|  |
| --- |
| PARALEL PROGRAMLAMA |
|  |
| **Trapezoid Kuralına göre 2. derece integralin sayısal olarak hesaplanması (C + MPI ile kod çalıştırma, performans ve speedup ölçümü)** |

15.12.2014

İçindekiler

[1. İntegrasyon 3](#_Toc406379361)

[2. Sayısal İntegrasyon 4](#_Toc406379362)

[3. Trapezoid Kuralı 5](#_Toc406379363)

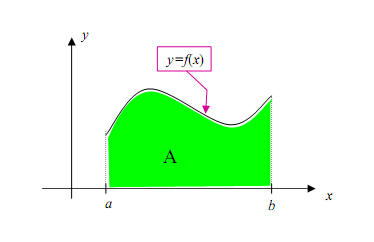
[4. MPI İmplementasyonu 5](#_Toc406379364)

[5. Performans Ölçümleri 7](#_Toc406379365)

[6. Kaynaklar 8](#_Toc406379366)

# İntegrasyon

İntegrasyon verilen bir f(x) fonksiyonu için, y = f(x) eğrisi ile x ekseni arasında kalan bölgenin alanının hesaplanmasıdır.



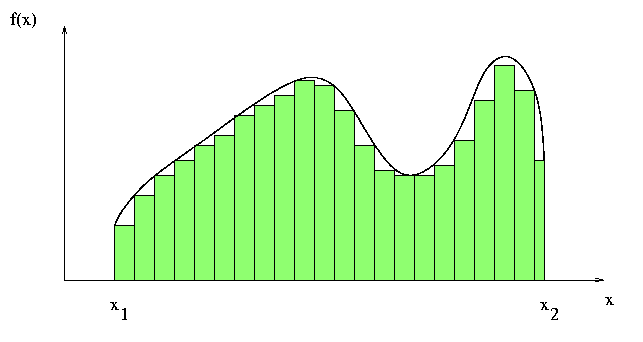
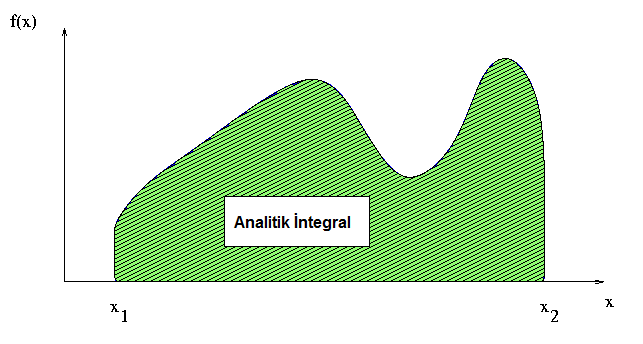
A =

f(x) -> integrali alınacak fonksiyon

a -> integralin alt limiti

b -> integralin üst limiti

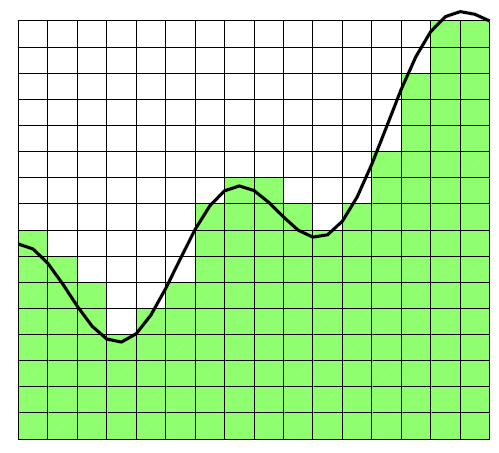
Bu integralin hesabı analitik (gerçek çözüm) veya sayısal (yaklaşık çözüm) olarak hesaplanabilir.



1. Analitik integral b) Sayısal integral

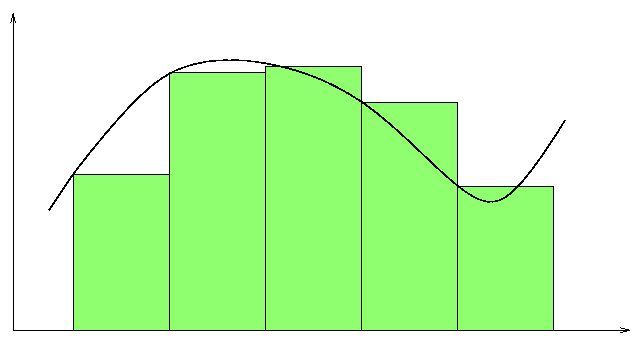
# Sayısal İntegrasyon

Sayısal integrasyon, integralin analitik çözümüne çeşitli yöntemlerle yaklaşma metodudur. Sayısal integrasyon yöntemleri grafik üzerinde görüldüğü gibi yaklaşık değerleri verir.



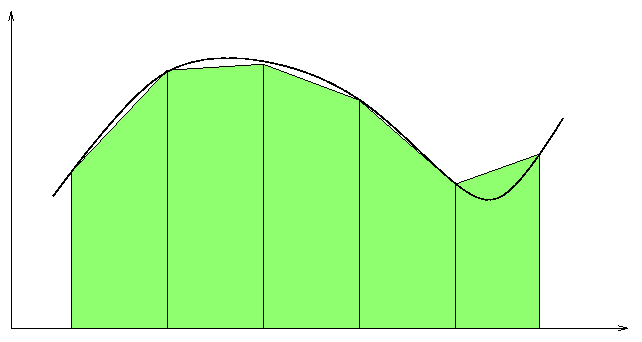
Sayısal integrasyonda kullanılan yöntemler;

1. Kare yöntemi, bu yöntemde fonksiyonun her bir aralıkta sabit bir değeri olduğu varsayılır.



I ) Kare yöntemi

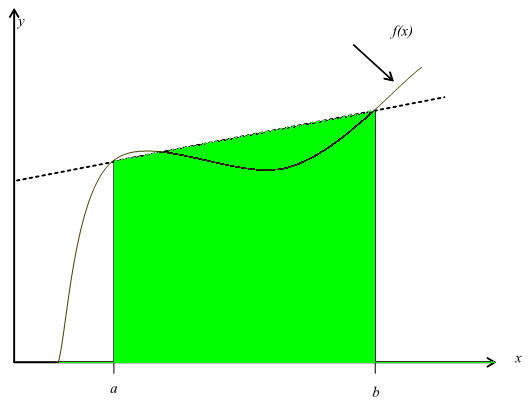
1. Trapezoid yöntemi, bu yöntemde kareler yerine yamuklar kullanılır. Verilen fonksiyonun integrali hesaplanırken a ve b aralığında x ekseni ile arasına uygun bir yamuk çizilerek bu yamukların alanı hesaplanır. Kare yöntemine göre daha yaklaşık sonuç verir.



II) Trapezoid yöntemi

# Trapezoid Kuralı

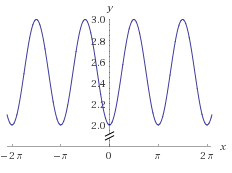
Trapezoid kuralı integrale yamuklar ile yaklaşma metodudur. Verilen a ve b noktalarının oluşturacağı yamukların alanı yaklaşık integral değerini verir.



≈ yamuğun alanı

# MPI İmplementasyonu

MPI kütüphanesi kullanılarak aşağıdaki fonksiyonun integrali paralel olarak hesaplanacak ve performans ölçümleri yapılacaktır.



f(x) fonksiyonunun hesaplanması C dilinde aşağıdaki kod ile yapılmaktadır. C dilindeki sin() fonksiyonu radyan cinsinden giriş parametresi alır. Bir radyan 180/PI derece olduğu için hesaplanacak olan x derecesi PI/180 ile çarpılarak radyana çevrilir.

double f(double x)

{

return pow(sin(x \* (3.14/180)), 2) + 2;

}

Program ilk çalışmaya başladığında Get\_input() fonksiyonu ile kullanıcıdan hesaplanacak integralin alt ve üst sınırlarını girmesi istenir (a - b) . Daha sonra n parametresi ile hesaplamada bu iki sınır arasında kaç yamuk kullanılacağı bilgisi alınır. Yamuk sayısı arttıkça gerçek sonuca daha çok yaklaşılır ancak hesaplama süresi artar. Kullanıcı girişi tamamlandığında bu parametreler MPI\_Bcast() fonksiyonu ile tüm proseslere gönderilir.

void Get\_input(int my\_rank, double\* a\_p, double\* b\_p, int\* n\_p)

{

if (my\_rank == 0)

{

printf("Enter a, b, and n\n");

scanf("%lf %lf %d", a\_p, b\_p, n\_p);

}

MPI\_Bcast(a\_p, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(b\_p, 1, MPI\_DOUBLE, 0, MPI\_COMM\_WORLD);

MPI\_Bcast(n\_p, 1, MPI\_INT, 0, MPI\_COMM\_WORLD);

}

Her prosesin hesaplayacağı alt ve üst sınır aralıkları aşağıdaki gibi hesaplanır. Kullanıcı tarafından girilen yamuk sayısı (n) proses sayısına bölünerek hesaplanacak yamuk sayısı proseslere eşit olarak dağıtılır ve bu parametreler ile Trap() fonksiyonu çağırılır. h parametresi birim yükseklik bilgisini içerir.

h = (b-a)/n;

local\_n = n/comm\_sz;

local\_a = a + my\_rank\*local\_n\*h;

local\_b = local\_a + local\_n\*h;

local\_int = Trap(local\_a, local\_b, local\_n, h);

Trap() fonksiyonu ile her proses için hesaplanan alt ve üst sınırlar ile proses başına düşen yamuk sayısı kadar alan hesaplanır.

double Trap(double left\_endpt, double right\_endpt, int trap\_count, double base\_len)

{

double estimate, x;

int i;

estimate = (f(left\_endpt) + f(right\_endpt))/2.0;

for (i = 1; i <= trap\_count-1; i++)

{

x = left\_endpt + i\*base\_len;

estimate += f(x);

}

estimate = estimate\*base\_len;

return estimate;

}

Tüm prosesler kendine ayrılan alanları hesapladıktan sonra MPI\_Reduce() fonksiyonu bu alanları MPI\_SUM parametresi yardımıyla toplar ve toplam alan hesaplanmış olur.

MPI\_Reduce(&local\_int, &total\_int, 1, MPI\_DOUBLE, MPI\_SUM, 0, MPI\_COMM\_WORLD);

# Performans Ölçümleri

Performans hesaplamaları alt sınır (a) 1, üst sınır (b) 100000 ve yamuk sayısı (n) 1000 olacak şekilde yapılmıştır. İlk olarak 1 proses ile programın seri çalışma süresi hesaplanmış daha sonra 2, 4, 8 ve 16 proses için ayrı ayrı hesaplama zamanları elde edilmiştir.

|  |  |  |
| --- | --- | --- |
| **PROSES SAYISI** | **TOPLAM HESAPLAMA SÜRESİ** | **SPEEDUP FACTOR** |
| 1 | 0,217247 | 1 |
| 2 | 0,117613 | 1,85 |
| 4 | 0,071914 | 1,64 |
| 8 | 0,053618 | 1,34 |
| 16 | 0,059216 | 0,91 |

1. Proses sayısı – İşlem süresi grafiği

b) Proses sayısı – Speedup factor grafiği

# Kaynaklar

1. <http://mcs.une.edu.au/~comp309/Lectures/Lecture_21/lecture/>
2. <http://www.wolframalpha.com/input/?i=sin%28x%29%5E2%2B2>
3. <http://en.wikipedia.org/wiki/Trapezoidal_rule>
4. <http://www.symbolab.com/solver/definite-integral-calculator/%5Cint_%7B1%7D%5E%7B100000%7D%20sin(x)%5E%7B2%7D%2B2/?origin=button>
5. <http://www.wolframalpha.com/widgets/view.jsp?id=8ab70731b1553f17c11a3bbc87e0b605>