



# Yıldız Teknik Üniversitesi

## Elektrik-Elektronik Fakültesi

### Bilgisayar Mühendisliği Bölümü

Gr: 2

İsim: Yusuf Yılmaz

No:

E-posta:

## 1.1) Denklem Formatı

Kullanıcının diferansiyel denklemi  $Ay' + By + g(x) = 0$  formatında girmesi beklenir. Katsayıları ondalıklı sayı olarak girebilir. Fonksiyonların üssünü tam sayı olarak girmelidir. İd bilgisi olarak 1-2-3 değerlerinden farklı bir değer giremez, girdiği takdirde uyarı alır. Bu denklem aşağıdaki **struct** yapısında tutulur:

**a1:** y' ifadesinin katsayısı (A)

**a2:** y ifadesinin katsayısı (B)

**expectedResult:** bulunması gereken gerçek sonuç

**myResult:** Runge Kutta 4 yöntemi kullanıldıktan sonra elde edilen sonuç

**struct F:** g(x) denklemini tutan yapı

**n:** g(x) denkleminin terim sayısı

**array :** n \* 4 boyutundaki g(x) fonksiyonunun katsayı ve desteklenen fonksiyonların id bilgisini tutan dizi

array dizisinin her bir gözünde aşağıdaki bilgiler tutulur:

- **0.indeks :** desteklenen fonksiyonun id bilgisi (id bilgilerine dair açıklama ileride yapılacaktır.)
- **1.indeks :** girilen terimin katsayısı
- **2.indeks :** girilen terimin derecesi
- **3.indeks :** sin(bx) ve cos(bx) fonksiyonları için parantez içindeki b katsayısı

```
typedef struct{
    int n;
    int **array;
}F;

typedef struct{
    double a1;
    double a2;
    double expectedResult; // certain real result of differential equations
    double myResult; // calculated result
    F function;
}ODE;
```

## 1.2) Denklem Girişinin Yapılması

Diferansiyel denklem **getEquations()** fonksiyonu ile g(x) fonksiyonu da **getFunction()** yardımcı fonksiyonu ile alınır.

### void getEquations()

**parametreler:** diferansiyel denklemi tutan ODE yapısı

**n:** kullanıcıdan  $g(x)$  fonksiyonunun terim sayısını alır.

Alınan terim sayısına göre getFunction() yardımcı fonksiyonu çağırılır.

**a1:**  $y'$  ifadesinin katsayısı

**a2:**  $y$  ifadesinin katsayısı

```
void getEquations(ODE *equations){
    // bu fonksiyon ile kullanıciden denklemi alın
    printf("Enter the number of elements of the function f(x):");
    scanf("%d",&(equations->function.n));

    getFunction(equations);
    printf("\n-----\n");
    printf("Enter the coefficient of [A] A* y' : ");
    scanf("%lf",&equations->a1);
    printf("Enter the coefficient of [B] B* y : ");
    scanf("%lf",&equations->a2);
}
```

### void getFunction()

**parametreler:** diferansiyel denklemi tutan ODE yapısı

$n$  ifadesine bağlı olarak array dizisi için  $n*4$  boyutunda yer ayrılır.

$g(x)$  fonksiyonunun terimlerinin bilgileri alınır:

Girilen terimin id bilgisi alınır, desteklenen fonksiyon türleri için aşağıdaki formattadır:

- **ID 1:** polinom fonksiyon,  $a \cdot x^n$  formatındadır
- **ID 2:**  $\sin(x)$ ,  $asin^n(Bx)$  formatındadır
- **ID 3:**  $\cos(x)$ ,  $acos^n(Bx)$  formatındadır

girilen id bilgisine bağlı olarak ilgili kod parçacığı çalışır ve katsayı ile üs bilgilerini alır.

Geçersiz id bilgisi girilmesi halinde uyarı verir ve tekrar id bilgisi alır. Geçerli bir değer girilene kadar tekrar eder.

```

void getFunction(ODE *equations){
    printf("\nThe following types can be found in the function:\n");
    printf("\t- ID:1 polynomial\n\t- ID:2 sin(Ax)\n\t- ID:3 cos(Ax)\n");
    int i;
    int n = equations->function.n;
    // memory allocation for the array
    equations->function.array = (int**) malloc(n*sizeof(int*));
    for(i=0; i< n ;i++){
        equations->function.array[i] = (int*) malloc(4*sizeof(int));
    }

    switch (equations->function.array[i][0])
    {
    case 1:
        /* a*(x^n) */
        printf("Enter the coefficient [a] of a*(x^n) : ");
        scanf("%d",&equations->function.array[i][1]);
        printf("Enter exponent [n] of a*(x^n) : ");
        scanf("%d",&equations->function.array[i][2]);
        break;
    case 2:
        /* a*sin(Bx) */
        printf("Enter the coefficient [a] of a*[ sin(Bx) ]^n : ");
        scanf("%d",&equations->function.array[i][1]);
        printf("Enter sub coefficient [B] of a*[ sin(Bx) ]^n : ");
        scanf("%d",&equations->function.array[i][3]);
        printf("Enter the exponent [n] of a*[ sin(Bx) ]^n : ");
        scanf("%d",&equations->function.array[i][2]);
        break;
    case 3:
        /* a*cos(Bx) */
        printf("Enter the coefficient [a] of a*[ cos(Bx) ]^n : ");
        scanf("%d",&equations->function.array[i][1]);
        printf("Enter sub coefficient [B] of a*[ cos(Bx) ]^n : ");
        scanf("%d",&equations->function.array[i][3]);
        printf("Enter the exponent [n] of a*[ cos(Bx) ]^n : ");
        scanf("%d",&equations->function.array[i][2]);
        break;
    default:
        printf("Enter a valid function ID !\n");
        i--;
        break;
    }
}
}

```

### 1.3) Girilen Diferansiyel Denklemin Ekrana Yazdırılması

**printFunction()** fonksiyonu ile alınan denklem ekrana  $Ay' + By + g(x) = 0$  formatında yazdırılır.

```
void printFunction(ODE equations){
    int i;
    printf("\n-----\n%lf*y' + %lf*y + ",equations.a1,equations.a2);
    printf("[ ");
    for (i = 0; i < equations.function.n; i++)
    {
        switch (equations.function.array[i][0])
        {
            case 1:
                printf("%d*x^%d + ",equations.function.array[i][1],equations.function.array[i][2]);
                break;
            case 2:
                printf("%d*[ sin(%dx) ]^%d + ",equations.function.array[i][1],equations.function.array[i][3],equations.function.array[i][2]);
                break;
            case 3:
                printf("%d*[ cos(%dx) ]^%d + ",equations.function.array[i][1],equations.function.array[i][3],equations.function.array[i][2]);
                break;
        }
    }
    printf("] = 0\n\n");
}
```

### 1.4) Hesaplama İşlemi Gerçekleştiren Fonksiyonlar

#### double calculateFx():

**parametreler:** diferansiyel denklemi tutan ODE yapısı, hesaplanması istenilen x değeri

$g(x)$  fonksiyonunun tutulduğu **array** üzerinde dolaşarak id bilgisine göre ilgili kod parçacığını çalıştırır ve **result** değişkeninin değerini artırır. Hesaplamalar tamamlandıca **result** değişkenini döndürür.

```
double calculateFx(ODE equations, double xi){
    double result =0,x ,sin_x,cos_x;
    int i,pow;

    for (i = 0; i < equations.function.n; i++)
    {
        x = xi;
        switch (equations.function.array[i][0])
        {
            case 1:
                // polynomial a*(x^n)
                if(equations.function.array[i][2] >=1){
                    for(pow=1; pow < equations.function.array[i][2] ;pow++){
                        // x has the value of the 1st power, so if the exponent is greater than 1, we will multiply x by itself
                        x *=x;
                    }
                    result += equations.function.array[i][1] * x;
                }
                else{
                    result += equations.function.array[i][1];
                }
                break;
            case 2:
                // sinx a*sin(Bx)
                sin_x = sin(equations.function.array[i][3] * x);

                for(pow=1; pow < equations.function.array[i][2] ;pow++){
                    // if exponent greater than 1, we will multiply x by itself otherwise do nothing because we have already calculate sin() once
                    sin_x *= sin_x;
                }
                result += equations.function.array[i][1] * sin_x;
                break;
            case 3:
                // cosx a*cos(Bx)
                cos_x = cos(equations.function.array[i][3] * x);
                for(pow=1; pow < equations.function.array[i][2] ;pow++){
                    // if exponent greater than 1, we will multiply x by itself otherwise do nothing because we have already calculate cos() once
                    cos_x *= cos_x;
                }
                result += equations.function.array[i][1] * cos_x;
                break;
        }
    }
    return result;
}
```

### double rungeKuttaMethod():

**parametreler:** diferansiyel denklemin tutan ODE yapısı, başlangıç noktası, başlangıç değeri, adım büyüklüğünü tutan işaretçi (bu işaretçi main fonksiyonunda yapılan işlemler için kullanıcadan alınan adım büyüklüğünü )

**h:** adım büyüklüğü

**targetValue:** hedef noktası

adım büyüklüğü ve iterasyon sayısını kullanıcıdan aldıktan sonra aşağıdaki formülleri kullanarak hesaplama yapar:

$$y(x_i + h) = y(x_i) + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$k_1 = f(x_i, y_i)$$

$$k_2 = f\left(x_i + \frac{1}{2} \cdot h, y_i + \frac{1}{2} \cdot k_1 \cdot h\right)$$

$$k_3 = f\left(x_i + \frac{1}{2} \cdot h, y_i + \frac{1}{2} \cdot k_2 \cdot h\right)$$

$$k_4 = f(x_i + h, y_i + k_3 \cdot h)$$

$$f(x_i, y_i) = y' \quad y' = -(g(x_i) + By(x_i)) / A$$

### 2.1) Main Fonksiyonu İçerisindeki İşlemler

- ODE yapısındaki denklem değişkeni tanımlanır.
- Başlangıç noktası, başlangıç değeri ve adım büyüklüğünü tutmak için değişkenler tanımlanır.
- **data:** adım büyüklüğüne bağlı değişimleri görmek için eklenmiş dizidir. Test sayısı x 3 boyutunda yer açılır. Runge Kutta 4 yöntemi ile hesaplama yapıldıktan sonra adım büyüklüğü, elde edilen sonuç ve mutlak hata değerlerini tutar.
- **getEquations()** fonksiyonu çağırılır.
- **printFunction()** fonksiyonu çağırılır.
- Başlangıç noktası, başlangıç değeri ve beklenen sonuç bilgileri alınır.
- **rungeKuttaMethod()** fonksiyonu çağırılır ve hesaplamalar yapılır.
- Hesaplama tamamlandıktan sonra ekrana bulunan değer, beklenen değer ve mutlak hata bilgilerini yazdırır.
- data dizisine bulunan bilgileri ekler ve data dizisini genişletir.
- Kullanıcı farklı bir adım büyüklüğü kullanmak isterse tekrar **RungeKuttaMethod()** fonksiyonunu çağırır ve adım büyüklüğü ile hedef noktası bilgilerini girer. Kullanıcı işlemi sonlandırmayı seçene kadar devam eder ve elde edilen bilgiler data dizisinde tutulur.
- İşlem sonlandırıldıktan sonra data dizisindeki bilgiler ekrana tablo halinde yazdırılır. Burada kullanıcının aynı diferansiyel denklem üzerinde farklı parametreleri deneyebilme adımları hızlandırılmış ve sonuçlardaki adım büyüklüğüne bağlı olarak değişen mutlak hata değerleri arasındaki farkın incelenmesini kolaylaştırmak amaçlanmıştır.



- Yer açılan data dizisi temizlenir.
- ODE yapısında yer açılan array dizisi temizlenir.

### 3.1) Programın Çalıştırılması ve Örnekler

#### ÖRNEK 1:

**Diferansiyel denklem :**  $2y' + y - x = 0$

**Başlangıç noktası:** 0, **Başlangıç değeri:** 1, **Adım büyüklüğü:** 0.1, **Hedef noktası:** 0.2

**Diferansiyel denklemin çözümü:**  $y(x) = x - 2 + 3 \cdot e^{-\frac{x}{2}}$

**Gerçek Sonuç:** 0.91451225 , **Hesaplanan Sonuç:** 0.91451227 , **Mutlak Hata:** 0.0000000188

```
Enter the number of elements of the function f(x):1

The following types can be found in the function:
  - ID:1 polynomial
  - ID:2 sin(Ax)
  - ID:3 cos(Ax)

Enter the function_id for element #1
  - ID:1 polynomial
  - ID:2 sin(Ax)
  - ID:3 cos(Ax)
1
Enter the coefficient [a] of a*(x^n) : -1
Enter exponent [n] of a*(x^n) : 1

-----
Enter the coefficient of [A] A* y' : 2
Enter the coefficient of [B] B* y : 1

-----
2.000000*y' + 1.000000*y + [ -1*x^1 + ] = 0

Enter the x0 value : 0
Enter the y(0.000000) value : 1
Enter the expected result : 0.91451225
Enter the step size (h): 0.1
Enter the target value (x) : 0.2

-----
k1 = -0.500000  k2 = -0.462500  k3 = -0.463438  k4 = -0.426828
iter #1 y(0.100000) = 0.953688

k1 = -0.426844  k2 = -0.391173  k3 = -0.392065  k4 = -0.357241
iter #2 y(0.200000) = 0.914512

-----
Runge Kutta-4 result: 0.914512
Expected result      : 0.914512
Error value          : 0.000000
```

```
Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 1
Enter the step size (h): 0.05
Enter the target value (x) : 0.2
```

```
-----
k1 = -0.500000  k2 = -0.481250  k3 = -0.481484  k4 = -0.462963
iter #1 y(0.050000) = 0.975930
```

```
k1 = -0.462965  k2 = -0.444678  k3 = -0.444906  k4 = -0.426842
iter #2 y(0.100000) = 0.953688
```

```
k1 = -0.426844  k2 = -0.409009  k3 = -0.409232  k4 = -0.391613
iter #3 y(0.150000) = 0.933230
```

```
k1 = -0.391615  k2 = -0.374220  k3 = -0.374437  k4 = -0.357254
iter #4 y(0.200000) = 0.914512
```

```
-----
Runge Kutta-4 result: 0.914512
Expected result      : 0.914512
Error value          : 0.000000
```



```
-----
Runge Kutta-4 result: 0.914512
Expepected result      : 0.914512
Error value            : 0.000000
```

```
Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 1
Enter the step size (h): 0.02
Enter the target value (x) : 0.2
```

```
-----
k1 = -0.500000  k2 = -0.492500  k3 = -0.492538  k4 = -0.485075
iter #1 y(0.020000) = 0.990150
```

```
k1 = -0.485075  k2 = -0.477649  k3 = -0.477687  k4 = -0.470298
iter #2 y(0.040000) = 0.980596
```

```
k1 = -0.470298  k2 = -0.462947  k3 = -0.462983  k4 = -0.455668
iter #3 y(0.060000) = 0.971337
```

```
k1 = -0.455668  k2 = -0.448390  k3 = -0.448426  k4 = -0.441184
iter #4 y(0.080000) = 0.962368
```

```
k1 = -0.441184  k2 = -0.433978  k3 = -0.434014  k4 = -0.426844
iter #5 y(0.100000) = 0.953688
```

```
k1 = -0.426844  k2 = -0.419710  k3 = -0.419746  k4 = -0.412647
iter #6 y(0.120000) = 0.945294
```

```
k1 = -0.412647  k2 = -0.405584  k3 = -0.405619  k4 = -0.398591
iter #7 y(0.140000) = 0.937181
```

```
k1 = -0.398591  k2 = -0.391598  k3 = -0.391633  k4 = -0.384674
iter #8 y(0.160000) = 0.929349
```

```
k1 = -0.384675  k2 = -0.377751  k3 = -0.377786  k4 = -0.370897
iter #9 y(0.180000) = 0.921794
```

```
k1 = -0.370897  k2 = -0.364042  k3 = -0.364077  k4 = -0.357256
iter #10          y(0.200000) = 0.914512
```

```
-----
Runge Kutta-4 result: 0.914512
Expepected result      : 0.914512
Error value            : 0.000000
```

```
Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 0
```

```
All test values:
```

Step Size (h)	RK-4 Result	Error Value
0.10000000	0.91451227	0.0000000188
0.05000000	0.91451226	0.0000000050
0.02000000	0.91451225	0.0000000041

```
-----
Process exited after 93.64 seconds with return value 1
Press any key to continue . . .
```

## ÖRNEK 2:

Diferansiyel denklem :  $y' + y = 0$

Başlangıç noktası: 0, Başlangıç değeri: 1, Adım büyüklüğü: 0.1, Hedef noktası: 0.5

Diferansiyel denklemin çözümü:  $y(x) = e^{-x}$

Gerçek Sonuç: 0.60653066 ,Hesaplanan Sonuç: 0.60653093 ,Mutlak Hata: 0.0000002744

```
-----
1.000000*y' + 1.000000*y + [ ] = 0

Enter the x0 value : 0
Enter the y(0.000000) value : 1
Enter the expected result : 0.60653066
Enter the step size (h): 0.1
Enter the target value (x) : 0.5

-----
k1 = -1.000000  k2 = -0.950000  k3 = -0.952500  k4 = -0.904750
iter #1 y(0.100000) = 0.904837

k1 = -0.904837  k2 = -0.859596  k3 = -0.861858  k4 = -0.818652
iter #2 y(0.200000) = 0.818731

k1 = -0.818731  k2 = -0.777794  k3 = -0.779841  k4 = -0.740747
iter #3 y(0.300000) = 0.740818

k1 = -0.740818  k2 = -0.703778  k3 = -0.705630  k4 = -0.670255
iter #4 y(0.400000) = 0.670320

k1 = -0.670320  k2 = -0.636804  k3 = -0.638480  k4 = -0.606472
iter #5 y(0.500000) = 0.606531

-----
Runge Kutta-4 result: 0.606531
Expected result      : 0.606531
Error value          : 0.000000

Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 0

All test values:
Step Size (h) | RK-4 Result | Error Value
-----
0.100000000  | 0.60653093  | 0.0000002744

-----
Process exited after 113.9 seconds with return value 1
Press any key to continue . . .
```

### ÖRNEK 3:

Diferansiyel denklem :  $y' + y - \sin^2(3x) - 5 \cos(x) = 0$

Başlangıç noktası: 0, Başlangıç değeri: 1, Adım büyüklüğü: 0.02, Hedef noktası: 0.6

Diferansiyel denklemin çözümü:  $y(x) = \frac{\left((-6 \sin(6x) + 185\sqrt{2} \sin\left(x + \frac{\pi}{4}\right) - \cos(6x) + 37\right)e^x - 147\right)e^{-x}}{74}$

Gerçek Sonuç: 2.9327367 ,Hesaplanan Sonuç: 2.93273671 ,Mutlak Hata: 0.0000000105

```
Enter the number of elements of the function f(x):2

The following types can be found in the function:
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)

Enter the function_id for element #1
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)
2
Enter the coefficient [a] of a*[ sin(Bx) ]^n : -1
Enter sub coefficient [B] of a*[ sin(Bx) ]^n : 3
Enter the exponent [n] of a*[ sin(Bx) ]^n : 2

Enter the function_id for element #2
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)
3
Enter the coefficient [a] of a*[ cos(Bx) ]^n : -5
Enter sub coefficient [B] of a*[ cos(Bx) ]^n : 1
Enter the exponent [n] of a*[ cos(Bx) ]^n : 1

-----
Enter the coefficient of [A] A* y' : 1
Enter the coefficient of [B] B* y : 1

-----
1.000000*y' + 1.000000*y + [ -1*[ sin(3x) ]^2 + -5*[ cos(1x) ]^1 + ] = 0

Enter the x0 value : 0
Enter the y(0.000000) value : 1
Enter the expected result : 2.9327367
Enter the step size (h): 0.02
Enter the target value (x) : 0.6
```



-----  
k1 = 4.000000    k2 = 3.960650    k3 = 3.961043    k4 = 3.923375  
iter #1 y(0.020000) = 1.079223

k1 = 3.923373    k2 = 3.887372    k3 = 3.887732    k4 = 3.853354  
iter #2 y(0.040000) = 1.156979

k1 = 3.853353    k2 = 3.820571    k3 = 3.820898    k4 = 3.789657  
iter #3 y(0.060000) = 1.233399

k1 = 3.789655    k2 = 3.759915    k3 = 3.760213    k4 = 3.731908  
iter #4 y(0.080000) = 1.308605

k1 = 3.731906    k2 = 3.704985    k3 = 3.705255    k4 = 3.679643  
iter #5 y(0.100000) = 1.382712

k1 = 3.679641    k2 = 3.655276    k3 = 3.655520    k4 = 3.632318  
iter #6 y(0.120000) = 1.455823

k1 = 3.632317    k2 = 3.610206    k3 = 3.610427    k4 = 3.589317  
iter #7 y(0.140000) = 1.528033

k1 = 3.589315    k2 = 3.569124    k3 = 3.569326    k4 = 3.549957  
iter #8 y(0.160000) = 1.599420

k1 = 3.549956    k2 = 3.531321    k3 = 3.531507    k4 = 3.513504  
iter #9 y(0.180000) = 1.670051

k1 = 3.513504    k2 = 3.496038    k3 = 3.496213    k4 = 3.479179  
iter #10        y(0.200000) = 1.739975

---

k1 = 3.479179    k2 = 3.462480    k3 = 3.462647    k4 = 3.446172  
iter #11        y(0.220000) = 1.809227

k1 = 3.446173    k2 = 3.429823    k3 = 3.429987    k4 = 3.413652  
iter #12        y(0.240000) = 1.877825

k1 = 3.413653    k2 = 3.397232    k3 = 3.397396    k4 = 3.380779  
iter #13        y(0.260000) = 1.945771

k1 = 3.380781    k2 = 3.363868    k3 = 3.364037    k4 = 3.346719  
iter #14        y(0.280000) = 2.013048

k1 = 3.346722    k2 = 3.328903    k3 = 3.329081    k4 = 3.310654  
iter #15        y(0.300000) = 2.079626

k1 = 3.310657    k2 = 3.291530    k3 = 3.291721    k4 = 3.271791  
iter #16        y(0.320000) = 2.145456

k1 = 3.271796    k2 = 3.250977    k3 = 3.251186    k4 = 3.229382  
iter #17        y(0.340000) = 2.210474

k1 = 3.229387    k2 = 3.206519    k3 = 3.206747    k4 = 3.182725  
iter #18        y(0.360000) = 2.274603

k1 = 3.182731    k2 = 3.157483    k3 = 3.157735    k4 = 3.131180  
iter #19        y(0.380000) = 2.337751

k1 = 3.131187    k2 = 3.103264    k3 = 3.103543    k4 = 3.074180  
iter #20        y(0.400000) = 2.399814

---

```

k1 = 3.131187    k2 = 3.103264    k3 = 3.103543    k4 = 3.074180
iter #20        y(0.400000) = 2.399814

k1 = 3.074188    k2 = 3.043333    k3 = 3.043642    k4 = 3.011234
iter #21        y(0.420000) = 2.460679

k1 = 3.011242    k2 = 2.977242    k3 = 2.977582    k4 = 2.941937
iter #22        y(0.440000) = 2.520221

k1 = 2.941946    k2 = 2.904631    k3 = 2.905004    k4 = 2.865977
iter #23        y(0.460000) = 2.578312

k1 = 2.865987    k2 = 2.825236    k3 = 2.825644    k4 = 2.783139
iter #24        y(0.480000) = 2.634815

k1 = 2.783149    k2 = 2.738892    k3 = 2.739335    k4 = 2.693307
iter #25        y(0.500000) = 2.689591

k1 = 2.693318    k2 = 2.645535    k3 = 2.646012    k4 = 2.596468
iter #26        y(0.520000) = 2.742501

k1 = 2.596478    k2 = 2.545201    k3 = 2.545714    k4 = 2.492709
iter #27        y(0.540000) = 2.793404

k1 = 2.492720    k2 = 2.438031    k3 = 2.438578    k4 = 2.382222
iter #28        y(0.560000) = 2.842165

k1 = 2.382233    k2 = 2.324265    k3 = 2.324844    k4 = 2.265294
iter #29        y(0.580000) = 2.888651

k1 = 2.265305    k2 = 2.204240    k3 = 2.204850    k4 = 2.142310
iter #30        y(0.600000) = 2.932737

```

```

-----
Runge Kutta-4 result: 2.932737
Expexted result      : 2.932737
Error value          : 0.000000

```

Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 0

All test values:

```

Step Size (h) | RK-4 Result | Error Value
-----
0.02000000    | 2.93273671 | 0.0000000105

```

```

-----
Process exited after 161.7 seconds with return value 1

```

#### ÖRNEK 4:

Diferansiyel denklem :  $y' + y - 2x^3 + x - \cos(5x) = 0$

Başlangıç noktası: 0, Başlangıç değeri: -2, Adım büyüklüğü: 0.1, Hedef noktası: 0.8

Diferansiyel denklemin çözümü:  $y(x) = \frac{((52x^3 - 156x^2 + 286x + 5 \sin(5x) + \cos(5x) - 286)e^x + 233)e^{-x}}{26}$

Gerçek Sonuç: -1.1600003, Hesaplanan Sonuç: -1.22053599, Mutlak Hata: 0.0605356929

```
Enter the number of elements of the function f(x):3

The following types can be found in the function:
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)

Enter the function_id for element #1
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)
1
Enter the coefficient [a] of a*(x^n) : -2
Enter exponent [n] of a*(x^n) : 3

Enter the function_id for element #2
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)
1
Enter the coefficient [a] of a*(x^n) : 1
Enter exponent [n] of a*(x^n) : 1

Enter the function_id for element #3
- ID:1 polynomial
- ID:2 sin(Ax)
- ID:3 cos(Ax)
3
Enter the coefficient [a] of a*[ cos(Bx) ]^n : -1
Enter sub coefficient [B] of a*[ cos(Bx) ]^n : 5
Enter the exponent [n] of a*[ cos(Bx) ]^n : 1

-----
Enter the coefficient of [A] A* y' : 1
Enter the coefficient of [B] B* y : 1

-----
1.000000*y' + 1.000000*y + [ -2*x^3 + 1*x^1 + -1*[ cos(5x) ]^1 + ] = 0

Enter the x0 value : 0
Enter the y(0.000000) value : -2
Enter the expected result : -1.1600003
Enter the step size (h): 0.1
Enter the target value (x) : 0.8
```



```

-----
k1 = 3.000000    k2 = 2.768925    k3 = 2.780479    k4 = 2.499735
iter #1 y(0.100000) = -1.723358

k1 = 2.501140    k2 = 2.181002    k3 = 2.197009    k4 = 1.847159
iter #2 y(0.200000) = -1.504952

k1 = 1.848455    k2 = 1.485664    k3 = 1.503804    k4 = 1.141509
iter #3 y(0.300000) = -1.355471

k1 = 1.142408    k2 = 0.800117    k3 = 0.817231    k4 = 0.508801
iter #4 y(0.400000) = -1.274039

k1 = 0.509092    k2 = 0.252423    k3 = 0.265257    k4 = 0.071370
iter #5 y(0.500000) = -1.247109

k1 = 0.070965    k2 = -0.047730    k3 = -0.041795    k4 = -0.079504
iter #6 y(0.600000) = -1.250235

k1 = -0.080557    k2 = -0.032854    k3 = -0.035239    k4 = 0.097502
iter #7 y(0.700000) = -1.252222

k1 = 0.095966    k2 = 0.309677    k3 = 0.298992    k4 = 0.587880
iter #8 y(0.800000) = -1.220536

-----
Runge Kutta-4 result: -1.220536
Expepected result      : -1.160000
Error value           : 0.060536

Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 0

All test values:
Step Size (h) | RK-4 Result | Error Value
-----
0.10000000    | -1.22053599 | 0.0605356929

-----
Process exited after 75.9 seconds with return value 1

```

```

Do you want to calculate RK-4 with different stepsize (Yes:1 / No:0): 0

All test values:
Step Size (h) | RK-4 Result | Error Value
-----
0.10000000    | -1.22053599 | 0.0605356929
0.05000000    | -1.22053799 | 0.0605376927
0.02000000    | -1.22053813 | 0.0605378302
0.00100000    | -1.22053813 | 0.0605378339
0.00010000    | -1.22053813 | 0.0605378339
0.00000005    | -1.33594786 | 0.1759475562

-----
Process exited after 1791 seconds with return value 3221226356
Press any key to continue

```



#### 4.1) Çıkarım

Runge Kutta 4 Metodu ile bulunan sonuçlarda çok çok az bir hata yapıldığı görülmektedir. Örnek çözümlerinin genelinde karşılaşılan hata sonuçlarının metottan değil programın sakladığı gerçek sonuç bilgisinin basamak eksikliğinden kaynaklandığı fark edilmiştir.

Örnek 1 ve 2’de polinom fonksiyon ile çalışılmış ve hata değeri 0 olarak gözlenmiştir. Örnek 3’te sinüs ve cosinüs fonksiyonları ile çalışılmış ve hata değerinin 0’a yakın olduğu gözlenmiştir. Örnek 4’te ise polinom ve cosinüs fonksiyonu içeren karma bir  $g(x)$  fonksiyonu ile çalışılmış ve 0.06 hata değeri gözlenmiştir. Aynı fonksiyon için farklı adım büyüklüğü değerleri kullanılmasına rağmen hata değerinin azalmadığı aksine arttığı fark edilmiştir. Hatta son denemede adım büyüklüğü 0.00000005 ve 16.000.000 iterasyon olmasına rağmen hata sonucunun azalmadığı; tekrardan arttığı gözlenmiştir.

Bu bilgiler doğrultusunda Runge Kutta 4 metodunun sadece polinom ve sadece trigonometrik fonksiyon içeren örneklerde hatasız çalışabildiği; karma fonksiyon kullanılan Örnek 4’e bakılarak Runge Kutta 4 metodunun farklı fonksiyon türleri içeren örneklerde mutlak hata değerinin çok daha fazla olduğu söylenebilir. Yani başka bir deyişle Runge Kutta 4 metodunun karma fonksiyon örnekleri için kullanmaya uygun olmadığı söylenebilir. Ayrıca adım büyüklüğün doğru belirlenmesi de hata oranına büyük ölçüde etki etmektedir.

Elbette bu hipotezlerin kanıtlanması için daha çok örnek üstünde çalışıp daha fazla veri elde etmek gerekmektedir.

#### KAYNAKÇA

Örnek 1:

<https://atozmath.com/example/CONM/RungeKutta.aspx?q=rk4&m=1&q1=E21>

Örnek 2:

<https://atozmath.com/example/CONM/RungeKutta.aspx?q=rk4&m=1&q1=E23>

Örnek 3 ve Örnek 4:

\*Kendim hazırladım.

Diferansiyel Denklemin Çözülmesi ve  $y(x)$  Fonksiyonunun Elde Edilmesi:

<https://www.emathhelp.net/en/calculators/differential-equations/differential-equation-calculator/?i=y%27%3D+-y%2B2x%5E3-x%2Bcos%285x%29%2C+y%280%29%3D+-2>

$y(x)$  Fonksiyonunun grafiğinin oluşturulması ve istenilen noktadaki değerinin elde edilmesi:

<https://www.desmos.com/calculator/eoihogspri>