QUASI NEWTONS METHOD

1 L-BFGS 优化算法

BFGS 算法是由四位数学家名字的首字母命名的方法，是一种被认为是最有效的拟牛顿方法，其核心思想在于运用 BFGS 公式来不断更新拟合 Hessian 矩阵的值，通过这个拟合出来的 Hessian 矩阵实现经典牛顿法中对迭代点的二阶泰勒展开，从而用较少的计算量获得一个十分令人满意的下降步长。

但 BFGS 拟合 Hessian 矩阵需要运用到以往所有迭代点的梯度信息，使得拟合出的 Hessian 矩阵随着迭代次数的增加而愈发稠密，在面对大规模问题时会显著增加求解时间，因此需要设置一个 Buffer，不让过老的信息参与拟合，从而使拟合出的 Hessian 矩阵保持轻快，这种方法也被称为 Limitted-Memory BFGS (L-BFGS)。

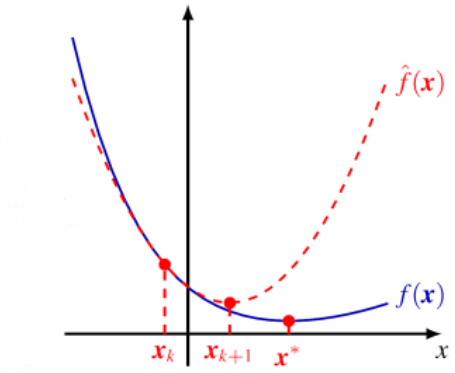
1.1 经典牛顿方法

经典牛顿方法是通过在迭代点处对函数进行二阶泰勒展开来拟合函数在 x^k 处的曲率特性：

$$f(x) \approx f(x_k) + (x - x_k)^T g + \frac{1}{2}(x - x_k)^T H (x - x_k)$$

视 $(x - x_k)$ 为自变量，可求得拟合函数最低点的表达式，同时也是牛顿法的更新公式：

$$x = x_k - H^{-1} g^k$$



对于 Hessian 矩阵正定的函数，牛顿法可以在极少的迭代步内收敛，但很多优化问题所构建的函数并没有正定的 Hessian 矩阵，且对于大规模问题，Hessian 矩阵求解的时间复杂度相当大。从评价优化函数性能的三个维度看来，牛顿法虽然 Convergence Speed 令人满意，但 Stability 和 Computation Work 在实际工程中都一塌糊涂，因此引入了拟牛顿方法，在不直接求解 Hessian 矩阵的前提下充分利用函数的高阶信息。

1.2 BFGS 拟牛顿方法

拟牛顿方法的核心思想同牛顿方法，也要做函数在迭代点处的二阶近似，但区别于牛顿方法中直接求解 Hessian 矩阵，拟牛顿方法中的 Hessian 矩阵 M 是一个近似值：

$$f(x) - f(x^k) = (x - x^k)^T g^k + \frac{1}{2}(x - x^k)^T M^k (x - x^k)$$

$$\rightarrow \text{Solve } \ M^k d^k = -g^k$$

对于拟合的 Hessian 矩阵 M，我们需要它满足以下条件：

- 为了简化计算，我们并不需要 M 包含所有二阶偏导数
- 设计的 M 应使线性方程组有闭式解（可以用公式表达的解析解）
- 需构造一个稀疏的矩阵，只在重要的部分对 Hessian 近似
- Quasi Newton Step 必须要让函数下降 → 下降方向和梯度负方向应呈锐角
- 需要包含曲率信息，需要逼近局部函数的二阶信息

1.2.1 下降方向和梯度方向呈锐角

为了让函数充分下降，需要保证下降方向 $M^k d^k$ 和梯度负方向 $-g$ 呈锐角：

$$\langle -g, -M^{-1} g \rangle = \langle g, M^{-1} g \rangle = g^T M^{-1} g > 0$$

所以，M^{-1} 需要是正定矩阵，相应的，M 也应当是正定矩阵方能保证函数是下降的。

1.2.2 拟合函数的曲率信息

为了拟合函数在迭代点处的曲率信息，可以对函数的梯度进行二阶泰勒展开：

$$\nabla f(x + p) \approx \nabla f(x^{k+1}) + \nabla^2 f(x) p$$

令 $y = x + p$，则有：

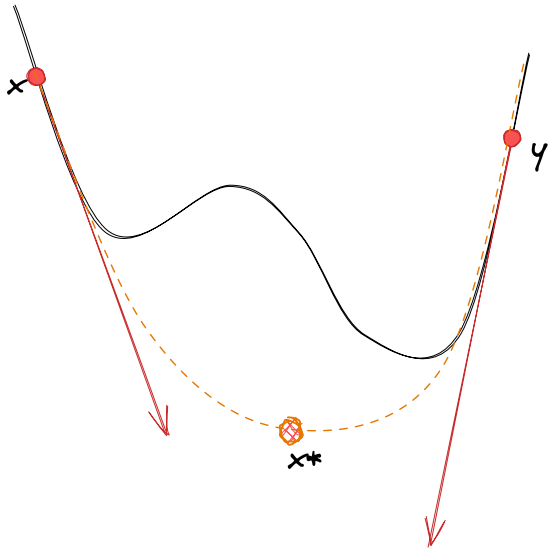
$$\nabla f(x) - \nabla f(y) \approx H(x - y)$$

这个方程也被称为正割条件（Secant Condition），拟合出来的 Hessian 矩阵需要满足 Secant Condition 的约束：$\Delta g \approx M^{k+1} \Delta x$

局部的约束必然会或多或少的反应一些函数的曲率特征，若收集一些列 Δx 和 Δg，则可以通过求解线性方程组 $\Delta g \approx M^{k+1} \Delta x$ 来得到 M 的估计值，并且这个估计值会随着更新次数的增多而逐渐逼近真实的 Hessian 矩阵。

为了提高求解效率，可以一步到位，直接估计 M^{-1}，省的求解完 M 后还要再求他的逆矩阵。令 $B = M^{-1}$，则 Secant Condition 可以表示为 $\Delta x \approx B^{k+1} \Delta g$

当 x 和 y 隔的很远时，拟合出来的结果可以认为是下图的形式：



所以说，不管说 x 和 y 的插值如何，这种拟合方式对函数的收敛是有积极意义的

1.2.3 BFGS 更新公式

对于 Secant Condition 公式，满足约束的 B^{k+1} 有很多个，因此要从中挑出一个最优的。传统方法是找到和 B^k 最接近的 B^{k+1}，并保证 B^{k+1} 是对称矩阵，但这样建立的优化问题会导致一些计算上的问题， 当 B 在有些方向尺度很大但有些方向尺度很小时，我们需要度量的是相对的 Error，所以需要用更新前后 Hessian 矩阵的均值对 B 进行归一化处理：

$$\begin{aligned} &\min ||H^{\frac{1}{2}}(B-B^k)H^{\frac{1}{2}}||^2 \\ \text{s.t.}\quad &B=B^T \\ &\Delta x = B\Delta g \end{aligned}$$

虽然 Hessian 矩阵是我们要拟合的对象，但现在又要用 Hessian 来进行归一化，这个看似矛盾的问题实则是有解的，Hessian 矩阵的均值有确切的解析解：

$$H=\int_0^1 \nabla^2 f[(1-\tau)x^k+\tau x^{k+1}]\,\mathrm{d}x$$

经过一番推导，可以得到 B 的更新公式，也称 BFGS 公式：

$$B^{k+1}=\left(I-\frac{\Delta x\Delta g^T}{\Delta g^T\Delta x}B^k\right)\left(I-\frac{\Delta g\Delta x^T}{\Delta g^T\Delta x}B^k\right)+\frac{\Delta x\Delta x^T}{\Delta g^T\Delta x}$$

在优化的初期，可以取 $B^0=I$，通过不断的迭代更新，可以令 B 不断逼近真值

对于 1.2 节开头提出的 M 矩阵需满足的条件，BFGS 更新公式无法保证下降方向外，其余条件都满足。在 1.2.1 节中又给出了证明，只要 B 矩阵正定，则 Newton Step 一定是下降方向，因此可以从这条结论下手，对 BFGS 公式加以修正：

> BFGS 更新的初值 B^0 可以很容易保证正定，而后续更新 B 时只需保证 $\Delta g^T\Delta x>0$ 即可确定 B 矩阵正定。（证明后续补上）

1.2.3.1 凸且光滑函数的 BFGS 优化

对于严格凸（Strictly Convex）的函数，$\Delta g^T \Delta x > 0$ 是一定满足的，因此 BFGS 的更新流程如下：

1.2.3.2 非凸但光滑函数的 BFGS 优化

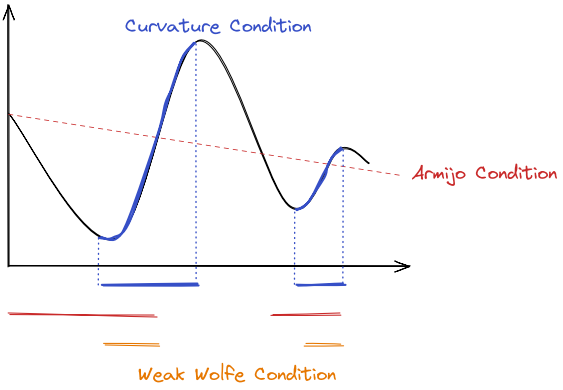
为了让函数的搜索方向满足 $\Delta g^T \Delta x > 0$ 的约束，我们引入了 Wolfe Conditions，只要在线搜索式满足 Wolfe Condition，则一定满足不等式约束。

1. Weak Wolfe Conditon:
可视为 Armijo Condition 和 Curvature Condition 的结合，需满足：$0 < c_1 < c_2 < 1$，一般取 $c_1 = 10^{-4}$，$c_2 = 0.9$

$$f(x^k) - f(x^k + \alpha d) \geq -c_1 \alpha d^T \nabla f(x^k) \qquad \text{Armijo Condition}$$

$$d^T \nabla f(x^k + \alpha d) \geq c_2 d^T \nabla f(x^k) \qquad \text{Curvature Condition}$$

几何意义为：在满足充分下降条件的前提下挑选出处于 “上坡” 处的迭代点，从而使得线搜索的 Progress 足够大。Wolfe Condition 就是 Armijo 和 Curvature 的交集。

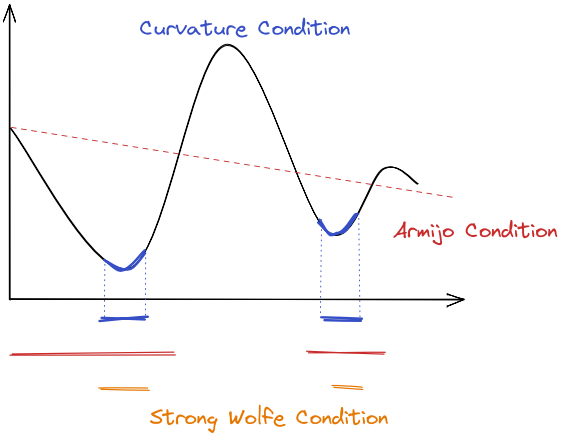


2. Strong Wolfe Conditon:
相较于 Weak 版本，Strong 版本对曲率的限制更加严格，需要迭代点处于更靠近山谷的地方，需满足：$0 < c_1 < c_2 < 1$，一般取 $c_1 = 10^{-4}$，$c_2 = 0.9$

$$f(x^k) - f(x^k + \alpha d) \geq -c_1 \alpha d^T \nabla f(x^k) \qquad \text{Armijo Condition}$$

$$d^T \nabla f(x^k + \alpha d) \geq c_2 d^T \nabla f(x^k) \qquad \text{Curvature Condition}$$

Strong Wolfe Condition 可以避免迭代过程中出现的震荡，并且满足 Strong Wolfe Condition 则一定满足 Weak Wolfe Condition



在实际工程中，两者的性能相似，而 Weak Wolfe Condition 更为常用，并且也适用于一些病态函数的优化。

此外，为了保证函数一定可以收敛到最低点，Li 和 Fukushima 提出了一个 Cautious Update 的方法可以保证 BFGS 方法必然收敛到梯度为 0 的点：

$$B^{k+1} = \begin{cases} \text{LBFGS Update}, & \text{if } \Delta g^T \Delta x > \varepsilon ||g_k|| \Delta x^T \Delta x \varepsilon = 10^{-6} \\ B^k, & \text{otherwise} \end{cases}$$

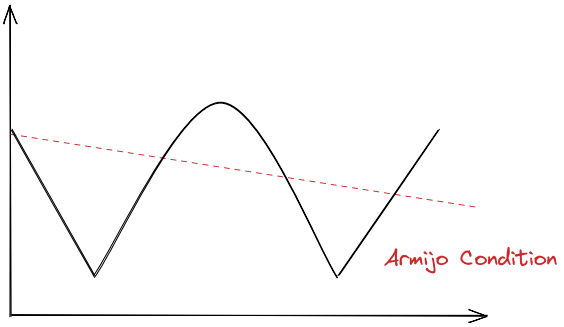
每次在执行完 Wolfe Condition 后判断以下是否满足 Cautions Update 条件，若不满足则不更新 Hessian 矩阵，以此来保证全局稳定性。

1.2.3.3 非凸非光滑函数的 BFGS 优化

非光滑函数主要面临的问题有：

- 梯度不一定存在
- 负梯度方向不一定是下降方向
- 曲率可能会无穷大 → Hessian 矩阵的条件数增大

若直接将 BFGS 方法应用到非凸非光滑函数的优化，会导致 Strong Wolfe Condition 找不到解。由于梯度不连续，所以强曲率条件没办法将曲率压的很平，在尖点附近的曲率并没有接近 0 的值，但 Weak Wolfe Condition 可以满足。



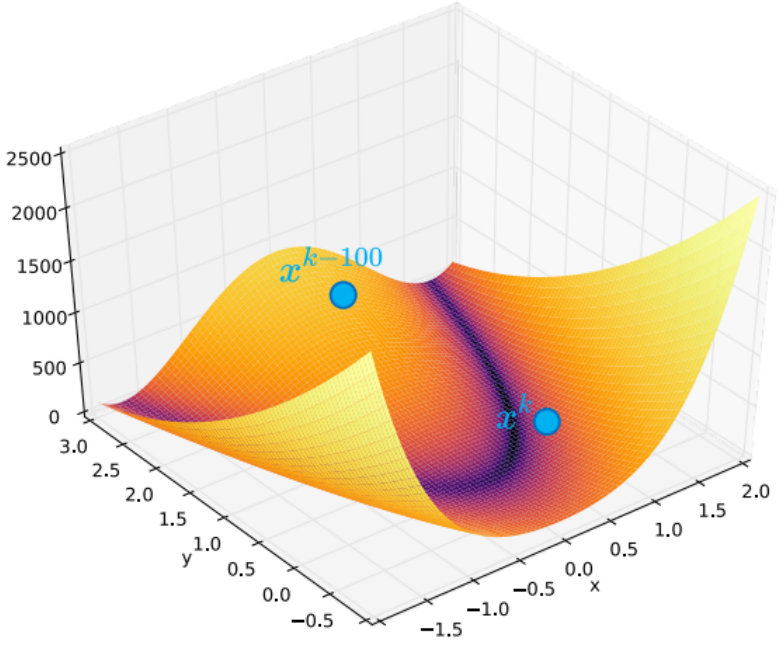
为了让 Weak Wolfe Condition 可以适用于非光滑函数的优化，需要引入 Lewis & Overton Line Search：通过 Armijo Condition 和 Curvature Condition 来维护取值区间的 Upper Bound 和 Lower Bound，经过几次迭代，得到满足 Weak Wolfe Condition 的迭代点：

```
l = 0
u = Inf
alpha = 1

while true, do
    if Armijo Condition fails:
        u = alpha
    else if Curvature Condition fails:
        l = alpha
    else
        return alpha
    if u < Inf
        alpha = (l + u)/2
    else
        alpha = 2*l
end while
```

> 注：初值 x_0 处需要可导，不然搜索无法进行。

1.3 Limited-memory BFGS

在迭代过程中，并非所有历史信息都是有效的。对于图中的 x^{k-100} 和 x^k，二者所处位置的曲率信息完全相反，x^{k-100} 提供的信息并没有什么必要，甚至会让计算出的 B 矩阵变得稠密，因此可以设置一个 Sliding Window，不让过老的信息参与拟合，即 Limited-memory BFGS。



Sliding Window 的尺寸 m 决定了 LBFGS 算法的复杂度，BFGS 的时间复杂度是 $O(n^2)$，LBFGS 则为 $O(mn)$，根据优化对象的不同，m 可以动态调整。

1.4 核心代码

LBFGS 的核心部分在于 Lewis & Overton Line Search:

```C++
int count = 0;
double upperBound = stpmax;
double lowerBound = stpmin;
double funcValInit = f;

while (true)
{
    x = xp + stp * s;
    f = cd.proc_evaluate(cd.instance, x, g);
    count = count + 1;

    /* Check if Armijo Condition fails */
    if (funcValInit - f < -param.f_dec_coeff * stp * gp.dot(s))
    {
        upperBound = stp;
    }

    /* Check if Curvature Condition fails */
    else if (g.dot(s) < param.s_curv_coeff * gp.dot(s))
    {
        lowerBound = stp;
    } else {
        return 999;
    }

    /* Update step size*/
    if (upperBound < stpmax)
    {
        stp = (upperBound + lowerBound) / 2;
    } else {
        stp = 2 * stp;
    }
}
```

2 路径平滑问题

路径平滑 Pipeline：RRT* 规划粗糙轨迹 → Cubic Spline 拟合光滑轨迹 → 建立代价函数并用 L-BFGS 方法求解能量最低的无碰撞光滑轨迹。

2.1 三次样条曲线 Cubic Spline

2.1.1 二维的情况

三次样条曲线是用三次多项式拟合一些列散点（$n+1$ 个散点）：

$$\{(x_0,y_0),(x_1,y_1),\cdots,(x_n,y_n)\}$$

每两个连续的散点之间都用三次多项式连接，并且这些拼接在一起的三次多项式还满足一定连续性约束，构成了一条光滑的轨迹曲线。曲线方程可以表达为如下形式（$n+1$ 个散点对应有 n 条曲线）：

$$\begin{gathered}p_i(x)=a_i+b_i(x-x_i)+c_i(x-x_i)^2+d_i(x-x_i)^3\\ i\in[0,n-1]\end{gathered}$$

上面的 n 条曲线方程带来了 $4n$ 个变量 $\rightarrow(a_i,b_i,c_i,d_i)$，为了的解出这 $4n$ 个变量，需要引入 $4n$ 个约束：

约束 1: 曲线需要依次穿过所有散点，共提供 $2n$ 个约束：

$$\begin{gathered}p_i(x_i)=y_i\\ p_i(x_{i+1})=y_{i+1}\end{gathered},\ \ i\in[0,n-1]$$

约束 2: 为了让曲线尽可能平滑，需要令曲线的一阶导和二阶导也连续，可以理解为第 i 条曲线末端处的导数和第 $i+1$ 条曲线起点的导数相等。由于第 0 条曲线的起点处不存在倒数连续条件，所以共提供 $2n-2$ 个约束：

$$\begin{gathered}p_i'(x_{i+1})=p_{i+1}'(x_{i+1})\\ p_i''(x_{i+1})=p_{i+1}''(x_{i+1})\end{gathered},\ \ i\in[0,n-2]$$

约束 3: 由于 **约束1** 与 **约束2** 共提供 $4n-2$ 条约束，无法求解 $4n$ 个变量，因此还需要补充两条约束。对于这补充的两条约束，常用的有自然边界（Natrual Condition）、非扭结边界（Not-a-knot Condition）和夹持边界（Clamped Condition）。在无其他附加条件时，一般选用非扭结边界条件，即分别令前两段曲线方程和后两段曲线方程等价，由于约束 1 和约束 2 的存在，非扭结条件可以进一步表示为函数的三阶导连续：

$$\begin{gathered}p_0(x)=p_1(x)\\ p_{n-2}x=p_{n-1}(x)\end{gathered}\rightarrow\begin{gathered}p_0'''(x_1)=p_1'''(x_1)\\ p_{n-2}'''x_{n-1}=p_{n-1}'''(x_{n-1})\end{gathered}$$

为了简化计算，令已知量 $h_i=x_{i+1}-x_i$，$\eta_i=y_{i+1}-y_i$，$i\in[0,n-1]$，此时可以将约束化简：

$$\begin{aligned}&y_i=a_i\\&\eta_i=b_ih_i+c_ih_i^2+d_ih_i^3\end{aligned}\qquad,i\in[0,n-1]$$

$$\begin{aligned}&b_{i+1}=3d_ih_i^2+2c_ih_i+b_i\\&2c_{i+1}=6d_ih_i+2c_i\end{aligned}\qquad,i\in[0,n-2]$$

$$\begin{aligned}&d_0=d_1\\&d_{n-2}=d_{n-1}\end{aligned}$$

则由二阶导连续的约束可推导出 d_i 关于 c_i 的表达式：

$$d_i = \frac{1}{3h_i}(c_{i+1} - c_i), \quad i \in [0, n-2]$$

将 d_i 代入原函数连续的约束，可推导出 b_i 关于 c_i 的表达式：

$$b_i = \frac{\eta_i}{h_i} - \frac{1}{3}h_i(c_{i+1} + 2c_i)$$

再将 b_i 和 d_i 代入到一阶导连续的约束，可建立 c_i 和已知量之间的关系：

$$\frac{1}{3}h_i c_i + \frac{2}{3}(h_i + h_{i+1})c_{i+1} + \frac{1}{3}h_{i+1}c_{i+2} = \frac{\eta_{i+1}}{h_{i+1}} - \frac{\eta_i}{h_i}$$

将非扭结条件化简，则有：

$$h_1 c_0 - (h_0 + h_1)c_1 + h_0 c_2 = 0$$
$$h_{n-1}c_{n-2} - (h_{n-2} + h_{n-1})c_{n-1} + h_{n-2}c_n = 0$$

写成矩阵形式，则有：

$$\begin{bmatrix} h_1 & -(h_0+h_1) & h_0 & \cdot & 0 & 0 & 0 \\ \frac{1}{3}h_0 & \frac{2}{3}(h_0+h_1) & \frac{1}{3}h_1 & \cdot & 0 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & 0 & 0 & \cdot & \frac{1}{3}h_{n-2} & \frac{2}{3}(h_{n-2}+h_{n-1}) & \frac{1}{3}h_{n-1} \\ 0 & 0 & 0 & \cdot & h_{n-1} & -(h_{n-2}+h_{n-1}) & h_{n-2} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \cdot \\ c_{n-1} \\ c_n \end{bmatrix}$$
$$= \begin{bmatrix} 0 \\ \frac{\eta_1}{h_1} - \frac{\eta_0}{h_0} \\ \cdot \\ \frac{\eta_{n-1}}{h_{n-1}} - \frac{\eta_{n-2}}{h_{n-2}} \\ 0 \end{bmatrix}$$

通过求解线性方程组 $\mathbf{A}_{n+1\times n+1}\mathbf{x}_{n+1\times 1} = \mathbf{b}_{n+1\times 1}$，可得到 $[b_0, b_1, \cdots, b_{n-1}, b_n]^T$，通过前面推导的对应关系，可得到其他系数。

2.1.2 二维样条的代码实现

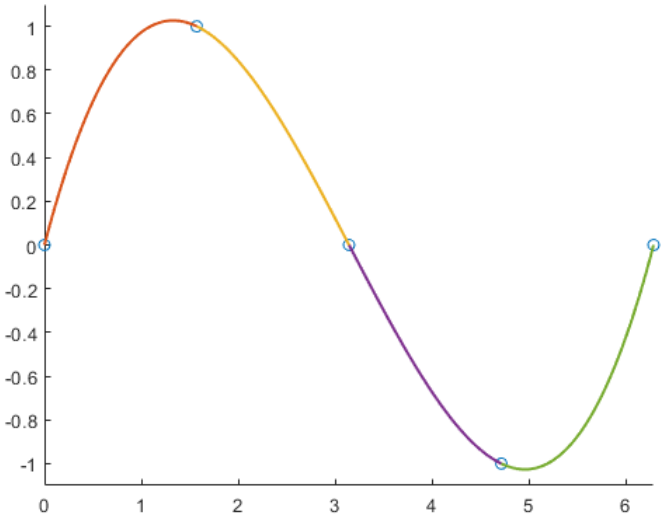
用 MATLAB 验证一下：

```
%% 生成离散点
x = linspace(0, 2*pi, 5);
y = sin(x);

%% 计算样条曲线所需常量
```

```
n = length(x) - 1;                  % n + 1个散点带来n段曲线
h = x(2 : end) - x(1 : end - 1);    %   h = x_{i+1} - x_{i}
eta = y(2 : end) - y(1 : end - 1); % eta = y_{i+1} - y_{i}

%% 建立矩阵
A = zeros(n + 1, n + 1);
b = zeros(n + 1, 1);

% 非扭结条件
A(1, 1) = h(2);
A(1, 2) = -(h(1) + h(2));
A(1, 3) = h(1);

A(n + 1, n - 1) = h(n);
A(n + 1, n) = -(h(n - 1) + h(n));
A(n + 1, n + 1) = h(n - 1);

% 按规律建立带状矩阵 Ax = b
idx = 1;
for i = 2 : n
    A(i, idx) = (1/3) * h(i - 1);
    A(i, idx + 1) = (2/3) * (h(i - 1) + h(i));
    A(i, idx + 2) = (1/3) * h(i);
    b(i, 1) = (eta(i) / h(i)) - eta(i - 1)/h(i - 1);
    disp(i);
    idx = idx + 1;
end

% 求解系数c
c = A\b;

a = zeros(n, 1);
b = zeros(n, 1);
d = zeros(n, 1);

% 求解其余系数并绘图
figure(1);
axis([0, 2*pi, -1.1, 1.1]);
hold on;
scatter(x, y);

for i = 1:n
    a(i) = y(i);
    b(i) = eta(i) / h(i) - (1/3) * h(i) * (c(i + 1) + 2 * c(i));
    d(i) = 1 / (3 * h(i)) * (c(i + 1) - c(i));
```

```
    x_interp = linspace(x(i), x(i+1), 100);
    y_interp = d(i) .* (x_interp - x(i)).^3 + ...
               c(i) .* (x_interp - x(i)).^2 + ...
               b(i) .* (x_interp - x(i)) + ...
               a(i);
    plot(x_interp, y_interp, 'LineWidth', 1.5);
end
```

代码结果非常正确，正是正弦函数该有的样子：



2.1.3 三维样条曲线

换个角度，从三维的视角再看待样条曲线，可以理解为分别拟合 $x-y$ 和 $x-z$，将拟合出来的 x,y,z 组合在一起，即是最终的拟合曲线。

若 $h_i=x_{i+1}-x_i=1$，并且设 $\mathbf{x}_i=[y_i,z_i]^T$ 则利用一阶导连续的约束以及曲线经过所有散点的约束，可以计算出各个参数的表达式：

$$\begin{cases} p_i(0)=[y_i,z_i]^T=a_i \\ p_i(1)=[y_{i+1},z_{i+1}]^T=b_i+2c_i+3d_i \\ Y_i'(0)=D_i=b_i \\ Y_i'(1)=D_{i+1}=b_i+2c_i+3d_i \end{cases} \rightarrow \begin{cases} a_i=y_i \\ b_i=D_i \\ c_i=3(\mathbf{x}_{i+1}-\mathbf{x}_i)-2D_i-D_{i+1} \\ d_i=2(\mathbf{x}_i-\mathbf{x}_{i+1})+D_i+D_{i+1} \end{cases}$$

将这些参数代回到二阶导连续的约束中化简，则有：

$$D_{i-1}+4D_i+D_{i+1}=3(y_{i+1}-y_{i-1})$$

写成矩阵 $D=A^{-1}B$ 的形式：(式中的 x 其实表示 $[y,z]^T$，应当理解为向量)

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

考虑自然边界条件：$D_0 = D_n = 0$

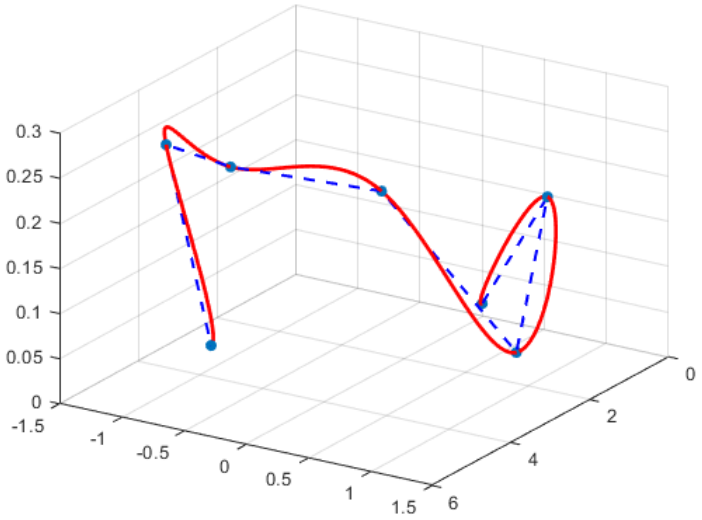
2.1.4 三位样条曲线的代码实现

```
%% 创建三维散点
x = 0: 1: 2*pi;
y = sin(x);
z = ones(length(x), 1);

for i = 1:length(z)
    z(i) = (rand^2)/2;
end
%% 计算样条曲线所需常量
n = length(x) - 1;
A = zeros((n - 1), n - 1);
B = zeros((n - 1), 2);

idx = 1;
for i = 2 : 1: n - 2
    B(i, 1) = 3 * (y(i + 2) - y(i));
    B(i, 2) = 3 * (z(i + 2) - z(i));

    A(i, idx) = 1;
    A(i, idx + 1) = 4;
    A(i, idx + 2) = 1;
    idx = idx + 1;
end

%% 补充矩阵的第一行和最后一行
A(1, 1) = 4;
A(1, 2) = 1;
A(n - 1, n - 2) = 1;
A(n - 1, n - 1) = 4;

B(1, 1) = 3 * (y(3) - y(1));
```

```
B(1, 2) = 3 * (z(3) - z(1));

B(n - 1, 1) = 3 * (y(n + 1) - y(n - 1));
B(n - 1, 2) = 3 * (z(n + 1) - z(n - 1));

D = A\B;

%% 添加自然边界
D = [0, 0;
     D   ;
     0, 0];

%% 还原曲线
a = zeros(n, 2);
b = zeros(n, 2);
c = zeros(n, 2);
d = zeros(n, 2);

figure;
view(3);
hold on;
grid on;
scatter3(x, y, z, 'filled');
plot3(x, y, z, 'b--', 'LineWidth', 1.5);

for i = 1:n
    a(i, 1) = y(i);
    a(i, 2) = z(i);

    b(i, 1) = D(i, 1);
    b(i, 2) = D(i, 2);

    c(i, 1) = 3*(y(i+1) - y(i)) - 2*D(i, 1) - D(i+1, 1);
    c(i, 2) = 3*(z(i+1) - z(i)) - 2*D(i, 2) - D(i+1, 2);

    d(i, 1) = 2*(y(i) - y(i+1)) + D(i, 1) + D(i+1, 1);
    d(i, 2) = 2*(z(i) - z(i+1)) + D(i, 2) + D(i+1, 2);

    interpX = linspace(x(i), x(i+1), 100);
    interpY = a(i, 1) +...
              b(i, 1).*(interpX - x(i)) + ...
              c(i, 1).*(interpX - x(i)).^2 + ...
              d(i, 1).*(interpX - x(i)).^3;

    interpZ = a(i, 2) +...
```

```
                b(i, 2).*(interpX - x(i)) + ...
                c(i, 2).*(interpX - x(i)).^2 + ...
                d(i, 2).*(interpX - x(i)).^3;

    plot3(interpX, interpY, interpZ, 'r', 'LineWidth', 2);
end
```

三维曲线成功拟合了所有散点，值得庆祝！



2.2 路径平滑

RRT* 或 Dikstra、JPS 等栅格地图路径规划算法生成的路径是由折线段构成的非光滑路径，这会让机器人的轨迹跟踪性能大打折扣，所以需要建立代价函数，用无约束优化方法优化三次样条曲线拟合后的粗糙路径，生成又安全又光滑的最终路径 ：

$$\min \alpha \cdot \text{Energy}(x_1, x_2, \cdots, x_n) + \beta \cdot \text{Potential}(x_1, x_1, \cdots, x_n)$$

2.2.1 Minimum Stretch Energy

将三次样条曲线的表达式定义为：

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

式中的 s 为 $(x - x_i)$，规定 $s \in [0, 1]$ 是为了简化计算，令拟合前的离散点横坐标插值为 1。可以理解为 `x = 0 : 1 : xMax`

定义能量函数：

$$\text{Energy}(x_0, x_1, \cdots, x_n) = \sum_{i=0}^{n} \int_0^1 ||p_i''(s)||^2 \, \mathrm{d}s$$

为了利用 BFGS 公式计算迭代步长，需要手动推导一下能量函数的梯度。

定义 $f_i = \int_0^1 ||p_i''(s)||^2 \, \mathrm{d}s$，则有：

$$\sum_{i=0}^{n-1} f_i = \sum_{i=0}^{n-1} 12d_i^T d_i + 12c_i^T d_i^T + 4c_i^T c_i$$

相应的，f_i 对 x 的偏导为：

$$\frac{\mathrm{d}f_i}{\mathrm{d}x} = 24(\frac{\mathrm{d}d_i}{\mathrm{d}x})^T d_i + 12(\frac{\mathrm{d}c_i}{\mathrm{d}x})^T d_i + 12(\frac{\mathrm{d}d_i}{\mathrm{d}x})^T c_i + 8(\frac{\mathrm{d}c_i}{\mathrm{d}x})^T c_i$$

式中的两个未知量分别为 $\frac{\mathrm{d}d_i}{\mathrm{d}x}$ 和 $\frac{\mathrm{d}c_i}{\mathrm{d}x}$。

重温一下 2.1.3 节的结论：

$$\begin{aligned}
a_i &= y_i \\
b_i &= D_i \\
c_i &= 3(\mathbf{x}_{i+1} - \mathbf{x}_i) - 2D_i - D_{i+1} \\
d_i &= 2(\mathbf{x}_i - \mathbf{x}_{i+1}) + D_i + D_{i+1} \\
D &= A^{-1}B, \;\; D_0 = D_n = 0
\end{aligned}$$

根据链式求导法则，

$$\begin{aligned}
\frac{\mathrm{d}d_i}{\mathrm{d}x} &= 2\frac{\mathrm{d}(x_{i+1} - x_i)}{\mathrm{d}x} + \frac{\mathrm{d}D_i}{\mathrm{d}x} + \frac{\mathrm{d}D_{i+1}}{\mathrm{d}x} \\
\frac{\mathrm{d}c_i}{\mathrm{d}x} &= -3\frac{\mathrm{d}(x_{i+1} - x_i)}{\mathrm{d}x} - 2\frac{\mathrm{d}D_i}{\mathrm{d}x} - \frac{\mathrm{d}D_{i+1}}{\mathrm{d}x}
\end{aligned}$$

式中：

$$\frac{\mathrm{d}}{\mathrm{d}x}\begin{bmatrix} x_0 - x_1 \\ \cdots \\ x_{i-1} - x_i \\ \cdots \\ x_{n-1} - x_n \end{bmatrix}_{n\times 1} = \begin{bmatrix} -1 & & & & \\ 1 & -1 & & & \\ & 1 & -1 & & \\ & & \ddots & \ddots & \\ & & & 1 & -1 \\ & & & & 1 \end{bmatrix}_{n\times(n-1)}$$

由于 $D = A^{-1}B$，且 A 为常数矩阵，所以有：

$$\begin{aligned}
\frac{\mathrm{d}D}{\mathrm{d}x} &= A^{-1}\frac{\mathrm{d}B}{\mathrm{d}x} \\
\frac{\mathrm{d}B}{\mathrm{d}x} &= \begin{bmatrix} 0 & 3 & & & \\ -3 & 0 & 3 & & \\ & \ddots & \ddots & \ddots & \\ & & -3 & 0 & 3 \\ & & & -3 & 0 \end{bmatrix}_{(n-1)\times(n-1)}
\end{aligned}$$

> 在具体实现时，需要特别注意矩阵的维度，并且 $\frac{\mathrm{d}D_0}{\mathrm{d}x} = \frac{\mathrm{d}D_N}{\mathrm{d}x} = 0$，并不涉及特殊的编程技巧，与三次样条曲线的 MATLAB 实现非常相似。

2.2.2 Minimum Potential

势场函数相较于能量函数，其结构就简单多了：

$$\mathrm{Potential}(x_1, x_2, \cdots, x_{N-1}) = 1000 \sum_{i=1}^{j=1} \sum_{j=1}^{M} r_j - \sqrt{(x_i - a_{j)^2+}(y_i - b_j)^2}$$

他的导数形式也很简洁明了，并不涉及矩阵运算

$$\frac{\mathrm{d}P}{\mathrm{d}x_i} = -1000 \sum_{j=1}^{M} \frac{x_i - a_j}{\sqrt{(x_i - a_j)^2 + (y_i - b_j)^2}}$$
$$\frac{\mathrm{d}P}{\mathrm{d}y_i} = -1000 \sum_{j=1}^{M} \frac{y_i - a_j}{\sqrt{(x_i - a_j)^2 + (y_i - b_j)^2}}$$

2.4 仿真

在完成作业的过程中遇到的最大问题并不在于数学，而在于对 C++ 的不熟悉，很多函数并不知道该如何使用，尤其和指针相关的操作更是一窍不通，因此 `path_smoother.hpp` 部分的代码参考了其他同学在 Github 上开源的作业。

运行节点，可以得到符合要求的平滑轨迹

```shell
catkin_make
source ./devel/setup.bash
roslaunch gcopter curve_gen.launch
```