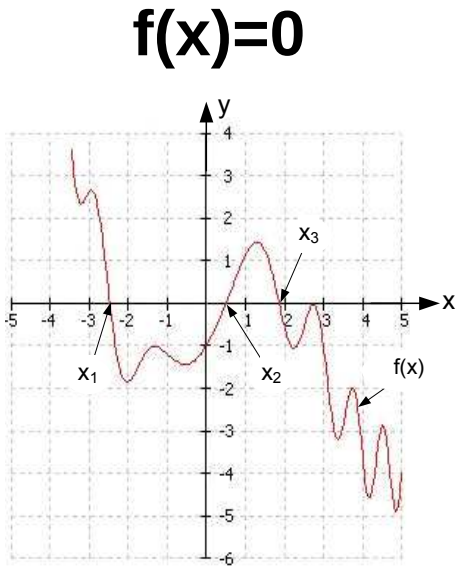
ESKİŞEHİR OSMANGAZİ ÜNİVERSİTESİ
Mühendislik Mimarlık Fakültesi
İnşaat Mühendisliği Bölümü
E-Posta: ogu.ahmet.topcu@gmail.com
Web: http://mmf2.ogu.edu.tr/atopcu

Bilgisayar Destekli

Nümerik Analiz

Ders notları 2014

Ahmet TOPÇU



36

DOĞRUSAL OLMAYAN FONKSİYONLARIN KÖKLERİ(SIFIR NOKTALARI)

- Bolzano
- Modified Regula Falsi
- Dekker-Brent
- Newton-Raphson
- Bairstow
- Müller
- Jenkins-Traub

Metodları

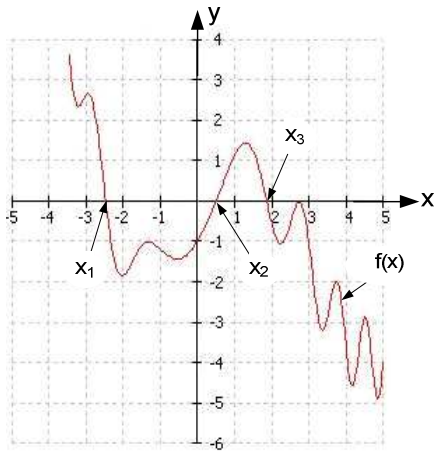
36. DOĞRUSAL OLMAYAN FONKSİYONLARIN KÖKLERİ(SIFIR NOKTALARI)

Bir fonksiyonun köklerini bulma problemi matematikçilerden çok uygulamalı bilim dalı çalışanlarının; özellikle mekanik, elektrik ve elektronik mühendislerinin ilgi alanına girer.

Bir değişkenin, diyelim x, yüksek dereceden kuvvetlerini, tersini, logaritmik veya trigonometrik terimlerini içeren f(x) fonksiyonuna doğrusal olmayan fonksiyon denir. f(x) sadece x in pozitif kuvvetlerini içeriyorsa "polinom", aksi halde "taransandant" fonksiyon adı verilir. Örnekler: f(x)=3x-5 doğrusal bir fonksiyon, $f(x)=5x^4-3x^3+x^2+8x-12$ doğrusal olmayan bir fonksiyon(polinom), $f(x)=\frac{2x^2}{\sqrt{x^{3-1}}}+Sin(x)^2-e^x$ doğrusal olmayan bir (taransandant) fonksiyondur.

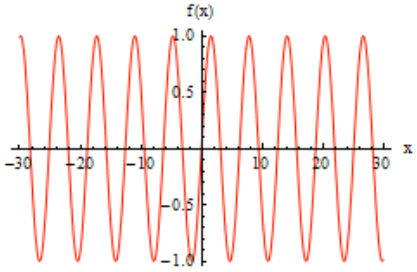
f(x)=0 fonksiyonun kökü; fonksiyonda yerine konulduğunda fonksiyonu sıfır yapan bir değerdir. Bir a sayısı ile f(a)=0 oluyorsa a sayısı f(x) fonksiyonun köküdür denir. Bu şu anlama gelmektedir: Grafiği çizildiğinde fonksiyonun x eksenini kestiği noktadaki x değeri fonksiyonun köküdür, çünkü f(x) bu x noktasında sıfır olmuştur. Bu nedenle fonksiyonun köküne fonksiyonun sıfır noktası(fonksiyonu sıfır yapan değer anlamında) da denir. Bir veya çok sayıda kök olabilir, kökler gerçek veya sanal olabilir.

Aşağıdaki grafikte $y=f(x)$ fonksiyonunun x eksenini x_1, x_2 ve x_3 noktalarında kestiği, bu noktalarda $f(x_1)=0$, $f(x_2)=0$ ve $f(x_3)=0$ olduğu görülmektedir. x_1, x_2 ve x_3 değerleri bu fonksiyonun kökleri(sıfır noktaları) dır.



Basit örnekler:

1. f(x) = x+4=0 ın sadece bir kökü vardır: x=-4.
2. f(x) = x^2-4=0 ın iki gerçek kökü vardır: x_1=2, x_2=-2.
3. f(x) = x^2+4=0 ın iki sanal kökü vardır: $x_{1,2}=\mp\sqrt{-4}=\mp 2\sqrt{-1}=\mp 2i$.
4. f(x) =ax^2+bx+c=0 ın iki kökü vardır: $x_{1,2}=\frac{-b\mp\sqrt{b^2-4ac}}{2a}$. Kökler gerçek veya sanal olabilir. Kök içindeki b^2-4ac değerine diskriminant(ayıran, belirten, fark yaratan) denir. Diskriminant $b^2-4ac>0$ ise her iki kök de gerçektir. $b^2-4ac=0$ durumunda her iki kök de gerçek fakat birbirine eşittir. $b^2-4ac<0$ durumunda ise her iki kök sanaldır.
5. f(x)=Sin(x)=0 fonksiyonu periyodiktir, sonsuz kökü vardır: k=0, 1, 2, … olmak üzere $x_k=\pm k\,\pi$.



Yukarıdaki basit çözümler klasik matematikten bilinen formüller ile yapılmıştır ve "analitik" veya "kapalı çözüm" adı verilir. Çoğu kez kapalı çözüm yoktur veya kullanılamayacak kadar karmaşıktır. Örnek: Üçüncü ve dördüncü derece polinomların çözümü için de formüller vardır, fakat kullanışsızdır. $a_0+a_1 x+a_2x^2+...+ a_n x^n = 0$ denkleminin n tane kökü vardır. Fakat $n\geq 5$ durumunda x_1, x_2, …, x_n kökleri için formül verilemeyeceği **Abel**[1] tarafından ispatlanmıştır. Çözüm sadece nümerik yöntemlerle bulunabilir.

Teknik problemlerde kullanılabilecek çok sayıda nümerik metot vardır. Hepsi de iterasyon(tekrarlama) temellidir. Her hangi bir fonksiyon için biri iyi sonuç verirken bir diğeri çözüm vermeyebilir. Çözüm bulunsa bile çoğu kez yaklaşıktır.

Bu bölümde, herhangi bir fonksiyonun [a,b] aralığındaki bir gerçek kökünün iterasyon ile belirlenmesinin temel ilkeleri örnekler ile açıklanacaktır. Ayrıca, teorisine girilmeksizin, sıkça kullanılan metotların QBASIC programları ve çözüm örnekleri verilecektir.

Basit iterasyon metodu(Picard[2] metodu, sabit nokta iterasyonu, doğrusal iterasyon):

f(x)=0 denkleminin [a,b] aralığında bir gerçek kökü olduğu biliniyorsa, bu kökün belirlenmesi için:
1) f(x) fonksiyonunun terimlerinin birinden bir x çekilir, sol tarafa yazılır. Örnek: $f(x)=x^3-2x+1=0$ fonksiyonu yerine, aynı anlama gelen $x= (x^3+1)/2$ yazılır.
2) Kökün sayısal değeri için x_0 ile göstereceğimiz bir tahmin yapılır $x=x_0$. Tahmin edilen bu x_0 değerine başlangıç değeri denir. Başlangıç değeri elden geldiğince aranan köke yakın olmalıdır.
3) Başlangıç değeri $x=(x^3+1)/2$ ifadesinin **sağ tarafında** yerine konarak sol taraftaki x yeni değeri hesaplanır. Bu yeni değere x_1 diyelim: $x_1=(x_0^3+1)/2$. x_1 sayısı kökü aranan $f(x)=x^3-2x+1$ fonksiyonunda yerine konur. Eğer x_1 aranan kök ise $f(x_1)=x_1^3-2x_1+1\approx 0$ olur, iterasyon durdurulur, aksi durumda sonraki adım ile devam edilir.
4) x_1 değeri $x=(x^3+1)/2$ ifadesinin **sağ tarafında** yerine konarak sol taraftaki x yeni değeri hesaplanır. Bu yeni değere x_2 diyelim: $x_2=(x_1^3+1)/2$. x_2 sayısı kökü aranan $f(x)=x^3-2x+1$ fonksiyonunda yerine konur. Eğer x_2 aranan kök ise $f(x_2)=x_2^3-2x_2+1\approx 0$ olur, iterasyon durdurulur, aksi durumda sonraki adım ile devam edilir.

5) ....

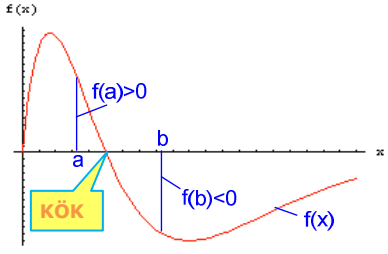
Yukarıda adımları verilen iterasyon yöntemi, sayısal örneklerden de anlaşılacağı gibi, çok basittir. Ancak, örneklere geçmeden önce, açıklanması gereken **sorular** vardır.

Sorular:
1) Hangi [a,b] aralığında kök arayacağımı nereden bileyim? 2) [a,b] aralığında kök olup olmadığını nasıl anlarım 3) x_0 başlangıç değerini nasıl tahmin edeyim? 4) İterasyonu ne zaman durdurayım 5) İterasyon her zaman çözüm verir mi(yakınsar mı)? 6) Yakınsamazsa ne yapayım?

Cevaplar:
1) Fonsiyonu başka biri vermişse a ve b sınırlarını, yani [a,b] aralığını da verir. Veya fonksiyonu bir fiziksel problemin çözümü için biz oluşturmuşsak hangi aralıkta çözüm aradığımızı da biliriz. Çünkü çözüm salt matematikten öte fiziksel bir anlam da taşımaktadır.

2) [a,b] aralığında kök olup olmadığını ya biri bize söylemiştir veya fonksiyonun fiziksel anlamından biliriz ya da şöyle anlayabiliriz: a ve b değerleri f(x) fonksiyonunda yerine konur, f(a) ve f(b) değerleri hesaplanır. Eğer

- **f(a) ve f(b) ters işaretli ise [a,b] aralığında kök vardır. Çünkü fonksiyon artıdan eksiye veya eksiden artıya geçmektedir, dolayısıyla x eksenini kesmek zorundadır.**
- **f(a) ve f(b) aynı işaretli ise [a,b] aralığında kök yoktur, yani fonksiyon x eksenini kesmiyor demektir.**
- **f(a)=0 ve/veya f(b)=0 oluyorsa a ve/veya b köktür.**



[1] Niels Henrik **Abel**, 1802-1829, Norveçli matematikçi. $n\geq 5$ olması halinde $a_0+a_1x+a_2x^2+....+a_nx^n = 0$ ifdesinin kapalı çözümünün olmadığını 1824 yılında ispatldı. Abel'den önceki matematikçiler 300 yıl boyunca çözümün varlığını aramışlardır.
[2] Charles Émile **Picard**, 1856–1941, Fransız matematikçi.

3) x_0 başlangıç değerini ya biri bize söylemiştir veya fonksiyonun fiziksel anlamından biliriz ya da şöyle seçebilirz: $x_0=a$, veya $x_0=b$ veya $x_0=(a+b)/2$.

4) İterasyonu durdurmak için şu koşullar kullanılabilir: i adımda $|f(x_i)|<\varepsilon$ veya $|x_i- x_{i-1}|<\varepsilon$ olunca iterasyon durdurulur. Burada ε yeterince küçük pozitif bir sayıdır, sonucun hassasiyetini ve iterasyon sayısını etkiler. Mesala $\varepsilon=0.0001$ seçilebilir. $|f(x_i)|<0.0001$ olduğunda, $f(x_i) \approx 0$ dır ve son hesaplanan x_i köktür anlamındadır. ε ne denli küçük seçilirse sonuç o denli gerçeğe yakın olur, fakat iterasyon sayısı artar.

5) İterasyonun yakınsaması fonksiyonun ne denli karmaşık olduğuna ve x_0 başlangıç değerinin aranan köke ne denli yakın olduğuna bağlıdır, yakınsamayabilir.

6) İterasyon yakınsamazsa başka bir x_0 başlangıç değeri seçilir. Gene yakınsamazsa başka bir iterasyon metodu ile çözüm bulunmaya çalışılır.

Sayısal örnek 1:
$f(x)=x^3-2x+1=0$ fonksiyonunun [a,b]=[0,0.9] aralığında kökü var mıdır? Varsa kökü belirleyiniz.

[0, 0.9] aralığında kök var mı? f(0)=1>0 ve f(0.9)=-0.07<0 olduğunda bu aralıkta kök vardır. Fonksiyonun -2x terimindeki x i çekerek sol tarafa geçirelim:

$x^3-2x+1=0 \rightarrow x=(x^3+1)/2$.
$x_0=0$, $\varepsilon=0.001$ seçelim.
İterasyon adımları aşağıda verilmiştir:

Önceki x | Sağ taraftan hesplanan yeni x | Yeni x için fonksiyonun aldığı değer

Adım	x	$x=(x^3+1)/2$	$f(x)=x^3-2x+1$
0	0 (x_0)	0.5 (x_1)	0.1250
1	0.5	0.5625 (x_2)	0.0530
2	0.5625	0.5890 (x_3)	0.0263
3	0.5890	0.6022	0.0140
4	0.6022	0.6092	0.0077
5	0.6092	0.6130	0.0043
6	0.6130	0.6152	0.0024
7	0.6152	0.6164	0.0014
8	0.6164	0.6171 (x_9)	0.0007

Kök

0.6171 değeri için $|f(0.6171)|=0.0007< \varepsilon=0.001$ olduğundan iterasyon durduruldu

Sayısal örnek 2:
$f(x)=x-\sqrt{x+6}=0$ fonksiyonunun [a,b]=[0, 5] aralığında kökü var mıdır? Varsa kökü belirleyiniz.

[0, 5] aralığında kök var mı? f(0)=-2.4<0 ve f(5)=1.7>0 olduğunda bu aralıkta kök vardır. Fonksiyonun x terimini çekerek sol tarafa geçirelim:

$f(x)=x-\sqrt{x+6}=0 \rightarrow x=\sqrt{x+6}$
$x_0=0$, $\varepsilon=0.0001$ seçelim.
İterasyon adımları aşağıda verilmiştir:

Önceki x | Sağ taraftan hesplanan yeni x | Yeni x için fonksiyonun aldığı değer

Adım	x	$x=\sqrt{x+6}$	$f(x)=x-\sqrt{x+6}=0$
0	0 (x_0)	2.4495 (x_1)	-0.4573
1	2.4495	2.9068 (x_2)	-0.0776
2	2.9068	2.9844 (x_3)	-0.0130
3	2.9844	2.9973	-0.0022
4	2.9973	2.9995	-0.0004
5	2.9995	2.9999 (x_6)	0.0000

Kök

2.9999 değeri için $|f(2.9999)|=0.0000<\varepsilon=0.0001$ olduğundan iterasyon durduruldu

Sayısal örnek 3: Bu örnek için hesap makinenizin açı birimini radyan olarak ayarlamalısınız!

$f(x)=\frac{1}{e^x}-\mathrm{Sin}\,x=0$ fonksiyonunun [a,b]=[0,1] aralığında kökü var mıdır? Varsa kökü belirleyiniz.

[0, 1] aralığında kök var mı? f(0)=1>0 ve f(1)=-0.47<0 olduğunda bu aralıkta kök vardır. Fonksiyonun bir x terimini çekerek sol tarafa geçirelim. Ancak bu fonksiyondaki hiç bir terimi çekilemez. Bu nedenle $\frac{1}{e^x}-\mathrm{Sin}\,x=0$ eşitliği bozulmayacak şekilde her iki tarafa x ekleyelim:

$\frac{1}{e^x}-\mathrm{Sin}\,x=0 \rightarrow x+\frac{1}{e^x}-\mathrm{Sin}\,x=x \rightarrow x=x+\frac{1}{e^x}-\mathrm{Sin}\,x$

$x_0=0$, $\varepsilon=0.001$ seçelim.

İterasyon adımları aşağıda verilmiştir:

Önceki x | Sağ taraftan hesplanan yeni x | Yeni x için fonksiyonun aldığı değer

Adım	x	$x=x+\frac{1}{e^x}-\mathrm{Sin}\,x$	$f(x)=\frac{1}{e^x}-\mathrm{Sin}\,x$
0	0 (x_0)	1 (x_1)	-0.4736
1	1	0.5264 (x_2)	0.0883
2	0.5264	0.6147 (x_3)	-0.0359
3	0.6147	0.5788	0.0136
4	0.5788	0.5924	-0.0054
5	0.5924	0.5870	0.0021
6	0.5870	0.5891	-0.0008
7	0.5891	0.5883 (x_8)	0.0003

Kök

0.5883 değeri için |f(0.5883)|=0.0003< ε=0.001 olduğundan iterasyon durduruldu

Sayısal örnek 4:

$f(x)=x^3-3=0$ fonksiyonunun [a,b]=[1, 5] aralığında kökü var mıdır? Varsa kökü belirleyiniz.

[1, 5] aralığında kök var mı? f(1)=-2<0 ve f(5)=122>0 olduğunda bu aralıkta kök vardır. Fonksiyonu $xx^2-3=0$ şeklinde yazalım ve x i çekerek sol tarafa geçirelim:

$xx^2-3=0 \rightarrow x=3/x^2$.

$x_0=1$, $\varepsilon=0.001$ seçelim.

İterasyon adımları aşağıda verilmiştir:

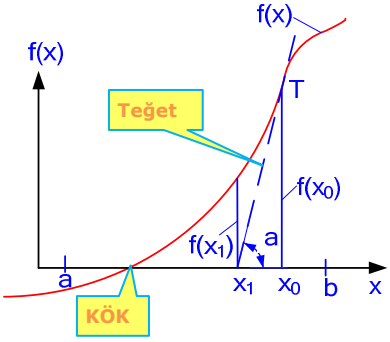
Önceki x | Sağ taraftan hesplanan yeni x | Yeni x için fonksiyonun aldığı değer

Adım	x	$x=3/x^2$	$f(x)=x^3-3$
0	1 (x_0)	3.0 (x_1)	24.0
1	3.0	0.3333 (x_2)	-2.9630
2	0.3333	27.0054 (x_3)	19691.8122
3	27.0054	0.0041 (x_4)	-3.0

YAKINSAMIYOR:
f(x) fonksiyonunun aldığı değer bir adımda büyüyor diğer adımda küçülüyor. İterasyona sonsuz kez devam edilse dahi |f(x)|<ε olamayacağı anlaşılıyor.

Newton-raphson(Newton) metodu:

f(x) fonksiyonun türevi olan f'(x) fonksiyonu da biliniyorsa kullanılabilir. Basit iterasyon metoduna nazaran daha hızlı yakınsar.



Soldaki çizimde görülen f(x) fonksiyonun verilmiş x_0 başlangıç değerini(bilinen değer) kullanarak köke daha yakın olan x_1 değerini bulabilir miyiz? Evet:

x_0 bilinen değerini fonksiyonda yerine koyarak $f(x_0)$ ordinatını hesaplayabiliriz. Bu ordinatın f(x) eğrisini kestiği T noktasında eğrinin teğetini çizdiğimizi düşünelim. Teğetin x eksenini kestiği noktaya x_1 diyelim. Teğetin eğim açısı α olsun. x_1Tx_0 alanı dik üçgendir. Bu bilgileri kullanak x_1 değerini hesaplamaya çalışalım. Teğetin eğimi

$$Tan\,\alpha=\frac{f(x_0)}{x_0-x_1}$$

dır.

Diğer taraftan, bir fonksiyonun bir noktadaki teğetinin eğimininin fonksiyonun o noktadaki türevine eşit olduğunu biliyoruz: $Tan\,\alpha = f'(x_0)$. f(x) verilmiş fonksiyon olduğundan türevi alınarak f'(x_0) hesaplanabilir. Yukarıdaki ifadede yerine yazarsak $f'(x_0) = \frac{f(x_0)}{x_0 - x_1}$ olur. Bu ifadede sadece x_1 bilinmemektedir, çekerek:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

bulunur. Şekilden de görüldüğü gibi, bu değere karşılık gelen $f(x_1)<f(x_0)$ dır, yani köke yaklaşılmıştır. Yaklaşım miktarı $\frac{f(x_0)}{f'(x_0)}$ dir. Bu değeri $\Delta=\frac{f(x_0)}{f'(x_0)}$ ile gösterirelim:

$x_1=x_0-\Delta$

olur. Eğer $|\Delta|$ yeterince küçük ise x_1 aranan köktür. Aksi halde $x_0=x_1$ alınarak yeni bir yaklaşım, x_2 hesaplanır.

Yukarıdaki son formül tekrar-tekrar kullanılarak(iterasyon) kök bulunabilir. f(x) ve [a,b] aralığı verilmiştir. [a,b] aralığında kök olup olmadığı kontrol edilir. Kök varsa f'(x) türev fonksiyonu hesaplanır, x_0 başlangıç değeri ve ε hassasiyeti seçilir, sonra iterasyona başlanır.

İterasyon adımları:

0. adım: f(x_0), f'(x_0), $\Delta=\frac{f(x_0)}{f'(x_0)}$ ve $x_1 = x_0 - \Delta$ değerlerini hesapla $|\Delta|<\varepsilon$ ise x_1 aranan köktür, iterasyonu durdur. Aksi halde bir sonraki adım ile devam et.
1. adım: f(x_1), f'(x_1), $\Delta=\frac{f(x_1)}{f'(x_1)}$ ve $x_2 = x_1 - \Delta$ değerlerini hesapla $|\Delta|<\varepsilon$ ise x_2 aranan köktür, iterasyonu durdur. Aksi halde bir sonraki adım ile devam et.
2. adım: f(x_2), f'(x_2), $\Delta=\frac{f(x_2)}{f'(x_2)}$ ve $x_3 = x_2 - \Delta$ değerlerini hesapla $|\Delta|<\varepsilon$ ise x_3 aranan köktür, iterasyonu durdur. Aksi halde bir sonraki adım ile devam et.
3. adım: ....

Sayısal örnek 5:
$f(x)=x^3-2x+1=0$ fonksiyonunun [a,b]=[0,0.9] aralığında kökü var mıdır? Varsa kökü belirleyiniz.

[0, 0.9] aralığında kök var mı? f(0)=1>0 ve f(0.9)=-0.07<0 olduğunda bu aralıkta kök vardır. Fonksiyonun türevi $f'(x)=3x^2-2$ dir. $x_0=0$, $\varepsilon=0.001$ seçelim.

İterasyon adımları aşağıda verilmiştir:

Adım	x (Önceki x)	f(x)=x^3-2x+1 (Fonksiyonun x için aldığı değer)	f'(x)=$3x^2$-2 (Fonksiyonun türevinin x için aldığı değer)	Δ=f(x)/f'(x) (x deki değişim miktarı)	x-Δ (Yeni x)
0	0 (x_0)	1	-2	-0.5	0.5 (x_1)
1	0.5	0.125	-1.25	-0.1	0.6 (x_2)
2	0.6	0.016	-0.92	-0.0174	0.6174 (x_3)
3	0.6174	0.0005	-0.8565	-0.0006	0.6180 (x_4)

Değişim miktarı |-0.0006|<ε=0.001 olduğundan iterasyon durduruldu

Kök

Basit iterasyon ile aynı fonksiyon örnek 1 de 9. adımda yakınsamamıştı, bulunan kök 0.6171 idi. Newton-Raphson ise 4 iterasyon sonucunda kökü 0.6180 olarak vermiştir. 6 hane hassasiyetli kök 0.618034 dir. Buna göre Newton-Raphson metodu daha hızlıdır ve daha doğru sonuç vermiştir.

Sayısal örnek 6: Bu örnek için hesap makinenizin açı birimini radyan olarak ayarlamalısınız!

$f(x)=\frac{1}{e^x}-\mathrm{Sin}\,x=0$ fonksiyonunun [a,b]=[0,1] aralığında kökü var mıdır? Varsa kökü belirleyiniz.

[0,1] aralığında kök var mı? f(0)=1>0 ve f(1)=-0.047<0 olduğunda bu aralıkta kök vardır. Fonksiyonun türevi $f'(x)=-\frac{1}{e^x}-\mathrm{Cos}\,x=0$ dir. x_0=0, ε=0.001 seçelim.

İterasyon adımları aşağıda verilmiştir:

Adım	x (Önceki x)	$f(x)=\frac{1}{e^x}-\mathrm{Sin}\,x=0$ (Fonksiyonun x için aldığı değer)	$f'(x)=-\frac{1}{e^x}-\mathrm{Cos}\,x=0$ (Fonksiyonun türevinin x için aldığı değer)	Δ=f(x)/f'(x) (x deki değişim miktarı)	x-Δ (Yeni x)
0	0 (x_0)	1	-2	-0.5	0.5 (x_1)
1	0.5	0.1271	-1.4841	-0.0856	0.5856 (x_2)
2	0.5856	0.0041	-1.3902	-0.0029	0.5885 (x_3)
3	0.5885	0.0000	-1.3869	0.0000	0.5885 (x_4)

Değişim miktarı |0.0000|<ε=0.001 olduğundan iterasyon durduruldu

Kök

Basit iterasyon ile aynı fonksiyonun örnek 3 de 8 iterasyon sonucunda bulunan kökü 0.5883 idi. Newton-Raphson ise 4 iterasyon sonucunda kökü 0.5885 olarak vermiştir. 6 hane hassasiyetli kök 0.588533 tür. Newton-Raphson metodu daha hızlıdır ve daha doğru sonuç vermiştir.

Sayısal örnek 7:
$f(x)=x^3-3=0$ fonksiyonunun [a,b]=[1, 5] aralığında kökü var mıdır? Varsa kökü belirleyiniz.
[1, 5] aralığında kök var mı? f(1)=-2<0 ve f(5)=122>0 olduğunda bu aralıkta kök vardır.

Fonksiyonun türevi $f(x)=3x^2$ dir. x_0=1, ε=0.001 seçelim.

İterasyon adımları aşağıda verilmiştir:

Adım	x (Önceki x)	$f(x)=x^3-3$ (Fonksiyonun x için aldığı değer)	$f'(x)=3x^2$ (Fonksiyonun türevinin x için aldığı değer)	Δ=f(x)/f'(x) (x deki değişim miktarı)	x-Δ (Yeni x)
0	1 (x_0)	-2	3	-0.6667	1.6667 (x_1)
1	1.6667	1.6299	8.3337	0.1956	1.4711 (x_2)
2	1.4711	0.1837	6.4924	0.0283	1.4428 (x_3)
3	1.4428	0.0034	6.2450	0.0005	1.4423 (x_4)

Değişim miktarı |0.0005|<ε=0.001 olduğundan iterasyon durduruldu

Kök

Basit iterasyon ile aynı fonksiyon örnek 4 de yakınsamamıştı. Newton-Raphson ise 4 iterasyon sonucunda kökü 1.4423 olarak vermiştir. Kökün gerçek değeri $\sqrt[3]{3}=1.44224957$ dir. Newton-Raphson metodu yakınsamıştır.

*Her iterasyon yönteminin kendine özgü **iyi-kötü** tarafları vardır. Hangisinin **"en iyi"** olduğunu söylemek zordur. Mesala, yukarıdaki örneklerden anlaşıldığı gibi, Newton-Rahson daha hızlı yakınsar, fakat fonksiyonun analitik türevi her zaman hesaplanamaz veya türev sıfır olabilir. Diğer taraftan aynı fonksiyon için biri yakınsarken diğeri yakınsamayabilir. Önemli olan, fonksiyona uygun bir metodun seçilmesi ve x_0 başlangıç değerinin aranan köke mümkün olduğunca yakın tahmin edilmesidir.*

Aitken iterasyon hızlandırıcısı

İterasyon metotlarında AITKEN hızlandırıcısı kullanılarak iterasyon hızlandırılabilir. k. adımda hesaplanan x değerini x^k, k-1. adımdakini x^{k-1}, k-2. adımdakini x^{k-2} ile gösterelim. **AITKEN**'e göre x^k nın iyileştirilmiş değeri $x^k_{\text{iyileştirilmiş}} \leftarrow x^k - \frac{(x^k - x^{k-1})^2}{x^k - 2x^{k-1} + x^{k-2}}$ dir. Bu formülde birbini izleyen üç değer olduğundan üçüncü ve sonraki adımlarda uygulanabilir. Örnek olması açısından, Örnek 1 in ilk üç adımı sonunda bulunan $x_1=0.5$, $x_2=0.5625$, $x_3=0.5890$ değerlerini kullanarak x_3 ün iyileştirilmiş değerini bulalım:

$$x_3 \leftarrow 0.5890 - \frac{(0.5890-0.5625)^2}{0.5890-2\cdot 0.5625+0.5} = 0.6085$$

Örnek 1 de 5. adımda bu değere yaklaşılmıştır.

Simgesel program kodları: Bildiğiniz bir programlama dili ile kodlayarak test ediniz. Maxit değişkeni neden kullanılmıştır?

Örnek 1 için basit iterasyon(Picard iterasyunu)

```
Maxit=100
x0=0, Eps=0.000001
Adım=0
Repeat
  x=x0
  x1=(x*x*x+1)/2
  Fx=x1*x1*x1-2*x1+1
  Print Adım, xo, x1, Fx
 Adım=Adım+1
 x0=x1
Until |Fx|<Eps or Adım>Maxit
```

Örnek 1 için Newton-Raphson iterayonu

```
Maxit=100
x0=0, Eps=0.000001
Adım=0
Repeat
  x=x0
  Fx=x*x*x-2*x+1
  Fxtürev=3*x*x-2
  if |Fxtürev| =1.0E-8 then  Print Türev sıfır oldu, dur.
  Delta=Fx/Fxtürev
  x0=x-Delta
  Print Adım, x, Fx, Fxtürev, Delta, x0
 Adım=Adım+1
until |Delta|<Eps or Adım>Maxit
```

PROGRAMLAR:

1. **Bolzano:** f(x) transandant fonksiyonunun [a, b] aralığındaki gerçek köklerini Bolzano metoduna[3] göre bulur.
2. **RegulaFalsi:** f(x) transandant fonksiyonunun [a, b] aralığındaki gerçek köklerini Regula-Falsi[4] metoduna göre bulur.
3. **DekkerBrent:** f(x) transandant fonksiyonunun [a, b] aralığındaki gerçek köklerini Dekker-Brent[5] metoduna göre bulur.
4. **NewtonRaphson:** Gerçek katsayılı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun gerçek ve sanal köklerini Newton-Raphson[6] metoduna göre bulur.
5. **Bairstow:** Gerçek katsayılı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun gerçek ve sanal köklerini Bairstow[7] metoduna göre bulur.
6. **Muller:** Gerçek katsayılı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun gerçek köklerini Müller[8] metoduna göre bulur.
7. **JenkinsTraub:** Gerçek katsayılı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun gerçek ve sanal köklerini Jenkins-Traub[9] metoduna göre bulur.

[3] Bernard **Bolzano** tarafından 1817 de ortaya kondu.

[4] MÖ 3. yüzyıla ait bir Hindistan matematik kitabında izine rastlanmaktadır. Hint-Çin- Arap matematik dünyasından İtalyan matematikçi Leonardo **Fibonacci**(yaklaşık 1170-1250) tarafından 1202 yılında Avrupa'ya tanıtılmıştır.

[5] 1969 yılında Hollandalı Theodorus **Dekker** geliştirdi, Avustralyalı Richard **Brent** tarafından 1973 yılında iyileştirildi. Genelde Brent metodu olarak anılır.

[6] İngiliz Isaac **Newton** 1669 da geliştirdi, 1690 yılında İngiliz Joseph **Raphson**(1648-1715) tarafından iyileştirildi.

[7] İngiliz Leonard **Bairstow** tarafından 1920 yılında geliştirildi.

[8] Amerikalı David E. **Müler** tarafından 1956 yılında yayınlandı.

[9] Michael A. G. **Jenkins** ve Joseph Frederick **Traub** tarafından 1966 yılında geliştirildi.

Bolzano metodu
(Bisection, interval halfing, binary search)

Bolzano programı f(x) transandant fonksiyonunun [a, b] aralığındaki gerçek köklerini bulur. Aşağıdaki örnekler için test edilmiştir.

Örnekler:

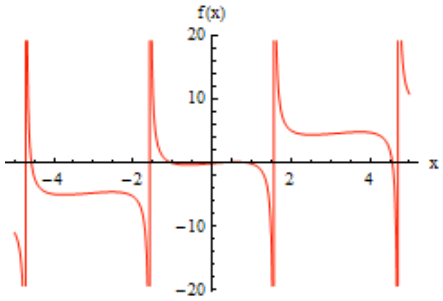
1. $f(x)=1.5x-Tan(x)-0.1$, aralık $[-5,5]$
2. $f(x)=3x+Sin(x)-e^{-x}$, aralık $[-1,1]$
3. $f(x)=x^2-Ln(x)-2$, aralık $[0.1,3]$
4. $f(x)=x^8-1$, aralık $[-3,3]$
5. $f(x)=Sin\sqrt{\frac{1}{Cos(x)}+x^3e^{\frac{5x}{Tan(x)}}}-e^{-x}$, aralık $[0.1,1]$
6. $f(x)=x^3-21x-20$, aralık $[-5,8]$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$, aralık $[-12,12]$
8. $f(x)=Sin(8x)-e^{Cos(x)}+1$, aralık $[\pi,3\pi]$

Bernard **Bolzano**
1781-1848, İtalyan asıllı Avusturyalı

Bolzano sonuçları:

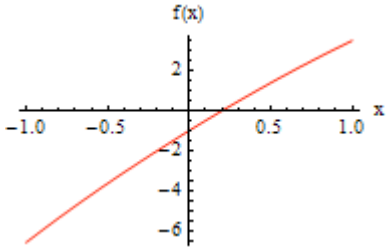
1. $f(x)=1.5x-Tan(x)-0.1=0$ ın [-5,5] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-5 , 5 ] aralığında  5 kök bulundu(Bolzano):

x 1 =-4.5695732537657        f(x 1 )= 2.358821224717620-09
x 2 =-1.02086716890335       f(x 2 )= 1.981698434798570-09
x 3 = .205921697616576       f(x 3 )=-3.435615883039430-10
x 4 = .890390598773955       f(x 4 )=-1.540044236667680-09
x 5 = 4.56526547223329       f(x 5 )=-8.196318123342150-09
```



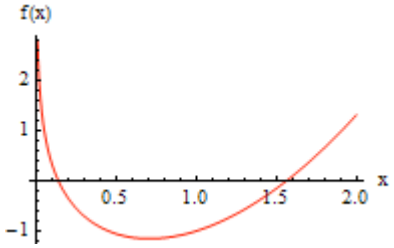
2. $f(x)=3x+Sin(x)-e^{-x}=0$ ın [-1,1] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-1 , 1 ] aralığında  1 kök bulundu(Bolzano):

x 1 = .204182364940644       f(x 1 )=-8.148176058029620-10
```



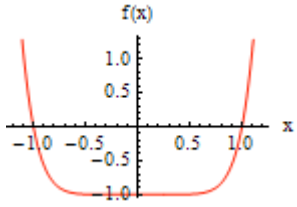
3. $f(x)=x^2-Ln(x)-2=0$ ın [0.1,3] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[ .1000000001490116 , 3 ] aralığında  2 kök bulundu(Bolzano):

x 1 = .137934824983655       f(x 1 )= 4.055955941982890-09
x 2 = 1.56446225645395       f(x 2 )=-6.977310388310570-09
```



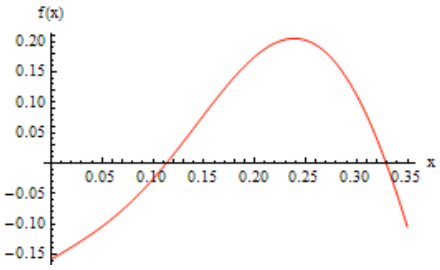
4. $f(x)=x^8-1=0$ ın [-3,3] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-3 , 3 ] aralığında  2 kök bulundu(Bolzano):

x 1 =-1.000000001192090      f(x 1 )= 9.536728779994360-09
x 2 = 1.000000001192100      f(x 2 )= 9.536762530723550-09
```



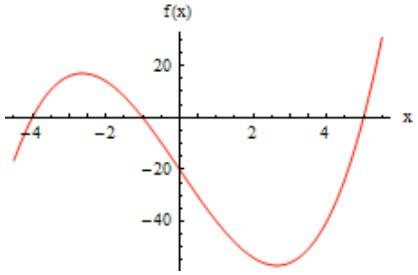
5. $f(x) = Sin\sqrt{\frac{1}{Cos(x)} + x^3 e^{\frac{5x}{Tan(x)}}} - e^{-x} = 0$ ın [0.1,1] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[ .100000001490116 , 1 ] aralığında  2 kök bulundu(Bolzano):

x 1 = .113343403376898       f(x 1 )=-3.310077997830710-09
x 2 = .329130216755585       f(x 2 )= 6.958919121231430-09
```



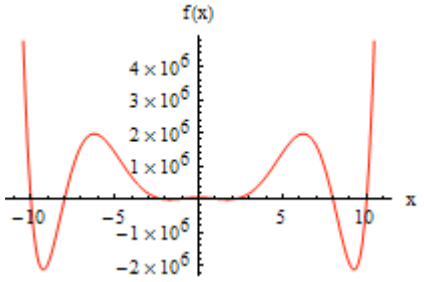
6. $f(x) = x^3 - 21x - 20 = 0$ ın [-5,8] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-5 , 8 ] aralığında  3 kök bulundu(Bolzano):

x 1 =-4.000000000003725      f(x 1 )=-1.005851416047140-09
x 2 =-1.000000000014902      f(x 2 )= 2.682270405784950-09
x 3 = 5.000000000011175      f(x 3 )= 6.034628796669490-09
```



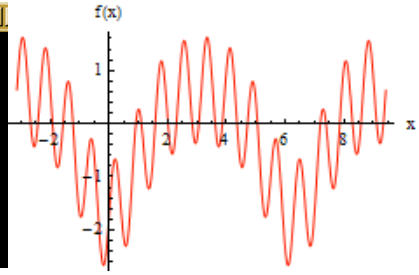
7. $f(x) = x^8 - 170\,x^6 + 7392\,x^4 - 39712\,x^2 + 51200 = 0$ ın [-12,12] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-12 , 12 ] aralığında  8 kök bulundu(Bolzano):

x 1 =-10       f(x 1 )=-1.309672370553020-10
x 2 =-8        f(x 2 )= 3.823970473604280-09
x 3 =-2.000000000000014        f(x 3 )= 6.302798283286390-09
x 4 =-1.414213562372890        f(x 4 )= 7.101981225332570-09
x 5 = 1.414213562372910        f(x 5 )= 6.506692074026430-09
x 6 = 2.000000000000015        f(x 6 )= 7.100879884092140-09
x 7 = 8        f(x 7 )= 0
x 8 = 10       f(x 8 )=-1.309672370553020-10
```



8. $f(x) = Sin(8x) - e^{Cos(x)} + 1 = 0$ ın [π,3π] aralığında

```
C:\Basic\QBasic.EXE
[ 3.141592653589790 , 9.424777960769380 ] aralığında  10 kök bulundu(Bolzano):

x 1 = 3.613127983303830      f(x 1 )= 1.374330485678050-09
x 2 = 3.857187901326020      f(x 2 )= 9.593247808712800-09
x 3 = 4.356976031891760      f(x 3 )= 4.400004403469670-09
x 4 = 4.712388980384690      f(x 4 )=-1.993761059027040-14
x 5 = 4.712388980384690      f(x 5 )=-1.993761059027040-14
x 6 = 5.053939109307080      f(x 6 )= 8.658194642361190-09
x 7 = 7.194501871460410      f(x 7 )=-4.313253044009940-09
x 8 = 7.385322340346880      f(x 8 )=-5.344418398068800-09
x 9 = 7.853981633974460      f(x 9 )=-1.866083242790810-13
x 10 = 7.853981633974460     f(x 10 )=-1.866083242790810-13
```



Bolzano

```
'--------------------Ana Program Bolzano-----------------------------
' Doğrusal olmayan f(x) fonksiyonun [a,b] aralığındaki köklerini bulur
' Sadece gerçek kökler bulunur, sanal kök bulunmaz.
' Çağrılan alt programlar: Bolzano
'--------------------------------------------------------------------------
DEFINT I-N
DEFDBL A-H, O-Z
DECLARE SUB Bolzano (a, b, x(), MaxBul, iBulundu)
' aralığın ve fonksiyonun tanımlanması
  a = -5: b = 5: DEF fnf (x) = 1.5 * x - TAN(x) - .1
  'a = -1: b = 1: DEF fnf (x) = 3 * x + SIN(x) - EXP(-x)
  'a = .1: b = 3: DEF fnf (x) = x ^ 2 - LOG(x) - 2
  'a = -3: b = 3: DEF fnf (x) = x ^ 8 - 1
  'a = .1: b = 1: DEF fnf (x) = SIN(SQR(1 / COS(x) + x ^ 3 * EXP(5 * x / TAN(x)))) - EXP(-x)
  'a = -5: b = 8: DEF fnf (x) = x ^ 3 - 21 * x - 20
  'a = -12: b = 12: DEF fnf (x) = x ^ 8 - 170 * x ^ 6 + 7392 * x ^ 4 - 39712 * x ^ 2 + 51200
  'a = 4 * ATN(1): b = 3 * a: DEF fnf (x) = SIN(8 * x) - EXP(COS(x)) + 1

  MaxBul = 10:  ' [a,b] aralığında en fazla 10 kök bul
  DIM x(MaxBul)

  CLS
  CALL Bolzano(a, b, x(), MaxBul, iBulundu)

  IF iBulundu = 0 THEN PRINT "Kök bulunamadı(Bolzano)": END

  PRINT a; "-"; b; "aralığında bulunabilen kökler(Bolzano):"
  PRINT
  FOR i = 1 TO iBulundu
   PRINT "x"; i; "="; x(i), "f(x"; i; ")="; fnf(x(i))
  NEXT i

END 'Bolzano ana
```

```
SUB Bolzano (a, b, x(), MaxBul, iBulundu)
'--------------------------------------------------------------------------
' f(x) fonksiyonunun [a,b] aralığındaki gerçek köklerini bulur
' Metot: Bolzano
' Diğer adları: Bisection metodu, aralığı ikiye bölme, binary search
' f(x) fonksiyonu ana programda DEF FNF(X)=... ile tanımlanmış olmalı
' aralığın alt sınırı a ve üst sınırı b ana programda tanımlı olmalı
' MaxAaltAralik: [a, b] aralığında alınacak alt aralık sayısı
' MaxBol: alt aralığın ikiye bölme sayısı
' Hassasiyet: Hassasiyet
' MaxBul: aranacak maksimum kök sayısı
' iBulundu: bulunabilen kök sayısı
' x(MaxBul): Köklerin depolandığı vektör, ana programda boyutlandırılmış olmalı

' Çağrılan program: Yok
'--------------------------------------------------------------------------
  MaxAaltAralik = 100
  MaxBol = 50
  Hassasiyet = 1E-08
  iBulundu = 0

  IF a = b THEN MaxAaltAralik = 1
  D = (b - a) / MaxAaltAralik
  x2 = a
  WHILE x2 < b
   IF iBulundu >= MaxBul THEN EXIT SUB
   x1 = x2
   x2 = x1 + D
   IF x2 > b THEN x2 = b

' [x1,x2] aralığında kök var mı?
   f1 = fnf(x1)
   IF ABS(f1) < Hassasiyet THEN
    xYeni = x1
    iBulundu = iBulundu + 1
    x(iBulundu) = xYeni
    GOTO 6
    END IF

   f2 = fnf(x2)
   IF ABS(f2) < Hassasiyet THEN
    xYeni = x2
    iBulundu = iBulundu + 1
    x(iBulundu) = xYeni
    GOTO 6
   END IF
```

```
   iF1 = SGN(f1)
   iF2 = SGN(f2)

' [x11,x22] aralığını ikiye böl
   IF iF1 * iF2 <> 1 THEN
    x11 = x1
    x22 = x2
    xk = x1
    FOR j = 1 TO MaxBol
     xYeni = .5 * (x11 + x22)
     f3 = fnf(xYeni)
     IF ABS(f3) < Hassasiyet THEN
      iBulundu = iBulundu + 1
      x(iBulundu) = xYeni
      GOTO 6
     END IF

     iF3 = SGN(f3)
     IF iF3 * iF1 = -1 THEN
      x22 = xYeni
     ELSE
      x11 = xYeni
      iF1 = iF3
     END IF
     xk = xYeni
    NEXT j
   END IF

6  WEND

END SUB ' Bolzano sonu
```

Regula-Falsi Metodu
(Method of false position, (Linear interpolation method)

RegulaFalsi programı f(x) transandant fonksiyonunun [a, b] aralığındaki gerçek köklerini bulur. Aşağıdaki örnekler için test edilmiştir.

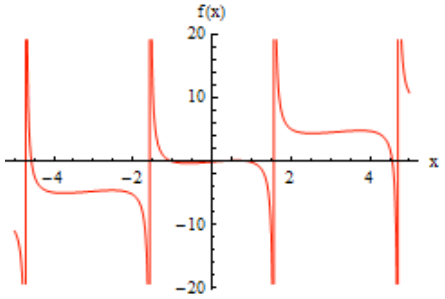
Örnekler:

1. $f(x)=1.5x-Tan(x)-0.1$, aralık $[-5,5]$
2. $f(x)=3x+Sin(x)-e^{-x}$, aralık $[-1,1]$
3. $f(x)=x^2-Ln(x)-2$, aralık $[0.1,3]$
4. $f(x)=x^8-1$, aralık $[-3,3]$
5. $f(x)=Sin\sqrt{\frac{1}{Cos(x)}+x^3e^{\frac{5x}{Tan(x)}}}-e^{-x}$, aralık $[0.1,1]$
6. $f(x)=x^3-21x-20$, aralık $[-5,8]$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$, aralık $[-12,12]$
8. $f(x)=Sin(8x)-e^{Cos(x)}+1$, aralık $[\pi,3\pi]$

RegulaFalsi sonuçları:

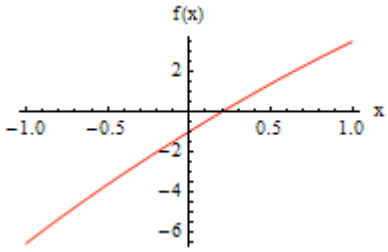
1. $f(x)=1.5x-Tan(x)-0.1=0$ ın [-5,5] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-5 , 5 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 =-4.56957325371537       f(x 1 )=-5.01491318090352D-11
x 2 =-1.02086716623222       f(x 2 )=-3.79100034144601D-09
x 3 = .205921698369788       f(x 3 )= 1.80083595605063D-13
x 4 = .8903905972739         f(x 4 )= 2.16432774480246D-14
x 5 = 4.56526547205125       f(x 5 )= 1.9021798025598D-12
```



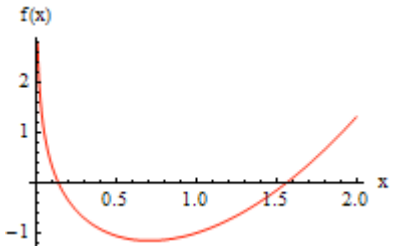
2. $f(x)=3x+Sin(x)-e^{-x}=0$ ın [-1,1] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-1 , 1 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 = .204182365110591       f(x 1 )=-4.67291136341252D-17
```



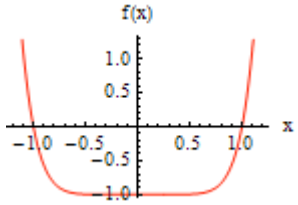
3. $f(x)=x^2-Ln(x)-2=0$ ın [0.1,3] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[ .100000001490116 , 3 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 = .137934825565436       f(x 1 )=-1.34223730220706D-12
x 2 = 1.56446225925616       f(x 2 )=-5.79486653974615D-13
```



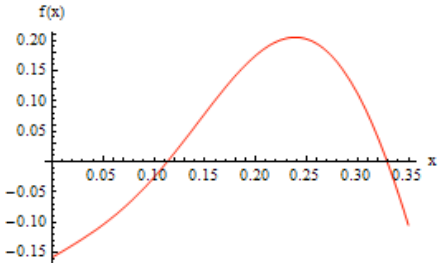
4. $f(x) = x^8 - 1 = 0$ ın [-3,3] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-3 , 3 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 =-.999999999933757       f(x 1 )=-5.29946752984584D-10
x 2 = .999999999999942       f(x 2 )=-4.64517313503166D-13
```



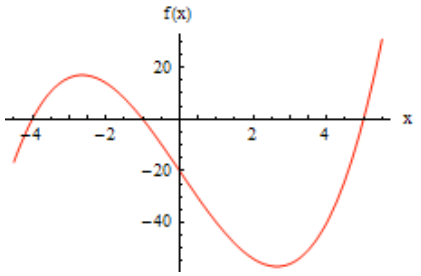
5. $f(x) = Sin\sqrt{\frac{1}{Cos(x)} + x^3 e^{\frac{5x}{Tan(x)}}} - e^{-x} = 0$ ın [0.1,1] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[ .100000001490116 , 1 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 = .113343404997058       f(x 1 )=-1.34602620932914D-12
x 2 = .329130218252623       f(x 2 )= 1.09119382366713D-10
```



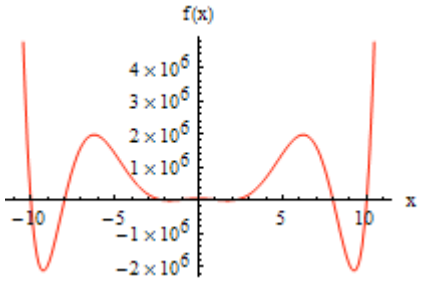
6. $f(x) = x^3 - 21x - 20 = 0$ ın [-5,8] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-5 , 8 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 =-3.99999999999996       f(x 1 )= 1.01918126715894D-12
x 2 =-1.00000000000354       f(x 2 )= 6.36211083815397D-11
x 3 = 4.99999999999734       f(x 3 )=-1.43453263157234D-10
```



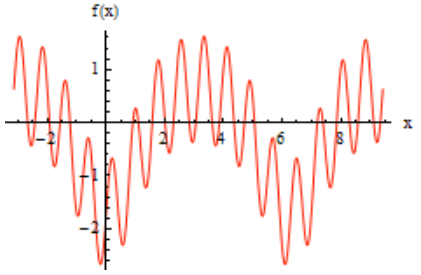
7. $f(x) = x^8 - 170\,x^6 + 7392\,x^4 - 39712\,x^2 + 51200 = 0$ ın [-12,12] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[-12 , 12 ] aralığında bulunabilen kökler(RegulaFalsi):

x 1 =-10      f(x 1 )=-1.30967237055302D-10
x 2 =-8       f(x 2 )= 1.89061211131048D-09
x 3 =-1.99999999999989       f(x 3 )=-5.05450969967569D-09
x 4 =-1.4142135623731        f(x 4 )=-8.72688588060555D-11
x 5 = 1.4142135623731        f(x 5 )=-3.31468186232087D-12
x 6 = 2       f(x 6 )=-6.13837869423151D-11
x 7 = 8       f(x 7 )= 0
x 8 = 10      f(x 8 )=-1.30967237055302D-10
```



8. $f(x) = Sin(8x) - e^{Cos(x)} + 1 = 0$ ın [π,3π] aralığında bulunabilen kökleri:

```
C:\Basic\QBasic.EXE
[ 3.14159265358979 , 9.42477796076938
] aralığında bulunabilen kökler(RegulaFalsi):

x 1 = 3.61312798355843       f(x 1 )=-3.180625082971690-10
x 2 = 3.85718789984473       f(x 2 )=-2.61518237620884D-14
x 3 = 4.35697603290773       f(x 3 )=-4.04134037920809D-09
x 4 = 4.71238898038469       f(x 4 )=-1.99376105902704D-14
x 5 = 4.71238898038469       f(x 5 )=-1.99376105902704D-14
x 6 = 5.05393911030726       f(x 6 )=-3.22008045228195D-16
x 7 = 7.19450187221299       f(x 7 )= 2.46764414457701D-15
x 8 = 7.38532233931212       f(x 8 )= 1.37827455611722D-12
x 9 = 7.85398163397446       f(x 9 )=-1.866083242790810-13
x 10 = 7.85398163397446      f(x 10 )=-1.866083242790810-13
```



RegulaFalsi

```
'---------------------------Ana Program RegulaFalsi-------------------------
' Doğrusal olmayan bir fonksiyonun [a,b] aralığındaki köklerinin hesabı
' sanal kök bulunmaz, sadece gerçek kökler bulunur
' Çağrılan alt programlar: RegulaFalsi
'---------------------------------------------------------------------------
DEFINT I-N
DEFDBL A-H, O-Z
DECLARE SUB RegulaFalsi (a, b, x(), iFind, iFound)
' aralığın ve fonsiyonun tanımlanması
  'a = -5: b = 5: DEF fnf (x) = 1.5 * x - TAN(x) - .1
  'a = -1: b = 1: DEF fnf (x) = 3 * x + SIN(x) - EXP(-x)
  'a = .1: b = 3: DEF fnf (x) = x ^ 2 - LOG(x) - 2
  'a = -3: b = 3: DEF fnf (x) = x ^ 8 - 1
  'a = .1: b = 1: DEF fnf (x) = SIN(SQR(1 / COS(x) + x ^ 3 * EXP(5 * x / TAN(x)))) - EXP(-x)
  'a = -5: b = 8: DEF fnf (x) = x ^ 3 - 21 * x - 20
  'a = -12: b = 12: DEF fnf (x) = x ^ 8 - 170 * x ^ 6 + 7392 * x ^ 4 - 39712 * x ^ 2 + 51200
  a = 4 * ATN(1): b = 3 * a: DEF fnf (x) = SIN(8 * x) - EXP(COS(x)) + 1

  iFind = 10: ' [a,b] aralığında en fazla 10 kök bul
  DIM x(iFind)

  CLS

  CALL RegulaFalsi(a, b, x(), iFind, iFound)

  IF iFound = 0 THEN PRINT "Kök bulunamadı (RegulaFalsi)!": END

  PRINT "["; a; ","; b; "] aralığında bulunabilen kökler(RegulaFalsi):"
  PRINT
  FOR i = 1 TO iFound
   PRINT "x"; i; "="; x(i), "f(x"; i; ")="; fnf(x(i))
  NEXT i

END ' RegulaFalsi Ana
```

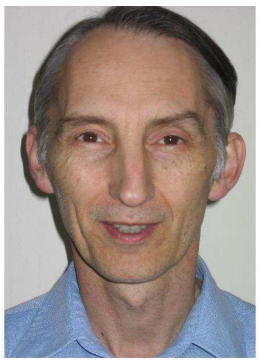
```
SUB RegulaFalsi (a, b, x(), iFind, iFound)
'---------------------------------------------------------------------------
' f(x) fonksiyonunun [a,b] aralığındaki gerçek köklerini bulur
' Metot: Regula falsi
' Diğer adları: Method of false position, linear interpolation method
' f(x) fonksiyonu ana programda DEF FNF(X)=... ile tanımlanmış olmalı
' aralığın alt sınırı a ve üst sınırı b ana programda tanımlı olmalı
' iSubint  :[a, b] aralığında alınacak alt aralık sayısı
' maxalve: alt aralığın ikiye bölme sayısı
' epsf     :Hassasiyet
' iFind    :aranacak maksimum kök sayısı
' iFound   :bulunabilen kök sayısı
' x(iFind) :Köklerin depolandığı vektör, ana programda boyutlandırılmış olmalı

' Çağrılan program: Yok
'---------------------------------------------------------------------------
  iSubint = 100
  Maxhalve = 50
  epsf = 1E-08
  iFound = 0
  iHata = 0
  IF a > b THEN EXIT SUB

  IF a = b THEN iSubint = 1
  D = (b - a) / iSubint
  x2 = a
  WHILE x2 < b
   IF iFound >= iFind THEN EXIT SUB
   x1 = x2
   x2 = x1 + D
   IF x2 > b THEN x2 = b

' [x1,x2] aralığında kök var mı?
   f1 = fnf(x1)
   IF ABS(f1) < epsf THEN
    xNew = x1
    iFound = iFound + 1
    x(iFound) = xNew
    GOTO 6
   END IF

   f2 = fnf(x2)
   IF ABS(f2) < epsf THEN
    xNew = x2
    iFound = iFound + 1
    x(iFound) = xNew
    GOTO 6
  END IF
```

```
  iF1 = SGN(f1)
  iF2 = SGN(f2)
  IF iF1 * iF2 = 1 GOTO 6
' [ak,bk] aralığında Regula Falsi iterasyonu
  ak = x1
  bk = x2
  aL = f1
  r = f2
  xk = x1
  fxk = aL
  FOR k = 1 TO Maxhalve
   IF ABS(r - aL) < epsf THEN
    xNew = ak
   ELSE
    xNew = (ak * r - bk * aL) / (r - aL)
   END IF
   fk1 = fnf(xNew)
   IF ABS(fk1) < epsf THEN
    iFound = iFound + 1
    x(iFound) = xNew
    GOTO 6
   END IF
   IF fnf(ak) * fk1 < 0 THEN
    bk = xNew
    r = fk1
    IF fxk * r > 0 THEN aL = .5 * aL
   ELSE
    ak = xNew: aL = fk1
    IF fxk * aL > 0 THEN r = .5 * r
   END IF
   xk = xNew
   fxk = fk1
  NEXT k

6 WEND

END SUB ' RegulaFalsi sonu
```

Dekker-Brent metodu

DekkerBrent programı f(x) transandant fonksiyonunun [a, b] aralığındaki gerçek köklerini bulur. Aşağıdaki örnekler için test edilmiştir.

Theodorus **Dekker,** 1927- , Hollandalı



Richard **Brent,** 1946- , Avustralyalı

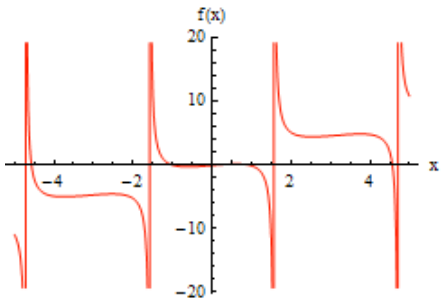
Örnekler:

1. $f(x)=1.5x-Tan(x)-0.1$, aralık $[-5,5]$
2. $f(x)=3x+Sin(x)-e^{-x}$, aralık $[-1,1]$
3. $f(x)=x^2-Ln(x)-2$, aralık $[0.1,3]$
4. $f(x)=x^8-1$, aralık $[-3,3]$
5. $f(x)=Sin\sqrt{\frac{1}{Cos(x)}+x^3 e^{\frac{5x}{Tan(x)}}}-e^{-x}$, aralık $[0.1,1]$
6. $f(x)=x^3-21x-20$, aralık $[-5,8]$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$, aralık $[-12,12]$
8. $f(x)=Sin(8x)-e^{Cos(x)}+1$, aralık $[\pi,3\pi]$

Bolzano sonuçları:

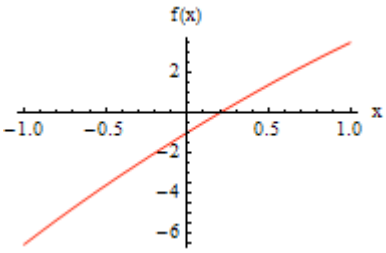
1. $f(x)=1.5x-Tan(x)-0.1=0$ ın [-5,5] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[-5 , 5 ] aralığında 5  kök bulundu(DekkerBrent)
x 1 =-4.56957325371642      f(x 1 )= 1.3838322848736D-14
x 2 =-1.02086716798638      f(x 2 )= 1.13082286590238D-16
x 3 = .205921698369393      f(x 3 )= 2.846030702774445D-19
x 4 = .890390597273921      f(x 4 )=-1.35525271560688D-17
x 5 = 4.56526547205129      f(x 5 )=-1.77405833079458D-14
```



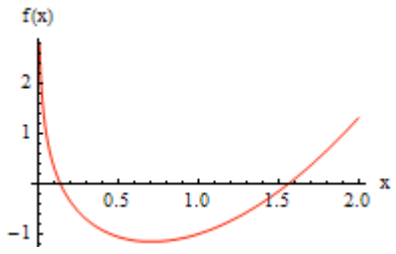
2. $f(x)=3x+Sin(x)-e^{-x}=0$ ın [-1,1] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[-1 , 1 ] aralığında 1  kök bulundu(DekkerBrent)
x 1 = .204182365110591      f(x 1 )=-4.672911363412520-17
```



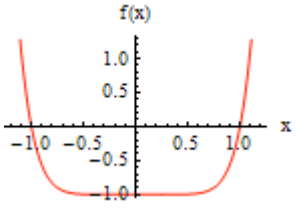
3. $f(x)=x^2-Ln(x)-2=0$ ın [0.1,3] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[ .100000001490116 , 3 ] aralığında 2  kök bulundu(DekkerBrent)
x 1 = .137934825565243      f(x 1 )=-5.431852884152380-17
x 2 = 1.56446225925639      f(x 2 )=-1.203464411458910-16
```



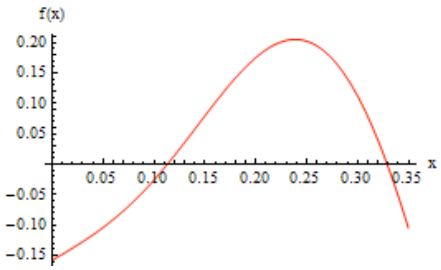
4. $f(x)=x^8-1=0$ ın [-3,3] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[-3 , 3 ] aralığında 2  kök bulundu(DekkerBrent)
x 1 =-1       f(x 1 )= 0
x 2 = 1       f(x 2 )=-8.881784197001250-16
```



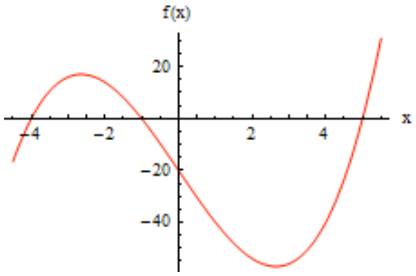
5. $f(x) = Sin\sqrt{\frac{1}{Cos(x)} + x^3 e^{\frac{5x}{Tan(x)}}} - e^{-x} = 0$ ın [0.1,1] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[ .100000001490116 , 1 ] aralığında 2  kök bulundu(DekkerBrent)
x 1 = .113343404997717      f(x 1 )= 5.421010862427520-20
x 2 = .329130218276471      f(x 2 )= 2.932766876573290-16
```



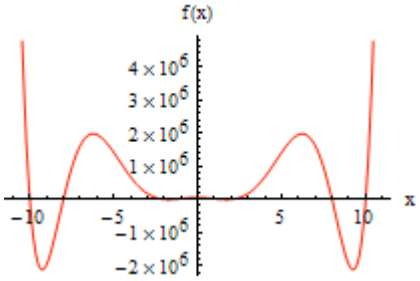
6. $f(x) = x^3 - 21x - 20 = 0$ ın [-5,8] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[-5 , 8 ] aralığında 3  kök bulundu(DekkerBrent)
x 1 =-4         f(x 1 )= 0
x 2 =-1         f(x 2 )= 0
x 3 = 5         f(x 3 )=-1.387778780781450-17
```



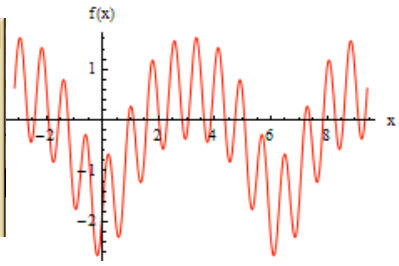
7. $f(x) = x^8 - 170\,x^6 + 7392\,x^4 - 39712\,x^2 + 51200 = 0$ ın [-12,12] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[-12 , 12 ] aralığında 7  kök bulundu(DekkerBrent)
x 1 =-10        f(x 1 )=-1.309672370553020-10
x 2 =-8         f(x 2 )= 0
x 3 =-2         f(x 3 )= 0
x 4 =-1.4142135623731           f(x 4 )=-2.622257966322650-11
x 5 = 1.4142135623731           f(x 5 )=-3.314681862320870-12
x 6 = 2         f(x 6 )= 0
x 7 = 8         f(x 7 )= 1.890612111310480-09
```



8. $f(x) = Sin(8x) - e^{Cos(x)} + 1 = 0$ ın [π,3π] aralığında bulunabilen kökleri:

```
C:\NANALIZ\Basic\QBasic.EXE
[ 3.14159265358979 , 9.42477796076938 ] aralığında 10  kök bulundu(DekkerBrent)
x 1 = 3.61312798351058      f(x 1 )=-6.338245900350260-16
x 2 = 3.85718789984473      f(x 2 )=-1.464757135027920-14
x 3 = 4.35697603242133      f(x 3 )= 3.623728921098300-15
x 4 = 4.71238898038469      f(x 4 )=-1.993761059027040-14
x 5 = 5.05393911030726      f(x 5 )=-3.220080452281950-16
x 6 = 7.19450187221299      f(x 6 )=-2.622901895676930-15
x 7 = 7.38532233931238      f(x 7 )= 1.718297813063650-15
x 8 = 7.85398163397448      f(x 8 )= 5.238214376146470-15
x 9 = 8.29069707917627      f(x 9 )=-1.946793420914970-15
x 10 = 8.57626761514359     f(x 10 )=-3.478662670419740-15
```



DekkerBrent

```
'------------------Ana program DekkerBrent------------------------
' Ahmet Topçu, Eskişehir Osmangazi Üniversitesi, 2010
' f(x) fonksiyonunun [a,b] aralığındaki köklerini bulur
' Metot: Dekker Brent
' f(x) fonksiyonnu DEF Fnf(x)=... deyimi ile tanımlanmış olmalıdır
' Veri:
'   x1,x2   : kök aranacak aralığın alt ve üst sınırı
'   MaxBul  : [a,b] aralığında bulunacak max kök sayısı

' Çıktı:
'   iBulundu: bulunabilen kök sayısı
'   x       : bulunan köklerin depolandığı vektör
'   iErr    =0  hatasız sonuçlandı
'           =1  [x1,x2] aralığında kök yok
'           =2  Max iterasyon sayısı aşıldı

' Çağrılan alt programlar: DekkerBrentCall, DekkerBrent
'-------------------------------------------------------------------

DEFINT I-N
DEFDBL A-H, O-Z
DECLARE SUB DekkerBrentCall (a, b, x(), MaxBul, iBulundu)
DECLARE FUNCTION DekkerBrent (x1, x2, iErr)
' aralığın ve fonksiyonun tanımlanması
   Pi = 4 * ATN(1): ' Pi sayısı
   'a = -5: b = 5: DEF fnf (x) = 1.5 * x - TAN(x) - .1
   'a = -1: b = 1: DEF fnf (x) = 3 * x + SIN(x) - EXP(-x)
   'a = .1: b = 3: DEF fnf (x) = x ^ 2 - LOG(x) - 2
   'a = -3: b = 3: DEF fnf (x) = x ^ 8 - 1
   'a = .1: b = 1: DEF fnf (x) = SIN(SQR(1 / COS(x) + x ^ 3 * EXP(5 * x / TAN(x)))) - EXP(-x)
   'a = -5: b = 8: DEF fnf (x) = x ^ 3 - 21 * x - 20
   'a = -12: b = 12: DEF fnf (x) = x ^ 8 - 170 * x ^ 6 + 7392 * x ^ 4 - 39712 * x ^ 2 + 51200
   a = Pi: b = 3 * Pi: DEF fnf (x) = SIN(8 * x) - EXP(COS(x)) + 1

   MaxBul = 10: ' [a,b] aralığında bulunması istenen max kök sayısı
   DIM x(MaxBul)

   CLS
   CALL DekkerBrentCall(a, b, x(), MaxBul, iBulundu)

  IF iBulundu = 0 THEN
   PRINT "["; a; ","; b; "] aralığında kök bulunamadı(DekkerBrent)"
  ELSE
   PRINT "["; a; ","; b; "] aralığında"; iBulundu; " kök bulundu(DekkerBrent)"
   FOR i = 1 TO iBulundu
    PRINT "x"; i; "="; x(i), "f(x"; i; ")="; fnf(x(i))
   NEXT i
  END IF

END ' DekkerBrent ana sonu
```

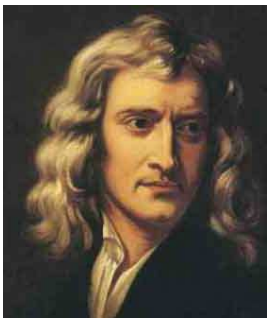
```
SUB DekkerBrentCall (a, b, x(), MaxBul, iBulundu)
'-------------------------------------------------
' DekkerBrent alt programını çağırır, bulunan kökü
' x vektöründe depolar
' iAltaralik: a-b aralığının bölüneceği
'             alt aralık sayısıdır, değiştirilebilir
' Maxbul: aranacak max kök sayısı
' iBulundu: bulunanbilen kök sayısı
' Çağrılan alt program: DekkerBrent
'-------------------------------------------------
  iAltaralik = 100
  iBulundu = 0
  d = (b - a) / iAltaralik
  x2 = a
  WHILE (x2 < b) AND (iBulundu < MaxBul)
   x1 = x2: x2 = x1 + d
   IF x2 > b THEN x2 = b

   x = DekkerBrent(x1, x2, iErr)

   IF iErr = 0 THEN
    iBulundu = iBulundu + 1
    x(iBulundu) = x
   END IF

  WEND

END SUB ' DekkerBrentCall sonu
```

```
FUNCTION DekkerBrent (x1, x2, iErr)
'------------------------------------------------------------------------
' This program finds a real root of a real function f(x)
' using Dekker-Brent method
' function f(x) must be declared in main program as def Fnf(x)=...

' INPUTS
'   x1,x2 : interval to be search for a root of f(x)
'   Maxit : maximum number of iterations

' OUTPUTS
'   DekkerBrent : root of function f(x), if found
'   iErr        =0 all OK
'               =1 no root found in interval [x1,x2]
'               =2 no more iterations(Maxit exeded)

' Fortran kodu 'Numerical Recipes in fORTRAN 77, Chapter 9.3' den alınmış,
' biraz değiştirilmiştir. Programın orijinal adı: zbrent
' http://www.haoli.org/nr/bookf.html

' Çağrılan alt program: yok
'------------------------------------------------------------------------
  Tol = 1E-14: ' Hassasiyet, 1E-8 - 1E-14 arası normal
  Maxit = 100: ' Aralık yarılama sayısı, 10-200 arası normal

' Machep
  Eps = 1
  DO
   Eps = Eps / 2
   s = 1 + Eps
  LOOP UNTIL s <= 1
  Eps = 3 * Eps

  Zero = 1E-08: ' değiştirmeyiniz
  iErr = 0

  a = x1
  b = x2
  Fa = fnf(a)
  Fb = fnf(b)
  IF (Fa > 0 AND Fb > 0) OR (Fa < 0 AND Fb < 0) THEN
   iErr = 1
   EXIT FUNCTION
  END IF

  c = b
  Fc = Fb
  FOR iTer = 1 TO Maxit
   IF (Fb > 0 AND Fc > 0) OR (Fb < 0 AND Fc < 0) THEN
    c = a
    Fc = Fa
    d = b - a
    e = d
   END IF

   IF ABS(Fc) < ABS(Fb) THEN
    a = b
    b = c
    c = a
    Fa = Fb
    Fb = Fc
    Fc = Fa
   END IF

   Tol1 = 2 * Eps * ABS(b) + .5 * Tol
   xm = .5 * (c - b)
   IF (ABS(xm) <= Tol1) AND (ABS(Fb) <= Zero) THEN
' A root has been found
    DekkerBrent = b
    EXIT FUNCTION
   END IF

   IF (ABS(e) >= Tol1) AND (ABS(Fa) > ABS(Fb)) THEN
    s = Fb / Fa
    IF ABS(a - c) < Zero THEN
     p = 2 * xm * s
     q = 1 - s
    ELSE
     q = Fa / Fc
     r = Fb / Fc
     p = s * (2 * xm * q * (q - r) - (b - a) * (r - 1))
     q = (q - 1) * (r - 1) * (s - 1)
    END IF

  IF p > Zero THEN q = -q
    p = ABS(p)

    aMin = 3 * xm * q - ABS(Tol1 * q)
    amin1 = ABS(e * q)
    IF amin1 < aMin THEN aMin = amin1
    IF 2 * p < aMin THEN
     e = d
     d = p / q
    ELSE
     d = xm
     e = d
    END IF

   ELSE
    d = xm
    e = d
   END IF

   a = b
   Fa = Fb
   IF ABS(d) > Tol1 THEN
    b = b + d
   ELSE
    IF xm >= 0 THEN
     b = b + ABS(Tol1)
    ELSE
     b = b - ABS(Tol1)
    END IF
   END IF

   Fb = fnf(b)
  NEXT iTer

  iErr = 2

  DekkerBrent = 0

END FUNCTION ' End of DekkerBrent
```

Newton-Raphson(Newton) metodu

NewtonRaphson programı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun n tane olan tüm sanal ve/veya gerçek köklerini bulur. Aşağıdaki örnekler için test edilmiştir.

Örnekler:



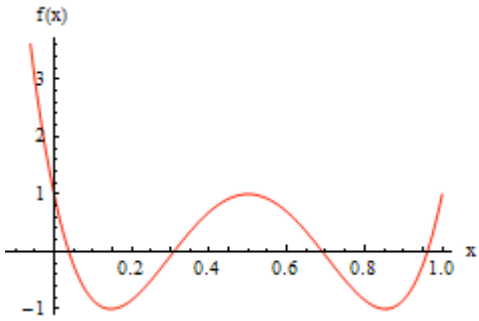
Isaac **Newton,** 1643-1727, İngiliz

1. $f(x)=128x^4-256x^3+160x^2-32x+1$
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4$
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040$
4. $f(x)=x^8-1$
5. $f(x)=x^5-3.7x^4+7.4x^3-10.8x^2+10.8x-6.8$
6. $f(x)=x^3-21x-20$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$
8. $f(x)=x^4-10x^3+35x^2-50x+24$

NewtonRaphson sonuçları:

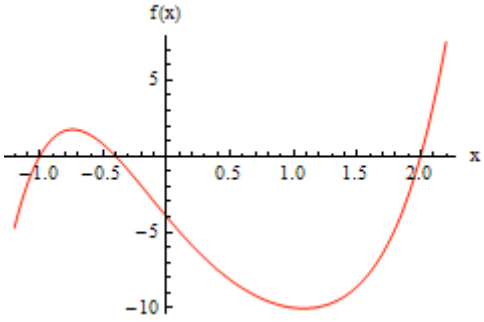
1. $f(x)=128x^4-256x^3+160x^2-32x+1=0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1     3.80602337443566D-02      0
     2     .308658283817455          0
     3     .691341716182545          0
     4     .961939766255643          0
```



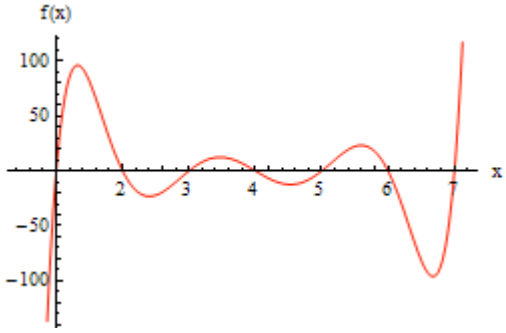
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4=0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1    -.402627941186124          0
     2    -1                         0
     3     2                         0
     4     1.20131397059306         -1.87728790691624
     5     1.20131397059306          1.87728790691624
```



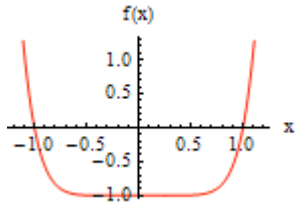
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040=0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1     1                         0
     2     2.00000000000001          0
     3     2.9999999999997           0
     4     4.00000000000072          0
     5     5.00000000000151          0
     6     6.00000000000032          0
     7     6.99999999999968          0
```



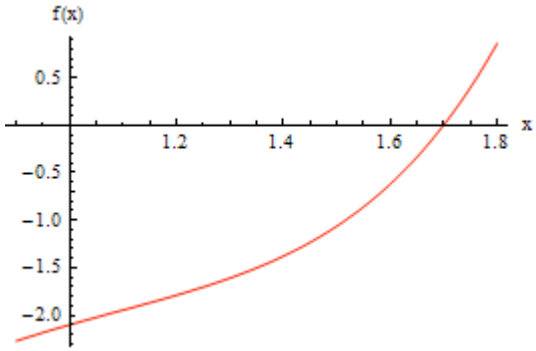
4. $f(x) = x^8 - 1 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1     1                         0
     2    -1                         0
     3     .707106781186548         -.707106781186548
     4     .707106781186548          .707106781186548
     5    -.707106781186548         -.707106781186548
     6    -.707106781186548          .707106781186548
     7     0                        -1
     8     0                         1
```



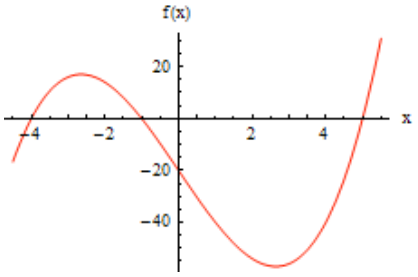
5. $f(x) = x^5 - 3.7x^4 + 7.4x^3 - 10.8x^2 + 10.8x - 6.8 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1     1.7                       0
     2    -7.02205094185976D-17     -1.4142135623731
     3    -7.02205094185976D-17      1.4142135623731
     4     1                        -1
     5     1                         1
```



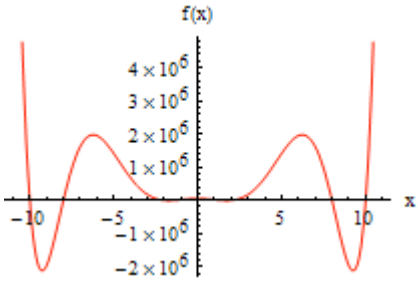
6. $f(x) = x^3 - 21x - 20 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1    -1                         0
     2    -4                         0
     3     5                         0
```



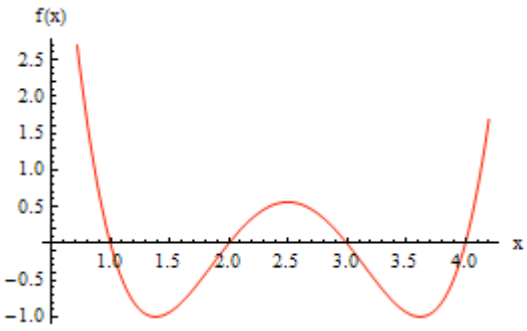
7. $f(x) = x^8 - 170x^6 + 7392x^4 - 39712x^2 + 51200 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1    -2                         0
     2     2                         0
     3    -8                         0
     4     1.4142135623731           0
     5    -1.4142135623731           0
     6     8                         0
     7    -10                        0
     8     10                        0
```



8. $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(NewtonRapson):

Kök No    Gerçek kısım              Sanal kısım
     1     1                         0
     2     2                         0
     3     3.00000000000001          0
     4     4                         0
```



NewtonRaphson

```
'-----------------Ana Program NewtonRaphson----------------------
' n. dereceden f(x) polinomunun sanal ve/veya gerçek köklerini bulur
'Çağrılan program: NewtonRaphson
'----------------------------------------------------------------
DEFINT M-N
DEFDBL A-H, O-Z
DECLARE SUB NewtonRaphson (n, Coeff(), rPart(), sPart(), ihata)
' Fonksiyonun derecesi n ve a0, a1,a2,..,an katsayıları
  n = 4: DATA 1,-32,160,-256,128
  'n = 5: DATA -4,-10,2,4,-3,1
  'n = 7: DATA -5040,13068,-13132,6769,-1960,322,-28,1
  'n = 8: DATA -1,0,0,0,0,0,0,1
  'n = 5: DATA -6.8,10.8,-10.8,7.4,-3.7,1
  'n = 3: DATA -20,-21,0,1
  'n = 8: DATA 51200,0,-39712,0,7392,0,-170,0,1
  'n = 4: DATA 24,-50,35,-10,1
  DIM Coeff(n + 1):  ' Katsayılar
  DIM rPart(n) 'gerçek kökler
  DIM sPart(n): ' sanal kökler
  FOR i = 1 TO n + 1
   READ Coeff(i)
  NEXT i
  CLS
  CALL NewtonRaphson(n, Coeff(), rPart(), sPart(), ihata)

  IF ihata <> 0 THEN
   PRINT "Kökler bulunamadı(NewtonRaphson)"
   END
  END IF

  PRINT "Polinomun kökleri(NewtonRapson):"
  PRINT
  PRINT "Kök No"; TAB(10); "Gerçek kısım"; TAB(35); "Sanal kısım"
  FOR i = 1 TO n
   PRINT TAB(5); i; TAB(10); rPart(i); TAB(35); sPart(i)
  NEXT i

END ' NewtonRaphson ana
```

```
SUB NewtonRaphson (n, Coeff(), rPart(), sPart(), iHata)
'-------------------------------------------------------------------------
' f(x)=a0+a1*x+a2*x^2+...+an*x^n polinomunun gerçek ve/veya sanal köklerinin esabı
' Metot: Newton-Raphson
' n+1 katsayı Coeff(n+1) vektöründe aşağıdaki gibi depolanmış olmalıdır:
' coeff(1)=a0, coeff(2)=a2,...,coeff(n+1)=an.
' n: polinomun derecesi
' eps : hassasiyet
' Maxit :maximum iterasyon sayısı
' rpart(n): köklerin gerçek kısmı
' sPart: köklerin sanal kısmı
' çağrılan alt program: yok
'-------------------------------------------------------------------------
  DIM b(n + 1) 'ara işlem vektörü
  Maxit = 200
  eps = 1E-08
  iFit = 0
  n9 = n
  nx = n9
  nxx = n9 + 1
  n2 = 1
  k = n9 + 1
  FOR L = 1 TO k
   b(k - L + 1) = Coeff(L)
  NEXT L

3  x0 = .00500101# ' # işareti DOUBLE anlamındadır
  y0 = .01000101#
  in = 0

4  x = x0
  x0 = -10 * y0
  y0 = -10 * x
  x = x0
  y = y0
  in = in + 1
  GOTO 6

5  iFit = 1
  xpr = x
  ypr = y
```

NewtonRaphson sonraki sayfada devam ediyor

NewtonRaphson devamı

```
6  iCt = 0
7  ux = 0
   uy = 0
   v = 0
   xt = 1
   yt = 0
   u = b(n9 + 1)
   IF u = 0 GOTO 11

   FOR i = 1 TO n9
    L = n9 - i + 1
    temp = b(L)
    xt2 = x * xt - y * yt
    yt2 = x * yt + y * xt
    u = u + temp * xt2
    v = v + temp * yt2
    ux = ux + i * xt * temp
    uy = uy - i * yt * temp
    xt = xt2
    yt = yt2
   NEXT I

sumsq = ux * ux + uy * uy
   IF sumsq = 0 GOTO 9
   dx = (v * uy - u * ux) / sumsq
   x = x + dx
   dy = -(u * uy + v * ux) / sumsq
   y = y + dy
   IF ABS(dy) + ABS(dx) < eps GOTO 8
   iCt = iCt + 1
   IF iCt <= Maxit GOTO 7
   IF iFit <> 0 GOTO 8
   IF in < 5 GOTO 4
   iHata = 2 ' maksimum iterasyon sayısı aşıldı
   EXIT SUB

8  FOR L = 1 TO nxx
    mt = k - L + 1
    temp = Coeff(mt)
    Coeff(mt) = b(L)
    b(L) = temp
   NEXT L
   iTemp = n9
   n9 = nx
   nx = iTemp
   IF iFit = 0 THEN GOTO 5 ELSE GOTO 10

9  IF iFit = 0 GOTO 4
   x = xpr
   y = ypr

10 iFit = 0
   IF ABS(x) >= 1E-20 THEN IF ABS(y / x) < 1E-10 GOTO 12
   IF ABS(x) < 1E-20 THEN x = 0
   alpha = x + x
   sumsq = x * x + y * y
   n9 = n9 - 2
   GOTO 13

11 x = 0
   nx = nx - 1
   nxx = nxx - 1

12 y = 0
   sumsq = 0
   alpha = x
   n9 = n9 - 1

13 b(2) = b(2) + alpha * b(1)
   FOR L = 2 TO n9
    b(L + 1) = b(L + 1) + alpha * b(L) - sumsq * b(L - 1)
   NEXT L

14 sPart(n2) = y
   rPart(n2) = x
   n2 = n2 + 1
   IF sumsq = 0 GOTO 15
   y = -y
   sumsq = 0
   GOTO 14

15 IF n9 > 0 THEN GOTO 3

END SUB ' NewtonRaphson
```

Bairstow metodu

Bairstow programı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun n tane olan tüm sanal ve/veya gerçek köklerini bulur. Aşağıdaki örnekler için test edilmiştir.

Leonard **Bairstow**
1880-1963, İngiliz

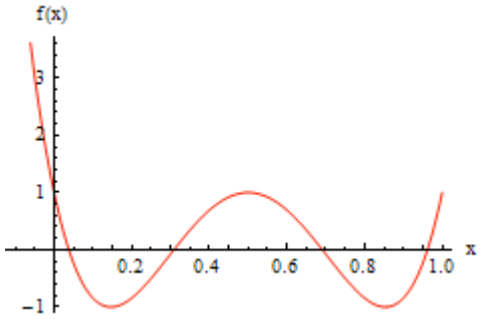
Örnekler:

1. $f(x)=128x^4-256x^3+160x^2-32x+1$
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4$
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040$
4. $f(x)=x^8-1$
5. $f(x)=x^5-3.7x^4+7.4x^3-10.8x^2+10.8x-6.8$
6. $f(x)=x^3-21x-20$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$
8. $f(x)=x^4-10x^3+35x^2-50x+24$

Bairstow sonuçları:

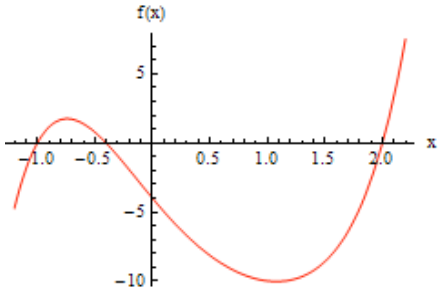
1. $f(x)=128x^4-256x^3+160x^2-32x+1=0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1     .691341716182554          0
     2     .961939766255638          0
     3     3.806023374435660-02      0
     4     .308658283817455          0
```



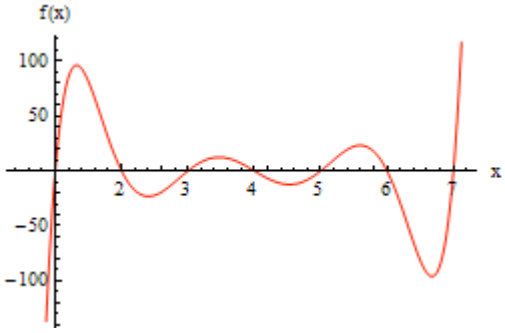
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4=0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1    -.402627941185324          0
     2     1.20131397059296         -1.87728790691623
     3     1.20131397059296          1.87728790691623
     4    -1                         0
     5     2                         0
```



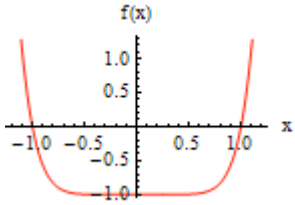
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040=0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1     5.00000000212871          0
     2     3.00000000006263          0
     3     3.99999999993396          0
     4     2.00000000000088          0
     5     5.99999999999981          0
     6     1                         0
     7     7.00000000000042          0
```



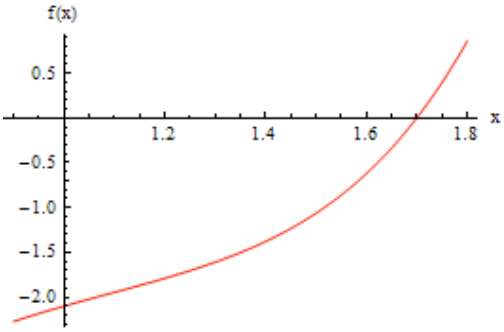
4. $f(x) = x^8 - 1 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1    -.707106781567498         -.707106781208143
     2    -.707106781567498          .707106781208143
     3     .707106781574345         -.707106781959346
     4     .707106781574345          .707106781959346
     5    -3.850391445526D-10       -1.00000000054839
     6    -3.850391445526D-10        1.00000000054839
     7    -1                         0
     8     1                         0
```



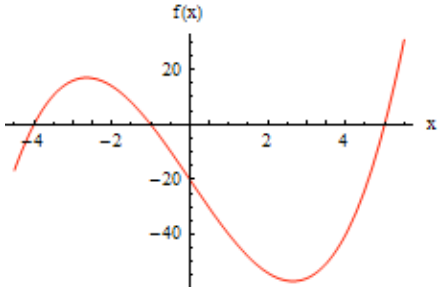
5. $f(x) = x^5 - 3.7x^4 + 7.4x^3 - 10.8x^2 + 10.8x - 6.8 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1     1.7000000000249          0
     2     1.00000000000039         -.999999999932704
     3     1.00000000000039          .999999999932704
     4     3.72066223228296D-17     -1.4142135623731
     5     3.72066223228296D-17      1.4142135623731
```



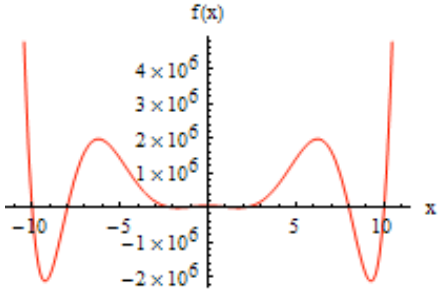
6. $f(x) = x^3 - 21x - 20 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1    -.999999999999917         0
     2    -4                        0
     3     5                        0
```



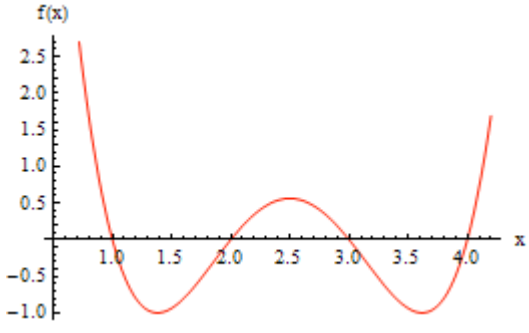
7. $f(x) = x^8 - 170x^6 + 7392x^4 - 39712x^2 + 51200 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1    -10.00000000000045        0
     2     10.00000000000045        0
     3    -7.99999999998793         0
     4     7.99999999998793         0
     5    -2.00000000056097         0
     6     2.00000000056097         0
     7    -1.4142135623731          0
     8     1.4142135623731          0
```



8. $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24 = 0$

```
C:\Basic\QBasic.EXE
Polinomun kökleri(Bairstow):
Kok no    Gerçek kısım              Sanal kısım
     1     2.00000000000023         0
     2     2.99999999999962         0
     3     1                        0
     4     4                        0
```



Bairstow

```
'-------------------------Ana Program Bairstow----------------------
' n. dereceden f(x) polinomunun sanal ve/veya gerçek köklerini bulur
' Çağrılan alt program: Bairstow
'-------------------------------------------------------------------
DEFINT I-N
DEFDBL A-H, O-Z
DECLARE SUB Bairstow (n, a(), rPart(), sPart(), iHata)

' f(x) polinomunu derecesi n ve katsayıları an,..., a2,a1
  n = 4: DATA 128,-256,160,-32,1
  'n = 5: DATA 1,-3,4,2,-10,-4
  'n = 7: DATA 1,-28,322,-1960,6769,-13132,13068,-5040
  'n = 8: DATA 1,0,0,0,0,0,0,0,-1
  'n = 5: DATA 1,-3.7,7.4,-10.8,10.8,-6.8
  'n = 3: DATA 1,0,-21,-20
  'n = 8: DATA 1,0,-170,0,7392,0,-39712,0,51200
  'n = 4: DATA 1,-10,35,-50,24

  DIM a(n + 3) ' katsayılar vektörü
  DIM rPart(n) ' kökün gerçek kısmı
  DIM sPart(n) ' kökün sanal kısmı

  FOR i = 3 TO n + 3
   READ a(i)
  NEXT i

  CLS

  CALL Bairstow(n, a(), rPart(), sPart(), iHata)

  IF iHata <> 0 THEN
   PRINT "Hata:"; iHata; ", Kök bulunamadı (Bairstow)"
   END
  END IF

  PRINT "Polinomun kökleri(Bairstow):"
  PRINT "Kok no"; TAB(10); "Gerçek kısım"; TAB(35); "Sanal kısım"
  FOR i = 1 TO n
   PRINT TAB(5); i; TAB(10); rPart(i); TAB(35); sPart(i)
  NEXT i

END ' Baistow ana sonu
```

```
SUB Bairstow (n, a(), rPart(), sPart(), iHata)
'-------------------------------------------------------------------------
' f(x)=an*x^n+...+a3*x^3+a2*x^2+a1*x+a0 polinomunun sanal ve gerçek köklerin bulur.
'Metod: Bairstow
'veri:
' a(n+3): katsayıların depolandığı n+3 boyutlu vektör. n+1 katsayı aşağıdaki gibi
' depolanmış olmalıdır: a(3)=an,..., a(n+1)=a2,a(n+2)=a1,a(n+3)=a0
' n: polinomun derecesi
'çıktı:
' rpart(n): köklerin gerçek kısmı
' sPart(n): köklerin sanal kısmı
' iHata =0 : kökler bulundu
'      <>0 : kökler bulunamadı

' çağrılan alt program: yok
'-------------------------------------------------------------------------
  iHata = 0   ' hata bayrağı
  eps = 1E-08 ' Hata yüzdesi
  maxit = 500 ' maksimum iterasyon
  r1 = 0      ' başlangıç değeri
  s1 = 0      ' başlangıç değeri
  FOR i = 1 TO n
   rPart(i) = 0
   sPart(i) = 0
  NEXT i

  n3 = n + 3
  DIM b(n3), c(n3) ' Ara işlem vektörleri
  n1 = n
  IF n1 > 0 AND a(3) = 0 THEN
   iHata = 1 ' Polinomun tanımı hatalı
   EXIT SUB
  END IF

  IF n1 = 1 THEN
   b(3) = a(3)
   b(4) = a(4)
   GOTO 11
  END IF
```

Bairstow sonraki sayfada devam ediyor

Bairstow devamı

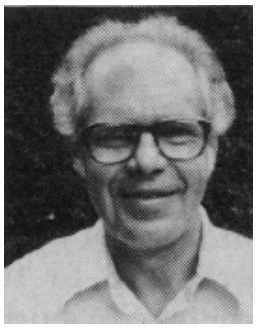
```
3  n3 = n1 + 3
   IF n1 = 2 THEN
    r = -a(4) / a(3)
    s = -a(5) / a(3)
    GOTO 7
   END IF
   iSay = 1
   b(1) = 0
   b(2) = 0
   c(1) = 0
   c(2) = 0

4  r = r1
   s = s1
5  FOR i = 3 TO n3
    b(i) = a(i) + r * b(i - 1) + s * b(i - 2)
    c(i) = b(i) + r * c(i - 1) + s * c(i - 2)
   NEXT i
   f = c(n1 + 1) ^ 2 - c(n1 + 2) * c(n1)
   IF f = 0 THEN
    r1 = r1 + 1
    s1 = s1 + 1
    GOTO 4
   END IF

   deLr = (-b(n1 + 2) * c(n1 + 1) + c(n1) * b(n3)) / f
   deLi = (-c(n1 + 1) * b(n3) + b(n1 + 2) * c(n1 + 2)) / f
   r = r + deLr
   s = s + deLi
   IF ABS(deLr) + ABS(deLi) <= eps GOTO 7

' maks iterasyon aşıldı mı?
   IF iSay >= maxit THEN
    iHata = 2 ' Maks iterasyon aşıldı
    EXIT SUB
   END IF

6  iSay = iSay + 1
   GOTO 5

7  diskr = s + .25 * r * r
   IF diskr < 0 GOTO 8
   IF diskr > 0 GOTO 9
   x1 = .5 * r
   x2 = x1
   y1 = 0: y2 = 0
   GOTO 10

8  x1 = .5 * r
   x2 = x1
   y1 = SQR(-diskr)
   y2 = -y1
   GOTO 10

9  x1 = .5 * r + SQR(diskr)
   x2 = -s / x1
   y1 = 0
   y2 = 0

10 rPart(n1) = x1
   sPart(n1) = y1
   rPart(n1 - 1) = x2
   sPart(n1 - 1) = y2
   n1 = n1 - 2
   IF n1 < 1 THEN EXIT SUB
   IF n1 = 1 GOTO 11
   n3 = n1 + 3
   FOR i = 3 TO n3
    a(i) = b(i)
   NEXT i
   GOTO 3

11 x1 = -b(4) / b(3)
   rPart(n1) = x1
   sPart(n1) = 0

END SUB ' Bairstow
```

Müller metodu

Muller programı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun gerçek köklerini bulur, sanal kök bulmaz. Aşağıdaki örnekler için test edilmiştir.

Örnekler:

1. $f(x)=128x^4-256x^3+160x^2-32x+1$
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4$
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040$
4. $f(x)=x^8-1$
5. $f(x)=x^5-3.7x^4+7.4x^3-10.8x^2+10.8x-6.8$
6. $f(x)=x^3-21x-20$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$
8. $f(x)=x^4-10x^3+35x^2-50x+24$

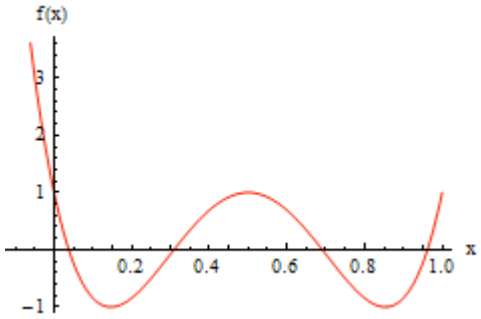


David E. **Müller**, 1924-2008
Amerikalı

Muller sonuçları:

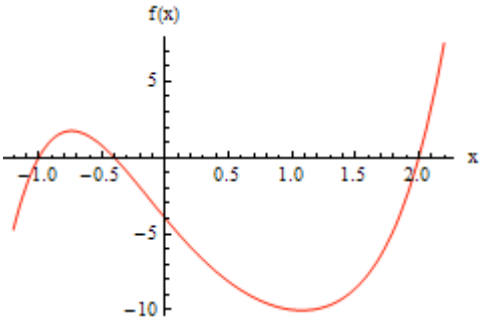
1. $f(x)=128x^4-256x^3+160x^2-32x+1=0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                          f(x)
 1         3.80602337443566D-02      -3.93565388612238D-17
 2         .308658283817455           1.96023752785379D-16
 3         .691341716182545          -2.77555756156289D-16
 4         .961939766255643          -1.66533453693774D-16
```



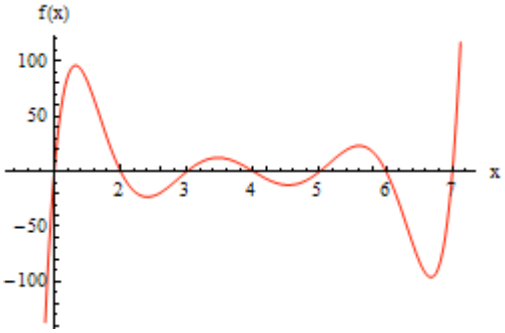
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4=0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                          f(x)
 1        -.402627941186124          -1.96674274088871D-16
 2        -1                          0
 3         2                          0
```



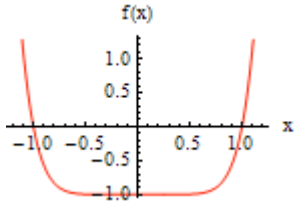
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040=0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                          f(x)
 1         1                          0
 2         3                          2.1316282072803D-14
 3         4                         -1.06581410364015D-14
 4         2                          0
 5         6                          6.67910171614494D-13
 6         5                          2.38031816479634D-13
 7         7                         -2.20268248085631D-12
```



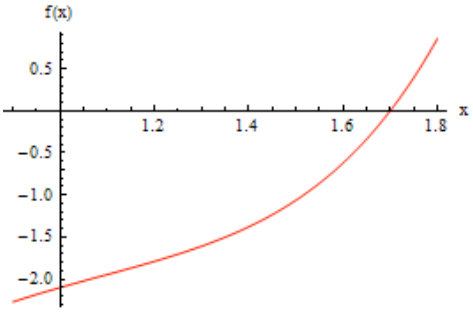
4. $f(x) = x^8 - 1 = 0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                        f(x)
 1         1                        0
 2        -1                       -1.77635683940025D-15
```



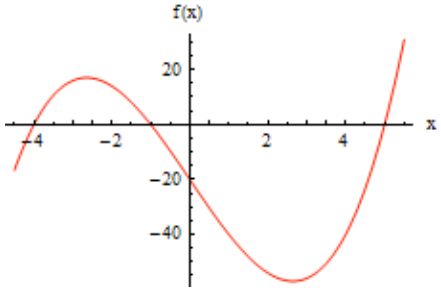
5. $f(x) = x^5 - 3.7x^4 + 7.4x^3 - 10.8x^2 + 10.8x - 6.8 = 0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                        f(x)
 1         1.70000004768372         -6.93889390390723D-18
```



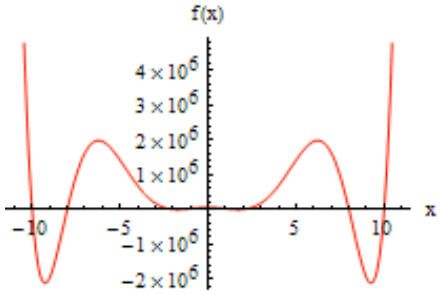
6. $f(x) = x^3 - 21x - 20 = 0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                        f(x)
 1        -1                        0
 2        -4                        0
 3         5                       -1.38777878078145D-17
```



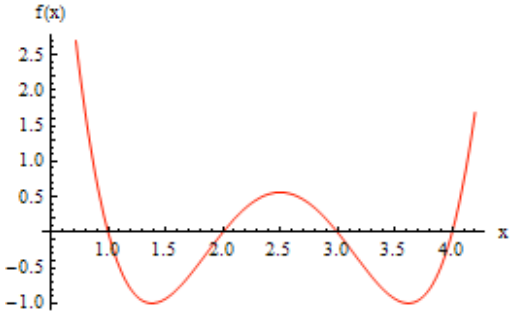
7. $f(x) = x^8 - 170x^6 + 7392x^4 - 39712x^2 + 51200 = 0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                        f(x)
 1        -1.4142135623731         -3.21787287967847D-12
 2         1.4142135623731         -3.21787287967847D-12
 3         2                        0
 4        -2                        0
 5        -8                        0
 6         8                        0
 7         10                      -1.30967237055302D-10
 8        -10                      -1.30967237055302D-10
```



8. $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24 = 0$

```
C:\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(Muller):
Kök no    x                        f(x)
 1         1                        0
 2         3                       -2.77555756156289D-17
 3         2                        0
 4         4                        0
```



Muller

```
'----------------------------Ana Program Muller-----------------------------------------
' n. dereceden f(x) polinomunun sadece gerçek köklerini bulur, sanal kök bulmaz.
' Çağrılan alat program: Muller
`-----------------------------------------------------------------------------------------
DEFINT I-N
DEFDBL A-H, O-Z
DECLARE SUB Muller (n, zeros(), iFound)
' polinomun ve derecesinin tanımı
   n = 4: DEF fnf (x) = 128 * x ^ 4 - 256 * x ^ 3 + 160 * x ^ 2 - 32 * x + 1
   'n = 5: DEF fnf (x) = x ^ 5 - 3 * x ^ 4 + 4 * x ^ 3 + 2 * x ^ 2 - 10 * x - 4
   'n = 7: DEF fnf (x) = x ^ 7 - 28 * x ^ 6 + 322 * x ^ 5 - 1960 * x ^ 4 + 6769 * x ^ 3 - 13132 * x ^ 2 + 13068 * x - 5040
   'n = 8: DEF fnf (x) = x ^ 8 - 1
   'n = 5: DEF fnf (x) = x ^ 5 - 3.7 * x ^ 4 + 7.4 * x ^ 3 - 10.8 * x ^ 2 + 10.8 * x - 6.8
   'n = 3: DEF fnf (x) = x ^ 3 - 21 * x - 20
   'n = 8: DEF fnf (x) = x ^ 8 - 170 * x ^ 6 + 7392 * x ^ 4 - 39712 * x * x + 51200
   'n = 4: DEF fnf (x) = x ^ 4 - 10 * x ^ 3 + 35 * x ^ 2 - 50 * x + 24

   DIM zeros(n): ' vector of zeros found

   CLS

   CALL Muller(n, zeros(), iFound)

   IF iFound = 0 THEN PRINT "Kök bulunamadı(Muller)": END

   PRINT "Polinomun bulunabilen kökleri(Muller):"
   PRINT "Kök no"; TAB(10); "x"; TAB(35); "f(x)"
   FOR i = 1 TO iFound
    PRINT i; TAB(10); zeros(i); TAB(35); fnf(zeros(i))
   NEXT i

END ' Muller ana
```

```
SUB Muller (n, zeros(), iFound)
'-------------------------------------------------------------------------
' f(x)=an*x^n+...+a3*x^3+a2*x^2+a1*x+a0 polinomunun gerçek ve/veya sanal köklerinin hesabı
' Metod: Müller
' f(x) fonksiyonu ana programda tanımlanmış olmalıdır
' n boyutlu Zeros(n) vektörü ana programda boyutlandırılmış olmalıdır
' n: polinomun derecesi
' iFound: bulunabilen kök sayısı
' zeros(n): köklerin depolandığı vektör
' maxit: herbir kök için maksimum iterasyon sayısı
' epsf: hassasiyet

' çağrılan alt program: deflate
'-------------------------------------------------------------------------
   DIM x(n)
   epsf = 1E-08
   maxit = 50

   FOR i = 1 TO n
    x(i) = 0
   NEXT i

   iFound = 0
   FOR i = 1 TO n
    iCount = 0
2   x2 = x(i)
    x0 = x2 - .5
    x1 = x2 + .5

    z = x0
    GOSUB 4
    f0 = fzrdfL

    z = x1
    GOSUB 4
    f1 = fzrdfL

    z = x2
    GOSUB 4
    f2 = fzrdfL

    h1 = x1 - x0
    h2 = x2 - x1
    f21 = (f2 - f1) / h2
    f10 = (f1 - f0) / h1
```

Muller sonraki sayfada devam ediyor

Muller devamı

```
3   f210 = (f21 - f10) / (h2 + h1)
    c = f21 + h2 * f210
    sq = c * c - 4 * f2 * f210

    IF sq < 0 THEN sq = 0 ' sanal kökü atla

    IF c < 0 THEN sq = c - SQR(sq) ELSE sq = c + SQR(sq)
    IF ABS(sq) = 0 THEN h3 = -2 * f2 ELSE h3 = -2 * f2 / sq

5   x3 = x2 + h3
    z = x3
    GOSUB 4
    f3 = fzrdfL

' Yakınsama var mı?
    IF ABS(f2) < epsf THEN
     iFound = iFound + 1
     zeros(iFound) = x3
     GOTO 6
    END IF

' maksimum iterasyon sayısı aşıldı mı?
    IF iCount > maxit GOTO 6

' ıraksama var mı?
    IF ABS(f3) >= 5 * ABS(f2) THEN
     h3 = .5 * h3
     GOTO 5
    END IF

' değerleri aktar
    f32 = (f3 - f2) / h3
    f10 = f21
    f21 = f32
    h1 = h2
    h2 = h3
    f2 = f3
    x2 = x3
    GOTO 3
6   x(i) = x3
   NEXT i

   EXIT SUB

' Alt program deflate
4  iCount = iCount + 1
   fzr = fnf(z): fzrdfL = fzr
   IF i < 2 THEN RETURN
   FOR j = 2 TO i
    sq = z - x(j - 1)
    IF ABS(sq) = 0 THEN
     x(i) = z * 1.001
     RETURN
    END IF

    fzrdfL = fzrdfL / sq
   NEXT j
   RETURN

END SUB ' Muller sonu
```

Jenkins-Traub metodu

JenkinsTraub programı $f(x)=a_0+a_1x+a_2x^2+\ldots+a_nx^n$ polinomunun n tane olan tüm sanal ve/veya gerçek köklerini bulur. Aşağıdaki örnekler için test edilmiştir.

Joseph Frederick **Traub**, 1932-Amerikalı

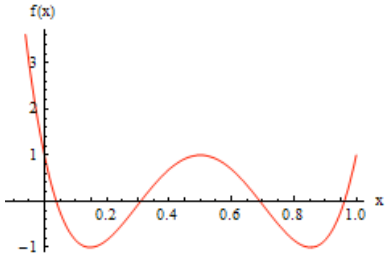
Örnekler:

1. $f(x)=128x^4-256x^3+160x^2-32x+1$
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4$
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040$
4. $f(x)=x^8-1$
5. $f(x)=x^5-3.7x^4+7.4x^3-10.8x^2+10.8x-6.8$
6. $f(x)=x^3-21x-20$
7. $f(x)=x^8-170x^6+7392x^4-39712x^2+51200$
8. $f(x)=x^4-10x^3+35x^2-50x+24$

JenkinsTraub sonuçları:

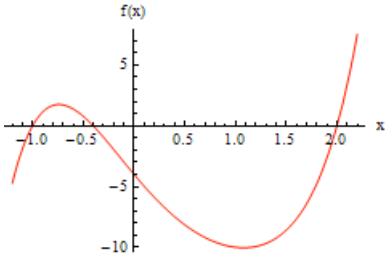
1. $f(x)=128x^4-256x^3+160x^2-32x+1=0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:            Sanal kısım:
     1     3.806023374435666D-02      0
     2     .308658283817455           0
     3     .691341716182544           0
     4     .961939766255645           0
```



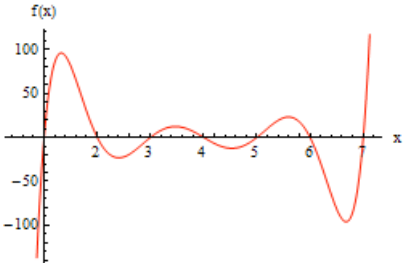
2. $f(x)=x^5-3x^4+4x^3+2x^2-10x-4=0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:            Sanal kısım:
     1    -.4026279411861 24         0
     2    -1                         0
     3     1.20131397059306          1.87728790691624
     4     1.20131397059306         -1.87728790691624
     5     2                          0
```



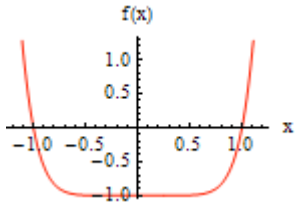
3. $f(x)=x^7-28x^6+322x^5-1960x^4+6769x^3-13132x^2+13068x-5040=0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:            Sanal kısım:
     1     1                          0
     2     1.99999999999945           0
     3     3.00000000000028           0
     4     3.99999999999415           0
     5     5.00000000000636           0
     6     5.99999999999638           0
     7     7.00000000000086           0
```



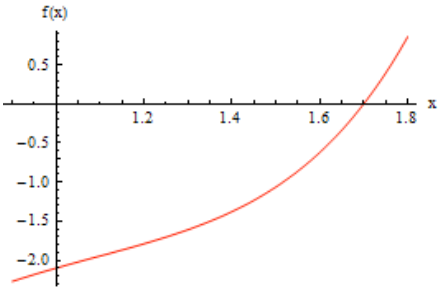
4. $f(x) = x^8 - 1 = 0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:              Sanal kısım:
     1     .707106781186548          .707106781186548
     2     .707106781186548         -.707106781186548
     3    -.707106781186548          .707106781186548
     4    -.707106781186548         -.707106781186548
     5    -1                         0
     6     1                         0
     7     0                         1
     8     0                        -1
```



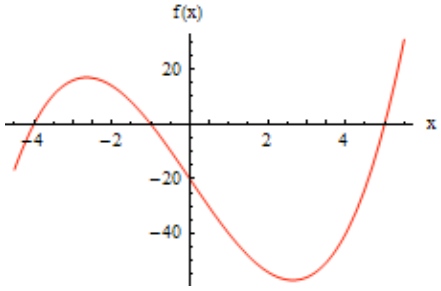
5. $f(x) = x^5 - 3.7x^4 + 7.4x^3 - 10.8x^2 + 10.8x - 6.8 = 0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:              Sanal kısım:
     1     1                          1
     2     1                         -1
     3    -2.78525455852482D-16       1.41421356237731
     4    -2.78525455852482D-16      -1.41421356237731
     5     1.7                        0
```



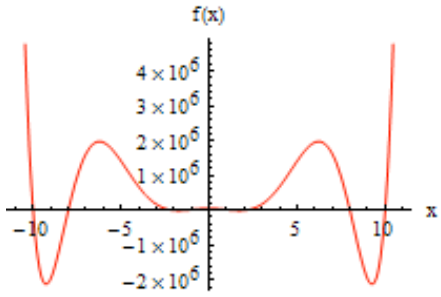
6. $f(x) = x^3 - 21x - 20 = 0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:              Sanal kısım:
     1    -1                         0
     2    -4                         0
     3     5                         0
```



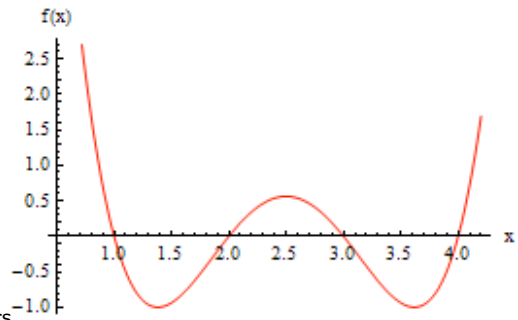
7. $f(x) = x^8 - 170x^6 + 7392x^4 - 39712x^2 + 51200 = 0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:              Sanal kısım:
     1     1.41421356237731          0
     2    -1.41421356237309          0
     3     2                         0
     4    -2.00000000000001          0
     5     8                         0
     6     10                        0
     7    -7.99999999999999          0
     8    -10                        0
```



8. $f(x) = x^4 - 10x^3 + 35x^2 - 50x + 24 = 0$

```
C:\NANALIZ\Basic\QBasic.EXE
Polinomun bulunabilen kökleri(JenkinsTraub):
Kök No:   Gerçek kısım:              Sanal kısım:
     1     1                         0
     2     2                         0
     3     3.00000000000001          0
     4     4                         0
```



JenkinsTraub

```
'------------------Ana program JenkinsTraub----------------------------
' Ahmet Topçu, Eskişehir Osmangazi Üniversitesi, 2010
' Gerçek katsayılı n. derece f(x)=an x^n+...+a2x^2+a1x+a0
' polinomunun gerçek ve sanal köklerini bulur

' i,j,k,L,m,n harfleri ile başlayan değişkenler tam sayı(INTEGER),
' tüm diğerleri çift hassasitli(DOUBLE) değişkendir.

' Metot: Jenkins-Traub

' Veri:
'   Op(n+1) : polinomun katsayılarının depolandığı n+1 boyutlu vektör
'   n     : polinomun derecesi

' Çıktı:
'   iDegree=0 kök bulunamadı
'         <>0 iDegree adet kök bulundu
'   ZeroR  : bulunan köklerin gerçek kısmının depolandığı vektör
'   Zeroi  : bulunan köklerin sanal kısmının depolandığı vektör

' Çağrılan alt program: Rpoly

' Alt programların alındığı yer: http://www.netlib.org/toms/493
'------------------------------------------------------------------------------

 DEFINT I-N
 DEFDBL A-H, O-Z

 DECLARE SUB Rpoly (op(), iDegree, Zeror(), Zeroi())
 DECLARE SUB Quadsd (nn, u, v, p(), q(), a, b)
 DECLARE SUB Fxshfr (L2, nz)
 DECLARE SUB Quadit (uu, vv, nz)
 DECLARE SUB Realit (sss, nz, iflag)
 DECLARE SUB Calcsc (itype)
 DECLARE SUB Nextk (itype)
 DECLARE SUB Newest (itype, uu, vv)
 DECLARE SUB Quad (a, b1, c, sr, si, dLr, dLi)

' f(x) polinomunun derecesi n1 ve katsayıları an,..., a2,a1
  'n = 4: DATA 128,-256,160,-32,1
  'n = 5: DATA 1,-3,4,2,-10,-4
  'n = 7: DATA 1,-28,322,-1960,6769,-13132,13068,-5040
  'n = 8: DATA 1,0,0,0,0,0,0,-1
  'n = 5: DATA 1,-3.7,7.4,-10.8,10.8,-6.8
  'n = 3: DATA 1,0,-21,-20
  'n = 8: DATA 1,0,-170,0,7392,0,-39712,0,51200
  'n = 4: DATA 1,-10,35,-50,24

' Global değişkenler
 DIM SHARED p(n + 1), qp(n + 1), dk(n + 1), qk(n + 1), svk(n + 1)
 DIM SHARED sr, si, u, v, a, b, c, d, a1, a2, a3, a6, a7, e, f, g
 DIM SHARED h, szr, szi, dLzr, dLzi, Eta, are, dmre, n1, nn

  CLS

  DIM op(n + 1), Zeror(n), Zeroi(n)

  FOR i = 1 TO n + 1
   READ op(i): ' Polinomun katsayıları
  NEXT i

  iDegree = n
  CALL Rpoly(op(), iDegree, Zeror(), Zeroi())

  IF iDegree = 0 THEN
   PRINT "Kökler bulunamadı(JenkinsTraub)"
   END
  END IF

  PRINT "Polinomun bulunabilen kökleri(JenkinsTraub):"
  PRINT "Kök No:"; TAB(10); "Gerçek kısım:"; TAB(35); "Sanal kısım:"
  FOR i = 1 TO iDegree
   PRINT TAB(5); i; TAB(10); Zeror(i); TAB(35); Zeroi(i)
  NEXT i

END ' Ana program JenkinsTraub sonu
```

```
SUB Rpoly (op(), iDegree, Zeror(), Zeroi())
'---------------------------------------------------------
' Fınds The Zeros Of A Real Polynomıal
' Op  : Double Precısıon Vector Of Coeffıcıents In
'       order of decreasıng powers.
' iDegree: Degree of polynomıal.
' ZeroR : Output vector of real parts of the zeros.

' Zeroi : Output vector of ımagınary parts of the zeros.

' All calculatıons for the ıteratıons are done ın double
' precısıon.
' Subroutines called: Quad, FXSHFR

' Fortran code:http://www.netlib.org/toms/493
'---------------------------------------------------------
     Ndim = iDegree + 1
     DIM temp(Ndim), pt(Ndim)

' Variable BASE is renamed to dBase
' LZerok is used as boolean variable

' the followıng statements set machıne constants used
' ın varıous parts of the program. the meanıng of the
' four constants are…
' eta    The maxımum relatıve representatıon error
'        whıch can be descrıbed as the smallest
'        posıtıve floatıng poınt number such that
'        1.d0+eta ıs greater than 1.
' dinFin  The largest floatıng-poınt number.
' smaLno  The smallest posıtıve floatıng-poınt number
'        ıf the exponent range dıffers ın sıngle and
'        double precısıon then smalno and dinfin
'        should ındıcate the smaller range.
' dbase   the base of the floatıng-poınt number system used.

' The values below set machine constants
     r4 = 1
     r4t = 1 + (r4 / 2)
1    IF 1 < r4t THEN
      r4 = r4 / 2
      r4t = 1 + (r4 / 2)
      GOTO 1
     END IF
     Epsilon = .5 * r4: 'machine Epsilon

     dBase = 2: ' Binary taban
     Eta = Epsilon: 'Hassasiyet
     dinFin = 1E+30: ' en büyük sayı
     smaLno = 1E-37: ' en küçük sayı
' are and dmre refer to the unıt error ın + and *
' respectıvely. they are assumed to be the same as eta.
     are = Eta
     dmre = Eta
     dLo = smaLno / Eta
' ınıtıalızatıon of constants for shıft rotatıon
     xx = SQR(.5)
     yy = -xx
     Rot = 94: ' Rotation 94 degree
     Rot = Rot * 4 * ATN(1) / 180': Convert degree to radian
     cosr = COS(Rot)
     sinr = SIN(Rot)
     n1 = iDegree
     nn = n1 + 1
' algorıthm faıls ıf the leadıng coefficıent ıs zero.
     IF op(1) = 0 THEN
      iDegree = 0
      EXIT SUB
     END IF
' remove the zeros at the orıgın ıf any
103   IF op(nn) = 0 THEN
      j = iDegree - n1 + 1
      Zeror(j) = 0
      Zeroi(j) = 0
      nn = nn - 1
      n1 = n1 - 1
      GOTO 103
     END IF
' make a copy of the coefficıents
     FOR i = 1 TO nn
       p(i) = op(i)
     NEXT i
```

Rpoly devam ediyor

```
' start the algorıthm for one zero
41    IF n1 <= 2 THEN
      IF n1 < 1 THEN EXIT SUB
' calculate the fınal zero or paır of zeros
      IF n1 <> 2 THEN
       Zeror(iDegree) = -p(2) / p(1)
       Zeroi(iDegree) = 0
       EXIT SUB
      END IF
      CALL Quad(p(1), p(2), p(3), Zeror(iDegree - 1), Zeroi(iDegree - 1), Zeror(iDegree), Zeroi(iDegree))
      EXIT SUB
     END IF
' fınd largest and smallest modulı of coefficıents.
     dMax = 0
     dMin = dinFin
     FOR i = 1 TO nn
       x = ABS(p(i))
       IF x > dMax THEN dMax = x
       IF x <> 0 AND x < dMin THEN dMin = x
     NEXT i
' scale ıf there are large or very small coefficıents
' computes a scale factor to multıply the
' coefficıents of the polynomıal. the scalıng ıs done
' to avoıd overflow and to avoıd undetected underflow
' ınterferıng wıth the convergence crıterıon.
' the factor ıs a power of the dbase
     sc = dLo / dMin
     IF sc <= 1 THEN
      IF dMax < 10 THEN GOTO 110
      IF sc = 0 THEN sc = smaLno
     ELSE
      IF (dinFin / sc) < dMax THEN GOTO 110
     END IF
     L = LOG(sc) / LOG(dBase) + .5
     factor = dBase ^ L
     IF factor <> 1 THEN
      FOR i = 1 TO nn
       p(i) = factor * p(i)
      NEXT i
     END IF
' compute lower bound on modulı of zeros.
110   FOR i = 1 TO nn
       pt(i) = ABS(p(i))
     NEXT i
     pt(nn) = -pt(nn)
' compute upper estımate of bound
     x = EXP((LOG(-pt(nn)) - LOG(pt(1))) / n1)
     IF pt(n1) <> 0 THEN
' ıf newton step at the orıgın ıs better, use ıt.
      xm = -pt(nn) / pt(n1)
      IF xm < x THEN x = xm
     END IF
' chop the ınterval (0,x) untıl ff .le. 0
130   xm = x * .1
     ff = pt(1)
     FOR i = 2 TO nn
       ff = ff * xm + pt(i)
     NEXT i
     IF ff > 0 THEN
      x = xm
      GOTO 130
     END IF
     dx = x
' do newton ıteratıon untıl x converges to two
' decımal places
160   IF ABS(dx / x) > .005 THEN
      ff = pt(1)
      df = ff
      FOR i = 2 TO n1
        ff = ff * x + pt(i)
        df = df * x + ff
      NEXT i
      ff = ff * x + pt(nn)
      dx = ff / df
      x = x - dx
      GOTO 160
     END IF
     bnd = x
```

Rpoly devam ediyor

```
' compute the derıvatıve as the ıntıal dk polynomıal
' and do 5 steps wıth no shıft
    nm1 = n1 - 1
    FOR i = 2 TO n1
      dk(i) = (nn - i) * p(i) / n1
    NEXT i
    dk(1) = p(1)
    aa = p(nn)
    bb = p(n1)
    Lzerok = (dk(n1) = 0)
    FOR jj = 1 TO 5
      cc = dk(n1)
      IF Lzerok = 0 THEN
' use scaled form of recurrence ıf value of dk at 0 ıs nonzero
       t = -aa / cc
       FOR i = 1 TO nm1
        j = nn - i
        dk(j) = t * dk(j - 1) + p(j)
       NEXT i
       dk(1) = p(1)
       Lzerok = (ABS(dk(n1)) <= ABS(bb) * Eta * 10)
      ELSE
' use unscaled form of recurrence
       FOR i = 1 TO nm1
        j = nn - i
        dk(j) = dk(j - 1)
       NEXT i
       dk(1) = 0
       Lzerok = (dk(n1) = 0)
      END IF
    NEXT jj
' save dk for restarts wıth new shıfts
    FOR i = 1 TO n1
      temp(i) = dk(i)
    NEXT i
' loop to select the quadratıc  correspondıng to each
' new shıft
    FOR icnt = 1 TO 20
' quadratıc corresponds to a double shıft to a
' non-real poınt and ıts complex conjugate. the poınt
' has modulus bnd and amplıtude rotated by 94 degrees
' from the prevıous shıft
      xxx = cosr * xx - sinr * yy
      yy = sinr * xx + cosr * yy
      xx = xxx
      sr = bnd * xx
      si = bnd * yy
      u = -2 * sr
      v = bnd
' second stage calculatıon, fıxed quadratıc
      CALL Fxshfr(20 * icnt, nz)
      IF nz <> 0 THEN
' the second stage jumps dırectly to one of the thırd
' stage ıteratıons and returns here ıf successful.
' deflate the polynomıal, store the zero or zeros and
' return to the maın algorıthm.
       j = iDegree - n1 + 1
       Zeror(j) = szr
       Zeroi(j) = szi
       nn = nn - nz
       n1 = nn - 1
       FOR i = 1 TO nn
        p(i) = qp(i)
       NEXT i
       IF nz = 1 THEN GOTO 41
       Zeror(j + 1) = dLzr
       Zeroi(j + 1) = dLzi
       GOTO 41
      END IF
' ıf the ıteratıon ıs unsuccessful another quadratıc
' ıs chosen after restorıng dk
      FOR i = 1 TO n1
        dk(i) = temp(i)
      NEXT i
    NEXT icnt
' return wıth faılure ıf no convergence wıth 202 shıfts
    iDegree = iDegree - n1

END SUB  ' Rpoly
```

```
SUB Fxshfr (L2, nz)
'-----------------------------------------------------------
' computes up to  l2  fixed shift dk-polynomials,
' testing for convergence in the linear or quadratic
' case. initiates one of the variable shift
' iterations and returns with the number of zeros found.
' l2 - limit of fixed shift steps
' nz - number of zeros found
' Subroutines called: Quadsd,Calcsc,Nextk,Newest, Quadit
'                     Realit
'-----------------------------------------------------------
' Varible TYPE is renamed to iType
' Lvpass, Lspass, Lvtry, Lstry are used as Boolean variables:
' 1=TRUE, 0=FALSE
     nz = 0
     betav = .25
     betas = .25
     oss = sr
     ovv = v
' evaluate polynomial by synthetic division
     CALL Quadsd(nn, u, v, p(), qp(), a, b)
     CALL Calcsc(itype)
     FOR j = 1 TO L2
' calculate next dk polynomial and estimate v
       CALL Nextk(itype)
       CALL Calcsc(itype)
       CALL Newest(itype, ui, vi)
       vv = vi
' estimate s
       ss = 0
       IF dk(n1) <> 0 THEN ss = -p(nn) / dk(n1)
       tv = 1
       ts = 1
       IF j <> 1 AND itype <> 3 THEN
' compute relative measures of convergence of s and v
' sequences
         IF vv <> 0 THEN tv = ABS((vv - ovv) / vv)
         IF ss <> 0 THEN ts = ABS((ss - oss) / ss)
' if decreasing, multiply two most recent
' convergence measures
         tvv = 1
         IF tv < otv THEN tvv = tv * otv
         tss = 1
         IF ts < ots THEN tss = ts * ots
' compare with convergence criteria
         Lvpass = tvv < betav
         Lspass = tss < betas
         IF Lspass OR Lvpass THEN
' at least one sequence has passed the convergence
' test. store variables before iterating
          svu = u
          svv = v
          FOR i = 1 TO n1
           svk(i) = dk(i)
          NEXT i
          s = ss
' choose iteration according to the fastest
' converging sequence
          Lvtry = 0: ' set to FALSE
          Lstry = 0: ' set to FALSE
          IF Lspass AND (Lvpass = 0 OR tss < tvv) THEN GOTO 40
21         CALL Quadit(ui, vi, nz)
          IF nz > 0 THEN EXIT SUB
' quadratic iteration has failed. flag that it has
' been tried and decrease the convergence criterion.
          Lvtry = 1: 'Set to  TRUE
          betav = betav * .25
' try linear iteration if it has not been tried and
' the s sequence is converging
          IF Lstry OR Lspass = 0 THEN GOTO 50
          FOR i = 1 TO n1
           dk(i) = svk(i)
          NEXT i
40         CALL Realit(s, nz, iflag)
          IF nz > 0 THEN EXIT SUB
' linear iteration has failed. flag that it has been
' tried and decrease the convergence criterion
          Lstry = 1: ' set to TRUE
          betas = betas * .25
          IF iflag <> 0 THEN
' if linear iteration signals an almost double real
' zero attempt quadratic interation
           ui = -(s + s)
           vi = s * s
           GOTO 21
          END IF
' restore variables
50         u = svu
          v = svv
          FOR i = 1 TO n1
           dk(i) = svk(i)
          NEXT i
' try quadratic iteration if it has not been tried
' and the v sequence is converging
          IF Lvpass AND Lvtry = 0 THEN GOTO 21
' recompute qp and scalar values to continue the
' second stage
          CALL Quadsd(nn, u, v, p(), qp(), a, b)
          CALL Calcsc(itype)
         END IF
        END IF
        ovv = vv
        oss = ss
        otv = tv
        ots = ts
      NEXT j

END SUB 'Fxshfr
```

```
SUB Quadit (uu, vv, nz)
'---------------------------------------------------------------------
' varıable-shıft dk-polynomıal ıteratıon for a
' quadratıc factor converges only ıf the zeros are
' equımodular or nearly so.
' uu,vv - coeffıcıents of startıng quadratıc
' nz - number of zero found
' Subroutines called: Quad,Quadsd,Calcsc,Newest, Nextk,Quadit
'---------------------------------------------------------------------
' Ltried is used as boolean variable
     nz = 0
     Ltried = 0: 'set to FALSE
     u = uu
     v = vv
     j = 0
' maın loop
101   CALL Quad(1, u, v, szr, szi, dLzr, dLzi)
' return ıf roots of the quadratıc are real and not
' close to multıple or nearly equal and  of opposıte sıgn
     IF ABS(ABS(szr) - ABS(dLzr)) > .01 * ABS(dLzr) THEN EXIT SUB
' evaluate polynomıal by quadratıc synthetıc dıvısıon
     CALL Quadsd(nn, u, v, p(), qp(), a, b)
     dmp = ABS(a - szr * b) + ABS(szi * b)
' compute a rıgorous  bound on the roundıng error ın evalutıng p
     zm = SQR(ABS(v))
     ee = 2 * ABS(qp(1))
     t = -szr * b
     FOR i = 2 TO n1
       ee = ee * zm + ABS(qp(i))
     NEXT i
     ee = ee * zm + ABS(a + t)
     ee = (5 * dmre + 4 * are) * ee - (5 * dmre + 2 * are) * (ABS(a + t) + ABS(b) * zm) + 2 * are * ABS(t)
' ıteratıon has converged sufficıently ıf the
' polynomıal value ıs less than 20 tımes thıs bound
     IF dmp <= 20 * ee THEN
      nz = 2
      EXIT SUB
     END IF
     j = j + 1
' stop ıteratıon after 20 steps
     IF j > 20 THEN EXIT SUB
     IF j >= 2 THEN
     IF (relstp > .01 OR dmp < omp OR Ltried) = 0 THEN
' a cluster appears to be stallıng the convergence.
' fıve fıxed shıft steps are taken wıth a u,v close
' to the cluster
      IF relstp < Eta THEN relstp = Eta
      relstp = SQR(relstp)
      u = u - u * relstp
      v = v + v * relstp
      CALL Quadsd(nn, u, v, p(), qp(), a, b)
      FOR i = 1 TO 5
       CALL Calcsc(itype)
       CALL Nextk(itype)
      NEXT i
      Ltried = 1: ' set to TRUE
      j = 0
     END IF
    END IF
    omp = dmp
' calculate next dk polynomıal and new u and v
     CALL Calcsc(itype)
     CALL Nextk(itype)
     CALL Calcsc(itype)
     CALL Newest(itype, ui, vi)
' ıf vı ıs zero the ıteratıon ıs not convergıng
     IF vi = 0 THEN EXIT SUB
     relstp = ABS((vi - v) / vi)
     u = ui
     v = vi
     GOTO 101

END SUB ' Quadit
```

```
SUB Realit (sss, nz, iflag)
'---------------------------------------------------------
' varıable-shıft h polynomıal ıteratıon for a real zero.
' sss   - startıng ıterate
' nz    - number of zero found
' ıflag - flag to ındıcate a paır of zeros near real axıs.
'---------------------------------------------------------
     nz = 0
     s = sss
     iflag = 0
     j = 0
' MAIN LOOP
106   pv = p(1)
' evaluate p at s
     qp(1) = pv
     FOR i = 2 TO nn
      pv = pv * s + p(i)
      qp(i) = pv
     NEXT i
     dmp = ABS(pv)
' compute a rıgorous bound on the error ın evaluatıng p
     dms = ABS(s)
     ee = (dmre / (are + dmre)) * ABS(qp(1))
     FOR i = 2 TO nn
      ee = ee * dms + ABS(qp(i))
     NEXT i
' ıteratıon has converged sufficıently ıf the
' polynomıal value ıs less than 20 tımes thıs bound
     IF dmp <= 20 * ((are + dmre) * ee - dmre * dmp) THEN
      nz = 1
      szr = s
      szi = 0
      EXIT SUB
     END IF
     j = j + 1
' stop ıteratıon after 10 steps
     IF j > 10 THEN EXIT SUB
     IF j >= 2 THEN
      IF ABS(t) <= .001 * ABS(s - t) AND dmp > omp THEN
' a cluster of zeros near the real axıs has been encountered
' return wıth ıflag set to ınıtıate a quadratıc ıteratıon
       iflag = 1
       sss = s
       EXIT SUB
      END IF
     END IF
' return ıf the polynomıal value has ıncreased sıgnıfıcantly
     omp = dmp
' compute t, the next polynomıal, and the new ıterate
     dkv = dk(1)
     qk(1) = dkv
     FOR i = 2 TO n1
      dkv = dkv * s + dk(i)
      qk(i) = dkv
     NEXT i
     IF ABS(dkv) > ABS(dk(n1)) * 10 * Eta THEN
' use the scaled form of the recurrence ıf the value
' of dk at s ıs nonzero
      t = -pv / dkv
      dk(1) = qp(1)
      FOR i = 2 TO n1
       dk(i) = t * qk(i - 1) + qp(i)
      NEXT i
     ELSE
' USE UNSCALED FORM
      dk(1) = 0
      FOR i = 2 TO n1
       dk(i) = qk(i - 1)
      NEXT i
     END IF
     dkv = dk(1)
     FOR i = 2 TO n1
      dkv = dkv * s + dk(i)
     NEXT i
     t = 0
     IF ABS(dkv) > ABS(dk(n1)) * 10 * Eta THEN t = -pv / dkv
     s = s + t
     GOTO 106

END SUB ' Realit
```

```
SUB Calcsc (itype)
'-----------------------------------------------------------
' thıs routıne calculates scalar quantıtıes used to
' compute the next dk polynomıal and new estımates of
' the quadratıc coefficıents.
' itype - ınteger varıable set here ındıcatıng how the
' calculatıons are normalızed to avoıd overflow
'-----------------------------------------------------------
' synthetic divisin of dk by the quadratıc 1,u,v
     CALL Quadsd(n1, u, v, dk(), qk(), c, d)
     IF ABS(c) <= ABS(dk(n1)) * 100 * Eta THEN
      IF ABS(d) <= ABS(dk(n1 - 1)) * 100 * Eta THEN
       itype = 3
' itype=3 indicates the quadratıc is almost a factor of dk
       EXIT SUB
      END IF
     END IF
     IF ABS(d) >= ABS(c) THEN
      itype = 2
' itype=2 ındıcates that all formulas are dıvıded by d
      e = a / d
      f = c / d
      g = u * b
      h = v * b
      a3 = (a + g) * e + h * (b / d)
      a1 = b * f - a
      a7 = (f + u) * a + h
      EXIT SUB
     END IF
     itype = 1
' itype=1 ındıcates that all formulas are dıvıded by c
     e = a / c
     f = d / c
     g = u * e
     h = v * b
     a3 = a * e + (h / c + g) * b
     a1 = b - a * (d / c)
     a7 = a + g * d + h * f

END SUB ' Calcsc
```

```
SUB Nextk (itype)
'------------------------------------------------------------
' computes the next dk polynomıals usıng scalars
' computed ın calcsc
'------------------------------------------------------------
     IF itype <> 3 THEN
      temp = a
      IF itype = 1 THEN temp = b
      IF ABS(a1) <= ABS(temp) * Eta * 10 THEN
' ıf a1 ıs nearly zero then use a specıal form of the recurrence
       dk(1) = 0
       dk(2) = -a7 * qp(1)
       FOR i = 3 TO n1
        dk(i) = a3 * qk(i - 2) - a7 * qp(i - 1)
       NEXT i
       EXIT SUB
      END IF
' use scaled form of the recurrence
      a7 = a7 / a1
      a3 = a3 / a1
      dk(1) = qp(1)
      dk(2) = qp(2) - a7 * qp(1)
      FOR i = 3 TO n1
       dk(i) = a3 * qk(i - 2) - a7 * qp(i - 1) + qp(i)
      NEXT i
      EXIT SUB
     END IF
' use unscaled form of the recurrence ıf itype ıs 3
     dk(1) = 0
     dk(2) = 0
     FOR i = 3 TO n1
      dk(i) = qk(i - 2)
     NEXT i

END SUB ' Nextk
```

```
SUB Newest (itype, uu, vv)
'-----------------------------------------------------------
' compute new estımates of the quadratıc coefficıents
' usıng the scalars computed ın calcsc.
'-----------------------------------------------------------
' USE FORMULAS APPROPRIATE TO SETTING OF iType.
  IF itype <> 3 THEN
   IF itype <> 2 THEN
    a4 = a + u * b + h * f
    a5 = c + (u + v * f) * d
   ELSE
    a4 = (a + g) * f + h
    a5 = (f + u) * c + v * d
   END IF
' evaluate new quadratıc coefficıents.
   b1 = -dk(n1) / p(nn)
   b2 = -(dk(n1 - 1) + b1 * p(n1)) / p(nn)
   c1 = v * b2 * a1
   c2 = b1 * a7
   c3 = b1 * b1 * a3
   c4 = c1 - c2 - c3
   temp = a5 + b1 * a4 - c4
   IF temp <> 0 THEN
    uu = u - (u * (c3 + c2) + v * (b1 * a1 + b2 * a7)) / temp
    vv = v * (1 + c4 / temp)
    EXIT SUB
   END IF
  END IF
' ıf itype=3 the quadratıc ıs zeroed
  uu = 0
  vv = 0

END SUB ' Newest
```

```
SUB Quadsd (nn, u, v, p(), q(), a, b)
'-----------------------------------------------
' dıvıdes p by the quadratıc  1,u,v  placıng the
' quotıent ın q and the remaınder ın a,b
'-----------------------------------------------
     b = p(1)
     q(1) = b
     a = p(2) - u * b
     q(2) = a
     FOR i = 3 TO nn
      cc = p(i) - u * a - v * b
      q(i) = cc
      b = a
      a = cc
     NEXT i

END SUB 'Quadsd
```

```
SUB Quad (a, b1, c, sr, si, dLr, dLi)
'-----------------------------------------------------------
' calculate the zeros of the quadratıc a*z**2+b1*z+c=0.
' the quadratıc formula, modıfied to avoıd
' overflow, ıs used to fınd the larger zero ıf the
' zeros are real and both zeros are complex.
' the smaller real zero ıs found dırectly from the
' product of the zeros c/a.
'-----------------------------------------------------------
     IF a <> 0 THEN GOTO 200
     sr = 0
     IF b1 <> 0 THEN sr = -c / b1
     dLr = 0
100   si = 0
     dLi = 0
     EXIT SUB

200   IF c = 0 THEN
      sr = 0
      dLr = -b1 / a
      GOTO 100
     END IF
' compute dıscrımınant avoıdıng overflow
     bb = b1 / 2
     IF ABS(bb) >= ABS(c) THEN
      ee = 1 - (a / bb) * (c / bb)
      dd = SQR(ABS(ee)) * ABS(bb)
     ELSE
      ee = a
      IF c < 0 THEN ee = -a
      ee = bb * (bb / ABS(c)) - ee
      dd = SQR(ABS(ee)) * SQR(ABS(c))
     END IF
     IF ee >= 0 THEN
' REAL ZEROS
      IF bb >= 0 THEN dd = -dd
      dLr = (-bb + dd) / a
      sr = 0
      IF dLr <> 0 THEN sr = (c / dLr) / a
      GOTO 100
     END IF
' complex conjugate zeros
     sr = -bb / a
     dLr = sr
     si = ABS(dd / a)
     dLi = -si

END SUB 'Quad
```

Karşılaştırma

Polinom köklerini bulan **JenkinsTraub, Bairstow, NewtonRaphson** ve **Muller** programları aşağıda verilen özel polinomlar için karşılaştırılacaktır:

1. $f(x)=(x-1)^7=x^7-7x^6+21x^5-35x^4+35x^3-21x^2+7x-1$
2. $f(x)=(x-10)^4(x-9.999)=x^5-49.99899999999999x^4+999.96x^3-9999.399999999999x^2+49996x-99990$
3. $f(x)=(x+0.001)(x-0.001)(x+1)(x-1)(x+1000)(x-1000)=x^6-1000001.000001x^4+1000001.000001x^2-1$
4. $f(x)=(x-1+2i)^2(x-1-2i)^2=x^4-4x^3+14x^2-20x+25$
5. $f(x)=x^{20}-1$

Bu polinomların her birinin farklı özellikleri vardır:

1.polinomun tüm kökleri çakışmaktadır: $x_{1,2,3,4,5,6,7}=1$

2.polinomun çakışan ve birbirine çok yakın kökleri vardır: $x_{1,2,3,4}=10,\ x_5=9.999$

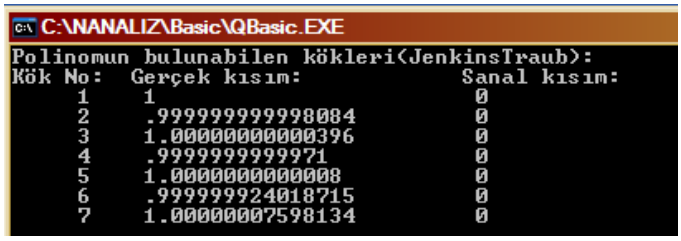
3.polinomun kökleri mutlak değerce çakışmakta, çok küçük ve çok büyük kökler içermektedir: $x_{1,2}=\mp 0.001,\ x_{3,4}=\mp 1,\ x_{5,6}=\mp 1000$

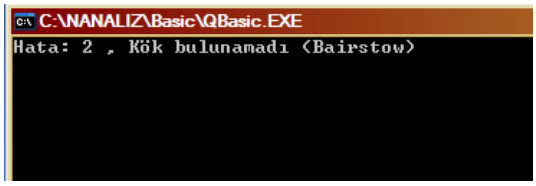
4.polinomun tüm kökleri sanaldır ve çakışmaktadır: $x_{1,2,3,4}=1\mp 2i$

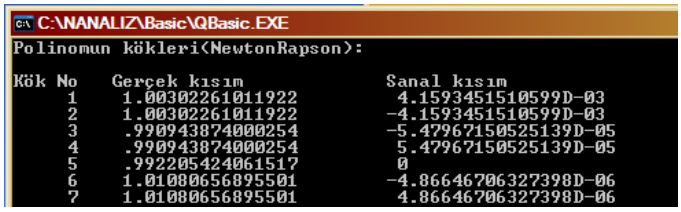
5.polinomun derecesi çok yüksektir, sadece iki gerçek kökü vardır: $x_{1,2}=1$, diğerleri sanaldır.

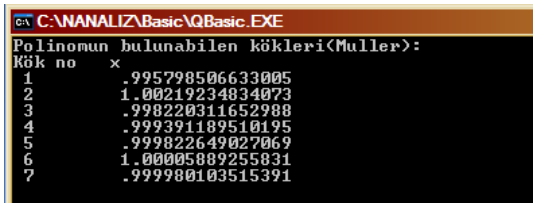
Karşılaştırmanın amacı, bu çok özel durumlarda programların başarılı olup olmadığını ortaya koymaktır.

1. $f(x)=(x-1)^7=x^7-7x^6+21x^5-35x^4+35x^3-21x^2+7x-1$ için programların sonuçları:



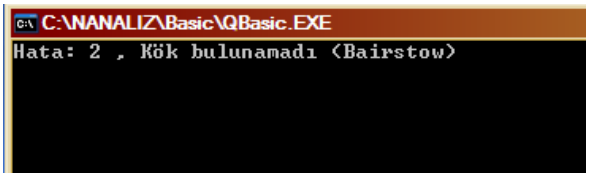


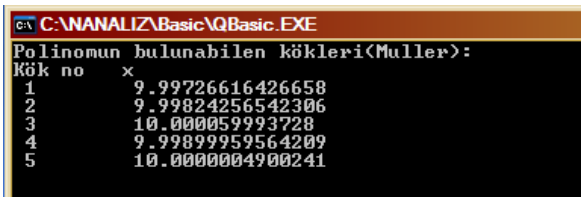




Yorum: En başarılı: JenkinsTraub, en başarısız: Bairstow

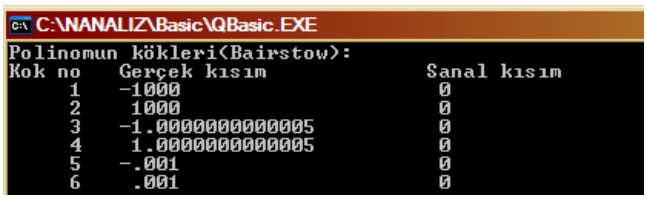
2. $f(x)=(x-10)^4(x-9.999)=x^5-49.99899999999999x^4+999.96x^3-9999.399999999999x^2+49996x-99990$ için programların sonuçları:

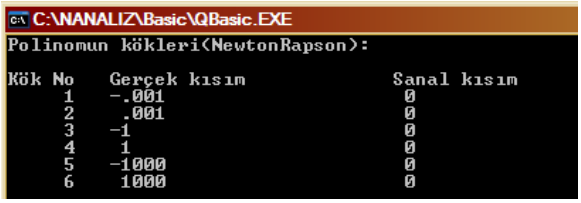


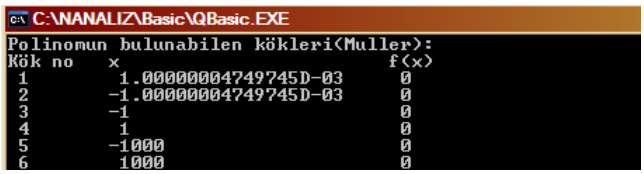


Yorum: En başarılı: JenkinsTraub, en başarısız: Bairstow ve NewtonRaphson

3. $f(x)=(x+0.001)(x-0.001)(x+1)(x-1)(x+1000)(x-1000)=x^6-1000001.000001x^4+1000001.000001x^2-1$ için programların sonuçları:

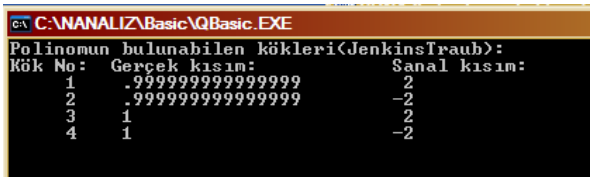


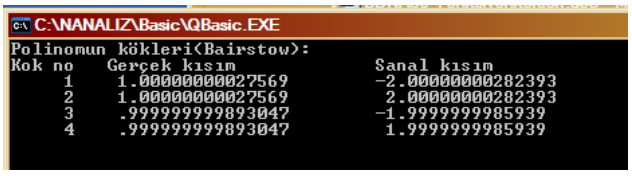




Yorum: En başarılı: hepsi de başarılı, NewtonRaphson ve JenkinsTraub daha doğru.

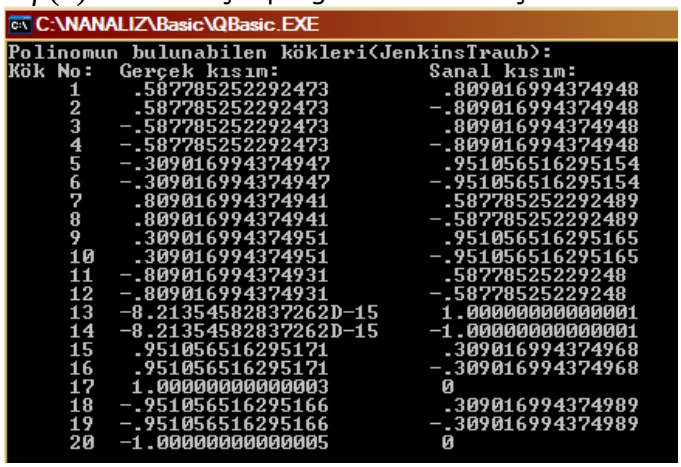
4. $f(x)=(x-1+2i)^2(x-1-2i)^2=x^4-4x^3+14x^2-20x+25$ için programların sonuçları:

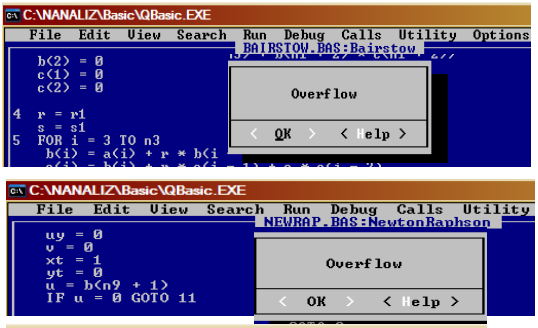




Yorum: En başarılı: JenkinsTraub ve Bairstow, en başarısız: NewtonRaphson. Muller zaten sanal kök bulmaz.

5. $f(x)=x^{20}-1$ için programların sonuçları:





Yorum: En başarılı: JenkinsTraub. Bairstow ve NewtonRaphson başarısız(sayı taşması sonucu program kırılıyor). Sadece gerçek kök bulan Muller de başarılı fakat polinomun sadece iki gerçek kökü varken, üç kök buluyor.

Sonuç olarak, **JenkinsTraub** en başarılı programdır. Konuya yönelik kaynaklarda da bu görüş hakimdir, en hassas ve en hızlı metod olarak değerlendirmektedir. Ancak, program çok fazla ve karmaşık kod içermektedir. Program kaynaklarda RPOLY olarak anılmaktadır.

Jenkins-Traub metodunun ayrıca sanal katsayılı polinomlar için CPOLY adlı bir başka bir programı daha vardır. Her iki programın FORTRAN ve C kodları internette bulunabilir.

RPOLY FORTRAN kodu:
http://www.netlib.org/toms/493
http://people.sc.fsu.edu/~jburkardt/f77_src/toms493/toms493.f

CPOLY FORTRAN kodu:
http://www.netlib.org/toms/419
http://people.sc.fsu.edu/~jburkardt/f77_src/toms419/toms419.f

RPOLY C++ kodu: http://www.akiti.ca/rpoly_ak1_cpp.html

CPOLY C kodu: http://svn.r-project.org/R/trunk/src/appl/cpoly.c