system. After the code, I show the requested plots.

$$\frac{dy_1}{dt} = 4y_1 - y_1 y_2$$

$$\frac{dy_2}{dt} = -2y_2 + y_1 y_2$$

```
h = 0.1;
y1(1) = 1;
y2(1) = 4;
t = 0:h:10;
for i = 2:length(t)
    y1(i) = y1(i-1) + 0.5*h*f1(y1(i-1),y2(i-1));
    y2(i) = y2(i-1) + 0.5*h*f2(y1(i-1),y2(i-1));
end

tiledlayout(1,2)
nexttile
plot(y1,y2)
grid on
title('Phase : h=0.025');
xlabel('y1(t)');ylabel('y2(t)');

for k = 1:length(y1)-1
    y1t(k) = abs(y1(k+1) - y1(k));
end

nexttile
plot(t(1:end-1),y1t)
grid on
title('Dist in y1(t) : h=0.025');
xlabel('t');ylabel('Dist');


function f1 = f1(y1,y2)
    f1 = 4*y1 - y1*y2;
end

function f2 = f2(y1,y2)
    f2 = -2*y2 + y1*y2;
end
```

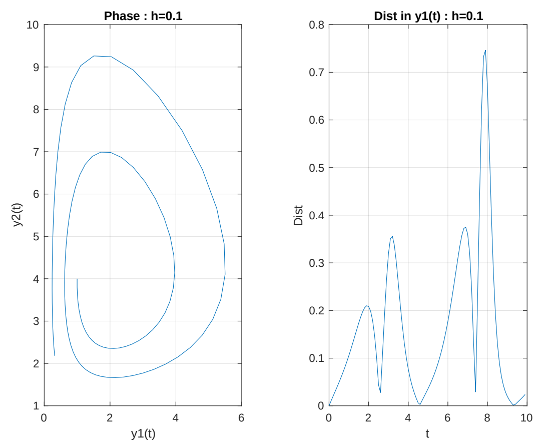

Figure 2: Plots for $h = 0.1$

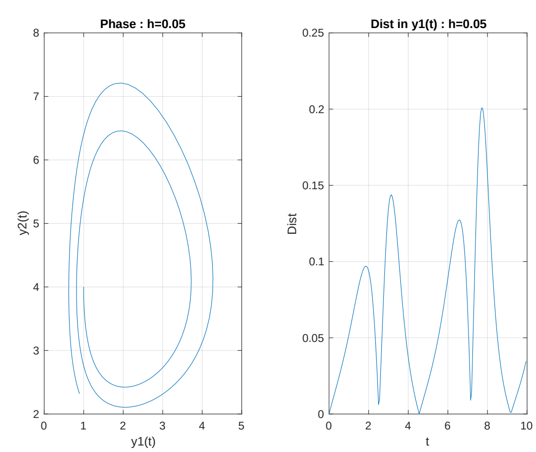

Figure 3: Plots for $h = 0.05$

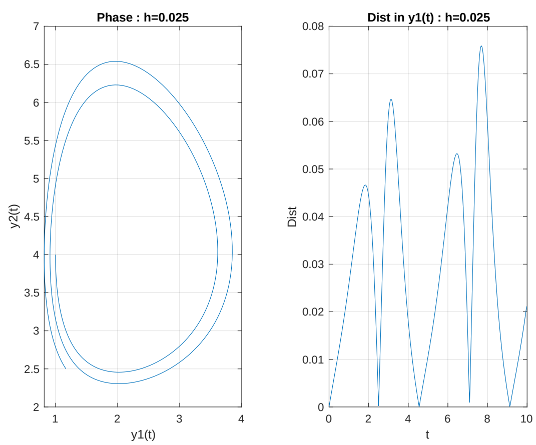

Figure 4: Plots for $h = 0.025$

Question 3:

a) Here are my four functions to implement the methods.

```
function [x,y] = euler(f,a,y0,b,stepsize)
    h = stepsize;
    y(1) = y0;
    x = a:h:b;
    for i = 2:length(x)-1
        y(i) = y(i-1) + h*f(x(i-1),y(i-1));
    end
end

function [x,y] = eulerMod(f,a,y0,b,stepsize)
    h = stepsize;
    hh = h/2;
    y(1) = y0;
    x = a:h:b;
    for i = 2:length(x)-1
        y(i) = y(i-1) + h*f(x(i-1)+hh,y(i-1)+hh*f(x(i-1),y(i-1)));
    end
end

function [x,y] = eulerImp(f,a,y0,b,stepsize)
    h = stepsize;
    hh = h/2;
    y(1) = y0;
    x = a:h:b;
    for i = 2:length(x)-1
        y(i) = y(i-1) + hh*(f(x(i-1),y(i-1))+f(x(i),y(i-1)+h*f(x(i-1),y(i-1))));
    end
end

function [x,y] = rungeKutta4(f,a,y0,b,stepsize)
    h = stepsize;
    hh = h/2;
    y(1) = y0;
    x = a:h:b;
    for i = 2:length(x)-1
        k1 = h*f(x(i-1),y(i-1));
        k2 = h*f(x(i-1)+hh,y(i-1)+k1/2);
        k3 = h*f(x(i-1)+hh,y(i-1)+k2/2);
        k4 = h*f(x(i-1)+h,y(i-1)+k3);
        y(i) = y(i-1) + (1/6)*(k1 + 2*k2 + 2*k3 + k4);
```

```
        end
end
```

b) Here is some code to solve the system $y' = -y^3/2, y(0) = 1$ with $h = 1/40$. I also provide the plots to see the solutions are appropriate.

```
[x0,y0] = euler(@(x,y)-(y^3)/2,0,1,5,1/40);
[x1,y1] = eulerMod(@(x,y)-(y^3)/2,0,1,5,1/40);
[x2,y2] = eulerImp(@(x,y)-(y^3)/2,0,1,5,1/40);
[x3,y3] = rungeKutta4(@(x,y)-(y^3)/2,0,1,5,1/40);


plot(x0(1:end-1),y0)
hold on
grid on
plot(x1(1:end-1),y1)
plot(x2(1:end-1),y2)
plot(x3(1:end-1),y3)
fplot(@(x)1/(x+1)^0.5)
title("Solutions");
xlabel('x');ylabel('y');
legend("Euler","Mod Euler","Imp Euler","Rk4","Exact")
```

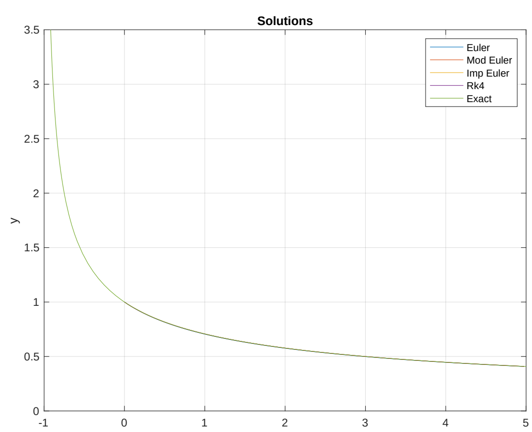

Figure 5: Solutions for $h = 1/40$ zoomed out

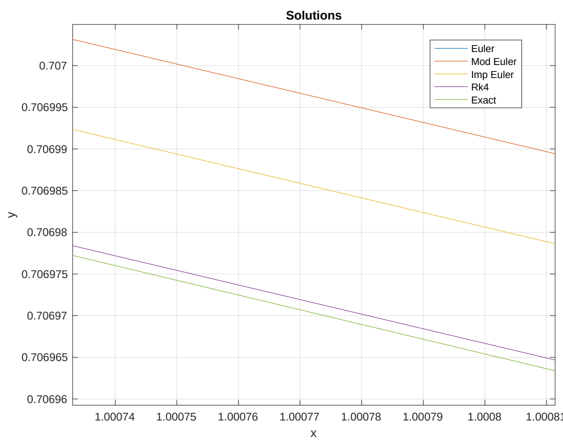

Figure 6: Solutions for $h = 1/40$ zoomed in

For $h = 1/80$ we run

```
[x0,y0] = euler(@(x,y)-(y^3)/2,0,1,5,1/80);
[x1,y1] = eulerMod(@(x,y)-(y^3)/2,0,1,5,1/80);
[x2,y2] = eulerImp(@(x,y)-(y^3)/2,0,1,5,1/80);
[x3,y3] = rungeKutta4(@(x,y)-(y^3)/2,0,1,5,1/80);

plot(x0(1:end-1),y0)
hold on
grid on
plot(x1(1:end-1),y1)
plot(x2(1:end-1),y2)
plot(x3(1:end-1),y3)
fplot(@(x)1/(x+1)^0.5)
title("Solutions");
xlabel('x');ylabel('y');
legend("Euler","Mod Euler","Imp Euler","Rk4","Exact")
```

Now we will calculate the errors at $x = 1$.

```
function y = yexact(x)
    y = 1/(x+1)^0.5;
end
[x0,y0] = euler(@(x,y)-(y^3)/2,0,1,5,1/40);
[x1,y1] = euler(@(x,y)-(y^3)/2,0,1,5,1/80);

abs(yexact(1) - y0(41))/2
abs(yexact(1) - y1(81))
```

```
ans =
    0.0012
ans =
    0.0012
```

The error is reduced by a half for Euler as h halves.

```
[x0,y0] = eulerMod(@(x,y)-(y^3)/2,0,1,5,1/40);
[x1,y1] = eulerMod(@(x,y)-(y^3)/2,0,1,5,1/80);

abs(yexact(1) - y0(41))/4
abs(yexact(1) - y1(81))
```

```
ans =
   6.1900e-06
ans =
   6.1156e-06
```

The error is reduced by a quarter for Modified Euler as h halves.

```
[x0,y0] = eulerImp(@(x,y)-(y^3)/2,0,1,5,1/40);
[x1,y1] = eulerImp(@(x,y)-(y^3)/2,0,1,5,1/80);

abs(yexact(1) - y0(41))/4
abs(yexact(1) - y1(81))
```

```
ans =
   3.4884e-06
ans =
   3.4707e-06
```

The error is reduced by a quarter for Improved Euler as h halves.

```
[x0,y0] = rungeKutta4(@(x,y)-(y^3)/2,0,1,5,1/40);
[x1,y1] = rungeKutta4(@(x,y)-(y^3)/2,0,1,5,1/80);

abs(yexact(1) - y0(41))/16
abs(yexact(1) - y1(81))
```

```
ans =
   4.4290e-12
ans =
   4.6783e-12
```

The error is reduced by 1/16 for RK4 as h halves.

Lastly, we compare the methods. First I will say from the plot and the errors, RK4 drastically outperforms the others as it lies closest to the exact solution. However, let us look at performance speed.

```
tic
[x0,y0] = euler(@(x,y)-(y^3)/2,0,1,5,1/40);
toc
tic
[x1,y1] = eulerMod(@(x,y)-(y^3)/2,0,1,5,1/40);
toc
tic
[x2,y2] = eulerImp(@(x,y)-(y^3)/2,0,1,5,1/40);
toc
tic
[x3,y3] = rungeKutta4(@(x,y)-(y^3)/2,0,1,5,1/40);
toc

tic
[x0,y0] = euler(@(x,y)-(y^3)/2,0,1,5,1/80);
toc
tic
[x1,y1] = eulerMod(@(x,y)-(y^3)/2,0,1,5,1/80);
toc
tic
[x2,y2] = eulerImp(@(x,y)-(y^3)/2,0,1,5,1/80);
toc
tic
[x3,y3] = rungeKutta4(@(x,y)-(y^3)/2,0,1,5,1/80);
toc
```

```
Elapsed time is 0.001856 seconds.
Elapsed time is 0.005344 seconds.
Elapsed time is 0.001697 seconds.
Elapsed time is 0.004949 seconds.

Elapsed time is 0.001863 seconds.
Elapsed time is 0.009241 seconds.
Elapsed time is 0.006890 seconds.
Elapsed time is 0.012574 seconds.
```

From the output, we see that Euler's runtime does not significantly increase as h halves whereas the other methods do. We also see that at both h values, Euler and Improved Euler have similar speeds and RK4 and modified Euler have similar speeds, the former methods being faster.

Question 4:

First, we analyze the stability of the method $y_{k+1} = 4y_k - 3y_{k-1} - 2hf(t_{k-1}, y_{k-1})$ for $f(y) = -\lambda y$. Using an idea from the textbook, we transform the equation into a quadratic in q via the substitution $y_k = q^k$ resulting in

$$q^2 = 4q + (2h\lambda - 3)$$

$\implies$

$$q = 2 \pm \sqrt{1 + 2h\lambda}$$

Then we plot the regions as follows.

```
hSpan = linspace(0,3,101);
for j = 1 : length(hSpan)
h = hSpan(j);
    q(:,j) = roots([1,-4,-2*h+3])';
end
plot(hSpan,abs(q),'.b');
hold on
grid on
yline(1,"r")
yline(-1,"r")
title("Stability Plot");
xlabel('z');ylabel('|q|');
```

From the plot, we see that there is on root in the stability region and one above the stability region on $[0, 3]$, thus, the method is unstable.

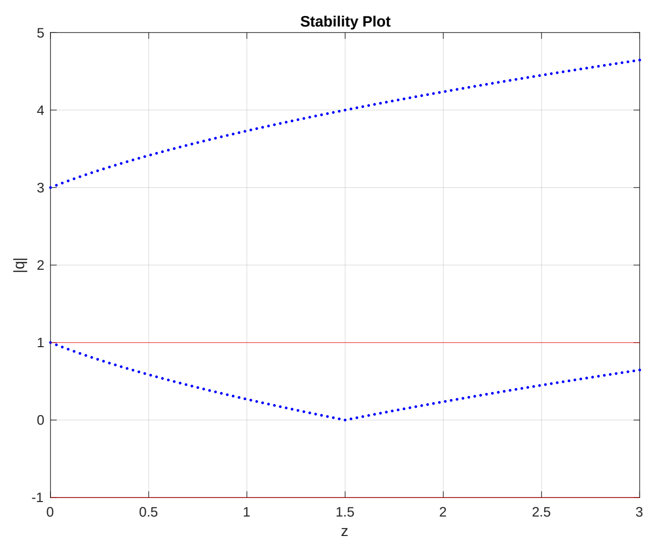

Figure 7: Stability

Using Wolfram Alpha to quickly find the solution, we have $y_{\text{exact}} = 2^{e^t}$. So we can plot the error of the implementation as follows.

```
h = 0.05;
y(1) = 2;
y(2) = 8;
t = 0:h:3;
for i = 3:length(t)
    y(i) = 4*y(i-1)-3*y(i-2)-2*h*f(t(i-2),y(i-2));

end
for j = 1:length(t)
    error(j) = abs(ff(t(j)) - y(j));
end
loglog(t,error)
xlabel("t")
ylabel("Error")
title("Error Plot")
grid on

function f = f(t,y)
    f = -y*log(y);
end

function f2 = ff(x)
    f2 = 2^(exp(x));
end
```

We can see that the error is accumulating quite heavily as one moves towards the right limit of the interval $[0, 3]$.

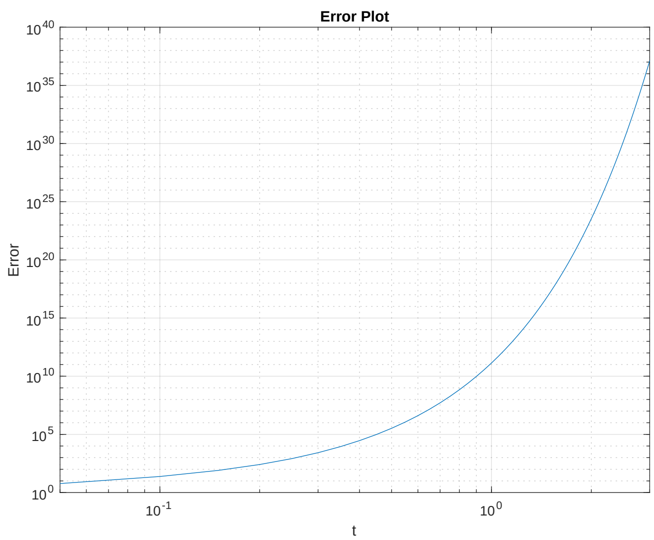

Figure 8: Error