**Unit 3: Numerical Differentiation and Integration**

**Numerical Differentiation**

Need for differentiation of a function arises quite often in engineering and scientific problems. If the function has a closed form representation in terms of standard calculus, then its derivatives can be found exactly. However, in many situations, we may not know the exact function. What we know is only the values of the function at a discrete set of points. For instance, we are given the distance travelled by a moving object at some regular time intervals and asked to determine its velocity at a particular time. In some other instances, the function is known but it is so complicated that an analytic differentiation is difficult. In both these situations, we seek the help of numerical techniques to obtain the estimates of function derivatives. The method of obtaining the derivative of a function using a numerical technique is known as numerical differentiation. There are essentially two situations where numerical differentiation is required.

1. The function values are known but the function is unknown. Such functions are called tabulated function
2. The function to be differentiated is complicated and therefore it is difficult to differentiate

In this unit we will discuss various numerical differentiation methods that could be applied to both tabulated and continuous functions

The derivative or differentiation of a function represents the rate of change of a variable with respect to another variable. For example, the velocity of a body is defined as the rate of change of location of the body with respect to time. The location is the dependent variable while time is independent variable. Now if we measure the rate of change of velocity with respect to time, we get the acceleration of the body. In this case, the velocity is the dependent variable while time is the independent variable. Numerical differentiation is the process of obtaining the value of the derivative of a function from a set of numerical values of that function.

**Differentiating Continuous Function**

Here we discuss the process of approximating the derivatives f’(x) of a function f(x), when the function itself is available. If the function becomes too complex, it is sometimes easier to differentiate numerically rather than analytically.

**Two Point Forward Difference Formula**

Taylor’s theorem says that if you know the value of a function f(x) at a point xi and all its derivatives at that point, provided the derivatives are continuous between xi and xi+1 then

f(xi+1) = f(xi)+f’(xi)(xi+1-xi)+ +….

Substituting for convenience h = xi+1-xi

f(xi+h) = f(xi)+f’(xi)++………..

or

f’(xi) = +E =

Here **E** is the error term in the approximation which is of the order of O(h). This equation is called **Two Point Forward Difference Formula**

**Algorithm**

1. Start
2. Read the value at which derivative is needed say x
3. Read interval gap, say h
4. Calculate f(xi) and f(xi+h)
5. Calculate d = f’(xi) = (f(xi+h)-f(xi)/h
6. Display the value derivative
7. Stop

**C Program for Two-Point Forward Difference Formula**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define PI 3.1416

#define f(x) sin(x)+1

int main()

{

float angle,h,x,d,x1,x2;

printf("Enter angle in degree\n");

scanf("%f",&angle);

printf("Enter increment\n");

scanf("%f",&h);

x = PI\*angle/180;

x1 = f(x+h);

x2 = f(x);

d = (x1-x2)/h;

printf("The value of derivative =%f\n",d);

getch();

return 0;

}

Output:

Enter angle in degree

45

Enter increment

0.1

The value of derivative =0.670602

**Example 1:** Find value of derivative at x=1 for the function f(x) = x2 by using h=0.2 and 0.05

Solution

For h=0.2

We know that,

f’(xi) =

Or f’(1) = = (1.44-1)/0.2 = 2.2

True value of derivative at x =1 is

f(x) = 2\*x = 2

Error = |(2-.2.2)/2|\*100 =10%

For h=0.05

We know that,

f’(xi) =

Or f’(1) = = (1.1025-1)/0.05 = 2.05

Error = |(2-.2.05)/2|\*100 =2.05%

Form this example; it is clear that error decreases as the value of h becomes smaller

Example 2 : Find the value of derivatives at x=450 for the function f(x) = sinx+1 by using h = 0.1 and 0.001.

Solution:

Angle in degree = 45

Angle in radian = (3.1416\*45)/180 = 0.788

We know that,

f’(xi) =

Or f’(450) = = (0.776-709)/0.1 = 0.67

True value of derivative at 450 is

F’(x) = cos x = 0.705

Error = |(0.705-0.67)/0.705| = 0.049 = 4.9%

For h=0.01

We know that,

f’(xi) =

Or f’(450) = = (0.716-709)/0.01 = 0.70

True value of derivative at 450 is

F’(x) = cos x = 0.705

Error = |(0.705-0.70)/0.705| = 0.007 = 0.7%

**Two Point Backward Difference Formula**

Taylor’s theorem says that if you know the value of a function f(x) at a point xi and all its derivatives at that point, provided the derivatives are continuous between xi and xi+1 then

f(xi+1) = f(xi)+f’(xi)(xi+1-xi)+ +….

Note that xi+1 is the point behind xi. Substituting for convenience h = xi+1-xi

f(xi-h) = f(xi)-f’(xi)h++………..

or

f’(xi) = +E =

Here **E** is the error term in the approximation which is of the order of O(h). This equation is called **Two Two Point Backward Difference Formula**

Algorithm for Two Point Backward Difference Formula

1. Start
2. Read the value at which derivative is needed say, x
3. Read interval gap, say h
4. Calculate f(xi) and f(xi-h)
5. Calculate d = f’(xi) = (f(xi)-f(xi-h)/h
6. Display the value of derivative
7. Stop

**C Program for Two Point Backward Difference Formula**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define PI 3.1416

#define f(x) sin(x)+1

int main()

{

float angle,h,x,d,x1,x2;

printf("Enter angle in degree\n");

scanf("%f",&angle);

printf("Enter increment\n");

scanf("%f",&h);

x = PI\*angle/180;

x1 = f(x-h);

x2 = f(x);

d = (x2-x1)/h;

printf("The value of derivative =%f\n",d);

getch();

return 0;

}

Output:

Enter angle in degree

45

Enter increment

0.1

The value of derivative =0.741253

**Example 1: Find value of derivative at x=1 for the function f(x) = x2 by using h=0.2 and 0.05**

**Solution**

For h=0.2

We know that,

f’(xi) =

Or f’(1) = = (1-0.64)/0.2 = 1.8

True value of derivative at x =1 is

f(x) = 2\*x = 2

Error = |(2-1.8)/2|\*100 =10%

For h=0.05

We know that,

f’(xi) =

Or f’(1) = = (1-0.9025)/0.05 = 1.95

Error = |(2-.1.95)/2|\*100 =2.5%

Form this example; it is clear that error decreases as the value of h becomes smaller

**Example 2: Find the value of derivatives at x=450 for the function f(x) = sinx+1 by using h = 0.1 and** 0.001.

**Solution:**

Angle in degree = 45

Angle in radian = (3.1416\*45)/180 = 0.788

We know that,

f’(xi) =

Or f’(450) = = (f(0.788)-f(0.688))/0.1 = (0.709-0.635)/0.1 = 0.74

True value of derivative at 450 is

F’(x) = cos x = 0.705

Error = |(0.705-0.74)/0.705| = 0.049 = 4.9%

For h=0.01

We know that,

f’(xi) =

Or f’(450) = = (f(0.788)-f(0.778)/0.01 = (0.709-0.702)/0.01 = 0.70

True value of derivative at 450 is

F’(x) = cos x = 0.705

Error = |(0.705-0.70)/0.705| = 0.007 = 0.7%

**Three Point Formula**

From Taylor series, we have

f(xi+h) = f(xi)+ f’(xi)h+ h2+……………. (1)

f(xi-h) = f(xi) -f’(xi)h+ h2+……………. (2)

Subtracting equation (2) from equation (1) we get

f(xi+h)-f(xi-h) = f’(xi)2h+ h2………………….

Or

F’(xi) = +E =

Hence, we have obtained a more accurate formula as the error is of order of O(h2). This equation is called **Three Point Formula**

**Algorithm for Three Point Formula**

1. Start
2. Read the value at which derivative is needed, say x
3. Read interval gap say h
4. Calculate f(xi) and f(xi+h)
5. Calculate d = f’(xi) = f(xi+h)-f(xi-h)/2h
6. Display the value of derivative
7. Stop

**C program for Three Point Formula**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define PI 3.1416

#define f(x) sin(x)+1

int main()

{

float angle,h,x,d,x1,x2;

printf("Enter angle in degree\n");

scanf("%f",&angle);

printf("Enter increment\n");

scanf("%f",&h);

x = PI\*angle/180;

x1 = f(x+h);

x2 = f(x-h);

d = (x1-x2)/(2\*h);

printf("The value of derivative =%f\n",d);

getch();

return 0;

}

Output:

Enter angle in degree

45

Enter increment

0.01

The value of derivative =0.707096

**Differentiating Discrete (Tabulated) Functions**

When the functional value is known at some points but function is not known, we can still find derivatives of such tabulated functions. In such cases first we have to interpolate the tabulated function and then differentiate it. Differentiation of tabulated functions suffers from conflict between round off errors (due to limited machine precision) and errors inherent in interpolation. For this reason, a derivative of a function can never be computed with the same precision as the function itself. Two situations exists here:

* If arguments are equally spaced, we will use Newton Gregory forward formula. If we desire to find the derivative of the function at a point near to beginning. If we desire to find the derivative of the function at a point near to end then we will use Newton Gregory formula. And if the derivative at a point is near the middle of the table we apply the central difference formula.
* In case the arguments are unequally spaced then we should use Newton’s divided difference formula.

**Derivative Using Newton’s Divided Difference Formula**

We know that, the general form of the Newton’s Divided difference polynomial for n+1 data points (x0,y0), (x1,y1),(x2,y2)…..(xn-1,yn-1), (xn,yn) is given by

pn(x) = f(x) = a0+a1(x-x0)+a2(x-x0)(x-x1)+….+an(x-x0)(x-x1)(x-x2)……..(x-xn-1)

Or pn(x) = f(x) = f[x0]+f[x1,x0](x-x0)+[x2,x1,x0](x-x0)(x-x1)+….+f[xn,xn-1…x0](x-x0)(x-x1)(x-x2)……..(x-xn-1)……………………………….(1)

Where the definition of the mth divided difference is

f[xm,….x0] =

Differentiating equation (1) with respect to x we get

f’(x) = f[x1,x0]+f[x2,x1,x0]{(x-x1)+(x-x0)}+f[x3,x2,x1,x0]{(x-x1)(x-x2)+(x-x0)(x-x2)+(x-x0)(x-x1)+……….(2)

Putting x=a in equation (2) we get value of first derivative at x=a. Again, differentiating equation (2) with respect to x we get

f’’(x) = 2f[x2,x1,x0]+2f[x3,x2,x1,x0]{(x-x0)+(x-x1)+(x-x2)}+………..(3)

Putting x=a in equation (3) we can get value of second derivative at x=a

**Algorithm**

1. Start
2. Read numbers of points ,n
3. Read n data points
4. Read the value at which derivative is needed say x
5. Compute the divided difference as below

For i=0 to n-1

dd[i] = fx[i]

End for

For i=0 to n-1

For n= n-1 to i+1

dd[j] =

End for

End for

1. Compute differentiated value as below

Set v = dd[1]

For i=2 to n-1

term =0;

for j=0 to i

factor = 1

for k=0 to i

if(j!=k) factor = factor \*(x-x[k])

term = term+factor

end for

vod = vod+dd[i]

end for

1. Print the value of first derivative which is value of variable v
2. Stop

**C Program for computing differentiation using divided difference polynomial**

#include<stdio.h>

#include<conio.h>

int main()

{

int n,i,k,j;

float factor, term, vod, xv,x[10],fx[10],a[10];

printf("Enter the number of points\n");

scanf("%d",&n);

printf("Enter value of data points\n");

for(i=0;i<n;i++)

{

scanf("%f%f",&x[i],&fx[i]);

}

printf("Enter the value at which derivative is required\n");

scanf("%f",&xv);

for(i=0;i<n;i++)

{

a[i] = fx[i];

}

for(i=0;i<n;i++)

{

for(j=n-1;j>i;j--)

{

a[j] = (a[j]-a[j-1])/(x[j]-x[j-1-i]);

}

}

vod = a[1];

for(i=2;i<n;i++)

{

term = 0;

for(j=0;j<i;j++)

{

factor = 1;

for(k=0;k<i;k++)

{

if(k!=j)

factor = factor\*(xv-x[k]);

}

term = term+factor;

}

vod = vod+(a[i]\*term);

}

printf("Value of first derivative =%f\n",vod);

}

Output:

Enter the number of points

5

Enter value of data points

3 -13

5 23

11 899

27 17315

34 35606

Enter the value at which derivative is required

10

Value of first derivative =233.000000

**Example: Find f’(10) from the following data points.**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 3 | 5 | 11 | 27 | 34 |
| f(x) | -13 | 23 | 899 | 17315 | 35606 |

**Solution:**

In this case the values of x are not equally spaced. So we shall use Newton’s divided difference formula. The divided difference table is given below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | f(X) | First Divided Difference | Second Divided Difference | Third Divided Difference | Fourth Divided Difference |
| 3 | -13 |  |  |  |  |
|  |  | (23-(-13)/(5-3)=18 |  |  |  |
| 5 | 23 |  | (146-18)/(11-3) = 16 |  |  |
|  |  | (899-23)/(11-5)=146 |  | (40-16)/(27-3) = 1 |  |
| 11 | 899 |  | (1026-146)/(27-5) =40 |  | (1-1)/(24-3)=0 |
|  |  | (17315-899)/(27-11)=1026 |  | (69-49)/(34-5) = 1 |  |
| 27 | 17315 |  | (2613-1026)/(34-11) = 69 |  |  |
|  |  | (35606-17315  )/(34-27)=2613 |  |  |  |
| 34 | 35606 |  |  |  |  |

We know that,

f’(x) = f[x1,x0]+f[x2,x1,x0]{(x-x1)+(x-x0)}+f[x3,x2,x1,x0]{(x-x1)(x-x2)+(x-x0)(x-x2)+(x-x0)(x-x1)+………

At x=10

f’(10) = 18+16{(10-5))+(10-3)}+1{(10-5)(10-11)+(10-3)(10-11)+(10-3)(10-5)}

= 18+16\*12+1\*(-5-7+35}

=18+192+23 = 233

**Example 2:** A slider in a machine moves along a fixed straight rod. Its distance covered by the machine is given below for various values of the time. Find the velocity of the slider when at x=0.3 seconds.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Time (x) | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
| Distance f(x) | 31.62 | 32.87 | 33.64 | 33.95 | 33.81 |

**Solution:**

Since we have to find the velocity at t-0.3 which lies around center of data points, we shall use Newton’s divided difference formula. The divided difference table is given below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | f(X) | First Divided Difference | Second Divided Difference | Third Divided Difference | Fourth Divided Difference |
| 0.1 | 31.62 |  |  |  |  |
|  |  | 12.5 |  |  |  |
| 0.2 | 32.87 |  | -24 |  |  |
|  |  | 7.7 |  | 3.3333 |  |
| 0.3 | 33.64 |  | -23 |  | 0.00 |
|  |  | 3.1 |  | 3.3333 |  |
| 0.4 | 33.95 |  | -22 |  |  |
|  |  | -1.3 |  |  |  |
| 0.5 | 33.81 |  |  |  |  |

We know that,

f’(x) = f[x1,x0]+f[x2,x1,x0]{(x-x1)+(x-x0)}+f[x3,x2,x1,x0]{(x-x1)(x-x2)+(x-x0)(x-x2)+(x-x0)(x-x1)+………

At x=0.3

f’(0.3) = 12.5-24\*{(0.3-0.2)+(0.3-0.1)}+3.3333\*{(0.3-0.2)\*(0.3-0.3)+(0.3-0.1)+(0.3-0.3)+(0.3-0.1)\*(0.3-0.2)}

F’(0.3) = 12.5-7.2+0.6666 = 5.97cm/sec

**Derivatives Using Newton’s Forward Difference Formula**

Newton’s forward difference formula for n+1 data points, (x0,f(x0)), (y1, f(y1))….(xn, f(xn)) can be written as

f(x) = f(x0)+ s ∆ f(x0) + s(s-1) ∆2 f(x0)+ s(s-1)(s-2) ∆3 f(x0)+……………….(1)

where

s = (x-x0)/h

Differentiating equation (1) with respect to x, we get

f’(x) = {∆ f(x0)+ (2s-1) ∆2 f(x0)+ (3s2-6s+2) ∆3 f(x0)+ (4s3-18s2+22s-6) ∆4 f(x0)+……}………………(2)

since

s = (x-x0)/h🡪 =

Thus equation (2) becomes

f’(x) = {∆ f(x0)+ (2s-1) ∆2 f(x0)+ (3s2-6s+2) ∆3 f(x0)+ (4s3-18s2+22s-6) ∆4 f(x0)+……}…………….(3)

By putting x=a in equation (3) can be used to find the value of first derivative at the point x=a

Again differentiating equation (3) with respect with x we get

f’’(x) = { ∆2 f(x0)+ (6s-6) ∆3 f(x0)+ (12s2-36s+22) ∆4 f(x0)+……} ………………..(4)

Putting x=a in equation (4) we will get the value of second derivative at the point x=a

**Example 1: Find the first and second derivatives of the functions tabulated at below at 1.1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | 1.0 | 1.2 | 1.4 | 1.6 | 1.8 | 2.0 |
| f(X) | 0.0 | 0.128 | 0.544 | 1.296 | 2.432 | 4.000 |

**Solution:**

Since x=1.1 lies near the beginning of the table therefore in this case we shall use Newton’s Gregory forward formula. The difference table can be constructed as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | F(x) | First Divided Difference | Second Divided Difference | Third Divided Difference | Fourth Divided Difference | Fifth Divided Difference |
| 1.0 | 0.0 |  |  |  |  |  |
|  |  | 0.128 |  |  |  |  |
| 1.2 | 0.128 |  | 0.288 |  |  |  |
|  |  | 0.416 |  | 0.048 |  |  |
| 1.4 | 0.544 |  | 0.336 |  | 0 |  |
|  |  | 0.732 |  | 0.048 |  | 0 |
| 1.6 | 1.296 |  | 0.384 |  | 0 |  |
|  |  | 1.136 |  | 0.048 |  |  |
| 1.8 | 2.432 |  | 0.432 |  |  |  |
|  |  | 1.568 |  |  |  |  |
| 2.0 | 4.0 |  |  |  |  |  |

We know that

f’(x) = {∆ f(x0)+ (2s-1) ∆2 f(x0)+ (3s2-6s+2) ∆3 f(x0)+ (4s3-18s2+22s-6) ∆4 f(x0)+……}

Here,

H=0.2 and s = (x-x0)/h = (1.1-1.0)/0.2 = 0.5

Thus,

f’(x) = {0.128+ (2\*0.5-1) \*0.288+ (3\*0.52-6\*0.5+2)\*0.048)+……} = 0.63

Again, since

f’’(x) = { ∆2 f(x0)+ (6s-6) ∆3 f(x0)+ (12s2-36s+22) ∆4 f(x0)+……}

f’’(x) = { 0.288+ (6\*.05-6) \*0.048+ (12\*0.52-36\*0.5+22) \*0+……}

=6.6

**Example 2:** Find the first derivative of the function tabulated at below at 1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 |
| f(X) | 7.989 | 8.403 | 8.781 | 9.129 | 9.451 |

Solution:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | F(x) | First Divided Difference | Second Divided Difference | Third Divided Difference | Fourth Divided Difference |
| 1.0 | 7.989 |  |  |  |  |
|  |  | 0.414 |  |  |  |
| 1.1 | 8.403 |  | -0.036 |  |  |
|  |  | 0.378 |  | 0.006 |  |
| 1.2 | 8.781 |  | -0.03 |  | -0.002 |
|  |  | 0.348 |  | 0.004 |  |
| 1.3 | 9.129 |  | -0.026 |  |  |
|  |  | 0.322 |  |  |  |
| 1.4 | 9.451 |  |  |  |  |

We know that

f’(x) = {∆ f(x0)+ (2s-1) ∆2 f(x0)+ (3s2-6s+2) ∆3 f(x0)+ (4s3-18s2+22s-6) ∆4 f(x0)+……}

Here,

H=0.1 and s = (x-x0)/h = (1.1-1.0)/0.1 = 1

Thus,

f’(x) = {0.414+ (2\*1-1) \*(-0036)+ (3\*12-6\*1+2)\*0.006)+ (4\*13-18\*12+22\*1-6)\*(-0.002)= 3.95

**Algorithm**

1. Start
2. Read numbers of points ,n
3. Read the value at which derivative is needed say xp
4. Read n data points say, x[i] and fx[i]
5. Set h = x[i]-x[0] and s = (xp-x[0])/h
6. Compute the forward difference as below

For i=0 to n-1

fd[i] = fx[i]

End for

For i=0 to n-1

For n= n-1 to i+1

fd[j] = fd[j]-fd[j-1]

End for

End for

1. Compute differentiated value as below

Val = fd[1]

prevterm = 1

for i=2 to n-1

term1 = 1

for k=2 to i

term1 = term1 \*(s-k+2);

end for

term2 = term2 +(s-i+1)\*prevterm

prevterm = term1+term2

val = val+(fd[i]\*(term1+term2))/i!;

End for

1. Print the value of first derivative (val)
2. Stop

**C Program for calculating value of derivative using forward difference**

#include<stdio.h>

#include<conio.h>

int fact(int n)

{

if(n==1)

return 1;

else

return n\*fact(n-1);

}

int main()

{

int n,i,j,k;

float val=0, p,xp,x[10],fx[10],fd[10],h,s,term1,term2, prev;

printf("Enter the number of points\n");

scanf("%d",&n);

printf("Enter value of x and fx\n");

for(i=0;i<n;i++)

{

scanf("%f%f",&x[i],&fx[i]);

}

printf("Enter the value at which derivative is needed\n");

scanf("%f",&xp);

h = x[1]-x[0];

s = (xp-x[0])/h;

for(i=0;i<n;i++)

{

fd[i]=fx[i];

}

for(i=0;i<n;i++)

{

for(j=n-1;j>i;j--)

{

fd[j] = (fd[j]-fd[j-1]);

}

}

val = fd[1];

prev = 1;

for(i=2;i<n;i++)

{

term1 = 1;

for(k=2;k<=i;k++)

{

term1 = term1\*(s-k+2);

}

term2 = (s-i+1)\*prev;

prev = (term1+term2);

val = val +(fd[i]\*(term1+term2))/(fact(i));

}

val = val/h;

printf("Value of first derivative = %f",val);

getch();

return 0;

}

Output:

Enter the number of points

7

Enter value of x and fx

1.0 2.71

1.2 3.32

1.4 4.05

1.6 4.95

1.8 6.04

2.0 7.38

2.2 9.02

Enter the value at which derivative is needed

1.2

Value of first derivative = 3.258337

**Derivatives Using Newton’s Backward Difference Formula**

Newton’s backward difference formula for n+1 data points (x,f(x0)), (x1,f(x1)),…..(xn,f(xn)) can be written as

f(x) = f(xn)+ s ∇ f(xn) + s(s+1) ∇2f(xn)+ s(s+1)(s+2) ∇3f(xn) )+……………….(1)

where

s = (x-xn)/h

Differentiating equation (1) with respect to x, we get

f’(x) = {∇f(xn))+ (2s+1) ∇2f(xn)+ (3s2+6s+2)∇3f(xn)+……}………………(2)

since

s = (x-x0)/h🡪 =

Thus equation (2) becomes

f’(x) = {∇f(xn))+ (2s+1) ∇2f(xn)+ (3s2+6s+2)∇3f(xn)+……}…………….(3)

By putting x=a in equation (3) can be used to find the value of first derivative at the point x=a

Again differentiating equation (3) with respect with x we get

f’’(x) = { ∇2f(xn)+ (6s+6) ∇3f(xn)+ (12s2+36s+22) ∇4f(xn)+……} ………………..(4)

Putting x=a in equation (4) we will get the value of second derivative at the point x=a

**Algorithm**

1. Start
2. Read numbers of points ,n
3. Read the value at which derivative is needed say xp
4. Read n data points say, x[i] and fx[i]
5. Set h = x[i]-x[0] and s = (xp-x[n-1])/h
6. Compute the backward difference as below

For i=0 to n-1

bd[i] = fx[i]

End for

For i=n-1 to 1

For j=0 to i-1

bd[j] = d[j+1j]-fd[j]

End for

End for

1. Compute differentiated value as below

val = bd[n-2]

prevterm = 1

for i=2 to n-1

term1 = 1

for k=2 to i

term1 = term1 \*(s+k-2);

end for

term2 = term2 +(s+i-1)\*prevterm

prevterm = term1+term2

val = val+(bd[n-i-1]\*(term1+term2))/i!;

End for

1. Print the value of first derivative (val)
2. Stop

………………

C Program for calculating derivative using backward divided differences

#include<stdio.h>

#include<conio.h>

int fact(int n)

{

if(n==1)

return 1;

else

return n\*fact(n-1);

}

int main()

{

int n,i,j,k;

float val=0, p,xp,x[10],fx[10],bd[10],h,s,term1,term2, prev;

printf("Enter the number of points\n");

scanf("%d",&n);

printf("Enter value of x and fx\n");

for(i=0;i<n;i++)

{

scanf("%f%f",&x[i],&fx[i]);

}

printf("Enter the value at which derivative is needed\n");

scanf("%f",&xp);

h = x[1]-x[0];

s = (xp-x[n-1])/h;

for(i=0;i<n;i++)

{

bd[i]=fx[i];

}

for(i=n-1;i>0;i--)

{

for(j=0;j<i;j++)

{

bd[j] = (bd[j+1]-bd[j]);

}

}

val = bd[n-2];

prev = 1;

for(i=2;i<n;i++)

{

term1 = 1;

for(k=2;k<=i;k++)

{

term1 = term1\*(s+k-2);

}

term2 = (s+i-1)\*prev;

prev = (term1+term2);

val = val +(bd[n-i-1]\*(term1+term2))/(fact(i));

}

val = val/h;

printf("Value of first derivative = %f",val);

getch();

return 0;

}

Output:

Enter the number of points

7

Enter value of x and fx

1.0 2.71

1.2 3.32

1.4 4.05

1.6 4.95

1.8 6.04

2.0 7.38

2.2 9.02

Enter the value at which derivative is needed

2.2

Value of first derivative = 8.870843

Example 1: Find the first and second derivatives of the functions tabulated below at x=9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 5 | 6 | 7 | 8 | 9 |
| f(x) | 10.0 | 14.5 | 19.5 | 25.5 | 32.0 |

Solution:

Since x=9 lies at the end of the table therefore in this case must use Newton Gregory backward formula. The difference table is as below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | F(x) | First Divided Difference | Second Divided Difference | Third Divided Difference | Fourth Divided Difference |
| 5 | 10 |  |  |  |  |
|  |  | 4.5 |  |  |  |
| 6 | 14.5 |  | 0.5 |  |  |
|  |  | 5.0 |  | 0.5 |  |
| 7 | 19.5 |  | 1.0 |  | -1.0 |
|  |  | 6.0 |  | -0.5 |  |
| 8 | 25.5 |  | 0.5 |  |  |
|  |  | 6.5 |  |  |  |
| 9 | 32.0 |  |  |  |  |

We know that

f’(x) = {∇f(xn))+ (2s+1) ∇2f(xn)+ (3s2+6s+2)∇3f(xn)+ (4s3+18s2+22s+6) ∇4f(xn)……}

Here,

h = 1.0, and s = (s-xn)/h = (9-9)/1.0=0

Thus,

f’(9) = {6.5+ (2\*0+1) \*0.5+ (3\*02+6\*0+2)\*(-0.5)+ (4\*03+18\*02+22\*0+6)\*(-1)}

=6.334

Again, since

f’’(x) = { ∇2f(xn)+ (6s+6) ∇3f(xn)+ (12s2+36s+22) ∇4f(xn)}

f’’(9) = { 0.5+ (6\*0+6) \*(-0.5)+ (12\*02+36\*0+22) \*(-1)} = -0.9166

Example 2: Find the first derivative of the function tabulated below at 1.4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 1.0 | 1.1 | 1.2 | 1.3 | 1.4 |
| F(x) | 7.989 | 8.403 | 8.781 | 9.129 | 9.451 |

Solution: Since x=1.5 lies at the end of the table therefore in this case we shall use Newton Gregory backward formula. The difference table is as below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | F(x) | First Divided Difference | Second Divided Difference | Third Divided Difference | Fourth Divided Difference |
| 1.0 | 7.989 |  |  |  |  |
|  |  | 0.414 |  |  |  |
| 1.1 | 8.403 |  | -0.036 |  |  |
|  |  | 0.378 |  | 0.006 |  |
| 1.2 | 8.781 |  | -0.03 |  | -0.002 |
|  |  | 0.348 |  | 0.004 |  |
| 1.3 | 9.129 |  | -0.026 |  |  |
|  |  | 0.322 |  |  |  |
| 1,4 | 9.451 |  |  |  |  |

We know that

f’(x) = {∇f(xn))+ (2s+1) ∇2f(xn)+ (3s2+6s+2)∇3f(xn)+ (4s3+18s2+22s+6) ∇4f(xn)……}

Here,

h = 1.0, and s = (x-xn)/h = (1.5-1.4)/0.1=1

Thus,

f’(9) = {0.322+ (2\*1+1) \*(-0.026)+ (3\*12+6\*1+2)\*(0.004)+ (4\*13+18\*12+22\*1+6)\*(-0.00)}

=0.898

**Maxima and Minima of Tabulated Functions**

We know that maximum and minimum values of a function can be computed by equating first derivative of the function to zero and solving for unknown variables. The same concept can be applied to determine the maxima and minima of tabulated functions

We know that, Newton’s forward difference formula is given by

f(x) = f(x0)+ s ∆ f(x0) + s(s-1) ∆2 f(x0)+ s(s-1)(s-2) ∆3 f(x0)+……………….(1)

where

s = (x-x0)/h

Differentiating equation (1) with respect to x, we get

f’(x) = {∆ f(x0)+ (2s-1) ∆2 f(x0)+ (3s2-6s+2) ∆3 f(x0)+…..}………………(2)

For maxima and minima f’(x0) must be zero. Terminating the terms after third order difference in RHS and equating it to zero, we get

∆ f(x0)+ (2s-1) ∆2 f(x0)+ (3s2-6s+2) ∆3 f(x0)=0

Or {∆3 f(x0)}s2+{∆2f(x0) - ∆3 f(x0)}s + {∆f(x0)- ∆2 f(x0)+∆3 f(x0)} =0 ----------(3)

Equation can be written in the form

as2+bs+c = 0

Where

a = ∆3 f(x0)

b = ∆2f(x0) - ∆3 f(x0)

c = ∆f(x0)- ∆2 f(x0)+∆3 f(x0)

Equation (3) is quadratic in ‘s’ and can be solved. Then the values of x can be computed from relation x = x0+sh

Example: Find the maximum and minimum values of the unction tabulated below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 0 | 1 | 2 | 3 |
| f(x) | -5 | -7 | -3 | 13 |

**Solution:**

The forward difference table is calculated as below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| x | F(x) | First Divided Difference | Second Divided Difference | Third Divided Difference |
| 0 | **-5** |  |  |  |
|  |  | **-2** |  |  |
| 1 | -7 |  | **6** |  |
|  |  | 4 |  | **6** |
| 2 | -3 |  | 12 |  |
|  |  | 16 |  |  |
| 3 | 13 |  |  |  |

We know that

as2+bs+c = 0………………….(1)

Where

a = ∆3 f(x0)

b = ∆2f(x0) - ∆3 f(x0)

c = ∆f(x0)- ∆2 f(x0)+∆3 f(x0)

Now,

a = ∆3 f(x0) = ½\*6 = 3

b = ∆2f(x0) - ∆3 f(x0) = 6-6=0

c = ∆f(x0)- ∆2 f(x0)+∆3 f(x0) = -2-1/2\*6+1/3\*6 = -3

Thus equation (1) becomes

as2+bs+c = 0

3s2-3=0

Or s = ±1

Again,

x = x0+sh

Here, x0=0 and h=1

So, x = ±1

This implies s = x

Again putting s=x in Newton’s forward difference formula

f(x) = (-5)+x(-2)+ x(x-1)6+(x-1)(x-2).6 =

Or f(x) = x3-3x-5

f’(x) = 3x2-3

f’’(x) = 6x

Therefore, we have maxima at x = -1 and minima at x=1

Algorithm

1. Start
2. Read number of data points, say n
3. Read n data points, say x[i] and fx[i]
4. Set h = x[1]-x[0]
5. Compute forward differences as below

For i=0 to n-1

Fd[i] = fx[i]

End for

For i=0 to n-1

For j=n-1 to n-1

For j=n-1 to i+1

Fd[j] = fd[j]-fd[j-1]

End for

End for

1. Calculate

a = (1/2.0)\*fd[3]

b = fd[2]-fd[3]

c = fd[1]-((1/2.0)\*fd[2])\*fd[2])+(1/3.0)\*fd[3])

1. Calculate

s1 = (-b+sqrt(b\*b-4\*a\*c))/(2\*a);

s2 = (-b-sqrt(b\*b-4\*a\*c))/(2\*a);

1. Calculate x1 = x[0]+s1\*h

X2 = x[0]+s2\*h

1. Find the point of maxima and minima as below

val = (fd[2]+(((6\*s1-6)\*fd[3])/6))/(h\*h);

if(val<0)

printf("Maxima exists at x =%f\n",x1);

else

print "Minima exists at x1”

val = (fd[2]+(((6\*s2-6)\*fd[3])/6))/(h\*h);

if(val<0)

print "Maxima exists at x2;

else

print "Minima exists at x2”

1. Stop

C program

#include<conio.h>

#include<stdio.h>

#include<math.h>

int main()

{

int n,i,j;

float val, x[10],fx[10],fd[10],h,s1,s2,x1,x2,a,b,c;

printf("Enter the number of poinnts\n");

scanf("%d",&n);

printf("Enter values of x and fx\n");

for(i=0;i<n;i++)

{

scanf("%f%f",&x[i],&fx[i]);

}

h = x[1]-x[0];

for(i=0;i<n;i++)

{

fd[i]=fx[i];

}

for(i=0;i<n;i++)

{

for(j=n-1;j>i;j--)

{

fd[j]=(fd[j]-fd[j-1]);

}

}

a = (1/2.0)\*fd[3];

c = fd[1]-((1/2.0)\*fd[2])+((1/3.0)\*fd[3]);

b = fd[2]-fd[3];

s1 = (-b+sqrt(b\*b-4\*a\*c))/(2\*a);

s2 = (-b-sqrt(b\*b-4\*a\*c))/(2\*a);

x1 = x[0]+s1\*h;

x2 = x[0]+s2\*h;

val = (fd[2]+(((6\*s1-6)\*fd[3])/6))/(h\*h);

if(val<0)

printf("Maxima exists at x =%f\n",x1);

else

printf("Minima exists at x =%f\n",x1);

val = (fd[2]+(((6\*s2-6)\*fd[3])/6))/(h\*h);

if(val<0)

printf("Maxima exists at x =%f\n",x2);

else

printf("Minima exists at x =%f\n",x2);

getch();

return 0;

}

Output:

Enter the number of points

4

Enter values of x and fx

0 -5

1 -7

2 -3

3 13

Minima exists at x =1.000000

Maxima exists at x =-1.000000

**Numerical Integration**

**Introduction**

Numerical integration is the process of computing the approximate value of a definite integral using a set of numerical values of the integrand. Integration is the process of measuring the area under a function plotted on a graph. It is represented by

Where the symbol is an integral sign and a and b are the lower and upper limits of the integration respectively. The function f is the integrand of the integral and x is the variable of integration

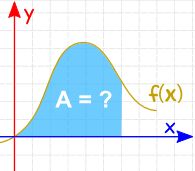


Figure: The definite integral as the area of a region under the curve, Area =

Why we integrate a function. Among the most common examples are finding the velocity of a body from an acceleration function and displacement of a body from a velocity function. Throughout many science and engineering fields, there are countless applications for integral calculus. Sometimes, the evaluation of expressions involving these integrals can become daunting, if not intermediate. For this reason, a wide variety of numerical methods has been developed to simplify the integral. Methods of numerical integration can be divided into two grouped. Newton Cotes formulas and Gaussian quadrature. Newton Cotes formulae are characterized by equally spaced abscissa and include well known methods such as the trapezoidal rule and Simpson’s rule. They are most useful if f(x) has already been computed at equal intervals or can be computed at low cost. Since, Newton-Cotes formulas are based on local interpolation, they require only a piecewise fit to a polynomial. In Gaussian quadrature, the locoseations of the abscissas are chosen to yield the best possible accuracy. Since Gaussian quadrature require fewer evaluations of the integrand for a given level of precision, it is popular in cases where f(x) is expensive to evaluate.

**Newton Cotes Integration Formula**

This is the most common numerical integration schemes. The strategy is to replace a complicated function or tabulated data with simpler polynomial function for easy integration. Newton-Cotes Integration comes in following two forms:

* **Close Form:** If the limits of integration a and b are in the set of interpolating points then the formula is referred to as closed forms
* **Open Form:** If the limits of integration a and b lie beyond the set of interpolating points then the formula is referred to as open form

Since open form is not used for definite integration. In this topic we only discuss about closed form of methods. Some of the closed methods are:

1. Trapezoidal Rule
2. Simpson’s 1/3 Rule
3. Simpson’s 3/8 Rule

All of above rules can be formulated by using interpolation polynomial for approximating the function f(x).

A general Quadrature Formula for Equally Spaced Arguments

Let us suppose we have to evaluate …………….(1)

Let us divide the interval (x0,xn) into n sub intervals of equal width so that h =

Thus, x1 = x0+h, x2 = x0+2h, x3 = x0+3h…………..xn = x0+nh

From Newton’s Gregory forward difference formula, we know that

f(x) = f(x0)+ s ∆ f(x0) + s(s-1) ∆2 f(x0)+ s(s-1)(s-2) ∆3 f(x0)+……………….(2)

where

s = (x-x0)/h

Now equation (1) can be written as:

= f(x0)+ s ∆ f(x0) + s(s-1) ∆2 f(x0)+ s(s-1)(s-2) ∆3 f(x0)+…}dx

Since x = x0+sh🡪dx = hds

= h f(x0)+ s ∆ f(x0) + s(s-1) ∆2 f(x0)+ s(s-1)(s-2) ∆3 f(x0)+…}ds

Or = h [ s.f(x0)+ ∆ f(x0) + [ - ] ∆2 f(x0)+ [ –s3+s2]∆3 f(x0)+…}]0n

= h [nf(x0)+ ∆ f(x0) + [ - ] ∆2 f(x0)+ [ –n3+n2]∆3 f(x0)+…}]

= nh[f(x0)+ ∆ f(x0) + (2n2-3n)∆2 f(x0)+ [n3-4n2+4n]]∆3 f(x0)+…}]

This equation is called general quadrature formula. From this formula we can obtain different integration by putting n=1, 2, 3,.. etc.

**Trapezoidal Rule**

The trapezoidal rule is based on the Newton Cote formula. The trapezoidal rule works by

Approximating the region under the graph of the function f(x) as a trapezoid and calculates its area. The trapezoidal rule assumes that n=1. That is, it approximates the integral by a linear polynomial (straight line). General quadrature formula for integration is given by

= nh[f(x0)+ ∆ f(x0) + (2n2-3n)∆2 f(x0)+ [n3-4n2+4n]]∆3 f(x0)+…}]……………(2)

By putting n=1 in the above relation and neglecting higher order forward difference, it can be written as

= h[f(x0)+ ∆ f(x0)] = h{f(x0)+ ∆ f(x0)] = h[f(x0)+ (f(x1)-f(x0)) = (f(x0)+fx(x1)

🡪 = (f(x0)+fx(x1)……………(2)

Equation (2) is called trapezoidal rule and it is the area of the trapezoid whose width is (x1-x0) and height is the average of f(x0) and f(x1)

f(x0)

f(x0)

x0

x1

Error

Figure: Geometric representation of trapezoidal rule

**Algorithm for trapezoidal rule**

1. Start
2. Read value of lower and upper limit say x0 and x1
3. Calculate f(x0) and f(x1)
4. Calculate h = (x1-x0)
5. Calculate the value of integration by using formula

v = (f(x0)+fx(x1)

1. Display the value of integration “v”
2. Stop

**C program for trapezoidal rule**

#include<stdio.h>

#include<conio.h>

#define F(x) (x\*x\*x+3)

int main()

{

float h, x0,x1,x2,x,v,f0,f1;

printf("Enter upper limit and lower limit\n");

scanf("%f%f",&x1,&x0);

h = x1-x0;

f1 = F(x1);

f0 = F(x0);

v = h\*(f0+f1)/2;

printf("The value of integration = %f",v);

getch();

return 0;

}

**Output:**

Enter upper limit and lower limit

8 2

The value of integration = 1578.000000

**Example 1:** Find (x3+2) dx by using trapezoidal rule

**Solution:**

Here x0 = 2 and x1= 8

So, h= x1-x0= 8-2=6

From Two point trapezoidal rule, we know that

= h[f(x0)+f(x1)]/2 = 6[10+514]/2 = 1572

**Example 2** Find (e-x2) dx by using trapezoidal rule

Solution:

Here x0 = 0 and x1= 1

So, h= x1-x0= 1-0=1

From Two point trapezoidal rule, we know that

= 1\*[f(x0)+f(x1)]/2 = 6[1+0.368]/2 = 0.684

**Composite Trapezoidal Rule**

In order to improve the accuracy of the trapezoidal rule, the integration interval can be divided into k segments of equal width. The equations are called the multiple segments or composite integration formulae. Divide (xn-x0) into k equal segments then the width of each segment is

h =

Now, the integral can be broken into k intervals as

= + +………++ …(1)

Applying the trapezoidal rule on each segment, equation (1) becomes

{(x0+h)-x0){ } + {(x0+2h)-(x0+h)}{}+……+{(xn-(x0+(k-1)h)}{ }

= {f(x0)+f(x0+h)+f(x0+h)+f(x0+2h)+…..+f(x0+(k-1)h)+f(xn)}

= {f(x0)+2{+f(xn)} ………( 2)

The equations (2) is called composite trapezoidal formula

**Example 1:** ex dx for k=2 and k=4 by using composite trapezoidal rule

Solution:

For k=2

Here x0 =1, xn = 1

So h = (xn-x0)/k = (1-(-1))/2 = 1

Now,

ex dx = {f(x0)+2{+f(xn)}

= 1/2[e-1+2\*e0+e1) = 2.54

For k=4

Here x0 =1, xn = 1

So h = (xn-x0)/k = (1-(-1))/4 = 0.5

Now,

ex dx = {f(x0)+2{++f(xn)}

= 0.5/2[e-1+2\*e-0.5+2\*e0+2\*e0.5+e1) = 2.399

Example 2: Compute the integral dx for k=4 and for k=8 by using composite trapezoidal rule

Solution

Here x0 = 1, xn = 5

For k=4

h = (xn-x0) /k = (5-1)/4 = 1

Now,

dx = [f(x0) +2{+f(xn)]

= [1.414 +2{2.236+3.162+4.123}+5.099)] = 12.78

For k=8

h = (xn-x0) /k = (5-1)/8 = 0.5

Now,

dx = [f(x0) +2{+f(xn)]

= [1.414 +2{1.803+2.236+2.696+3.162+6.64+4.123+4.609}+5.099)] = 12.76

**Algorithm for composite trapezoidal rule**

1. Start
2. Read value upper limit xn and lower limit x0
3. Read number of segments say k
4. Compute h = (xn-x0)/k
5. Computer result = f(x0)+f(xn)
6. For i=1 to k-1

result = result+ 2\*f(x0+i\*h)

1. Stop

**C program to composite trapezoidal rule**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) 3\*x\*x+2\*x-5

int main()

{

float x0,xn,f0,fn,result,h,a,r;

int i,k;

printf("Enter lower limit\n");

scanf("%f",&x0);

printf("Enter upper limit\n");

scanf("%f",&xn);

printf("Enter value of k\n");

scanf("%d",&k);

h = (xn-x0)/k;

f0 = f(x0);

fn = f(xn);

result = f0+fn;

for(i=1;i<=k-1;i++)

{

a = x0+i\*h;

result = result+2\*(f(a));

}

result = result\*h/2;

printf("The value of integration =%f\n",result);

getch();

return 0;

}

Output:

First Run

Enter lower limit

0

Enter upper limit

2

Enter value of k

2

The value of integration =3.000000

Second Run

Enter lower limit

0

Enter upper limit

2

Enter value of k

8

The value of integration =2.062500

**Simpson’s 1/3 Rule**

Trapezoidal rule is based on approximating the integrands by a first order polynomials and then integrating the polynomial over interval of integration. Simpson’s 1/3 rule is an extension of Trapezoidal rule where the integrand is approximated by a second order polynomial. The Simpson’s 1/3 rule assumes n=2. General quadrature formula for integration is given by

= = nh[f(x0)+ ∆ f(x0) + (2n2-3n)∆2 f(x0)+ [n3-4n2+4n]]∆3 f(x0)+…}]………(1)

Here h =

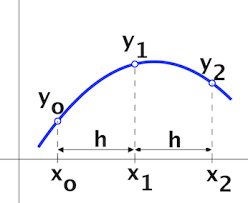
By putting n=2 in above relation and neglecting higher order forward difference, it can be written as:

= = 2h[f(x0)+ ∆ f(x0) + (2.22-3\*2)∆2 f(x0)]

= h[f(x0)+f(x1)-f(x0)+ (f(x0)-(f(x2)-2(f(x0)+f(x0))]

=[f(x0)+4f(x1)+f(x2)]….(2)

This equation (2) is called Simpson’s 1/3 rule



**Figure:** Geometric interpretation of Simpson’s 1/3 rule

**Example 1: Apply Simpson’ 1/3 rule to calculate dx**

**Solution:**

Here x0 =0, x1 = 1, h = = 0.5

Simpson’s 1/3 rule is given by

I = [f(x0)+4f(x1)+f(x2)]

Here, f(x) =

Therefore,

f(x0) = f(0) = 1

f(x1) = f(x0+h) = f(0.5) = 0.866

f(x2) = f(x0+2h) = f(1) = 0

Thus,

I = [f(x0)+4f(x1)+f(x2)] = [1+4\*0.866+0] = 0.744

**Example 2: Apply Simpson’ 1/3 rule to calculate sinx dx**

**Solution:**

Here x0 =0, x1 = 1, h = = π/2

Simpson’s 1/3 rule is given by

I = [f(x0)+4f(x1)+f(x2)]

Here, f(x) =sinx

Therefore,

f(x0) = f(0) = 0

f(x1) = f(x0+h) = f π/2 ) = sin π/2 = 1

f(x2) = f(x0+2h) = f(π) = sin π = 0

Thus,

I = [f(x0)+4f(x1)+f(x2)] = [0+1+0] = 2.101

**Algorithm for Simpson’s 1/3 rule**

1. Start
2. Read upper limit xn and lower limit x0
3. Set n=2
4. Compute h = (xn-x0)/n
5. Compute x1 = x0+h
6. Calculate I = \*[f(x0)+4\*f(x1)+f(x2)]
7. Print “I”
8. Stop

**C program for Simpson’s 1/3 rule**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) 3\*x\*x+2\*x-5

int main()

{

float x0,x1,x2,f0,f2,f1,i,h,a,r;

printf("Enter lower limit\n");

scanf ("%f",&x0);

printf ("Enter upper limit\n");

scanf ("%f",&x2);

h = (x2-x0)/2;

x1 = x0+h;

f0 = f(x0);

f1 = f(x1);

f2 = f(x2);

i = (f0+4\*f1+f2)\*h/3;

printf("The value of integration =%f\n",i);

getch();

return 0;

}

Output:

Enter lower limit

0

Enter upper limit

2

The value of integration =2.000000

**Composite Simpson’s 1/3 rule**

**It is also called multi segment Simpson’s 1/3 rule. Just line in multiple segment trapezoidal rule, one can subdivide the interval [a,b] into k segments and apply Simpson’s 1/3 rule repeatedly over every two segments. Note that k needs to be even. Divide interval [a,b] into n equal segments so that the segment width is given by**

**h =**

**Apply Simpson’s 1/3 Rule over each two interval**

= = + +…..++

= \*[f(x0)+4\*f(x1)+f(x2)]+ \*[f(x2)+4\*f(x3)+f(x4)]+….+ \*[f(xn-2)+4\*f(xn-1)+f(xn)]

= \*[f(x0)+4\*{f(x1)+f(x3)+….+f(xn-1)}+ 2\*[f(x2)+f(x4)+f(x6)]+….+f(xn-2)}+f(xn)].(2)

Equation (2) is called composite Simson’s 1/3 rule

**Example:** Evaluate the ∫01exdx, by Composite Simpson’s ⅓ rule.

**Solution:**

Let us divide the range (0,1) into six equal parts by taking h = 1/6.

When, x0 = 0 then y0 =e0 = 1

Now, when;

x1 = x0+ h = ⅙, then y1 = e1/6 = 1.1813

x2 = x0+ 2h = 2/6 = 1/3 then, y2 = e1/3= 1.3956

x3 = x0+ 3h = 3/6 = ½ then y3 = e1/2= 1.6487

x4 = x0+ 4h = 4/6 ⅔ then y4= e2/3 = 1.9477

x5 = x0+ 5h = ⅚ then y5 = e5/6= 2.3009

x6= x0+ 6h = 6/6 = 1 then y6 = 2.7182

We know by Simpson’s ⅓ rule;

∫ab f(x) dx = h/3[(y0+yn) + 4(y1+y3+y5+….+yn-1)+2(y2+y4+y6+…..+yn-2)]

Therefore,

∫01exdx = 1/18[(1+2.718)+4(1.1813+1.6487+2.3009)+2(1.39561+1.9477)]

= 0.055[3.7182 + 20.52422 + 6.6866]

= 1.71828

**Example 2:** Apply Simpson’s 1/3 rule to calculate dx by using 4 segment and 8 segments

Solution:

Here, x0, xn=1.

For k=4

h = (xn-x0)/k = (1-0)/4 = 0.25

Now,

F(x0) = f(0) = 1

F(x1) = f(x0+h) = f(0.25) = 0.968

F(x2) = f(X0+2h) = f(0.5) = 0.866

F(x3) = f(x0+3h) = f(0.75) = 0.661

F(x4) = f(x0+4h) = f(1) = 0

From Simpson’s 1/3 composite rule, we have

= \*[f(x0)+4\*{f(x1)+f(x3)+….+f(xn-1)}+ 2\*[f(x2)+f(x4)+f(x6)]+….+f(xn-2)}+f(xn)]

= \*[f(x0)+4\*{f(x1)+f(x3)}+ 2\*{[f(x2)}+f(x4)]

= \*[1+4\*{0.968+0.661)}+ 2\*{0.866}+0)] = 0.771

For k=8.

Algorithm for Composite Simpson’s 1/3 rule

1. Start
2. Read value of upper limit xn and value lower limit x0
3. Read number of segments say k
4. Calculate h = (xn-x0)/k
5. Set result = f(x0)+f(xn)
6. For i-1 to k-1

Set result = result+4\*f(x0+i\*h)

i = i+2

1. For i=2 to k-2

Set result = result+2\*f(x0+i\*h)

i= i+2

1. Calculate the value of integration by using formula I = \*result
2. Display the value of integration “I”
3. Stop

**C program for composite Simpson’s1/3 rule**

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) sqrt(1-x\*x)

int main()

{

float x0,x1,xn,f0,fn,f1,h,a,result,I;

int i,k;

printf("Enter lower limit\n");

scanf("%f",&x0);

printf("Enter upper limit\n");

scanf("%f",&xn);

printf("Enter number of segments\n");

scanf("%d",&k);

h = (xn-x0)/k;

x1 = x0+h;

f0 = f(x0);

fn = f(xn);

result = f0+fn;

for(i=1;i<=k-1;i=i+2)

{

a =x0+i\*h;

result = result+4\*f(a);

}

for(i=2;i<=k-2;i=i+2)

{

a =x0+i\*h;

result = result+2\*f(a);

}

I = h\*result/3;

printf("The value of integration =%f\n",I);

getch();

return 0;

}

Output:

Enter lower limit

0

Enter upper limit

1

Enter number of segments

4

The value of integration =0.770899

**Example 2:**  Apply Simpson’1/3 rule to calculate sinx dx using 6 segment

Solution:

Here x0=0, xn = π

For k=6,

H = (xn-x0)/6 = π/6

f(x0) = f(0) = 0

f(x1) = f(x0+h) = f(π/6) = 0.5

f(x2) = f(x0+2h) = f(π /3) = 0.866

f(x3) = f(x0+3h) = f(π/2) = 1

f(x4) = f(x0+4h) = f(2 π/3) = 0.866

f(x5) = f(x0+5h) = f(5 π /6)=0.5

f(x6) = f(x0+6h) = f(π) = 0

Now,

sinx dx = h/3{f(x0)+4{f(x1)+f(x3)}+2{f(x2)+f(x4)}+f(x6)} = π/18{0+4\*(0.5+1+0.5)+2\*(0.866+0.866)+0} = 2.007

**Simpson’s 3/8 Rule**

Simpson’s 3/8 rule for integration can be derived approximation the given function f(x with the third order polynomial or cubic polynomial.

= nh[f(x0)+ ∆ f(x0) + (2n2-3n)∆2 f(x0)+ [n3-4n2+4n]]∆3 f(x0)+…}]

Where h = (xn-x0)/n

By putting n=3 in above relation and neglecting higher order forward differences, it can be written as

= 3h[f(x0)+ ∆ f(x0) + ∆2 f(x0)+ ∆3 f(x0)}]

=[f(x0)+3f(x1)+3f(x2)+f(x3)]

This equation is called Simpson’3/8 rule.

**Example:** Apply Simpson’s 3/8 rule to calculate dx

Solution:

Here x0=0,xn = 1

h = (xn-x0)/3 = 0.33

Simpson’3/8 rule is given by

[f(x0)+3f(x1)+3f(x2)+f(x3)]

Since f(x) = dx

F(x0) = f(0) = 1

F(x1) = f(x0+h) = f(0.33) = 0.944

F(x2) = f(x0+2h) = f(0.66) = 0.751

F(X3) = f(x0+3h) = f(1) = 0

Thus,

[f(x0)+3f(x1)+3f(x2)+f(x3)] = [1+3\*0.944+3\*0.751+0] = 0.753

Example 2: Apply Simpson’s 3/8 rule to calculate )dx

Solution:

Here, h = (2-0)/3 = 0.666

Simpson’s 3/8 rule is given by:

[f(x0)+3f(x1)+3f(x2)+f(x3)]

Since f(x) = )dx

f(x0) = f(0) = 3

f(x1) = f(x0+h) = f(0.666) = 1.939

f(x2) = f(x0+2h) = f(1.332) = 1.328

f(x3) = (x0+3h) = f(2) = 1.049

Thus,

[f(x0)+3f(x1)+3f(x2)+f(x3)] = [3+3\*1.939+3\*1.328+1.049)] = 3.46

**Algorithm for Simpson’s 3/8 Rule**

1. Start
2. Read value of upper limit say x3
3. Read value of lower limit say x0
4. Set n=3
5. Computer h = (x3-x0)/n
6. Set x1 = x0+h
7. Set x2 = x0+2h
8. Compute f0 = f(x0), f1 = (x1), f2(x2), f3 = (x3)
9. Calculate the value of integration by using formula I = 3/8\*[f0+3\*f1+3\*f2+f3]
10. Display value of “I”
11. Stop

………………………………………………….

C program for composite Simpson’s 3/8 rule

//Lab 21: Write a C program to find integration

//using simpson 1/3 rule

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) sqrt(1-x\*x);

int main()

{

float x0,x1,x2,x3,h,I,f0,f1,f2,f3;

printf("Enter upper limit\n");

scanf("%f",&x3);

printf("Enter lower limit\n");

scanf("%f",&x0);

h = (x3-x0)/3;

x1 = x0+h;

x2 = x0+2\*h;

x3 = x0+3\*h;

f0 = f(x0);

f1 = f(x1);

f2 = f(x2);

f3 = f(x3);

I = 3\*h\*(f0+3\*f1+3\*f2+f3)/8;

printf("Integration =%f",I);

getch();

return 0;

}

Output:

Enter upper limit

1

Enter lower limit

0

Integration =0.758062

**Composite Simpson’s 3/8 Rule :** It is also called multi-segment Simpson’s 3/8 rule. It divided the interval ]x0,xn] into k segments and apply the Simpson’s 3/8 rule repeatedly over each segment. Therefore k needs to be multiple of 3. The segment width is given by

h =

Apply Simpson’s 3/8 rule over each interval, then we have

= =[f(x0)+3f(x1)+3f(x2)+f(x3)]+[f(x3)+3f(x4)+3f(x5)+f(x6)]+……………+[f(xn-6)+3f(xn-5)+3f(xn-4)+f(xn-3)]+[f(xn-3)+3f(xn-2)+3f(xn-1)+f(xn)]

Or = =[f(x0)+3+2+f(xn)]…..(2)

This equation is called Composite Simpson’s 3/8 rule

Example 1: Calculate the integral of following tabulated function from x= 0 to x=1.7 using Simpson’s 3/8 rule.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | 0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 | 1.2 | 1.4 | 1.6 |
| F(X) | 0 | 0.24 | 0.55 | 0.92 | 1.63 | 1.84 | 2.37 | 2.95 | 3.56 |

Solution:

Here h=0.2

Composite Simpson’s 3/8 rule is given by = =[f(x0)+3+2+f(xn)]

I = [f(x0)+3{f(x1)+fx(x2)+f(x4)+f(x5)+f(x7)+f(x8)+2{f(x3)+f(x6)+f(x9)]

=[0+3(0.24+0.55+1.63+1.84+2.95)+2(0.92+2.37)] = 2.28

Example 2: Calculate the integral value of dx by using composite Simpson’s 3/8 rule with 9 segments:

Solution:

Here, h = (3-0)/9 = 0.3333

Composite Simpson’s 3/8 rule is given by

I = [f(x0)+3+2+f(xn)]

I = 3{f(x1)+f(x2)+f(x4)+f(x5)+f(x7)+f(x8)}+2{f(x3)+f(x6)+f(x9)}

Since f(x) =

F(x0) =f(0) = 0.25

F(x1) = f(x0+h) = f(0.333) = 0.23

F(x2) = f(x0+2h) = f(0.666) = 0.214

F(x3) = f(x0+3h) = f(1) = 0.2

F(x4) = f(x0+4h) = f(1.333) = 0.188

F(x5) = f(x0+5h) = f(1.666) = 0.176

F(x6) = f(x0+6h) = f(2) = 0.167

F(x7) = f(x0+7h) = f(2.333) = 0.158

F(x8) = f(x0+8h) = f(2.666) = 0.15

F(x9) = f(x0+9h) = f(3) = 0.143

Thus,

I = 3{f(x1)+f(x2)+f(x4)+f(x5)+f(x7)+f(x8)}+2{f(x3)+f(x6)+f(x9)}

= [0.25+3{0.23+0.214+0.188+0.176+0.158+0.15)+2\*(0.2+0.16)+0.167] = 1.118

**Algorithm for Composite Simpson 3/8 rule**

1. Start
2. Read the value of lower limit and upper limit x0 and xn
3. Read the number of segments say k
4. Calculate h = (xn-x0)/k
5. Computer result = f(x0)+f(xn)
6. For i=1 to k-1

if(mod3!=0)

result = result+3\*f(x0+i\*h)

else

result = result+2\*f(x0+i\*h)

1. End for
2. Calculate the value of integration by using formula v = 3/8\*h\*result
3. Stop

…………………………………………………

C p//Lab 21: Write a C program to find integration

//using simpson 3/8 rule

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) sqrt(1-x\*x)

int main()

{

float x0,x1,xn,h,f0,fn,result,a,I;

int k,i;

printf("Enter lower limit\n");

scanf("%f",&x0);

printf("Enter upper limit\n");

scanf("%f",&xn);

printf("Enter number of segments\n");

scanf("%d",&k);

h = (xn-x0)/k;

f0 =f(x0);

fn = f(xn);

result = f0+fn;

for(i=1;i<=k-1;i++)

{

if(i%3!=0)

{

a = x0+i\*h;

result = result+3\*f(a);

}

else

{

a = x0+i\*h;

result = result+2\*f(a);

}

}

I= 3\*h\*result/8;

printf("Integration =%f",I);

getch();

return 0;

}

Output:

Enter lower limit

0

Enter upper limit

1

Enter number of segments

6

Integration =0.775806

**Gaussian Integration**

Newton Cotes formula was derived by integrating the Newton Gregory difference interpolating polynomial. Consequently, all the rules are based on evenly faced sampling points within the range of integral.

Gauss integration is based on the concept that the accuracy of numerical integration can be improved by choosing the sampling points wisely rather than on the basis of equal spacing. Consider a simple trapezoidal rule as shown below figure (a. Now, consider figure(b). Here, the straight line has been moved up such that area B = A+C. The sampling points are moved away from the end points. The function values at the end points are not in computation. Rather function values f(x1) and f(x2) are used to compute the shaded area. It is clear that that the area obtained from figure (b) would be much closer to the actual area. The problem is to compute the values of x1 and x2 given the values of a and b and to choose appropriate “weights w1 and w2. The method of implementing the strategy of finding appropriate values of xi and wi and obtaining the integral of f(x) is called the Gaussian integration or quadrature.

Thus, the two point Gauss Quadrature rule is an extension of the trapezoidal rule approximation where the arguments of the function are not predetermined as a and b but as unknown x1 and x2. So in the two point Gauss Quadrature rule, the integral is approximated as

I = dx ≈  c1f(x1)+c2f(x2)

There are four unknowns x1, x2, c1 and c2. These are found by assuming that the formula gives exact results for integrating a general third order polynomial,

f(x) = a0+a1x+a2x2+a3x3.

Hence,

dx = a0+a1x+a2x2+a3x3 dx = 2a0+ a2 …(1)

The formula gives,

dx = c1f(x1)+c2f(x2) = c1(a0+a1x1+a2x12+a3x13)+c2(a0+a1x2+a2x22+a3x23)

=a0(c1+c2)+a1(c1x1+c2x2)+a2(c1x12+c2x22)+a3(c1x13+c2x23) ….(2)

Thus, from equation (1) and equation (2), we get

2a0+ a2 = a0(c1+c2)+a1(c1x1+c2x2)+a2(c1x12+c2x22)+a3(c1x13+c2x23)

Equating the terms on both side, we get

c1+c2= 2

c1x1+c2x2 = 0

c1x12+c2x22 =

c1x13+c2x23 = 0

We can find that the above simultaneous nonlinear equations have only one acceptable solution

c1 = c2 = 1

x1 = - () = -0.5773505 and x2 = () = 0.5773505

dx = f(()+f(())

Since two points are chosen, it is called the two-point Gauss quadrature rule. Higher point versions can also be developed. The generalized n-point Gaussian quadrature rule is given as:

dx =

By using the procedure used in deriving two point Gaussian quadrature rule, we can calculate the parameters wi, zi for higher order versions of Gaussian quadrature. Some parameters for Gaussian integration is listed below:

For n=3

w1 = 0.55556 w2 = 0.88889 w3 = 0.55556

x1 = -0.7746 x2 = 0 x3 = 0.7746

For n = 4

w1 = 0.34785 w2 = 0.65215 w3 = 0.65215 w4 = 34785

x1 = -0.86114 x2 = -0.33998 x3 = 0.33998 x4 = 0.86114

Example: Compute ex dx using two point Gauss Legendre formula

Solution:

Using two-point Gauss Legendre formula, we have

ex dx = f(x1) +f(x2)

Where x1 and x2 are Gaussian quadrature points and are given by

X1 = - () = -0.5773505 and x2 = () = 0.5773505

Therefore,

I = e(-0.5773505) +e(0.5773505)

= 2.3426961.

**Changing Limit of Integration:** Note that the Gaussian formula imposes a restriction on the limits of integration to be from -1 to 1. This restriction can be overcome by using the technique of “interval transformation” used in calculus. Let

f(x) dx = C g(z) dz

Assume the following transformation between x and the new variable z

x = Az+B

This must satisfy the following conditions:

At x = a ,z=-1 and x=b z=1

That is,

B-A = a

A+B = b

Then

A = and B =

Therefore,

x = z+

dx =

This implies that

C =

Then the integral becomes,

g(z) dz

The Gaussian formula for this integration is:

g(z) dz = wig(zi)

Where wi and zi are the weights and quadrature points for the integration domain (-1,1)

**Example:** Compute the integral e-x/2 dx using Gaussian two point formula

Solution:

Here n =2 and therefore

Ig = [w1g(z1)+w2g(z2)]

x = z+ = 2z

C = = (2+2)/2 = 2

Therefore,

g(z) = e-(2z)/2 = e-z

For a two point formula

w1 = w2 = 1

z1 = - () = -0.5773505 and z2 = () = 0.5773505

Upon substitution of these values, we get

Ig = 2\*[e -()+e ()] = 4.685322

**Algorithm for Gaussian Integration**

1. Start
2. Read value of lower and upper bound say a and b
3. Set w1 = w2 =1, z1 = -0.57735, z2 = 0.57735
4. Compute x1 = z1+
5. x1 = z1+
6. Compute v = x1 = {f(x1)+f(x2)}
7. Display “v”
8. Stop

C Program to calculate integral using Gaussian Integration

#include<stdio.h>

#include<conio.h>

#include<math.h>

#define f(x) x\*x\*x+1

int main()

{

float a,b,z1,z2,c1,c2,x1,x2,v;

printf("Enter lower limit\n");

scanf("%f",&a);

printf("Enter upper limit\n");

scanf("%f",&b);

c1=c2=1;

z1 = -0.57735;

z2 = 0.57735;

x1 = (b-a)/2\*z1+(b+a)/2;

x2 = (b-a)/2\*z2+(b+a)/2;

v = (b-a)/2\*((f(x1))+f(x2));

printf("Value of integration =%f",v);

getch();

return 0;

}

Output:

Enter lower limit

2

Enter upper limit

4

Value of integration =61.999996

**Romberg Integration**

It is clear that the accuracy of a numerical integration process can be improved in two ways:

1. By increasing the number of subintervals. That is by decreasing h. This decreases the magnitude of error items. Here the order of the method is fixed.
2. By using higher order methods. This eliminates the lower order errors items. Here the order of the method is varied and therefore, this method is known as variable order approach

The variable order method can be implemented using Richardson’s extrapolation technique. As we know, this technique involves combining two estimates of a given order to obtain a third estimate of higher order. The method that incorporates this process to the trapezoidal rule is called Romberg integration.

According to the Euler Maclaurin formula, the error expansion for trapezoidal rule approximation to a define a integral is of the form:

f(x) dx –T(h) = a2h2+a4h4+a6h6+…… (1)

T(h) is the trapezoidal approximation with step size = (b-a)/n = h

Let us define

T(h,0) = T(h)

to indicate that T(h) is the trapezoidal rule with no Richardson’s extrapolation being applied(zero level extrapolation). Thus equation (1) can be written as

I = T(h,0)+ a2h2+a4h4+a6h6+……(2)

Let us have another estimate with step size = (b-a)/2n = h/2 (at zero level extrapolation) as

I = T(h/2,0) + h2+ h4 + h6+…………………(3)

By multiplying equation (3) by 4 and then subtracting equation (2) from the resultant equation, we get

I = +b4h4+b6h6+…….

= T(h/2,1)+ b4h4+b6h6+……………………..(4)

Where

T(h/2,1) =

is the corrected trapezoidal formula using Richardson’s extrapolation technique “once” (level 1). The truncation error is of the order h4 instead of h2 which is the order in the “uncorrected” trapezoidal formula.

Now we can apply Richardson’s extrapolation technique once more to equation (4) to eliminate the error term containing h4. The result would be

I = +b6h6+…….

= T(h/4,2)+C6h6+….

Where

T(h/4,2) =

Is the estimate, refined again by applying Richardson’s extrapolation a second time (level 2). Similarly, we can obtain an estimate with third level correction as:

T(h/8,3) =

The entire process of repeated use of Richardson’s extrapolation technique can be represented in general form as:

T(h/2j,j) = ……(5)

Where i=0,1,2… denotes the depth of division and j ≤ i denotes the level of improvement

We can further simplify the notation of equation (5) by defining

Ri,j = T(h/2j,j)

Thus, we have,

Rij = ………(5)

Equation (5) is known as Romberg integration formula. Note that this equation when extended will form a lower diagonal matrix. The elements of the matrix R are computed row by row in the order indicated as in the following figure. The circled numbers indicate the order of computations and the arrows indicate the dependencies of elements. An element at the head end depends on the element at the tail end.

R(0,0)

R(2,0)

R(1,0)

R(1,1)

R(3,0)

R(2,1)

R(3,1)

R(2,2)

R(3,2)

R(3,3)

Elements in the first column represent trapezoidal rule at h, h/2 , h/4 etc. They can be evaluated recursively as follows:

h = (b-a)

R(0,0) = h/2[f(a)+f(b)]

R(i,0) = +hi f(x2k-1) for i=1,2.. (6)

Where

hi = (b-a)/2i

xk = a+khi

Equation (6) is known as recursive trapezoidal rule

**Example:** Compute Romberg estimate R22 for 1/x dx

Solution:

First we apply the basic trapezoidal rule to obtain R(0,0)

R(0,0) = h/2[f(a)+f(b)] = (2-1)/2[1+1/2] = 0.75

Now, we obtain R(1,0) = + h1f(x1) = + \* = 0.7083333

R(2,0) = +h2[f(x1)+f(x3)] = + [f(1.25)+f(1.75) = 0.6970237

Now, Romberg approximation can be obtained using equation

R(1,1) = = = 0.6944444

R(2,1) = = = 0.6932538

R(2,2) = = = 0.6931744

Correct answer = 0.6931471

Error = 0.0000273

**Algorithm for Romberg Integration**

1. Start
2. Read lower limit and upper limit say x0 and xn
3. Compute h = (xn-x0)
4. Read value of p and q for Tpq
5. Compute T(0,0) using formula T(0,0) = h/2\*[f(x0)+f(x1))
6. For i=1 to p

T[i][0] = T[i-1][0]/2+h/2i\* f(x0+(2k-1)/2i)

1. End for
2. For c=1 to p

For k=1 to c

m= c-k

if(k<=q)

T[m+k][k] = (4k \*T[m+k][k-1]-T[m+k-1][k-1])/4k-1)

1. End for
2. Display Romberg estimate of integration T[p][q]
3. stop

C program for Romberg Integration

#include<stdio.h>

#include<conio.h>

#include<math.h>

float f(float x)

{

if(x==0)

return 1.0;

else

return sin(x)/x;

}

int main(){

float x0,xn,t[10][10],h,sm,sl,a;

int i,k,c,r,m,p,q;

printf("Enter lower and upper limit:");

scanf("%f%f",&x0,&xn);

printf("enter p and q required T(p,q):");

scanf("%d%d",&p,&q);

h=xn-x0;

t[0][0]=h/2\*((f(x0))+ (f(xn)));

for(i=1;i<=p;i++){

sl=pow(2,i-1);

sm=0;

for(k=1;k<=sl;k++){

a=x0+(2\*k-1)\*h/pow(2,i);

sm=sm+(f(a));

}

t[i][0]=t[i-1][0]/2+sm\*h/pow(2,i);

}

for(c=1;c<=p;c++){

for(k=1;k<=c && k<=q;k++){

m=c-k;

t[m+k][k]=(pow(4,k)\*t[m+k][k-1]-t[m+k-1][k-1])/(pow(4,k)-1);

}

}

printf("Romberg estimate of integration =%f",t[p][q]);

return 0;

}