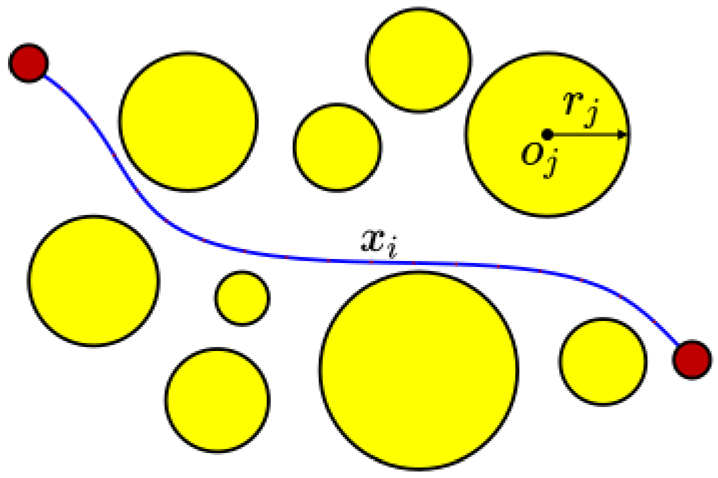
hw-l2-numerical optiamization of robot

PROBLEM



$$\mathrm{Potential}(x_1, x_2 \ldots, x_{N-1}) = 1000 \sum_{i=1}^{N-1} \sum_{j=1}^{M} \max(r_j - \|x_i - o_j\|,\ 0)$$

Lbfgs

code analysis

MinimizationExample 类中 run 方法

传入 N 个节点，初始值给定

```C++
for (int i = 0; i < N; i += 2)
        {
            x(i) = -1.2;
            x(i + 1) = 1.0;
        }
```

设定参数结构体

```C++
int mem_size = 8;近似逆 hessian 矩阵 B_的校正次数。

double g_epsilon = 0.0;收敛限度

int past = 3;基于迭代需要计算的距离。

double delta = 1.0e-6;//代价函数差值限度

int max_iterations = 0; 最大迭代次数

int max_linesearch = 64; 最大线搜索次数

 double min_step = 1.0e-20;线搜索最小步

double max_step = 1.0e+20;线搜索最大步

double f_dec_coeff = 1.0e-4;线搜索精度

double s_curv_coeff = 0.9; 线搜索精度

double cautious_factor = 1.0e-6;确保非凸情况的全局收敛
```

```
double machine_prec = 1.0e-16;机器浮点数精度
```

例子的设置

```C++
params.g_epsilon = 1.0e-8;
        params.past = 3;
        params.delta = 1.0e-8;
```

开始最小化

```C++
int ret = lbfgs::lbfgs_optimize(x,
                                  finalCost,
                                  costFunction,
                                  monitorProgress,
                                  this,
                                  params);
```

代价回调函数

```C++
static double costFunction(void *instance,
                           const Eigen::VectorXd &x,
                           Eigen::VectorXd &g)//注意这里非const
{
    const int n = x.size();
    double fx = 0.0;
    for (int i = 0; i < n; i += 2)
    {
        const double t1 = 1.0 - x(i);
        const double t2 = 10.0 * (x(i + 1) - x(i) * x(i));
        g(i + 1) = 20.0 * t2;
        g(i) = -2.0 * (x(i) * g(i + 1) + t1);
        fx += t1 * t1 + t2 * t2;
    }
    return fx;
}
```

打印输出

```C++
static int monitorProgress(void *instance,
                           const Eigen::VectorXd &x,
                           const Eigen::VectorXd &g,
                           const double fx,
                           const double step,
                           const int k,
                           const int ls)
{
    std::cout << std::setprecision(4)
              << "================================" << std::endl
              << "Iteration: " << k << std::endl
              << "Function Value: " << fx << std::endl
              << "Gradient Inf Norm: " << g.cwiseAbs().maxCoeff() << std::endl
              << "Variables: " << std::endl
              << x.transpose() << std::endl;
    return 0;
}
```

客户端程序的用户数据指针 this

参数-常量

- 返回值

状态代码。如果最小化过程终止时没有一个错误,此函数返回非负。否则返回负整数表示错误。

打印状态和最终代价

```C++
/* Report the result. */
        std::cout << std::setprecision(4)
              << "================================" << std::endl
              << "L-BFGS Optimization Returned: " << ret << std::endl
              << "Minimized Cost: " << finalCost << std::endl
              << "Optimal Variables: " << std::endl
              << x.transpose() << std::endl;
```

最后返回状态 结束主程序

lbfgs_optimize

判断输入参数是否不符合规定

定义中间变量初始化内存限制

```C++
/* Prepare intermediate variables. */
        Eigen::VectorXd xp(n);
        Eigen::VectorXd g(n);
        Eigen::VectorXd gp(n);
        Eigen::VectorXd d(n);
        Eigen::VectorXd pf(std::max(1, param.past));

        /* Initialize the limited memory. */
        Eigen::VectorXd lm_alpha = Eigen::VectorXd::Zero(m);
        Eigen::MatrixXd lm_s = Eigen::MatrixXd::Zero(n, m);
        Eigen::MatrixXd lm_y = Eigen::MatrixXd::Zero(n, m);
        Eigen::VectorXd lm_ys = Eigen::VectorXd::Zero(m);
```

回调数据结构体构造，给定回调函数的初始值

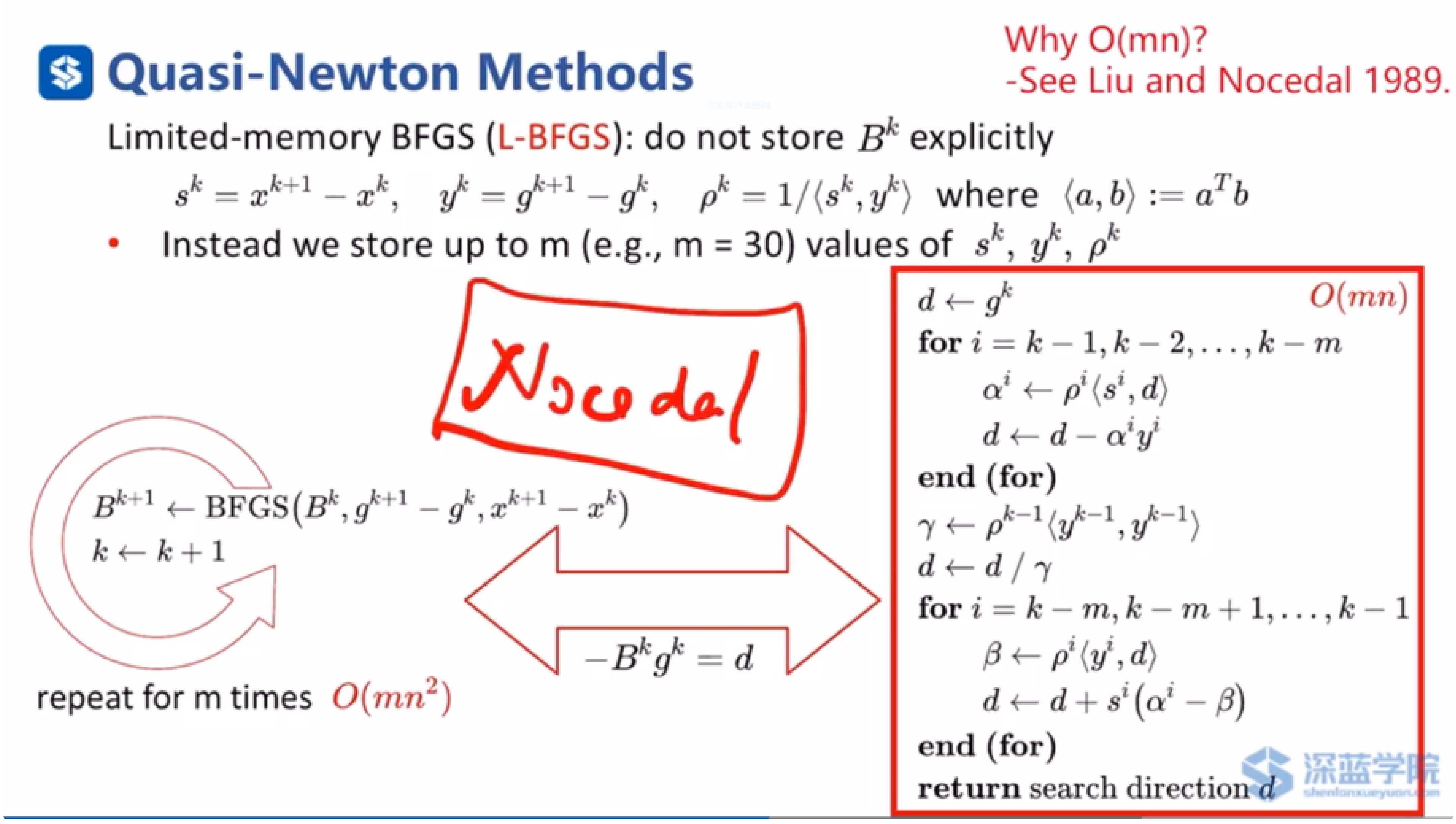
获取方向，假设初始海森矩阵 H_0 为单位矩阵

确保初始变量不是平稳点。

```C++
/* Construct a callback data. */
callback_data_t cd;
cd.instance = instance;
cd.proc_evaluate = proc_evaluate;//代价函数
cd.proc_progress = proc_progress;

/* Evaluate the function value and its gradient. */
fx = cd.proc_evaluate(cd.instance, x, g);

/* Store the initial value of the cost function. */
pf(0) = fx;

/*
Compute the direction;
we assume the initial hessian matrix H_0 as the identity matrix.
*/
d = -g;

/*
Make sure that the initial variables are not a stationary point.
*/
gnorm_inf = g.cwiseAbs().maxCoeff();
xnorm_inf = x.cwiseAbs().maxCoeff();

if (gnorm_inf / std::max(1.0, xnorm_inf) < param.g_epsilon)
{
    /* The initial guess is already a stationary point. */
    ret = LBFGS_CONVERGENCE;
}
```

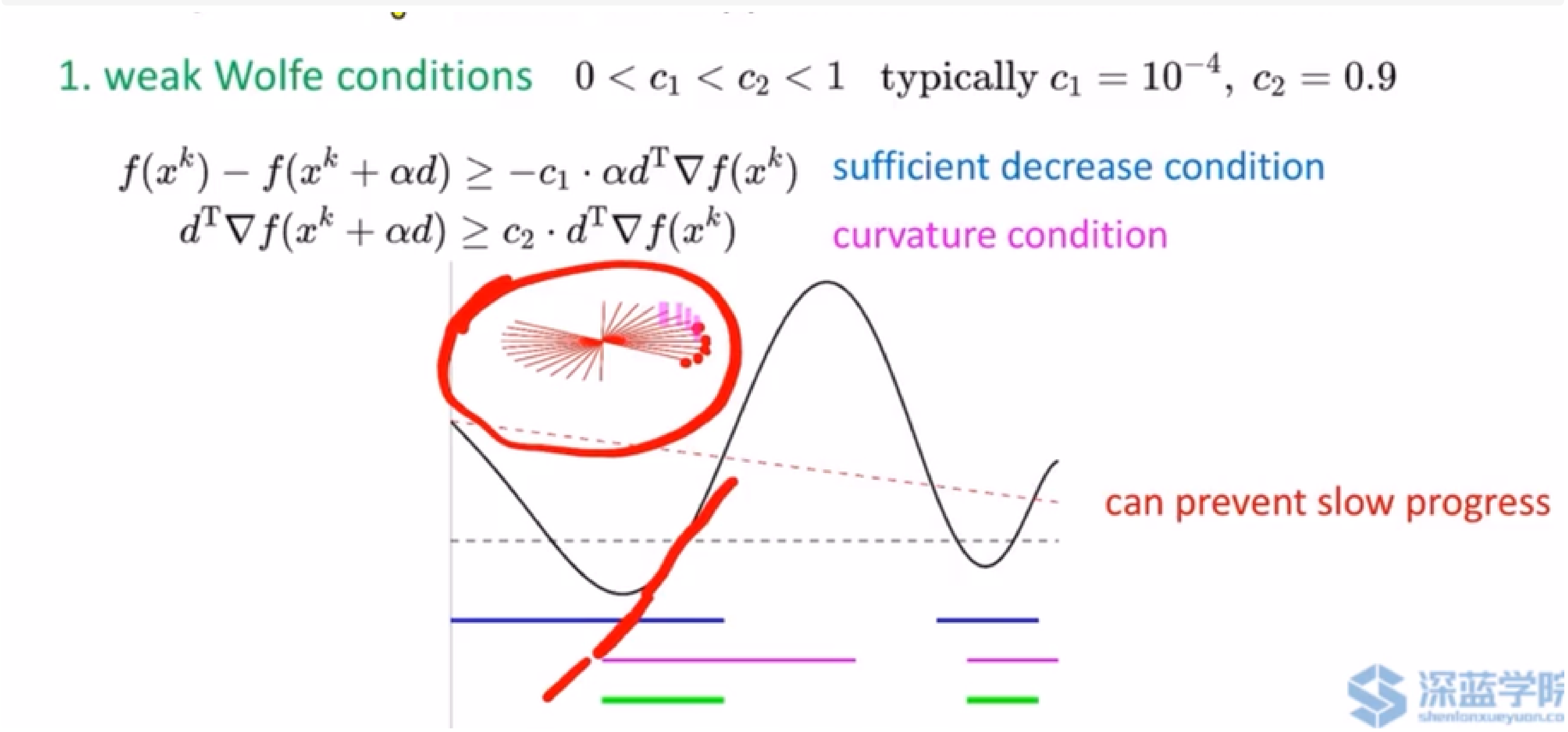
否则需要优化

```C++
else
{
    /*
    Compute the initial step:
    */
    step = 1.0 / d.norm();

    k = 1;//迭代次数
    end = 0; //当前相关变量范围 <m
    bound = 0;//范围

    while (true)
    {
        /* Store the current position and gradient vectors. */
        xp = x;
        gp = g;

        /* If the step bound can be provied dynamically, then apply it. */
        step_min = param.min_step;
        step_max = param.max_step;

        /* Search for an optimal step. */
        ls = line_search_lewisoverton(x, fx, g, step, d, xp, gp,
                                      step_min, step_max, cd, param);
```



line_search_lewisoverton

```C++
//x是变量向量 //f是x处的函数值 //g是x处的梯度值
//stp是线搜索的初始步长 //s是搜索方向向量 //xp是当前迭代的决策变量向量
//gp是当前迭代的梯度向量 //stpmin是允许的最小步长 //stpmax是允许的最大步长
//struct param包含所有必要的参数 //cd包含所有必要的回调函数
```



lewisoverton

$$B^{k+1} = \begin{cases} \left(I - \dfrac{\Delta x \Delta g^T}{\Delta g^T \Delta x}\right) B^k \left(I - \dfrac{\Delta g \Delta x^T}{\Delta g^T \Delta x}\right) + \dfrac{\Delta x \Delta x^T}{\Delta g^T \Delta x} & \text{if } \Delta g^T \Delta x > \epsilon ||g_k|| \Delta x^T \Delta x, \ \epsilon = 10^{-6} \\ B^k & \text{otherwise} \end{cases}$$

代码如下

```C++
inline int line_search_lewisoverton(Eigen::VectorXd &x,
                                    double &f,
```

```
                                        Eigen::VectorXd &g,
                                        double &stp,
                                        const Eigen::VectorXd &s,
                                        const Eigen::VectorXd &xp,
                                        const Eigen::VectorXd &gp,
                                        const double stpmin,
                                        const double stpmax,
                                        const callback_data_t &cd,
                                        const lbfgs_parameter_t &param)
{
      double c1=1.0e-4,c2=0.9;
    bool dec_cond=false,cur_cond=false;//下降条件标志，曲率条件标志
    double dec_stp=0.0,cur_stp=0.0;
    double fkp=cd.proc_evaluate(cd.instance,xp,g);//2420
    // A logic error (negative line-search step)
    if(stp<0){
        return LBFGSERR_INVALIDPARAMETERS;
    }
    for(int i=0;i<=param.max_linesearch;i++){
        if(i==param.max_linesearch){
            return LBFGSERR_MAXIMUMLINESEARCH;
        }

        x=xp+stp*s;
        f=cd.proc_evaluate(cd.instance,x,g);//cur 改变g
        // double debugr=-c1*stp*s.transpose()*gp;
        if((fkp-f)>=(-c1*stp*s.transpose()*gp))//下降 note - && 原有-迭代后的
        {
            dec_cond=true;
            dec_stp=stp;
            // double debugl=s.transpose()*g;
            //  debugr=c2*s.transpose()*gp;
            if(s.transpose()*g>= c2*s.transpose()*gp){//曲率
                cur_cond=true;

                return i;
                break;
            }
            else
            {
                cur_stp=stp;
                cur_cond=false;
            }
        }
        else{
            dec_stp=stp;
            dec_cond=false;
        }

        if(dec_cond==false){
            // 不满足下降条件需要减少步数，采用二分策略
            stp=(dec_stp+cur_stp)/2.0;
        }
        else{
            stp*=2.0;
        }
 /** The line-search step became smaller than lbfgs_parameter_t::min_step. */
        if(stp<stpmin){
            return LBFGSERR_MINIMUMSTEP;
        }

        if(stp>stpmax){
            return LBFGSERR_MAXIMUMSTEP;
        }
    }
}
```

结果

测试程序运行最终结果

```bash
Minimized Cost: 2.429e-18
Optimal Variables:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
   1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[1] + Done
```

三次样条曲线避障 + 平滑

code analysis

初始化

```cpp
int main(int argc, char **argv)
{
    ros::init(argc, argv, "curve_gen_node");
    ros::NodeHandle nh_;

    CurveGen curveGen(nh_);
    ros::Rate lr(100.0);
    while (ros::ok())
    {
        curveGen.vizObs();
        lr.sleep();
        ros::spinOnce();
    }

    return 0;
}
```

```cpp
    CurveGen(ros::NodeHandle &nh_)
        : config(ros::NodeHandle("~")),
          nh(nh_),
          visualizer(nh)
    {
        targetSub = nh.subscribe("/move_base_simple/goal", 1, &CurveGen::targetCallBack, this,
                                 ros::TransportHints().tcpNoDelay());

        path_pub_ = nh.advertise<visualization_msgs::Marker>("waypoint_generator", 1);
        route_pub_ = nh.advertise<visualization_msgs::Marker>("route_generator", 1);
        // map
        obs_info_ = config.mcircleObs;

        // ros::spin();
    }
```

一旦有起点订阅，+1， 有两次便确定触发路径,中间处理过程不改变状态

```C++
    inline void targetCallBack(const geometry_msgs::PoseStamped::ConstPtr &msg)
    {

        if (startGoal.size() >= 2)
        {
            startGoal.clear();
        }

        startGoal.emplace_back(msg->pose.position.x, msg->pose.position.y);

        // plan();
        if(plan_homework())
            ROS_INFO("Success to solve");

        return;
    }
```

```C++
bool plan_homework(){
        double max_vel = 1.0;
        int cnt=0;
        if (startGoal.size() == 2)
        {
            ROS_INFO_STREAM("it is cnt "<<(++cnt));
            const int mid_pt_num= (startGoal.back() - startGoal.front()).norm() / max_vel;
            Evx x;
            int ret = run(2*mid_pt_num,x);
            ROS_INFO("RET= %d\n",ret);
            VisPath(x);
            if(std::isnan(ret)){
                return false;
            }
            return true;
        }
        else{
            return false;
        }

    }
```

运行函数

```C++
    int run(const int N,Evx& x){
        double finalCost=0;
        x.resize(N);
        const int mid_pt_num=N/2;
        int cnt=0;
        ROS_INFO_STREAM("it is run cnt "<<(++cnt));
        getTrajMat(mid_pt_num + 1, AD_, Ac_, Ad_, AE_);

        for(int i=0;i<mid_pt_num;i++){
            //i+1 mid_pt_num+2初始末端不包含，不参与优化
            x(i)=(startGoal[1](0)-startGoal[0](0))/(mid_pt_num+2)*(i+1)+startGoal[0](0);
            x(i+mid_pt_num)=(startGoal[1](1)-startGoal[0](1))/(mid_pt_num+2)*(i+1)+startGoal[0](1);
        }
        lbfgs::lbfgs_parameter_t params;
        params.g_epsilon = 1.0e-8;
        params.past = 3;
        params.delta = 1.0e-8;

        /* Start minimization */
        int ret = lbfgs::lbfgs_optimize(x,
```

```
                                            finalCost,
                                            &CurveGen::costFunction,
                                            nullptr,
                                            &CurveGen::monitorProgress,
                                            this,
                                            params);

        //TODO 可视化

        return ret;
    }
```

construtor cubic spline

三次样条表达式及满足位置，速度，加速度连续条件

$$p_{i-1}(1) = p_i(0), p_{i-1}^{(1)}(1) = p_i^{(1)}(0), p_{i-1}^{(2)}(1) = p_i^{(2)}(0)$$

$$p_0^{(1)}(0) = 0, \quad p_n^{(1)}(1) = 0$$

$$p_i(s) = a_i + b_i s + c_i s^2 + d_i s^3, s \in [0,1] \tag{1}$$

According to the assumptions of cubic spline curves with natural boundary conditions, we have

$$C^2: \; p_{i-1}(1) = p_i(0), p_{i-1}^{(1)}(1) = p_i^{(1)}(0), p_{i-1}^{(2)}(1) = p_i^{(2)}(0), \tag{2}$$

$$p_0^{(1)}(0) = 0, \;\; p_n^{(1)}(1) = 0 \tag{3}$$

根据轨迹之间的一阶和二阶连续以及边界条件进行参数求取

$$\begin{aligned}
a_i &= x_i \\
b_i &= D_i \\
c_i &= 3(x_{i+1} - x_i) - 2D_i - D_{i+1} \\
d_i &= 2(x_i - x_{i+1}) + D_i + D_{i+1}
\end{aligned}$$

$$\begin{bmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \\ \vdots \\ D_{n-2} \\ D_{n-1} \end{bmatrix} = \begin{bmatrix} 4 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 3(x_2 - x_0) \\ 3(x_3 - x_1) \\ 3(x_4 - x_2) \\ 3(x_5 - x_3) \\ \vdots \\ 3(x_{n-1} - x_{n-3}) \\ 3(x_n - x_{n-2}) \end{bmatrix}, \text{ and } D_0 = D_N = 0$$

令 $\mathbf{x} = [x_0, x_1, \cdots, x_{N-1}, x_N]^T$ ，则

$$\mathbf{a} = \begin{bmatrix} a_0 \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 & 0 \end{bmatrix}_{N\times N+1} \mathbf{x}$$

$$\mathbf{b} = \begin{bmatrix} b_0 \\ \vdots \\ b_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & \ddots & & \\ & & & 1 & \\ & & & & 1 & 0 \end{bmatrix}_{N\times N+1} \mathbf{D}$$

$$\mathbf{D} = \begin{bmatrix} D_0 \\ \vdots \\ D_N \end{bmatrix} = \mathbf{A}_D \mathbf{x}$$

其中,

$$\mathbf{A}_D = 3 \begin{bmatrix} \mathbf{0} \\ \mathbf{D}_D \\ \mathbf{0} \end{bmatrix}_{N+1\times N-1} \begin{bmatrix} -1 & 0 & 1 & & & & \\ & -1 & 0 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & -1 & 0 & 1 & \\ & & & & -1 & 0 & 1 \end{bmatrix}_{N-1\times N+1}$$

$$\mathbf{D}_D = \begin{bmatrix} 4 & 1 & & & & & \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & 1 & 4 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 4 & 1 \\ & & & & & 1 & 4 \end{bmatrix}_{N-1\times N-1}^{-1}$$

对于系数 $\mathbf{c}$,

$$\mathbf{c}=\begin{bmatrix} c_0 \\ \vdots \\ c_{N-1} \end{bmatrix}=\mathbf{A}_c\mathbf{x}$$

其中，

$$\mathbf{A}_c=3\begin{bmatrix} -1 & 1 & & & & \\ & -1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & -1 & 1 & \\ & & & & -1 & 1 \end{bmatrix}_{N\times N+1}+\begin{bmatrix} -2 & -1 & & & & \\ & -2 & -1 & & & \\ & & \ddots & \ddots & & \\ & & & -2 & -1 & \\ & & & & -2 & -1 \end{bmatrix}_{N\times N+1}\mathbf{A}_D$$

对于系数 $\mathbf{d}$,

$$\mathbf{d}=\begin{bmatrix} d_0 \\ \vdots \\ d_{N-1} \end{bmatrix}=\mathbf{A}_d\mathbf{x}$$

其中，

$$\mathbf{A}_d=2\begin{bmatrix} 1 & -1 & & & & \\ & 1 & -1 & & & \\ & & \ddots & \ddots & & \\ & & & 1 & -1 & \\ & & & & 1 & -1 \end{bmatrix}_{N\times N+1}+\begin{bmatrix} 1 & 1 & & & & \\ & 1 & 1 & & & \\ & & \ddots & \ddots & & \\ & & & 1 & 1 & \\ & & & & 1 & 1 \end{bmatrix}_{N\times N+1}\mathbf{A}_D$$

C++

```cpp
void getTrajMat(const int N, Eigen::MatrixXd &AD, Eigen::MatrixXd &Ac, Eigen::MatrixXd &Ad, Eigen::MatrixXd &AE)
    {
        Emx A_D1 = Emx::Zero(N + 1, N - 1),
            A_D2 = Emx::Zero(N - 1, N + 1);
        Emx c_D1 = Emx::Zero(N, N + 1),
            c_D2 = Emx::Zero(N, N + 1);
        Emx d_D1 = Emx::Zero(N, N + 1),
            d_D2 = Emx::Zero(N, N + 1);
        // Evx i4=Evx::Ones(N)*4;
        // Emx D_D(i4.asDiagonal());
        Emx D_D = Emx::Identity(N - 1, N - 1);
        D_D *= 4;
        for (int i = 0; i < N; ++i)
        {
            if (i < N - 2)
            {
                D_D(i, i + 1) = 1;
                D_D(i + 1, i) = 1;
                A_D2(i, i) = -1;
                A_D2(i, i + 2) = 1;
            }
            c_D1(i, i) = -1;
            c_D1(i, i + 1) = 1;
            c_D2(i, i) = -2;
            c_D2(i, i + 1) = -1;

            d_D1(i, i) = 1;
            d_D1(i, i + 1) = -1;
            d_D2(i, i) = 1;
            d_D2(i, i + 1) = 1;
        }
        A_D1.block(1, 0, N - 1, N - 1) = D_D.inverse();
        A_D2(N - 2, N) = 1;
        A_D2(N - 2, N - 2) = -1;

        AD = 3 * A_D1 * A_D2;
        Ac = 3 * c_D1 + c_D2 * AD;
        Ad = 2 * d_D1 + d_D1 * AD;
        AE = 4 * Ac.transpose() * Ac + 12 * Ac.transpose() * Ad +
```

```
                    12 * Ad.transpose() * Ad;
    }
    static double costFunction(void *instance, const Eigen::VectorXd &x, Eigen::VectorXd &g){
        CurveGen &obj = *(CurveGen*)instance;
        const int n = x.size();
        const int mid_pt_num=n/2;
        int N=mid_pt_num+1; //
        g = Evx::Zero(n); //梯度向量
        Evx all_x=Evx::Zero(N+1),
            all_y=Evx::Zero(N+1);//mid_pt_num+2
        all_x(0) = obj.startGoal[0](0);
        all_y(0) = obj.startGoal[0](1);
        all_x(N) = obj.startGoal[1](0);
        all_y(N) = obj.startGoal[1](1);
        all_x.segment(1,mid_pt_num) = x.segment(0,mid_pt_num);//二维优化
        all_y.segment(1,mid_pt_num) = x.segment(mid_pt_num,mid_pt_num);

        double fx=0.0; //whole cost
        double potential=0.0;//potential cost
        double energy  =0.0; //energy cost
        energy +=all_x.transpose()*obj.AE_*all_x;
        energy +=all_y.transpose()*obj.AE_*all_y;
        Evx grad_seg=(obj.AE_+obj.AE_.transpose())*all_x;
        //note 只记录中间点的梯度
        g.segment(0,mid_pt_num) +=grad_seg.segment(1,mid_pt_num);
        grad_seg=(obj.AE_+obj.AE_.transpose())*all_y;
        g.segment(mid_pt_num,mid_pt_num)+=grad_seg.segment(1,mid_pt_num);
        Eigen::Vector2d pt;//记录点
        Eigen::Vector2d dis_obs;//距离obs
        for(int i=0;i<mid_pt_num;i++){

            pt(0) = x(i);
            pt(1) = x(i+mid_pt_num);
            for(int j=0;j<obj.obs_info_.rows();j++){
                dis_obs(0)=pt(0)-obj.obs_info_(j,0);
                dis_obs(1)=pt(1)-obj.obs_info_(j,1);
                double sub_potentil=obj.obs_info_(j,2)-dis_obs.norm();
                if(sub_potentil>0){
                    potential+=sub_potentil;
                    //cal grad
                    g(i)-=1000*dis_obs(0)/dis_obs.norm();
                    g(i+mid_pt_num)-=1000*dis_obs(1)/dis_obs.norm();
                }


            }

        }
        fx=1000*potential+energy;
        return fx;

    }
```

ros::TransportHints Class Reference

objective function

如果需要曲线尽量平滑，那么需要求曲线的二阶导数积分（需要取绝对值），评估**每段**起始点变化率

代价函数

$$\text{Energy}\left(x_1, x_2, \ldots, x_{N-1}\right)=\sum_{i=0}^{N-1} \int_0^1\left\|p_i^{(2)}(s)\right\|^2 \mathrm{~d} s$$

其中

$p_i^{(2)}(s) = 2c_i + 6d_i s$

$\left|p_i^{(2)}(s)\right| = 4c_i^2 + 24c_i d_i s + 36d_i^2 s^2$

$E_i = \int_0^1 \left|p_i^{(2)}(s)\right|^2 \, \mathrm{d}s = 4c_i^2 + 12c_i d_i + 12d_i^2$

则

$$\mathbf{E} = 4\mathbf{c}^T\mathbf{c} + 12(\mathbf{c}^T\mathbf{d} + \mathbf{d}^T\mathbf{d}) = \mathbf{x}^T(4\mathbf{A}_c^T\mathbf{A}_c + 12\mathbf{A}_c^T\mathbf{A}_d + 12\mathbf{A}_d^T\mathbf{A}_d)\mathbf{x} = \mathbf{x}^T\mathbf{A}_E\mathbf{x}$$

梯度

$$\frac{\partial \mathbf{E}}{\partial \mathbf{x}} = (\mathbf{A}_E + \mathbf{A}_E^T)\mathbf{x}$$

对于二维的情况，令 $\mathbf{x} = [x_0, \cdots, x_N, y_0, \cdots, y_N]^T_{2(N+1)\times 1}$ ，则

$$\mathbf{E} = \mathbf{x}^T\mathbf{A}_E^{'}\mathbf{x} = \mathbf{x}^T \begin{bmatrix} \mathbf{A}_E & \\ & \mathbf{A}_E \end{bmatrix}_{2(N+1)\times 2(N+1)} \mathbf{x}$$

$$\frac{\partial \mathbf{E}}{\partial \mathbf{x}} = \begin{bmatrix} \mathbf{A}_E + \mathbf{A}_E^T & \\ & \mathbf{A}_E + \mathbf{A}_E^T \end{bmatrix}_{2(N+1)\times 2(N+1)} \mathbf{x}$$

由于存在障碍物，需要引入 Potential 函数

$$\text{Potential}\,(x_1, x_2 \ldots, x_{N-1}) = 1000 \sum_{i=1}^{N-1} \sum_{j=1}^{M} \max\left(r_j - \|x_i - o_j\|, 0\right)$$

$$\frac{\partial P}{\partial x} = 1000 \begin{bmatrix} \sum_{j=1}^M g_{1,j} \\ \vdots \\ \sum_{j=1}^M g_{N-1,j} \end{bmatrix}$$

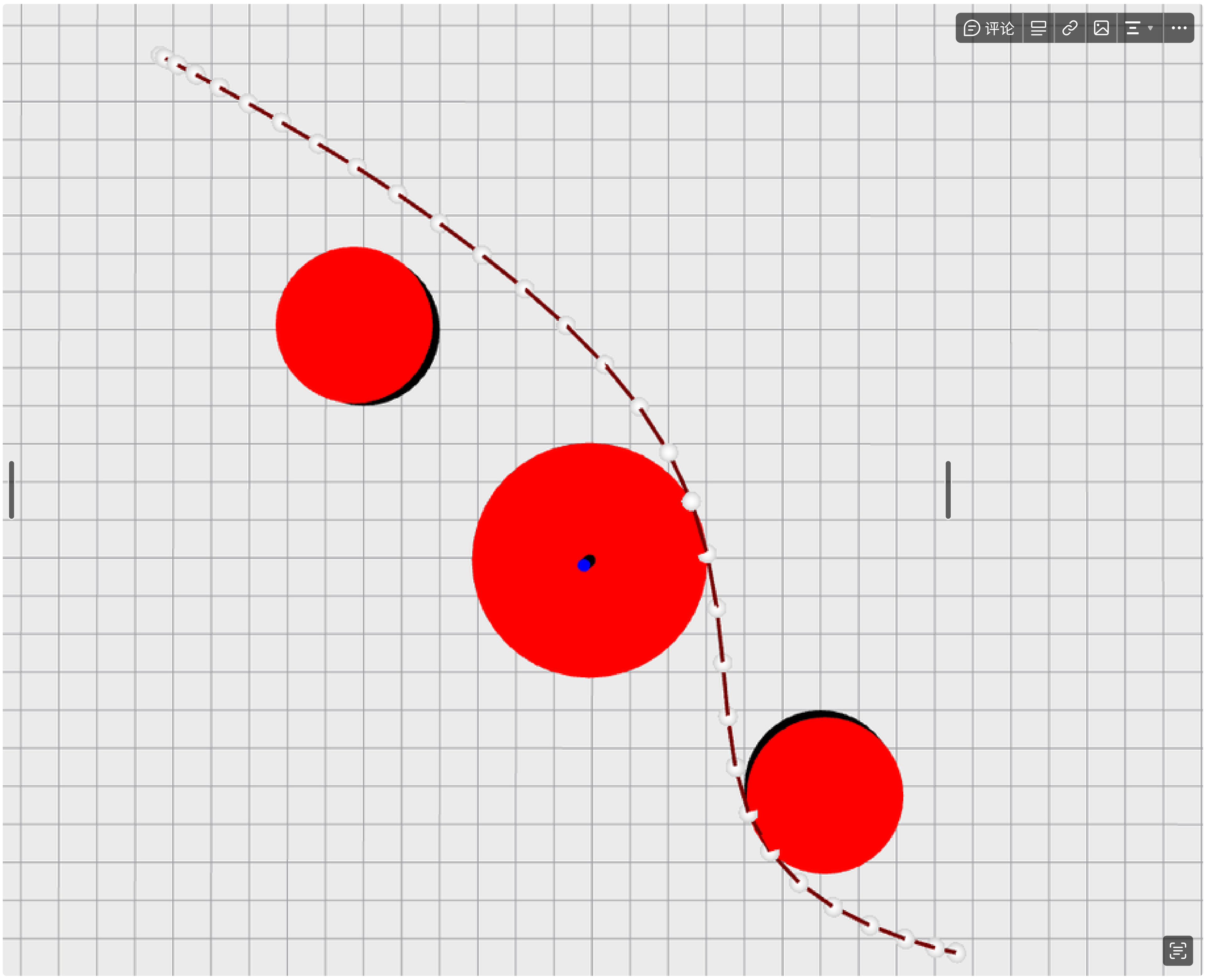
其中

$$g_{i,j} = \begin{cases} -\frac{x_i - o_j}{\|x_i - o_j\|} & , \text{if } r_j - \|x_i - o_j\| > 0 \\ 0 & , \text{otherwise} \end{cases}$$

```C++
static double costFunction(void *instance, const Eigen::VectorXd &x, Eigen::VectorXd &g){
        CurveGen &obj = *(CurveGen*)instance;
        const int n = x.size();
        const int mid_pt_num=n/2;
        int N=mid_pt_num+1; //
        g = Evx::Zero(n); //梯度向量
        Evx all_x=Evx::Zero(N+1),
            all_y=Evx::Zero(N+1);//mid_pt_num+2
        all_x(0) = obj.startGoal[0](0);
        all_y(0) = obj.startGoal[0](1);
        all_x(N) = obj.startGoal[1](0);
        all_y(N) = obj.startGoal[1](1);
        all_x.segment(1,mid_pt_num) = x.segment(0,mid_pt_num);//二维优化
        all_y.segment(1,mid_pt_num) = x.segment(mid_pt_num,mid_pt_num);

        double fx=0.0; //whole cost
        double potential=0.0;//potential cost
        double energy  =0.0; //energy cost
```

```
        energy +=all_x.transpose()*obj.AE_*all_x;
        energy +=all_y.transpose()*obj.AE_*all_y;
        Evx grad_seg=(obj.AE_+obj.AE_.transpose())*all_x;
        //note 只记录中间点的梯度
        g.segment(0,mid_pt_num) +=grad_seg.segment(1,mid_pt_num);
        grad_seg=(obj.AE_+obj.AE_.transpose())*all_y;
        g.segment(mid_pt_num,mid_pt_num)+=grad_seg.segment(1,mid_pt_num);
        Eigen::Vector2d pt;//记录点
        Eigen::Vector2d dis_obs;//距离obs
        for(int i=0;i<mid_pt_num;i++){

            pt(0) = x(i);
            pt(1) = x(i+mid_pt_num);
            for(int j=0;j<obj.obs_info_.rows();j++){
                dis_obs(0)=pt(0)-obj.obs_info_(j,0);
                dis_obs(1)=pt(1)-obj.obs_info_(j,1);
                double sub_potentil=obj.obs_info_(j,2)-dis_obs.norm();
                if(sub_potentil>0){
                    potential+=sub_potentil;
                    //cal grad
                    g(i)-=1000*dis_obs(0)/dis_obs.norm();
                    g(i+mid_pt_num)-=1000*dis_obs(1)/dis_obs.norm();
                }


            }

        }
        fx=1000*potential+energy;
        return fx;

    }
```

Result



Problem

在生成障碍物时，我想要动态赋值 Eigen::Matrix，但却报错

`mcircleObs=Eigen::Map<Eigen::Matrix<double, Eigen::Dynamic, 3, Eigen::RowMajor>>(circleObsVec.data());`

不得不使用

`mcircleObs=Eigen::Map<Eigen::Matrix<double, 3, 3, Eigen::RowMajor>>(circleObsVec.data());`

reference

https://reference.wolfram.com/language/tutorial/UnconstrainedOptimizationOverview.html

https://zhuanlan.zhihu.com/p/269230598

https://en.wikiversity.org/wiki/Cubic_Spline_Interpolation

https://mathworld.wolfram.com/SmoothCurve.html

三次样条推导

https://www.cnblogs.com/xpvincent/archive/2013/01/26/2878092.html

-
-
-
-
-

新页面