1. **NUMERIK HATALAR ve BILGISAYAR ARITMETIGI**

**ODEV 1 - 18.02.2020:**

ex fonksiyonunda e0.5 =1.648721271’dir. Bu sayiya 4 basamakli bir sistemde kesme ve yuvarlama yapan C kodunu yaziniz.

#include <stdio.h>

#include <math.h>

double kesme(double normalize, int basamaksistemi)

{

    /\*  --ALGORİTMA--

    1.  normalize edilmiş float sayı(ör. 0.123456) ve basamak sistemi paramatre olarak alınır.

    2.  basamak sistemi kaç ise o kadar 10 ile çarpılarak nokta sağa kaydırılır.(4 için 1234.56)

    3.  elde edilen sayı tam kısım ve floatlı kısımdan oluşur.Float kısmı kesmeden dolayı yok

        sayılacak kısımdır.

    4.  integer ın çalışma prensibinden faydalanırak atama yapılır.Ve float kısım kaybedilir.

    5.  Kaçkez 10 ile çarpıldıysa sayı bu sefer, o kadar 10 a bölünür ve kesmeye maruz kalan

        sayı elde edilmiş olur.

    \*/

    int tmp = normalize \* pow(10, basamaksistemi);

    normalize = tmp / (pow(10, basamaksistemi) \* 1.0);

    return normalize;

}

double yuvarla(double normalize, int basamaksistemi)

{

    /\*  --ALGORİTMA--

    1.  normalize edilmiş float sayı(ör. 0.123456) ve basamak sistemi paramatre olarak alınır.

    2.  yuvarlamada basamak sistemi kaç ise bir sonraki basamağın değeri önemlidir.O yüzden bir

        değişkende tutulur.

    3.  basamak sistemi kaç ise 1 fazlası kadar 10 ile çarpılarak nokta sağa kaydırılır.(4 için 12345.6)

    4.  integer ın çalışma prensibinden faydalanırak atama yapılır.Ve float kısım kaybedilir.

    5.  sayının 10 a bölümünden kalan değişekene alınır.

    6.  10 bölünerek sayı istenilen basamak sistemine uygun hale gelir.

    7.  yuvarlamaya karar verilmek için değişkene alınan değer eğer 5'ten büyük ve eşitse değeri bir artılır.

        değilse bir işlem yapılmaz.

    8.  Kaçkez 10 ile çarpıldıysa sayı bu sefer, o kadar 10 a bölünür ve yuvarlamaya maruz kalan

        sayı elde edilmiş olur.

    \*/

    int yuvarlamabasamagi;

    int tmp = normalize \* pow(10, basamaksistemi + 1);

    yuvarlamabasamagi = tmp % 10;

    tmp /= 10;

    if (yuvarlamabasamagi >= 5)

    {

        tmp++;

    }

    normalize = tmp / (pow(10, basamaksistemi) \* 1.0);

    return normalize;

}

int main(int argc, char const \*argv[])

{

    int kuvvet = 0;

    double sayi = exp(0.5);

    double normalize;

    int basamaksistemi;

    printf("basamak sistemini giriniz. \n");

    scanf("%d", &basamaksistemi);

    normalize = sayi;

    while (normalize > 1)

    {

        kuvvet++;

        normalize /= 10;

    }

    printf("sayının  normalize  hali: %lfx10^%d\n", normalize, kuvvet);

    sayi = kesme(normalize, basamaksistemi);

    printf("sayının   kesilmiş  hali: %gx10^%d\n", sayi, kuvvet);

    sayi = yuvarla(normalize, basamaksistemi);

    printf("sayının yuvarlanmış hali: %gx10^%d\n", sayi, kuvvet);

    return 0;

}

**CALISMA1 – 18.02.2020:**

Verilen 10’luk tabandaki sayiyi 2’lik tabana ceviren C kodunu yaziniz.

#include <stdio.h>

#include <math.h>

/\*

--ALGORİTMA--

1. start

2. sayıyı al.i değişekni sıfır yap index olarak kullan

3. sayı sıfırdan farklı ise adım 4 e git.Değilse adım 8'e git

4. dizinin i değerindeki index'e sayının ikiye bölümünden kalanı ekle

5. sayıyı iikiye böl.(float kısmı alınmayacak)

6. i değerini bir arttır

7. adım 3' e git

8. for ile diziyi ters çevir.(dizideki binary sayı tersten okunur.ters çevirerek sayı düzeltilir)

9. sayıyı ekrana yazdır

10. stop

\*/

short binaryreverse[32];

short binary[32];

int main(int argc, char const \*argv[])

{

    int decimal = 5;

    int i = 0;

    //döngüde ikilik sayı ters olarak bulunur

    while (decimal != 0)

    {

        binaryreverse[i] = decimal % 2;

        decimal /= 2;

        i++;

    }

    // ters sayımızı düzeltiğimiz kod bloğu

    for (int i = 0; i < 32; i++)

    {

        binary[i] = binaryreverse[31 - i];

    }

    for (int i = 0; i < 32; i++)

    {

        printf("%d", binary[i]);

    }

    return 0;

}

1. **GERCEK SAYILARIN BELLEKTE TUTULMASI**

**ODEV 2 - 25.02.2020:**

64 bitlik sayiyi 10’luk sayiyi cevirme.

#include <stdio.h>

#include <math.h>

char sayi[64];

int main()

{

    //64 bit sayi

    int s;        //işaret biti

    int c = 0;    //üstel bit

    double f = 0; //kesir kısım

    int sayac1 = 0;

    int sayac2 = 0;

    int s1 = 11;

    int s2 = 12;

    printf("çevrilecek 64 bit sayiyi giriniz ");

    scanf("%s", &sayi);

    if (sayi[0] == '0')

        s = 0;

    else if (sayi[0 == '1'])

        s = 1;

    while (sayac1 < 11)

    {

        if (sayi[s1] == '1')

            c += pow(2, sayac1);

        s1--;

        sayac1++;

    }

    while (sayac2 < 52)

    {

        if (sayi[s2] == '1')

            f += pow(2, -(sayac2 + 1)); //değer taraf üsü 1 den başlatan sayac2

        s2++;

        sayac2++;

    }

    double sonuc = pow(-1, s) \* pow(2, (c - 1023)) \* (1 + f);

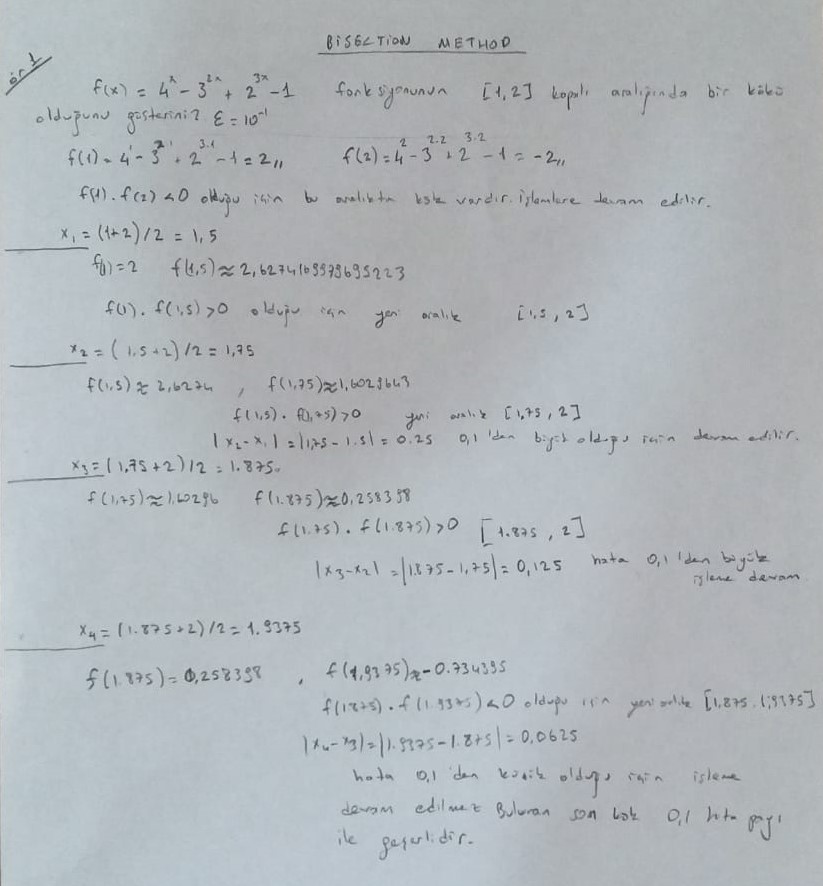
    printf("ondalık sayi=> %f", sonuc);

    return 0;

}

1. **BISECTION METOD**

**Örnek 1**



#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  Verilen iki kökü fonksiyona koy ve çıkan değerleri çarp

4.  Sonucun sıfırdan küçük olup olmadığını kontrol et küçükse bu aralıkta kök vardır. Adım 5' e

    geç. Küçük değilse aralıkta kök yoktur adım 16'ya git.

5.  En son bulunan iki kök arasındaki farkı bul.Bulunan değer hatapayıdır.son kökü değikene

    al ezilip kaybolmaması için

6.  Hata payı toleranstan büyükse döngüye gir(Adım 7 den devam).Değilse Adım 15 e git.

7.  Yeni kökü iki kökün aritmetik ortalamasını alarak bul.

8.  Lower kök ile yeni bulunan kökün fonksiyonlardaki değerlerini çarp

9.  Sonuç sıfırdan küçükse upper kökü yeni bulunan kök olarak kabul et.

10. Büyükse  lower kökü yeni bulunan kök olarak kabul et.

11. Sıfıra eşitse kök zaten bulunmuş demektir.adım 15'e git.

12. Son iki kökün farkını alarak hata payını bul.

13. Son bulunan kökü değişkende tut.

14. Adım 6 ya git.

15. Bulunan kökü yazdır.

16. Stop

 \*/

double f(double x)

{

    return pow(4, x) - pow(3, 2 \* x) + pow(2, 3 \* x) - 1;

}

int main(int argc, char const \*argv[])

{

    int iterasyon = 0;

    double xlower = 1.0, xupper = 2.0, xr = 0.0, hata = 0.0, tolerans = 0.1;

    if ((f(xlower) \* f(xupper)) < 0)

    {

        hata = fabs(xupper - xlower);

        double sonkok = xupper;

        while (hata > tolerans)

        {

            iterasyon++;

            xr = (xlower + xupper) / 2;

            if (f(xlower) \* f(xr) < 0)

            {

                xupper = xr;

            }

            else if (f(xlower) \* f(xr) > 0)

            {

                xlower = xr;

            }

            else

            {

                sonkok = xr;

                break;

            }

            hata = fabs(sonkok - xr);

            sonkok = xr;

        }

        printf("Kök %g dir.%d iterasyonda bulundu.", sonkok, iterasyon);

    }

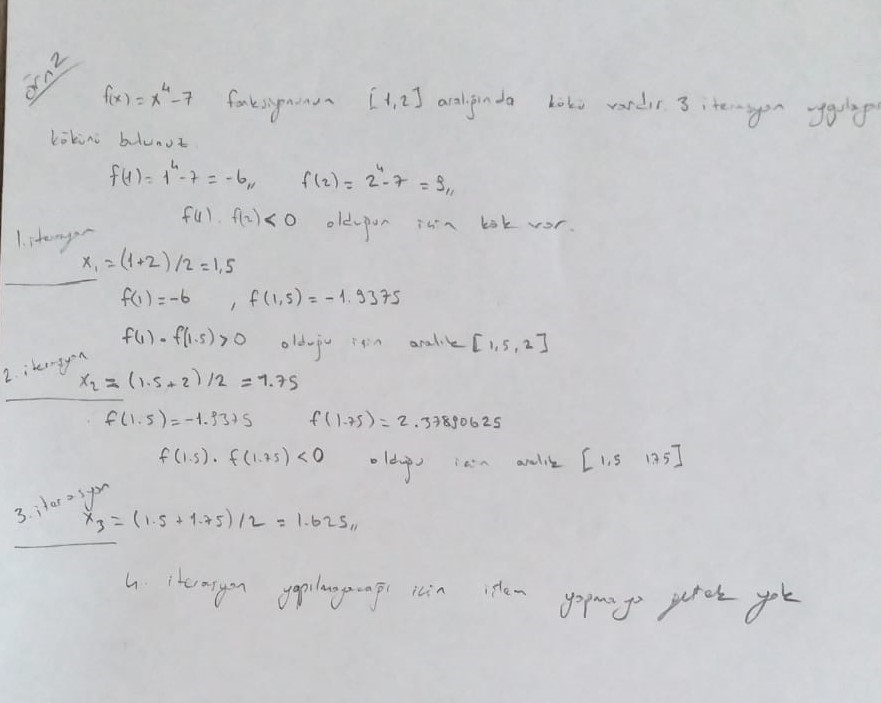
    else

        printf("[%g %g] aralığında kök yoktur", xlower, xupper);

    return 0;

}

Örnek 2



#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  Kullanıcıdan iterasyon sayısını girmek istediğini mi yoksa iterasyon önerisi mi almak istediğni sor

    ( Switch Case 1'i seçerse adım 4'e, 2'yi seçerse adım 5'e git)

4.  İterasyon sayısını al. Adım 7'ye git.

5.  İstenilen hata payını kullanıcıdan al.minimum iterasyon sayısını hesapla.

6.  Ekrana minimum iterasyon sayısını yazdır adım 4 egit.

7.  Verilen iki kökü fonksiyona koy ve çıkan değerleri çarp

8.  Sonucun sıfırdan küçük olup olmadığını kontrol et küçükse bu aralıkta kök vardır. Adım 9'a

    geç. Küçük değilse aralıkta kök yoktur adım 21'e git.

9.  En son bulunan iki kök arasındaki farkı bul.Bulunan değer hatapayıdır.son kökü değikene

    al ezilip kaybolmaması için

10. İterasyon değeri sıfırdan büyükse döngüye gir.(Adım 11 den devam).Değilse Adım 20 e git.

11. İterasyon değerini 1 azalt.

12. Yeni kökü iki kökün aritmetik ortalamasını alarak bul.

13. Lower kök ile yeni bulunan kökün fonksiyonlardaki değerlerini çarp

14. Sonuç sıfırdan küçükse upper kökü yeni bulunan kök olarak kabul et.

15. Büyükse  lower kökü yeni bulunan kök olarak kabul et.

16. Sıfıra eşitse kök zaten bulunmuş demektir.adım 20'e git.

17. Son iki kökün farkını alarak hata payını bul.

18. Son bulunan kökü değişkende tut.

19. Adım 10 na git.

20. Bulunan kökü yazdır.

21. Stop

 \*/

double f(double x);

int iterasyononeri(double hatapayi, double aralik1, double aralik2);

int main(int argc, char const \*argv[])

{

    int secim;

    int iterasyon;

    double xlower = 1.0, xupper = 2.0, xr = 0.0, hata = 0.0, tolerans = 0.1;

    printf("\n1.) Kaç iterasyon işlem yapmak istiyorsunuz?\n2.)Hata payı girerek minimum kaç iterasyon yapılmalı öğren\n");

    scanf("%d", &secim);

    switch (secim)

    {

    case 1:

    {

    iterasyongir:

        printf("\nİterasyon sayısı:");

        scanf("%d", &iterasyon);

        goto islem;

    }

    case 2:

    {

        printf("Hata payı: ");

        scanf("%lf", &tolerans);

        printf("\n%lf hata payı için en az %d iterasyon işlem yapılmalıdır", tolerans, iterasyononeri(tolerans, xlower, xupper));

        goto iterasyongir;

        printf("");

    }

    default:

    {

        printf("\nHatalı seçim yatınız.Promram sonlandırıldı.");

        return 0;

    }

    }

islem:

    if ((f(xlower) \* f(xupper)) < 0)

    {

        hata = fabs(xupper - xlower);

        double sonkok = xupper;

        while (iterasyon > 0)

        {

            iterasyon--;

            xr = (xlower + xupper) / 2;

            if (f(xlower) \* f(xr) < 0)

            {

                xupper = xr;

            }

            else if (f(xlower) \* f(xr) > 0)

            {

                xlower = xr;

            }

            else

            {

                sonkok = xr;

                break;

            }

            hata = fabs(sonkok - xr);

            sonkok = xr;

        }

        printf("Kök %g dir.", sonkok);

    }

    else

        printf("[%g %g] aralığında kök yoktur", xlower, xupper);

    return 0;

}

double f(double x)

{

    return pow(x, 4) - 7;

}

int iterasyononeri(double hatapayi, double aralik1, double aralik2)

{

    int n = (log10(fabs(aralik1 - aralik2)) - log10(2 \* hatapayi)) / log10(2);

    /\* +2 olma sebebi

    +1 formüldeki n bulunan değerden büyük olduğu için

    diğer +1 ise formülde hep 1 eksik çıkıyor hatalı yani yada

    formül bisection algoritmasındaki ilk bulduğuz kökü iterasyondan saymıyor.

    (algoritmada ilk kök fonksiyonların çarpımının 0'dan küçük olup olamadığına bakılmadan hesaplanıyor).

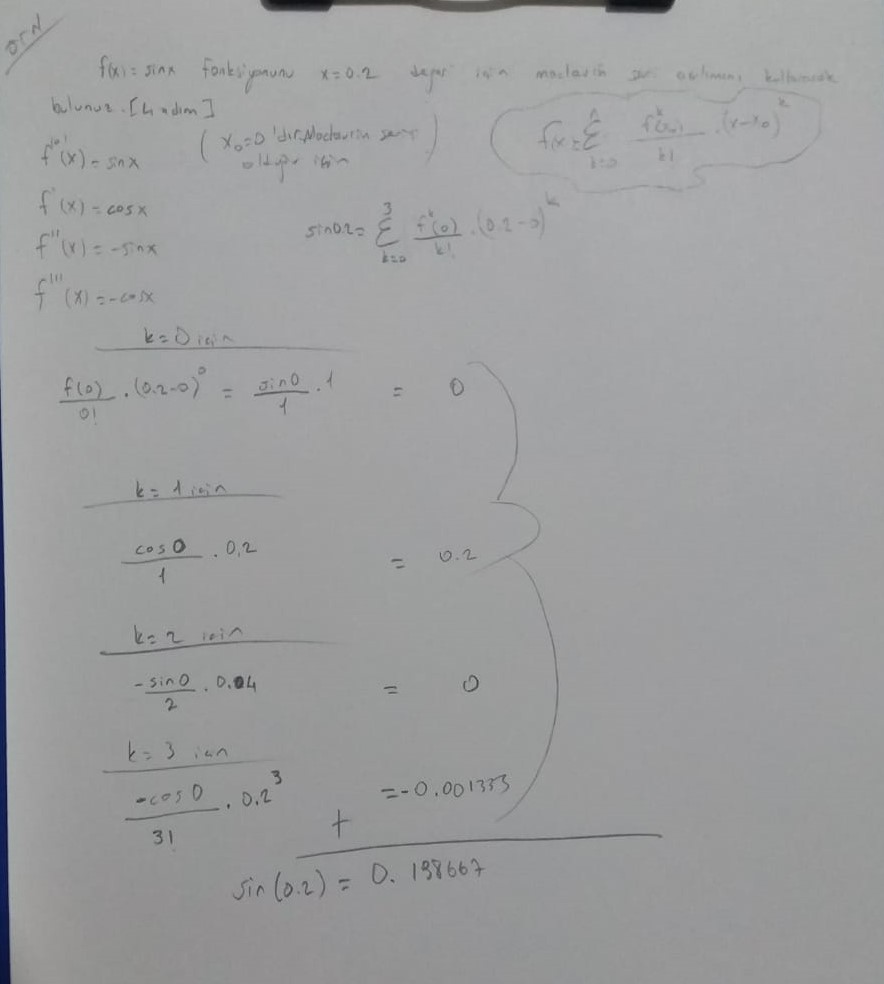
    \*/

    return n + 2;

}

1. **TAYLOR ve MACLAURIN SERILERI**

**Örnek 1**

****

#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  start

2.  değişkenlerin değerlerini ata (x, toplam, turev, x0, k)

3.  fonksiyon 4 katlarında tekrarlayan bir rutin içinde ilerlemektedir.

4.  k 4 den küçükmü kontrol edilir. true ise adım 5 e. değilse adım 8 e git

5.  k nın mod 4 ü alınır. modun sonucuna göre turev değeri alınır

6.  bulunan değerler formülde yazılarak serinin geçerli aşamadaki

    değeri bulunur toplama eklenir.

7.  bir sonraki iterasyon için k 1 artırılır.Adım 4 e git.

8.  serinin x değeri için değeri ekrana yazılır.

9.  stop

\*/

double fak(int a)

{

    if (a == 1 || a == 0)

        return 1;

    else

        return a \* fak(a - 1);

}

int main(int argc, char const \*argv[])

{

    double x = 0.2, toplam = 0.0, turev, x0 = 0;

    int k = 0;

    while (k < 4)

    {

        int mod = k % 4;

        if (mod == 0)

        {

            turev = sin(x0);

        }

        else if (mod == 1)

        {

            turev = cos(x0);

        }

        else if (mod == 2)

        {

            turev = -sin(x0);

        }

        else if (mod == 3)

        {

            turev = -cos(x0);

        }

        toplam += (pow(x - x0, k) \* turev) / fak(k);

        k++;

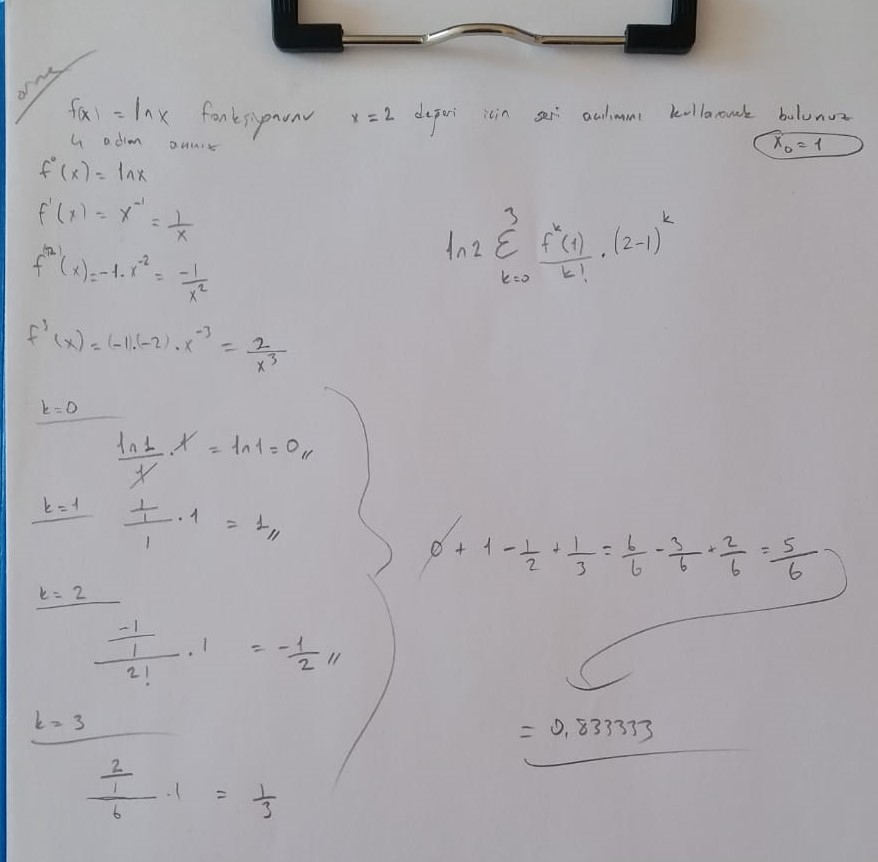
    }

    printf("toplam = %lf ", toplam);

    return 0;

}

Örnek 2



#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  start

2.  değişkenlerin değerlerini ata (x, toplam, turev, x0, k)

3.  lnx in değeri alınır .

4.  serinin ilk aşaması için formulde yerine yazılır toplama eklenir.

5.  bir sonraki iterasyon için k 1 artırılır.

6.  sabit diye bir değişken oluşturulup değeri 1 yapılır.

    (sabit değişkeni türev alırken kullanılmak için kullanılıyor.)

7.  k 4 den küçükmü kontrol edilir. true ise adım 8 e.

    false ise adım 12 ye gidilir.

8.  türev alınır

9.  türev formülde yerine konur.sonuç toplama eklenir.

10. k 1 artırılır.

11. adım 4 e gidilir.

12. serinin x değeri için değeri ekrana yazılır.

13. stop

\*/

double fak(int a)

{

    if (a == 1 || a == 0)

        return 1;

    else

        return a \* fak(a - 1);

}

int main(int argc, char const \*argv[])

{

    double x = 2, toplam = 0.0, turev, x0 = 1;

    int k = 0;

    turev = log(x0);

    toplam = (pow(x - x0, k) \* turev) / fak(k);

    k = 1;

    turev = 1 / x0;

    toplam += (pow(x - x0, k) \* turev) / fak(k);

    k++;

    int sabit = 1;

    while (k < 4)

    {

        sabit = sabit \* -(k - 1);

        turev = sabit / pow(x0, k);

        toplam += (pow(x - x0, k) \* turev) / fak(k);

        k++;

    }

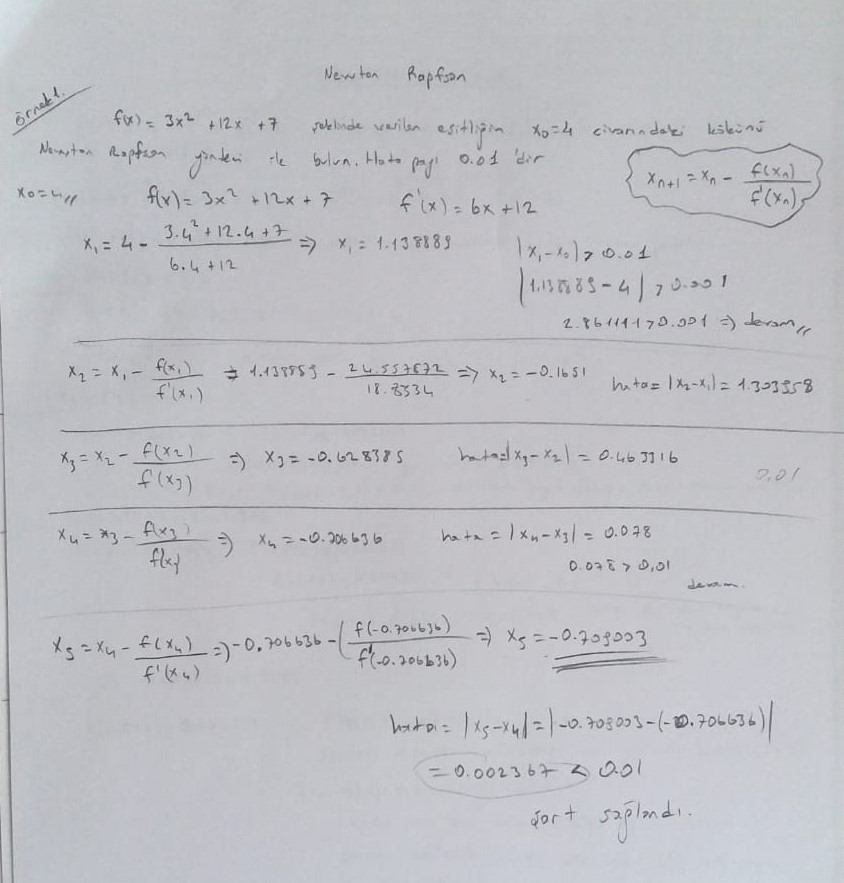
    printf("toplam = %lf ", toplam);

    return 0;

}

1. **NEWTON – RAPHSON METODU**

**Örnek 1**



#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  yeni kökü formülü kullanarak bul.(x = x0 - f(x0) / f1(x0);)

4.  son iki kökün farkını bularak hata payını belirle.

5.  bi önceki kökü son bulduğun kökün değerine eşle

6.  hata payı büyükmü toleranstan kontrol et.büyükse adım 3'e git.değilse adım 7.

7.  bulunan kökü yazdır

8.  stop

 \*/

double f(double x)

{

    return 3 \* pow(x, 2) + 12 \* x + 7;

}

double f1(double x)

{

    return 6 \* x + 12;

}

int main()

{

    int iterasyon = 0;

    double x0 = 4, x = 0.0, y = 0.0, hata = 0.0, tolerans;

    printf("Kaç hata payı ile bulmak istiyorsunuz? ");

    scanf("%lf", &tolerans);

    do

    {

        iterasyon++;

        x = x0 - f(x0) / f1(x0);

        hata = fabs(x - x0);

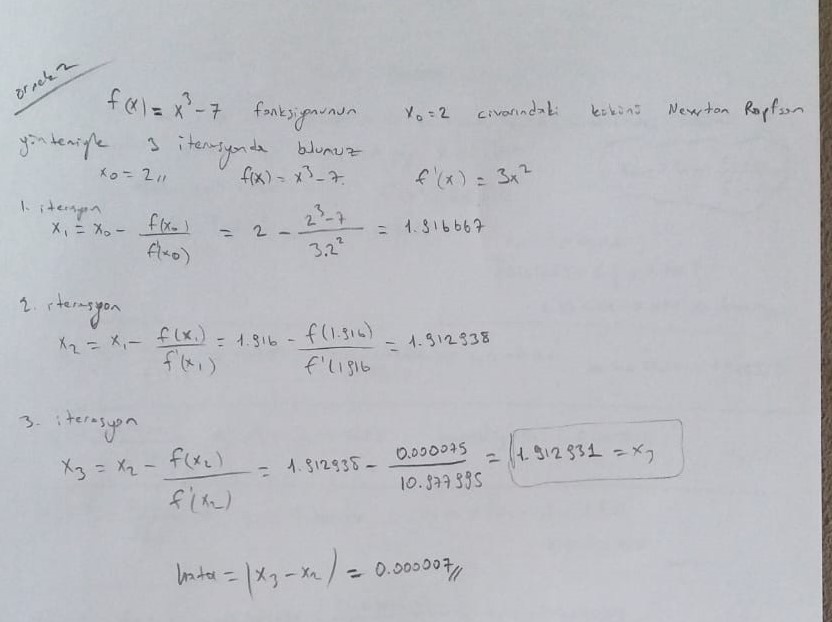
        x0 = x;

    } while (hata > tolerans);

    printf("Kok -> %lf", x);

    return 0;

}

ÖRNEK2

#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  yeni kökü formülü kullanarak bul.(x = x0 - f(x0) / f1(x0);)

4.  son iki kökün farkını bularak hata payını belirle.

5.  bi önceki kökü son bulduğun kökün değerine eşle

6.  hata payı büyükmü toleranstan kontrol et.büyükse adım 3'e git.değilse adım 7.

7.  bulunan kökü yazdır

8.  stop

 \*/

double f(double x)

{

    return pow(x, 3) - 7;

}

double f1(double x)

{

    return 3 \* pow(x, 2);

}

int main()

{

    int i = 0;

    int iterasyon;

    double x0 = 2, x = 0.0, y = 0.0, hata = 0.0;

    printf("iterasyon sayısını giriniz\n");

    scanf("%d", &iterasyon);

    do

    {

        i++;

        x = x0 - f(x0) / f1(x0);

        hata = fabs(x - x0);

        x0 = x;

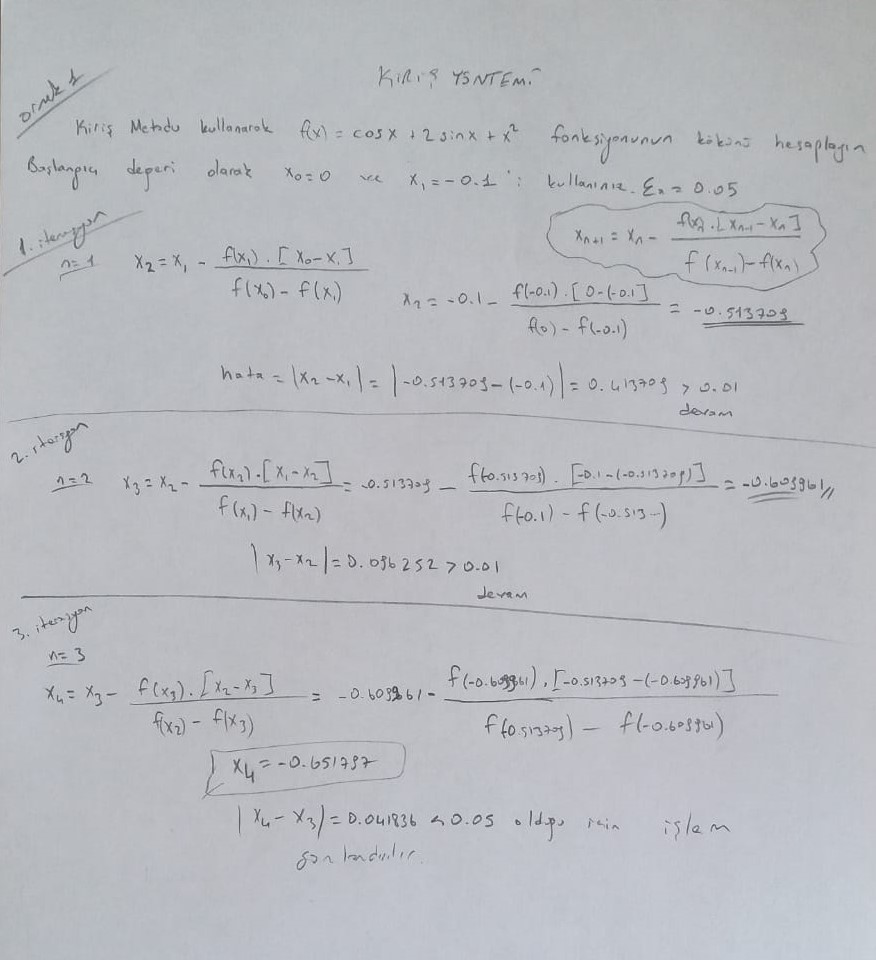
    } while (i < iterasyon);

    printf("Kok -> %lf\nHata -> %lf", x, hata);

    return 0;

}

1. **KIRIS YONTEMI/SECANT METHOD (6. HAFTA)**

**Örnek 1**

#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  Yeni kökü x0 ve x1 değerlerini formülde yerine

    yazarak bul. (x2 = x1 - ((f(x1) \* (x0 - x1)) / (f(x0) - f(x1)));)

4.  son iki kökün farkını bularak hata payını belirle.

5.  x0'a x1 kökünün değerini koy

6.  x1'e yeni kökün değerini koy

7.  Hata payı toleranstan büyükse döngüye gir(Adım 3'e git).Değilse Adım 8 e git.

8.  En son bulunan kökü yazdır.

8.  stop

 \*/

double f(double x)

{

    return cos(x) + 2 \* sin(x) + x \* x;

}

void Secant(void)

{

    int iterasyon = 0;

    double x0 = 0, x1 = -0.1, x2 = 0.0, hata = 0.0, tolerans = 0.05;

    do

    {

        iterasyon++;

        x2 = x1 - ((f(x1) \* (x0 - x1)) / (f(x0) - f(x1)));

        hata = fabs(x2 - x1);

        x0 = x1;

        x1 = x2;

    } while (hata > tolerans);

    printf("aranan kök -> %1f hata ->%1f  iterasyon: %d", x2, hata, iterasyon);

}

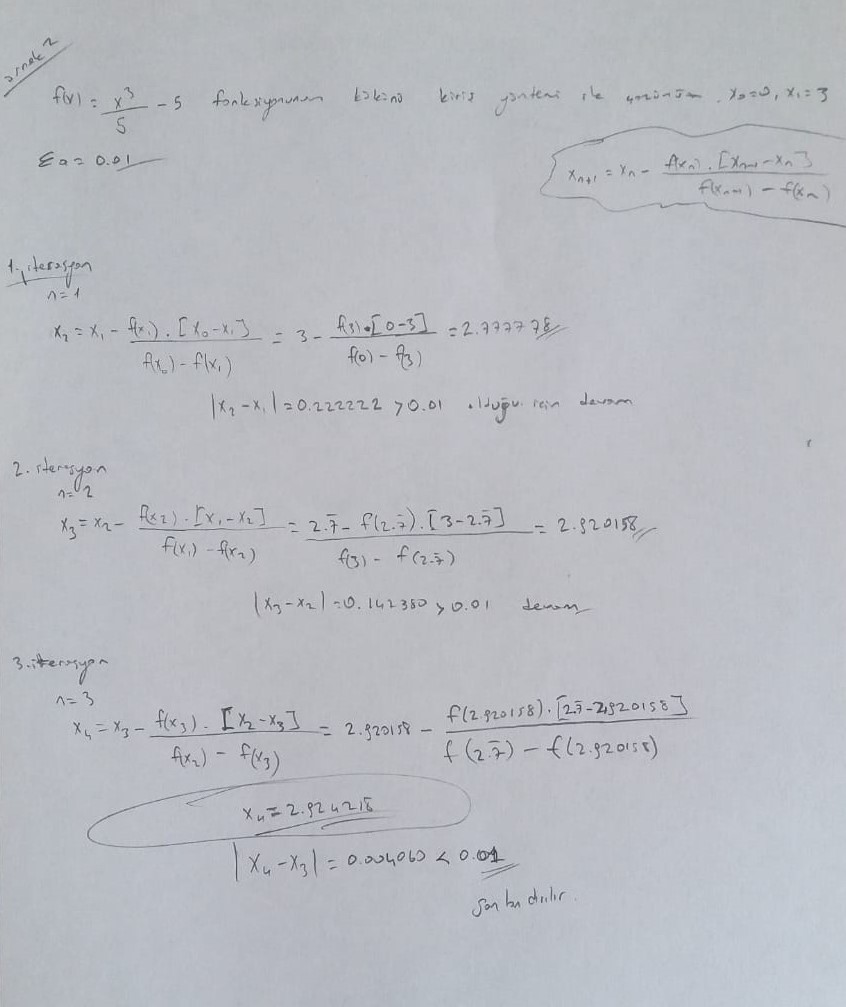
int main()

{

    Secant();

    return 0;

}

**Örnek 2**

#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  Yeni kökü x0 ve x1 değerlerini formülde yerine

    yazarak bul. (x2 = x1 - ((f(x1) \* (x0 - x1)) / (f(x0) - f(x1)));)

4.  son iki kökün farkını bularak hata payını belirle.

5.  x0'a x1 kökünün değerini koy

6.  x1'e yeni kökün değerini koy

7.  Hata payı toleranstan büyükse döngüye gir(Adım 3'e git).Değilse Adım 8 e git.

8.  En son bulunan kökü yazdır.

8.  stop

 \*/

double f(double x)

{

    return pow(x,3)/5-5;

}

void Secant(void)

{

    int iterasyon = 0;

    double x0 = 0, x1 = 3, x2 = 0.0, hata = 0.0, tolerans = 0.01;

    do

    {

        iterasyon++;

        x2 = x1 - ((f(x1) \* (x0 - x1)) / (f(x0) - f(x1)));

        hata = fabs(x2 - x1);

        x0 = x1;

        x1 = x2;

    } while (hata > tolerans);

    printf("aranan kök -> %1f hata ->%1f  iterasyon: %d", x2, hata, iterasyon);

}

int main()

{

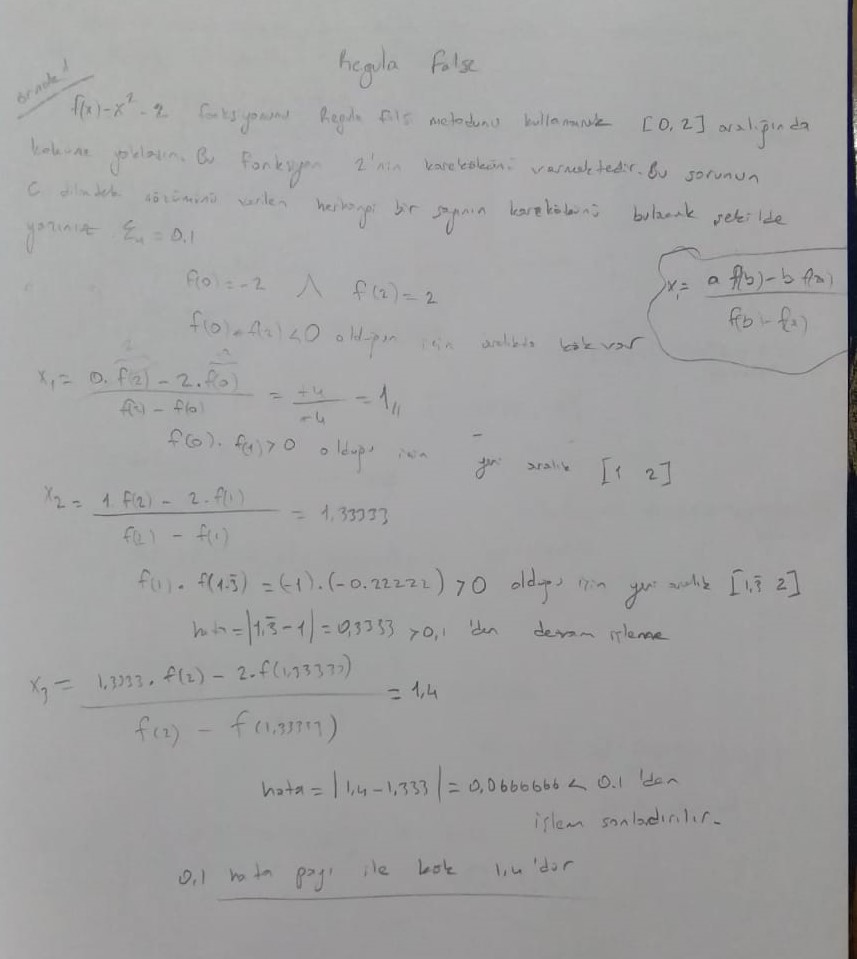
    Secant();

    return 0;

}

1. **DOGRUSAL ENTERPOLASYON (REGULA FALSI METHOD)**

**Örnek 1**

****

// karekök bulan fonsiyonu dogrusal ent kullanarak yazınız

#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  Verilen iki kökü fonksiyona koy ve çıkan değerleri çarp

4.  Sonucun sıfırdan küçük olup olmadığını kontrol et küçükse bu aralıkta kök vardır. Adım 5' e

    geç. Küçük değilse aralıkta kök yoktur adım 16'ya git.

5.  En son bulunan iki kök arasındaki farkı bul.Bulunan değer hatapayıdır.son kökü değikene

    al ezilip kaybolmaması için

6.  Hata payı toleranstan büyükse döngüye gir(Adım 7 den devam).Değilse Adım 15 e git.

7.  Yeni kökü (xlower \* f(xupper) - xupper \* f(xlower)) / (f(xupper) - f(xlower) formülünü

    kullanarak bul.

8.  Lower kök ile yeni bulunan kökün fonksiyonlardaki değerlerini çarp

9.  Sonuç sıfırdan küçükse upper kökü yeni bulunan kök olarak kabul et.

10. Büyükse  lower kökü yeni bulunan kök olarak kabul et.

11. Sıfıra eşitse kök zaten bulunmuş demektir.adım 15'e git.

12. Son iki kökün farkını alarak hata payını bul.

13. Son bulunan kökü değişkende tut.

14. Adım 6 ya git.

15. Bulunan kökü yazdır.

16. Stop

 \*/

// Algoritma Aynıdır tek farklılık kullanılan fonsiyondaki sabit değer

// kullanıcıdan alınarak dinamiklik kazandırılmıştır.

double f(double x, double a)

{

    return pow(x, 2) - a;

}

double karekokbul(double a)

{

    double xlower = 0.0, xupper = a, xr = 0.0, hata = 0.0, tolerans = 0.1;

    if (a >= 0)

    {

        hata = fabs(xupper - xlower);

        double sonkok = xupper;

        while (hata > tolerans)

        {

            xr = (xlower \* f(xupper, a) - xupper \* f(xlower, a)) / (f(xupper, a) - f(xlower, a));

            if (f(xlower, a) \* f(xr, a) < 0)

            {

                xupper = xr;

            }

            else if (f(xlower, a) \* f(xr, a) > 0)

            {

                xlower = xr;

            }

            else

            {

                sonkok = xr;

                break;

            }

            hata = fabs(sonkok - xr);

            sonkok = xr;

        }

    }

    else

        printf("Verilen sayının real bir karakökü yoktur\n");

    return xr;

}

int main(int argc, char const \*argv[])

{

    double sayi;

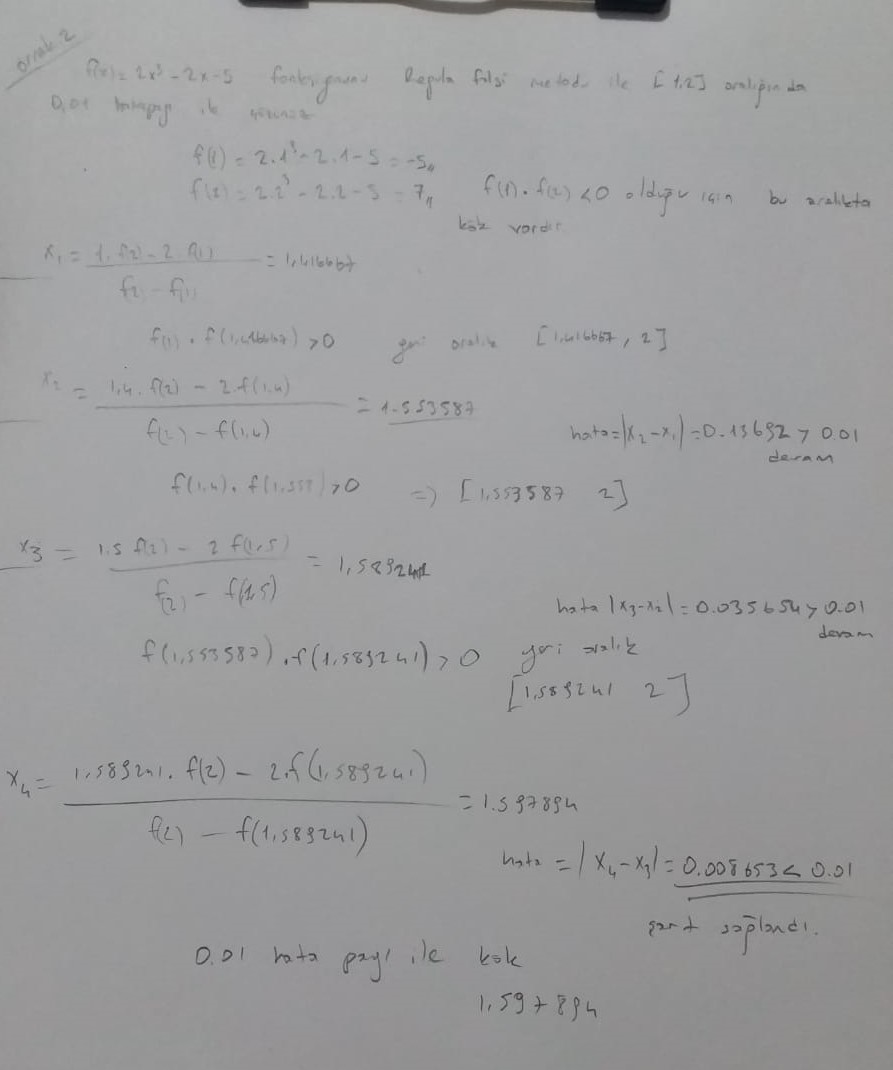
    printf("hangi sayının karekökünü bulmak istiyorsunuz : ");

    scanf("%lf", &sayi);

    printf("\n%lf\n", karekokbul(sayi));

    return 0;

}

Örnek2

#include <stdio.h>

#include <math.h>

/\*  --ALGORİTMA--

1.  Start

2.  Verileri al

3.  Verilen iki kökü fonksiyona koy ve çıkan değerleri çarp

4.  Sonucun sıfırdan küçük olup olmadığını kontrol et küçükse bu aralıkta kök vardır. Adım 5' e

    geç. Küçük değilse aralıkta kök yoktur adım 16'ya git.

5.  En son bulunan iki kök arasındaki farkı bul.Bulunan değer hatapayıdır.son kökü değikene

    al ezilip kaybolmaması için

6.  Hata payı toleranstan büyükse döngüye gir(Adım 7 den devam).Değilse Adım 15 e git.

7.  Yeni kökü (xlower \* f(xupper) - xupper \* f(xlower)) / (f(xupper) - f(xlower) formülünü

    kullanarak bul.

8.  Lower kök ile yeni bulunan kökün fonksiyonlardaki değerlerini çarp

9.  Sonuç sıfırdan küçükse upper kökü yeni bulunan kök olarak kabul et.

10. Büyükse  lower kökü yeni bulunan kök olarak kabul et.

11. Sıfıra eşitse kök zaten bulunmuş demektir.adım 15'e git.

12. Son iki kökün farkını alarak hata payını bul.

13. Son bulunan kökü değişkende tut.

14. Adım 6 ya git.

15. Bulunan kökü yazdır.

16. Stop

 \*/

double f(double x)

{

    return 2\* pow(x, 3) - 2 \* x -5;

}

int main(int argc, char const \*argv[])

{

    int iterasyon = 0;

    double xlower = 1.0, xupper = 2.0, xr = 0.0, hata = 0.0, tolerans = 0.01;

    if ((f(xlower) \* f(xupper)) < 0)

    {

        hata = fabs(xupper - xlower);

        double sonkok = xupper;

        while (hata > tolerans)

        {

            iterasyon++;

            xr = (xlower \* f(xupper) - xupper \* f(xlower)) / (f(xupper) - f(xlower));

            if (f(xlower) \* f(xr) < 0)

            {

                xupper = xr;

            }

            else if (f(xlower) \* f(xr) > 0)

            {

                xlower = xr;

            }

            else

            {

                sonkok = xr;

                break;

            }

            hata = fabs(sonkok - xr);

            sonkok = xr;

        }

        printf("Kök %g dir.%d iterasyonda bulundu.", sonkok, iterasyon);

    }

    else

        printf("[%g %g] aralığında kök yoktur", xlower, xupper);

    return 0;

}

BILGISAYAR ARITMETIGI

Sayısal bilgisayarlarda hesaplama problemlerinin sonuçlandırılması için verileri işleyen aritmetik buyruklar vardır. Bu buyruklar aritmetik hesaplamaları yaparlar ve bilgisayar içindeki tüm veri işlemlerinden sorumludurlar. 4 temel işlem toplama, çıkarma, çarpma ve bölmedir. Bu dört temel işlemden aritmetik fonksiyonları elde etmek ve sayısal hesaplamalarla bilimsel problemleri çözmek mümkündür.

Aritmetik işlemci, işlem biriminin bir parçası olup aritmetik işlemleri icra eder. Aritmetik işlemlerin icrası sırasında işlemci yazaçlarında bulunan veri tipleri buyruğun içinde belirtilmiştir. Bir aritmetik buyruk ikili veya ondalık veriyi belirler. Her iki durumda da veri sabit noktalı veya kayan noktalı olabilir. Sabit noktalı sayılar tamsayıları veya kesirli sayıları gösterebilirler. Negatif sayılar işaretli veya tümleyen gösterimiyle verilebilir.

*Kaynak: ftp://bilgisayar@smyoedu.cumhuriyet.edu.tr/MatematikProgramlama/bilgisayararitmatigi.pdf*