[梯度下降与最小二乘法](http://www.cnblogs.com/pixels/archive/2011/04/19/2021442.html)

#include "stdafx.h"  
#include "cv.h"  
#include "cxcore.h"  
#include "highgui.h"  
#include <time.h>  
#include <stdlib.h>  
#include <iostream>  
#include <vector>  
  
#pragma comment(lib, "cv.lib")  
#pragma comment(lib, "cxcore.lib")  
#pragma comment(lib, "highgui.lib")  
using namespace std;  
  
double ComputeGradient(vector<CvPoint2D32f> pts, double theta0, double theta1, int index)  
{  
    double retVal = 0;  
    if(index == 0)  
    {  
        for(int i = 0; i < pts.size(); ++i)  
        {  
            retVal += ((theta0 + theta1\*pts[i].x)-pts[i].y)\*1;  
        }  
    }  
    else  
    {  
        for(int i = 0; i < pts.size(); ++i)  
        {  
            retVal += ((theta0 + theta1\*pts[i].x)-pts[i].y)\*pts[i].x;  
        }  
    }  
  
    return retVal;  
}  
  
int \_tmain(int argc, \_TCHAR\* argv[])  
{  
    IplImage \*img = cvCreateImage(cvSize(800, 600), 8, 3);  
    cvZero(img);  
  
    // create data  
    vector<CvPoint2D32f> pts;  
    srand(time(NULL));  
    for(int x = 0; x < img->width/4; x += 3)  
    {  
        double y = 2\*x + rand()%50;  
        CvPoint2D32f pt = cvPoint2D32f(x, y);  
        pts.push\_back(pt);  
    }  
  
    // gradient descent method  
    double theta0 = 0.0, theta1 = 0.0;  
    double alpha = 0.000001;  
    cvNamedWindow("disp", 1);  
    for(int l = 0; l < 100; ++l)  
    {  
        cout << "(theta0, theta1): " << "(" << theta0 << ", " << theta1 << ")" << endl;  
        double tempTheta0 = theta0 - alpha\*ComputeGradient(pts, theta0, theta1, 0);  
        double tempTheta1 = theta1 - alpha\*ComputeGradient(pts, theta0, theta1, 1);  
          
        double eps = sqrt( (tempTheta0-theta0)\*(tempTheta0-theta0) + (tempTheta1-theta1)\*(tempTheta1-theta1));  
        if(eps < 0.001)  
            break;  
        else  
        {  
            theta0 = tempTheta0;  
            theta1 = tempTheta1;  
        }  
  
        // update result      
        cvZero(img);  
        for(int i = 0; i < pts.size(); ++i)  
        {  
            cvDrawCircle(img, cvPoint(pts[i].x, pts[i].y), 2, CV\_RGB(250, 0, 0),  
                2, CV\_AA);  
        }  
        cvDrawLine(img, cvPoint(0, theta0), cvPoint(200, theta0+200\*theta1), CV\_RGB(0, 250, 0),  
            2, CV\_AA);  
  
        cvShowImage("disp", img);  
        cvWaitKey(1000);  
    }  
      
    // least square method  
    CvMat \*XT = cvCreateMat(2, pts.size(), CV\_32FC1);  
    CvMat \*X = cvCreateMat(pts.size(), 2, CV\_32FC1);  
    CvMat \*Y = cvCreateMat(pts.size(), 1, CV\_32FC1);  
    CvMat \*XTX = cvCreateMat(2,2,CV\_32FC1);  
    CvMat \*THETA = cvCreateMat(2, 1, CV\_32FC1);  
    for(int i = 0; i < pts.size(); ++i)  
    {  
        cvSet2D(XT, 0, i, cvScalarAll(1) );  
        cvSet2D(XT, 1, i, cvScalarAll(pts[i].x));  
        cvSet2D(Y, i, 0, cvScalarAll(pts[i].y));  
    }  
    // core of method THETA = invert(Xt\*X)\*Xt\*Y;  
    cvTranspose(XT, X);  
    cvMatMul(XT, X, XTX);  
    cvInvert(XTX, XTX);  
    cvMatMul(XTX, XT, XT);  
    cvMatMul(XT, Y, THETA);  
    theta0 = cvGet2D(THETA, 0, 0).val[0];  
    theta1 = cvGet2D(THETA, 1, 0).val[0];  
    cvDrawLine(img, cvPoint(0, theta0), cvPoint(200, theta0+200\*theta1), CV\_RGB(250, 250, 0),2, CV\_AA);  
    cvShowImage("disp", img);  
    cout << "THETA: " << "(" << cvGet2D(THETA, 0, 0).val[0] << ", " << cvGet2D(THETA, 1, 0).val[0] << ")" << endl;  
    cvWaitKey(0);  
    return 0;

}