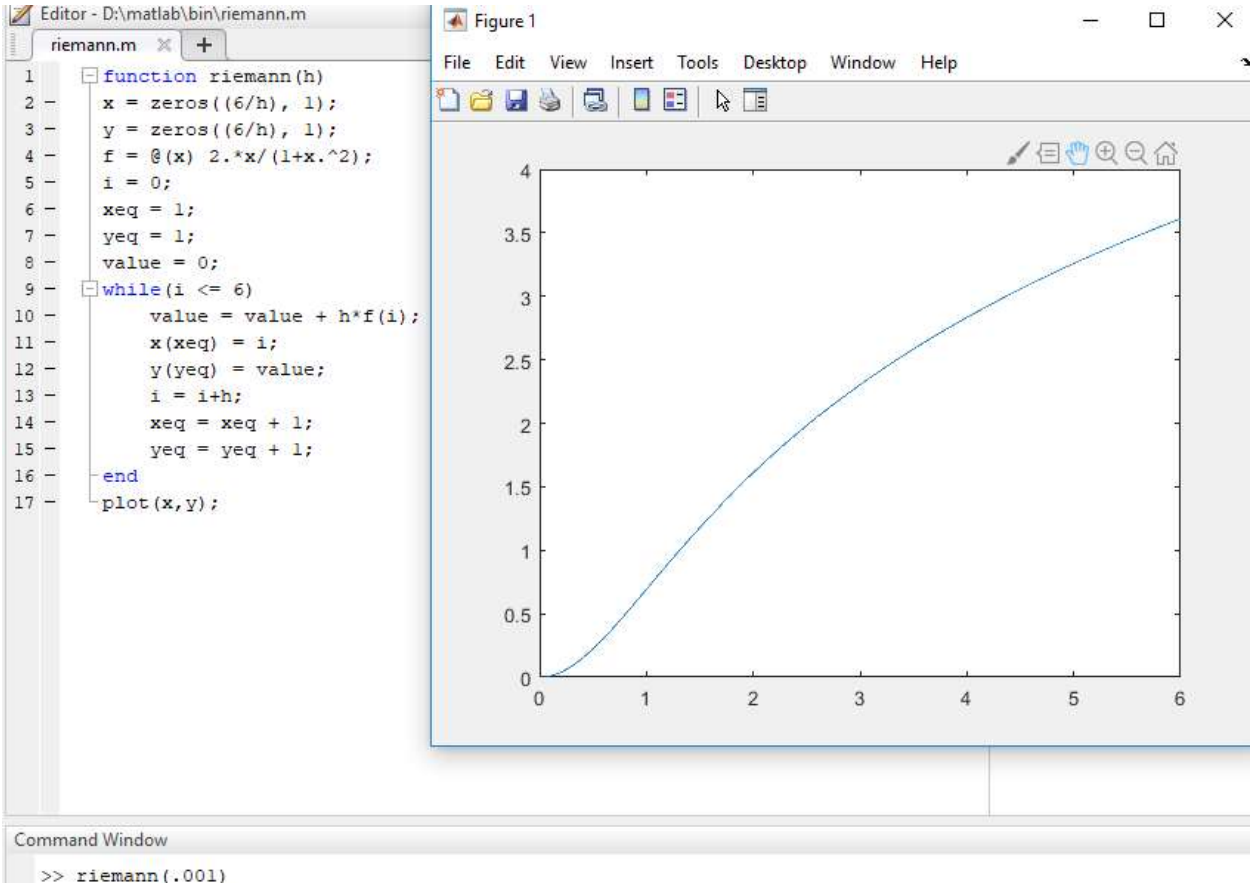
Project 2

Tristan Millman

4396-7960

1. Riemann Sums

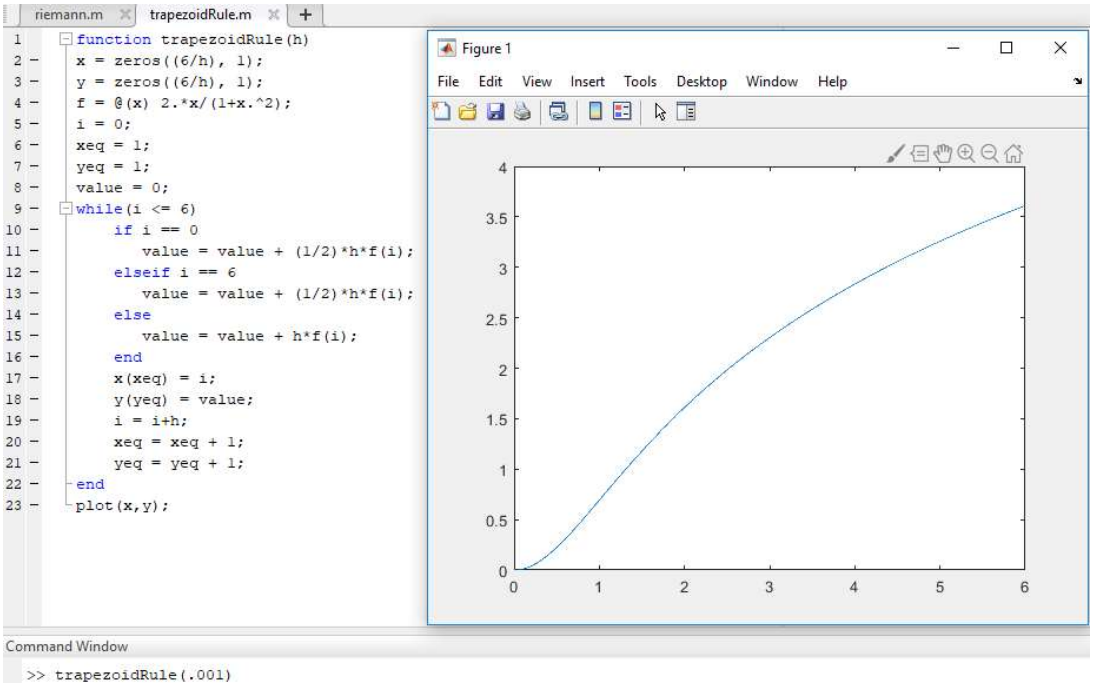
You can see the code and graph of the area from 0 to 6 here.



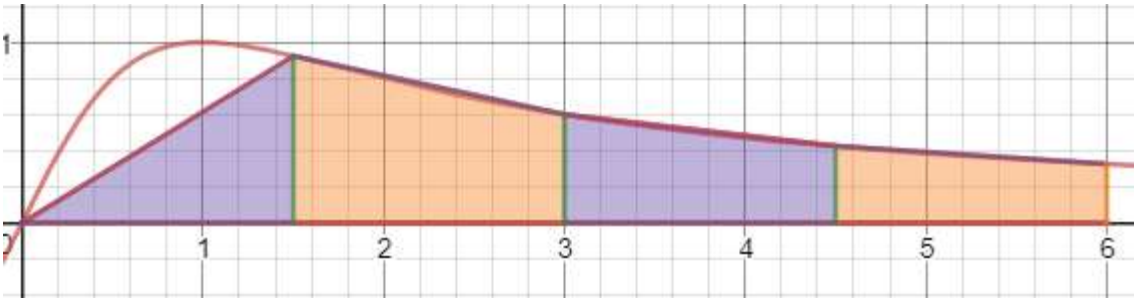
This picture shows the Riemann sum approximation on the original function.

2. Trapezoid Rule

The code and the graph of the area from 0 to 6 are shown in this picture



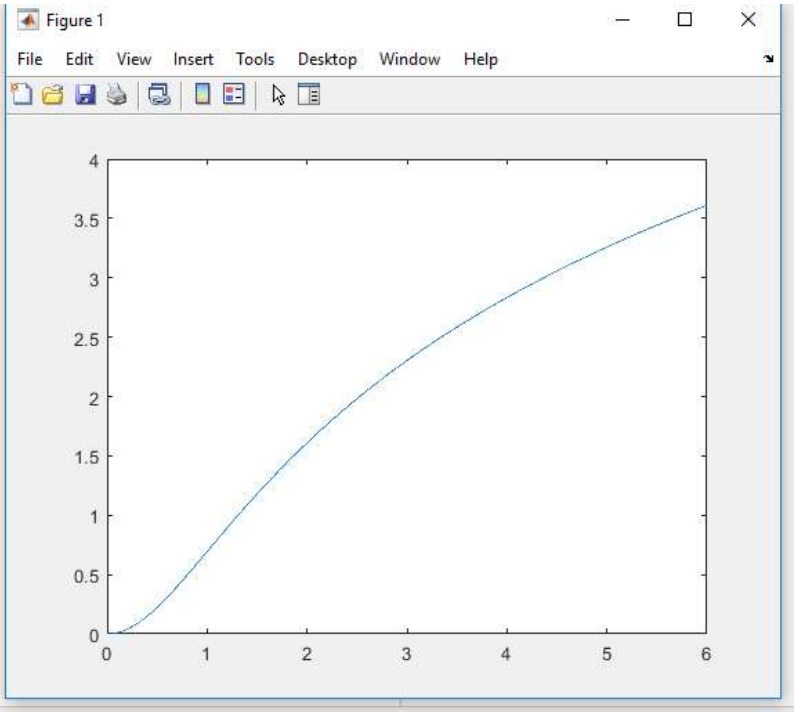
You can see the Trapezoid rule applied on the original function in the picture below.



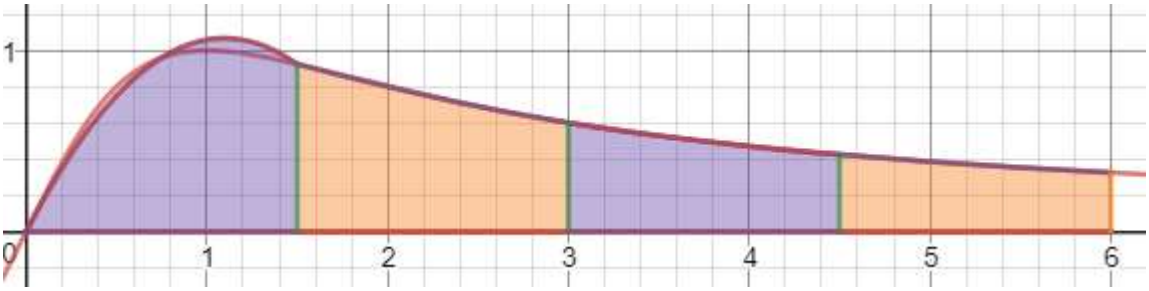
3. Simpsons Rule

The code I used and the graph of the area given by Simpsons Rule can be seen below.

```
function simpsonsRule(h)
x = zeros((6/h), 1);
y = zeros((6/h), 1);
f = @(x) 2.*x/(1+x.^2);
i = 0;
xeq = 1;
yeq = 1;
value = 0;
count = 0;
while(i <= 6)

    if i == 0
        value = value + h*f(i);
    elseif i == 6
        value = value + h*f(i);
    elseif count == 1
        value = value + 4*h*f(i);
    elseif count == 2
        count = 0;
        value = value + 2*h*f(i);
    end
    x(xeq) = i;
    y(yeq) = value;
    i = i+h;
    xeq = xeq + 1;
    yeq = yeq + 1;
    count = count +1;
end
```

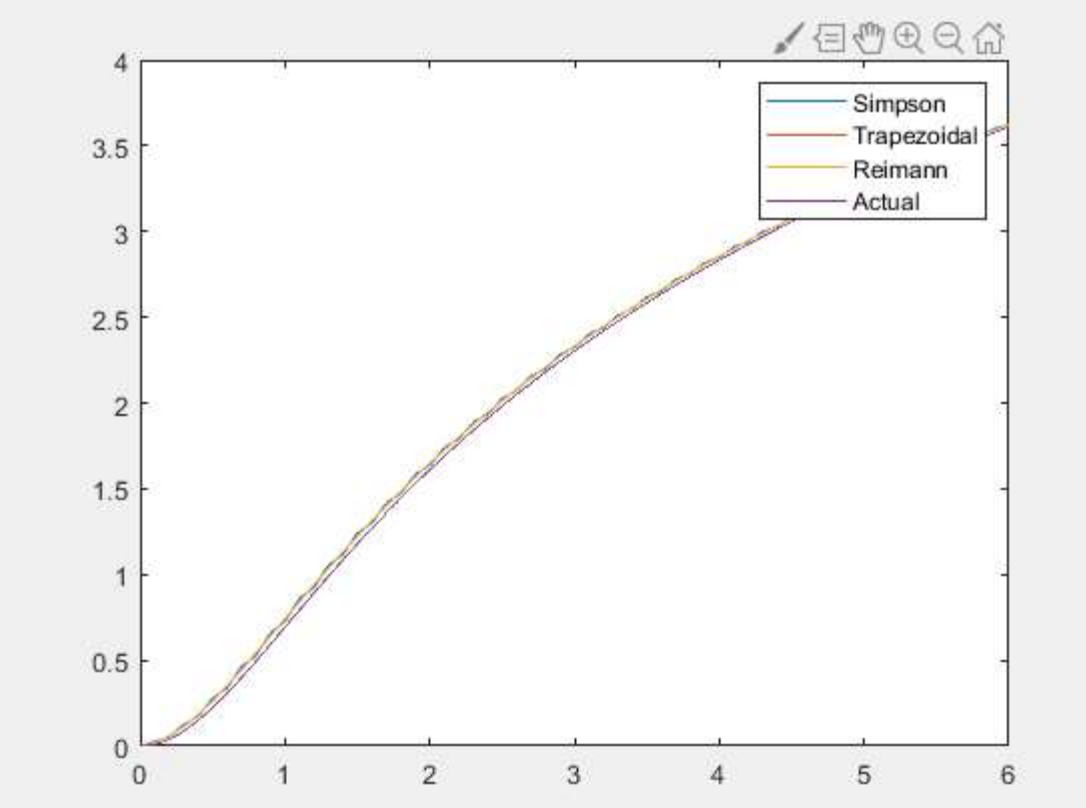


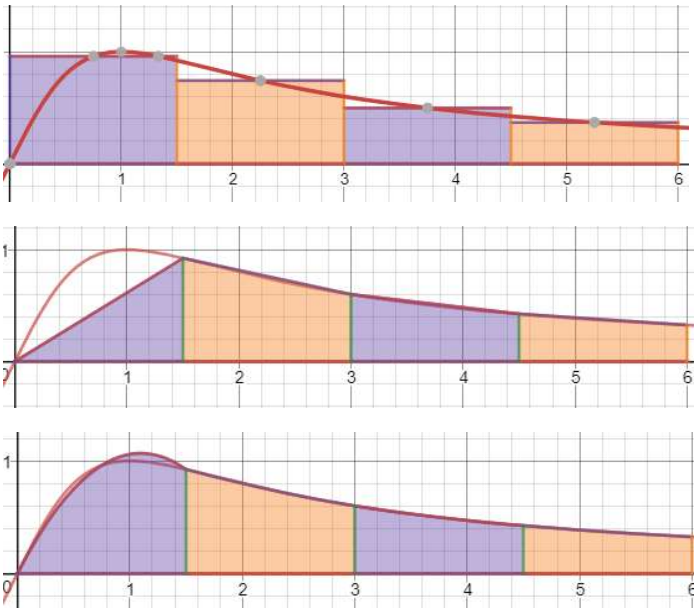
You can see the Simpson's rule being applied to the original function in the graph below.



4. Comparison

Here you can see that the areas were actually all quite close to one another, when graphed. However, Simpson's Rule is the most accurate and returns the most accurate area. This is more evident int the next set of pictures. In all of the graphs from Matlab for problems 1-4, the lines represent an approximation of ln(1+x^2) as that is the integral of the provided function.





In these three pictures you can see the difference in approximation from Riemann sum, Trapezoid Rule, and Simpsons Rule. Simpsons Rule visibly has the best approximation, and it will return the closest to the actual area of all three functions.

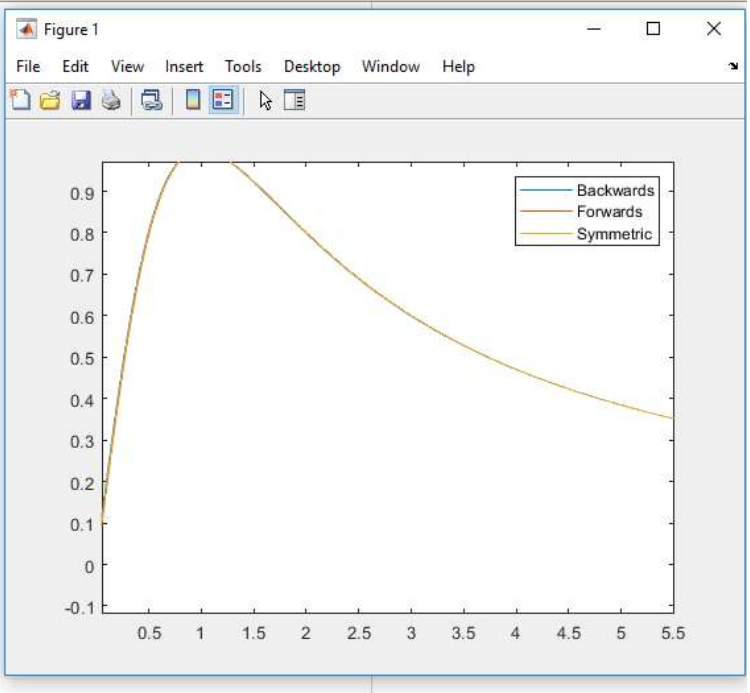
These three pictures and their counterparts in problems 1, 2, and 3 were obtained using Desmos, an online graphing tool.

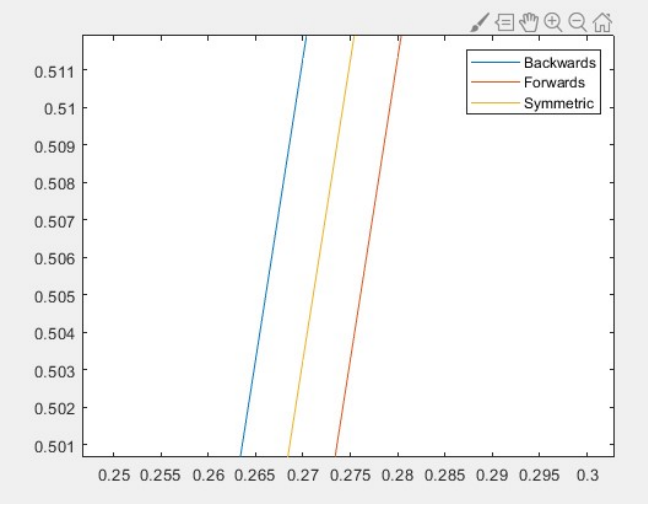
5. Backwards, Forwards, and Symmetric Difference

Below you can see the code and the graph of the backwards, forwards, and symmetric difference.

bfsDifference.m | riemann.m | allThree.m

```
function bfsDifference()

f = @(x) log(1+x.^2);
x = zeros((6/.01), 1);
y = zeros((6/.01), 1);
xeq = 1;
yeq = 1;
%backwards
i = 0;
while i <= 6
    ii = i +.01;
    x(xeq) = i;
    y(yeq) = (f(ii)-f(i))/.01;
    i = i + .01;
    xeq = xeq + 1;
    yeq = yeq + 1;
end

%forwards
xx = zeros((6/.01), 1);
yy = zeros((6/.01), 1);
xeq = 1;
yeq = 1;
i = 0;
while i <= 6
    ii = i -.01;
    xx(xeq) = i;
    yy(yeq) = (f(i)-f(ii))/.01;
    i = i + .01;
    xeq = xeq + 1;
    yeq = yeq + 1;
end

%symmetric
xxx = xx;
yyy = (yy+y)/2;

plot(x,y,xx,yy,xxx,yyy);
legend('Backwards','Forwards','Symmetric');
```





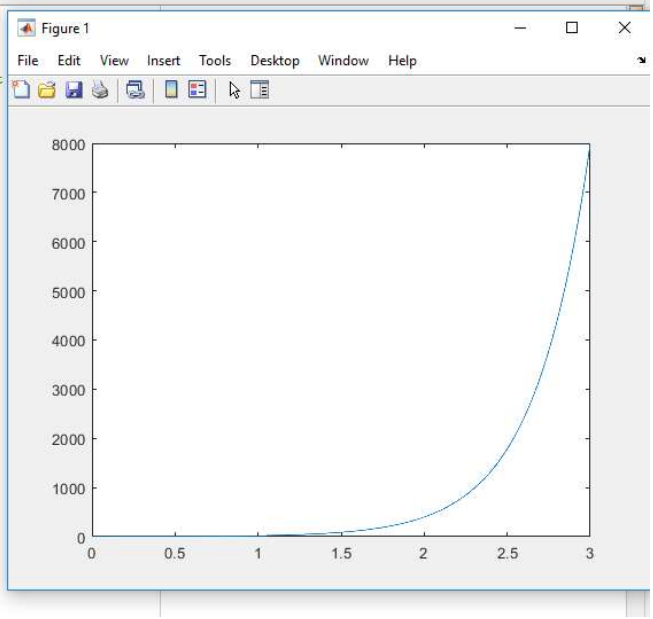
On the left is a zoomed in version of the graph shown above. You can see that each curve acts as you would expect it to. The Backwards difference is offset to the left, the forwards difference is offset to the right, and the Symmetric difference appears to be an average of the two.

6. Euler's Method

Below is the Euler's code I used, and a plotting of the resulting vector.

eulers.m | bfsDifference.m

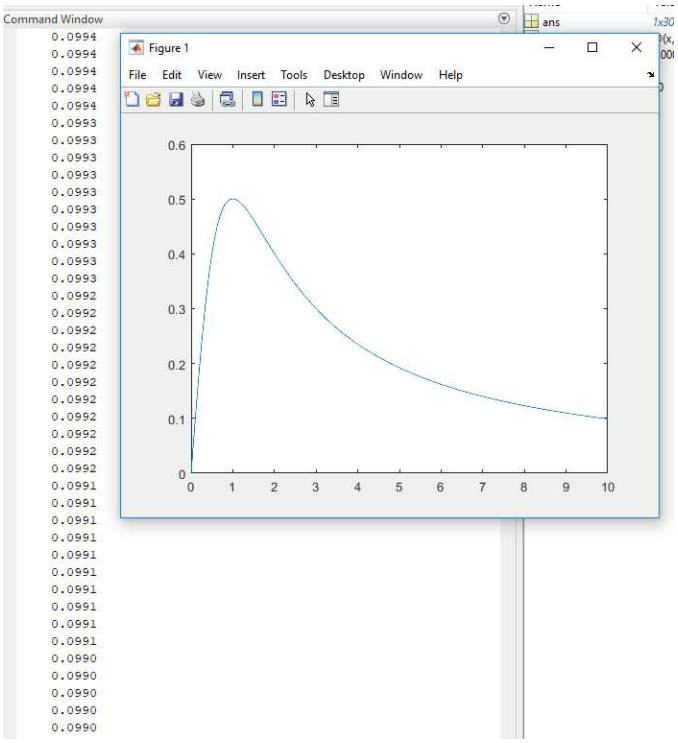
```
function eulers(F,t0,h,tfinal,y0)

   %F is the given funciton
   %t0 and tfinal are the beginning and ending values of t
   %h is the step size
   %y0 is first y



y = y0;
yout = y;
for t = t0 : h : tfinal-h
    s = F(t,y);
    y = y + h*s;
    yout = [yout; y];
end
x = [t0:h:tfinal];
plot(x,yout);
f = @(t,y) exp(3*t);
```



Command Window

```
    7.1132
    7.1345
    7.1559
    7.1774
    7.1989
    7.2205
    7.2422
    7.2639
    7.2857
    7.3075
    7.3295
    7.3514
    7.3735
    7.3956
    7.4178
    7.4401
    7.4624
    7.4848
    7.5072
    7.5297
    7.5523
    7.5750
    7.5977
    7.6205
    7.6434
    7.6663
    7.6893
    7.7124
    7.7355
    7.7587
    7.7820
    7.8053
    7.8288
    7.8522
    7.8758
    7.8994
    7.9231
    7.9469
    7.9707
    7.9946

>>
```

Additionally, the code outputs the resulting vector that is plotted in the graph shown.



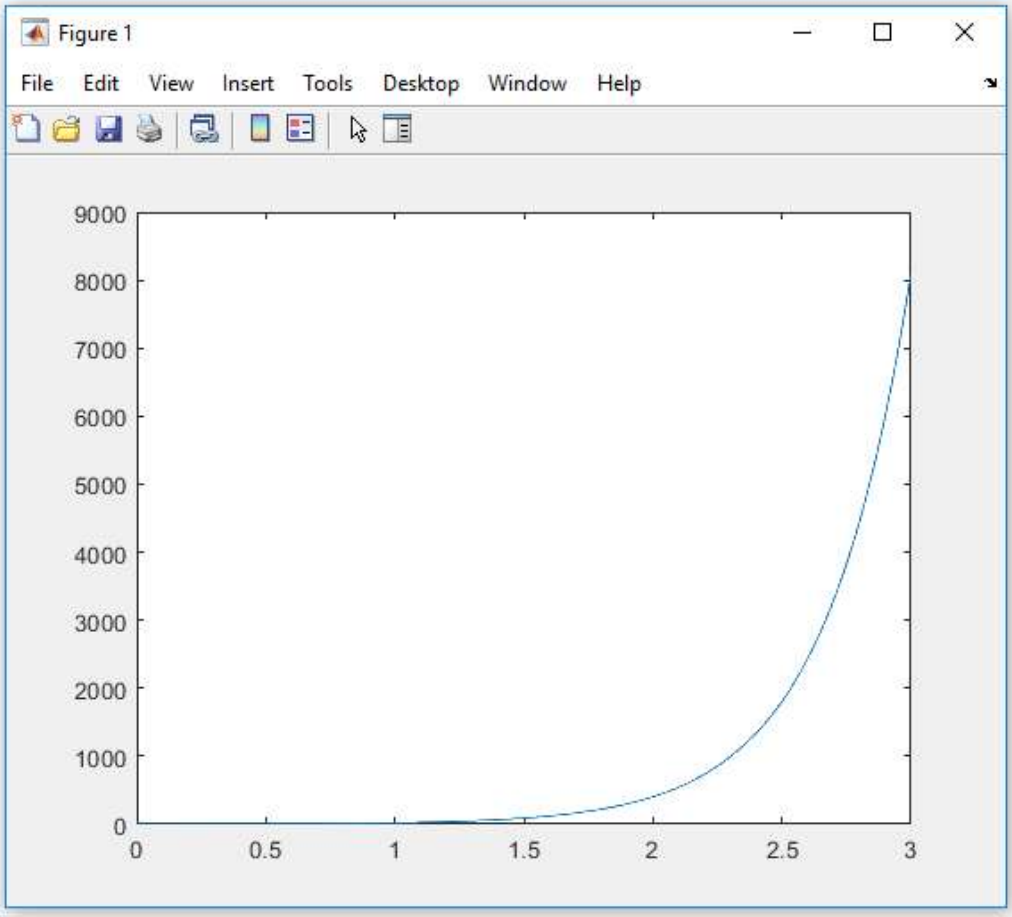
Euler's Method, second problem.

7. Midpoint Method

```
function midpointMethod(F,t0,h,tfinal,y0)
y = y0;
yout = y;
for t = t0 : h : tfinal-h
    s1 = F(t,y);
    s2 = F(t+h/2, y+h*s1/2);
    y = y + h*s2;
    yout = [yout; y];
end
disp(yout)
x = [t0:h:tfinal];
plot(x,yout);
```

Command Window

```
   7.4501
   7.4725
   7.4950
   7.5175
   7.5401
   7.5627
   7.5854
   7.6082
   7.6311
   7.6540
   7.6770
   7.7001
   7.7232
   7.7464
   7.7697
   7.7930
   7.8165
   7.8399
   7.8635
   7.8871
   7.9108
   7.9346
   7.9584
   7.9823
   8.0063
   8.0304
   8.0545
   8.0787
   8.1030
```



fx >>

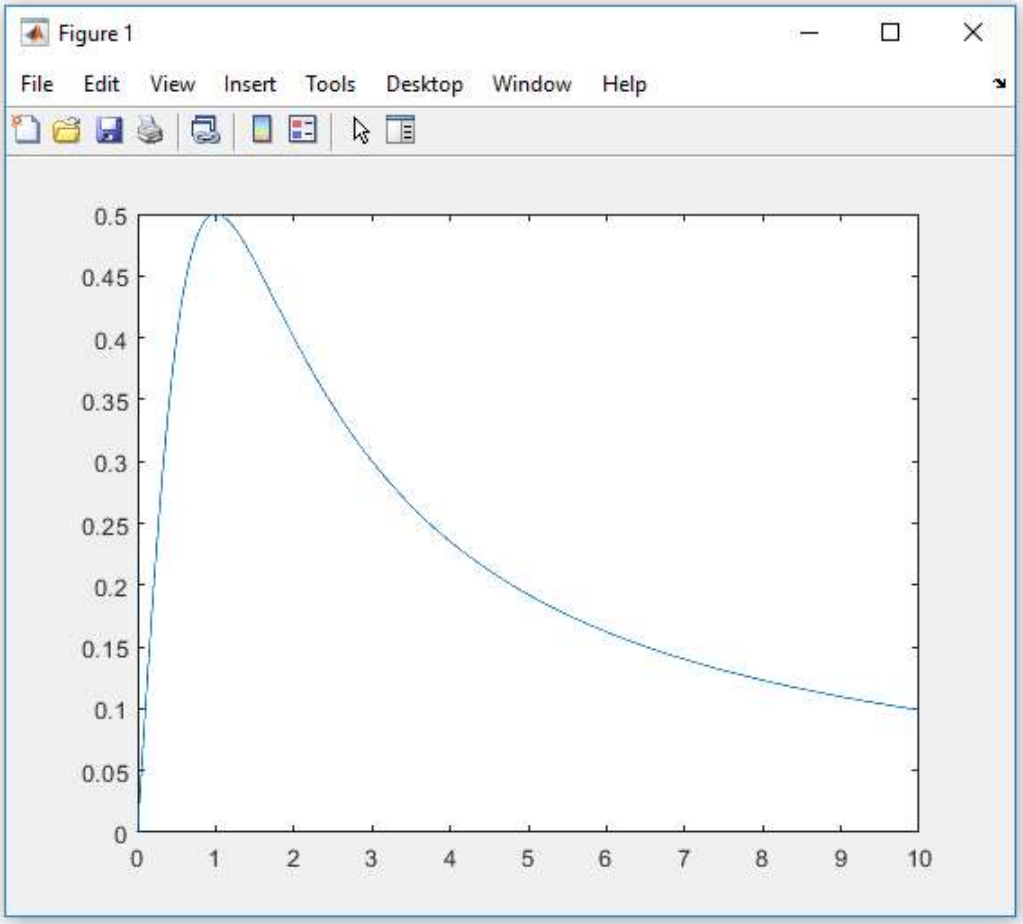
Problem 1

midpointMethod.m | eulers.m

```
function midpointMethod(F,t0,h,tfinal,y0)
y = y0;
yout = y;
for t = t0 : h : tfinal-h
    s1 = F(t,y);
    s2 = F(t+h/2, y+h*s1/2);
    y = y + h*s2;
    yout = [yout; y];
end
disp(yout)
x = [t0:h:tfinal];
plot(x,yout);
```

Command Window

```
0.0993
0.0993
0.0993
0.0993
0.0992
0.0992
0.0992
0.0992
0.0992
0.0992
0.0992
0.0992
0.0992
0.0991
0.0991
0.0991
0.0991
0.0991
0.0991
0.0991
0.0991
0.0991
0.0991
0.0990
0.0990
0.0990
0.0990
0.0990
```



Problem 2

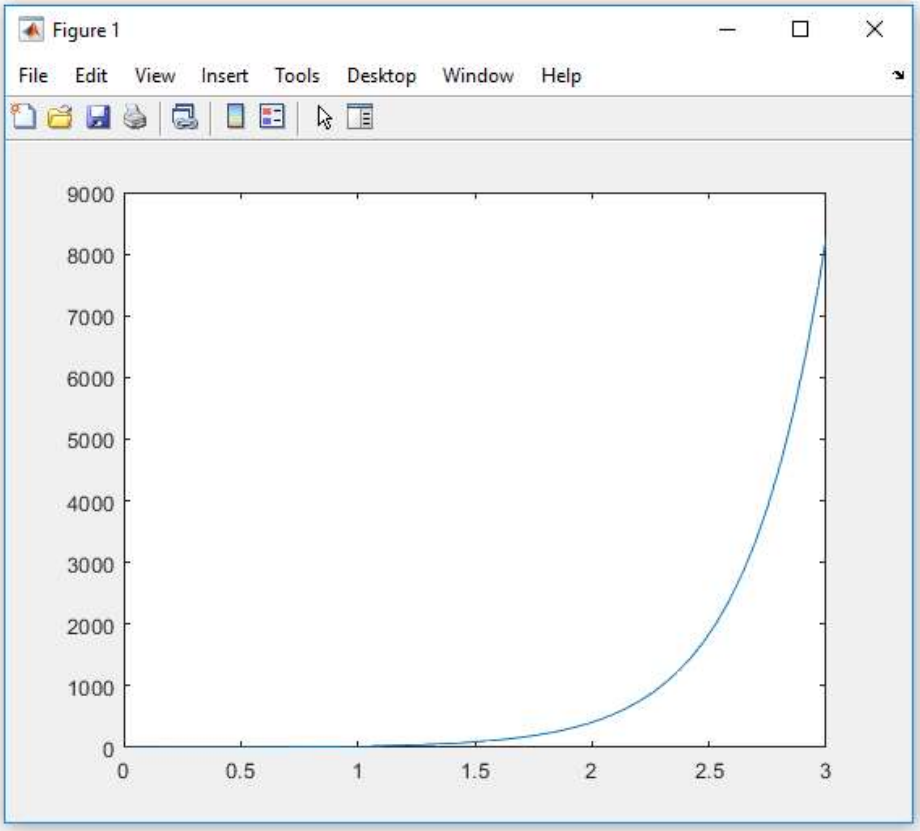
8. Trapezoid Method

midpointMethod.m | eulers.m | trapezoidMethod.m

```
function trapezoidMethod(F,t0,h,tfinal,y0)
tspan = [t0 tfinal];
[t,y] = ode23t(F, tspan, y0)
plot(t,y)
```

Command Window

```
    0.0898
    0.1058
    0.1246
    0.1468
    0.1729
    0.2037
    0.2400
    0.2827
    0.3331
    0.3924
    0.4622
    0.5445
    0.6415
    0.7558
    0.8903
    1.0489
    1.2357
    1.4557
    1.7150
    2.0204
    2.3802
    2.8040
    3.3034
    3.8916
    4.5846
    5.4011
    6.3629
    7.4960
    8.2672

fx >>
```



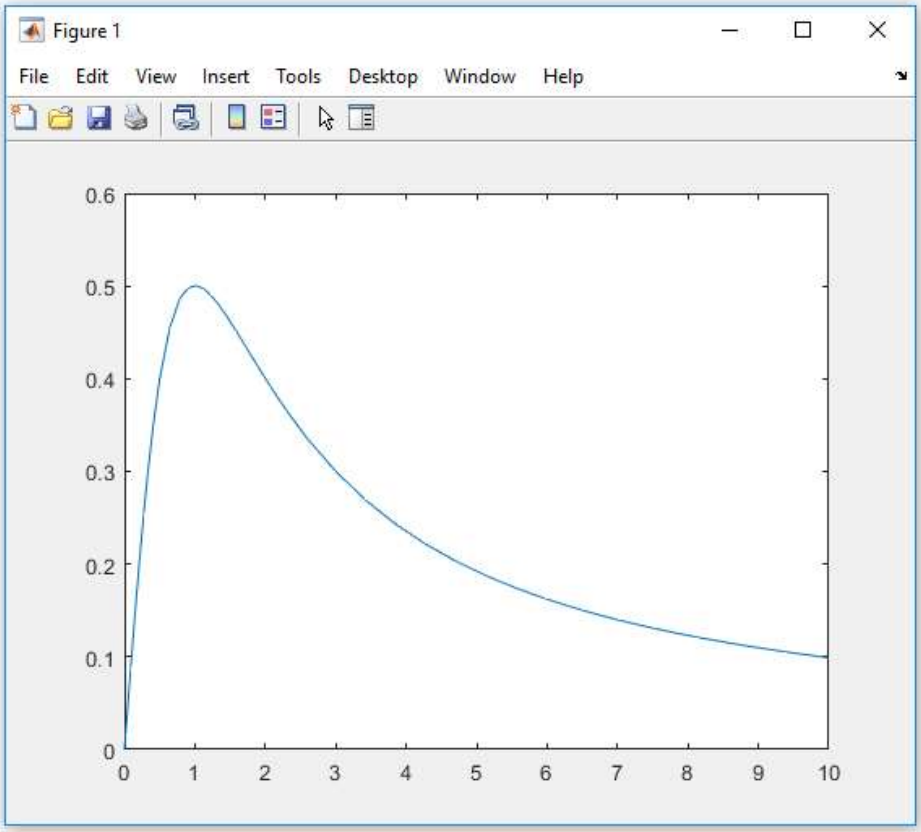
Problem 1

midpointMethod.m | eulers.m | trapezoidMethod.m

```
function trapezoidMethod(F,t0,h,tfinal,y0)
tspan = [t0 tfinal];
[t,y] = ode23t(F, tspan, y0)
plot(t,y)
```

Command Window

```
    0.4557
    0.4870
    0.4978
    0.5005
    0.4971
    0.4896
    0.4793
    0.4671
    0.4506
    0.4335
    0.4119
    0.3910
    0.3646
    0.3359
    0.3000
    0.2690
    0.2432
    0.2216
    0.2034
    0.1878
    0.1743
    0.1626
    0.1503
    0.1397
    0.1304
    0.1207
    0.1124
    0.1051
    0.0990
```



Problem 2

9. Midpoint method seems to work the best in this case. It reduces the error faster and has the lowest overall error. It seems to be followed by the trapezoidal rule and then Euler's method, which seems to be the worst out of the three.