Аппроксимация, или приближение — научный метод, состоящий в замене одних объектов другими, в том или ином смысле близкими к исходным, но более простыми.

Аппроксимация позволяет исследовать числовые характеристики и качественные свойства объекта, сводя задачу к изучению более простых или более удобных объектов (например, таких, характеристики которых легко вычисляются, или свойства которых уже известны).

Пусть, например, у нас есть множество точек в плоскости. Через них не всегда можно провести прямую. Но как поступить, если мы знаем, что данные точки не лежат на одной прямой лишь из-за неточности измерений. Можно ли узнать эту прямую? Ответ напрашивается сам собой: «Нужно провести аппроксимацию прямой по заданным точкам!» Остается выбрать метод, с которым мы будем работать.

§1 Аппроксимация прямой методом наименьших квадратов.

Метод заключается в том, что по набору точек мы восстанавливаем прямую, руководствуясь такой идеей. Сумма квадратов расстояний от прямой до точек должна быть минимальна.

Другими словами, дан набор точек при . Пусть

*Ф* = .

Необходимо провести прямую (найти коэффициенты ), так, чтобы *Ф* принимала минимальное значение в точке , при условии, что .

При данном условии Ф, будет суммой расстояний от прямой до точек . А задача поиска коэффициентов сводится к задаче условного экстремума, которую можно решить методом множителей Лагранжа.

Запишем функцию Лагранжа

*L* = – ().

Запишем частные производные

*,*

*,*

*,*

*.*

Вспомним, что точка функции называется особой, если , и неособой, если

Для того, чтобы неособая точка функции *Ф* была точкой условного экстремума этой функции необходимо, чтобы при некотором имело место равенство т.е. чтобы все частные производные функции *L* по переменным и обращались в нуль.

Для нахождения коэффициентов запишем систему и проведем ряд её преобразований

Выразим из третьего уравнения полученной системы и подставим его в первые два. Получим систему из двух уравнений.

Заметим, что вектор будет собственный для матрицы A, соответственно собственные значения. Где матрица А выглядит следующим образом.

Находим собственные векторы , подставляем данные в формулу и находим . В итоге имеем два набора коэффициентов выбираем тот набор на которых функция *Ф* принимает наименьшее значение.

Данную идею, однако, можно использовать не только для аппроксимации прямых , но и для аппроксимации плоскостей .

§2 Аппроксимация плоскости методом наименьших квадратов.

Дан набор точек при . Пусть

*Ф* =

Необходимо провести плоскость так, чтобы *Ф* принимала минимальное значение в точке , при условии, что .

При данном условии Ф будет суммой расстояний от плоскости до точек . А задача поиска коэффициентов сводится к задаче условного экстремума, которую также можно решить методом множителей Лагранжа, как это было проделано выше.

Запишем функцию Лагранжа

*L* = – ().

Запишем частные производные

Для того, чтобы неособая точка функции *Ф* была точкой условного экстремума этой функции необходимо, чтобы при некотором имело место равенство т.е. чтобы все частные производные функции *L* по переменным и обращались в нуль.

Для нахождения коэффициентов запишем систему и проведем ряд её преобразований

Выразим из четвертого уравнения полученной системы и подставим его в первые три. Получим систему из трёх уравнений.

что равносильно следующей системе:

Заметим, что вектор будет собственный для матрицы A, соответственно собственное значение. Где матрица А выглядит следующим образом.

Находим собственные векторы , подставляем данные в формулу и находим . В итоге имеем три набора коэффициентов выбираем тот набор на которых функция *Ф* принимает наименьшее значение.

Стоит заметить, что используя метод множителей Лагранжа можно накладывать не одно условие, как мы это делали до этого, а несколько. При этом возникнет еще одно уравнение связи. Этот факт можно использовать для решения следующей задачи.

§3 Нахождение плоскости, проходящей через данную точку и минимально отклоняющихся от остальных заданных точек.

Данa точка набор точек при . Пусть

*Ф* =

Необходимо провести плоскость так, чтобы *Ф* принимала минимальное значение в точке , при условии, что , и при этом искомая плоскость проходила через точку .

Воспользуемся полученными ранее навыками и запишем функцию Лагранжа

*L* = – () - ().

Запишем частные производные

*(1)*

*(2)*

*(3)*

*(4)*

*(5)*

*(6)*

Руководствуясь, теми же соображениями что и раньше запишем систему уравнений в которой все частные производные приравняем нулю. Для краткости записи можно сразу заметить, что из уравнения (4) следует, что , а из (6) получаем, что . Сделаем подстановку в уравнения (1)-(3) и запишем систему.

Упростим систему и сделаем замену

Заметим, что вектор будет собственный для матрицы A, соответственно собственное значение. Где матрица А выглядит следующим образом.

Находим собственные векторы , подставляем данные в формулу и находим . В итоге имеем три набора коэффициентов выбираем тот набор на которых функция *Ф* принимает наименьшее значение.

4 Численное тестирование

Чтобы на практике убедиться, что данные методы действенны и эффективны реализуем данные алгоритмы в *с++*. Далее программный код алгоритма и результаты тестов, которые показывают, что если множество точек лежит на прямой, то эта прямая восстанавливается с довольно большой точностью. Если же точки отличаются от тех, которые лежат на прямой на произвольную величину, то прямая тоже восстанавливается и точность её тем больше, чем больше точек и чем ближе они лежат к прямой. Стоит заметить, что на решение задачи не влияет ориентация прямой относительно осей координат. Прямая может, как пересекать оси, так и быть им параллельна.

#include <iostream>

#include <cmath>

#define eps 1e-20

void aprox(int n,double \* x,double \* y,double& a,double& b,double& c);

void jacobi ( int n, double \*\* a, double \* d, double \*\* v );

using namespace std;

int main (int argc, char\* argv[])

{

int n;

double \*x,\*y,a=0,b=0,c=0;

double a\_=0,b\_=0,c\_=0;

cout<<"Enter factor 'a' in a straight line ax + by + c = 0\n";

cin>>a;

cout<<"Enter factor 'b' in a straight line ax + by + c = 0\n";

cin>>b;

cout<<"Enter factor 'c' in a straight line ax + by + c = 0\n";

cin>>c;

if(b==0 && a==0)

{

cout<<"The incorrect task of a straight line\n";

return 1;

}

cout<<"Enter number of points for testing\n";

cin>>n;

cout

<<"Primary straight line"<<"("<<a<<")x + ("<<b<<")y + "<<c<<" = 0\n"

<<"Equivalent record"<<"("<<a/(sqrt(a\*a+b\*b))<<")x + ("<<b/(sqrt(a\*a+b\*b))<<")y + "<<c/(sqrt(a\*a+b\*b))<<" = 0\n";

FILE \*file1,\*file2;

file1 = fopen("output\_first\_test","w");

file2 = fopen("output\_second\_test","w");

fprintf(file1,"%d\n",n);

fprintf(file2,"%d\n",n);

x=new double[n];

y=new double[n];

//first test

if (b!=0)

{

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

{

x[i] = (double)(rand()%1000 -500)/10;

y[i] = -(a\*x[i] + c)/b;

fprintf(file1,"%lf %lf\n",x[i],y[i]);

}

aprox(n,x,y,a\_,b\_,c\_);

cout

<<"Straight line after the first test\n"

<<"("<<a\_<<")x + ("<<b\_<<")y + "<<c\_<<" = 0\n";

}

else if(a!=0)

{

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

{

y[i] = (double)(rand()%1000 -500)/10;

x[i] = -(b\*y[i] + c)/a;

fprintf(file1,"%lf %lf\n",x[i],y[i]);

}

fclose(file1);

aprox(n,x,y,a\_,b\_,c\_);

cout

<<"Straight line after the first test\n"

<<"("<<a\_<<")x + ("<<b\_<<")y + "<<c\_<<" = 0\n";

}

double err1 = 0;

if (err1< fabs(a/(sqrt(a\*a+b\*b))-a\_)) err1=fabs(a/(sqrt(a\*a+b\*b))-a\_);

if (err1< fabs(b/(sqrt(a\*a+b\*b))-b\_)) err1=fabs(b/(sqrt(a\*a+b\*b))-b\_);

if (err1< fabs(c/(sqrt(a\*a+b\*b))-c\_)) err1=fabs(c/(sqrt(a\*a+b\*b))-c\_);

cout<<" max\_error "<<err1<<"\n";

//second test

if (b!=0)

{

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

{

x[i] = (double)(rand()%1000 -500)/10;

y[i] = -(a\*x[i] + c)/b + (double)(rand()%1000 -500)/100;

x[i]+=(double)(rand()%1000 -500)/100;

fprintf(file2,"%lf %lf\n",x[i],y[i]);

}

aprox(n,x,y,a\_,b\_,c\_);

cout

<<"Straight line after the first test\n"

<<"("<<a\_<<")x + ("<<b\_<<")y + "<<c\_<<" = 0\n";

}

else if(a!=0)

{

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

{

y[i] = (double)(rand()%1000 -500)/10;

x[i] = -(b\*y[i] + c)/a + (double)(rand()%1000 -500)/100;

y[i] +=(double)(rand()%1000 -500)/100;

fprintf(file2,"%lf %lf\n",x[i],y[i]);

}

fclose(file1);

aprox(n,x,y,a\_,b\_,c\_);

cout

<<"Straight line after the first test\n"

<<"("<<a\_<<")x + ("<<b\_<<")y + "<<c\_<<" = 0\n";

}

double err2 = 0;

if (err2< fabs(a/(sqrt(a\*a+b\*b))-a\_)) err2=fabs(a/(sqrt(a\*a+b\*b))-a\_);

if (err2< fabs(b/(sqrt(a\*a+b\*b))-b\_)) err2=fabs(b/(sqrt(a\*a+b\*b))-b\_);

if (err2< fabs(c/(sqrt(a\*a+b\*b))-c\_)) err2=fabs(c/(sqrt(a\*a+b\*b))-c\_);

cout<<" max\_error "<<err2<<"\n";

fclose(file1);

fclose(file2);

delete [] x;

delete [] y;

return 0;

}

void aprox(int n,double \* x,double \* y,double& a,double& b,double& c)

{

double E,F,G,L1,L2;

double Ex2=0,Ex=0,Ey2=0,Ey=0,Exy=0;

double\*\* matr;

double\*\* m\_it;

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

{

Ex+=x[i];

Ey+=y[i];

Ex2+=x[i]\*x[i];

Ey2+=y[i]\*y[i];

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

}

E=Ex2-Ex\*Ex/n;

F=Exy-Ex\*Ey/n;

G=Ey2-Ey\*Ey/n;

matr = new double\*[2];

for(int i=0;i<2;i++)

matr[i]=new double[2];

m\_it = new double\*[2];

for(int i=0;i<2;i++)

m\_it[i]=new double[2];

double \*lam = new double[2];

matr[0][0]=E;

matr[0][1]=F;

matr[1][0]=F;

matr[1][1]=G;

jacobi(2,matr,lam,m\_it);

double C1=-(Ex\*m\_it[0][0]+Ey\*m\_it[1][0])/n;

double C2=-(Ex\*m\_it[0][1]+Ey\*m\_it[1][1])/n;

double s1=0;

double s2=0;

double sw;

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

{

sw=(x[i]\*m\_it[0][0]+y[i]\*m\_it[1][0]+C1);

s1+=sw\*sw;

}

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

{

sw=(x[i]\*m\_it[0][1]+y[i]\*m\_it[1][1]+C2);

s2+=sw\*sw;

}

if(s1<s2)

{

a=m\_it[0][0];

b=m\_it[1][0];

c=C1;

}

else

{

a=m\_it[0][1];

b=m\_it[1][1];

c=C2;

}

for(int i=0;i<2;i++)

delete []matr[i];

delete []matr;

for(int i=0;i<2;i++)

delete []m\_it[i];

delete []m\_it;

delete[] lam;

}

void jacobi ( int n, double \*\* a, double \* d, double \*\* v )

{

if ( n == 0 ) return;

double \* b = new double[n+n];

double \* z = b + n;

unsigned int i, j;

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

{

z[i] = 0.;

b[i] = d[i] = a[i][i];

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

{

v[i][j] = i == j ? 1. : 0.;

}

}

for ( i = 0; i < 50; ++i )

{

double sm = 0.;

unsigned int p, q;

for ( p = 0; p < n - 1; ++p )

{

for ( q = p + 1; q < n; ++q )

{

sm += fabs ( a[p][q] );

}

}

if ( fabs(sm)<1e-20 ) break;

const double tresh = i < 3 ? 0.2 \* sm / ( n\*n ) : 0.;

for ( p = 0; p < n - 1; ++p )

{

for ( q = p + 1; q < n; ++q )

{

const double g = 1e12 \* fabs ( a[p][q] );

if ( i >= 3 && fabs ( d[p] ) > g && fabs ( d[q] ) > g ) a[p][q] = 0.;

else

if ( fabs ( a[p][q] ) > tresh )

{

const double theta = 0.5 \* ( d[q] - d[p] ) / a[p][q];

double t = 1. / ( fabs(theta) + sqrt(1.+theta\*theta) );

if ( theta < 0 ) t = - t;

const double c = 1. / sqrt ( 1. + t\*t );

const double s = t \* c;

const double tau = s / ( 1. + c );

const double h = t \* a[p][q];

z[p] -= h;

z[q] += h;

d[p] -= h;

d[q] += h;

a[p][q] = 0.;

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

{

const double g = a[j][p];

const double h = a[j][q];

a[j][p] = g - s \* ( h + g \* tau );

a[j][q] = h + s \* ( g - h \* tau );

}

for ( j = p+1; j < q; ++j )

{

const double g = a[p][j];

const double h = a[j][q];

a[p][j] = g - s \* ( h + g \* tau );

a[j][q] = h + s \* ( g - h \* tau );

}

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

{

const double g = a[p][j];

const double h = a[q][j];

a[p][j] = g - s \* ( h + g \* tau );

a[q][j] = h + s \* ( g - h \* tau );

}

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

{

const double g = v[j][p];

const double h = v[j][q];

v[j][p] = g - s \* ( h + g \* tau );

v[j][q] = h + s \* ( g - h \* tau );

}

}

}

}

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

{

d[p] = ( b[p] += z[p] );

z[p] = 0.;

}

}

delete[] b;

}

Для прямой при количестве точек, равном 10, 50 и 100 результаты таковы:

  

Для прямой при количестве точек, равном 10, 50 и 100 результаты таковы:

  

Для прямой при количестве точек, равном 10, 50 и 100 результаты таковы:

  

Для прямой при количестве точек, равном 10, 50 и 100 результаты таковы:

  