**Numerical Integration  
Trapezoidal Rule and Gauss Quadrature**

Arely Miramontes1, Berenise Miramontes2

*1Computational Mathematics, University of New Mexico*

arelym@unm.edu

*2Computational Mathematics, University of New Mexico*

beremts12@unm.edu

***Abstract – In this report we discuss approximating integrals using two different methods. The first method is the Trapezoidal Rule, and the second is the Gauss Quadrature.***

***Keywords – Fortran, numerical integration, Trapezoidal Rule, Gauss Quadrature, MATLAB*.**

**I. INTRODUCTION**

To compute the value of I, we consider the following integral:

We will use this integral to approximate ***I*** with two different methods: Trapezoidal Rule and Gauss Quadrature.

**II. TRAPEZOIDAL RULE,**

“The trapezoidal rule belongs to a class of quadrature called Newton-Cotes quadrature that approximates integrals using equidistant grids. The order of the composite Newton-Cotes method is typically or , where is the number of points in each panel (2 for the Trapezoidal Rule).” [1]

For a grid the composite trapezoidal rule is:

Our assignment was to write a Fortran code that implements the Trapezoidal Rule. Using 100 points, we approximated the integral:

First we write a function called “trapRule.m” that evaluates the integral f(x) dx using the Trapezoidal Rule:

function [s,valuesofErr] = trapRule(f,a,b,m,c)

% f is the function

% a and b are the limits

% m is the number of subintervals

% c is the exact integral

disp(['--------------------------------'])

disp([' Step ', ' Approx ','Error'])

disp([' sizeh ',' solution ',''])

disp(['--------------------------------'])

valuesofErr = zeros(1,m);

for i=0 : m

m = 2^i;

h = (b-a)/m;

s = 0;

for k=1 : (m-1)

x = a+h\*k;

s = s+feval(f,x);

end

s = h\*(feval(f,a)+feval(f,b))/2+h\*s;

err = abs(c-s);

valuesofErr(i+1) = err;

disp([h s err])

end

end

We use two functions to review our data, , and . We write our script file that grabs the Trapezoidal value and the value of the errors, and plot the error against n.

clear

clc

close all

n = 10;

a = -1;

b = 1;

f = @(x) exp(cos(pi\*x));

c = integral(f,a,b)

f2 = @(x) exp(cos(pi\*pi\*x));

c2 = integral(f2,a,b)

[s,valuesofErr] = trapRule(f,a,b,n,c);

[s2, valuesofErr2] = trapRule(f2,a,b,n,c2);

x = 1:1:n+1;

loglog(x,valuesofErr,'ro-','linewidth',3)

hold on

loglog(x,valuesofErr2,'b\*-','linewidth',3)

axis([0 n+1 0 1000])

xlabel('n')

ylabel('Maximum Error')

legend('Error for: e^{cos{pix}}','Error for: e^{cos{pi^2x}}')

title('Trapezoidal Rule')

grid on

When executing this script, we get output the exact integral solution, h step, the approximate solution, and the error.

Let’s look at the data for :

|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **Step size h** | **Approximate Solution** | **Error** |
| 1 | 2.0000 | 0.7358 | 1.7964 |
| 2 | 1.0000 | 3.0862 | 0.5540 |
| 3 | 0.5000 | 2.5431 | 0.0109 |
| 4 | 0.2500 | 2.5321 | 0.0000 |
| 5 | 0.1250 | 2.5321 | 0.0000 |
| 6 | 0.0625 | 2.5321 | 0.0000 |
| 7 | 0.0312 | 2.5321 | 0.0000 |
| 8 | 0.0156 | 2.5321 | 0.0000 |
| 9 | 0.0078 | 2.5321 | 0.0000 |
| 10 | 0.0039 | 2.5321 | 0.0000 |
| 11 | 0.0020 | 2.5321 | 0.0000 |

Now, let us look at the data for

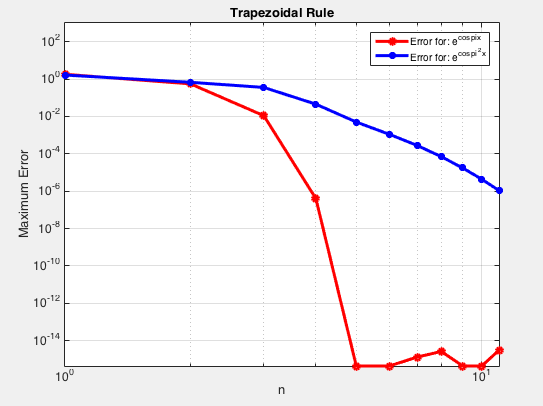
|  |  |  |  |
| --- | --- | --- | --- |
| **i** | **Step size h** | **Approximate Solution** | **Error** |
| 1 | 2.0000 | 0.8110 | 1.6413 |
| 2 | 1.0000 | 3.1238 | 0.6715 |
| 3 | 0.5000 | 2.8087 | 0.3564 |
| 4 | 0.2500 | 2.4070 | 0.0453 |
| 5 | 0.1250 | 2.4573 | 0.0050 |
| 6 | 0.0625 | 2.4534 | 0.0011 |
| 7 | 0.0312 | 2.4526 | 0.0003 |
| 8 | 0.0156 | 2.4524 | 0.0001 |
| 9 | 0.0078 | 2.4523 | 0.0000 |
| 10 | 0.0039 | 2.4523 | 0.0000 |
| 11 | 0.0020 | 2.4523 | 0.0000 |

Now, you notice that the first function with converges a lot faster than the function who has .

QUESTION: What is special with the integrand in the case k=π and why does it make the method converge faster then expected?

When k=π2, and we get a very large error, this is an upper bound on the error, because we have neglected the possibility that the errors from different intervals will be of opposite signs and mostly cancel.

Now, let us view the plot to compare the convergence.



You can now graphically see that converges a lot faster than using the Trapezoidal Rule.

**III. GUASS QUADRATURE**

“In Guass quadrature the location of the grid-points (usually referred to as nodes) and weights, 𝜔i,

are chosen so that the order of the approximation to the weighted integral is maximized.” [1]

Here, w(z) is the weight function and it is both positive and integratable. For a given w(i) the nodes, called zi, are the zeroes of the polynomial satisfying

for all polynomials q(z) of degree less than n. [1]

To compute the Gauss Quadrature in MATLAB, we first imported “lglnodes.m” by the author Greg vib Winckel:

function [x,w,P]=lglnodes(N)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%

% lglnodes.m

%

% Computes the Legendre-Gauss-Lobatto nodes, weights and the LGL Vandermonde

% matrix. The LGL nodes are the zeros of (1-x^2)\*P'\_N(x). Useful for numerical

% integration and spectral methods.

%

% Reference on LGL nodes and weights:

% C. Canuto, M. Y. Hussaini, A. Quarteroni, T. A. Tang, "Spectral Methods

% in Fluid Dynamics," Section 2.3. Springer-Verlag 1987

%

% Written by Greg von Winckel - 04/17/2004

% Contact: gregvw@chtm.unm.edu

%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Truncation + 1

N1=N+1;

% Use the Chebyshev-Gauss-Lobatto nodes as the first guess

x=cos(pi\*(0:N)/N)';

% The Legendre Vandermonde Matrix

P=zeros(N1,N1);

% Compute P\_(N) using the recursion relation

% Compute its first and second derivatives and

% update x using the Newton-Raphson method.

xold=2;

while max(abs(x-xold))>eps

xold=x;

P(:,1)=1; P(:,2)=x;

for k=2:N

P(:,k+1)=( (2\*k-1)\*x.\*P(:,k)-(k-1)\*P(:,k-1) )/k;

end

x=xold-( x.\*P(:,N1)-P(:,N) )./( N1\*P(:,N1) );

end

w=2./(N\*N1\*P(:,N1).^2);

Next, we write our own MATLAB script called, “gq\_test.m”, that calls the lglnodes function, calculates the Gauss Quadrature, and plots the error by the number of subintervals, n.

clear

clc

N = 10;

a = -1;

b = 1;

[x,w,P] = lglnodes(N)

f1 = exp(cos(pi\*pi.\*x));

f2 = exp(cos(pi.\*x));

for i=1 : N+1

integralVal1(i) = sum(f1.\*w(i));

integralVal2(i) = sum(f2.\*w(i));

f\_x1 = @(x) exp(cos(pi\*pi\*x));

f\_x2 = @(x) exp(cos(pi.\*x));

exactIntegral1 = integral(f\_x1,a,b)

exactIntegral2 = integral(f\_x2,a,b)

Err1(i) = abs(exactIntegral1 - integralVal1(i))

Err2(i) = abs(exactIntegral2 - integralVal2(i))

end

subintervals = 1:1:N+1;

loglog(subintervals,Err1,'r\*-','linewidth',2)

hold on

loglog(subintervals,Err2,'bo-','linewidth',2)

grid on

xlabel('n')

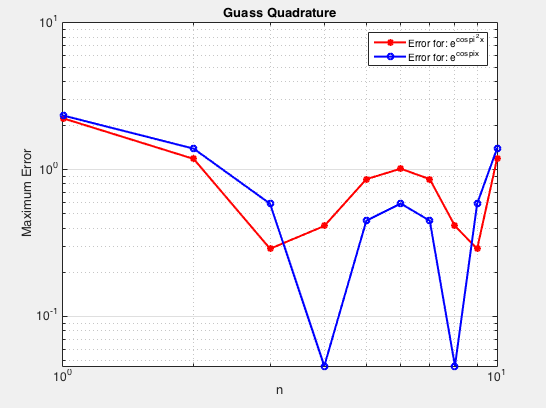
ylabel('Maximum Error')

legend('Error for: e^{cos{pi^2x}}','Error for: e^{cos{pix}}')

title('Guass Quadrature')

axis([0 10 0 10])

When we run our script, we get the following plot:



As you can see it does not come as a surprise that the function converges a lot faster than .

**IV. CONCLUSIONS**

There are other Integration methods we could use to approximate an integral over an interval. The Trapezoidal rule was the easiest than the Gauss Quadrature, but both have their pros and cons. For the Trapezoidal Rule it can be very useful because the solution is very rapidly convergent, it consists the fact that the weighting coefficients are nearly equal to each other, and also it is very easy to implement. The downside to this method is that the error can be much higher compared to other methods. The advantage of the Gauss Quadrature is that it has a high degree of precision. The disadvantages are that it can be difficult to obtain the weights and nodes, and also irrational weights and nodes can be a factor.

Overall these two are good methods for numerical Integration.

**V. REFERENCES**

[1]: Appelo, Daniel. "Homework 3." *Math 471*. UNM, n.d. Web. <http://math.unm.edu/~appelo/teaching/Math471F15/html/Homework3.html>.