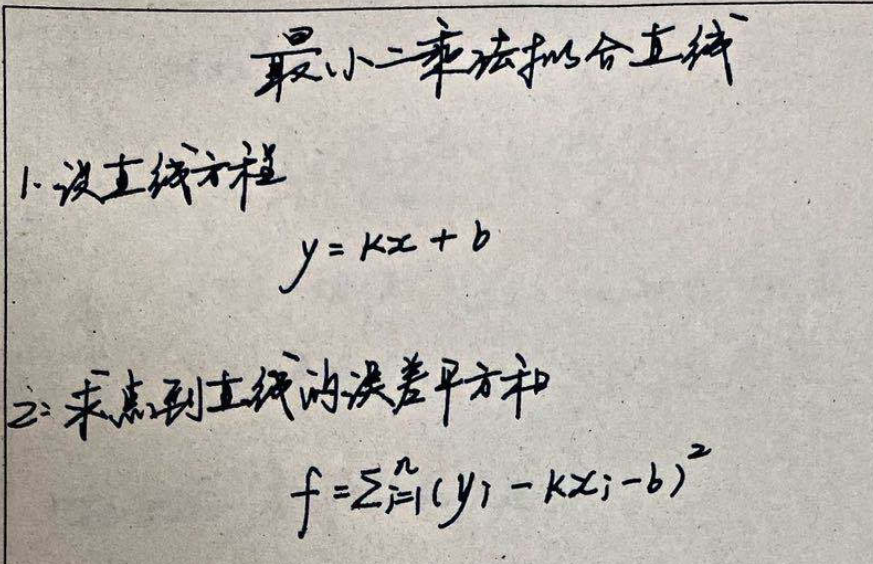
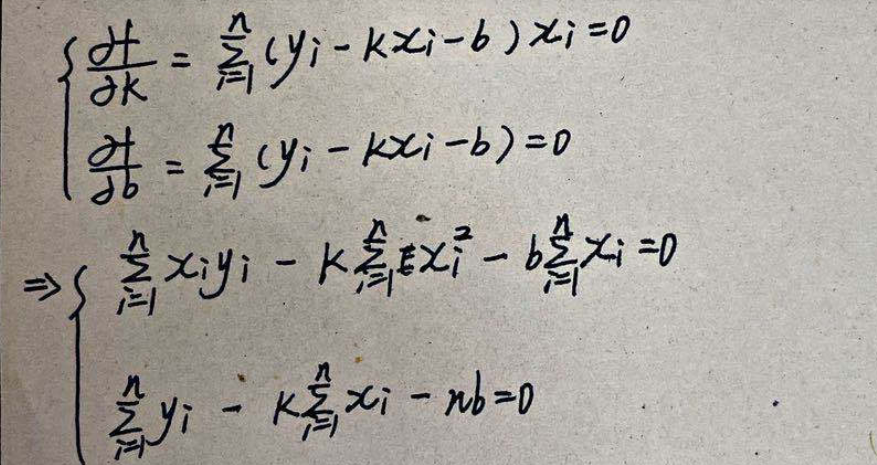
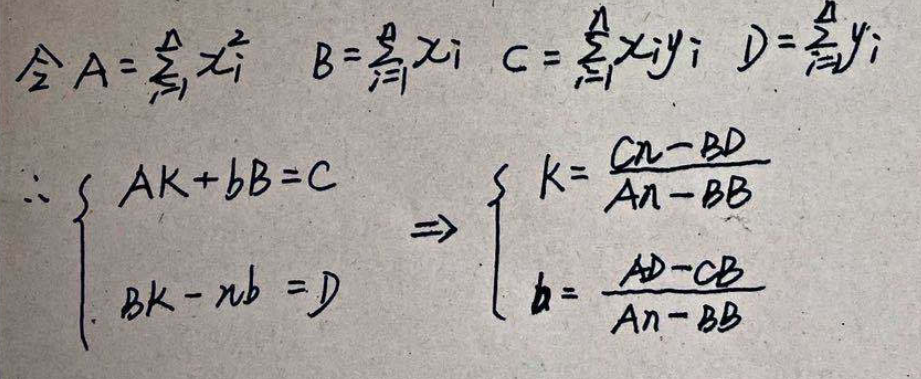
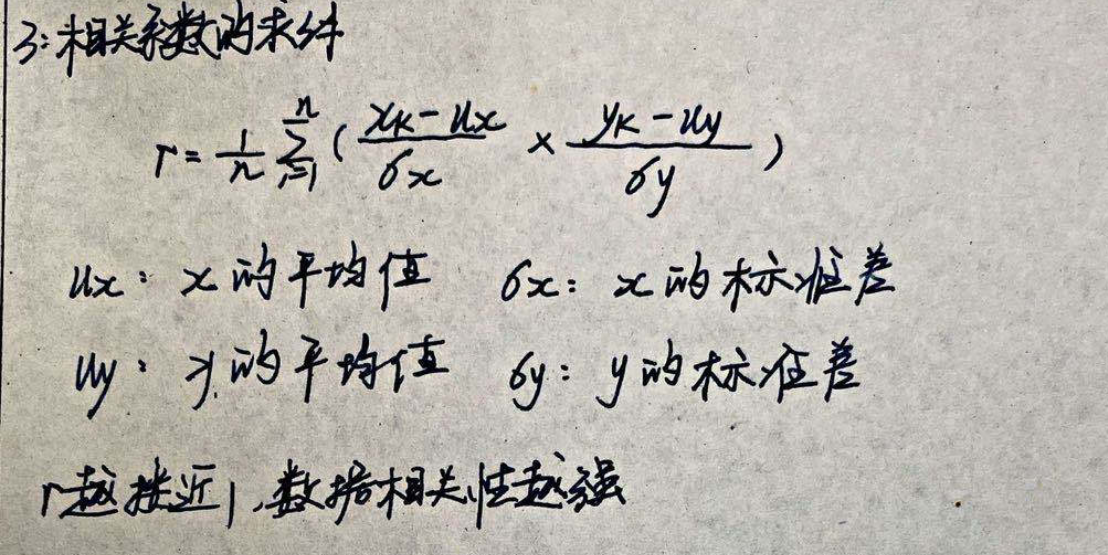
最小二乘法拟合直线

1. 基本原理









1. C++及opencv验证

void LineFitLeastFit(const std::vector<cv::Point2f> &\_points,float & \_k,float & \_b,float & \_r) {

float B = 0.0f;

float A = 0.0f;

float D = 0.0f;

float C = 0.0f;

int N = \_points.size();

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

{

B += \_points[i].x;

A += \_points[i].x \* \_points[i].x;

D += \_points[i].y;

C += \_points[i].x \* \_points[i].y;

}

if ((N \* A - B \* B) == 0)

return;

\_k = (N \* C - B \* D) / (N \* A - B \* B);

\_b = (A \* D - C \* B) / (N \* A - B \* B);

//计算相关系数

float Xmean = B / N;

float Ymean = D / N;

float tempX = 0.0f;

float tempY = 0.0f;

float rDenominator = 0.0;

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

{

tempX += (\_points[i].x - Xmean) \* (\_points[i].x - Xmean);

tempY += (\_points[i].y - Ymean) \* (\_points[i].y - Ymean);

rDenominator += (\_points[i].x - Xmean) \* (\_points[i].y - Ymean);

}

float SigmaXY = sqrt(tempX) \* sqrt(tempY);

if (SigmaXY == 0)

return;

\_r = rDenominator / SigmaXY;

}

————————————————

：https://blog.csdn.net/jjjstephen/article/details/108053148

1. Matlab实现

3.1 直接计算

clc

clear all

points\_x = [1 2 3 4 5 6 7 8 9];

points\_y = [110 77 80 64 57 41 34 22 15];

plot(points\_x , points\_y,'r\*');

title('\bf 基于离散坐标的最小二乘法拟合直线，拟合方程y = kx + b');

xlabel('points\_x');

ylabel('points\_y');

text(9,40, 'y = kx + b');

[line\_fit\_k, line\_fit\_b] = StraightLineFitbyLeastSquare(points\_x, points\_y);

size\_points\_x = size(points\_x);

size\_points\_y = size(points\_y);

if(size\_points\_x == size\_points\_y)

fit\_start\_x = points\_x(1,1) - 1;

fit\_end\_x = points\_x(1,size\_points\_x(1,2)) + 1;

fit\_x = linspace(fit\_start\_x, fit\_end\_x,10000);

fit\_y = line\_fit\_k \* fit\_x + line\_fit\_b;

hold on

plot(fit\_x,fit\_y,'b.');

hold off

end

function [line\_fit\_k, line\_fit\_b] = StraightLineFitbyLeastSquare(points\_vector\_x, points\_vector\_y)

[points\_num\_rawx,points\_num\_colx] = size(points\_vector\_x);

[points\_num\_rawy,points\_num\_coly] = size(points\_vector\_y);

if ((points\_num\_rawx == 1) &&(points\_num\_rawy == 1)&& (points\_num\_colx == points\_num\_coly))

for i = 1:1:points\_num\_colx

calc\_coef\_A = points\_vector\_x \* points\_vector\_x' ;

calc\_coef\_B = sum(points\_vector\_x);

calc\_coef\_C = sum(points\_vector\_x .\* points\_vector\_y);

calc\_coef\_D = sum(points\_vector\_y);

end

calc\_coef\_base = points\_num\_colx \* calc\_coef\_A - calc\_coef\_B \* calc\_coef\_B;

if(calc\_coef\_base == 0)

disp('Line fit base error!');

end

line\_fit\_k = (points\_num\_colx \* calc\_coef\_C - calc\_coef\_B \* calc\_coef\_D ) / calc\_coef\_base;

line\_fit\_b = (calc\_coef\_A \* calc\_coef\_D - calc\_coef\_B \* calc\_coef\_C) / calc\_coef\_base;

else

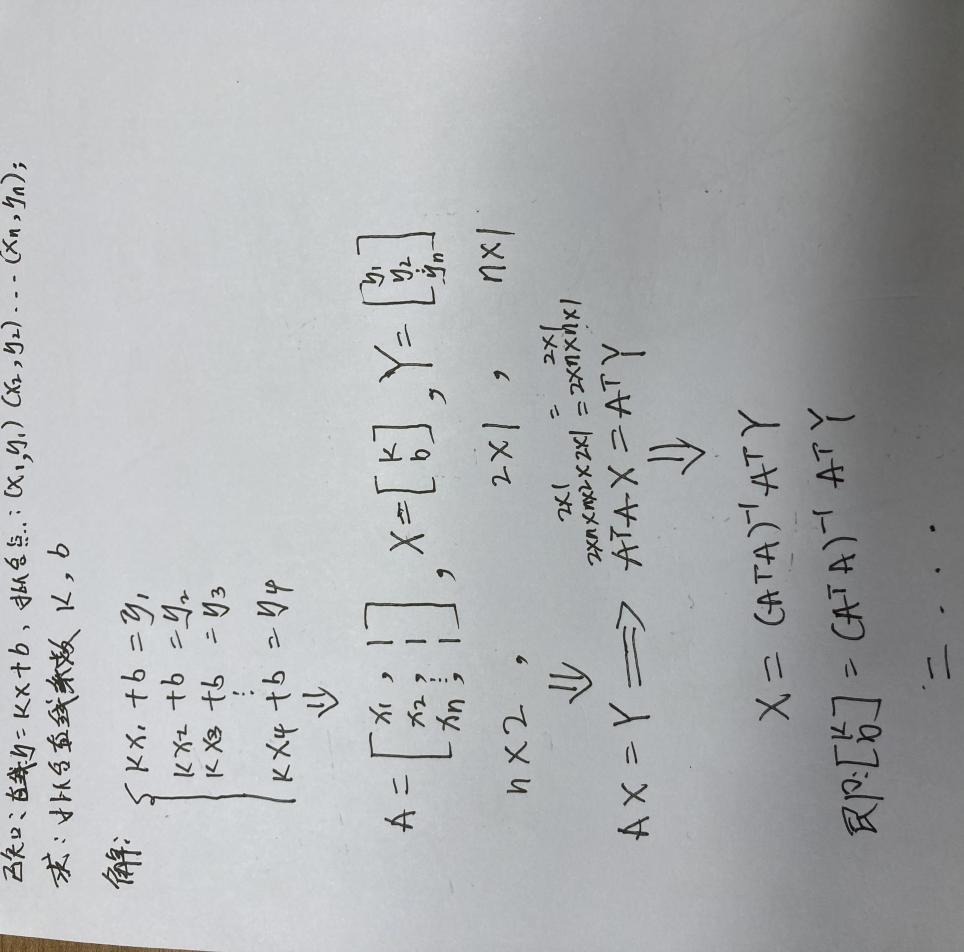
disp('please check put\_in parameter which is a vector!');

end

end

3.2 矩阵计算（原理+Matlab实现）

3.2.1 计算原理



3.2.2 matlab代码实现

clc

clear all

matrix\_A = [1 1; 2 1; 3 1; 4 1; 5 1; 6 1; 7 1; 8 1; 9 1];

matrix\_Y = [110; 77; 80; 64; 57; 41; 34; 22; 15];

matrix\_X = zeros(2,1);

plot(matrix\_A(:,1) , matrix\_Y,'r\*');

title('\bf 基于离散坐标的最小二乘法拟合直线，拟合方程y = kx + b');

xlabel('points\_x');

ylabel('points\_y');

text(9,40, 'y = kx + b');

matrix\_X = ((matrix\_A'\*matrix\_A)^-1)\*matrix\_A'\*matrix\_Y;

size\_matrix\_A = size(matrix\_A);

size\_matrix\_Y = size(matrix\_Y);

if(size\_matrix\_A(1,1) == size\_matrix\_Y(1,1))

fit\_start\_x = matrix\_A(1,1) - 1;

fit\_end\_x = matrix\_A(size\_matrix\_A(1,1),1) + 1;

fit\_x = linspace(fit\_start\_x, fit\_end\_x,10000);

fit\_y = matrix\_X(1,1) \* fit\_x + matrix\_X(2,1);

hold on

plot(fit\_x,fit\_y,'b.');

hold off

end