- 插值：求过一直有限个数据点的近似函数
- 拟合：不要求过已知数据点

插值

拉格朗日多项式插值

根据区间内$n+1$个点求n次多项式：

$ \phi(x) = a_0 + a_1 x + \dots + a_n x^n $

求解

令：$l_i(x) = \dfrac {(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)} {(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)}$

则$l_i(x)$满足

$ l_i(x_j) = \begin{cases} 0, \enspace j\neq i\ 1, \enspace j = i \end{cases} $

拉格朗日插值多项式：

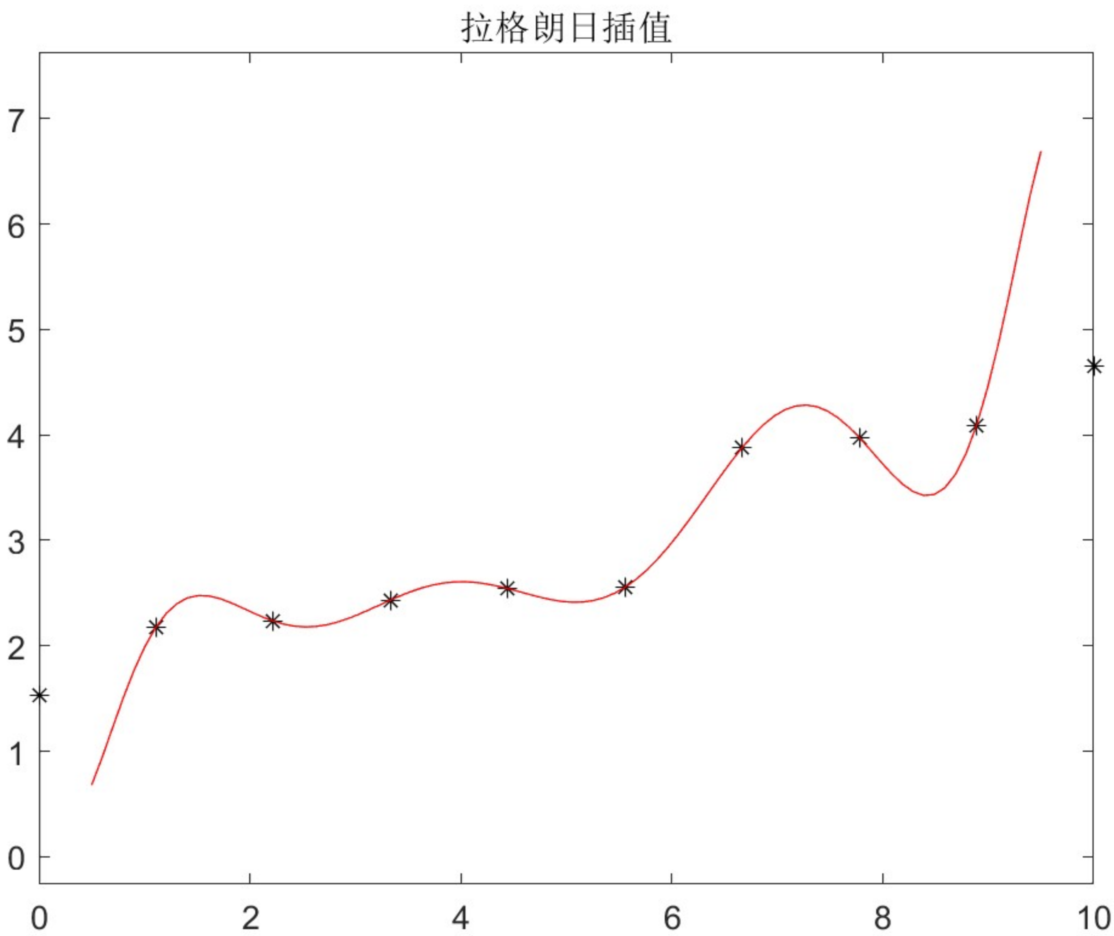
$ L_n(x)= \sum_{i=0}^ny_il_i(x) $

代码实现

```
function y = lagrange(x0, y0, x)
    % x0, y0 是已知节点向量，x为插值点

    m = length(x0);
    a = ones(m);
    y = 0;
    for i=1:m
        for j=1:m
            if i ~= j
                a(i) = a(i)*(x-x0(j))/(x0(i)-x0(j));
            end
        end
        y = a(i)*y0(i) + y;
    end

end
```



牛顿插值

差商

一阶差商：$f[x_i,x_j] = \dfrac {f(x_i) - f(x_j)} {x_i - x_j}$

二阶差商：$f[x_i, x_j, x_k] = dfrac {f[x_1, x_j] - f[x_j, x_k]} {x_0 - x_k}$

Newton插值公式

$ N_n(x) = f(x_0) + (x-x_0)f[x_0, x_1] + \dots + (x-x_0)(x-x_1)\dots(x-x_{n-1})f[x_0, x_1, \dots , x_n] \ N_{n+1}(x) = N_n(x) + (x-x_0)\dots(x-x_n)f[x_0,x_1,\dots, x_{n+1}] $

代码实现

```
function y = newtonInterpolation(x0, y0, x)
    n = length(x0);
    m = length(x);
    A = zeros(n,n); A(:,1) = y0';
    y = zeros(1, n);
    for j=2:n
        for i=j:n
            A(i,j) = (A(i,j-1)- A(i-1,j-1)) / (x0(i)-x0(i-j+1));
```

```
            end
        end
        for t=1:m
            z = x(t);
            s = 0.0; y = 0.0;
            for k = 1:n
                p = 1.0;
                for j = 1:k-1
                    p = p*(z-x0(j));
                end
                s = s+A(k,k)*p;
            end
            y(t) = s;
        end
    end
```

差分

节点等距时，关于节点间差商可以用差分表示： $ \Delta f_k = f_{k+1} - f_{k} $

二阶差分： $ \Delta ^2 f_k = \Delta f_{k+1} - \Delta f_k $

还有向后差分与中心差分： $ \nabla f_k = f_k - f_{k-1} $

$ \delta f_k = f\left(x_k + \frac{h}{2}\right) - f\left(x_k - \frac{h}{2}\right) $

用差分替代差商，有牛顿向前插值公式： $ N_n(x_0 + th) = f_0 + t\Delta f_0 + \cdots + \frac{t(t-1)\cdots(t-n+1)}{n!}\Delta^n f_0 $

分段线性插值

用函数表作插值计算时，分段线性插值精度足够。如数学、物理中的特殊函数表。

matlab自带分段线性插值逻辑：interp1

埃尔米特插值

要求插值函数与原函数有相同的一节、二阶甚至更高阶导数值： $ H(x_i) = y_i, \enspace H'(x_i) = y'_i $

$ H(x) = \sum_{i=0}^n h_i[(x_i-x)(2a_iy_i-y'_i)+y_i] $

```
function y = hermite(x0, y0, y1, x)
    n = length(x0);
    m = length(x);
    for k=1:m
        yy = 0.0;
        for i=1:n
            h = 1.0;
            a = 0.0;
            for j = 1:n
                if j~=i
                    h = h*((x(k)-x0(j))/(x0(i)-x0(j))) ^2;
```

```
                    a = 1/(x0(i)-x0(j)) + a;
                end
            end
        yy = yy + h*((x0(i)-x(k))*(2*a*y0(i)-y1(i))+y0(i));
        end
        y(k) = yy;
    end
```

样条插值

连接点处有连续的曲率。k次样条函数：

- 每个小区间上市k次多项式
- 具有k-1阶连续导数

二次样条插值

二次样条函数： $ s_2(x) = \alpha_0 + \alpha_1 x + \frac{\alpha_2}{2!} x^2 + \sum_{j=1}^{n-1} \frac{\beta_j}{2!} (x - x_j)+^2 \in S_p(\Delta, 2), \ (x-x_j)^2+ = \begin{cases}(x-x_j)^2,\enspace x\geq x_j\0,\enspace x<x_j \end{cases} $

1. 已知插值节点x_i和相应的函数值y_i以及一个端点的导数。

2. 已知插值节点x_i和相应的导数值y'_i，以及端点x_0处的函数值y_0。

令$X = (\alpha _0, \alpha _1, \alpha _2, \beta _1, \dots , \beta _{n-1})^T$，$C = (y_0, y_1, \dots , y_n, y'_0$

$ A = \begin{bmatrix} 1 & x_0 & \dfrac{1}{2}x_0^2 & 0 & \cdots & 0 \ 1 & x_1 & \dfrac{1}{2}x_1^2 & 0 & \cdots & 0 \ 1 & x_2 & \dfrac{1}{2}x_2^2 & \dfrac{1}{2}(x_2 - x_1)^2 & \cdots & 0 \ \vdots & \vdots & \vdots & \vdots & & \vdots \ 1 & x_n & \dfrac{1}{2}x_n^2 & \dfrac{1}{2}(x_n - x_1)^2 & \cdots & \dfrac{1}{2}(x_n - x_{n-1})^2 \ 0 & 1 & x_0 & 0 & \cdots & 0 \end{bmatrix} $

三次样条插值

$ s_3(x) = \alpha_0 + \alpha_1 x + \frac{\alpha_2}{2!} x^2 + \frac{\alpha_3}{3!} x^3 + \sum_{j=1}^{n-1} \frac{\beta_j}{3!} (x - x_j)+^3 \in S_p(\Delta, 3),\(x-x_j)^3+ = \begin{cases}(x-x_j)^3,\enspace x\geq x_j\0,\enspace x<x_j \end{cases}$

B样条函数插值方法

二维插值

拟合

线性最小二乘

线性最小二乘法笔记

线性最小二乘法是解决曲线拟合最常用的方法，基本思路是构建一个函数 $f(x)$，使其在某种准则下与所有数据点最为接近。

函数构建

$$ f(x) = a_1 r_1(x) + a_2 r_2(x) + \cdots + a_m r_m(x) \quad (11) $$

其中 $r_k(x)$ 是事先选定的一组线性无关的函数，a_k 是待定系数 $(k=1,2,\cdots, m, m<n)$。

拟合准则

拟合准则是使 $y_i, i=1,2,\cdots, n$，与 $f(x_i)$ 的距离 δ_i 的平方和最小，称为最小二乘准则。

系数 a_k 的确定

记

$$ J(a_1,\cdots, a_m) = \sum_{i=1}^n \delta_i^2 = \sum_{i=1}^n [f(x_i) - y_i]^2 \quad (12) $$

为求 $a_1,\cdots, a_m$ 使 J 达到最小，只需利用极值的必要条件 $\frac{\partial J}{\partial a_k} = 0 , (k=1,\cdots, m)$，得到关于 $a_1,\cdots, a_m$ 的线性方程组：

$$ \sum_{i=1}^n r_j(x_i) [\sum_{k=1}^m a_k r_k(x_i) - y_i] = 0, \quad (j=1,\cdots, m) $$

即

$$ \sum_{k=1}^m a_k [\sum_{i=1}^n r_j(x_i) r_k(x_i)] = \sum_{i=1}^n r_j(x_i) y_i, \quad (j=1,\cdots, m) \quad (13) $$

记

$$ R = \begin{bmatrix} r_1(x_1) & \cdots & r_m(x_1) \\ \vdots & \vdots & \vdots \\ r_1(x_n) & \cdots & r_m(x_n) \end{bmatrix}_{n \times m}, $$

$$ A = [a_1, \cdots, a_m]^T, \quad Y = (y_1, \cdots, y_n)^T $$

方程组 (13) 可表示为：

$$ R^T R A = R^T Y \quad (14) $$

当 ${r_1(x), \cdots, r_m(x)}$ 线性无关时，R 列满秩，$R^T R$ 可逆，于是方程组 (14) 有唯一解：

$$ A = (R^T R)^{-1} R^T Y $$

code

```
x = [x1, x2, ..., xn];
y = [y1, y2, ..., yn];

m = 2;
R = zeros(length(x), m);
for i = 1:m
```

```
    R(:, i) = x.^(i-1);
end

A = (R' * R) \ (R' * y);
f = @(x) A(1) + A(2) * x + A(3) * x.^2;

% 绘制
xx = linspace(min(x), max(x), 100);
yy = f(xx);

figure;
plot(x, y, 'o', xx, yy, '-');
xlabel('x');
ylabel('y');
title('Linear Least Squares Fit');
legend('Data', 'Fit');
```

最小二乘优化

lsqlin

`lsqlin` 用于求解具有边界或线性约束的线性最小二乘问题。基本用法如下：

```
x = lsqlin(C, d, A, b, Aeq, beq, lb, ub)
```

- C 和 d 定义了线性方程组。
- A 和 b 定义了线性不等式约束 A*x <= b。
- Aeq 和 beq 定义了线性等式约束 Aeq*x = beq。
- lb 和 ub 定义了变量的下界和上界。

lsqcurvefit

`lsqcurvefit` 用于非线性曲线拟合问题

```
x = lsqcurvefit(fun, x0, xdata, ydata)
```

- fun 是待拟合的非线性函数。
- x0 是初始参数估计。
- xdata 和 ydata 是已知的数据点

lsqnonlin

`lsqnonlin` 用于非线性最小二乘问题

```
x = lsqnonlin(fun, x0, lb, ub)
```

- fun 是非线性目标函数。
- x0 是初始参数估计。
- lb 和 ub 是参数的下界和上界。

lsnonneg

`lsnonneg` 用于求解具有非负约束的线性最小二乘问题

```
x = lsnonneg(C, d)
```

- C 和 d 定义了线性方程组。
- 所有解向量 x 的元素都必须非负。