Jong Park

Jan 30, 2018

Math 351 – Numerical Analysis

Prof. Higdon

HW 2

**1. Compute x = ln b = logeb,** b >0.

**a)** Equation in form f(x) = 0 and b is root. Newton’s method to derive an algorithm for computing approximate solution. May use exponential function, but not log.

%% 1. part a

format long

b = 1:5; % B value from 1 to 5

x = b; % x0

f = 'exp(x)- b'; % f(x) = e^x-b

fprime = 'exp(x)'; % f'(x) = e^x

maxiterations = 20;

iters = x; % Used to accumulate iterations

for k = 1 : maxiterations % 20 by default

xnew = x - eval(f) ./ eval(fprime);

iters = [iters; xnew];

test = abs(xnew -x) ./ (abs(xnew)+eps);

if max(test) < 10\*eps, break, end

x = xnew;

end

iters

**iters = ln(b), 1 ≤ b ≤ 5 using newton’s method**

1 1.000000000000000 2.000000000000000 3.000000000000000 4.000000000000000 5.000000000000000

2 0.367879441171442 1.270670566473225 2.149361205103592 3.073262555554937 4.033689734995427

3 0.060080068726789 0.831957303739969 1.499036978194360 2.258342355762720 3.122234076139850

4 0.001769199442645 0.702350584017167 1.169072406768518 1.676436772482895 2.342527216428071

5 0.000001564110790 0.693189402250512 1.101037314084009 1.424593786414320 1.822949740405518

6 0.000000000001223 0.693147181451268 1.098615226666857 1.387018509851694 1.630692346543120

7 0.000000000000000 0.693147180559945 1.098612288672426 1.386294623252305 1.609662196100816

8 0.000000000000000 0.693147180559945 1.098612288668110 1.386294361119925 1.609437937583802

9 0.000000000000000 0.693147180559945 1.098612288668110 1.386294361119891 1.609437912434101

10 0.000000000000000 0.693147180559945 1.098612288668110 1.386294361119891 1.609437912434100

**b)** For what choices of initial guess x0 does the iteration converge?

Include diagram of iteration (graph of f and some tangent lines).

Disregard any overflow/underflow in finite-precision floats of ℝ.

%% 1. part b

x = -10:0.1:4.10;

b = 3;

y = exp(x)-b;

x0 = [-10,-5,-2,-1,0,1,2,5,10]; % Initial guess values (x0)

y0 = exp(x0) -b;

for k=1:sizeof(x0)

figure

hold on

plot (x,y)

axis([-10, 10, -5, 10]);

m = exp(x0(k)); % slope

yp = m .\* x - m .\* x0(k) + y0(k); p% Graph of the new slope

plot (x,yp)

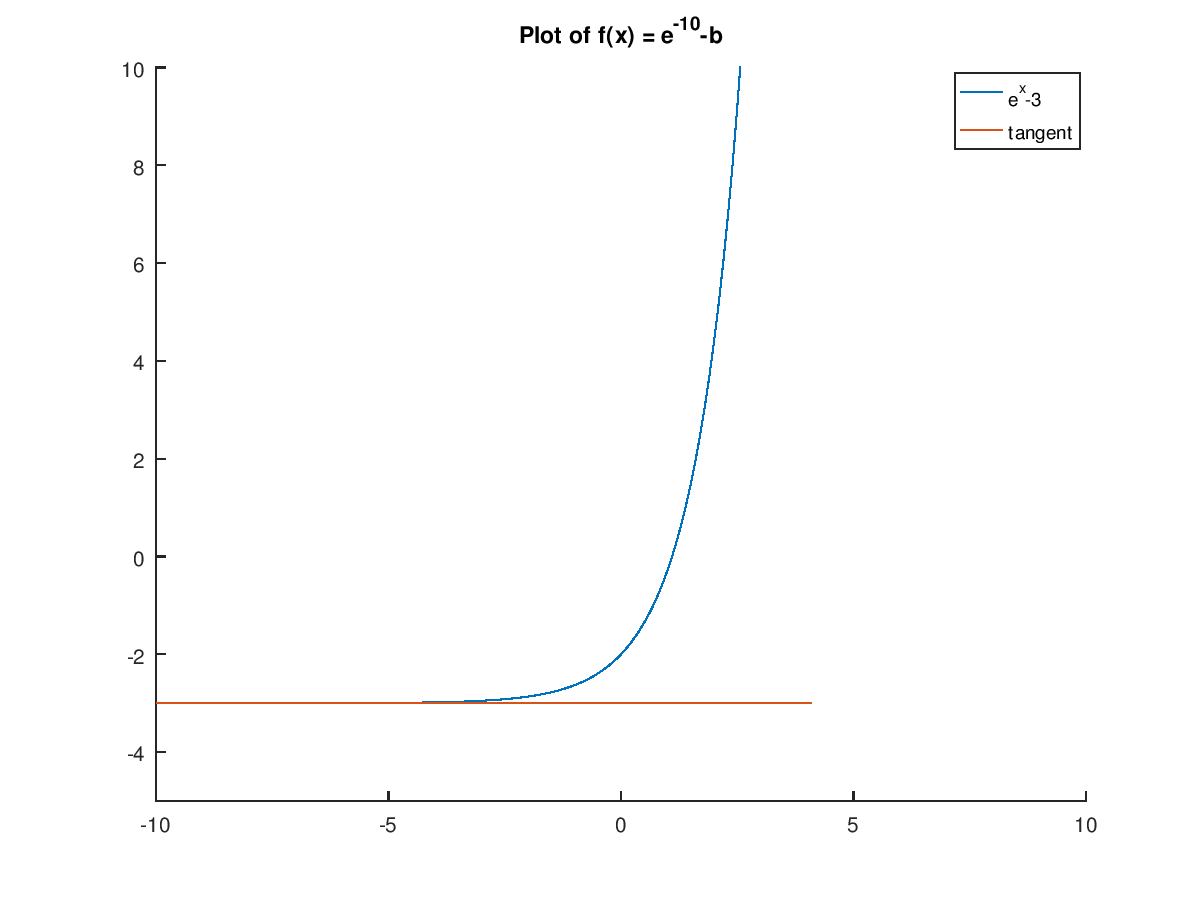
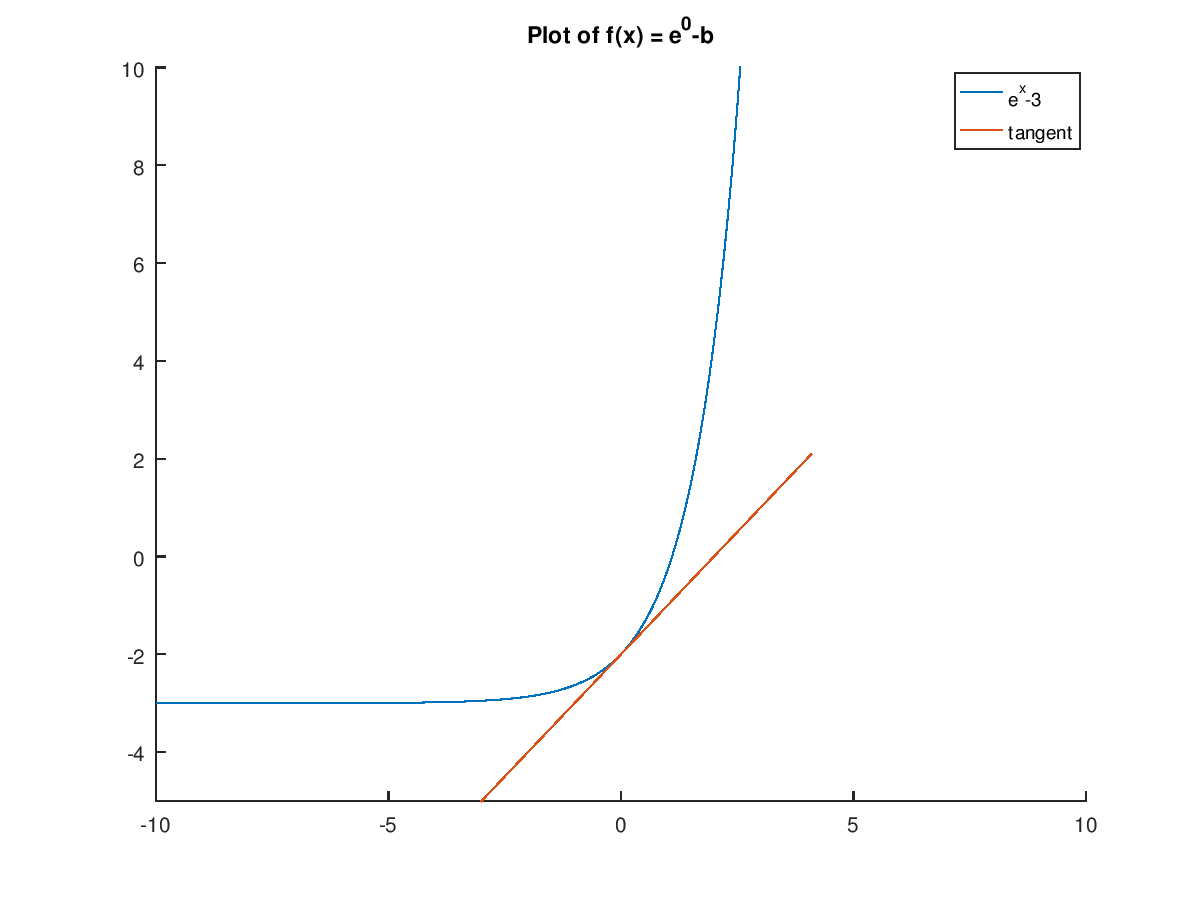
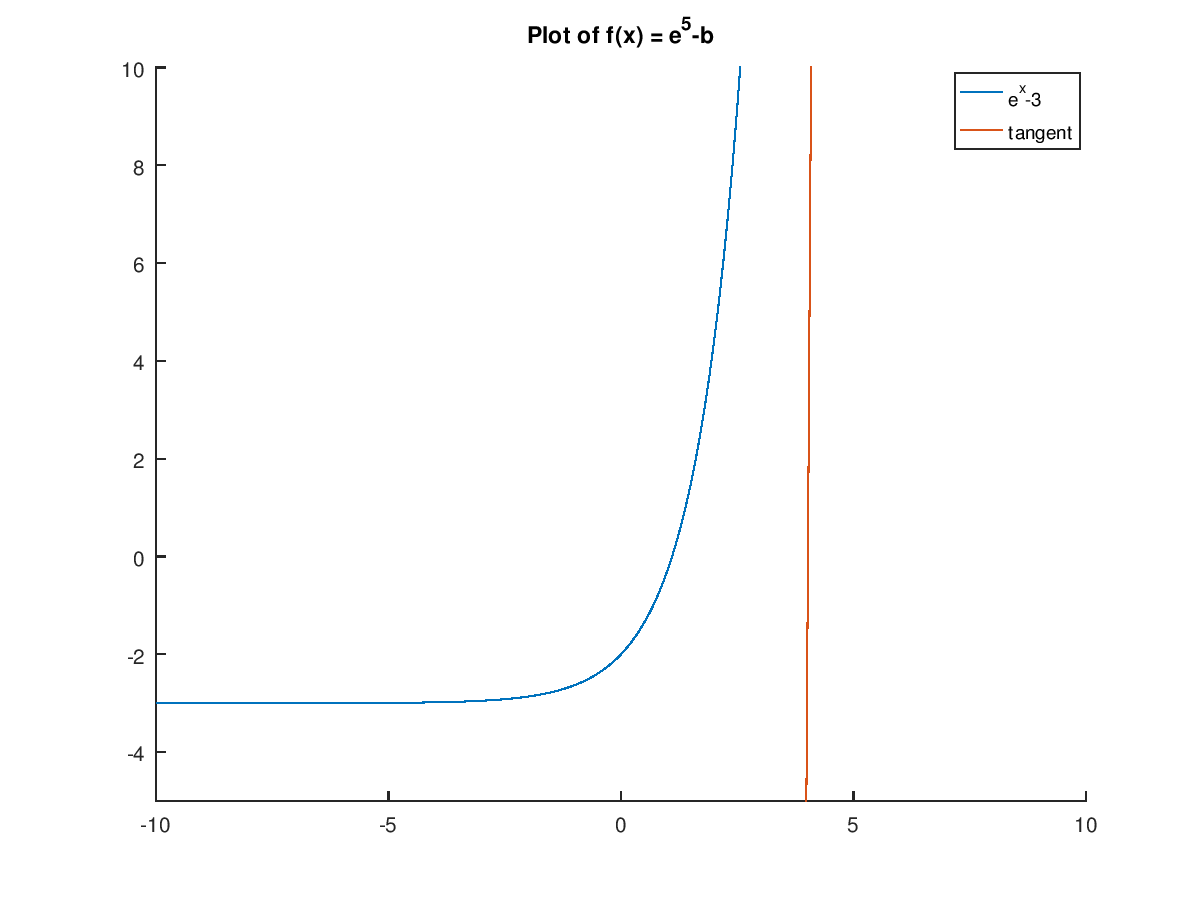
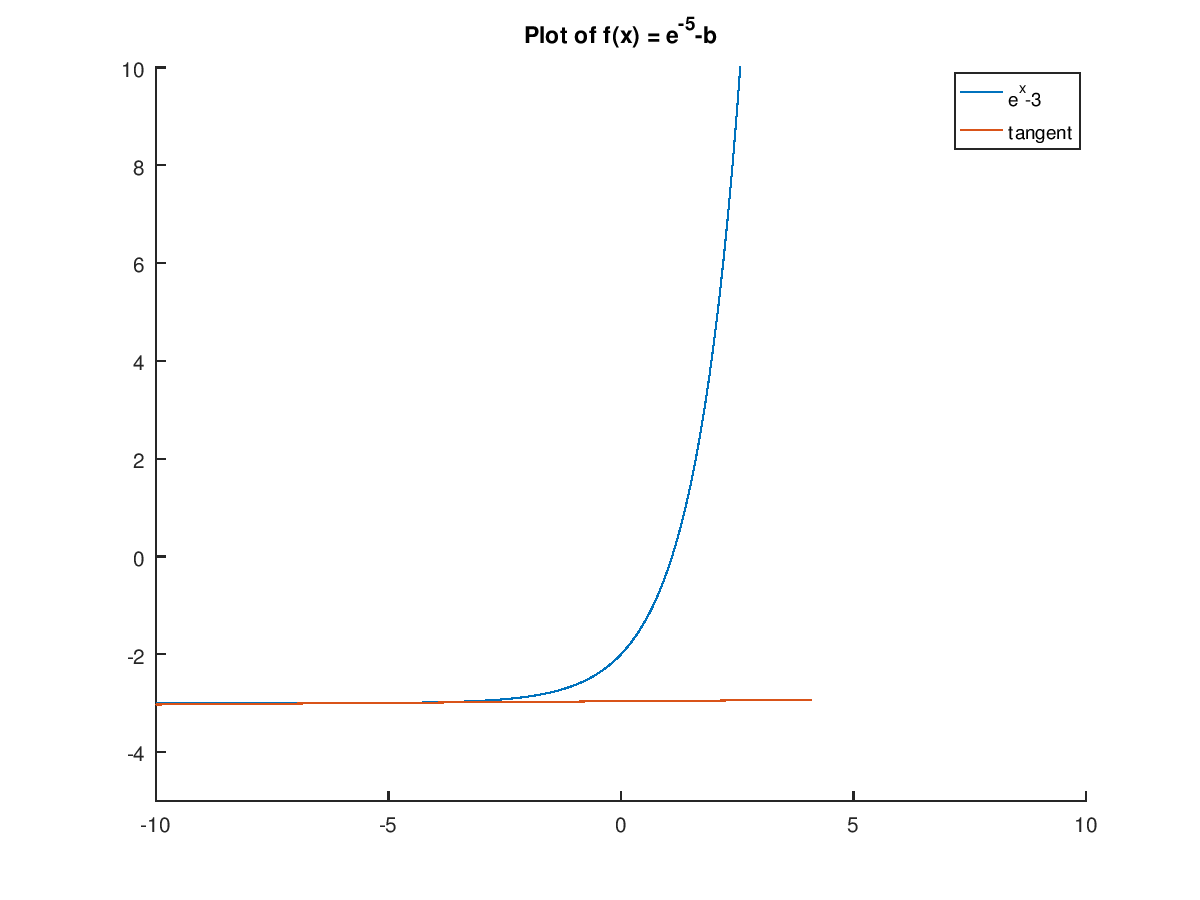
title(sprintf("Plot of f(x) = e^{%d}-b",x0(k)))

legend('e^x-3','tangent')

end

For values extremely small and extremely large, 20 iteration is not enough to converge.

On plots x = -10 and x = -5, you can see that their tangent line is very close to y = -b = -3. Similar thing be seen on x = 5 where the tangent line is vertical line. On x=0, you can see a y=x as a tangent line because that’s the point where the slope drastically changes from horizontal to vertical.



2. **Ill-conditioning of roots of polynomials.**

For any positive integer n, let

**Unperturbed polynomial:**

=

for all real x. Roots = 1, 2, …, n.

**Perturbed polynomial:**

=

a) Execute the line in pert.m

format long

n=2; % only one with real root.

n=5; % Try these values as well.

n=10;

n=20;

n=21; % The computed roots aren't even real.

p = poly(1:n); % p Contains coefficients (in descending order of a polynomial whose roots are components of the vector 1:n

r = roots(p); % r contains the computed roots of this polynomial.

>> n=2; p=poly(1:n); r=roots(p)

r =

2

1

>> n=5; p=poly(1:n); r=roots(p)

r =

5.000000000000138

3.999999999999776

3.000000000000124

1.999999999999966

1.000000000000004

>> n=10; p=poly(1:n); r=roots(p)

r =

10.000000000340687

8.999999998493760

8.000000002751277

6.999999997320477

6.000000001506791

4.999999999494381

4.000000000107472

2.999999999983562

2.000000000001650

0.999999999999950

>> n=20; p=poly(1:n); r=roots(p)

r =

19.999874055724192

19.001295393676987

17.993671562737585

17.018541647321989

15.959717574548915

15.059326234074415

13.930186454760916

13.062663652011070

11.958873995343460

11.022464271003383

9.991190949230132

9.002712743189727

7.999394310958664

7.000096952230211

5.999989523351082

5.000000705531480

3.999999973862455

3.000000000444877

1.999999999998383

0.999999999999949

>> n=21; p=poly(1:n); r=roots(p)

r =

20.998100413694775 + 0.000000000000000i

20.017033950662430 + 0.000000000000000i

18.918164261947613 + 0.000000000000000i

18.171333577200958 + 0.000000000000000i

16.515480336345934 + 0.131337608147565i

16.515480336345934 - 0.131337608147565i

14.714057397598715 + 0.000000000000000i

14.190464878645530 + 0.000000000000000i

12.953491191832381 + 0.000000000000000i

12.003238491516015 + 0.000000000000000i

11.005259635037602 + 0.000000000000000i

9.997370845609455 + 0.000000000000000i

9.000560720953006 + 0.000000000000000i

7.999976310600354 + 0.000000000000000i

6.999984161455235 + 0.000000000000000i

6.000003849858241 + 0.000000000000000i

4.999999627121142 + 0.000000000000000i

4.000000013554953 + 0.000000000000000i

3.000000000022612 + 0.000000000000000i

1.999999999997278 + 0.000000000000000i

0.999999999999971 + 0.000000000000000i

b) Given n =7, perturb the coefficient of x6 by subtracting 0.002 of that coefficient.

(Example 3.5.4, p 112-113). Use roots function to compute the roots of the unperturbed polynomial () and perturbed () polynomial.

Display the roots of these polynomials side-by side in adjacent columns of an array.

%% 2. part b

epsilon = -0.002;

n=7;

v = poly(1:n); % Creates vector of polynomials from 1 to n

vpert = v;

vpert(2) = vpert(2) + epsilon; % Subtract eps from coefficient of x^6

r = roots(v); % Roots of normal polynomial

rpert = roots(vpert); % Roots of perturbed polynomial

format long

pertable = [sort(r(:)) sort(rpert(:))]

pertable =

1 1.000000000000020 + 0.000000000000000i 1.000002777842993 + 0.000000000000000i

2 1.999999999999781 + 0.000000000000000i 1.998938173110114 + 0.000000000000000i

3 3.000000000000553 + 0.000000000000000i 3.033125347258262 + 0.000000000000000i

4 4.000000000000223 + 0.000000000000000i 3.819569248146413 + 0.000000000000000i

5 4.999999999997638 + 0.000000000000000i 5.458675826856370 - 0.540125780968025i

6 6.000000000002801 + 0.000000000000000i 5.458675826856370 + 0.540125780968025i

7 6.999999999999008 + 0.000000000000000i 7.233012799929453 + 0.000000000000000i

c) Explain the difference between the unperturbed polynomial () and perturbed () polynomial using graphs. (2 of the roots of isn’t real).

%% 2. part c

dx = 0.05;

x = 0 : dx : n+1;

y = polyval(v, x);

ypert = polyval(vpert, x);

perturbation = epsilon \* x.^(n-1);

n1 = 1+(1/dx); n2 = 1+(n/dx); % Indicies for x=1 and x=n respectively.

maxy = max(abs(y(n1:n2)));

maxypert = max(max(ypert(n1:n2)));

maxperturb = max(max(perturbation(n1:n2)));

maxy = 1.1 \* max([maxy, maxypert, maxperturb]);

plot(x, y, x, ypert, '--', x, perturbation, ':', ...

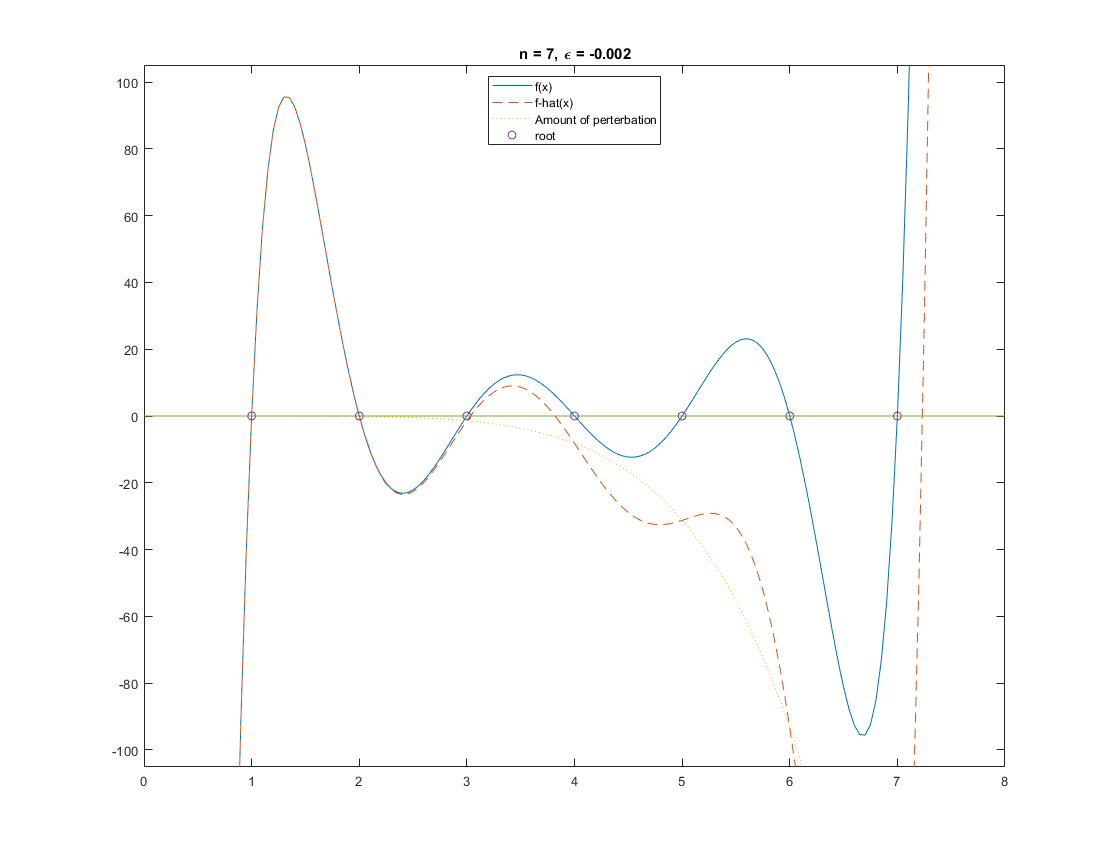
1:n, zeros(1,n), 'o', x, zeros(size(x)), '-')

axis([0 n+1 -maxy maxy])

s = ['n = ', num2str(n), ', \epsilon = ', num2str(epsilon)];

title(s)

legend({'f(x)','f-hat(x)','Amount of perterbation','root'},'Location','north')

You can see from the graph that both and has similar roots from n=1 to n=4. However, on n=5 and n=6, the (dotted line) does not cross the x-axis because at n=5 and n=6, it has imaginary root for. On n=7, the crosses the x-axis again because it has real roots.