Gazi Üniv. Müh. Mim. Fak. Der.
Cilt 17, No 2, 87-102, 2002

J. Fac. Eng. Arch. Gazi Univ.
Vol 17, No 2, 87-102, 2002

NÜMERİK YÖNTEMLERDE HATA ANALİZİ VE BİR NÜMERİK ÇÖZÜM PAKETİNİN HAZIRLANMASI

Selma YÜNCÜ*, Cihan ASLAN**
* Elektrik Elektronik Mühendisliği Bölümü, Mühendislik Mimarlık Fakültesi, GaziÜniversitesi, Maltepe 06570 Ankara, yuncu@mmf.gazi.edu.tr
** Şeker Bilişim Sanayi A.Ş. ascihan@yahoo.com

ÖZET

Bu çalışmada nümerik hataların analizi yapılarak örnekler verilmiştir. Ayrıca Bairstow kök bulma, Gauss yok etme, LU-Crout çözümleme, Cubicspline enterpolasyon, Romberg integral, Runge-Kutta yöntemlerinin programları C++ ile yazılmış, görsel nitelikli nümerik çözüm paketi haline getirilmiştir. Yazılan programlar normal, çift duyarlıklı ve uzun çift duyarlıklı değişken tipleri ile örneklendirilerek, sonuçlar çizelgeler ile birlikte sunulmuş, gerçek değerleri bilinen örneklerde hatalar hesaplanmıştır.

Anahtar Kelimeler: Nümerik analiz, nümerik hata, C++

ERROR ANALYSIS OF NUMERICAL METHODS AND PREPARATION OF A NUMERICAL SOLUTION PACKAGE

ABSTRACT

In this study, numerical error analysis was made and samples were provided. Additionally, programs for: Bairstow root finding, Gauss elimination method, LU - Crout analysis, Cubicspline interpolation, Romberg integral and Runge - Kutta methods were written with C++ and made into a visual interface numerical solution package. Samples of the written programs were given in float, double and long double variable types and the results were presented as drawings, and errors in the sample calculations were determined from samples with known real values.

Keywords: Numerical analysis, numerical error, C++

GİRİŞ

Nümerik yöntemler matematik problemlerinin formüle edilip, aritmetik işlemlerle çözülmesini sağlayan tekniklerdir. Çok sayıda ve türde nümerik yöntemler bulunduğu halde hepsinin özelliği çok miktardaki sıkıcı aritmetik işlemlerin yapılmasıdır. 1940 larda bilgisayarların bulunmasıyla birlikte nümerik yöntemlerin verimli şekilde kullanılması ve gelişmesi mümkün olmuştur. Son yıllarda mühendislik problemlerinin nümerik yöntemlerle çözüm yollarının artışında, yüksek hızlı ve verimli bilgisayarların gelişmesinin rolü büyüktür [1]. Bu konuda son zamanlarda literatürde bulunan çalışmalardan bir kısmı aşağıdadır.

1990 yılında Rekha P. Kulkarni ve Balmohan V. Limaye [2], Schrölinger operatörünün basit eigen değeri ve buna uygun dalga fonksiyonunun ,başlangıç değeri olarak Sloan 'ın Galerkin yönteminin iterasyonu ile yaklaşımı kullanılmıştır.

1995 yılında L. Banks-E.Chu [3], büyük sistemlerin lineer denklemlerin çözümünde yeni bir paralel iteratif algoritması sunmuşlardır.

1998 yılında X.-W. Chang [4], A=LU matrisi içerisinde, A gibi norm değeri yeterince küçük katsayıların LU faktorizasyon analizi ile elde edilen hassasiyetini incelemiştir.

1998 yılında, P. Langlois -F. Nativel [5], kayan nokta aritmetiğinde yuvarlatma hatalarının kontrolü ve azaltılması için yöntem sunmuşlardır.

2000 yılında M.Calvo-S.Gonzales-J.I.Montijano [6], cebirsel denklemlerde iterasyon çözümlerde Runge-Kutta yöntemini uygulamışlardır.

2000 yılında Chaya Gurwitz [7], quasi-newton yöntemlerinde kesme hatalarının, quasi-newton yönteminin yaklaşımında sınırsız biçimde artacağını göstermiştir.

2000 yılında T.G. Robertazzi ve S.C. Schwartz [8], sonlu yaklaşım aritmetiğinde kayan noktalı toplama işlemlerinin yapıldığı zaman, hassasiyet sonucunda doğru toplama sırasının etkisinin testini yapmıştır.

2. NÜMERİK YÖNTEMLERDE HATALAR

Nümerik yöntemlerde hatalar başlıca kesme ve yuvarlatma hataları olarak iki gurup altında incelenirler.

2.1. Kesme Hataları

Matematik işlemler yerine, yaklaşık olanlarının alınmasıyla ortaya çıkan hatalardır. Bu hata türüne örnek olarak Taylor serisin açılımıyla yapılan hatayı gösterebiliriz.

Taylor açılımına göre bir f fonksiyonu ve onun ilk n+1 aralığındaki türevleri x_{i+1} ve x_i aralığında sürekli ise, x_{i+1} deki fonksiyonun değeri, $h = x_{i+1} - x_i$ olarak alındığında, formül (2.1) göre bulunur.

$$f(x_{i+1}) = f(x_i) + f'(x_i)h + \frac{f''(x_i)}{2!}h^2 + \cdot\cdots\cdots + \frac{f^n(x_i)}{n!}h^n + R_n \qquad (2.1)$$

Taylor serisinde n inci terime kadar alınan toplamda R_n kesme hatasını verir. Eşitlik 2.2 yapılan hatayı gösterir.

$$R_n = \frac{f^{n+1}(\xi)}{(n+1)!}h^{n+1} \qquad (2.2)$$

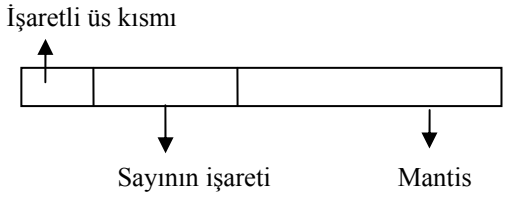
Formül 2.2'de ξ x_{i+1} ila x_i arasında yeri belirlenmeyen bir yerde bulunmakta, f^{n+1} ise f fonksiyonunun n+1'inci derecedeki türevi olmakta ve bilinmemektedir. Bu durumda eşitlik 2.2 nin değeri tam olarak bulunamasa da R_n'in, adım uzunluğu h^{n+1} ile doğru orantılı olduğu görülmektedir. Bu ise kesme hatasını azaltmanın adım uzunluğunu küçültmekle mümkün olacağını gösterir.

2.2. Yuvarlatma Hataları

Bilgisayarlar fiziksel yapılarından dolayı sayıların sadece bir kısmını belleklerinde saklayabilirler. Tam ve kesirli sayılar belleklerde kelime (word) yapısında tutulurlar. Tam sayılar 1 kelimede yerleşirken kesirli sayılar en az iki kelimeye yerleşirler. Kelime yapısı 1 byte (16 bit) olan bilgisayarlarda en küçük sayı -32767, en büyük sayı ise +32768 dir. Bu sayılardan daha küçük veya büyük olanlar ise kesirli sayı olarak mb^e formatında Şekil 2.1'deki gibi bulunurlar. Burada m mantis b=taban e=üsttür. Mantis $0 \leq m < 1$ aralığındadır.

Mantis kısmında rasyonalize halinde saklanan sayılar özellikle iterasyonla yapılan işlemlerde yuvarlatma hatalarına sebep olurlar. Bu hataları azaltmak için programlamalarda mantis kısmının hassasiyetini iki kat artıran çift duyarlıklı değişkenler kullanılır.

Ayrıca kesirli sayıların işlemlerinde atma (chopping) ve yuvarlatma (rounding)



Şekil 2.1. Kesirli sayıların bilgisayarda yerleşme şekli.

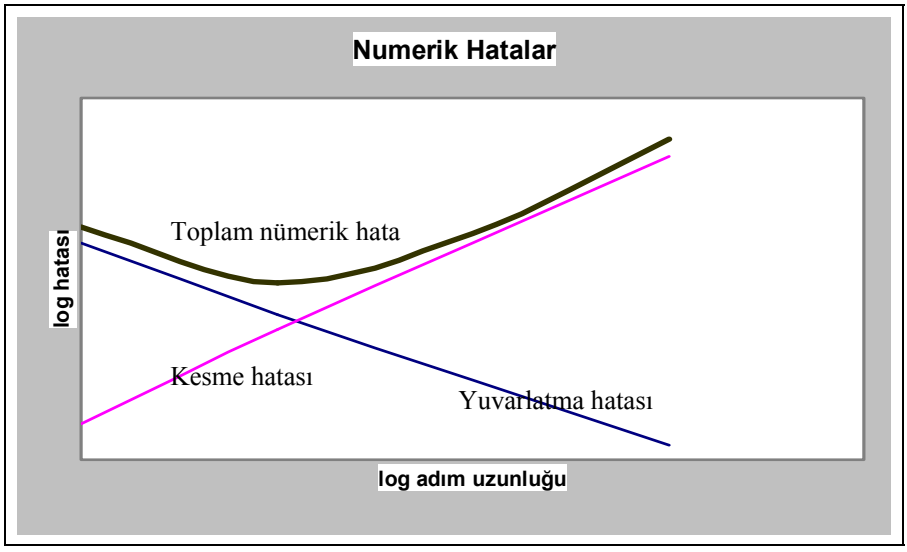
kullanırlar. Örneğin 6.7894856 gibi bir sayın 7 hanesi kullanılacaksa yuvarlatmada 6.789486 olurken atmada 6.789485 olur. Duyarlılığı fazla olan günümüz bilgisayarlarında atma yöntemi tercih edilmektedir; zira yuvarlatma için harcanan zaman çok fazla olmaktadır.

2.3. Toplam Nümerik Hata

Bilgisayarda işlem yaparken kesme ve yuvarlatma aynı anda yapılıyorsa, örneğin matematiksel bir açılımın ilk n parçası alınıyor ve işlemlerde yuvarlatma yapılıyorsa, iki hatanın toplamı olan hata toplam nümerik hatadır. Bu hataya hem kesmenin hem de yuvarlatmanın etkisi vardır. Genelde kesme hatalarını azaltmak için formül 2.2'deki h adım uzunluğunun azaltılması gerekir. Adım uzunluğu olan h değerinin azalması da yuvarlatma hatalarının artmasına sebep olacaktır. Şekil 2.2'de görüldüğü gibi toplam hatanın belirli bir noktadan sonra arttığı görülmektedir [1].

Gerek kesme, gerekse yuvarlatma hatalarında gerçek ve yaklaşık hata tanımları yapılır.

Gerçek hata bilgisayarlarda programı yazılan yöntemlerin doğruluğunu ispatlamak için, gerçek değeri bilinen durumlarda kullanılır. Nümerik hesaplamalarda gerçek değer bilinmediği için, özellikle iterasyonla yapılan işlemlerde de yaklaşık hata kullanılır. İngilizce true ve approximate kelimelerinin baş harfleri alınarak $\%\,\varepsilon_t$ ve $\%\,\varepsilon_a$ tanımları aşağıdaki gibi yapılır.



Şekil 2.2. Toplam nümerik hata.

ε_t = gerçek hata / gerçek değer

ε_a = (Şu andaki yak. değer-bir önceki yak.değer)/Şu andaki yak.değer)*100

Ayrıca iterasyon yapılan programlarda, n belirli hanenin doğru olmasını sağlayan formül 2.3'teki ε_s kullanılır.

$$\varepsilon_s = (0.5*10^{2-n})\% \qquad (2.3)$$

İterasyon işlemleri $|\varepsilon_a|<|\varepsilon_s|$ olana kadar devam eder. Bu kriter işlemlerde ancak $|\varepsilon_t|<|\varepsilon_a|$ olması durumunda kullanılır.

3. PAKET PROGRAMLAR

Bu çalışmada özellikleri aşağıda açıklanan nümerik yöntemlerle ilgili bir program paketi hazırlanmıştır [9]. C++ Builder ile yazılmış olan bu programa Şekil 3.1'deki bir anamenü yardımıyla ulaşılır. Bunlar:

- Bairstow kök bulma yöntemi
- Gauss yoketme yöntemi
- LU Crout çözümleme yöntemi
- Cubikspline enterpolasyon yöntemi
- Romberg integral alma yöntemi
- Runge Kutta adi diferansiyel çözümü yöntemi

Şekil 3.1. Nümerik analiz yöntemleri menüsü.

Bu programlarda yuvarlatma hatalarını minimuma indirmek için değişkenlerin bir kısmında çift duyarlıklı tanımlar yapılmıştır. Ayrıca programlar, normal (float), çift duyarlıklı (double) ve uzun çift duyarlıklı (long double) ile ayrı, ayrı çalıştırılarak sonuçlardaki farklar her yöntemin sonunda tablolarda gösterilmiştir.

3.1. Bairstow Yöntemi

Bairstow yöntemi bir kök bulma yöntemidir. Bu yöntemin diğer kök bulma yöntemlerine göre üstünlüğü gerçel kökleri bulduğu gibi sanal kökleri de bulmasıdır.

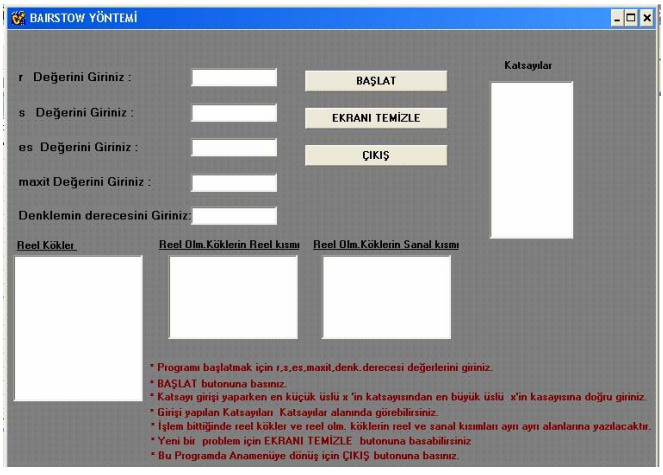
$f_n(x)= a_0 + a_1x + a_2x^2 +.....+a_nx^n$ (a_i gerçel sayı olmak üzere)

şeklinde verilen bir polinomun derecesi n tek ise, köklerden birisi mutlaka gerçeldir. Kalan kökler gerçel veya sanal olabilir. Eğer kökün biri sanal ise, diğer kök de mutlaka bu kökün eşleniği olan sanal bir sayıdır.

Bairstow yönteminde iterasyona başlamak için s ve r başlangıç noktaları alınır. Her adımdaki r ve s değerlerinin hesaplanan yaklaşık hatası aşağıdaki formüllerle bulunur.

$|\varepsilon_{a,r}|=|\Delta r|/|r|100\%$
$|\varepsilon_{a,s}|=|\Delta s|/|s|100\%$

Bu hatalar formül (2.3) ε_s kriter hatasından küçük olana kadar iterasyona devam eder. Bu yöntemde r ve s değerlerinin başlangıç değerlerinin seçilmesi önemlidir. Sonsuz döngüye girilmemesi için programa maksimum iterasyon sayısı konmuştur [1,10].



Şekil 3.2. Bairstow yöntemi programı.

Anamenüden birinci buton çalıştırılarak Şekil 3.2'deki bairstow menüsüne ulaşılır.

Örnek 3.1.

$x^5 + 5x^4 - 5x^3 - 25x^2 + 4x + 20 = 0 \; r = s = -1 \qquad \varepsilon_s = 1$

Tablo 3.1. Örnek 3.1'in Bairstow yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçları.

	C Float	C Long Double
X1	2.0000000000000000000	1.999999999999999300
X2	1.0000000000000000000	1.000000000000002910
X3	-1.000000000000000000	-1.000000132829983770
X4	-2.000000238418579100	-1.999999908406610850
X5	-4.999996185302734380	-4.999996284223565110

	C++ Float	C++Long Double	Xtrue
X1	2.0000000000000000000	1.999999999999999300	2
X2	1.0000000000000000000	1.000000000000002910	1
X3	-1.000000000000000000	-1.000000132829983770	-1
X4	-2.000000238418579100	-1.999999908406610850	-2
X5	-4.999996185302734380	-4.999996284223565110	-5

Tablo 3.2. Örnek 3.1 'in Bairstow yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçlarının hataları.

	%Hata C Float	%Hata C Long Double
X1	0,00000000000000000	0,000000000000035
X2	0,00000000000000000	0,000000000000291
X3	0,00000000000000000	0,000013282998377
X4	0,00001192092895500	0,0000045796694575
X5	0,00007629394531240	0,0000743155286978

	%Hata C++ Float	%Hata C++Long Double
X1	0,00000000000000000	0,000000000000035
X2	0,00000000000000000	0,000000000000291
X3	0,00000000000000000	0,000013282998377
X4	0,00001192092895500	0,0000045796694575
X5	0,00007629394531240	0,0000743155286978

3.2. Gauss Yoketme Yöntemi

$a_{11}x_1+a_{12}x_2+.....+a_{1n}x_n=c_1$
$a_{21}x_1+a_{22}x_2+.....+a_{2n}x_n=c_2$
.....
$a_{n1}x_1+a_{n2}x_2+.....+a_{nn}x_n=c_n$

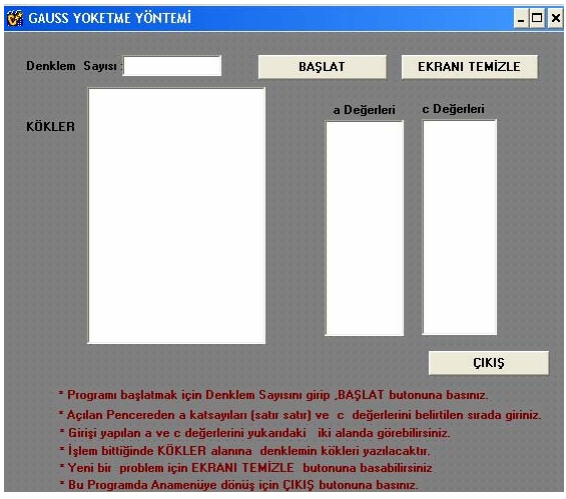
ile verilen denklem sistemi belirli bir algoritma ile yapılan cebrik işlemlerle bilinmeyen x vektorleri çözülür. Denklem sistemi rastgele (hiçbir işlem yapılmadan) çözülürlerse sıfıra bölme ve yuvarlatma hataları ortaya çıkar. Bu olumsuzlukları önlemek için denklem sistemini çözmeden önce pivotlama daha çok yarı pivotlama yapılır. Pivotlama sıfıra bölmeyi önlediği gibi yuvarlatma hatlarını da azaltır [1].

Ana menüden 2. buton çalıştırılarak Şekil 3.3'teki Gauss Yoketme menüsüne ulaşılabilir.

Örnek 3.2.

A =

7.6300	0.3000	0.1500	0.5000	0.3400	0.8400
0.3800	6.4000	0.7000	0.9000	0.2900	0.5700
0.8300	0.1900	8.3300	0.8200	0.3400	0.3700
0.5000	0.6800	0.8600	10.2100	0.5300	0.7000
0.7100	0.3000	0.8500	0.8200	5.9500	0.5500
0.4300	0.5400	0.5900	0.6600	0.3100	9.2500



Şekil 3.3. Gauss yoketme yöntemi programı.

C = [-9.4400, 25.2700, -48.0100, 19.7600, -23.6300, 62.5900]^T
Gerçek Değerler = -2 4 –6 2 –4 7

Tablo 3.3. Örnek 3.2'nin Gauss yoketme yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçları.

	C Float	C Long Double
X1	-1.99999988079071045	-1.999999999999999950
X2	4.00000000000000000	3.999999999999999640
X3	-6.00000000000000000	-5.999999999999999710
X4	2.00000023841857910	1.999999999999999970
X5	-3.9999997615814209	-3.999999999999999830
X6	7.00000000000000000	7.000000000000000320

	C++ Float	C++ Long Double	G.
X1	-1.99999988790710450	-1.999999999999999950	-2
X2	4.00000000000000000	3.999999999999999640	4
X3	-6.00000000000000000	-5.999999999999999710	-6
X4	2.00000023841857910	1.999999999999999970	2
X5	-3.99999976158142090	-3.999999999999999830	-4
X6	7.00000000000000000	7.000000000000000320	7

Tablo 3.4. Örnek 3.2'nin Gauss yoketme yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçlarının hataları.

	%Hata C Float	%Hata C Long Double
X1	0.00000596046447753906	0.00000000000000000000
X2	0.00000000000000000000	0.00000000000001110223
X3	0.00000000000000000000	0.00000000000000000000
X4	0.00001192092895507812	0.00000000000000000000
X5	0.00000596046447753906	0.00000000000000000000
X6	0.00000000000000000000	0.00000000000000000000

	%Hata C++ Float	%Hata C++ LD
X1	0.00000596046447753906	0.00000000000000000000
X2	0.00000000000000000000	0.00000000000011102230
X3	0.00000000000000000000	0.00000000000000000000
X4	0.00001192092895507812	0.00000000000000000000
X5	0.00000596046447753906	0.00000000000000000000
X6	0.00000000000000000000	0.00000000000000000000

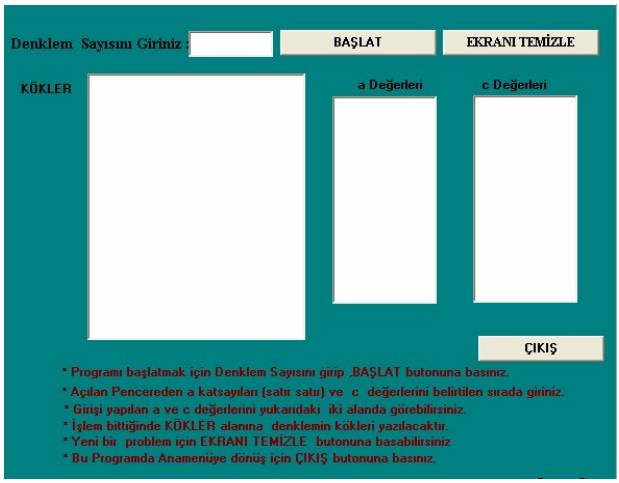
3.3. LU -Crout Çözümleme Yöntemi Programı

Bu yöntemde katsayılar matrisi olan kare matris biri alt üçgen, diğeri ise üst üçgen olarak aşağıda görüldüğü gibi iki matrise ayrılır.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} . \begin{bmatrix} 1 & u_{12} & u_{13} \\ 0 & 1 & u_{23} \\ 0 & 0 & 1 \end{bmatrix}$$

Bu eşitlikten yararlanarak bilinmeyen x değerleri çözülür [11].

Ana menüden 3. buton çalıştırılarak Şekil 3.4'teki LU Crout menüsüne ulaşılabilir.



Şekil 3.4. LU -Crout çözümleme yöntemi programı.

Örnek 3.3.

A=

4.3800	0.5900	0.4400	0.1200	0.8000	0.8400	0.3900	0.1600
0.0900	81.9300	0.3500	0.4500	0.9100	0.1700	0.5900	0.8700
0.0400	0.3700	43.1700	0.7200	0.2300	0.1700	0.1200	0.2400
0.6100	0.6300	0.6800	89.9300	0.2400	0.9900	0.0400	0.6500
0.6100	0.7200	0.7000	0.2700	73.5400	0.4400	0.4600	0.9700
0.0200	0.6900	0.7300	0.2500	0.0800	69.0700	0.8700	0.6600
0.0200	0.0800	0.4800	0.8700	0.6400	0.3100	35.5500	0.8700
0.1900	0.4500	0.5500	0.2300	0.1900	0.3700	0.2600	16.6100

C = [-23.49 87.25 170.88 -530.36 227.13 141.59 -136.83 117.43]^T

Tablo 3.5. Örnek 3.3'ün LU-Crout çözümleme yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçları.

LU	C Float	C Long Double
X1	-2.000000000000000000	-1.999999999999997622340
X2	0.999999940395355225	0.999999999999999161912
X3	4.000000476837158200	3.999999999999997272150
X4	-6.000000000000000000	-5.999999999999997055310
X5	3.000000000000000000	2.999999999999996849310
X6	2.000000238418579100	2.000000000000002504510
X7	-4.000000476837158200	-4.000000000000006865170
X8	6.999999523162841800	7.000000000000006583280

	C++ Float	C++ Long Double	G
X1	-2.000000000000000000	-1.999999999999999760	-2
X2	-0.999999940395355225	0.999999999999999916	1
X3	4.000000476837158200	3.999999999999999730	4
X4	-6.000000000000000000	-5.999999999999999710	-6
X5	3.000000000000000000	2.999999999999999680	3
X6	2.000000238418579100	2.000000000000000250	2
X7	-4.000000476837158200	-4.000000000000000690	-4
X8	6.999999523162841800	7.000000000000000660	7

Tablo 3.6. Örnek 3.3'ün LU-Crout çözümleme yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçların hataları.

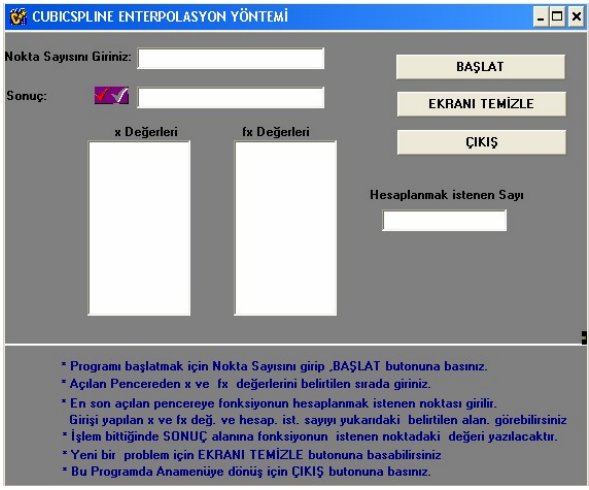
	% Hata C Float	% Hata C Long Double
X1	0.000000000000000000000	0.000000000000011102230
X2	0.000005960464477539062	0.000000000000011102230
X3	0.000011920928955078125	0.000000000000011102230
X4	0.000000000000000000000	0.000000000000000000000
X5	0.000000000000000000000	0.000000000000014802973
X6	0.000011920928955078125	0.000000000000022204460
X7	0.000011920928955078125	0.000000000000022204460
X8	0.000006811959402901785	0.000000000000012688263

	% Hata C++ Float	% Hata C++ Long Double
X1	0.000000000000000000000	0.000000000000111022302
X2	0.000005960464477539062	0.000000000000111022302
X3	0.000011920928955078125	0.000000000000111022302
X4	0.000000000000000000000	0.000000000000000000000
X5	0.000000000000000000000	0.000000000000148029737
X6	0.000011920928955078125	0.000000000000222044605
X7	0.000011920928955078125	0.000000000000222044605
X8	0.000006811959402901785	0.000000000000126882631

3.4. Cubicspline Enterpolasyon Yöntemi

Deney sonucu veya benzer çalışmalar için bilinen değerleri kullanarak bilinmeyen noktalardaki değerleri yaklaşık olarak belirleme işlemine enterpolasyon denilmektedir. Bilinmeyen değerler bilinen değerlerin arasında bir noktada ise bilinen noktalar kullanılarak bulunabilir. Ancak istenen nokta bilinen değerlerin dışında bir yerde ise öncelikle bu noktaları sağlayan bir eşitlik bulunur. Daha sonra bu eşitlikten bilinmeyen nokta verilerek istenen karşılığı elde edilir [10].

Ana menüden 4. buton çalıştırılarak Şekil 3.5'teki Cubicspline menüsüne ulaşılabilir.



Şekil 3.5. Cubicspline enterpolasyon yöntemi programı.

Örnek 3.4.

x=(0.08823, 0.14706, 0.20588, 0.26471, 0.32353, 0.38235, 0.44118, 0.5, 0.55882, 0.61765, 0.67647, 0.73529, 0.79412, 0.85294, 0.91176)

f(x)=(0.185, 0.21, 0.23, 0.255, 0.28, 0.31, 0.34, 0.38, 0.42,, 0.45, 0.50, 0.54, 0.59, 0.65, 0.69)

Tablo 3.7. Örnek 3.4'ün cubicspline yöntemi ile ve değişik değişken türleri kullanılarak elde edilen çözümünün sonuçları.

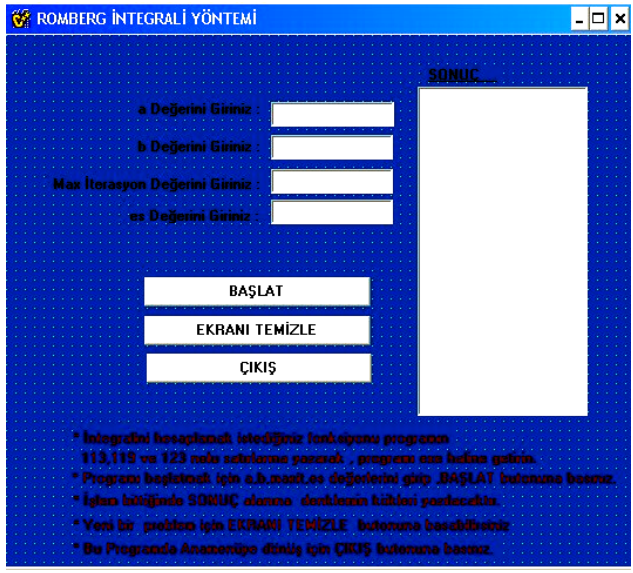
C float	C long double	C++ Builder Float	C++ Builder Long Dou
0.295075476169586182	0.280420035069537383	0.295075505971908569	0.280420035069537383

3.5. Romberg İntegrali

Fonksiyonların integrali nümerik yöntemlerde yamuk, simpson, Gauss, Romberg integrasyon yöntemleri ile çözülür. Bunlardan Romberg integrasyon yöntemi aşağıdaki formül yardımıyla iterasyonla çözülen yöntemdir [1].

$$I_{j,k} \cong \frac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1}$$

Burada $I_{j+1,k-1}$ ve $I_{j,k-1}$ sırasıyla en yüksek ve en düşük hassasiyetteki integrallerdir. Ana menüden 5. buton çalıştırılarak Şekil 3.6'daki Romberg İntegrali menüsüne ulaşılabilir.



Şekil 3.6. Romberg integrali yöntemi programı.

Örnek 3.5.

$f(x)=0.2+25x-200x^2+675x^3-900x^4+400x^5$ fonksiyonunun [0,0.8] aralığındaki integralini romberg yöntemi ile bulunuz.

a değeri= 0 b değeri=0.8 Max iterasyon değeri=4 es değeri=1

Tablo 3.8. Örnek 3.5'in Romberg yöntemi ile bulunan integral değerleri.

Romberg integrali (C++ Float)	*Romberg integrali (C++ LD)*
1.367466688156127930	1.367466666666666730
1.623466610908508300	1.623466666666666250
1.640533328056335450	1.640533333333332880
1.639466524124145510	1.639466666666666420
1.640533208847045900	1.640533333333333100
1.640533208847045900	1.640533333333333100
1.640466451644897460	1.640466666666666730
1.640533089637756350	1.640533333333333420
1.640533089637756350	1.640533333333333420
1.640533089637756350	1.640533333333333420
1.640533089637756350	1.640533333333333420

Gerçek Değeri=1.640533

Tablo 3.9. Örnek 3.5 'nin romberg yöntemi ile bulunan integral değerlerinin hataları.

%Hata (C++ Float)	%Hata (C++ LD)
0.000005463941069762083	0.000020318599712410539

3.6. Runge-Kutta Yöntemi

Runge Kutta Formülleri yardımıyla elde edilirler [12]. Runge Kutta yöntemlerinde n arttıkça hassasiyet artar. Birçok çözümde 4. mertebeden Runge Kutta yöntemi yeterli olurken, daha hassas çözüm isteyen durumlarda yüksek dereceden Runge Kutta yöntemleri kullanılır. Burada 6. mertebeden Runge Kutta yöntemini kullanan program yazılmıştır.

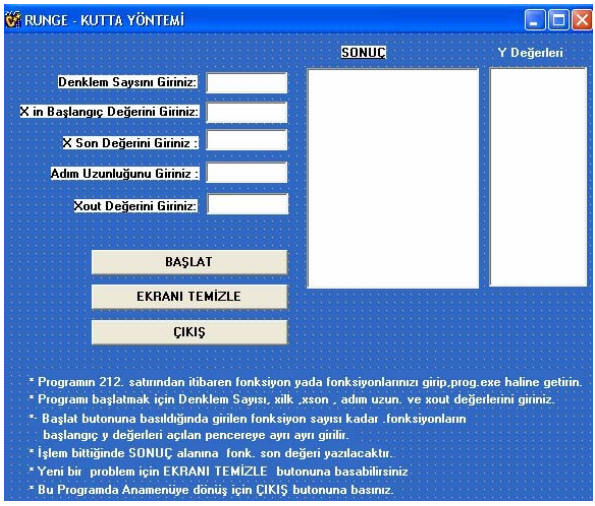
Ana menüden 6. buton çalıştırılarak Şekil 3.7'deki Runge-Kutta menüsüne ulaşılabilir.

Örnek 3.6.

y'_1=4*exp(0.8*x)-0.5*y1 Y(0)=0.5 h=0.5 Xout= 0.5

Tablo 3.10. Örnek 3.6'nın Runge-Kutta yöntemi ile bulunan sonuçları.

X	*Y C Float*	*Y C Long Double*
0	2.000000000000000000	2.000000000000000000
0.5	3.75152206420898438	3.751522082653013334501



Şekil 3.7. Yüksek dereceden Runge-Kutta yöntemi programı.

Tablo 3.11. Örnek 3.6'nın Runge-Kutta yöntemi ile bulunan sonuçların hataları.

X	*Y (Runge-Kutta Yöntemi)*	*Ytrue*	*Hatası (%)*
0	2.000000000000000000		
0.5	3.751522082653013270	3.751521	0,000028859041793

4. SONUÇ VE ÖNERİLER

Son yıllarda bilgisayarların hız ve kapasitelerinin artması; pek çok mühendislik problemlerinin de bilgisayarlarda çözülmesini, nümerik analiz yöntemlerinin artmasını, yeni algoritmaların gelişmesini sağlamıştır. Bu ise gerek algoritmik yöntemlerden gerekse bilgisayarların yapılarından kaynaklanan hataları da beraberinde getirmiştir.

Bu çalışmada nümerik yöntemlerdeki hataların analizi yapılmıştır. Nümerik metotların konularından birer yöntem seçilerek bir nümerik yöntem paketi hazırlanmıştır. Bu yöntemlerde değişik hassasiyette değişkenler kullanılarak örnekler çözülmüş, sonuçlar ve hatalar tablolar ve grafikler yardımıyla gösterilmiştir. Her yöntemin tanımı yapılarak, menülerle çalışması açıklanmıştır. Bazı yöntemlerde, sonuca ulaşmak için dikkat edilmesi gereken hususlar üzerinde durulmuştur.

Tablo 3.4 ve Tablo 3.6'daki sonuçlarda görüldüğü gibi değişkenleri çift duyarlıklı ve uzun çift duyarlıklı alınan durumlarda hataların daha az olduğu görülmektedir. Tablo 3.9'da ise normal duyarlıklı değerlerde hatalar daha az gibi gözükse de bunun nedeni hata hesaplarında gerçek değerlerin alınması ve işlemlerde yuvarlatma yapılmasıdır.

Aslında değişkenlerin normal, çift duyarlıklı ve uzun çift duyarlıklı alınarak yapılan işlemlerdeki büyük farklar, iterasyonlarla yapılan işlemler ve büyük denklem sistemlerinin çözümlerinde daha belirgin olur.

Değişik yöntemlerle de programlar yazılarak nümerik paket geliştirilebilir. Bu pakette Romberg ve Runge Kutta yöntemlerinde fonksiyonlar sabit olarak verilmiştir. Yani her yeni fonksiyon için, fonksiyon girildikten sonra programın tekrar derlenmesi gerekir. İdeal olanı ise fonksiyonların ekrandan girilebilecek hale getirilmeleridir. Ayrıca bu paket, daha değişik örneklerle denenip değişik sonuçlar alınabilir.

KAYNAKLAR

1. Chapra, S. C. ve Canale, R.P., **Numerical Methods for Engineers** McGraw-Hill, 1998.
2. Kulkarni, R. P. ve Limaye, V., Solution of a Schrödinger Equation by Iterative Rrefinement, **J Austral Math Soc Ser B**, 32, s. 115-132, 1990.
3. Banks, L. ve Chu, E., Parallel Gauss-Seidel Relaxation on Distributed-Memory Multiprocessor, **J.Comput.Inf**., 1, No 2, s. 434-450, 1995.
4. Chang, X.-W., On the Sensitivity of the LU Factorization, **BIT,** 3, s. 486-501, 1998.
5. Langlois, P. ve Nativel, F., Reduction and Bounding of the Rounding Error in Floating Point Arithmetic, **C.R.Acad.Sci. Paris**, 1, s. 781-786, 1998.
6. Calvo, M., Gonzales, S., Montijano, J.I., On the Iterative Solution of the Algebraic Equations in Fully Implicit Runge-Kutta Methods, **Numerical Algorithms**, 23 ,s. 97-113, 2000.
7. Gurwitz, C., A Test for Cancellation Errors in Quasi-Newton Methods, **ACM,** s.134-140, 2000.
8. Robertazzi, T.G. ve Schwartz, S.C., Best "Ordering" for Floating-Point Addition, **ACM**, s.101-110, 2000.
9. Aslan, C., **Nümerik Yöntemlerde Hata Analizi ve C++ Builder ile Yazılan Bir Nümerik Çözüm Paketinin Hazırlanması,** Master Tezi, Gazi Üniversitesi Fen Bilimleri Enstitüsü, 2002.
10. James, M.L., Smith, G.M., Wolford, J.C. **Applied Numerical Methods for Digital Computation,** Harper Collins, 1993.
11. Hoffmann, J.D., **Numerical Methods for Engineers and Scientists,** McGraw-Hill, 1993.
12. Fausett, L.V., **Applied numerical analysis using matlab,**Prentice Hall, 1999.
13. Ping-Tang, T.P, Table–Driven Implementation of the Logarithm Function in IEEE Floating Point Arithmetic, **ACM Transaction on Mathematical Software,** 4, s. 378-400., 1990.