**Unit 2: Interpolation**

Given n+1 data points (x0,y0),(x1,y1),…………….(xn-1,yn-1),(xn, yn), interpolation is the process of finding an equation y = f(x) that passes through above n+1 data points using this equation to find value of y at x, x0<x<xn. We can use interpolation to find value of y at x that is not given in the set of data points. And extrapolation is the process of finding an equation y = f(x) that passes through above n+1 data points and the using this equation to find value of y at x<x0 or x>xn.

A table of data may belong to one of the following two categories

1. **Table of values of well-defined functions :** Examples of such tables are logarithmic tables, trigonometric tables, interest tables etc
2. **Data tabulated from measurements made during an experiment:** In such experiments, values of the dependent variable are recorder at various values of the independent variable. There are numerous examples of such experiments- the relationship between voltage applied and speed of fan etc,

In the first case, the function is constructed such that it passes through all the data points. This method is constructing a function and estimating values at non-tabular points is called **interpolation**. The functions are known as **interpolating polynomials**.

In the second case, the values are not accurate and, therefore, it will be meaningless to try to pass the curve through every point. The best strategy would be to construct a single curve that would represent the general trend of data, without necessarily passing through the individual points. Such a functions are called approximating functions. One popular approach for finding an approximate function to fit a given set of experimental data is called least squares polynomials.

A polynomial is a common choice for an interpolating function because polynomials are easy to evaluate, differentiate and integrate relative to other choices such as trigonometric and exponential series

In this chapter, we discuss various methods of interpolation. They include:

1. Lagrange Interpolation
2. Newton’s Interpolation
3. Newton Gregory forward interpolation
4. Spline Interpolation

**Polynomial Forms :** The most common form of an nth order polynomial is

p(x) = a0+a1x+a2x2+….+anxn

This form, known as the power form, is very convenient for differentiating and integrating the polynomial function and therefore is most widely used in mathematical analysis.

However, there are many situations where this form has been found inadequate. The following example demonstrates this situation

**Example:** Consider the power form of p(x) for n=1,

p(x) = a0+a1x

Given that

p(100) = 3/7

p(101) = -4/7

Obtain the linear polynomial p(x) using four digit floating point arithmetic. Verify the polynomial by substituting back the values x = 100 and x=101

Solution:

p(100) = a0+100a1 = 0.4282

p(101) = a0+100a1 = -0.5714

Solving we get,

a1 = -1 and a0 = 100.4 (only four significant digits)

Therefore,

p(x) = 100.4-x

Using this polynomial, we obtain

p(100) = 0.4

p(101) = -0.6

Compare these results with the original values of p(100) and p(101). We have lost three decimal digits.

The power form of polynomial may not always produce accurate results. In order to overcome such problems, we have alternative forms of representing a polynomial. One of them is the shifted power form as shown below:

p(x) = a0 +a1(x-C)+a2(x-C)2+….+an(x-C)n

where C is a point somewhere in the interval of interest. This form of representation significantly improves the accuracy of the polynomial evaluation.

**Example:** Consider the power form of p(x) for n=1,

p(x) = a0+a1x

Given that

p(100) = 3/7

p(101) = -4/7

Obtain the linear polynomial p(x) using four digit floating point arithmetic using shifted power form. Verify the polynomial by substituting back the values x = 100 and x=101

Solution:

Shifted power form of first order p(x) is

p(x) = a0+a1(x-C)

Let us choose the center C as 100.

Then we have

p(x) = a0+a1(x-100)

This gives

p(100) = a0+ a1(100-100) = 3/7

This gives a0 = 3/7 = 0.4286

p(101) = 0.4286 + a1(101-100) = -0.5714

Thus, the linear polynomial becomes,

p(x) = 0.4282-(x-100)

Using this polynomial, we obtain

p(100) = 0.4286

p(101) = -0.5714

**Newton form of Polynomial**

There is a third form of polynomial known as Newton form. This is a generalized shifted power form as shown below:

p(x) = a0+a1(x-C1)+a2(c-C1)(x-C2)+a3(x-C1)(x-C2)(x-C3)+…..+an(x-C1)(x-C2)(x-C3)…(x-Cn)………(1)

Note that equation (1) reduces to shifted power form when C1 = C2 = C3 =Cn = C and to simple power form when Ci = 0 for all i.

Newton polynomial form plays an important role in the derivation of an interpolating polynomial

Polynomial can also be expressed in the form

p2(x) = b0(x-x1)(x-x2)+b1(x-x0)(x-x2)+b2(x-x0)(x-x1)

**Lagrange Interpolation**

A second order polynomial can be written in the form

p2(x) = b1(x-x0)(x-x1)+b2(x-x1)(x-x2)+b3(x-x2)(x-x0)

Let (x0,f0), (x1,f1) and (x2,f2) are three interpolating points. Substituting these points in equation (1) we get

p2(x0) =f0 = b2(x0-x1)(x0-x2)

p2(x1) =f1 = b2(x1-x2)(x1-x0)

p2(x2) =f2 = b2(x2-x0)(x1-x1)

From above three equations we can calculate value of b1 ,b2 and b3. Formulate for calculating these values are given below

b2 =

b3 =

b1 =

Substituting these values of b1, b2 and b3 in equation (1) we get

p2(x) = + + ……………………………(1)

Equation (1) can also be represented as

p2(x) = f0l0(x)+f1l1(x)+f1l2(x)

p2(x) =

where li(x) =

In general, for n+1 points, we have nth degree polynomial as

pn(x) =

**Example:** The table below gives squares roots for integers:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| x | 1 | 2 | 3 | 4 | 5 |
| f(x) | 1 | 1.4142 | 1.7321 | 2 | 2.2351 |

Determine the square root of **2.5**

1. Using linear polynomial
2. Using second order polynomial

Solution:

a). The given value of 2.5 lies between the points 2 and 3. Therefore,

x0 = 2 f(x0) = f0 = 1.4142

x1 = 3 f(x1) = f1 = 1.7321

x = 2.5

Also,

l0(x) = = = 0.5

l1(x) = = = 0.5

Now, p1(x) = f0l0(x)+f1l1(x) = 1.4142\*0.5+1.7321\*0.5 = 1.5732

b)

Let us consider following three points:

x0 = 2, x1 = 3 and x2 = 4

Then f0 = 1.4142 f1 = 1.7321 and f2 =2

For x = 2.5, we have

l0(2.5) = = 0.3750

l1(2.5) = = 0.7500

l2(2.5) = = -0.125

p2(2.5) = (1.4142)(0.3750)+(1.7321)(0.7500)+2.0\*(-0.125) = 1.5794

Example 2: Find the Langrange’s polynomial to fit the following data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| I | 0 | 1 | 2 | 3 |
| xi | 0 | 1 | 2 | 3 |
| ex-1 | 0 | 1.7183 | 6.3891 | 19.0855 |

Use the polynomial to estimate the value of e1.5

**Solution:**

Here, x0 = 0, x1 = 1, x2 = 2, x3 = 3

f0 = 0, f1 = 1.7183, f2 = 6.3891 f3 = 19.0855

l0(x) = =

l1(x) = =

l2(x) = =

l3(x) = =

The interpolation polynomial is

p(x) = f0 l0(x) + f1 l1(x) +f2l2(x) +f3 l3(x) = 0 + 1.7183\* + 6.3891 \* + 9.0856 \*

p2(x) = 0.8455x3-1.060x2+1.933x

Now p(1.5) = 3.3677

e1.5 = p(1.5)+1 = 4.3677

**Algorithm**

1. Start
2. Read the number of points say n
3. Read the value at which value is needed, say x
4. Read the given data points
5. Calculate the values Li as below

For i= 1 to n

For j=1 to n

If(j!=i)

L[i] = L[i]\*((x-x[j])/(x[i]-x[j]))

end if

end for

1. Calculate interpolated value at x as below

For i=1 to n

v = v +fx[i]\*L[i];

End for

1. Print the interpolation value v at x
2. Stop

**C program for Lagrange Interpolation**

#include<stdio.h>

#include<conio.h>

int main()

{

float x[100], y[100], xp, yp, p;

int i,j,n;

printf("Enter number of data: ");

scanf("%d", &n);

printf("Enter data:\n");

for(i=1;i<=n;i++)

{

printf("x[%d] = ", i);

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

printf("y[%d] = ", i);

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

}

printf("Enter interpolation point: ");

scanf("%f", &xp);

for(i=1;i<=n;i++)

{

p=1;

for(j=1;j<=n;j++)

{

if(i!=j)

{

p = p\* (xp - x[j])/(x[i] - x[j]);

}

}

yp = yp + p \* y[i];

}

printf("Interpolated value at %.3f is %.3f.", xp, yp);

getch();

}

**Output:**

Enter number of data: 3

Enter data:

x[1] = 2

y[1] = 4

x[2] = 3

y[2] = 9

x[3] = 4

y[3] = 16

Enter interpolation point: 2.5

Interpolated value at 2.500 is 6.250

**Newton Interpolation Polynomial**

We have seen that, in Lagrange interpolation, we cannot use the work that has already been done if we want to incorporate another data point in order to improve the accuracy of estimation. It is therefore necessary to look for the some other form of representation to overcome this drawback.

To overcome this problem Newton derived another form of interpolation.

The Newton form of polynomial is:

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

To construct the polynomial we need to find coefficients a0, a1,a2, a3….an. Let us suppose (x0,f(x0)), (x1, f(x1)),…………………..(xn, f(xn)) are given interpolating points. Now, at x = x0 equation (1) becomes

p(x0) = f(x0) = a0

🡪 a0 = f(x0)

Similarly, at x = x1 equation (1) becomes

p(x1) = f(x1) = a0+a1(x1-x0)

a1 = =

At x = x2 equation (1) becomes

p(x2) = f(x2) = a0+a1(x2-x0)+a2(x2-x0)(x2-x1)

🡪f(x2) = f(x0)+(x2-x1)+a2(x2-x0)(x2-x1)

This gives

a2 =

Note that a0, a1, a2 are finite divided differences. a0, a1 and a2 are the first, second and third divided differences respectively. We denote the first divided difference by

f[x0] = f(x0)

The second divided difference by

f[x1,x0] =

and the third divided difference by

f[x2,x1,x0] = =

This leads us to writing general form of the Newton’s divided difference polynomial for n+1 data points, (x0,y0), (x1,y1),….(xn-1,yn-1),(xn, yn) as

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

where

a0 = f[x0]

a1 = f[x1,x0]

a2 = f[x2,x1,x0]

………………………

an-1 = f[xn-1,xn-2,….x0]

Where the definition of the mth divided difference is

am = f[xm………..x0]

=

Substituting for ai coefficients in equation (1) we get

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

This can be written more compactly as

pn(x) = ……………………….(2)

Equation (2) is called **Newton’s divided difference interpolation polynomial**.

**Example:** Given below is a table of data for log x. Estimate log 2.5 using second order Newton interpolation polynomial

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 1 | 2 | 3 | 4 |
| Logx | 0 | 0.3010 | 0.4771 | 0.6021 |

Solution:

Second order polynomials require only three data points. We use the first three points.

a0 = f[x0] = 0

a1 = f[x0,x1] = = 0.3010/(2-1) = 0.3010

a2 = f[x0,x1,x2] =

f[x1,x2] = f(x2)-f(x1)/(x2-x1) = (0.4771-0.3010)/(3-2) = 0.1761

Therefore

a2 = = -0.06245

p2(x) = a0+a1(x-x0)+a2(x-x0)(x-x1)

p2(x) = 0+0.3010(x-1)+(-0.06245)(x-1)(x-2)

Now p2 (2.5) = 0+0.3010(2.5-1)+(-0.06245)(2.5-1)(2.5-2) = 0.4047

**Algorithm**

1. Start
2. Read number of points say n
3. Read the value at which interpolated value is needed say, x
4. Read given data points
5. Calculate first divided difference as

for i=0 to n-1

dd[i] = fx[i]

End for

1. Calculate second to nth divided difference as

For i=0 to n-1

For j = n-1 to i+1

Dd[i] = (dd[j]-dd[j-1])/(x[j]-x[j-1-i]

End for

End for

1. Set v=0, and p=1
2. Calculate interpolated value as

For i=0 to n-1

For j=0 to i-1

P = p\*(x-xj)

End for

v = v+dd[i]\*p;

reset p = 1;

End for

1. Print the interpolated value v
2. Stop

**C Program for Newton’s Interpolation**

#include<stdio.h>

#include<conio.h>

int main()

{

int n,i,j;

float v=0,p,xv,x[10],fx[10],a[10];

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

scanf("%d",&n);

printf("Enter the value of x\n");

scanf("%f",&xv);

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

{

printf("Enter the value of x and fx at i = %d\n",i);

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

}

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]);

}

}

v=0;

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

{

p=1;

for(j=0;j<=i-1;j++)

p = p\*(xv-x[j]);

v = v+a[i]\*p;

}

printf("Interpolation value = %f",v);

getch();

return 0;

}

Output:

Enter number of points

4

Enter the value of x

2.5

Enter the value of x and fx at i = 0

1 0

Enter the value of x and fx at i = 1

2 0.3010

Enter the value of x and fx at i = 2

3 0.4771

Enter the value of x and fx at i = 3

4 0.6021

Interpolation value = 0.400050

**Divided Difference Table**

We have seen that the coefficients of Newton’s divided difference interpolation polynomials are evaluated using the divided differences at the interpolating points. We have also seen that a higher order divided difference is obtained using the lower order differences. Finally, the first order divided differences use the given interpolating points. For example, consider the second order divided difference

a2 = f[x0,x1,x2] =

Where f[x1,x2] and f[x0,x1] are first order differences and are given by

f[x0,x1] = =

f[x1,x2] = =

This shows that, given the interpolating points, we can obtain recursively a higher order divided difference, starting from the first order differences. While this can be conveniently implemented in a computer, we can generate a divided difference table for manual computing. A particular entry in the table is obtained as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| I | Xi | f[x] | First Difference | Second Difference | Third Difference | Fourth Difference |
| 0 | x0 | f[x0] |  |  |  |  |
|  |  |  | f[x1,x0] |  |  |  |
| 1 | x1 | f[x1] |  | f[x2,x1,x0] |  |  |
|  |  |  | f[x1,x2] |  | f[x3,x2,x1,x0] |  |
| 2 | x2 | f[x2] |  | f[x3,x2,x1] |  | f[x4,x3,x2,x1,.x0] |
|  |  |  | f[x3,x2] |  | f[x4,x3,x1,x1] |  |
| 3 | x3 | f[x3] |  | f[x4,x3,x2] |  |  |
|  |  |  | f[x4,x3] |  |  |  |
| 4 | x4 | f[x4] |  |  |  |  |
|  |  |  |  |  |  |  |

Draw the two diagonals from the entry to be calculated through its neighboring entries to the left. If these lines terminate at f(xi) and f(xj) then divide the difference of the neighboring entries by the corresponding difference xj-xi. The result is the desired entry.

When the table is completed, the entries at the top of each column represent the divided difference coefficients.

**Example:** Given the following data points, create the table of divided differences. **Use** the table to estimate the value of f(1.8) by using second , third order polynomial.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 1 | 2 | 3 | 4 |
| f(x) | 0 | 7 | 26 | 63 |

Solution:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| I | Xi | f(xi) | First Diff | Second Diff | Third Diff |
| 0 | 1 | 0 = (a0) |  |  |  |
|  |  |  | 7 = (a1) |  |  |
| 1 | 2 | 7 |  | 6 = (a2) |  |
|  |  |  | 19 |  | 1 = (a3) |
| 2 | 3 | 26 |  | 9 |  |
|  |  |  | 37 |  |  |
| 3 | 4 | 63 |  |  |  |

The value of polynomial at x = 1.8 is computed as follows

Evaluating f(1.8) by using second order polynomial

p2(x) = a0+a1(x-x0)+a2(x-x0)(x-x1)

Therefore

p2(1.8) = 0+7(1.8-1)+6(1.8-1)(1.8-2) = 0+5.6-0,96 = 4.64

Evaluating f(1.8) by using third order polynomial

p3(x) = a0+a1(x-x0)+a2(x-x0)(x-x1)+a3(x-x0)(x-x1)(x-x2)

Therefore

p3(x) = 4.64+ a3(x-x0)(x-x1)(x-x2) = 4.64+1(1.8-1)(1.8-2)(1.8-3) = 4.832

**IMPORTANT POINT**

Note that p3(1.8) = 4.832. This implies that correct results can be obtained using the third order interpolation polynomial. It also illustrates that we can compute f(1.8) in stages recursively using interpolation polynomials in increasing order. Computation is terminated when two consecutive estimates are approximately equal or their difference is within a specified limit.

It is clear that the computational effort required in adding one more data point to the estimation process is very much reduced due to the recursive nature of computation.

**Example 2:** Given the following data points. Obtain the table of divided difference. Use the table to find third order Newton’s polynomial and use the polynomial to estimate the value of f(8)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 3 | 7 | 9 | 10 |
| f(x) | 168 | 120 | 72 | 63 |

**Solution:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| I | xi | f(xi) | First Diff | Second Diff | Third Diff |
| 0 | 3 | 168 = (a0) |  |  |  |
|  |  |  | -12= (a1) |  |  |
| 1 | 7 | 120 |  | -2 = (a2) |  |
|  |  |  | -24 |  | 1 = (a3) |
| 2 | 9 | 72 |  | 5 |  |
|  |  |  | -9 |  |  |
| 3 | 10 | 63 |  |  |  |

Finding third order polynomial

We know that

p3(x) = a0+a1(x-x0)+a2(x-x0)(x-x1)+a3(x-x0)(x-x1)(x-x2)

= 168+(-12)(x-3)+(-2)(x-3)(x-2)+1\*(x-3)(x-4)(x-4) = x3-21x2+119x-27

Evaluating p3(8) by using third order polynomial

f(8) = p3(8) = 83-21\*82+119\*8-27 = 93

**Newton’s Forward and Backward Difference Interpolation**

Newton’s divided difference formula can be expressed in a simplified form when x0,x1, x2, x3,,, xn are arranged consecutively with equal spacing. Newton’s forward and backward interpolation formula is used only for equal intervals. Newton’s Forward interpolation formula is used mainly to interpolate the values of f(x) near the beginning of a set of tabular values and also for extrapolating the value of f(x) a short distance ahead of f(x0). This means, is used to find the unknown values of y for some x which lies at the beginning of a set of tabular values. Newton Backward interpolation formula is used mainly to interpolate the values of f(x) near the end of a set of tabular values and also for extrapolating the value of f(x) at a short distance after f(xn). This means it is used to find the unknown values of f(x) for some x which lies at the end of a set of tabular values.

**Newton Forward Difference Interpolation**

We know that Newton’s divided difference polynomial is given by the formula

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

Let us introduce the notation h = xi+1-xi for each 0,1,2…n-1

Suppose x = x0+sh

Since xk = x0+kh

🡪x-xk = (s-k).h

This gives

x-x0 = sh

s-x1 = (s-1)h

…………………

……………….

x - xn-1 = (s-n+1)h

Now, Newton’s interpolation divided difference formula becomes

pn(x) = pn(x0+sh) = f[x0]+f[x1,x0]sh + f[x2,x1,x0]s(s-1)h2+…………….+f[xn,xn-1,…x1,x0]s(s-1)(s-2)…..(s-n+1)hn

Using binomial notation,

C(s,k) =

We can express pn(x) compactly as

Pn(x) = pn(x0+sh) = f[x0]+ hk f[x0,x1,x2….xk]……………..(1)

Newton forward difference formula is constructed by making use of the forward difference notation ∆

With this notation,

f[x0,x1] = = ∆f(x0)

f[x0,x1,x2] = = [(∆f(x1)- ∆f(x0)] = ∆2f(x0)

And in general

f[x0,x1,x2,……………..xk] = ∆kf(x0)

Now equation (1) can be written as

pn(x) = pn(x0+sh) = f[x0] +∆kf(x0)…………………………………………………………(2)

This equation is called Newton Gregory forward difference formula

We can construct Newton’s forward differences by constructing Newton’s Forwards difference table as below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X0 | F(x0) |  |  |  |
|  |  | ∆f(x0) |  |  |
| X1 | F(x1) |  | ∆2f(x0) |  |
|  |  | ∆f(x1) |  | ∆3f(x0) |
| X2 | F(x2) |  | ∆2f(x1) |  |
|  |  | ∆f(x2) |  |  |
| X3 | F(x3) |  |  |  |

Algorithm for Newton’s Forward Difference Interpolation

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

For i=0 to n-1

Fd[i]=fx[i]

End for

1. Calculate second to nth forward differences as below

For i=0 to n-1

For j=n-1 to i+1

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

End for

1. Set v = fd[0] and set p=1
2. Calculate interpolated value as below

For i=0 to n-1

For k=1 to i

P = p\*(s-k+1)

End for

v = v+(fd[i]\*p)/i!

Reset p=1;

End for

1. Print the interpolation value v at xp
2. Stop

**C program for Newton’s Forward Difference Interpolation**

…………………………………………………………………………………………..

#include<stdio.h>

#include<conio.h>

int factorial(int n)

{

if(n<=1)

return 1;

else

return n\*factorial(n-1);

}

int main()

{

int n,i,j,k;

float v=0, p,xp,x[10],fx[10], fd[10],h,s;

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

scanf("%d",&n);

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

scanf("%f",&xp);

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

{

printf("Enter the value of x and fx at i=%d\n",i);

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

}

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]);

}

}

v = fd[0];

for(i=1;i<n;i++)

{

p=1;

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

{

p = p\*(s-k+1);

}

v = v+(fd[i]\*p)/factorial(i);

}

printf("Interpolation value = %f",v);

getch();

return 0;

}

First Run

Enter number of points

4

Enter the value at which interpolated value is needed

0.95

Enter the value of x and fx at i=0

0.9 0.7833

Enter the value of x and fx at i=1

1.0 0.8415

Enter the value of x and fx at i=2

1.1 0.8912

Enter the value of x and fx at i=3

1.2 0.9320

**Interpolation value = 0.813437**

**S**econd Run

Enter number of points

6

Enter the value at which interpolated value is needed

0.0045

Enter the value of x and fx at i=0

0 1.121

Enter the value of x and fx at i=1

0.001 1.123

Enter the value of x and fx at i=2

0.002 1.1255

Enter the value of x and fx at i=3

0.003 1.127

Enter the value of x and fx at i=4

0.004 1.128

Enter the value of x and fx at i=5

0.005 1.1285

**Interpolation value = 1.128400**

**Example:** Construct Network’s forward difference table for data points in table below and then approximate the value of f(1.1) by using Newton’s forward difference formula and fourth divided difference.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 1.0 | 1.3 | 1.6 | 1.9 | 2.2 |
| f(x) | 0.7651977 | 0.6200860 | 0.4554022 | 0.2818186 | 0.11.3623 |

Solution:

Here h = xi+1-xi = 0.3

Since x = x0+sh 🡪s = (s-x0)/h = (1.1-1.0)/0.3 = 1/3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Xi | f(xi) | ∆f(x) | ∆2f(x) | ∆3f(x) | ∆4f(x) |
| 1.0 | 0.7651977 |  |  |  |  |
|  |  | -0.1451117 |  |  |  |
| 1.3 | 0.6200860 |  | -0.0195721 |  |  |
|  |  | -0.1646838 |  | 0.0106723 |  |
| 1.6 | 0.4554022 |  | -0.0088998 |  | 0.0003548 |
|  |  | -0.17365836 |  | 0.0110271 |  |
| 1.9 | 0.2818186 |  | 0.0021273 |  |  |
|  |  | -0.1714563 |  |  |  |
| 2.2 | 0.113623 |  |  |  |  |

Now the Newton forward divided difference formula can be used with the forward divided differences that have a solid underscore in above table

We know that

pn(x) = pn(x0+sh) = f[x0] +∆kf(x0)

Thus,

p4(x) = f[x0]+C(s,1) ∆f(x0)+ C(s,2) ∆2f(x0)+ C(s,3) ∆3f(x0)+C(s,4) ∆4f(x0)

p4(x) = f[x0]+s. ∆f(x0)+ s(s-1) ∆2f(x0)/2+s(s-1)(s-2)∆3f(x0)/3!+s(s-1)(s-2)(s-3) ∆4f(x0)/4!

Hence p4(1.1) = 0.7651977+(1/3)\*(-0.1451117)+(1/3\*-2/3)/2 \*(-.00195721)+(1/3+-2/3\*-5/3)/6 \*0.0106721)+ (1/3\*-2/3\*-5/3\*-8/3)/24 \*(0.0003548) = 0.712

**Example 2:** Obtain the Newton’s forward interpolation polynomial p5(x) for the following tabular data and interpolate the value of the function at x = 0.0045

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | 0 | 0.001 | 0.002 | 0.003 | 0.004 | 0.005 |
| F(X) | 1.121 | 1.123 | 1.125 | 1.127 | 1.128 | 1.1285 |

Solution:

Here, h = xi+1-xi = 0.001

x = x0+sh 🡪s = (x-x0)/h = (0.0045-0)/0.001 = 4.5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Xi | f(xi) | ∆f(x) | ∆2f(x) | ∆3f(x) | ∆4f(x) | ∆5f(x) |
| 0 | **1.121** |  |  |  |  |  |
|  |  | **0.0020** |  |  |  |  |
| 0.001 | 1.123 |  | **0.0005** |  |  |  |
|  |  | 0.0025 |  | **-0.0015** |  |  |
| 0.002 | 1.123 |  | -0.0015 |  | **0.002** |  |
|  |  | -0.0005 |  | 0.0005 |  | **-0.0025** |
| 0.003 | 1.125 |  | -0.0005 |  | -0.005 |  |
|  |  | 0.001 |  | 0 |  |  |
| 0,004 | 1.127 |  | -0.0005 |  |  |  |
|  |  | -0.005 |  |  |  |  |
| 0.005 | 1.1285 |  |  |  |  |  |

Now, the Newton’s forward divided difference formula can be used with the forward divided differences that have a solid underscore in above table.

We know that

pn(x) = pn(x0+sh) = f[x0] +∆kf(x0)

Thus,

p4(x) = f[x0]+C(s,1) ∆f(x0)+ C(s,2) ∆2f(x0)+ C(s,3) ∆3f(x0)+C(s,4) ∆4f(x0)+ C(s,5) ∆5f(x0)

p4(x) = f[x0]+s. ∆f(x0)+ s(s-1) ∆2f(x0)/2+s(s-1)(s-2)∆3f(x0)/3!+s(s-1)(s-2)(s-3) ∆4f(x0)/4! + ∆5f(x0)

Hence p5(x) = p5(x0+sh) = 1.121+0.002\*s + s(s-1)+ s(s-1)(s-2)+ s(s-1)(s-2)(s-3)+ s(s-1)(s-2)(s-3)(s-4)

Now, put x = 0.0045

p5(0.0045) = p5(0+4.5\*0.001) = 1.121+0.002\*s + 4.5\*(4.5-1)+ 4.5\*(4.5-1)(4.5-2)+ \* 4.5\*(4.5-1)(4.5-2)(4.5-3)+ 4.5(4.5-1)(4.5-2)(4.5-3)(4.5-4) = 1.1284

**Newton’s Backward Difference Interpolation**

If the interpolating nodes are reordered as xn , xn-1, …x0, Newton’s divided difference polynomial can be written as

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

Let us introduce the notation h = xi+1-xi for each i=0,1,……n-1

Suppose that x = xn+sh

Since, xk = xn+(k-n)h

* x-xk = (s+n-k)h

This gives

x-x1 = (s+n-1)h

x-x2 = (s+n-2)h

…………………………

x-xn-1 = (s+n-n+1)h = (s+1)h

x-xn = (s+n-n)h = sh

Now, Newton’s interpolation divided difference formula becomes

pn(x) = f[xn]+f[xn, xn-1]sh+ f[xn,xn-1,xn-2]s(s+1)h2+ ….. +f[xn,xn-1,……x1,x0](x-xn)s(s+1)(s+2)……(s+n-1)hn

Newton backward difference formula is constructed by making use of the backward difference notation ∇

With this notation,

f[xn,xn-1] = = ∇f(xn)

f[xn,xn-1,xn-2] = = [∇f(xn)- ∇f(xn-1)] = ∇2f(xn)

And in general,

f[xn,xn-1,..xn-k] = ∇kf(xn)

Thus, interpolation divided difference formula can be written as

pn(x) = pn(xn+sh] = f[xn]+ ∇f(xn)s+ ∇2f(xn)s(s+1)+ ….. + ∇kf(xn)s(s+1)(s+2)……(s+n-1)

This equation is called **Newton Gregory** backward difference formula

**Example:** Construct Newton’s backward difference table for data points in table below and then find the approximate the approximate value of sin (45) by sing Newton’s backward difference formula and fourth divided difference

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 10 | 20 | 30 | 40 | 50 |
| f(x) | 0.1736 | 0.3420 | 0.5000 | 0.6428 | 0.7660 |

**Solution:**

Here, h = xi+1-xi = 10

x = xn+sh 🡪 s = (45-50)/10 = -1/2

Now, the backward difference table can be constructed as follows

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Xi | F(xi) | ∇f(xn) | ∇2f(xn) | ∇3f(xn) | ∇4f(xn) |
| 10 | 0.1736 |  |  |  |  |
|  |  | 0.1684 |  |  |  |
| 20 | 0.3420 |  | -0.0152 |  |  |
|  |  | 0.1580 |  | 0.0048 |  |
| 30 | 0.5000 |  | -0.0152 |  | -0.0004 |
|  |  | 0.1428 |  | 0.0044 |  |
| 40 | 0.6428 |  | -0.0196 |  |  |
|  |  | 0.1232 |  |  |  |
| 50 | 0.7660 |  |  |  |  |

We know that Newton’s backward difference interpolation formula is given by

pn(x) = pn(xn+sh] = f[xn]+ ∇f(xn)s+ ∇2f(xn)s(s+1)+ ….. + ∇kf(xn)s(s+1)(s+2)……(s+n-1)

Thus,

P4(x) = p4(xn+sh] = f[xn]+ ∇f(xn)s+ ∇2f(xn)s(s+1) + ∇3f(xn)s(s+1)(s+2)+ ∇4f(xn)s(s+1)(s+2)(s+3)

Now putting x=45

P4(45) = p4(50-1/2\*10] = 0.7660+ 0.1232\*(-1/2)+ \* (0.0196)(-1/2)(-1/2+1)+1/6\*(0.0044)\*(-1/2)\*1/2\*3/2+1/24\*(0.0004)\*(-1/2)\*(1/2)\*3/2\*5/2 = 0.70659

Thus,

sin (45) = 0.70659

**Example 2:** The sales for the last five years are given table below. Find the 4th order polynomial that passes through the given data points and then use the polynomial to estimate the sales for the year 1979.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Year(x) | 1974 | 1976 | 1978 | 1980 | 1982 |
| Sales f(x) | 40 | 43 | 48 | 52 | 57 |

Solution:

Here h = xi+1-xi = 2

Since x = xn+sh 🡪 s = (s-xn)/ = (1979-1982)/2 = -3/2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Xi | F(xi) | ∇f(xn) | ∇2f(xn) | ∇3f(xn) | ∇4f(xn) |
| 1974 | 40 |  |  |  |  |
|  |  | 3 |  |  |  |
| 1976 | 43 |  | 2 |  |  |
|  |  | 5 |  | -3 |  |
| 1978 | 48 |  | -1 |  | 5 |
|  |  | 4 |  | 2 |  |
| 1980 | 52 |  | 1 |  |  |
|  |  | 5 |  |  |  |
| 1982 | 57 |  |  |  |  |

We know that Newton’s backward difference interpolation formula is given by

pn(x) = pn(xn+sh] = f[xn]+ ∇f(xn)s+ ∇2f(xn)s(s+1)+ ….. + ∇kf (xn)s(s+1)(s+2)……(s+n-1)

Thus,

P4(x) = p4(xn+sh] = f[xn]+ ∇f(xn)s+ ∇2f(xn)s(s+1) + ∇3f(xn)s(s+1)(s+2)+ ∇4f(xn)s(s+1)(s+2)(s+3)

Thus, fourth order polynomial is:

P4(x) = p4(xn+sh) = 57+ 5s+ ½\*1\*s(s+1) +1/6\*2\*s(s+1)(s+2)+ 1/24\*5\*s(s+1)(s+2)(s+3)

Now, put x = 1979

P4(1979) = p4(1982+(-3/2)\*2) = 57+ 5\*(-3/2)+ ½\*1\*(-3/2)\*(-3/2+1) +1/6\*2\*(-3/2)(-3/2+1)(-3/2+2)+ 1/24\*5\*(-3/2)(-3/2+1)(-3/2+2)(-3/2+3)

= 50.1172

Hence, estimated sales of year 1979 = 50.1172

**Algorithm for Newton’s Backward Difference Formula**

1. Start
2. Read numbers of points say n
3. Enter the value at which isolated value is required say xp
4. Read n data points
5. Set h = x1-x0 and s = (xp-s[n-1])/h
6. Calculate first backward difference as below
7. For i=n to n-1

bd[i] = fx[i];

End for

1. Set v = bd[n-1]
2. Calculate 2nd to nth backward difference as below

For i=1 to n-1

For k=1 to i

P = p\*(s+k-1)

End for

V = v+

Repeat p=1

End for

1. Print the interpolated value v at xp
2. Stop

**C Program for Newton Backward difference formula**

#include<stdio.h>

#include<conio.h>

int factorial(int n)

{

if(n<=1)

return 1;

else

return n\*factorial(n-1);

}

int main()

{

int n,i,j,k;

float v=0, p,xp,x[10],fx[10], bd[10],h,s;

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

scanf("%d",&n);

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

scanf("%f",&xp);

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

{

printf("Enter the value of x and fx at i=%d\n",i);

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

}

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]);

}

}

v = bd[n-1];

for(i=1;i<n;i++)

{

p=1;

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

{

p = p\*(s+k-1);

}

v = v+(bd[n-i-1]\*p)/factorial(i);

}

printf("Interpolation value = %f",v);

getch();

return 0;

}

Output:

First Run:

Enter number of points

4

Enter the value at which interpolated value is needed

1.15

Enter the value of x and fx at i=0

0.9 0.7833

Enter the value of x and fx at i=1

1.0 0.8412

Enter the value of x and fx at i=2

1.1 0.8912

Enter the value of x and fx at i=3

1.2 0.9320

Interpolation value = 0.912831

Second Run:

Enter number of points

5

Enter the value at which interpolated value is needed

1979

Enter the value of x and fx at i=0

1974 40

Enter the value of x and fx at i=1

1976 43

Enter the value of x and fx at i=2

1978 48

Enter the value of x and fx at i=3

1980 52

Enter the value of x and fx at i=4

1982 57

Interpolation value = 50.117188

**Spline Interpolation**

Spline interpolation is a form of interpolation where the interpolant is a special type of piecewise polynomial called spline. Spline interpolation is preferred over polynomial interpolation because the interpolation error can be made small even using low degree polynomials for the spline.

**Curve Fitting: Regression Analysis**

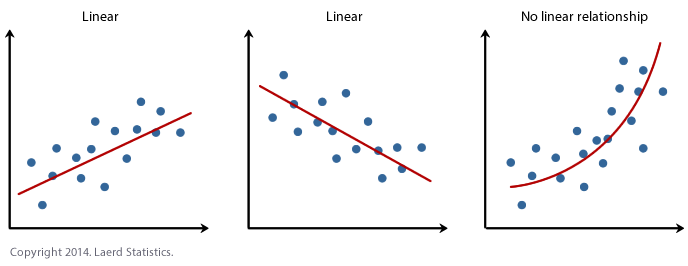
In many applications it often becomes necessary to establish a mathematical relationship between experimental values. This relationship may be used for either testing existing mathematical models or establishing new ones. The mathematical equation can also be used to predict or forecasts values of dependent variable. For example, we would like to know the maintenance cost of equipment as a function of age or the relationship between the literacy level and population growth. The process of establishing such relationships in the form of a mathematical equation is known as regression analysis or curve fitting.

Suppose the values of y for the different values of x are given. If we want to know the effect of x on y then we may write a functional relationship

Y = f(x)

The variable y is called the dependent variable and x the independent variable. The relationship may be either linear or non-linear. The type of relationship to be used should be decided by the experiment based on the nature of scatteredness of data.

Regression analysis is a form of predictive modeling technique which investigates the relationship between a dependent and independent variable. Dependent variables are also called target variables and independent variables are also called predictors. The goal of regression analysis is to express the dependent variable as a function of the independent variables.



**Interpolation vs. Regression**

In interpolation we are given with some data points, and we are supposed to find a curve which fits the input/output relationship perfectly. In case of interpolation, we don’t have worry about variance of the fitted curve. This means given the set of n data points (xi, yi) we look for the function f(x) that satisfies the relationship yi = f(xi) for all given data points.

When we do regression, we look for a function that minimizes some cost, usually sum of the squares of errors. We don’t require the function to have the exact values of given points, we just want a good approximation. In general, we search for the function y = f(x) that might not satisfy the relation yi = f(xi) for any given data points but the cost of function 2 will be the smallest of all the functions of given form.

**Parameter Estimation Methods**

Parameter estimation is the process of estimating values of parameters of the model based on the observed pairs of values and minimizing certain objective function. For example, consider the following variables and parameters

Dependent variable = y

Independent variable = x

Parameters = a, b

Dependent variable is linear with parameters. That is y = ax+b

Now, parameter estimation is used to find values of regression constants or parameters a and b by using given data points (x1,y1),(x2,y2)…. (xn,yn) such that some objective function or error function is minimized. A measure of goodness of fit of regression model is the magnitude of the residual error ei at each such of the n data points.

ei = yi-f(xi) i=0,1,2….n

Some methods of parameter estimation are as below

1. Minimize the sum of errors

That is minimize = for i=0,1,2,

1. Minimize the sum of absolute values of errors

That is minimize| | = | for i=0,1,2,

1. Minimize the sum of square of errors

That is minimize 2 = 2 for i=0,1,2,

Usually, first two methods are not used for parameter estimation because they do not yield unique line through the given data set. The reason behind this is that constants of the model can be chosen such that the average residual is zero without making individual residual small. But the third method will yield unbiased parameters with the smallest variance

**Least Square Method**

This is the most popular method of parameter estimation for coefficients of regression models. It has well known probability distributions and gives unbiased estimation of regression parameters with smallest variance. Ideally, if all the residuals ei are zero, one may have found an equation in which all the points lie on a model. Thus, minimization of the residual is an objective of obtaining regression coefficients. In least squares method, estimates of the constants of the models are chosen such that minimization of the sum of the squared residuals is achieved. In this method we need to minimize following objective function or error function

2 = 2 for i=0,1,2,……..

**Linear Regression**

Fitting a straight line is simplest approach of regression analysis which is called linear regression. A straight line can be represented by using the mathematical equation y = f(x) = a+bx. Where a and b are regression coefficients to be determined.

Let us sum of squares of individual errors can be expressed as

2 = 2 =2 for i=0,1,2,

We should choose regression coefficients a and b such that E is minimized. Necessary and sufficient condition for E to be minimum is

= 0 and

= 2(-1) = 0

And

2(-xi) = 0

Simplifying above equations, we get

na+b =

a+b2=

These equations are called normal equations. Solving for a and b, we get

b =

a = - b

**Algorithm**

1. **Start**
2. Read number of points, say n
3. Read given data points say x[i] and y[i]
4. Find summations of x, y, xy and x2 as below

For i=0 to n-1

sx = sx+x[i]

sy = sy+y[i]

sxy = sxy+x[i]\*y[i]

sx2 = sx2+x[i]\*x[i]

End for

1. Calculate value of a and b as

B = ((n\*sxy)-(sx\*sy)/((n\*sx2)-(sx\*sx)) and b = (sy/n)-(b\*sx/n)

1. Print equation y = a+bx
2. Stop

**C program for linear regression**

#include<stdio.h>

#include<conio.h>

int main()

{

int i,j,k,n;

float a=0,b=0, x[10],y[10],sx=0,sy=0,sxy=0,sx2=0;

printf("Enter 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],&y[i]);

}

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

{

sx = sx+x[i];

sy = sy+y[i];

sxy = sxy+x[i]\*y[i];

sx2 = sx2+x[i]\*x[i];

}

b = ((n\*sxy)-(sx\*sy))/((n\*sx2)-(sx\*sx));

a = (sy/n)-(b\*sx/n);

printf("Fitted line is %f+%fx",a,b);

getch();

}

Output:

Enter number of points

5

Enter value of x and fx

1 3

2 5

3 7

4 10

5 12

Fitted line is 0.500000+2.300000x

**Example 1:** Fit a straight line to the following set of data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 1 | 2 | 3 | 4 | 5 |
| Y | 3 | 4 | 5 | 6 | 8 |

Solution: The various summations are computed as follows

|  |  |  |  |
| --- | --- | --- | --- |
| Xi | Yi | xi2 | xi.yi |
| 1 | 3 | 1 | 3 |
| 2 | 4 | 4 | 8 |
| 3 | 5 | 9 | 15 |
| 4 | 6 | 16 | 24 |
| 5 | 8 | 25 | 40 |
| 15 | 26 | 55 | 90 |

Now b = = = 1.20

a = - b = - 1.20\* = 1.60

Therefore, the straight line that best fits through given data points is y = 1.6+1.2x

**Example 2:** Fit a straight line that best fits the following set of data by using linear regression

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 1 | 2 | 3 | 4 | 5 |
| F(X) | 3 | 5 | 7 | 10 | 12 |

Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| Xi | Yi | xi2 | xi.yi |
| 1 | 3 | 1 | 3 |
| 2 | 5 | 4 | 10 |
| 3 | 7 | 9 | 21 |
| 4 | 10 | 16 | 40 |
| 5 | 12 | 25 | 60 |
| = 15 | = 37 | 2 = 55 | = 134 |

Now b = = = 2.3

a = - b = - 2.3\* = 0.5

Therefore, the straight line that best fits through given data points is y = 2.3+0.5x

Example 3: The values of y and their corresponding values of y are shown in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| X | 0 | 1 | 2 | 3 | 4 |
| y = F(X) | 2 | 3 | 4 | 5 | 6 |

1. Find the least square regression line y = ax+b
2. Estimate the value of y when x = 10

Solution

|  |  |  |  |
| --- | --- | --- | --- |
| Xi | Yi | xi2 | xi.yi |
| 0 | 2 | 1 | 3 |
| 1 | 3 | 4 | 10 |
| 2 | 5 | 9 | 21 |
| 3 | 5 | 16 | 40 |
| 4 | 12 | 25 | 60 |
| = 15 | = 37 | 2 = 55 | = 134 |

Now b = = = 2.3

a = - b = - 2.3\* = 0.5

Therefore, the straight line that best fits through given data points is y = 2.3+0.5x

**Non Linear Regression**

Nonlinear regression is a form of regression analysis in which observational data are modeled by a function which is a nonlinear combination of the model parameters and depends on one or more independent variables. Nonlinear regression model can be described in two ways

1. By Fitting Polynomial Model
2. By fitting Exponential Model

**By Fitting Polynomial Model**

Polynomial regression is a form of regression analysis in which the relationship between the independent variable x and dependent variable y is modeled as an nth degree polynomial in x. let n data points are (x1,x2),(x2,y2)….(xn,yn) and we want to fit an mth order polynomial through the given data points. The general form f polynomial of degree m is given as below

y = a0+ a1x+a2x2+a3x3+….amxm , m<n

Now, the residual at each point is given by

ei = yi- a0+ a1xi+a2xi2+a3xi3+….amxim

The sum of the squares of the residuals is given by

E =

m)2

To find the constants of the polynomial regression model, we equate the derivatives with respect to ai to zero as shown below

= m)2(-1)=0

= m)2(-xi)=0

= m)2(-xi2)=0

…………………………………………………………………………..

…………………………………………………………………………..

= m)2(-xim)=0

This gives

na0+ a1 + a22+……………………….am m=

a0+ a12+ a23+……………………….am m= xi

………………………………………………………………………………………………………….

a0 m+ a1 m+1+ a2m+2+……………………….am 2m= xm

Solving these equations, we get values of ai s

Example: Fit a second order polynomial to the data in the table below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| X | 1.0 | 2.0 | 3.0 | 4.0 |
| Y | 6.0 | 11.0 | 18.0 | 27.0 |

Solution:

The order of the polynomial is 2 therefore we will have 3 simultaneous equations as shown below:

na1+ a2 + a3 2=

a1 + a2 2 +a3 3=

a12 +a23 +a3 4= xi2

The sum of powers and products can be evaluated in a tabular form as shown below

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X | Y | x2 | x3 | x4 | xy | x2y |
| 1 | 6 | 1 | 1 | 1 | 6 | 6 |
| 2 | 11 | 4 | 8 | 16 | 22 | 44 |
| 3 | 18 | 9 | 27 | 81 | 54 | 162 |
| 4 | 27 | 16 | 64 | 256 | 108 | 432 |
| 10 | 62 | 30 | 100 | 354 | 190 | 644 |

Substituting these values, we get

4a1+10a2+30a3 = 62

10a1+30a2+100a3 = 190

30a1+100a2+353a3 = 644

Solving, these equations we get

a1 = 3

a2 = 2

a3 = 1

Therefore, the least square quadratic polynomial is

y = 3+2x+x2

**Algorithm for polynomial Regression**

1. Start
2. Read number of points say n
3. Read the order of polynomial say m
4. Read given data points say x[i] and fx[i]
5. Calculate required summations as below

for i=1 to 2m

for j=0 to n-1

sx[i] = sx[i]+pow(x[i],i);

End for

for i=0 to m

for j=0 to n-1

sxy [i] = y[j]\*pow(x[i],i);

End for

End for

1. Construct RHS matrix of order (m+1)×(m+1)
2. Construct LHS matrix of order (m+1)×1
3. Solve for coefficients a0,a1,a2,..am using gauss elimination method
4. Display the equation y = a0+a1x+a2x2+….+amxm
5. Stop

**C program to implement the polynomial regression**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int i,j,k,m,n;

float a[20][20],b[20],z[20],x[20],fx[20];

float sum, pivot, term;

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

scanf("%d",&n);

printf("Enter degree of polynomial\n");

scanf("%d",&m);

printf("Enter data points\n");

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

{

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

}

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

{

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

{

sum=0;

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

{

sum = sum+pow(x[k],i+j);

}

a[i][j]=sum;

}

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

{

sum=0;

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

{

sum = sum+fx[i]\*pow(x[k],i);

}

b[i]=sum;

}

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

{

pivot = a[k][k];

if(pivot<0.0001)

printf("Method failed");

else

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

{

term = a[i][k]/pivot;

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

{

a[i][j]=a[i][j]-a[k][j]\*term;

}

b[i]=b[i]-b[k]\*term;

}

}

}

z[m] = b[m]/a[m][m];

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

{

sum=0;

for(j=i+1;j<=2;j++)

{

sum = sum+a[i][j]\*z[j];

}

z[i]=(b[i]-sum)/a[i][i];

}

printf("The polynomial of regression is :\n");

printf("y=%f+%fx",z[0],z[1]);

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

{

printf("+%fx^%d",z[i],i);

}

getch();

return 0;

}

**Fitting Exponential Model**

An exponential model is a model described by the equation y = aex. In this equation the coefficients a and b are constants of the exponential model. Non linear regression model can be obtained by fitting exponential through the given data points (x1, y1),(x2, y2),(x3, y3)….(xn, yn). When we fit exponential model through given data set, iterative methods are required to estimate the values of the model parameters. This can be time consuming and tedious which can be done easily by calculating logarithms of both sides.

We know that, exponential model is given by

y= aebx ….(1)

Taking natural log of both sides we get

Logy = log(aebx)

Or logy = log a+ bx ……………………(2)

This equation is similar to the form of linear equation y = a+ bx. This can evaluate parameters a and b by using the equation of linear regression model as shown below

b = …………………(3)

and loga = - b ……………….(4)

Let R = - b

Now equation (4) becomes

Log a = R

Taking antilog on both sides we get

a = eR ……………(6)

Now we can calculate values of regression coefficients a and b easily by using equations (3) and (6)

Algorithm for Nonlinear Regression with Exponential Model

1. Start
2. Read number of data points say n
3. Read given data points say x[i], y[i]
4. Calculate needed summation as below

sx = sx+x[i];

slogy = slogy+logy[i];

sxy = sxy+x[i]\*logy[i];

sx2 = sx2+x[i]\*x[i];

End for

1. Calculate values of b and a by using following formulae

a = ((n\*sxy)-(sx\*slogy))/((n\*sx2)-(sx\*sx))

r = (slogy)/n)-(b\*sx/n)

a = er

1. Display the equation y = aebx
2. Stop

**C program for nonlinear Regression with exponential Model**

#include<stdio.h>

#include<conio.h>

#include<math.h>

int main()

{

int n,i,j,k;

float a=0,b=0,r, x[10],y[10],sx, slogy=0, sxy=0, sx2=0;

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

scanf("%d",&n);

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

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

{

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

}

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

{

sx = sx+x[i];

slogy = slogy+log(y[i]);

sxy = sxy+x[i]\*log(y[i]);

sx2 = sx2+x[i]\*x[i];

}

b = ((n\*sxy)-(sx\*slogy))/((n\*sx2)-(sx\*sx));

r = (slogy/n)-(b\*sx/n);

a = exp(r);

printf("Fitted Curve is y = %fe^%fx",a,b);

getch();

}

Output:

Enter number of points

5

Enter the value of x and fx

2 4.077

4 11.084

6 30.128

8 81.897

10 222.62

Fitted Curve is y = 1.499900e^0.500009x

**Example:** Below is given the relative intensity of radiations as a function of time

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| X(time) | 0 | 1 | 3 | 5 | 7 | 9 |
| Y=intensity of radiation | 1.00 | 0.891 | 0.708 | 0.562 | 0.447 | 0.355 |

If the level of the relative intensity of radiation is related to time via an exponential formula y = aebx, find the value of the regression constants a and b. Also estimate intensity of radiation at time t = 13

Solution:

We know that exponential model is given by

y = aebx

Or log y = log a+ bx

This equation is similar in form to he linear equation y = a+bx. Thus we can evaluate parameters a and b by using the equation of linear regression model as below:

b =

and loga = - b

Now we calculate summation table as below

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| I | Xi | yi | log yi | xilogyi | xi2 |
| 1 | 0 | 1 | 0.00000 | 0.0000 | 0.00 |
| 2 | 1 | 0.891 | -0.11541 | -0.1154 | 1.00 |
| 3 | 3 | 0.708 | -0.34531 | -1.0359 | 9.00 |
| 4 | 5 | 0.562 | -0.57625 | -2.8813 | 25.00 |
| 5 | 7 | 0.447 | 0.80520 | -5.6364 | 49.00 |
| 6 | 9 | 0.355 | -0.0356 | -9.3207 | 81.00 |
|  | 25 |  | -2.878 | -18.99 | 165 |

Now,

b = = -0.115

log a = - (-0.115)\*25/6 = -0.0005

Since log a = -0.0005

**a** = e-0.0005 = 0.999

Thus the regression formula is

y = 0.9999e0.115x