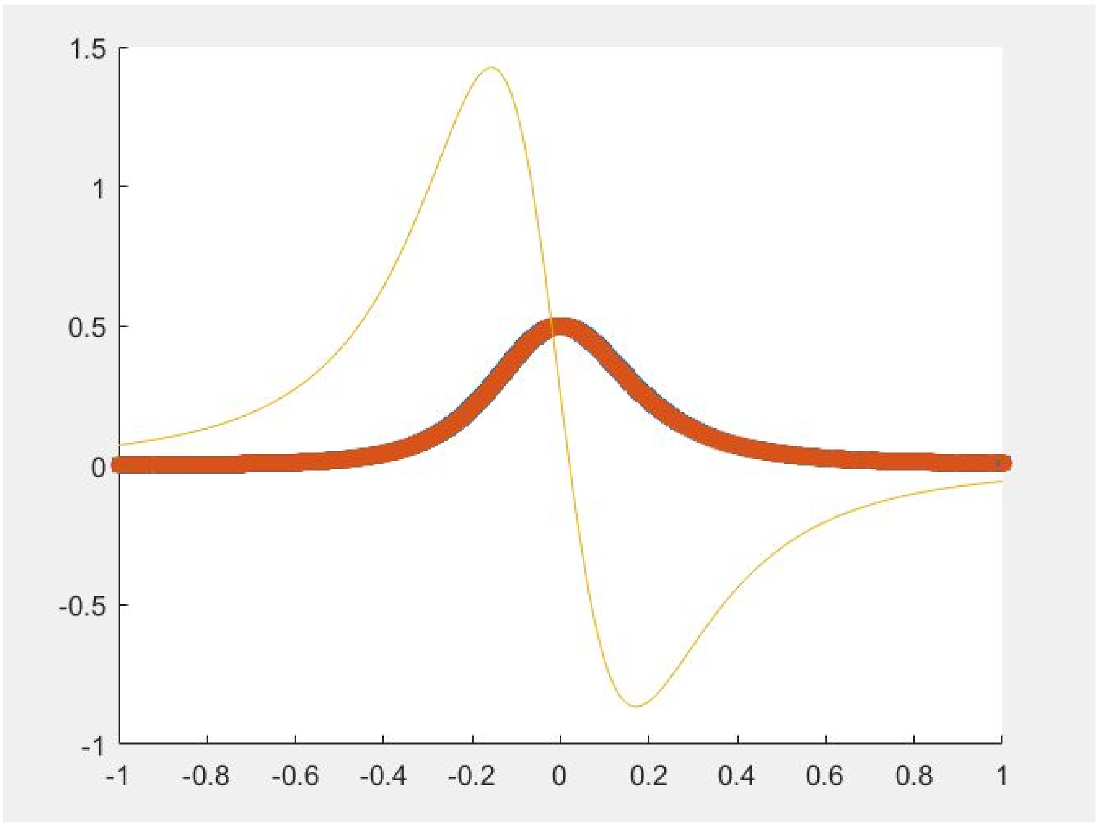
Problem Set 3: Numerical Differentiation and Integration
Jack Weissenberger

How do the errors of the methods change as a function of n? Does either method hit its maximum attainable accuracy?

In my code, the two methods reach a maximum attainable accuracy. This could either be because this is the best attainable accuracy of the methods or there is an error in my code. (My guess is that there is an error in my code because if the general definition of the derivative uses the limit as h approaches 0 of the forward difference scheme, then the error should continually decrease as the number of subintervals increases)

The error decreases as the number of subintervals increases. This makes sense because the approximations will better fit the function as the number of subintervals increases



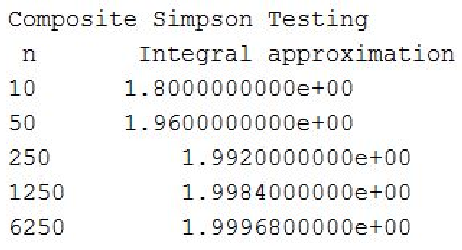
.

The graphs look like this because as the derivative is changing the quickest, the approximations do not model the the function well and thus the error is largest at these points.

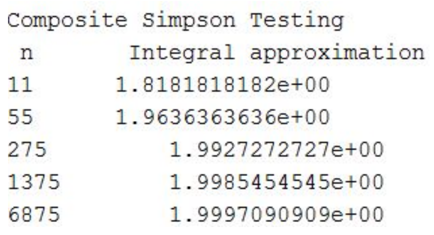
Then to test the composite simpson code I used this script
n = 10;

```
fprintf('\nComposite Simpson Testing\n');

while n < 10000
cs = comp_simp(ones(1, n));
fprintf('%1.6e \n', cs)

n = n*5;
end
```

This was the output:



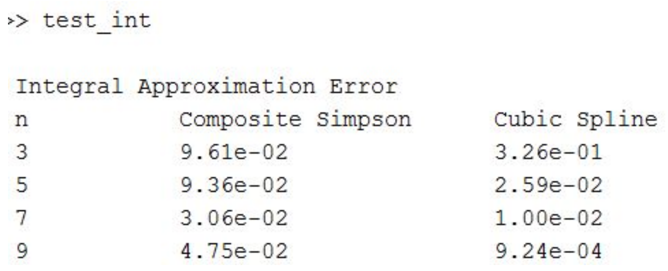
```
Composite Simpson Testing
 n       Integral approximation
10      1.8000000000e+00
50      1.9600000000e+00
250          1.9920000000e+00
1250         1.9984000000e+00
6250         1.9996800000e+00
```



```
Composite Simpson Testing
 n       Integral approximation
11      1.8181818182e+00
55      1.9636363636e+00
275          1.9927272727e+00
1375         1.9985454545e+00
6875         1.9997090909e+00
```

The function that I was modeling was a horizontal line at y = 1 from -1 to one. So the integral should be equal to 2. I tested the code for even and odd sub intervals so both cases in the code could be tested. As you can see the error gets closer to 2 as the number of subintervals increases and it eventually becomes accurate to 5 digits.

This is the output from the test_int code:



```
>> test_int

Integral Approximation Error
n           Composite Simpson      Cubic Spline
3           9.61e-02               3.26e-01
5           9.36e-02               2.59e-02
7           3.06e-02               1.00e-02
9           4.75e-02               9.24e-04
```

As the number of subintervals increases the error in the cubic splines decreases by orders of magnitude whereas the composite simpson rule decreases within the same order of magnitude.

It took 36 operations for my composite simpson to reach 1e-2 accuracy, and about 3000 to reach 1e-4, so there is clearly some error in my algorithm. If I had perfectly executed this algorithm it should of had O(h^4) accuracy. Which means that it can perfectly model cubic, quadratic and linear functions but will definitely be worse and slower than quad. This is because quad compares Simpson's rule to trapezoidal rule and then uses Romberg extrapolation to obtain a more accurate answer. Quad will also be quicker on linear functions because it compares Simpson's to trapezoidal, and trapezoidal rule works perfectly for linear functions which would allow quad to converge faster with less function evaluations. This is what we talked about in class about how when the function has less wiggle quad is able to know that it is fitting the function well if the error between trapezoidal and Simpson’s will be small. But a pure composite Simpson’s method would probably converge quicker to a function with a lot of wiggle if you started with enough subintervals because quad would have to go through all of the comparisons and evaluations until it realized that there is a lot of wiggle in the function and needs a lot of subintervals.