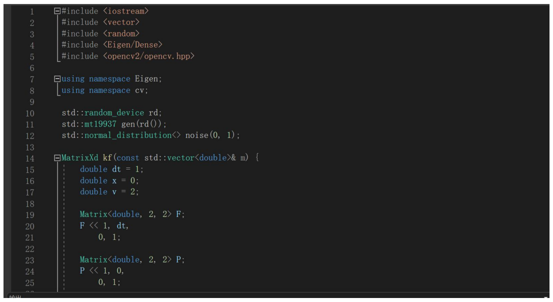
Task 3 选做题：卡尔曼滤波

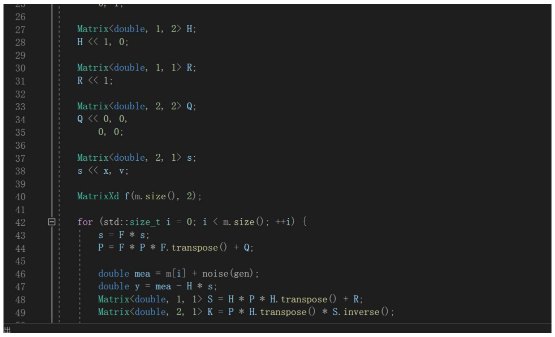
```
#include <iostream>
#include <vector>
#include <random>
#include <Eigen/Dense>
#include <opencv2/opencv.hpp>

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;
```

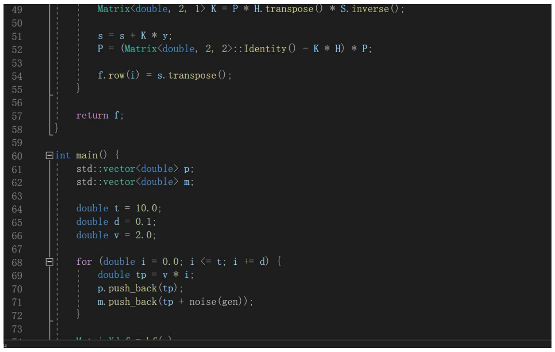
```
}

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;
```



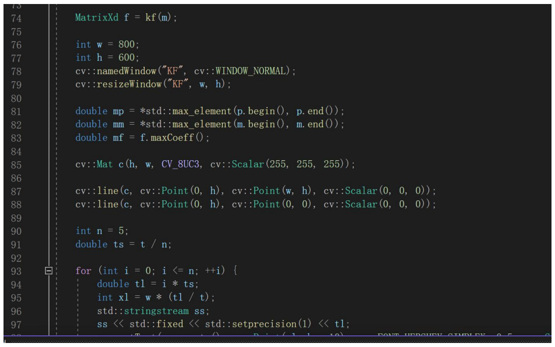
```
#include <iostream>
#include <vector>
#include <random>
#include <Eigen/Dense>
#include <opencv2/opencv.hpp>

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;
```



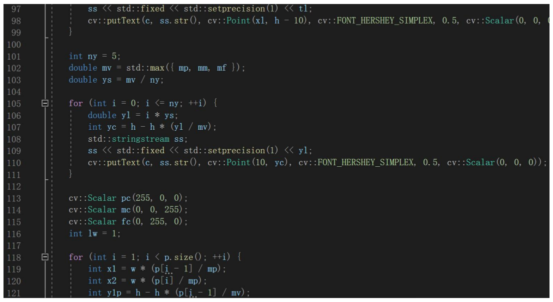
```

    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();
```



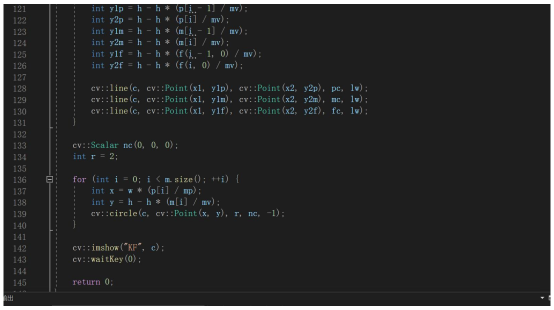
```
        Matrix<double, 2, 1> K = P * H.transpose() * S.inverse();

        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));
    }

```



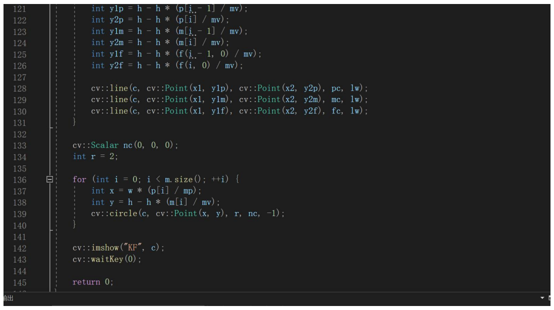
```
    MatrixXd f = kf(m);

    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;
```



```
        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, (
    }

    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 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;
```

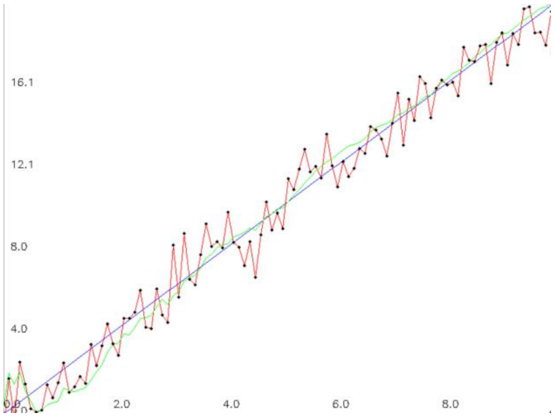


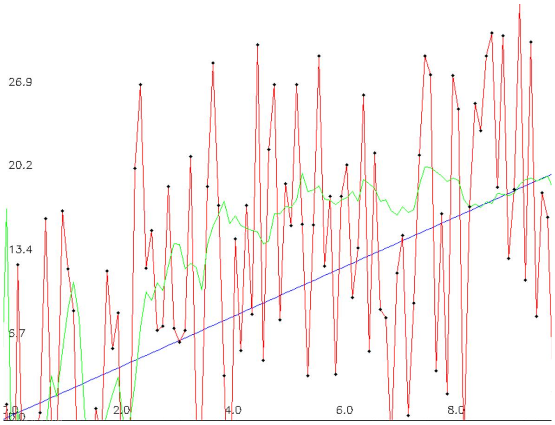
```
        int y1p = h - h * (p[i - 1] / mv);
        int y2p = h - h * (p[i] / mv);
        int y1m = h - h * (m[i - 1] / mv);
        int y2m = h - h * (m[i] / mv);
        int y1f = h - h * (f(i - 1, 0) / mv);
        int y2f = h - h * (f(i, 0) / mv);

        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;
```

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


16.1
12.1
8.0
4.0
0.0
2.0
4.0
6.0
8.0


26.9
20.2
13.4
6.7
0.0
2.0
4.0
6.0
8.0