Chapter VI

Least Squares Fitting of Discrete Points

The emphases in the previous chapters were on C++ programming along with some numerical schemes. We will continue to add more details regarding the C++ programming language and from time to time introduce numerical schemes. In this chapter we will examine two new features of C++:

- The **this** pointer.
- Static members of a class.

Prior to doing this we will first examine Least Squares Fitting.

6.1 Least Squares Straight line Fitting

In least-squares straight line fitting we attempt to fit a straight line to a given set of data (x_i,y_i) tabulated as shown:

x	x_0	x_1	x_2	...	x_{n-1}
y	y_0	y_1	y_2	...	y_{n-1}

where x is the independent variable and y the dependent variable. That is an experiment is carried-out where y is measured for each value of x.

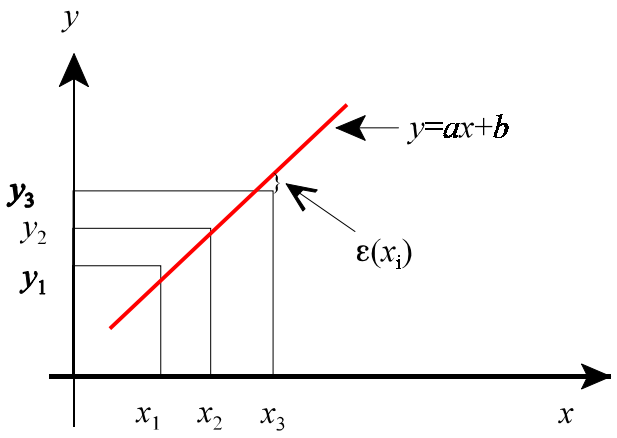

Figure 5.1 Least squares fitting.

Assuming a straight line is suitable to model the data set shown in Fig.1 we begin our development by forming the error between the straight line and an ordinate at x_i

$$\varepsilon(x_i) = ax_i + b - y_i$$

Sum the square of the errors:

$$E = \sum_{i=0}^{n-1} \varepsilon^2(x_i)$$
$$= \sum_{i=0}^{n-1} (ax_i + b - y_i)^2$$

In order to minimize E with respect to the coefficients (a, b) we differentiate E with respect to each coefficient and equate the results to zero as follows:

$$\frac{\partial E}{\partial a} = 2\sum_{i=0}^{n-1} (ax_i + b - y_i)x_i = 0$$
$$\frac{\partial E}{\partial b} = 2\sum_{i=0}^{n-1} (ax_i + b - y_i) = 0$$

This yields the two linear simultaneous equations in a and b:

$$\begin{bmatrix} \sum_{i=0}^{n-1} x_i^2 & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n} x_i & n \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n-1} y_i x_i \\ \sum_{i=0}^{n-1} y_i \end{bmatrix}$$

Applying Cramer's rule we get.

$$a = \frac{n\sum_{i=0}^{n-1} y_i x_i - \sum_{i=0}^{n-1} y_i \sum_{i=0}^{n-1} x_i}{\Delta}$$

$$b = \frac{\sum_{i=0}^{n-1} x_i^2 \sum_{i=0}^{n-1} y_i - \sum_{i=0}^{n-1} x_i \sum_{i=0}^{n-1} y_i x_i}{\Delta}$$

$$\text{where} \quad \Delta = n\sum_{i=0}^{n\text{-}1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2$$

This is sometimes referred as linear regression.

Example. Use linear straight line regression to fit the following data set:

x_i	0	1	2	3	5
y_i	0	1.4	2.2	3.5	4.4

Solution

The following table provides the various sums:

i	x_i	y_i	$x_i y_i$	x_i^2
0	0	0	0	0
1	1	1.4	1.4	1
2	2	2.2	4.4	4
3	3	3.5	10.5	9
4	5	4.4	22.0	25
$\sum$	11	11.5	38.3	39

Substituting the sums in the equations for linear regression we get:

$$a = \frac{5(38.3) - 11.5 \times 11}{5 \times 39 - 11^2} = 0.878$$

$$b = \frac{39 \times 11.5 - 11 \times 38.3}{74} = 0.368$$

Consequently, the line that can best fit in a least squares sense the above data is:

$$y = 0.878x + 0.368$$

Example. The distance required to stop an automobile is a function of its speed. The following experimental data was collected to quantify this relationship:

Speed, mi/h	15	20	25	30	40	50	60	70
Stopping distance, ft	23	27	32	37.5	52.5	73.5	103	144

Estimate the stopping distance for a car traveling at 90 mi/h. Use linear least-squares fit and assume an exponential model:

$$y = \alpha e^{\beta x}$$

Solution.
Taking the natural logarithm of both sides we get:

$$\ln\ y = \beta x + \ln \alpha$$

Therefore the problem is reduced to a straight linear regression problem with y being replaced by ln(y) and a being replaced by ln(α). Hence, to solve the problem the following table is formed (note that I used a spread sheet to calculate the various columns and hence the many significant digits.):

i	x	y	ln y	x ln y	x^2
0	15	23	3.13549421593	47.0324132389	225
1	20	27	3.295836866	65.9167373201	400
2	25	32	3.4657359028	86.64339757	625
3	30	37.5	3.62434093298	108.730227989	900
4	40	52.5	3.9608131696	158.432526784	1600
5	50	73.5	4.29728540622	214.864270311	2500
6	60	103	4.63472898823	278.083739294	3600
7	70	144	4.96981329958	347.88693097	4900
∑	310		31.3840487813	1307.59024348	14750

Applying the best fit equations we get:

$$a = \frac{8\times 1307.6 - 31.38\times 310}{8\times 14750 - (310)^2} = \frac{733}{21900} = 0.0335$$

$$b = \frac{14750\times 31.4 - 310\times 1307.56}{21900} = 2.64$$

$$\alpha = e^{2.49} = 14, \qquad \beta = 0.0335$$

$$y = 14e^{0.0335x}$$

At 90 mi/hr the car is estimated to stop at a distance of

$$y = 14e^{0.0335\times 90} \approx 286 \text{ ft.}$$

Example. Develop a C++ program for straight line linear regression and test it on the above two examples.

Solution

To develop a C++ program for such an application we first have to develop a class for the straight linear regression and then use that class in the main function to solve the above two examples as follows.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>

class LnReg
   {
   private:
      long double *x;
      long double *y;
      int n;
   public:
       LnReg(int nd=5); //Constructor
      ~LnReg();   //Destructor
      void SetData(int i, long double xd, long double yd);
      void Solv(long double &a, long double &b);
       };

LnReg::LnReg(int nd)
 {
```

```
  n=nd;
  x=new long double[n];
  y=new long double[n];
  }

LnReg::~LnReg()
   {
   delete []x;
   delete []y;
   }

void LnReg::SetData(int i, long double xd, long double yd)
  {
  x[i]=xd;
  y[i]=yd;
  }

void LnReg::Solv(long double &a, long double &b)
   {
   long double sumx=0, sumy=0, sumx2=0, sumxy=0;

   for(int i=0; i<n; i++)
     {
     sumx+=x[i];
     sumy+=y[i];
     sumx2+=x[i]*x[i];
     sumxy+=x[i]*y[i];
     }
    long double D=n*sumx2-sumx*sumx;
    a=(n*sumxy-sumy*sumx)/D;
    b= (sumx2*sumy-sumx*sumxy)/D;
    }

//---------------------------------------------------------------------------

int main()
{
   {
// First example
 long double x[]= {0, 1, 2, 3, 5};
 long double y[]= {0, 1.4, 2.2, 3.5, 4.4};
 LnReg StLn(5);
 int i;
```

```
long double a,b;

for(i=0; i<5; i++)
  StLn.SetData(i,x[i],y[i]);

StLn.Solv(a,b);
cout << "Solution: " << endl;
cout << "a = " << a << endl
     << "b = " << b << endl;

cout << " or " << endl;
cout << "    y = " << a << "*x " << "+ " << b << endl;
      }  //blocking off the first example
// Second example
long double x[]={15, 20, 25, 30, 40, 50, 60, 70};
long double y[]={23, 27, 32, 37.5, 52.5, 73.5, 103, 144};

LnReg StLn(8);
int i;
long double a,b;

for(i=0; i<8; i++)
 StLn.SetData(i,x[i], logl(y[i]));

 StLn.Solv(a,b);

long double alpha=expl(b);
long double beta=a;
cout << "Solution: ";
cout << "alpha = " << alpha << endl;
cout << "beta  = " << beta << endl;

cout << "  or" << endl;
cout << "y = " << alpha << "exp( " << beta << " x )" << endl;

cout << "Stopping distance at 90 mi/hr = " ;
cout << alpha*exp(beta*(90)) << " feet" << endl;
cout << endl;

cout << "speed    stopping distance     actual stopping distance " << endl;
for(i=0; i<8; i++)
  {
 cout << x[i];
```

```
 cout.width(20); cout.precision(5); cout.left;
 cout << alpha*expl(beta*x[i]);
 cout.width(15); cout.precision(5); cout.left;
 cout << "          " << y[i] << endl;
      }
getch();
return 1;
 }
```

6.2 Least Squares Polynomial Fitting

Assume an m[th] order polynomial

$$y = a_0 + a_1 x + a_2 x^2 + \ldots\ldots + a_m x^m$$

The difference between the calculated value of an ordinate at x_i and the actual measured value of the ordinate at the same point form the error function:

$$\varepsilon(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m - y_i$$

Form the sum of errors squared:

$$E = \sum_{i=0}^{n-1} \left(\varepsilon(x_i)\right)^2$$

Differentiating with respect to the coefficients we get:

$$\frac{\partial E}{\partial a_0} = 2\sum_{i=0}^{n-1} (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m - y_i) \times 1 = 0$$

$$\frac{\partial E}{\partial a_1} = 2\sum_{i=0}^{n-1} (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m - y_i) \times x_i = 0$$

$$\vdots$$

$$\frac{\partial E}{\partial a_k} = 2\sum_{i=0}^{n-1} (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m - y_i) \times x_i^k = 0$$

$$\vdots$$

$$\frac{\partial E}{\partial a_m} = 2\sum_{i=0}^{n-1} (a_0 + a_1 x_i + a_2 x_i^2 + \ldots + a_m x_i^m - y_i) \times x_i^m = 0$$

Rearranging and placing the set of linear algebraic equations in matrix form we get:

$$\begin{bmatrix} n & \sum x_i & \sum x_i^2 & \dots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \dots & \sum x_i^{m+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \dots & \sum x_i^{m+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \sum x_i^m & \sum x_i^{m+1} & \sum x_i^{m+2} & \dots & \sum x_i^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \\ \vdots \\ \sum y_i x_i^m \end{bmatrix}$$

This approach is sometimes referred to as polynomial linear regression.

Example. Using linear regression determine the coefficients of a second order polynomial, m=2, to approximate the following data set:

x_i	0	1	2	3	4	5	6	7	8
y_i	4	5	10	17	21	16	11	3	1

Solution.

I will leave the working out of the details to the reader. The coefficients for the second order polynomial obtained by solving the resultant 3 equations are:

$$a_0 = 1.29 \qquad a_1 = 7.97 \qquad a_2 = -1.03$$

We now add a few more points to our C++ programming.

6.3 The this pointer

Just to clarify matters this subject has nothing to do with curve fitting. Since our intention is to provide a balanced presentation of C++ and numerical techniques suitable for a first course on the subject we have to somehow present both subjects (C++ and numerical techniques) at a reasonable pace suitable for a one semester course. Presenting all the details of OOP in one go could be a strain on some. Hence after the initial presentation on C++ given in the previous chapters we will add some more details as we proceed through the various numerical schemes.

In member functions that returns a reference to, or pointer to the class, you will encounter the **this** pointer. So what is **this** pointer? The **this** pointer is a special pointer that is accessible to only to member functions. When you call a member function for an object, the compiler assigns the address of the object to the **this** pointer and then calls the function. The **this** pointer thus always points to the objects for which the member function is called and is not accessible outside the class. To illustrate how **this** works we will re-write the previous program as follows:

```
#include <iostream.h>
#include <conio.h>
```

```
#include <math.h>

class LnReg
   {
   private:
      long double *x;
      long double *y;
      int n;
   public:
       LnReg(int nd=5); //Constructor
      ~LnReg();   //Destructor
      void SetData(int i, long double xd, long double yd);
      LnReg& Solv(long double &a, long double &b);
      void copy(LnReg& STLN);
      long double getx(i)
        {
        return x[i];
        }
      long double gety(i)
        {
        return y[i];
        }
       };

LnReg::LnReg(int nd)
  {
  n=nd;
  x=new long double[n];
  y=new long double[n];
  }

void LnReg::copy(LnReg & STLN)
  {
  //delete the current x and y
  delete []x;
  delete []y;
  //Copy the new size
  n=STLN.n;

  //Allocate memory according to the new size
  x=new long double[n];
  y=new long double[n];
```

```
 //Copy the values x and y from STLN to the reallocated object
 for(int i=0; i<n; i++)
  {
  x[i]=STLN.x[i];
  y[i]=STLN.y[i];
  }
 }

LnReg::~LnReg()
  {
  delete []x;
  delete []y;
  }

void LnReg::SetData(int i, long double xd, long double yd)
 {
 //just to prove the point that "this" is a pointer
 //assigned by the compiler.
 this->x[i]=xd;
 this->y[i]=yd;
 }

LnReg& LnReg::Solv(long double &a, long double &b)
  {
  long double sumx=0, sumy=0, sumx2=0, sumxy=0;

  for(int i=0; i<n; i++)
    {
    sumx+=x[i];
    sumy+=y[i];
    sumx2+=x[i]*x[i];
    sumxy+=x[i]*y[i];
    }
   long double D=n*sumx2-sumx*sumx;
  a=(n*sumxy-sumy*sumx)/D;
  b=(sumx2*sumy-sumx*sumxy)/D;
   return *this;  //Since we returning a reference
                  //to the class we use "this"
   }

//---------------------------------------------------------------------------

int main()
```

```
{
// First example
 long double x[]= {0, 1, 2, 3, 5};
 long double y[]= {0, 1.4, 2.2, 3.5, 4.4};
 LnReg StLn(5);
 int i;
 long double a,b;

 for(i=0; i<5; i++)
   StLn.SetData(i,x[i],y[i]);

 StLn.Solv(a,b);
 cout << "Solution: " << endl;
 cout << "a = " << a << endl
      << "b = " << b << endl;

 cout << " or " << endl;
 cout << "    y = " << a << "*x " << "+ " << b << endl;

  // Second example
 long double x2[]={15, 20, 25, 30, 40, 50, 60, 70};
 long double y2[]={23, 27, 32, 37.5, 52.5, 73.5, 103, 144};

 LnReg StLn2(8);

 for(i=0; i<8; i++)
  StLn2.SetData(i,x2[i], logl(y2[i]));

 StLn.copy(StLn2.Solv(a,b)); //Solve and copy StLn2 to StLn

 long double alpha=expl(b);
 long double beta=a;
 cout << "Solution: ";
 cout << "alpha = " << alpha << endl;
 cout << "beta  = " << beta << endl;

 cout << "  or" << endl;
 cout << "y = " << alpha << "exp( " << beta << " x )" << endl;

 cout << "Stopping distance at 90 mi/hr = " ;
 cout << alpha*exp(beta*(90)) << " feet" << endl;
 cout << endl;
```

```
 cout << "speed    stopping distance     actual stopping distance " << endl;
 for(i=0; i<8; i++)
   {
//Note that since we copied StLn2 to StLn we can now
//refer to StLn to obtain x and ln(y)
  cout << StLn.getx(i);
  cout.width(20); cout.precision(5); cout.left;
  cout << alpha*expl(beta*StLn.getx(i));
  cout.width(15); cout.precision(5); cout.left;
  cout << "           " << exp(StLn.gety(i)) << endl;
    }
 getch();
 return 1;
   }
```

.

The main changes are highlighted in red. Notice the following:

- Member variables are actually pointed at by the **this** pointer. The programmer does not have to use **this** pointer explicitly since the compiler will handle it internally for each object.
- When a function returns a reference to the class (examine the above copy member function) then **this** is used in the return statement.
- In the main program in the second example you will notice the following statement:

  StLn.copy(StLn2.Solv(a,b)); //Solve and copy StLn2 to StLn

  This Solves for a and b and copies the values of x, y and n of StLn2 to StLn1.

For this particular example we could have avoided the use of the this pointer. However, later on we will find a need for such a pointer. The only thing that the program does is illustrates the presence of the **this** pointer. Hence it goes without saying that **this** is a reserved word that you cannot use for one of your variables.

6.4 Static Members

Data members declared as static are data that belong to the class rather any class instance. These can be used to track the number of class instances as we will demonstrate later in a program. The rules regarding static data members are as follows:

- A static data member is declared by placing the keyword **static** before the member's data type.
- Static data members can be accessed inside member functions.
- A static data member must be initialized outside the class declaration.
- Static data members exists separately from class instances, and hence you can access them before you create any class instance.

Let us now put static variables to the test. We will begin with the initial program we have developed

for linear regression and add to it a static variable to count the number of instances of the class. In order to access the static variable we will need to add a public member function to return the value of the static variable. To count the number of instances the static variable has to be incremented in the constructed and decremented in the destructor. The following program provides the changes needed on class LnReg to accommodate a static variable for instance counting.

```
#include <iostream.h>
#include <conio.h>
#include <math.h>

class LnReg
   {
   private:
      long double *x;
      long double *y;
      int n;
      static unsigned countInstances;   //static variable
   public:
       LnReg(int nd=5); //Constructor
      ~LnReg();   //Destructor
      void SetData(int i, long double xd, long double yd);
      void Solv(long double &a, long double &b);
      static unsigned getCountInstances()
        {
        return countInstances;
         }
       };

LnReg::LnReg(int nd)
  {
  countInstances++;
  n=nd;
  x=new long double[n];
  y=new long double[n];
  }

LnReg::~LnReg()
   {
   countInstances--;
   delete []x;
   delete []y;
   }
```

```
void LnReg::SetData(int i, long double xd, long double yd)
  {
  x[i]=xd;
  y[i]=yd;
  }

void LnReg::Solv(long double &a, long double &b)
   {
   long double sumx=0, sumy=0, sumx2=0, sumxy=0;

   for(int i=0; i<n; i++)
     {
     sumx+=x[i];
     sumy+=y[i];
     sumx2+=x[i]*x[i];
     sumxy+=x[i]*y[i];
     }
    long double D=n*sumx2-sumx*sumx;
    a=(n*sumxy-sumy*sumx)/D;
    b= (sumx2*sumy-sumx*sumxy)/D;
    }

// Initialize static variable
 unsigned LnReg::countInstances=0;
//------------------------------------------------------------------------

int main()
{
   {
// First example
 long double x[]= {0, 1, 2, 3, 5};
 long double y[]= {0, 1.4, 2.2, 3.5, 4.4};
 LnReg StLn(5);
 cout << " Number of instances = " << StLn.getCountInstances() << endl;
 int i;
 long double a,b;

 for(i=0; i<5; i++)
   StLn.SetData(i,x[i],y[i]);

 StLn.Solv(a,b);
 cout << "Solution: " << endl;
 cout << "a = " << a << endl
```

```
     << "b = " << b << endl;

 cout << " or " << endl;
 cout << "    y = " << a << "*x " << "+ " << b << endl;
      }  //blocking off the first example
// Second example
 long double x[]={15, 20, 25, 30, 40, 50, 60, 70};
 long double y[]={23, 27, 32, 37.5, 52.5, 73.5, 103, 144};

 LnReg StLn(8);
 cout << " Number of instances = " << StLn.getCountInstances() << endl;
 int i;
 long double a,b;

 for(i=0; i<8; i++)
  StLn.SetData(i,x[i], logl(y[i]));

  StLn.Solv(a,b);

 long double alpha=expl(b);
 long double beta=a;
 cout << "Solution: ";
 cout << "alpha = " << alpha << endl;
 cout << "beta  = " << beta << endl;

 cout << "  or" << endl;
 cout << "y = " << alpha << "exp( " << beta << " x )" << endl;

 cout << "Stopping distance at 90 mi/hr = " ;
 cout << alpha*exp(beta*(90)) << " feet" << endl;
 cout << endl;

 cout << "speed    stopping distance     actual stopping distance " << endl;
 for(i=0; i<8; i++)
   {
  cout << x[i];
  cout.width(20); cout.precision(5); cout.left;
  cout << alpha*expl(beta*x[i]);
  cout.width(15); cout.precision(5); cout.left;
  cout << "           " << y[i] << endl;
       }
 getch();
 return 1;
```

```
}
```

In the above program notice how the static variable is initialized outside the class. The scope operator along with the class name are used to identify the class in which the static variable belongs. The first example in main is blocked-in by two brackets and hence the variables and instances declared within that bracket are removed from memory once the execution of the program proceeds outside the block. A print-out of the result is shown next:

```
 Number of instances = 1
Solution:
a = 0.878378
b = 0.367568
 or
   y = 0.878378*x + 0.367568
 Number of instances = 1
Solution: alpha = 13.8515
beta  = 0.0334094
  or
y = 13.8515exp( 0.0334094 x )
Stopping distance at 90 mi/hr = 280.126 feet

speed    stopping distance     actual stopping distance
15            22.863             23
20             27.02            27
25            31.933             32
30            37.738             37.5
40            52.708             52.5
50            73.616             73.5
60            102.82             103
70             143.6            144
```

Notice that the number of instances after the block remained 1 although we have began a new instance. The reason, of course, is due to the blocking of the first instance within the first example.

Problems

1. Derive a second-order approximation polynomial for the following data.

x_i	0	1	2	3	4	5	6	7	8
y_i	4	5	10	17	21	16	11	3	1

2. The following experimental data relate the current *I* to time *t* and sheet resistance *R* in a device. Develop a functional approximation of the form $I = a_1 + a_2 R + a_3 t$

I	5.3	7.8	7.6	9.7	10.5	12.6
R	66	85	70	140	95	125
t	1.5	2.6	0.6	1.3	2.7	1.6

3. Problems of the type given in the previous question are called multiple regression problems. In these type of problems more than one independent variable are present. Develop a C++ class to solve a multiple linear regression problem of the form $y = a_1 + a_2 x + a_3 y$. Test the program on the numerical example given in the previous question.

4. Find the curve fit $y = B\exp(Ax)$ to fit the following data.

x_k	-1	0	1	3
y_k	13.65	1.38	0.49	0.15

5. Find the curve fit $y = (Ax + B)^{-2}$ by linearizing the equation using the substitution $Y = 1/\sqrt{y}$ for the following data:

x_k	-1	0	1	2
y_k	13.45	3.01	0.67	0.15

6. Explain the **this** pointer used in C++ classes and show its presence in a class _Complex that we have developed in previous sections.

7. Static data members are used to track the number of instances of a class. Using the class Matrix we have developed previously show how you can track the number of instances of that class with a static data member.