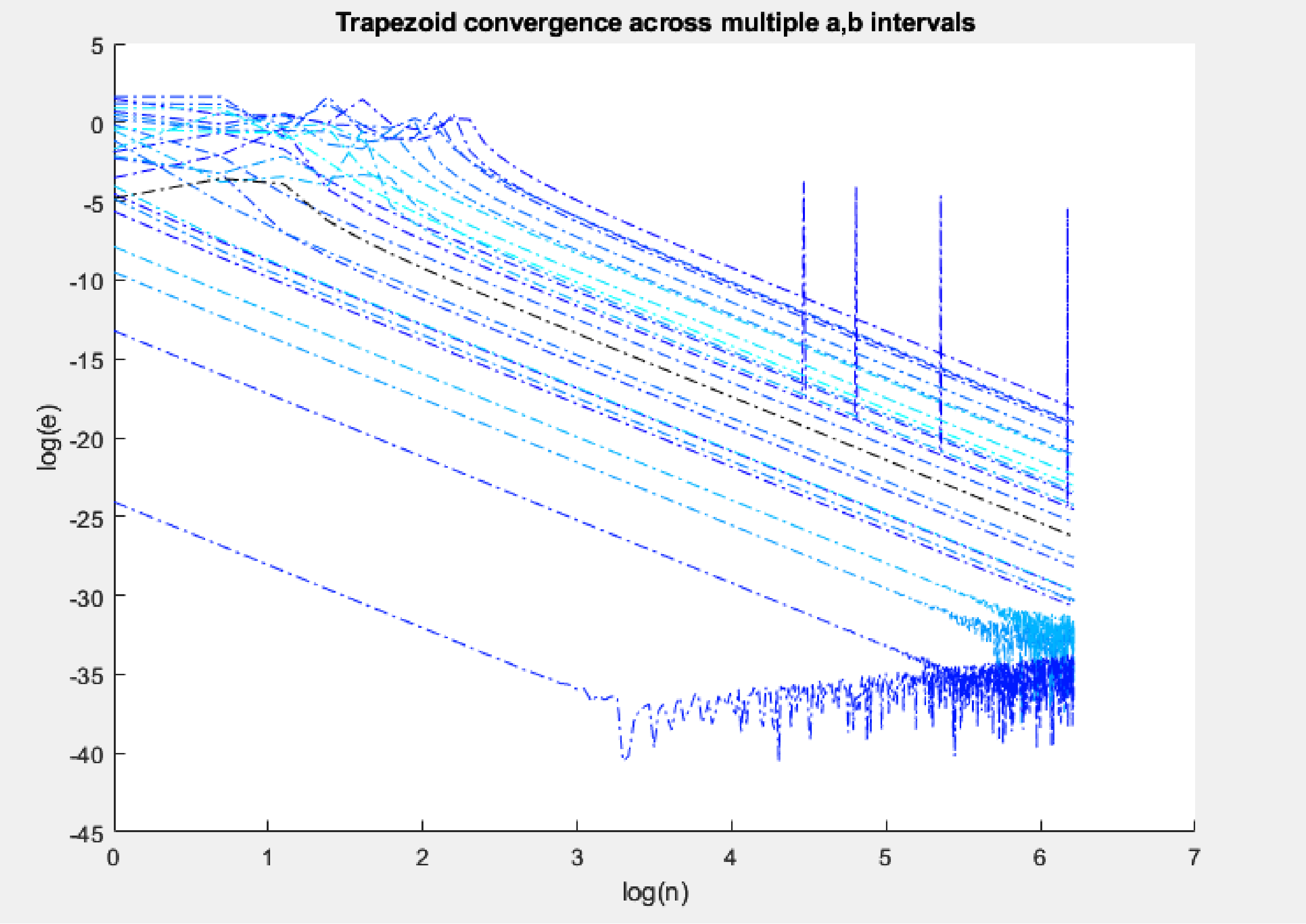
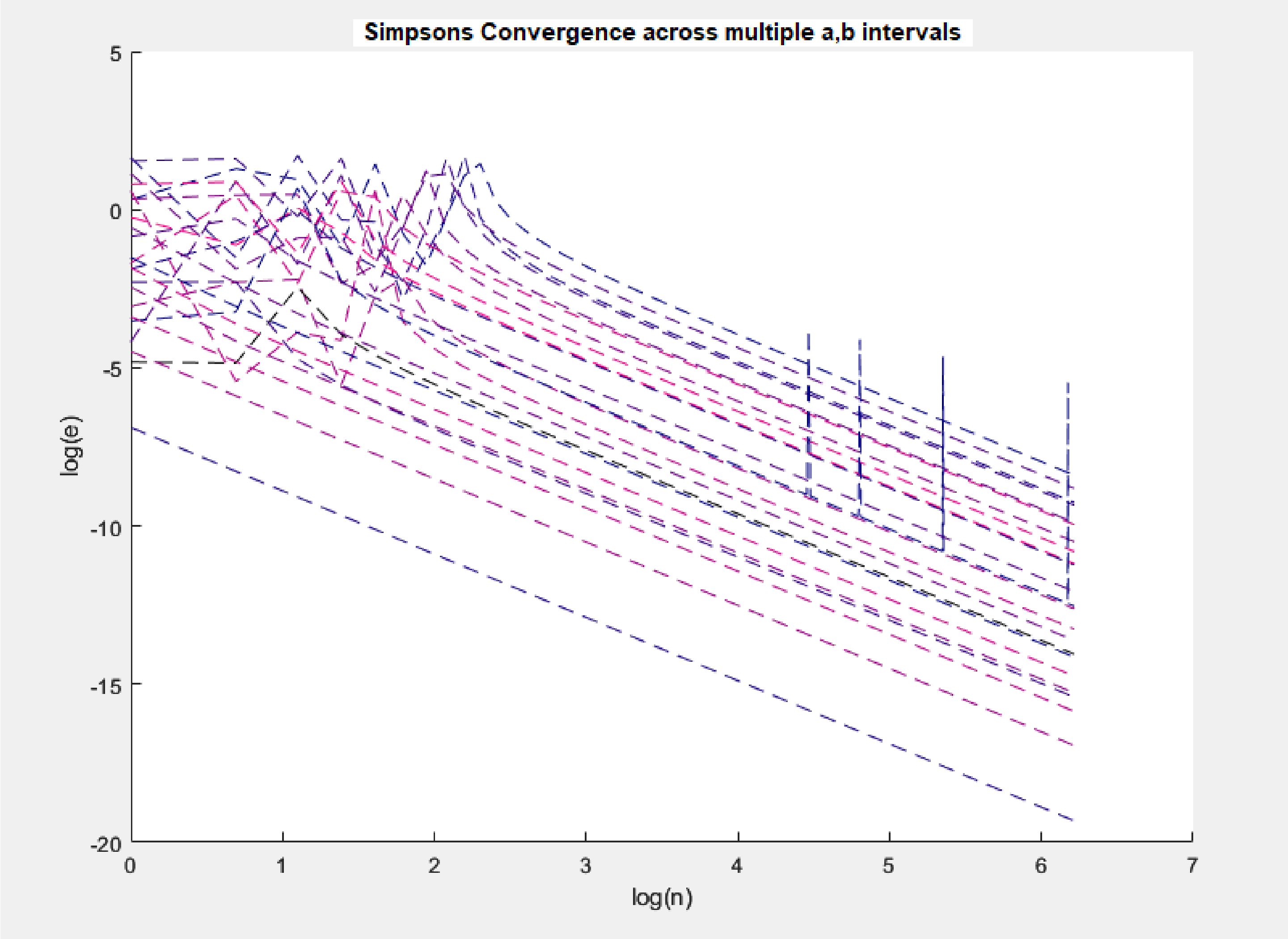
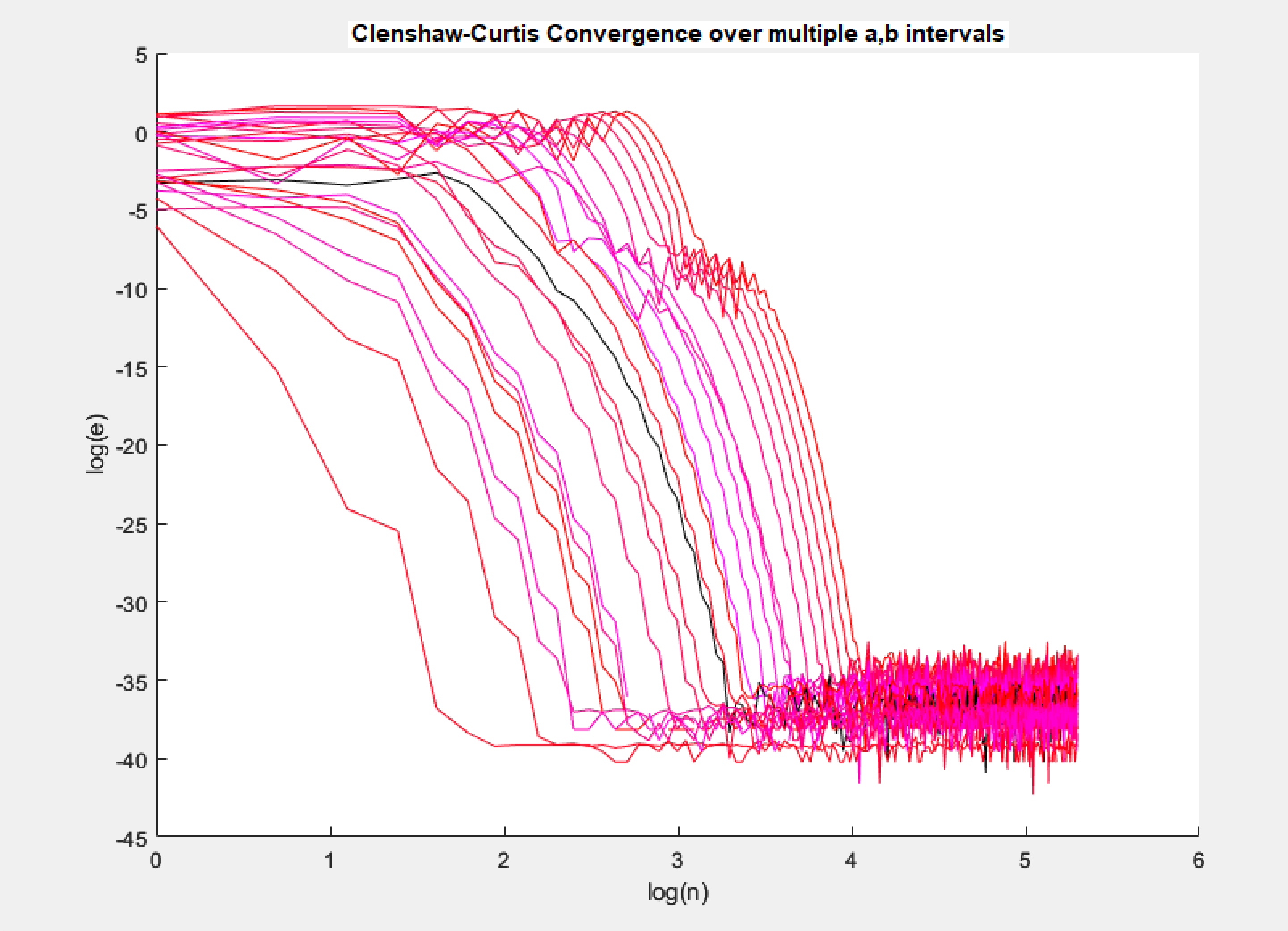
NumComp - Fall 2022
Project #11

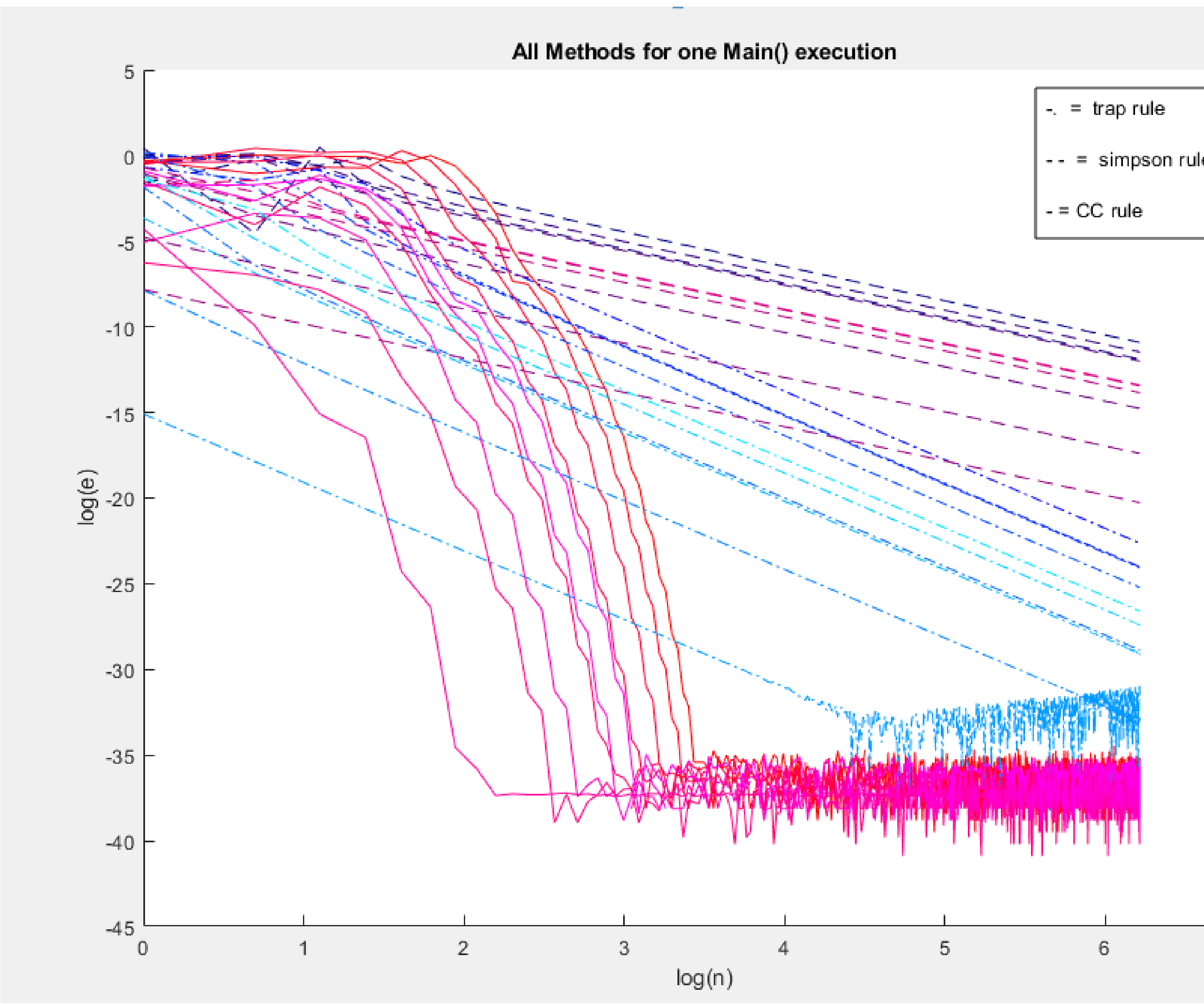
Due –
Isaiah Thomas

Plots!
The lighter the color, the smaller the interval of (a, b)




Simpsons Convergence across multiple a,b intervals
5
0
-5
-10
-15
-20
log(e)
0
1
2
3
4
5
6
7
log(n)


Clenshaw-Curtis Convergence over multiple a,b intervals
log(e)
5
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
log(n)
0
1
2
3
4
5
6


All Methods for one Main() execution
-. = trap rule
- - = simpson rul
- = CC rule
log(e)
log(n)
5
0
-5
-10
-15
-20
-25
-30
-35
-40
-45
0
1
2
3
4
5
6

1. Identify the asymptotic regime to estimate the convergence rate of each method.

   All regimes depend on a, b values. However, the convergence rate for each method is constant. At my surprise, Trap's rule converges at a quicker rate than simp's, despite both being linear and simp's adding another dimension to the approximation.

   For CC's rule, the convergence is obviously spectral from the plots. Its regime is a helluva lot smaller than the linear regimes and converges much quicker than the other rules.

   First, my initial thought was how changing interval distance and points impacted the asymptotic regime. Each method has some "nice" starting points that shift the regime to the left and lowers initial n error.
   For the point $[-.1667, .1667]$, Simpson's and Trap's rule started out immediately with an incredibly precise guess and immediate convergence, while CC's began with higher error and begins converging at a later n. It seems that integrals on intervals that result close to 0 are better for trap rule. The less symmetric the interval of our plot of our desired function, the more poorly trap's initial guess is.

   On the topic of CC being poorer for initial n values– it seems like a method that relies so heavily on polynomial interpolation reacts poorly to low n values. This makes sense as the function we are studying is differential heaven. The more rate and concavity changes, the more degrees the polynomial needs to reach decent accuracy.

   Again, this seems to all come back to Taylor series logic– how polynomial interpolant accuracy is entirely dependent on each value of each level of differentiation. Smoother, more symmetric functions / intervals appear to be better suited for Simp's and Trap's, while CC seems to be more suited for less symmetric, high rate of change functions.

   Overall, depends on how much n you can work with. If you can do as much n as you want, you should probably use CC. If you're limited in your n, simps and traps are prolly better.

Code!

```
function y = fd(x)

    y = cos(3 * pi * x);

end


function y = fint(x)

    y = sin(3 * pi * x) * 1/ (3 * pi);

end


function intapprox = traprule(a, b)

%(f(a) - f(b))/2 * (b - a)

    intapprox = (    (fd(a) + fd(b))/2   ) * (b - a);

end


function intapprox = simps(a, b)

    % (b-a)/6    *    ( fa -  4*f((a + b)/2) + f(b) )


    intapprox = (fd(a) +     4 * fd(  (a + b)/2 )   +  fd(b)  ) * ((b - a)/6);


end
```

```
function [x,w]=fclencurt(N1,a,b)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% fclencurt.m - Fast Clenshaw Curtis Quadrature
%
% Compute the N nodes and weights for Clenshaw-Curtis
% Quadrature on the interval [a,b]. Unlike Gauss
% quadratures, Clenshaw-Curtis is only exact for
% polynomials up to order N, however, using the FFT
% algorithm, the weights and nodes are computed in linear
% time. This script will calculate for N=2^20+1 (1048577
% points) in about 5 seconds on a normal laptop computer.
%
% Written by: Greg von Winckel - 02/12/2005
% Contact: gregvw(at)chtm(dot)unm(dot)edu
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
N=N1-1; bma=b-a;
c=zeros(N1,2);
c(1:2:N1,1)=(2./[1 1-(2:2:N).^2 ])'; c(2,2)=1;
f=real(ifft([c(1:N1,:);c(N:-1:2,:)]));
w=bma*([f(1,1); 2*f(2:N,1); f(N1,1)])/2;
x=0.5*((b+a)+N*bma*f(1:N1,2));



function intapprox = ccrule(a, b, npoints)
    [x,w] = fclencurt(npoints, a,b);
    intapprox = 0;
    for i = 1:1:size(x)

        intapprox = intapprox + fd( x(i) ) * w(i);
    end
    sprintf('evlautions:␣%i', i)
end
```

```
function [cerror, serror, terror] = generror(a,b, npoints)

    ccapprox = ccrule(a,b, npoints);
    sapprox = 0;
    tapprox = 0;
    xstep = (b - a)/npoints;
    k = 1;

    %start evlautations at a, a + step,
    % each iteration goes above a step
    % and then back one ti b is reached
    for i = a + xstep  :xstep:  b
        %sprintf('evlautiing: (%f - %f)', i - xstep, i)
        curs = simps(i - xstep, i  );
        curt = traprule(i - xstep, i);
        %sprintf('simp and trap approx: (%f , %f)', curs, curt)

        %sum the individual approximations
        sapprox = sapprox + curs;
        tapprox = tapprox + curt;
        k  = k + 1;
    end
    %sprintf('evlautions: %i', k-1)
    %calculate true values from true integral
    intruth = fint(b) - fint(a);
    cerror = abs(ccapprox - intruth);
    serror = abs(sapprox - intruth);
    terror = abs(tapprox - intruth);
end
```

```
function out1 = main(maxn, a, b)
    k = 0;
    %this is for testing multiple as and bs
    %parameters for plots were:
    %   n = 500
    %   [a,b] = -.1667,.1667
    %   [a,b] = -1,1
    %   [a,b] =-pi, pi
    %   [a,b] = 0,.1
    %step a and b at arbitrary linear rates
    for l = a:.25:b
        l = l *1.03;
        ns = zeros(1,maxn);
        cregime = zeros(1,maxn);
        tregime = zeros(1,maxn);
        sregime = zeros(1,maxn);

        for i = 1: 1: maxn
            ns(i) = i;
            [cregime(i), tregime(i), sregime(i)] = generror(l,b,i);

        end
        %decrease b by some arbitrarty amount
        b= b *.93;

        %plot
        hold on;
        %plot(log(ns), log(sregime),  LineStyle='--', Color=[k/10 , 0, .5]);
        %plot(log(ns), log(cregime), Color=[1 , 0, k/10]);
        plot(log(ns), log(tregime), LineStyle='-.', Color=[0 , k/10, 1]);
        %plot(ns, sregime);

        k = k + 1;

    end
    hold off;
    out1 = 1;
end
```