

Task 3 选做题：卡尔曼滤波

```
#include <iostream>
```

```
#include <vector>
```

```
#include <random>
```

```
#include <Eigen/Dense>
```

```
#include <opencv2/opencv.hpp>
```

```
using namespace Eigen;
```

```
using namespace cv;
```

```
std::random_device rd;
```

```
std::mt19937 gen(rd());
```

```
std::normal_distribution<> noise(0, 1);
```

```
MatrixXd kf(const std::vector<double>&
```

```
m) {
```

```
    double dt = 1;
```

```
    double x = 0;
```

```
    double v = 2;
```

```
Matrix<double, 2, 2> F;
```

```
F << 1, dt,
```

```
    0, 1;
```

```
Matrix<double, 2, 2> P;
```

```
P << 1, 0,
```

```
    0, 1;
```

```
Matrix<double, 1, 2> H;
```

```
H << 1, 0;
```

```
Matrix<double, 1, 1> R;
```

```
R << 1;
```

```
Matrix<double, 2, 2> Q;
```

```
Q << 0, 0,
```

```
    0, 0;
```

```
Matrix<double, 2, 1> s;
```

```
s << x, v;
```

```
MatrixXd f(m.size(), 2); }
```

```
for (std::size_t i = 0; i < m.size(); ++i) {
```

```
    s = F * s;
```

```
    P = F * P * F.transpose() + Q;
```

```
    double mea = m[i] + noise(gen);
```

```
    double y = mea - H * s;
```

```
    Matrix<double, 1, 1> S = H * P *
```

```
    H.transpose() + R;
```

```
    Matrix<double, 2, 1> K = P *
```

```
    H.transpose() * S.inverse();
```

```
    s = s + K * y;
```

```
    P = (Matrix<double, 2,
```

```
    2>::Identity() - K * H) * P;
```

```
    f.row(i) = s.transpose();
```

```
}
```

```
return f;
```

```
int main() {
```

```
    std::vector<double> p;
```

```
    std::vector<double> m;
```

```
    double t = 10.0;
```

```
    double d = 0.1;
```

```
    double v = 2.0;
```

```
    for (double i = 0.0; i <= t; i += d)
```

```
    {
```

```
        double tp = v * i;
```

```
        p.push_back(tp);
```

```
        m.push_back(tp + noise(gen));
```

```
    }
```

```
    MatrixXd f = kf(m);
```

```
    int w = 800;
```

```
    int h = 600;
```

```

cv::namedWindow("KF",
cv::WINDOW_NORMAL);

cv::resizeWindow("KF", w, h);

double mp =
*std::max_element(p.begin(), p.end());

double mm =
*std::max_element(m.begin(), m.end());

double mf = f.maxCoeff();

cv::Mat c(h, w, CV_8UC3, cv::Scalar(255,
255, 255));

cv::line(c, cv::Point(0, h), cv::Point(w, h),
cv::Scalar(0, 0, 0));

cv::line(c, cv::Point(0, h), cv::Point(0, 0),
cv::Scalar(0, 0, 0));

int n = 5;

double ts = t / n;

```

```

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

double tl = i * ts;

int xl = w * (tl / t);

std::stringstream ss;

ss << std::fixed <<

std::setprecision(1) << tl;

cv::putText(c, ss.str(), cv::Point(xl,
h - 10), cv::FONT_HERSHEY_SIMPLEX, 0.5,
cv::Scalar(0, 0, 0));

}

int ny = 5;

double mv = std::max({ mp, mm, mf });

double ys = mv / ny;

for (int i = 0; i <= ny; ++i) {

double yl = i * ys;

int yc = h - h * (yl / mv);

std::stringstream ss;

ss << std::fixed <<

std::setprecision(1) << yl;

```

```

        cv::putText(c, ss.str(), cv::Point(10,
yc), cv::FONT_HERSHEY_SIMPLEX, 0.5,
cv::Scalar(0, 0, 0));

    }

    cv::Scalar pc(255, 0, 0);

    cv::Scalar mc(0, 0, 255);

    cv::Scalar fc(0, 255, 0);

    int lw = 1;

    for (int i = 1; i < p.size(); ++i) {

        int x1 = w * (p[i - 1] / mp);

        int x2 = w * (p[i] / mp);

        int y1p = h - h * (p[i - 1] / mv);

        int y2p = h - h * (p[i] / mv);

        int y1m = h - h * (m[i - 1] / mv);

        int y2m = h - h * (m[i] / mv);

        int y1f = h - h * (f[i - 1, 0] / mv);

        int y2f = h - h * (f[i, 0] / mv);

        cv::line(c, cv::Point(x1, y1p),
cv::Point(x2, y2p), pc, lw);

        cv::line(c, cv::Point(x1, y1m),
cv::Point(x2, y2m), mc, lw);

        cv::line(c, cv::Point(x1, y1f),
cv::Point(x2, y2f), fc, lw);

    }

    cv::Scalar nc(0, 0, 0);

    int r = 2;

    for (int i = 0; i < m.size(); ++i) {

        int x = w * (p[i] / mp);

        int y = h - h * (m[i] / mv);

        cv::circle(c, cv::Point(x, y), r, nc, -1);

    }

    cv::imshow("KF", c);

    cv::waitKey(0);

    return 0;

```

```

1 #include <iostream>
2 #include <vector>
3 #include <random>
4 #include <Eigen/Dense>
5 #include <opencv2/opencv.hpp>
6
7 using namespace Eigen;
8 using namespace cv;
9
10 std::random_device rd;
11 std::mt19937 gen(rd());
12 std::normal_distribution<> noise(0, 1);
13
14 MatrixXd kf(const std::vector<double>& m) {
15     double dt = 1;
16     double x = 0;
17     double y = 2;
18
19     Matrix<double, 2, 2> F;
20     F << 1, 0;
21         0, 1;
22
23     Matrix<double, 2, 2> P;
24     P << 1, 0;
25         0, 1;

```

```

121 int y1p = h - h * (p[i] - 1) / mw;
122 int y2p = h - h * (p[i] / mw);
123 int y1m = h - h * (m[i] - 1) / mw;
124 int y2m = h - h * (m[i] / mw);
125 int y1f = h - h * (f[i] - 1, 0) / mw;
126 int y2f = h - h * (f[i], 0) / mw);
127
128 cv::line(c, cv::Point(x1, y1p), cv::Point(x2, y2p), mc, lw);
129 cv::line(c, cv::Point(x1, y1m), cv::Point(x2, y2m), mc, lw);
130 cv::line(c, cv::Point(x1, y1f), cv::Point(x2, y2f), fc, lw);
131
132
133 cv::Scalar mc(0, 0, 0);
134 int r = 2;
135
136 for (int i = 0; i < m.size(); ++i) {
137     int x = w * (p[i] / mw);
138     int y = h - h * (m[i] / mw);
139     cv::circle(c, cv::Point(x, y), r, mc, -1);
140
141     cv::imshow("kf", c);
142     cv::waitKey(0);
143
144     return 0;
145
146

```

```

26 // ..
27 Matrix<double, 1, 2> H;
28 H << 1, 0;
29
30 Matrix<double, 1, 1> R;
31 R << 1;
32
33 Matrix<double, 2, 2> Q;
34 Q << 0, 0;
35     0, 0;
36
37 Matrix<double, 2, 1> s;
38 s << x, y;
39
40 MatrixXd f(m.size(), 2);
41
42 for (std::size_t i = 0; i < m.size(); ++i) {
43     s = F * s;
44     P = P + P * F.transpose() + Q;
45
46     double mea = m[i] + noise(gen);
47     double y = mea - H * s;
48     Matrix<double, 1, 1> S = H * P * H.transpose() + R;
49     Matrix<double, 2, 1> K = P * H.transpose() * S.inverse();

```

```

121 int y1p = h - h * (p[i] - 1) / mw;
122 int y2p = h - h * (p[i] / mw);
123 int y1m = h - h * (m[i] - 1) / mw;
124 int y2m = h - h * (m[i] / mw);
125 int y1f = h - h * (f[i] - 1, 0) / mw;
126 int y2f = h - h * (f[i], 0) / mw);
127
128 cv::line(c, cv::Point(x1, y1p), cv::Point(x2, y2p), mc, lw);
129 cv::line(c, cv::Point(x1, y1m), cv::Point(x2, y2m), mc, lw);
130 cv::line(c, cv::Point(x1, y1f), cv::Point(x2, y2f), fc, lw);
131
132
133 cv::Scalar mc(0, 0, 0);
134 int r = 2;
135
136 for (int i = 0; i < m.size(); ++i) {
137     int x = w * (p[i] / mw);
138     int y = h - h * (m[i] / mw);
139     cv::circle(c, cv::Point(x, y), r, mc, -1);
140
141     cv::imshow("kf", c);
142     cv::waitKey(0);
143
144     return 0;
145
146

```

```

49 // ..
50 Matrix<double, 2, 1> K = P * H.transpose() * S.inverse();
51
52 s = s + K * y;
53 P = (Matrix<double, 2, 2>::Identity() - K * H) * P;
54
55 f.row(i) = s.transpose();
56
57 return f;
58
59
60 int main() {
61     std::vector<double> p;
62     std::vector<double> m;
63
64     double t = 10.0;
65     double d = 0.1;
66     double v = 2.0;
67
68     for (double i = 0.0; i <= t; i += d) {
69         double tp = v * i;
70         p.push_back(tp);
71         m.push_back(tp + noise(gen));
72
73     }
74
75     // ..

```

先看学习路径中的视频搞懂卡尔曼滤波器的

原理,再建一个卡尔曼滤波器类把计算过程用

到的矩阵放在里面,再写主函数。

我的模型是一个匀速运动模型,起始位置为 0,

速度为 2m/s,噪声服从分别为 (0,1) 和

(0,10)。然后再以图像模式呈现出来。

下面分别是噪声服从 (0,1) 分布和 (0,10)

分布的卡尔曼滤波图像

```

74 MatrixXd f = kf(m);
75
76 int w = 800;
77 int h = 600;
78 cv::namedWindow("kf", cv::WINDOW_NORMAL);
79 cv::resizeWindow("kf", w, h);
80
81 double mp = *std::max_element(p.begin(), p.end());
82 double mm = *std::max_element(m.begin(), m.end());
83 double mf = f.maxCoeff();
84
85 cv::Mat c(h, w, CV_8UC3, cv::Scalar(255, 255, 255));
86
87 cv::line(c, cv::Point(0, h), cv::Point(w, h), cv::Scalar(0, 0, 0));
88 cv::line(c, cv::Point(0, h), cv::Point(0, 0), cv::Scalar(0, 0, 0));
89
90 int n = 5;
91 double ts = t / n;
92
93 for (int i = 0; i <= n; ++i) {
94     double tl = i * ts;
95     int xl = w * (tl / t);
96     std::stringstream ss;
97     ss << std::fixed << std::setprecision(1) << tl;

```

```

98     ss << std::fixed << std::setprecision(1) << tl;
99     cv::putText(c, ss.str(), cv::Point(xl, h - 10), cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(0, 0, 0));
100
101     int ny = 5;
102     double mv = std::max({mp, mm, mf});
103     double ys = mv / ny;
104
105     for (int i = 0; i <= ny; ++i) {
106         double yl = i * ys;
107         int ye = h - h * (yl / mv);
108         std::stringstream ss;
109         ss << std::fixed << std::setprecision(1) << yl;
110         cv::putText(c, ss.str(), cv::Point(10, ye), cv::FONT_HERSHEY_SIMPLEX, 0.5, cv::Scalar(0, 0, 0));
111     }
112
113     cv::Scalar pc(255, 0, 0);
114     cv::Scalar mc(0, 0, 255);
115     cv::Scalar fc(0, 255, 0);
116     int lw = 1;
117
118     for (int i = 1; i < p.size(); ++i) {
119         int xl = w * (p[i] - 1) / mp;
120         int xm = w * (p[i] / mp);
121         int ylp = h - h * (p[i] - 1) / mw;

```

