//VC++的MFC实现的最小二乘法曲线拟合

BOOL YLF\_Curve\_CalPolyFactor(//计算多项式系数 calculate polynomial fact  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*输入\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 double \*x\_in,//拟合前数据点X轴坐标数组  
 double \*y\_in,//拟合前数据点Y轴坐标数组  
 unsigned long num\_out,//拟合后输出数据点组数  
 unsigned int nSimulateNum,//拟合方式(可选值1-4次曲线)  
 /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*输出\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  
 double \*factor//factor(长度固定为5)  
 )  
{  
 if (0 < num\_out)  
 {  
 double dbfactor[5];//系数数组  
 double \*pdbX,\*pdbY;//x,y数组零时变量  
  
 pdbX = new double[num\_out];  
 pdbY = new double[num\_out];  
 memcpy(pdbX,x\_in,sizeof(double)\*num\_out);  
 memcpy(pdbY,y\_in,sizeof(double)\*num\_out);  
  
 double \*b;  
 b = dbfactor;  
 int n = num\_out;  
 int m = nSimulateNum;  
 int i, j, k, l;  
 double B[6][7];  
 double temp, det;   
 temp = det = 0.0f;  
 BOOL BreakFlag;  
 double \*CurveX, \*CurveY,\*CurveS;  
 CurveX = new double[n];  
 CurveY = new double[n];  
 CurveS = new double[n];  
 ZeroMemory(B, 4 \* 6 \* 7);  
 ZeroMemory(CurveX, n);  
 ZeroMemory(CurveY, n);  
 ZeroMemory(CurveS, n);  
  
 for(i = 0; i < n; i++)  
 {  
 CurveX[i] = pdbX[i];  
 CurveY[i] = pdbY[i];  
 }  
 for(i = 0; i < 5; i++)  
 {  
 for(j = 0; j < 6; j++)  
 {  
 B[i][j] = 0;  
 }  
 }  
 B[0][0] = (double)n;  
 CurveS[0] = CurveX[0];  
 B[0][m + 1] = CurveY[0];  
 B[1][m + 1] = CurveY[0] \* CurveX[0];  
 B[1][0] = CurveX[0];  
  
 for(k = 1; k < n; k++)  
 {  
 B[0][m + 1] = B[0][m + 1] + CurveY[k];  
 B[1][m + 1] = B[1][m + 1] + CurveY[k] \* CurveX[k];  
 B[1][0] = B[1][0] + CurveX[k];  
 CurveS[k] = CurveX[k];  
 }  
  
 B[0][1] = B[1][0];  
 for(i = 2; i < m + 1; i++)  
 {  
 temp = 0;  
 B[i][m + 1] = 0;  
 for(k = 0; k < n; k++)  
 {  
 CurveS[k] = CurveS[k] \* CurveX[k];  
 B[i][m + 1] = B[i][m + 1] + CurveY[k] \* CurveS[k];  
 temp += CurveS[k];  
 }  
 for(j = 0; j < i + 1; j++)  
 {  
 l = i - j;  
 B[l][j] = temp;  
 }  
 }  
  
 for(j = 1; j < m + 1; j++)  
 {  
 temp = 0;  
 for(k = 0; k < n; k++)  
 {  
 CurveS[k] = CurveS[k] \* CurveX[k];  
 temp = temp + CurveS[k];  
 }  
 for(i = m; i >= j; i--)  
 {  
 l = m + j - i;  
 B[i][l] = temp;  
 }  
 }  
  
 BreakFlag = FALSE;  
 for(i = 0; i < m; i++)  
 {  
 det = B[i][i];  
 for(j = i; j < m + 2; j++)  
 {  
 if(fabs(det) > 0.00001)  
 B[i][j] = B[i][j] \* 1.0f / det;  
 else  
 {  
 BreakFlag = TRUE;  
 break;  
 }  
 }  
 if(BreakFlag == TRUE)  
 break;  
 for(j = i + 1; j < m + 1; j++)  
 {  
 det = B[j][i];  
 for(k = 0; k < m + 2; k++)  
 {  
 B[j][k] = B[j][k] - det \* B[i][k];  
 }  
 }  
 }  
 if(BreakFlag)  
 {  
 delete []CurveX;  
 delete []CurveY;  
 delete []CurveS;  
 delete []pdbX;  
 delete []pdbY;  
  
 ::MessageBox(NULL,"离散点数据数组无效!请检验有效性。","错误提示",MB\_OK|MB\_ICONWARNING);  
 return FALSE;//数据不合理  
 }  
  
 B[m][m + 1] = B[m][m + 1] \* 1.0f / B[m][m];  
 B[m][m] = 1;  
   
 for(i = m - 1; i >= 0; i--)  
 {  
 for(k = 1; k < m - i + 1; k++)  
 {  
 det = B[i][i + k];  
 for(j = 0; j < m + 2; j++)  
 {  
 B[i][j] = B[i][j] - B[i + k][j] \* det;  
 }  
 }  
 }  
 for(i = 0; i < m + 1; i++)  
 b[i] = B[i][m + 1];   
 delete []CurveX;  
 delete []CurveY;  
 delete []CurveS;  
 delete []pdbX;  
 delete []pdbY;  
  
 //输出系数数据数组  
 memcpy(factor,dbfactor,5\*sizeof(double));  
  
 return TRUE;  
 }   
 else return FALSE;  
}