几何建模与处理基础
GAMES102
作业 1：曲线拟合

Qingjun Chang

2020 年 10 月 17 日

1 目标

- 实现并分析四种拟合算法 [1]
- 尝试学习使用无境框架

2 拟合算法描述

问题：对于给定 $\mathbb{R}^2$ 域内一组点 $\{\mathbf{P}_i=(x_i,y_i)\}_{i=0}^n$，找到一个拟合这组点的函数 $f(x)$。

2.1 多项式插值

设 $f(x)$ 是多项式函数，即以幂函数为基的线性组合，平面内 $n+1$ 个互异数据点 $(x_i \neq x_j)$ 可以唯一确定 n 次多项式，设 $f(x)=\sum_{i=0}^n \alpha_i B_i(x)$，其中基函数 $B_i(x)=x^i$。将 $n+1$ 个数据点代入：

$$\begin{cases} y_0 &= \alpha_0+\alpha_1 x_0+\alpha_2 x_0^2+\cdots+\alpha_n x_0^n \\ y_1 &= \alpha_0+\alpha_1 x_1+\alpha_2 x_1^2+\cdots+\alpha_n x_1^n \\ & \vdots \\ y_n &= \alpha_0+\alpha_1 x_n+\alpha_2 x_n^2+\cdots+\alpha_n x_n^n \end{cases}$$

令

$$\mathbf{Y}=\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{B}=\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{bmatrix}, \quad \boldsymbol{\alpha}=\begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix}$$

即：通过求解关于 $\boldsymbol{\alpha}$ 的线性方程组 $\mathbf{Y}=\mathbf{B}\boldsymbol{\alpha}$ 即可得到插值这组点的 n 次多项式曲线

$$f(x)=\alpha_0+\alpha_1 x+\alpha_2 x^2+\cdots+\alpha_n x^n$$

2.2 高斯基函数插值

设 $f(x) = a + \sum_{i=0}^{n} b_i g_i(x)$，其中

$$g_i(x) = e^{-(x-x_i)^2/(2\sigma^2)}$$

将 $n+1$ 个数据点代入：

$$\begin{cases} y_0 &= a + b_0g_0(x_0) + b_1g_1(x_0) + \cdots + b_ng_n(x_0) \\ y_1 &= a + b_0g_0(x_1) + b_1g_1(x_1) + \cdots + b_ng_n(x_1) \\ & \vdots \\ y_n &= a + b_0g_0(x_n) + b_1g_1(x_n) + \cdots + b_ng_n(x_n) \end{cases} \tag{1}$$

可以发现公式 (1) 中，未知数个数多余方程个数，方程有多个解，为此可以添加约束条件。为了简单，在本实验中，我们添加的约束条件为插值最后两个点的中点，即

$$f(\frac{x_n + x_{n-1}}{2}) = \frac{y_n + y_{n-1}}{2}$$

代入 (1) 式有：

$$\begin{cases} y_0 &= a + b_0g_0(x_0) + b_1g_1(x_0) + \cdots + b_ng_n(x_0) \\ y_1 &= a + b_0g_0(x_1) + b_1g_1(x_1) + \cdots + b_ng_n(x_1) \\ & \vdots \\ y_n &= a + b_0g_0(x_n) + b_1g_1(x_n) + \cdots + b_ng_n(x_n) \\ \frac{y_n+y_{n-1}}{2} &= a + b_0g_0(\frac{x_n+x_{n-1}}{2}) + b_1g_1(\frac{x_n+x_{n-1}}{2}) + \cdots + b_ng_n(\frac{x_n+x_{n-1}}{2}) \end{cases}$$

令

$$\mathbf{Y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \\ \frac{y_n+y_{n-1}}{2} \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} 1 & g_0(x_0) & g_1(x_0) & \ldots & g_n(x_0) \\ 1 & g_0(x_1) & g_1(x_1) & \ldots & g_n(x_1) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & g_0(x_n) & g_1(x_n) & \ldots & g_n(x_n) \\ 1 & g_0(\frac{x_n+x_{n-1}}{2}) & g_1(\frac{x_n+x_{n-1}}{2}) & \ldots & g_n(\frac{x_n+x_{n-1}}{2}) \end{bmatrix}, \quad \boldsymbol{b} = \begin{bmatrix} a \\ b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}$$

即：通过求解关于 $\boldsymbol{\alpha}$ 的线性方程组 $\mathbf{Y} = \mathbf{G}\boldsymbol{b}$ 即可得到插值这组点的径向基函数插值函数

$$f(x) = a + \sum_{i=0}^{n} b_i g_i(x)$$

注： 在本例中，由于 x_i 的值较大，故我们取 $\sigma = 50$

2.3 最小二乘多项式拟合

注意到在第 2.1节中多项式插值时，当数据点个数较多时，插值会导致多项式曲线阶数过高，带来不稳定因素。可通过固定幂基函数的最高次数 $m(m < n)$。设 $f(x) = \sum_{i=0}^{m} \alpha_i B_i(x)$，其中基函数 $B_i(x) = x^i$。将 $n+1$ 个数据点代入：

$$\begin{cases} y_0 &= \alpha_0 + \alpha_1x_0 + \alpha_2x_0^2 + \cdots + \alpha_mx_0^m \\ y_1 &= \alpha_0 + \alpha_1x_1 + \alpha_2x_1^2 + \cdots + \alpha_mx_1^m \\ & \vdots \\ y_n &= \alpha_0 + \alpha_1x_n + \alpha_2x_n^2 + \cdots + \alpha_mx_n^m \end{cases} \tag{2}$$

注意到在在公式 (2) 中，方程的个数多余未知数个数，导致无精确解。使用最小二乘法可以使得函数曲线尽可能靠近数据点，即

$$\min_{\boldsymbol{\alpha}} \sum_{i=0}^{n} (y_i - \sum_{i=0}^{m} \alpha_i x^i)^2, \quad \boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_m]^{\mathrm{T}} \tag{3}$$

将 (2) 式改写为矩阵形式，令

$$\mathbf{Y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}$$

即：$\mathbf{Y} = \mathbf{B}\boldsymbol{\alpha}$，由于 $\mathbf{B}$ 是一个非方阵，且列满秩，方程无精确解。采用最小二乘的方式，对 (3) 式求偏导，可得到：

$$\mathbf{B}^{\mathrm{T}}\mathbf{Y} = \mathbf{B}^{\mathrm{T}}\mathbf{B}\boldsymbol{\alpha}$$

通过求解关于 $\boldsymbol{\alpha}$ 的线性方程组 $\mathbf{B}^{\mathrm{T}}\mathbf{Y} = \mathbf{B}^{\mathrm{T}}\mathbf{B}\boldsymbol{\alpha}$ 即可得到最小二乘拟合这组点的 m 次多项式曲线

$$f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_m x^m$$

2.4 岭回归拟合

在第 2.3节中，当 $\mathbf{B}^{\mathrm{T}}\mathbf{B}$ 接近于奇异时，$(\mathbf{B}^{\mathrm{T}}\mathbf{B})^{-1}$ 有较大误差，拟合结果不稳定。为此在最小二乘的误差函数中添加正则项，即：[2]

$$\min_{\boldsymbol{\alpha}} (\sum_{i=0}^{n} (y_i - \sum_{i=0}^{m} \alpha_i x^i)^2 + \lambda \sum_{i=0}^{m} \alpha^2, \quad \boldsymbol{\alpha} = [\alpha_0, \alpha_1, \alpha_m]^{\mathrm{T}} \tag{4}$$

通过对 (4) 式求偏导，并令其等于零，令

$$\mathbf{Y} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^m \\ 1 & x_1 & x_1^2 & \dots & x_1^m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^m \end{bmatrix}$$

可得到：

$$2\mathbf{B}^{\mathrm{T}}\mathbf{Y} - 2\mathbf{B}^{\mathrm{T}}\mathbf{B}\boldsymbol{\alpha} - 2\lambda\boldsymbol{\alpha} = 0$$

即：

$$\boldsymbol{\alpha} = (\mathbf{B}^{\mathrm{T}}\mathbf{B} + \lambda\mathbf{I})^{-1}\mathbf{B}^{\mathrm{T}}\mathbf{Y}$$

同样可得到最小二乘拟合这组点的 m 次多项式曲线

$$f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_m x^m$$

3 实现

3.1 Interpolation_PolynomialBaseFunction.cpp

```
Eigen::VectorXf Interpolation_PolynomialBaseFunction(
 std::vector<Ubpa::pointf2> points) {
  int n = points.size();

  Eigen::MatrixXf normal_equation=Eigen::MatrixXf::Zero(n,n);
  Eigen::VectorXf y = Eigen::VectorXf::Zero(n);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < n; ++j)
      normal_equation(i, j) = pow((points[i][0]), j);
    y(i) = points[i][1];
  }
  return normal_equation.inverse() * y;
}
```

3.2 Interpolation_GaussBaseFunction.cpp

```
float GaussBaseFunction(float x, float xi, float sigma) {
  return expf(-(x - xi) * (x - xi) / (2 * sigma * sigma));
}

Eigen::VectorXf Interpolation_GaussBaseFunction(std::vector
 <Ubpa::pointf2> points,float sigma = 1) {
  int n = points.size();
  Eigen::MatrixXf normal_equation = Eigen::MatrixXf::Zero(
    n + 1, n + 1);
  Eigen::VectorXf y = Eigen::VectorXf::Zero(n + 1);

  for (int i = 0; i < n; ++i) {
    normal_equation(i, 0) = 1;
    for (int j = 1; j < n + 1; ++j)
      normal_equation(i, j) = GaussBaseFunction(points[i][0]
       ,points[j-1][0], sigma);
    y(i) = points[i][1];
  }
  normal_equation(n, 0) = 1;
  for (int j = 1; j < n + 1; ++j)
    normal_equation(n, j) = GaussBaseFunction((points[n-1]
     [0] + points[n-2][0])/2.0f, points[j-1][0], sigma);
  y(n) = (points[n - 1][1] + points[n - 2][1]) / 2.0f;
  return normal_equation.inverse() * y;
}
```

3.3 Approximation_LeastSquare.cpp

```
Eigen::VectorXf Approximation_LeastSquare(std::vector<Ubpa::
 pointf2> points, int order = 3) {
  int n = points.size();

  Eigen::MatrixXf normal_equation = Eigen::MatrixXf::Zero(n,
   order + 1);
  Eigen::VectorXf y = Eigen::VectorXf::Zero(n);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j <= order; ++j)
      normal_equation(i, j) = pow((points[i][0]), j);
    y(i) = points[i][1];
  }

  return (normal_equation.transpose()*normal_equation)
   .inverse()*(normal_equation.transpose()*y);
}
```

3.4 Approximation_RidgeRegression.cpp

```
Eigen::VectorXf Approximation_RidgeRegression(std::vector<
 Ubpa::pointf2> points, int order=3, float lambda=0.5) {
  int n = points.size();
  Eigen::MatrixXf normal_equation = Eigen::MatrixXf::Zero(n,
   order + 1);
  Eigen::VectorXf y = Eigen::VectorXf::Zero(n);

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j <= order; ++j)
      normal_equation(i, j) = pow((points[i][0]), j);
    y(i) = points[i][1];
  }
  Eigen::MatrixXf I;
  I.setIdentity(order + 1, order + 1);

  return (normal_equation.transpose()*normal_equation + I*
   lambda).inverse()*(normal_equation.transpose()*y);
}
```

4 数值示例

- 在使用高斯基函数插值时，实验中使用 $\sigma = 50$
- 在逼近拟合中，多项式最高次数设置为 4
- 在岭回归中，$\lambda = 0.4$

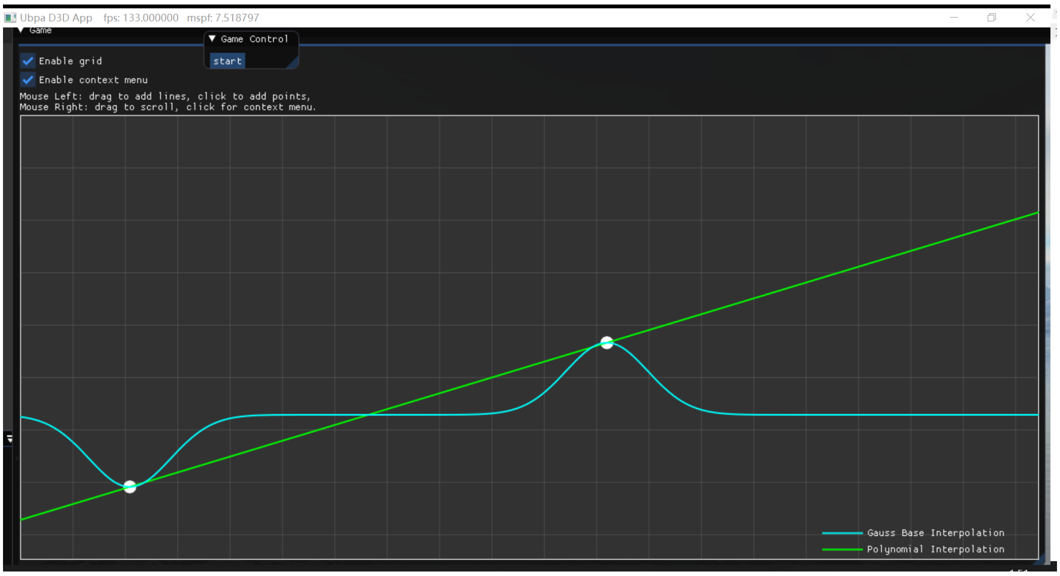

图 1: 分别用一次多项式曲线 (绿色) 和高斯函数曲线 (淡蓝色) 插值平面中的两点.

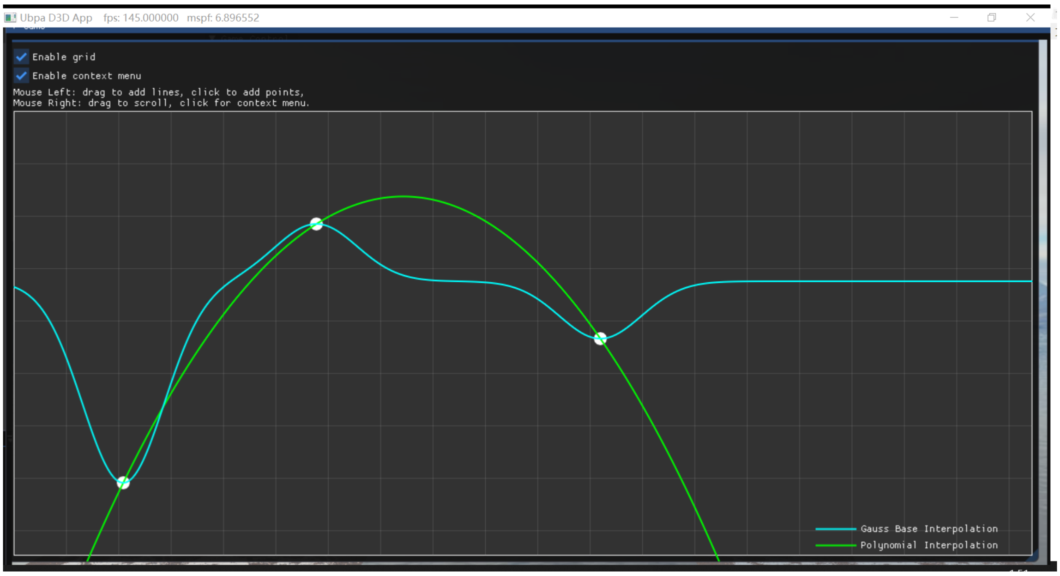

图 2: 分别用抛物线 (绿色) 和高斯函数曲线 (淡蓝色) 插值平面中的三点.

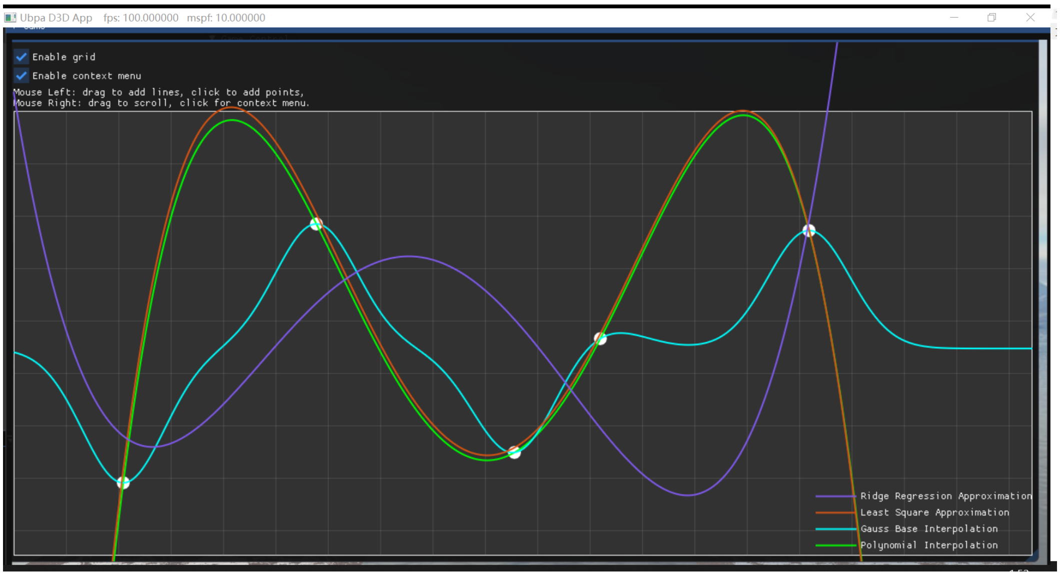

图 3: 分别用 4 次多项式 (绿色) 和高斯函数曲线 (淡蓝色) 插值平面中的 5 个点；分别用 4 次多项式最小二乘法 (红色) 和岭回归法 (紫色) 拟合平面中的 5 个点.

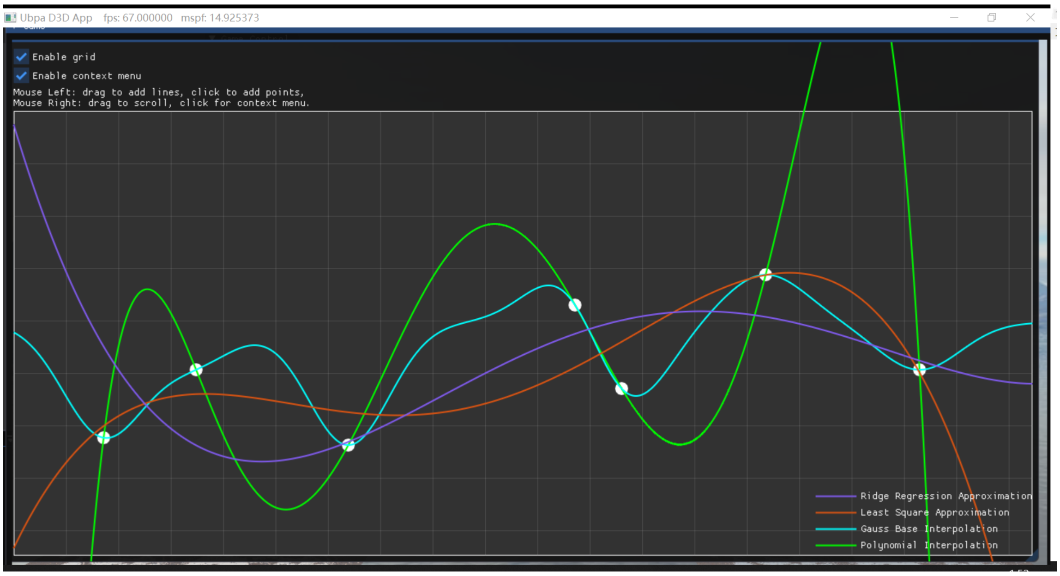

图 4: 分别用 6 次多项式 (绿色) 和高斯函数曲线 (淡蓝色) 插值平面中的 7 个点；分别用 4 次多项式最小二乘法 (红色) 和岭回归法 (紫色) 拟合平面中的 7 个点.

参考文献

[1] 码农家园. 径向基函数插值. [EB/OL]. https://www.codenong.com/cs107017441/ 2020-06-29.

[2] zhiyong_will. 简单易学的机器学习算法——岭回归 (ridge regression). [EB/OL]. https://blog.csdn.net/google19890102/article/details/27228279/ 2014-05-27.