**GENERAL LOGISTIC FUNCTION**

Şeyma Bayrak

**Abstract**

The code “GlogF.m” runs a program to display General Logistic Function. This word file reflects how GLF seems when the variables or parameters are changed. It all starts with fixing the inflection point and defining the bisection points, and calculates the other parameters which all mathematically depend on those points. Furthermore, it brings a nice aspect of finding out stationary points along the GLF curve.

What happens if I change “nu”? What does “singularity” mean?

clear all;

close all;

nu=-1:0.01:1;

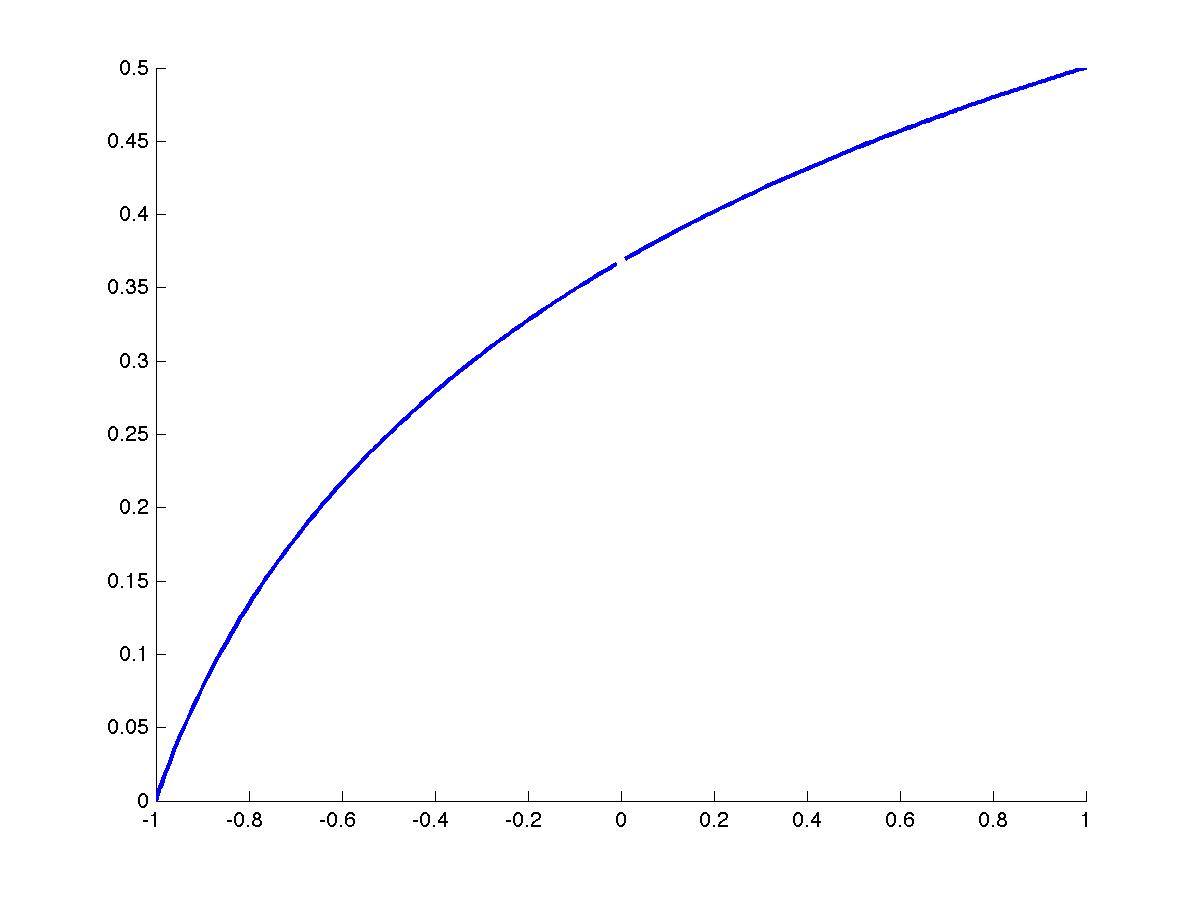
yi=(1+nu).^(-1./nu);

figure;

hold on

plot(nu,yi,'b','LineWidth',2);

hold off



Let us have a look at different constants multiplied by 1/nu:

clear all;

close all;

nu=-1:0.01:1;

yi1=(1+nu).^(-1./nu);

yi2=(1+nu).^(-1./(nu/2));

yi3=(1+nu).^(-1./(nu/3));

yi4=(1+nu).^(-1./(nu\*2));

figure;

hold on

plot(nu,yi1,'b','LineWidth',2);

plot(nu,yi2, 'k.', 'LineWidth',2);

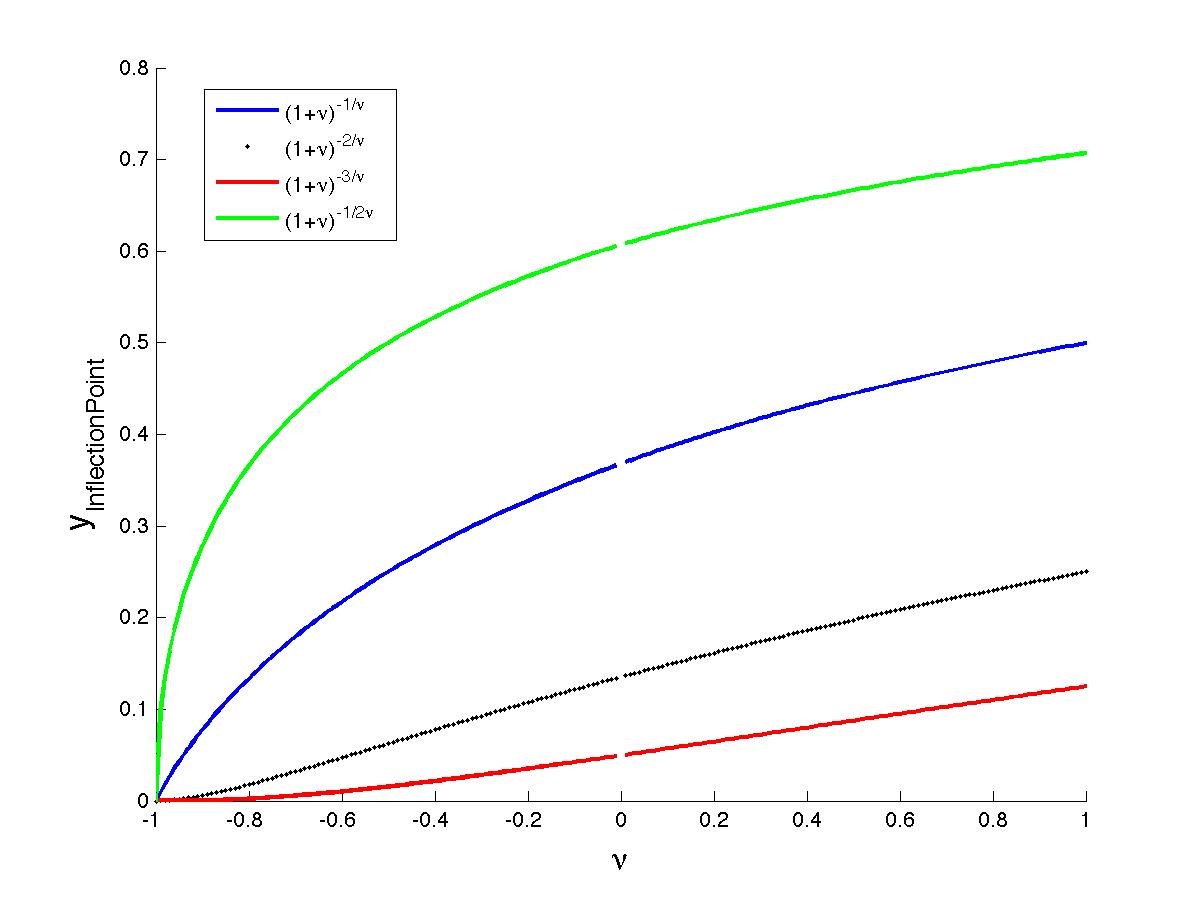
plot(nu,yi3, 'r', 'LineWidth',2);

plot(nu,yi4, 'g', 'LineWidth',2);

xlabel('\nu', 'FontSize', 16')

ylabel('y\_{InflectionPoint}', 'Fontsize',16)

hold off



clear all;

close all;

nu=-1:0.01:4;

yi=(1+nu).^(-1./nu);

% When /nu approaches to 0, limit of yi equals to 1/e (page 6!)

N=1000;

yi\_limit=(N/(N+1))^N;

figure;

hold on

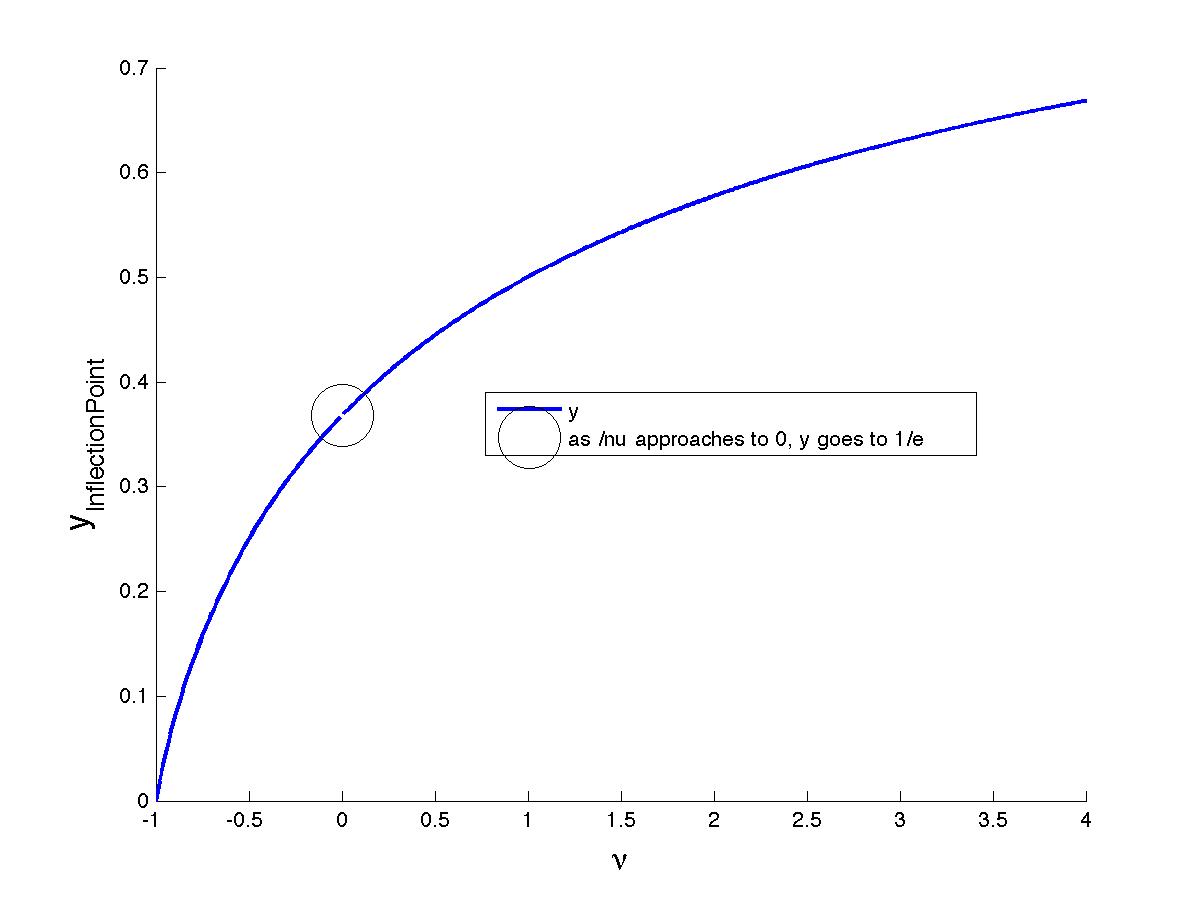
plot(nu,yi,'b','LineWidth',2);

plot(0,yi\_limit, 'ko', 'MarkerSize', 30)

xlabel('\nu', 'FontSize', 16')

ylabel('y\_{InflectionPoint}', 'Fontsize',16)

hold off



Note: please be aware that, as long as /nu is less than 0, then y is smaller than 1/e (~0.368), otherwise y is greater than 1/e.

At the inflection point (I choose what y point is), root of GLF is found. Now we know what /nu equals to. Note that command “fsolve” gives two roots having complex parts, so that I choose the real part of the first root as /nu1 = -0.5

Let us try to see how does GLF for different beta values look like.

x=0:.01:1;

a=1; % alpha constant

b=0.5:0.1:0.9; % different beta values

clr = 'rygbm';

figure;

for i=1:numel(b)

y=(1+exp(-a\*x+b(i))).^(-1/nu1);

hold on

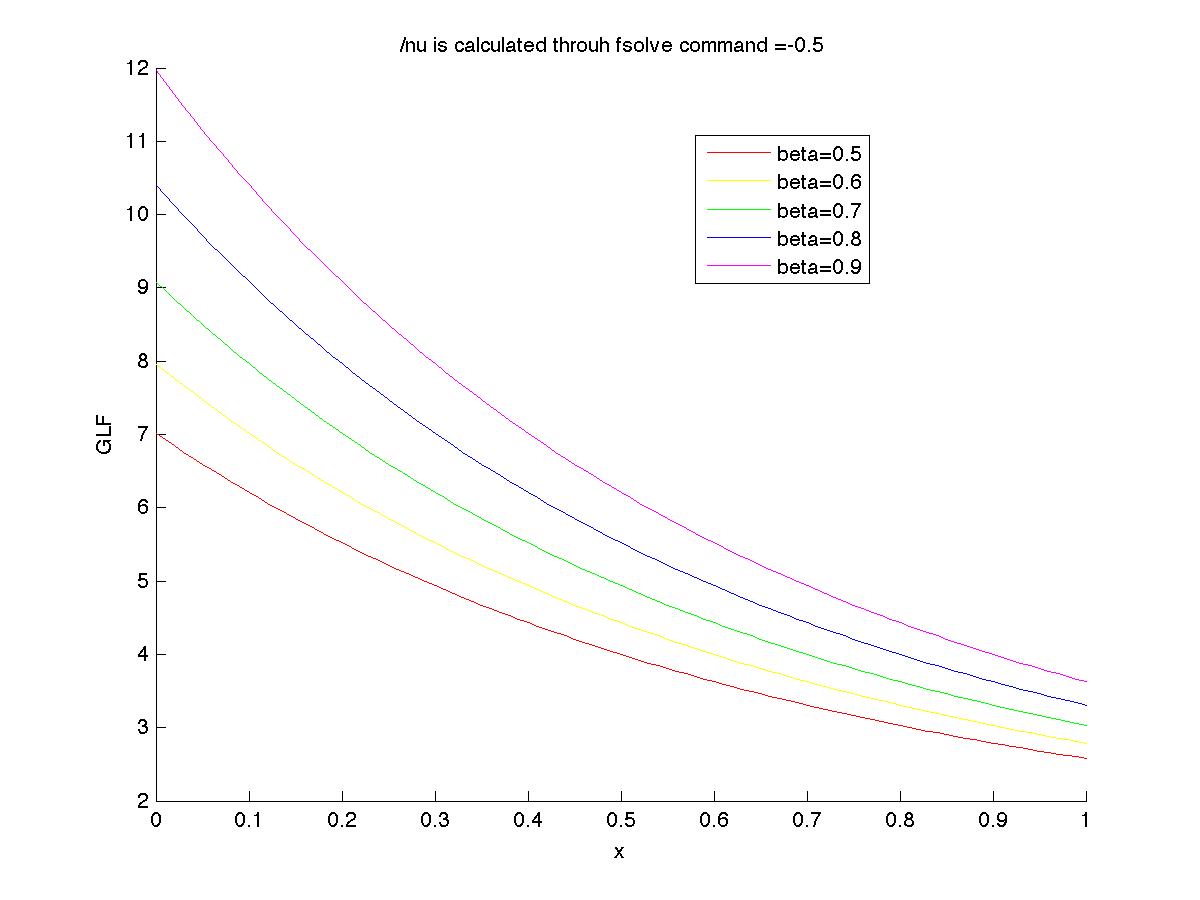
plot(x,y, clr(i))

xlabel('x')

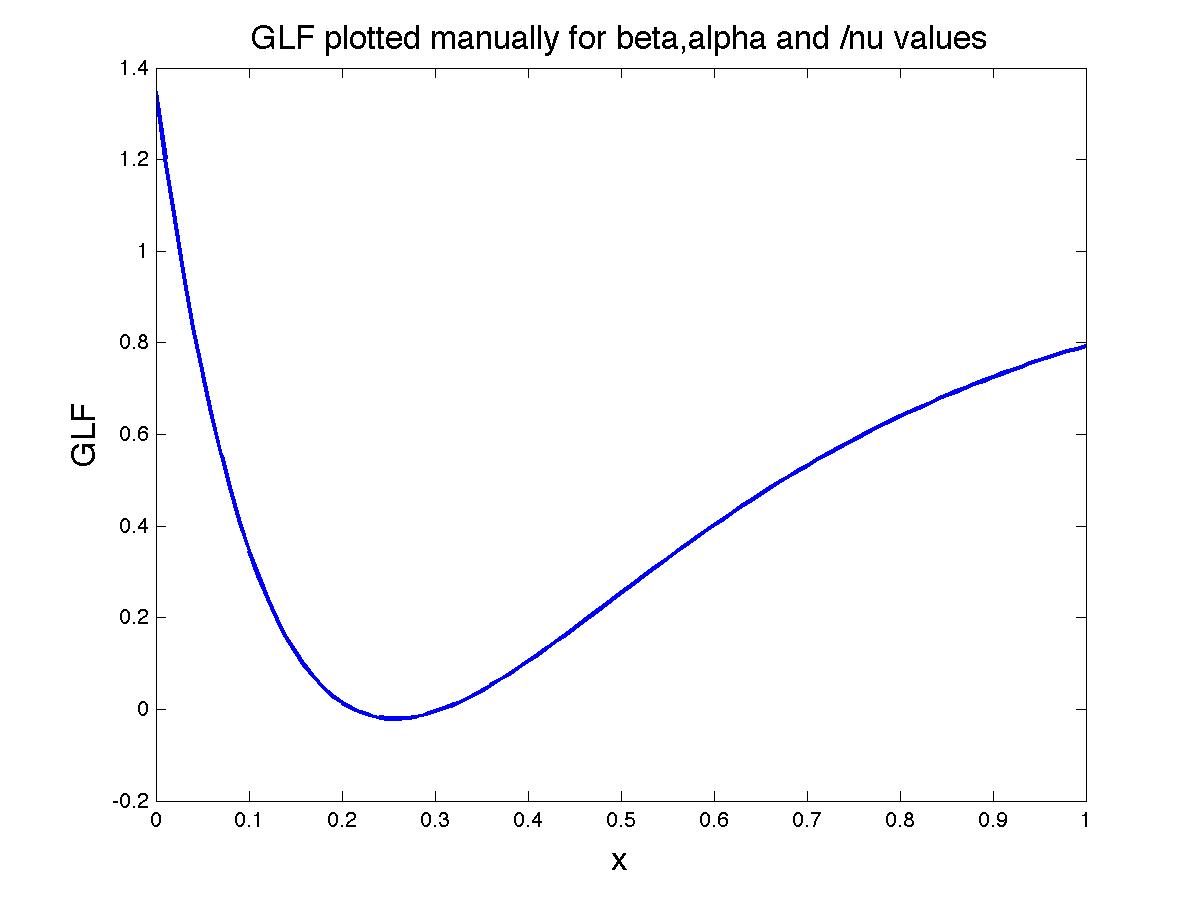
ylabel('GLF')

hold off

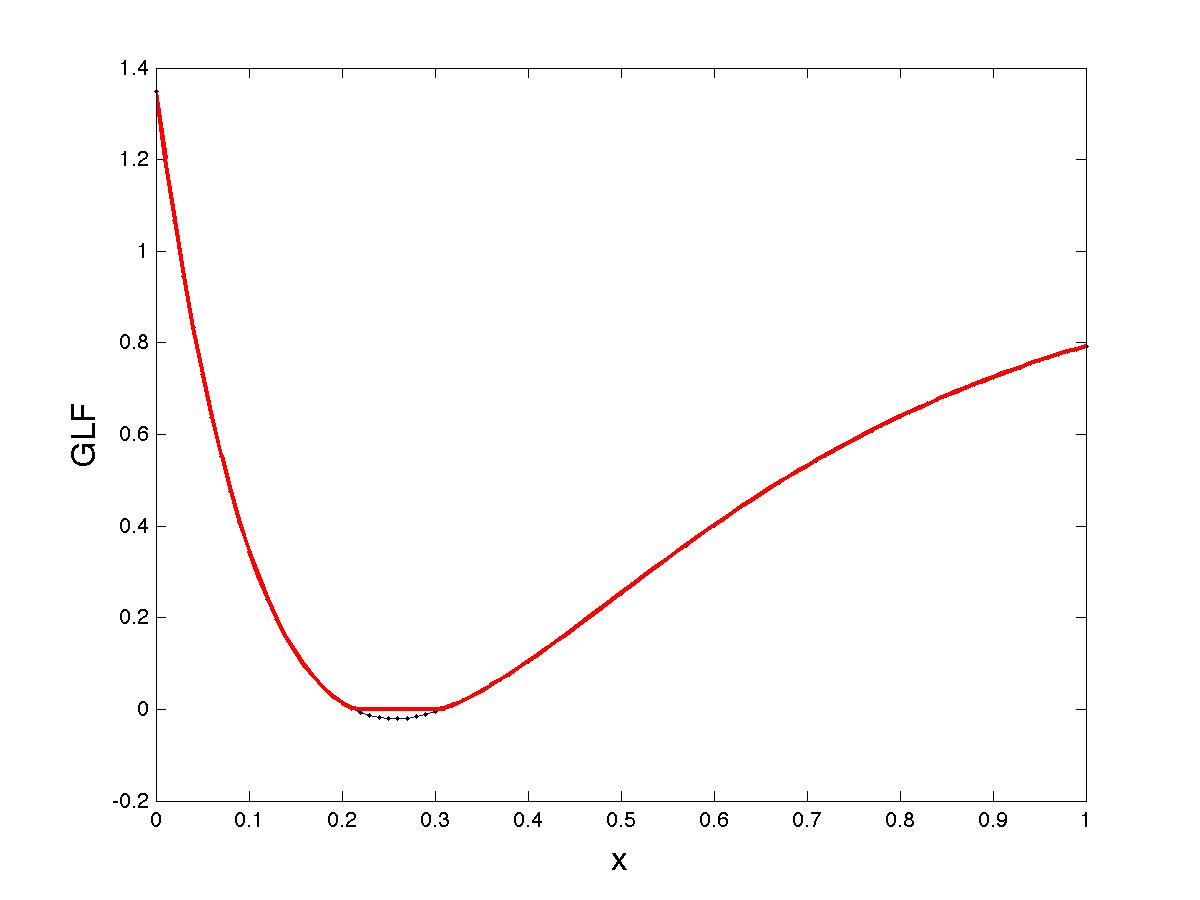
end



For the following values: beta=-3, alpha=0.8+3i, nu=-0.5, GLF is plotted as seen below.



Question: Why do I get a different graph, when plotted it as Achim did? The function GLF is defined somehow sophisticated. When I sort GLF as max(0, GLF), then I get the red graph. Hmmm… When GLF is negative, I set my values to 0.



Answer: The purpose is to ignore the GLF at the points where it is below zero, and also accept the whole points 0 up to the index where it is below 0.

This is the trick of above explanation:

function t=GLF(x, alpha, beta, nu)

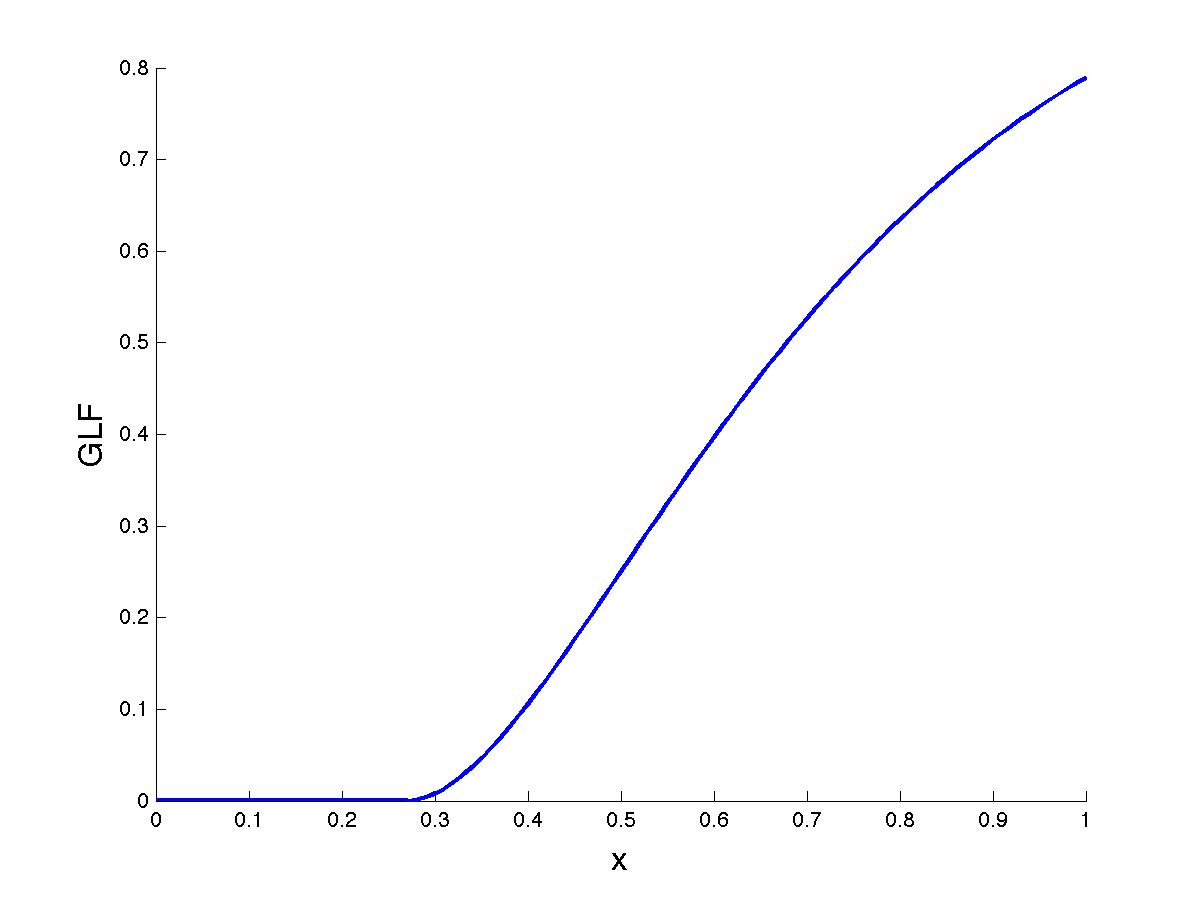
t=max(0, real((1+exp(-beta\*x+alpha)).^(-1/nu)));

tmin=min(t);

index=find(t==tmin);

t(1:index(1))=tmin\*ones(size(1:index(1)));

return;



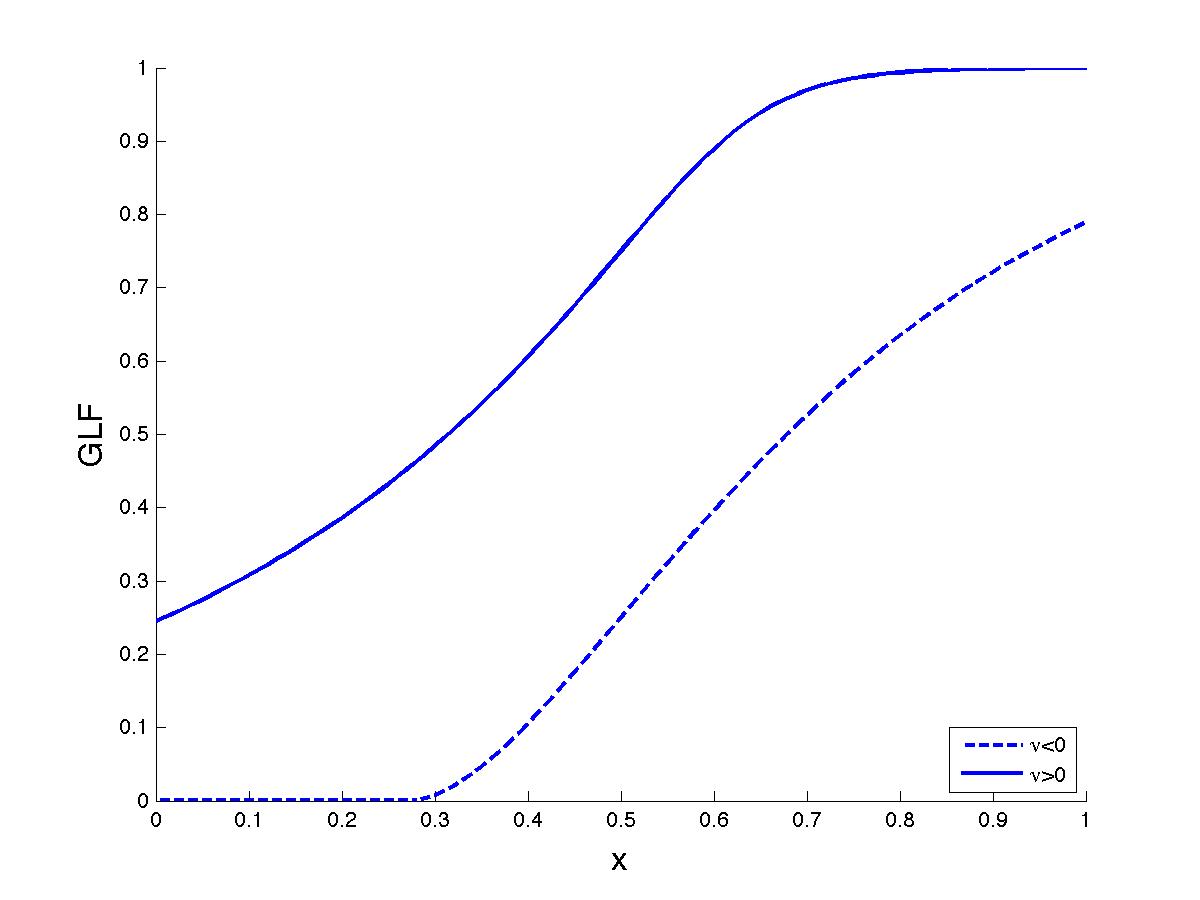
Now you can compare the above graph with the one on the previous page.

Let us repeat all procedures for another GLF plot, now assume that y\_inflection is > 1/e. The expected result for the /nu is to be greater than 0. When y\_inflection is < 1/e, /nu was smaller than 0.

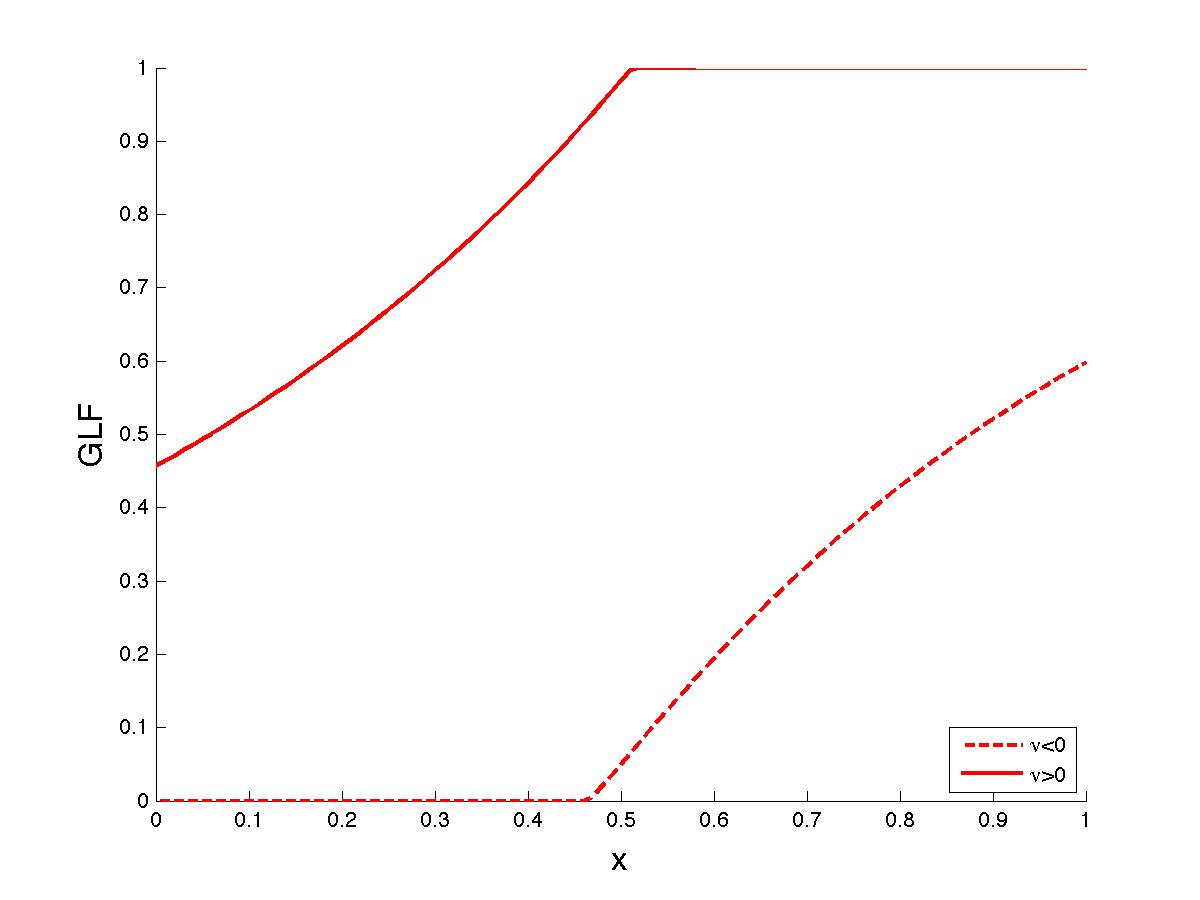
Yinfl2=0.75;

slope2=1.5;

Xinfl2=0.5;



Assume that Yinfl1=0.05 (dashed line) and Yinfl2=1 (solid line):



Now, let us add the bisection points, which means y\_half=0.5, all other variables are calculated through the formulas on page 6. Bisection points are exactly on the line that ([x], 0.5). However, the inflection points do not necessarily intersect with any shaded line.

Xhalf1=(alpha1-log(2^nu1-1))/beta1;

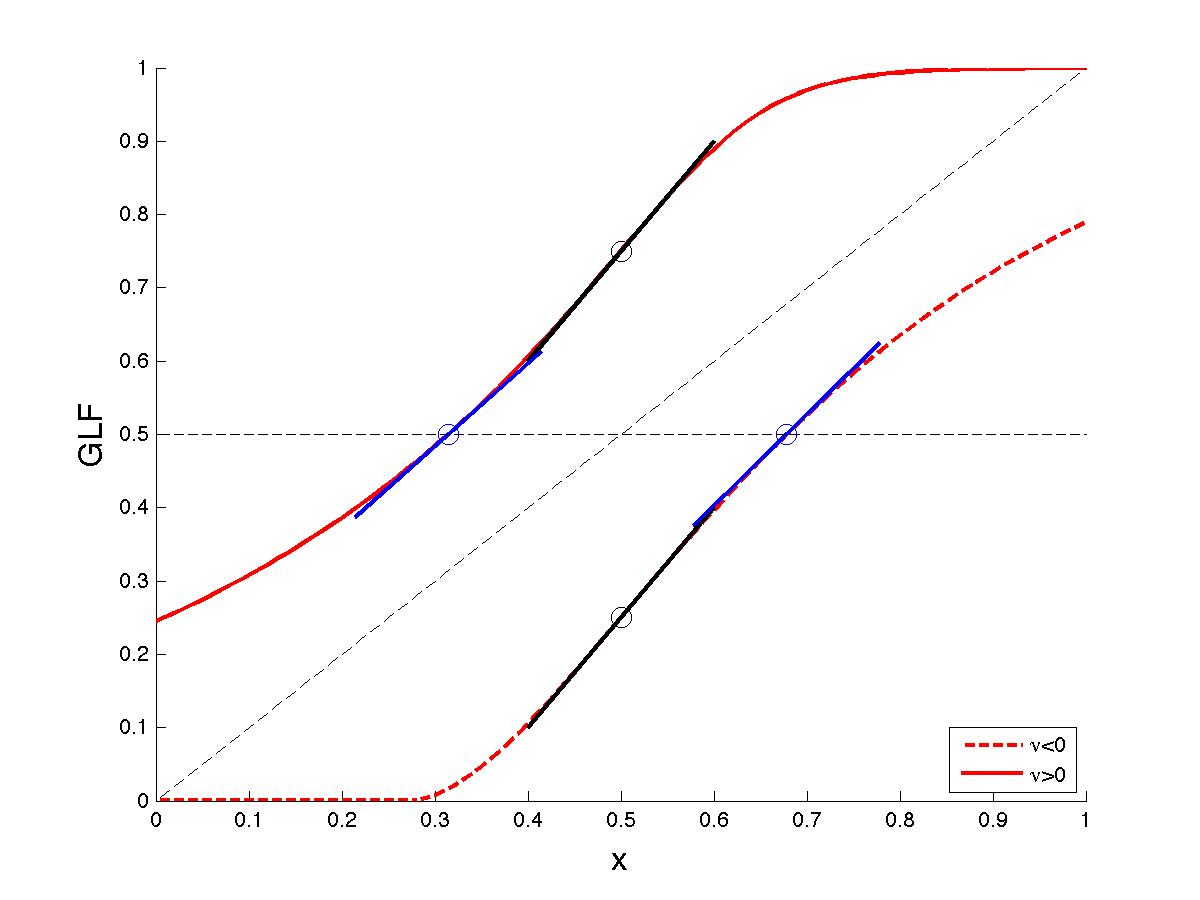
Yhalf1=0.5;

Hslope1=beta1\*(1-2^(-nu1))/(2\*nu1);

Xhalf2=(alpha2-log(2^nu2-1))/beta2;

Yhalf2=0.5;

Hslope2=beta2\*(1-2^(-nu2))/(2\*nu2);



How to find steady state points? Steady state points are where GLF is extremely close to x=y line. I try to find the roots of GLF, which satisfies that, GLF(x,alpha,beta,nu)-x =0. Of course the “fsolve” command plays a great role to find out estimated x values. The first important note is to think about the number of possible roots, since the sigmoidal GLF function could maximally 3 times intersect with x=y, it is necessary to guess 3 starting points inside “fsolve”. The second point is to check if found x values are close enough to x=y.

function ss=GLFss(xss, alpha,beta,nu)

ss=GLF(xss,alpha,beta,nu)-xss

end

xss=fsolve(@(x) GLFss(x,alpha,beta,nu), [0.01 0.5 0.99])

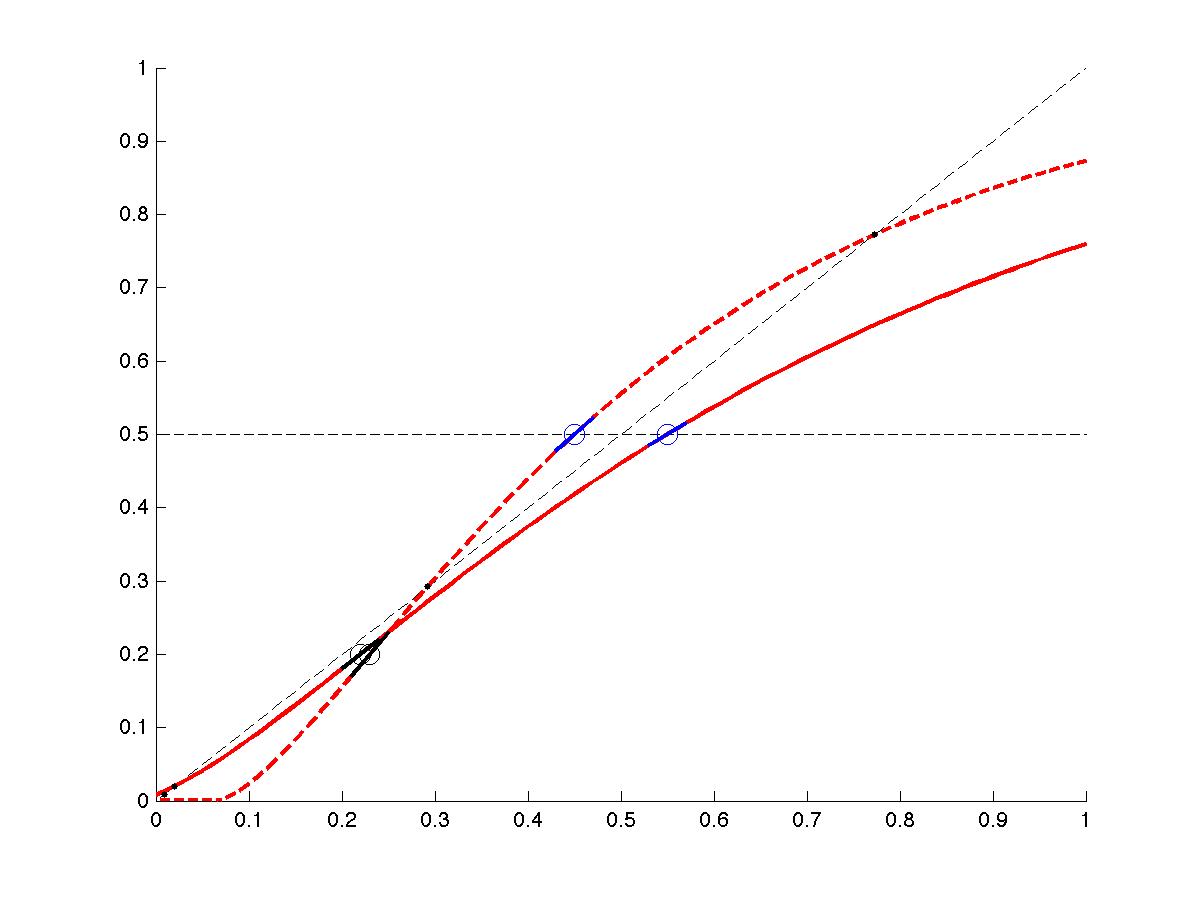
for i=1:numel(xss)

if abs(xss(i)-GLF(xss(i),alpha,beta,nu))<0.01

plot(xss(i), GLF(xss(i),alpha,beta,nu)) )

end

end



Next step: let us choose the inflection point on the y axes as 0.5, and plot GLF with different alpha values. We assume that alpha depends linearly on s, which is our imaginary stimulus.

slope=1.5;

si=linspace(0,1,5);

alpha=3.6 + (2.4-3.6)\*si;

figure;

hold on

plot(x,.5\*ones(size(x)),'k--');

plot(x,x,'k--');

clr='rygbm';

for i=1:numel(si)

plot(x,GLF(x,alpha(i),beta,nu), clr(i), 'LineWidth',2)

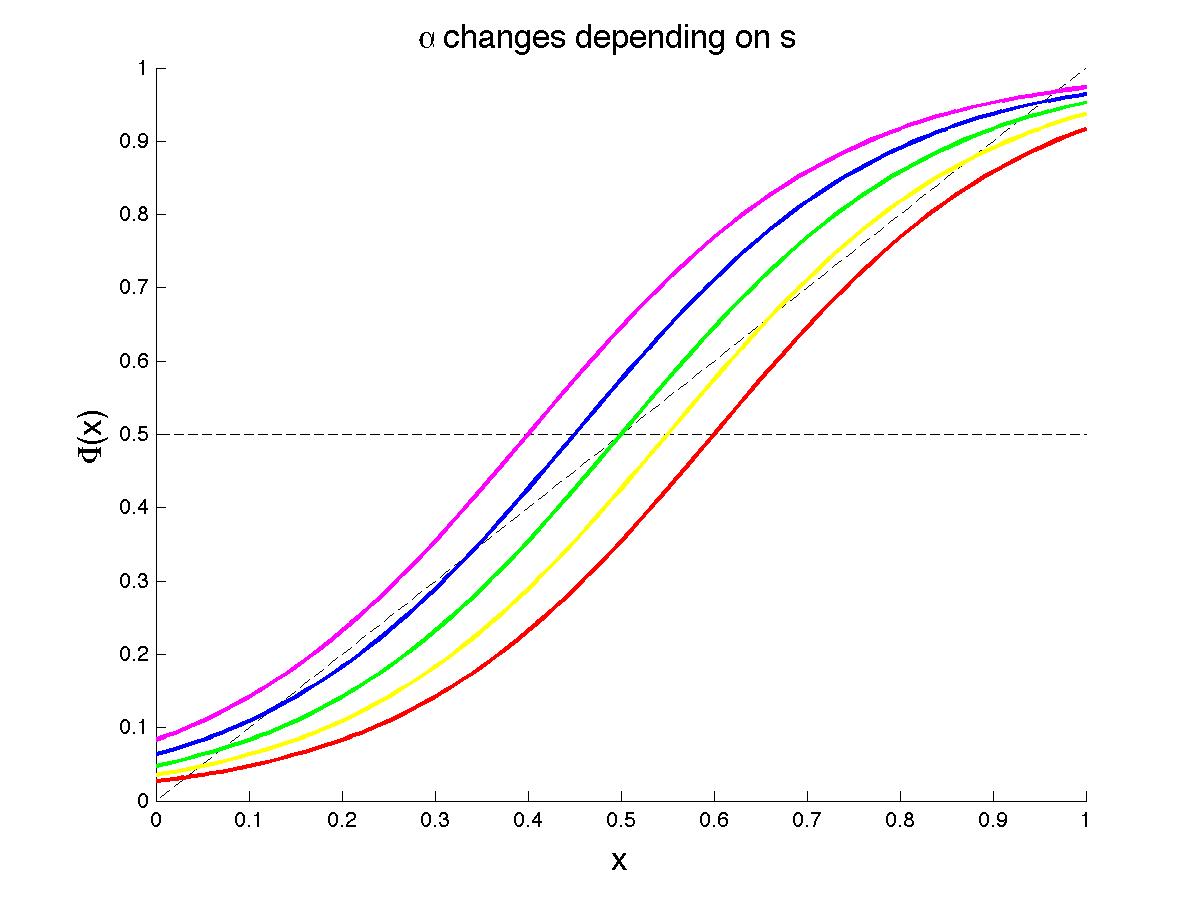
end

xlabel( 'x', 'FontSize', 16);

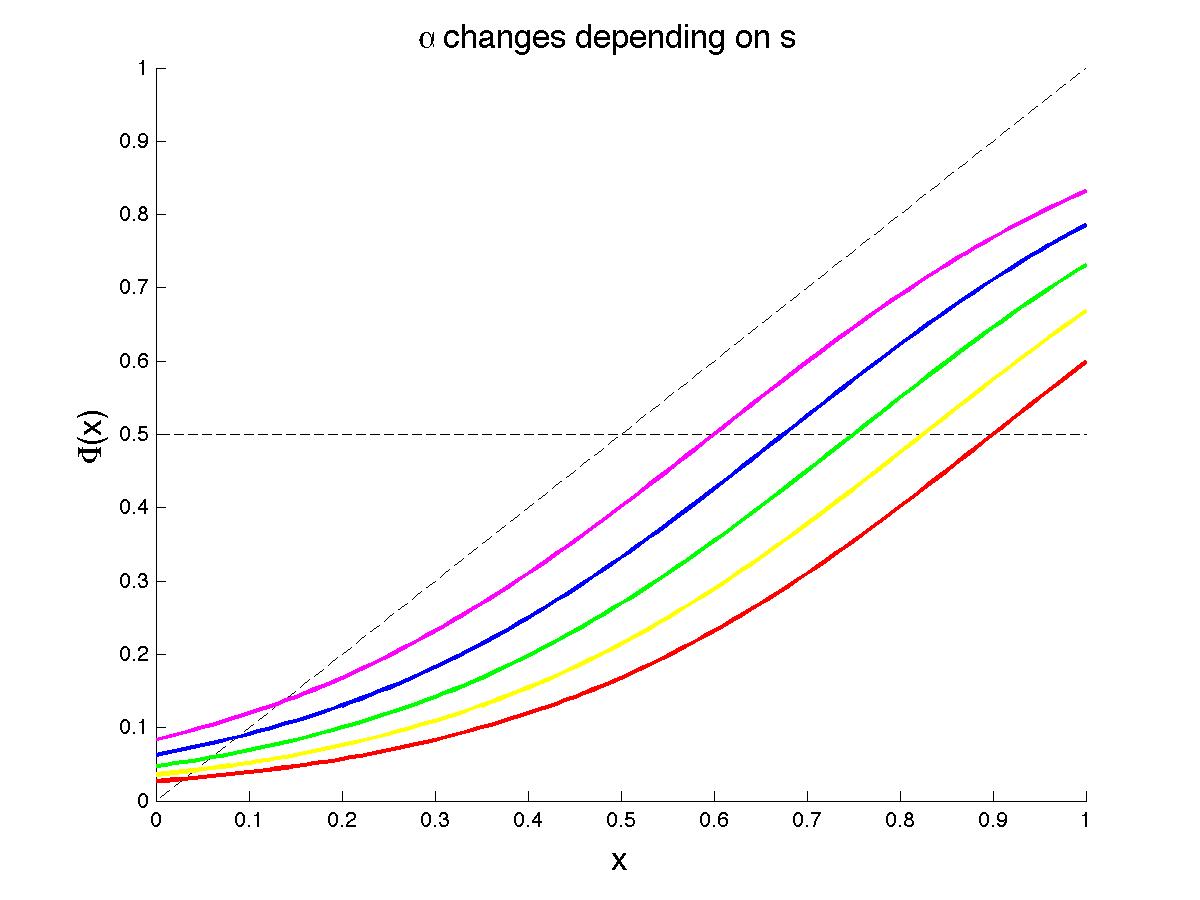
ylabel( '\Phi(x)', 'FontSize', 16);

title('\alpha changes depending on s', 'FontSize', 16)

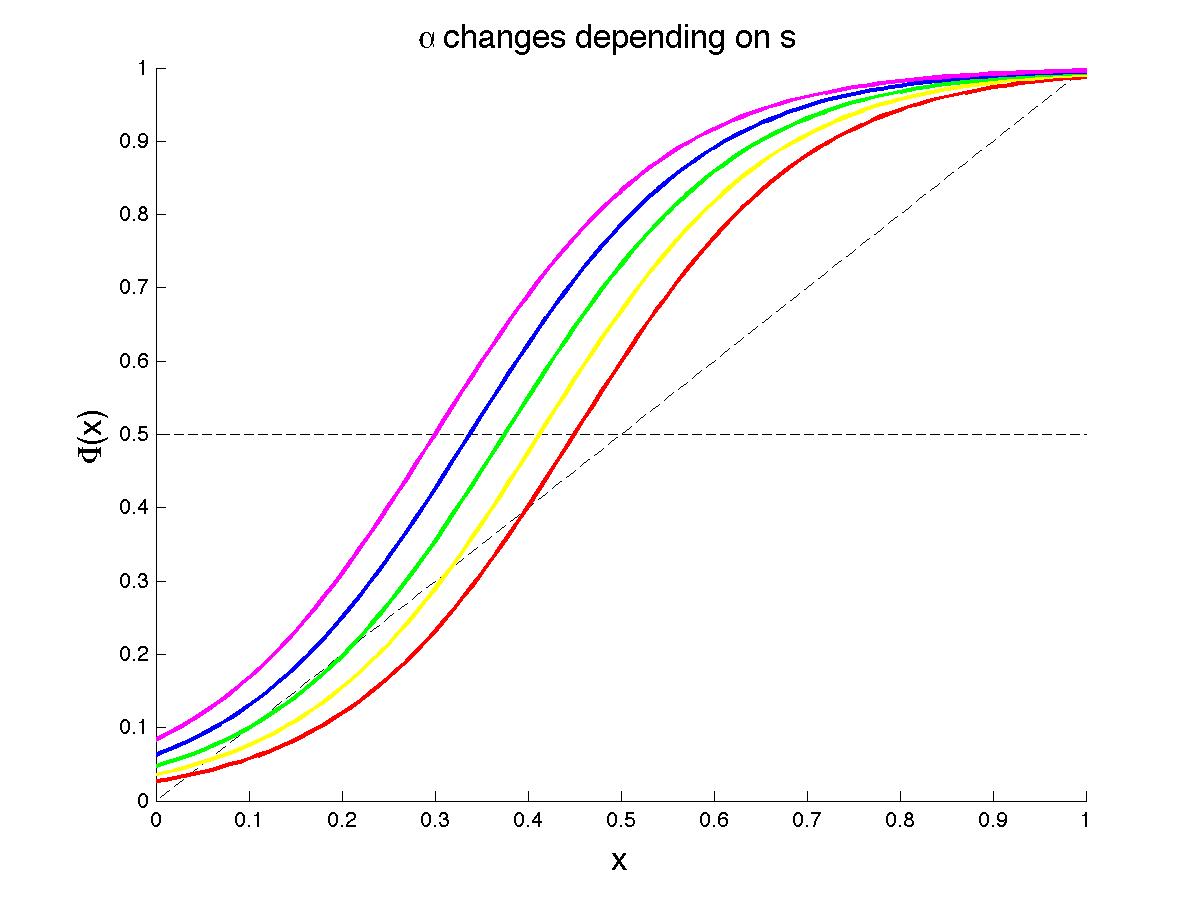
hold off



Just for fun, let us try different slopes at the inflection point.

Slope=1 >>>>>>>>>

Slope=2 >>>>>>>>>>>>



Which points have I fixed? >> y\_inflection point, slope at the inflection.

Now, instead of assigning alpha to a constant value, I would like to make it depended on a parameter called “s” as the following:

%%%%Inputs

Yinf=0.5;

slop=1.5;

si=linspace(0,1,5);

alpha=3.6 + (2.4-3.6)\*si;

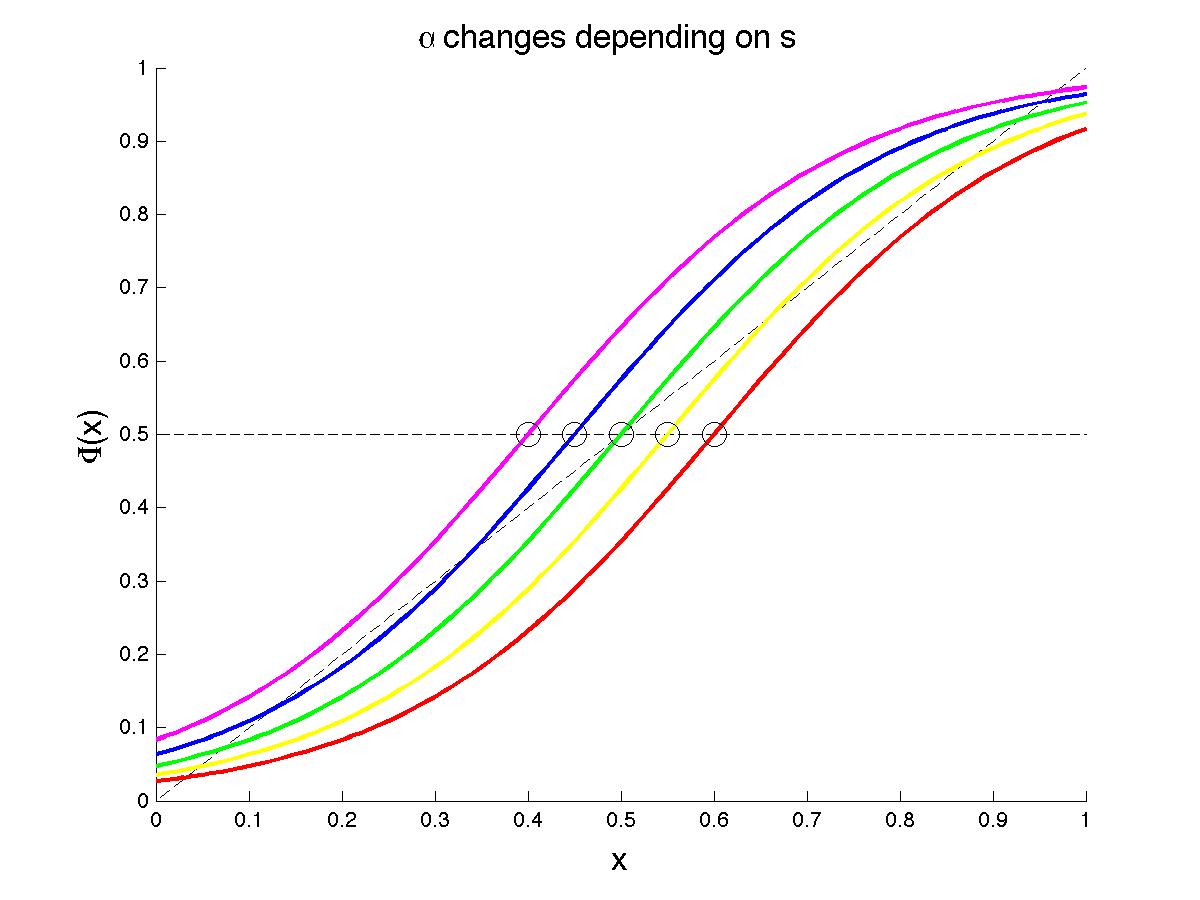
%%%%% Calculations

dummy=fsolve(@(nu) GLFinfl(nu,Yinf), [0.1 10], optimset('Display', 'Off'));

nu=real(dummy(1));

beta=slop\*(1+nu)^(1+1/nu);

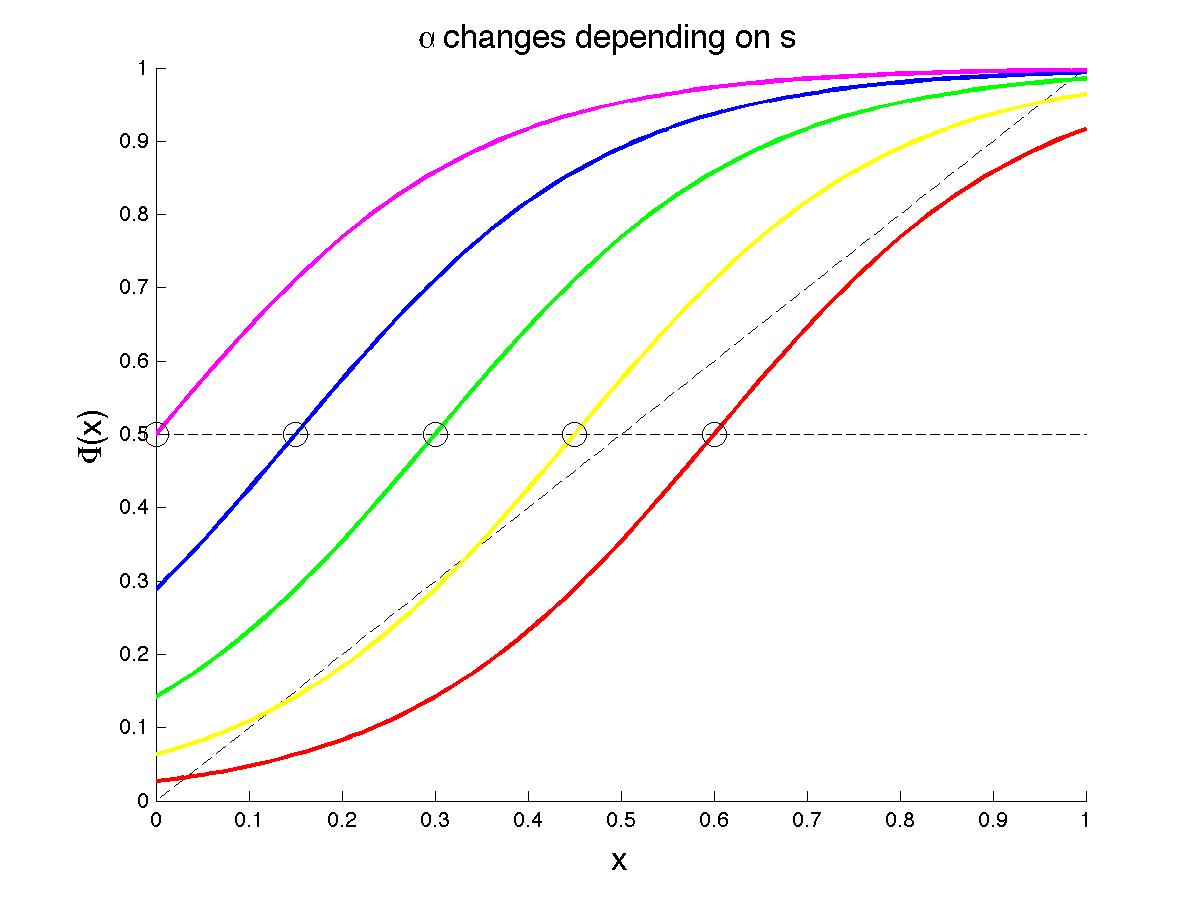
Xinf=(alpha-log(nu))/beta;



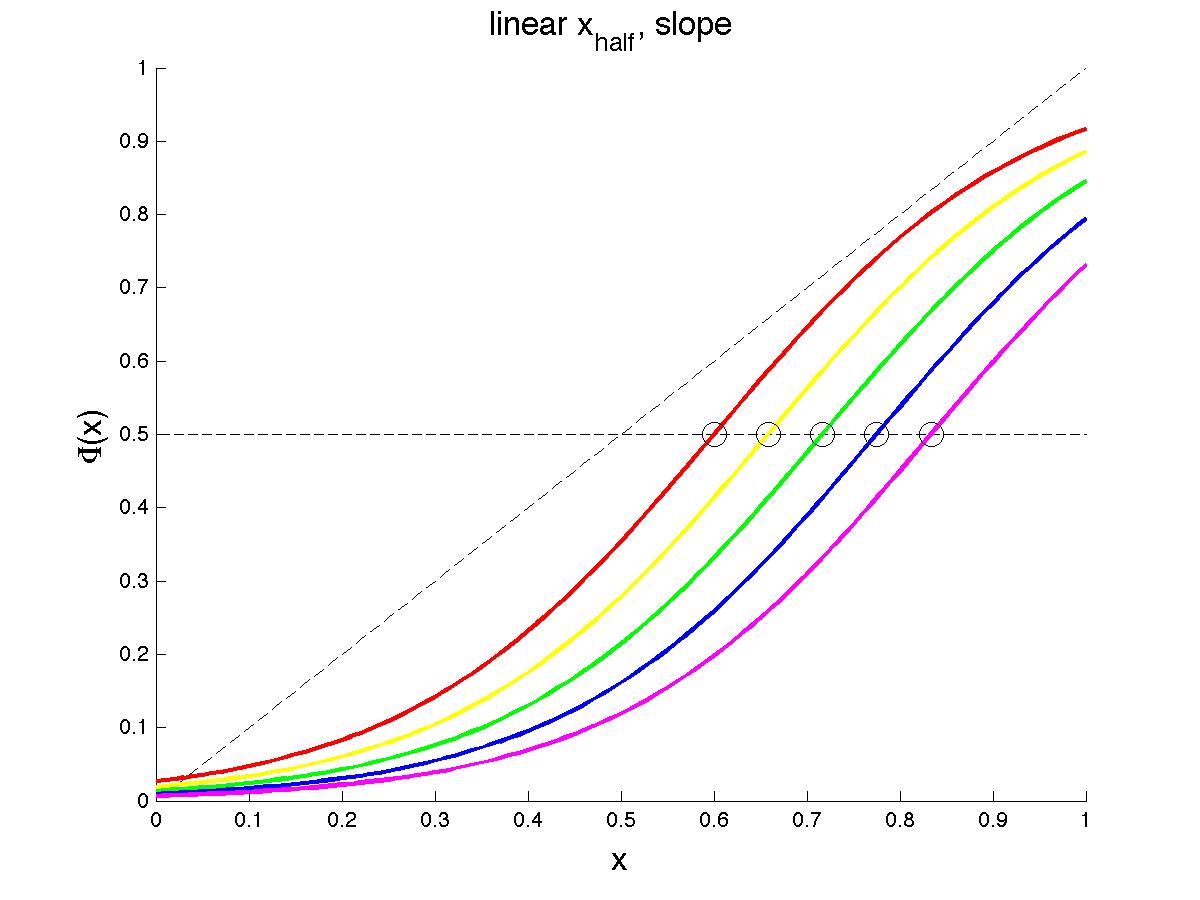
Note that, s is a vector of 5 elements, therefore alpha too. For the case of 5 different alpha values, we got 5 different graphs. Please pay attention to the inflection points at y=0.5 axis.

Let us play a bit with alpha. Originally: alpha=3.6 + (2.4-3.6)\*si;

Change alpha into : alpha=3.6 + (0-3.6)\*si;

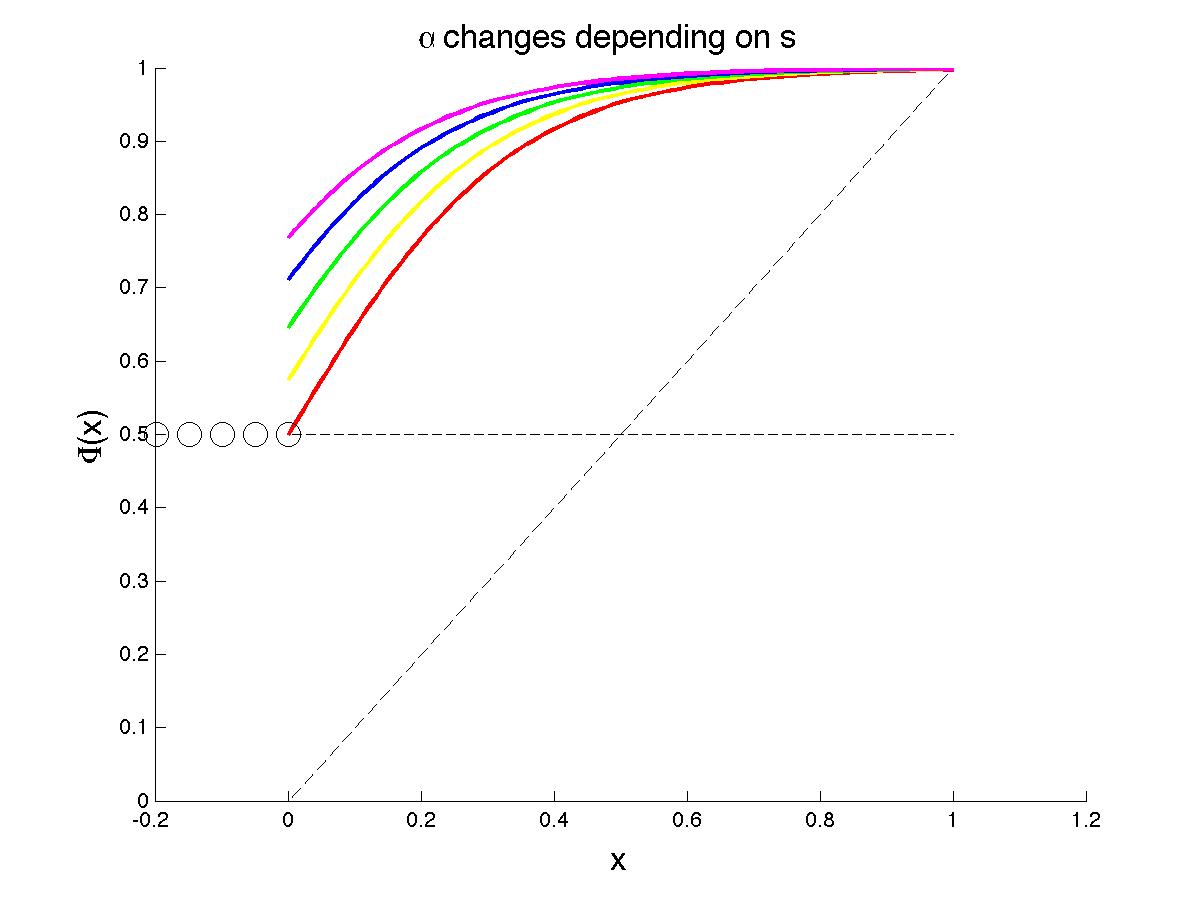


Change alpha into : alpha=3.6 + (5-3.6)\*si;

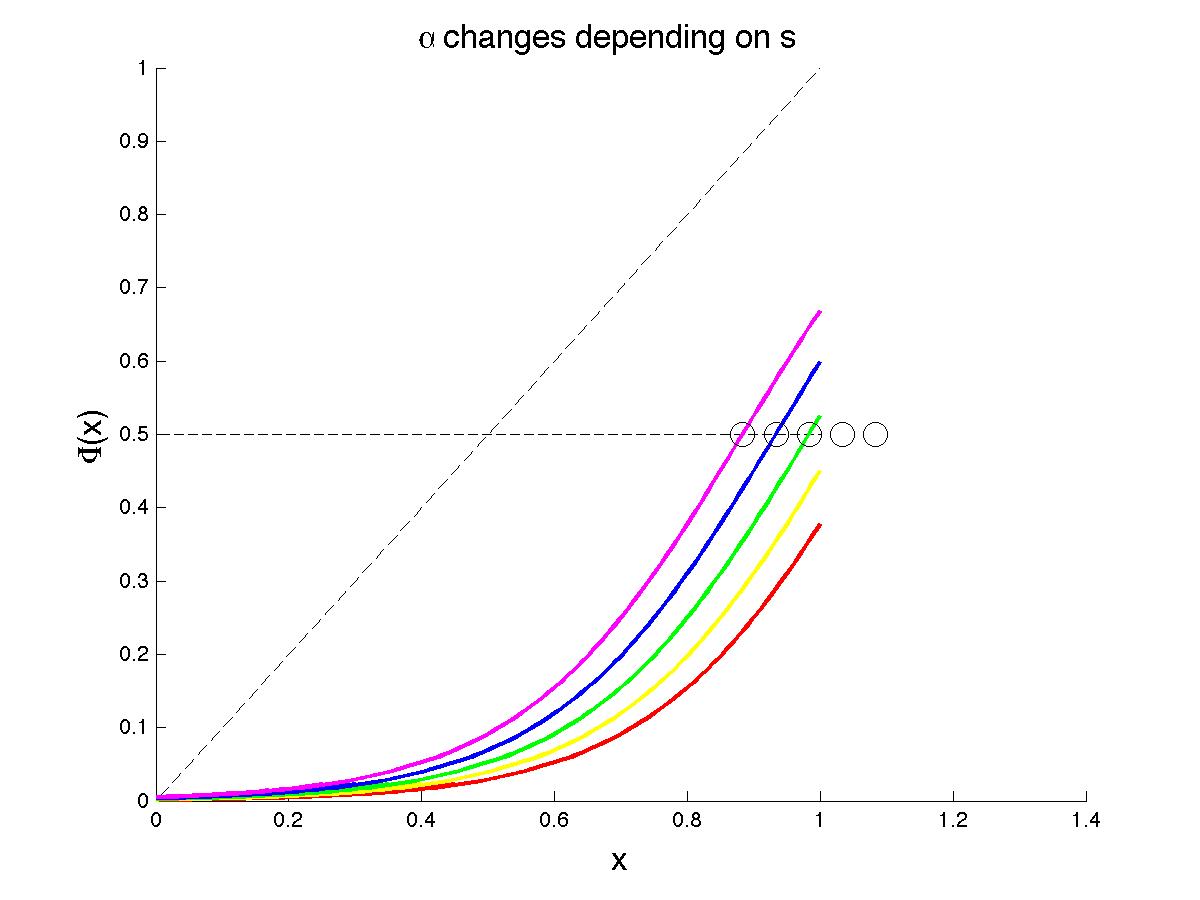


Something awkward: (Originally: alpha=3.6 + (2.4-3.6)\*si;)

alpha= 0 + (2.4-3.6)\*si;



alpha= 6.5 + (2.4-3.6)\*si;



Question: How to be sure about choosing proper alpha – s dependence?

Another approach to calculate alpha. This time, let us say that X\_half depends on s.

%%%% Inputs

Yinf=0.5;

slop=1.5;

si=linspace(0,1,5);

Xhalfi=0.6 - 0.5\* si;

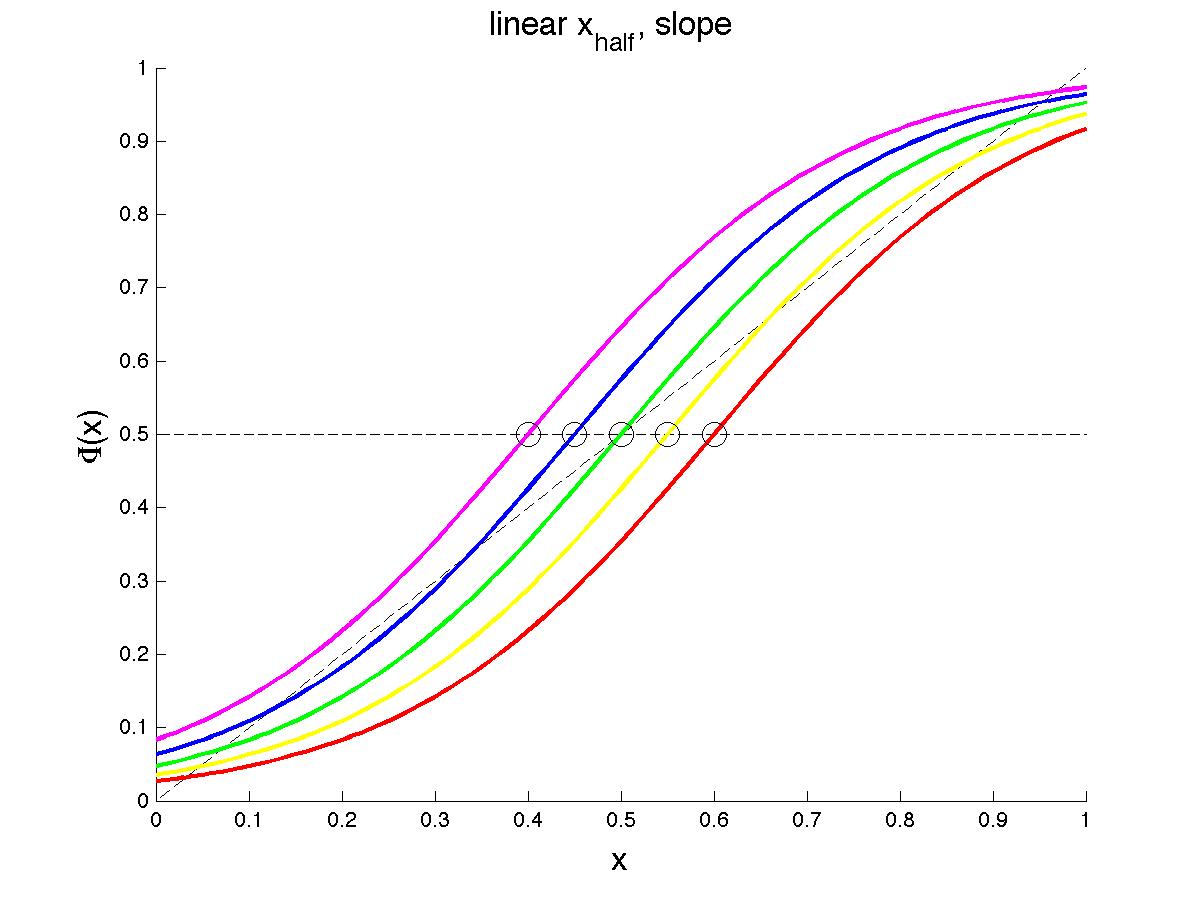
%%%% calculations

dummy=fsolve(@(nu) GLFinfl(nu,Yinf), [0.1 10], optimset('Display', 'Off'));

nu=real(dummy(1));

beta=slop\*(1+nu)^(1+1/nu);

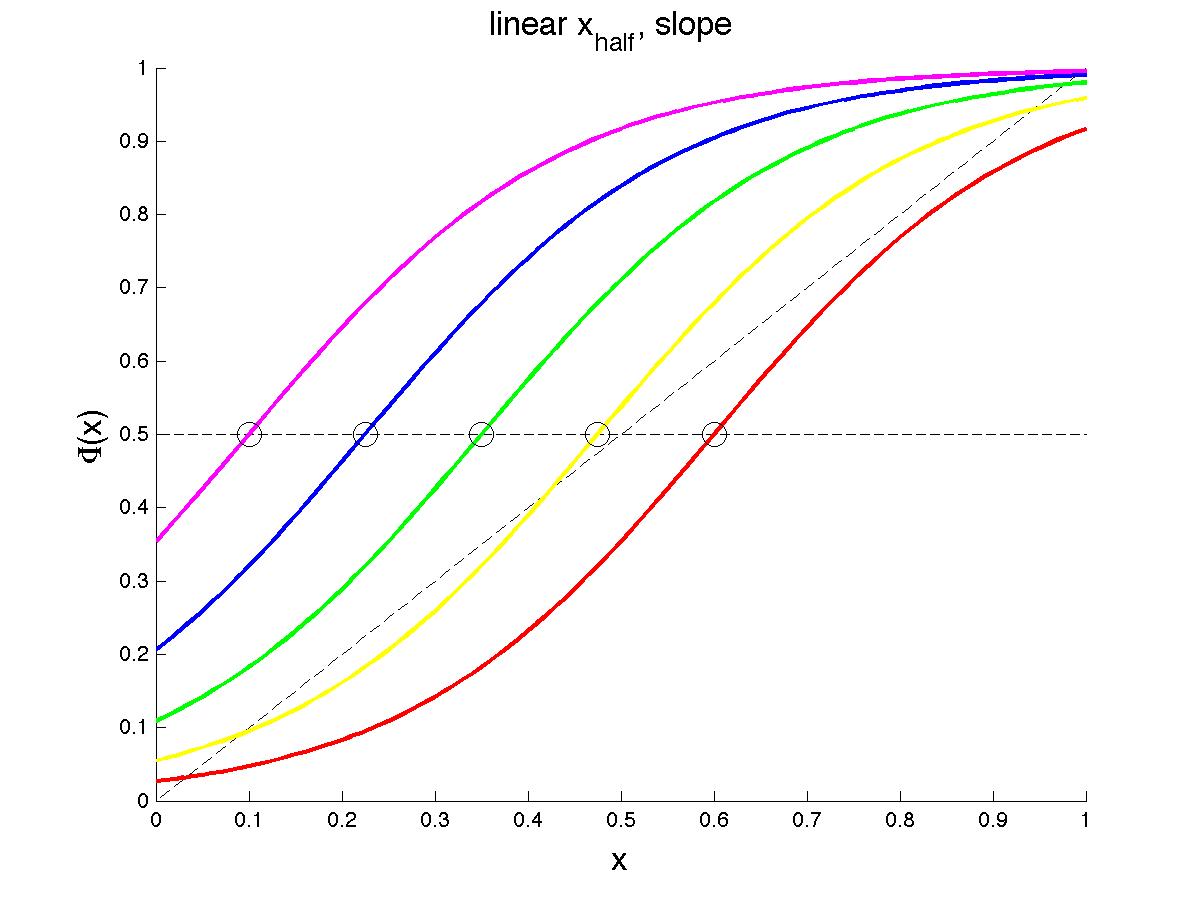
alpha(i) = beta \* Xhalfi(i) + log(2^nu - 1);



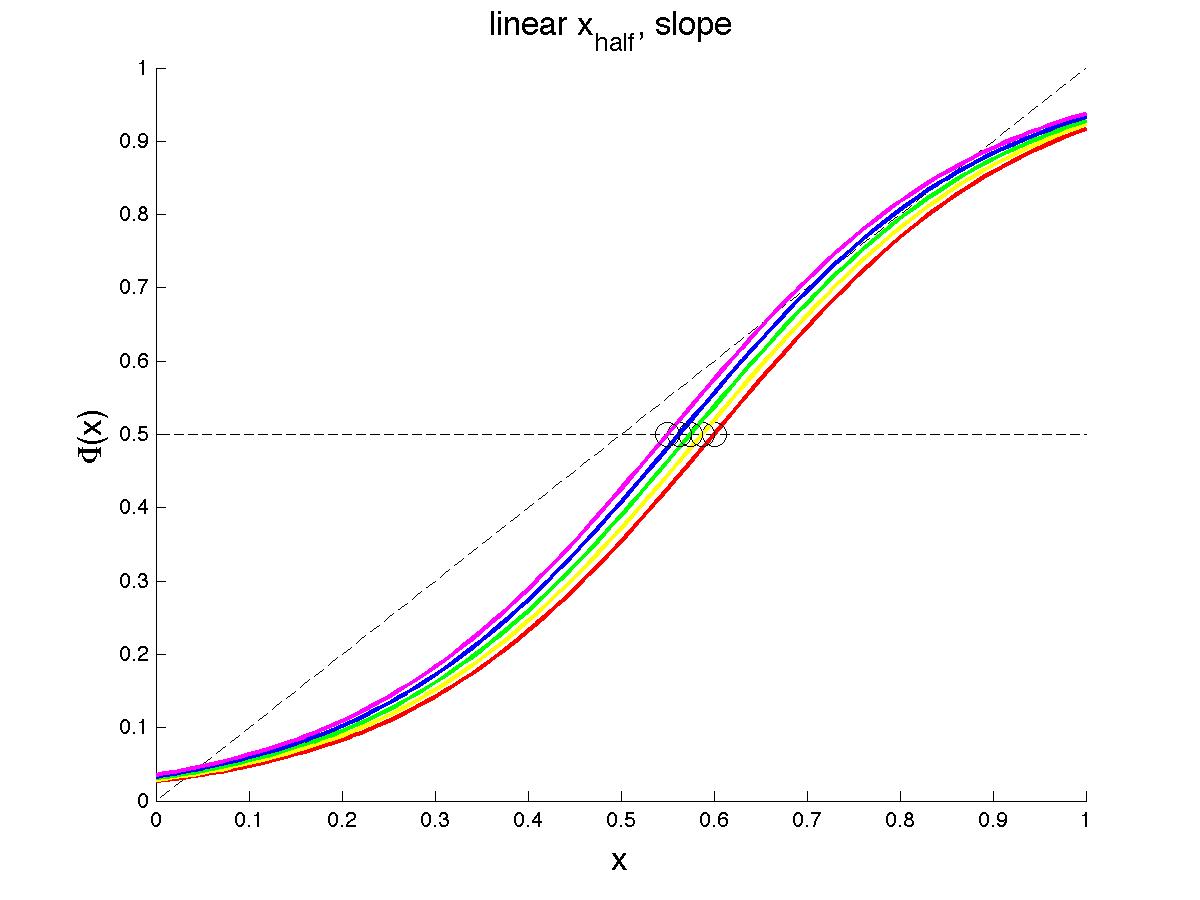
We get as the same graph as previously!

Let us play with X\_half point: Originally; Xhalfi=0.6 - 0.2\* si;

Change x into >>>>> Xhalfi=0.6 - 0.5\* si;



Change x into >>>>> Xhalfi=0.6 - 0.05\* si;



Question: How to determine x\_half and s relation?