本章作业内容分为 3 次样条曲线拟合与 LBFGS 优化

1. 三次样条曲线拟合

   a) 初始化参数

      确定不同数据的数目：

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

      - 曲线的数量 N
      - D 的数量 N+1
      - b 的数量 N-1
      - 矩阵的大小 (N-1)*(N-1)
      - 带状矩阵大小，上带宽 1，下带宽 1

```cpp
public:
  // 设置边界条件,起点与终点 + 曲线数量
  inline void setConditions(const Eigen::Vector2d &headPos,
                            const Eigen::Vector2d &tailPos,
                            const int &pieceNum) {
    headP = headPos;
    tailP = tailPos;
    N = pieceNum;
    // TODO
    // 确定各参数的数量
    // 所有点的数量N+1
    // 曲线数量N
    // 中间节点数量N-1
    // 求解的多项式矩阵维度 (N-1)*(N-1)
    // b矩阵的维度N-1
    // D的数量N+1

    // 设置B矩阵的数量与维度
    b.resize(N - 1, 2);

    // 设置A矩阵的维度与带状矩阵，上下带宽为1
    A.create(N - 1, 1, 1);

    return;
  }
```

   b) 计算多项式系数

      - 所有点的数量 N+1,构建 X 矩阵存储所有的点

```cpp
// x(0) = headP
// x(N) = tailP
// x(i+1) = inPs(i) (0<i<N)
Eigen::Matrix2Xd X; //将起点，中间点，进行整合
X.resize(2, N + 1);
X.col(0) = headP;
X.col(N) = tailP;
for (int i = 1; i < N; i++) {
  X.col(i) = inPs.col(i - 1);
}
```

- 填充 A 矩阵与 B 矩阵，进行方程组求解

```cpp
// 补充A矩阵
for (int i = 0; i < N - 1; ++i) {
  A(i, i) = 4;  // A矩阵每个对象线都是4
  if (i == 0) { // 第一行数据
    A(i, i + 1) = 1;
  } else if (i == N - 2) { //最后一行数据
    A(i, i - 1) = 1;
  } else { //中间的矩阵块
    A(i, i - 1) = 1;
    A(i, i + 1) = 1;
  }
  // b(i) = 3 *(x(i+2) - x(i))
  b.row(i) = 3 * (X.col(i + 2) - X.col(i)).transpose();
}

A.factorizeLU(); // A矩阵进行LU分解
A.solve(b);      //求解D
```

- 存储 D 矩阵

```cpp
// 存储D
Eigen::MatrixX2d D = Eigen::MatrixX2d::Zero(N + 1, 2);
for (int i = 1; i < N; i++) {
  D.row(i) = b.row(i - 1);
}
```

- 计算系数

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

```cpp
Eigen::Matrix<double, 2, 4> coeffMat; //单个系数矩阵

for (int i = 0; i < N; ++i) {
  // a(i) = x(i)
  coeffMat.col(3) = X.col(i);
  // b(i) = D(i)
  coeffMat.col(2) = D.row(i).transpose();
  // c(i) = 3(x(i+1) - x(i)) - 2D(i) - D(i+1)
  coeffMat.col(1) = 3 * (X.col(i + 1) - X.col(i)) -
                    2 * D.row(i).transpose() - D.row(i + 1).transpose();
  // d(i) = 2(x(i) - x(i+1)) + D(i) + D(i+1)
  coeffMat.col(0) = 2 * (X.col(i) - X.col(i + 1)) + D.row(i).transpose() +
                    D.row(i + 1).transpose();
  b.block(i * 4, 0, 4, 2) = coeffMat.transpose();
}

return;
```

- 获取拟合曲线

```cpp
inline void getCurve(CubicCurve &curve) const {
  // TODO
  curve.clear();
  for (int i = 0; i < N; ++i) {
    curve.emplace_back(
        CubicPolynomial(1.0, b.block(i * 4, 0, 4, 2).transpose()));
  }
  return;
}
```

- 计算 StretchEnergy

```cpp
inline void getStretchEnergy(double &energy) const {
  // TODO
  // p(s) = ai + bi*s + ci * s^2 + di * s^3
  // p''(s) = 2 * ci  +  6 * di * s
  // (p''(s))^2 = 4 * ci ^2 + 36 * di ^2 * s ^ 2 + 24 * ci * di * s

  // 定积分的值 = 4 * ci ^2 + 12 * di ^2 + 12 * * ci * di
  energy = 0;
  Eigen::Vector2d c, d;
  Eigen::Matrix<double, 2, 4> coeffMat;
  for (int i = 0; i < N; i++) {
    coeffMat = b.block(i * 4, 0, 4, 2).transpose();
    d = coeffMat.col(1);
    c = coeffMat.col(0);
    energy +=
        4.0 * c.squaredNorm() + 12.0 * d.squaredNorm() + 12.0 * d.dot(c);
  }

  return;
}
```

- 计算梯度，暂无

```cpp
inline void getGrad(Eigen::Ref<Eigen::Matrix2Xd> gradByPoints) const {
  // TODO
  // grad = 8 * ci * ci' + 24 * di * di' + 12 * di * ci' + 12 * di' * ci
  // grad = (8 * ci + 12 * di) * ci' + (24 * di + 12 * ci)* di'
  Eigen::MatrixXd partial_c = Eigen::MatrixXd::Zero(N, N - 1);
  Eigen::MatrixXd partial_d = Eigen::MatrixXd::Zero(N, N - 1);
  //
  partial_c.row(0) = -3 * partial_diff_x.row(0) - partial_D.row(0);
  partial_d.row(0) = 2 * partial_diff_x.row(0) + partial_D.row(0);
  for (int i = 1; i < N - 1; ++i) {
    partial_c.row(i) = -3 * partial_diff_x.row(i) - 2 * partial_D.row(i - 1)
                       partial_D.row(i);
    partial_d.row(i) =
        2 * partial_diff_x.row(i) + partial_D.row(i - 1) + partial_D.row(i);
  }
  partial_c.row(N - 1) =
      -3 * partial_diff_x.row(N - 1) - 2 * partial_D.row(N - 2);
  partial_d.row(N - 1) = 2 * partial_diff_x.row(N - 1) + partial_D.row(N - 2)

  gradByPoints.setZero();
  Eigen::Vector2d c, d;
  Eigen::Matrix<double, 2, 4> coeffMat;

  for (int i = 0; i < N; ++i) {
    coeffMat = b.block(i * 4, 0, 4, 2).transpose();
    d = coeffMat.col(0);
    c = coeffMat.col(1);
    gradByPoints += (24 * d + 12 * c) * partial_d.row(i) +
                    (12 * d + 8 * c) * partial_c.row(i);
  }
```

- 与障碍物的代价与梯度

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

计算距离

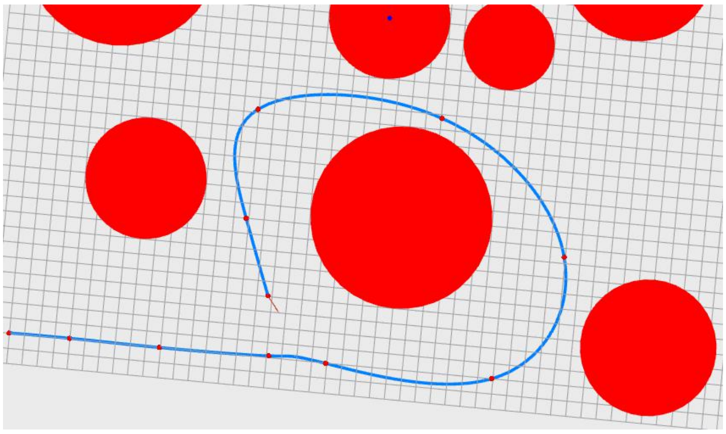
```
// 把二维的点变成一维
static inline double costFunction(void *ptr, const Eigen::VectorXd &x,
                                  Eigen::VectorXd &g) {
  // TODO
  std::cout << "in costFunction " << std::endl;
  // std::cout << "x is " << std::endl;
  // PRINT_MATRIX(x);
  double cost{0}, cost_engrgy{0}, cost_obstacle{0};

  // 静态成员函数 访问非静态成员需要传递对象指针变量，*ptr应该是用this指针
  auto ins = reinterpret_cast<path_smoother::PathSmoother *>(ptr);
  // std::cout << "get instance " << std::endl;

  // 更新 points 与 gradByPoints
  int in_ps_size = x.size() / 2;
  ins->points.row(0) = x.head(in_ps_size).transpose();
  ins->points.row(1) = x.tail(in_ps_size).transpose();
  // PRINT_MATRIX(ins->points);

  // 更新三次样条曲线中的中间节点
  ins->cubSpline.setInnerPoints(ins->points);
  // std::cout << "setInnerPoints " << std::endl;
  // 更新三次样条曲线的能量
  ins->cubSpline.getStretchEnergy(cost_engrgy);
  ins->gradByPoints.setZero();
```

- 曲线拟合截图,编写了程序，测试样条曲线的生成情况

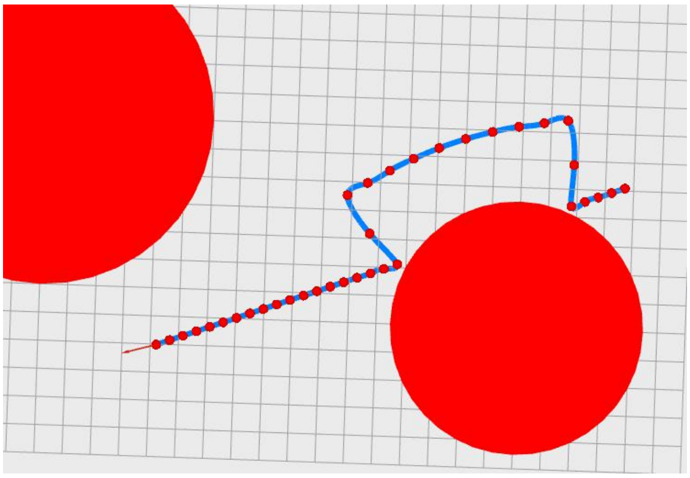
```cpp
for (int i = 0; i < ins->points.cols(); i++) {
  // 找到每个节点的最大障碍物
  for (int j = 0; j < ins->diskObstacles.cols(); j++) {
    // point_circle_center_vec
    // 是点与圆心的距离的向量，也是需要原理的梯度gxng 方向
    auto point_circle_center_vec =
        ins->points.col(i) - ins->diskObstacles.col(j).head(2);
    // 点到圆心的距离
    double point_dis_to_circle_center = point_circle_center_vec.norm();

    // 点越在圆的内部，代价越大
    double point_cost =
        ins->diskObstacles.col(j).z() - point_dis_to_circle_center;

    // 点的圆的内部，有代价，在圆的外部代价为0
    // max{r(j) - ||x(i) - o(j)||, 0}
    if (point_cost > 0) {
      // 增加惩罚系数
      cost_obstacle += ins->penaltyWeight * point_cost;
      // 与障碍物的梯度计算，
      ins->gradByPoints.col(i) +=
          ins->penaltyWeight *
          (-point_circle_center_vec / point_dis_to_circle_center);
    }
  }
}
```

```cpp
  // PRINT_MATRIX(ins->gradByPoints);

  cost = cost_engrgy + cost_obstacle;
  // 将二维的梯度转为一维
  // if (g.size() != x.size()) {
  //   g.resize(x.size());
  // }
  g.setZero();
  g.head(in_ps_size) = ins->gradByPoints.row(0).transpose();
  g.tail(in_ps_size) =  ins->gradByPoints.row(1).transpose();
  // for (int i = 0; i < ins->gradByPoints.cols(); i++) {
  //   g(2 * i) = ins->gradByPoints.col(i)[0];
  //   g(2 * i + 1) = ins->gradByPoints.col(i)[1];
  // }

  static int count = 0;
  std::cout << "count is" << count++ << std::endl;
  return cost;
}
```

- 曲线拟合截图,编写了程序，测试样条曲线的生成情况

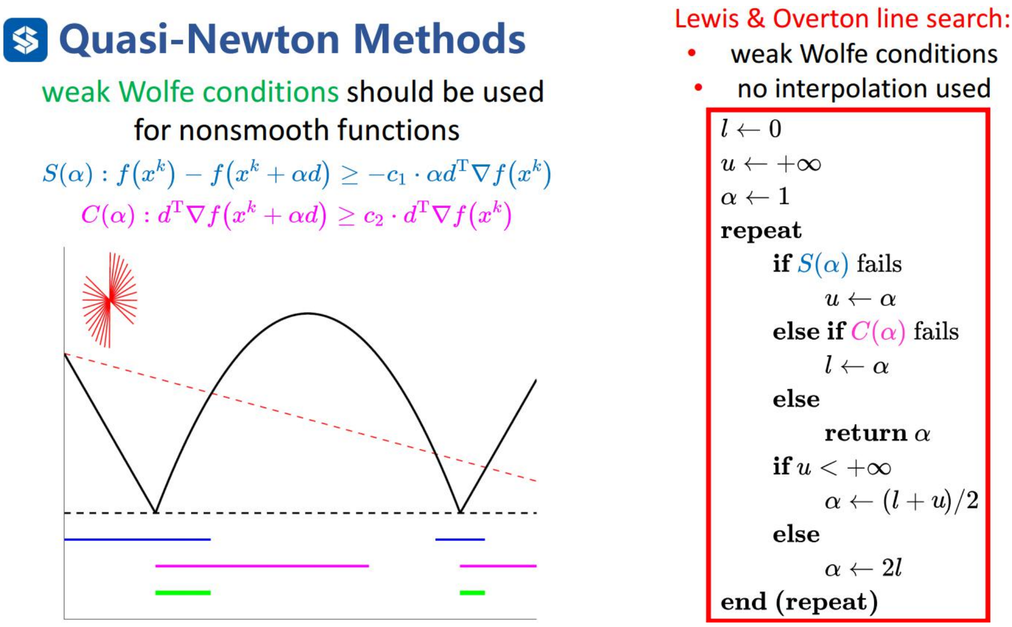


未使用样条曲线梯度



2. LBGFS 优化

主要完成 line_search_lewisoverton，参考汪博开源的 gcopter 项目并参考课程中的课件完成



- 首先初始化左右边界

```
// 寻找 weak wolfe condition 点
// 需要满足 s_alpha 和 c_alpha条件，得到一个搜索的区间
// 参考汪博开源代码

/* Check the input parameters for errors. */
if (!(stp > 0.0)) {
  return LBFGSERR_INVALIDPARAMETERS;
}

int count = 0;
double f_val_init = f;       //初始的函数值
double dg_init = gp.dot(s); // 初始的d * g

if (dg_init > 0) {
  return LBFGSERR_INCREASEGRADIENT;
}

// s_alpha条件
double s_alpha = param.f_dec_coeff * dg_init;
// c_alpha条件
double c_alpha = param.s_curv_coeff * dg_init;
double l = 0;            //初始左边界
double u = stpmax;       //初始右边界
bool brackt = false, touched = false;
```

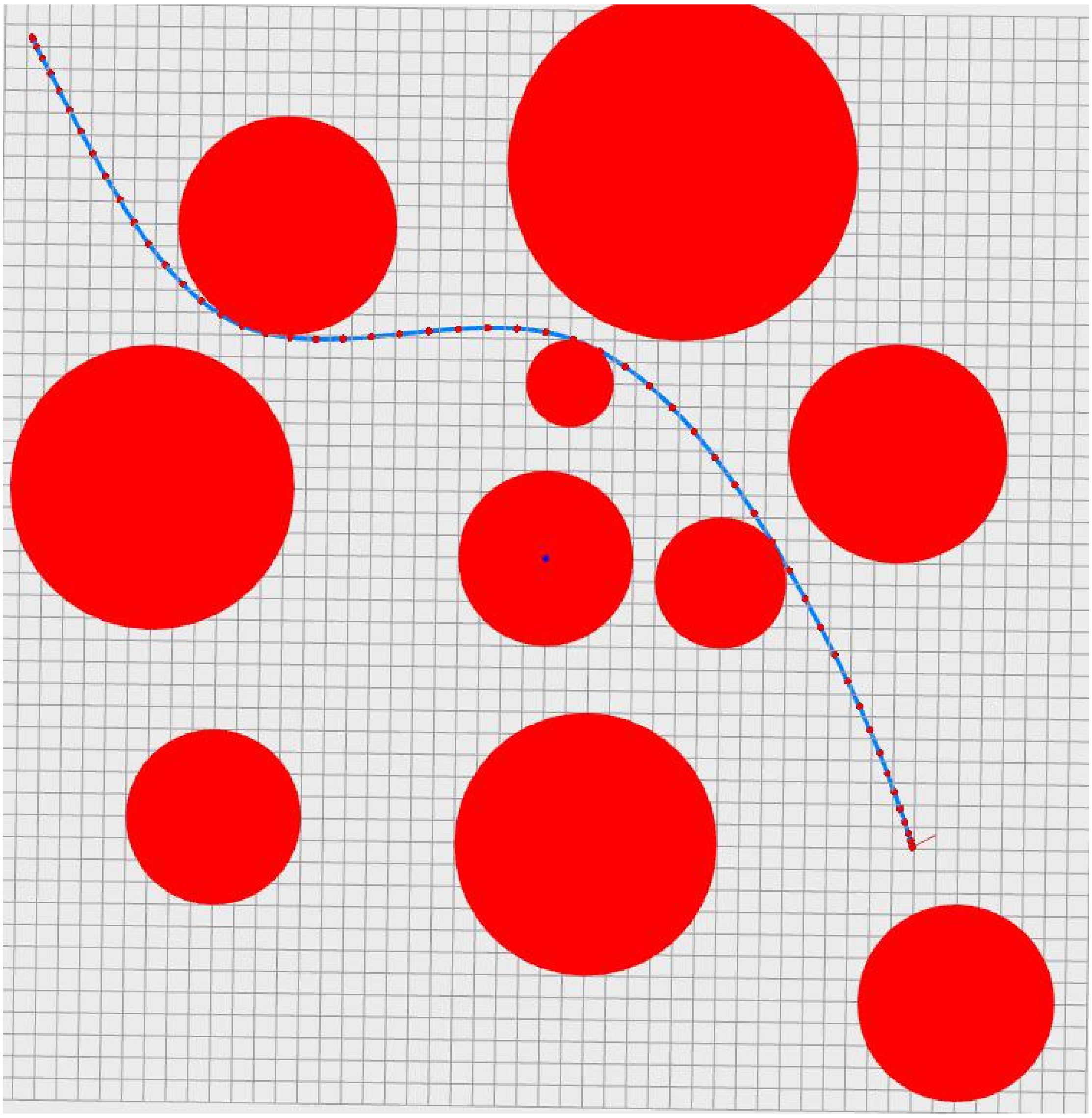
- 判断是否满足 alpha 条件

```cpp
// 选择区间的循环
while (true) {
  // 更新 函数值f， 目标向量， 梯度
  x = xp + stp * s;
  f = cd.proc_evaluate(cd.instance, x, g); //
  ++count;
  // 检查函数值
  if (std::isinf(f) || std::isnan(f)) {
    return LBFGSERR_INVALID_FUNCVAL;
  }
  // 判断是否满足 s_alpha条件
  if (f - f_val_init > stp * s_alpha) {
    // 更新右边界
    u = stp;
    brackt = true;
  } else {
    // 判断是否满足c_alpha条件
    if (g.dot(s) > c_alpha) {
      return count;
    } else {
      l = stp;
    }
  }
```

- 异常处理直接用汪博代码中

```cpp
  // 检查是否超出迭代次数
  if (count >= param.max_linesearch) {
    return LBFGSERR_MAXIMUMLINESEARCH;
  }
  // 间距太小了
  if (brackt && (u - l) < param.machine_prec * u) {
    return LBFGSERR_WIDTHTOOSMALL;
  }
  // 定义域缩减与扩大
  if (brackt) {
    stp = 0.5 * (u + l);
  } else {
    stp *= 2.0;
  }
  // 步长太小了
  if (stp < stpmin) {
    return LBFGSERR_MINIMUMSTEP;
  }
  // 步长太长了
  if (stp > stpmax) {
    // 根据源代码修改，还要多检查一次
    if (touched) {
      return LBFGSERR_MAXIMUMSTEP;
    } else {
      touched = true;
      stp = stpmax;
    }
  }
}
```

- 最终实现效果



```
dt is 0.0561819
```