情報数学C

Mathematics for Informatics C

第6回 補間法と回帰分析
(ラグランジュ補間,スプライン補間,最小2乗法)

情報メディア創成学類
藤澤誠

今回の講義内容

- **今日の問題**
- ラグランジュ補間とニュートン補間
- スプライン補間
- 最小2乗法による関数近似

今回の講義で解く問題

$$f(\boldsymbol{x}) = g(x_i, f_i)$$

$$(i = 1,2,\dots,n)$$

今回の講義で解く問題

離散点での関数値
$f(\boldsymbol{x}_i)$ $(i = 1,2,\dots,n)$が分かっている時

任意位置$\boldsymbol{x}$での$f(\boldsymbol{x})$は?

今回の講義で解く問題

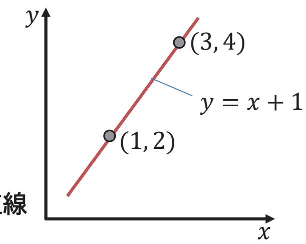
一次関数を使う場合(2点を通る1次関数を求める問題)

例) $(1,2)$と$(3,4)$を通る1次関数

1次関数を$y = ax + b$とすると,

$$a + b = 2$$
$$3a + b = 4$$

となり,これを解くことで
$(a,b) = (1,1)$となり,**2点を通る直線**
$y = x + 1$ が得られる.



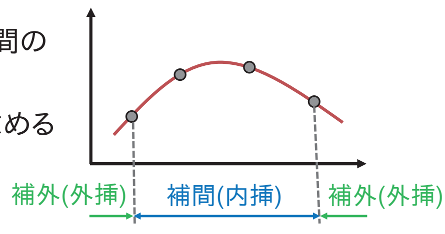
⇒ 直線の式が得られれば$x = 2$で$y = 3$といった**2点の間の値**や,$x = 10$で$y = 11$といった**外部の値**も求められるようになる.

今回の講義で解く問題

補間・補外

離散的なデータの間の値を求める(**内挿**),

もしくは外の値を求める(**外挿**)という問題



離散データ$\boldsymbol{x}_i$に合うような(フィットする)関数$f(\boldsymbol{x})$を求めるということで**関数フィッティング**と呼ぶこともある.

今回の講義で扱う問題

補間・補外は**どんなところで使われる?**

観測されるのは**離散データ**だけど解析するためには**関数・曲線・曲面**が必要. この**ギャップを埋める**ために使われる

設計・デザイン	CAD(Computer Aided Design)ではスプライン補間がデファクトスタンダード(車や飛行機, 新幹線のボディの曲線はすべてスプライン曲線でデザインされている), PCでのIllustratorなどのイラストソフトで曲線を描くために使われる. その他フォントなども.
観測データの解析・未来予測	現在までの離散データに関数をフィッティングすることで, 現在のデータの傾向を求め, 将来予測に使われる(株価予測, 人口変化予測などなど)

離散データから曲線の式を求めてから勾配や曲率を求めたりして, 他の数値計算に用いることもある

今回の講義で解く問題

今回の講義で扱う「補間」についての前提条件

- データ点(サンプリング点)$\boldsymbol{x}_i$とそこでの関数値(観測値)f_iは与えられているとする
- x_iは**スカラーorベクトル**, f_iは今回の講義では**スカラー値**とする(f_iをy_iと表すこともあるので注意)
- データ点として与えられていない**任意位置$\boldsymbol{x}$における関数値(観測値)**を求める, もしくは**観測データにフィットする関数$f(\boldsymbol{x})$**そのものを求める

今回の講義内容

- 今日の問題
- **ラグランジュ補間とニュートン補間**
- スプライン補間
- 最小2乗法による関数近似

線形補間

最も基本的な補間が**線形補間**

例) 2点$(x_0, y_0), (x_1, y_1)$を通る関数$y = f(x)$の$x = c$での関数値$f(c)$を**線形補間**で求める

1次関数を$y = ax + b$とすると,

$$x_0 a + b = y_0 \quad \cdots ①$$
$$x_1 a + b = y_1 \quad \cdots ②$$

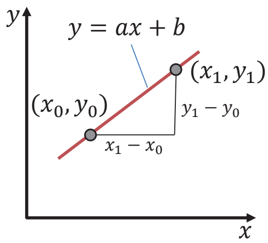
なので, ②式-①式より:

$$a = \frac{y_1 - y_0}{x_1 - x_0}$$

(図からも傾きaがこの式になるのが分かる)

また, ①式に代入すると:

$$b = y_0 - \frac{y_1 - y_0}{x_1 - x_0} x_0$$



線形補間

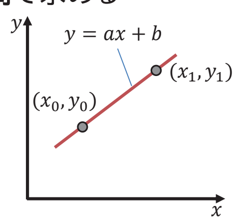
最も基本的な補間が**線形補間**

例) 2点$(x_0, y_0), (x_1, y_1)$を通る関数$y = f(x)$の$x = c$での関数値$f(c)$を**線形補間**で求める

1次関数$y = ax + b$にそれぞれの式を当てはめると

$$f(x) = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$

1次元の線形補間式



1次元での線形補間では**2点**が必要
(1点では補間式が定まらない)

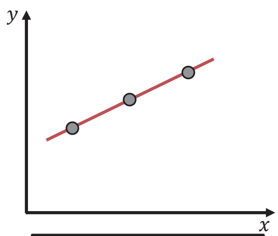
⇒ **通る点数が3以上**だとどうなる?

線形補間

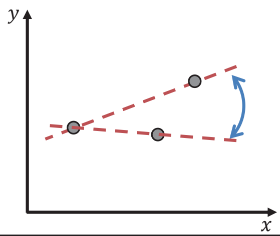
3点以上の場合の補間

3点以上あると1次多項式ではほとんどの場合**すべての点を通る関数は定まらない**

⇒ **2次以上の多項式**に拡張してみよう!



3点が1直線に並べばいいが…



実際には直線上に並ばないことがほとんど

ラグランジュ補間

n次の多項式を用いた補間に拡張

n次多項式: $P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$

$f(x) = P_n(x)$となる係数列$(a_0, a_1, \ldots, a_n)$を求めるためには

何個の点での関数値$f_i = f(x_i)$が必要か?

⇩

$n+1$個の異なる点があれば$P_n(x)$はただ一つに定まる

(ちなみにn個でも$n+2$個でも×)

($P_n(x_i) = f_i$も条件)

ラグランジュ補間

「$n+1$個の異なる点があれば$P_n(x)$はただ一つに定まる」

証明

$n+1$個の点x_iとその点での関数値$f_i = f(x_i)$が与えられたとすると$(i = 0,1,\ldots,n)$, $P_n(x_i) = f_i$なので,

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & & x_1^n \\ \vdots & & & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}$$

行列Vとするとその行列式$\det(V)$はヴァンデルモンドの行列式になり*,

$$\det(V) = \prod_{0 \le i < j \le n} (x_j - x_i)$$

と$i < j$の$(x_j - x_i)$の積の形になる. x_i $(i = 0,1,\ldots,n)$がすべて異なる点ならば$\det(V) \neq 0$でV^{-1}が存在し(正則), $(a_0, a_1, \ldots, a_n)$**がただ一つに定まる**

(証明終了)

*ヴァンデルモンドの行列式がこの形になることの証明は付録参照

ラグランジュ補間

n次の場合の補間式は?

前ページの線形システムを解けばよい?

⇒ 解かなくても**補間多項式は得られる**

ラグランジュ(Lagrange)の補間公式

$n+1$個の点集合(x_i, f_i)があるとき, ラグランジュの補間多項式$L_n(x)$として以下を定義する.

$$L_n(x) = \sum_{i=0}^{n} f_i\, l_i(x)$$

ここで$l_i(x)$は基底多項式と呼ばれ, 以下のように定義する

$$l_i(x) = \prod_{j=0\sim n, j\neq i} \frac{x - x_j}{x_i - x_j} \qquad (i = 0, \ldots, n)$$

ラグランジュ補間

ラグランジュ補間

ちゃんと**すべての点を通る関数**になっているのか?

($L_n(x_k) = f(x_k)$となっているか?)

点x_kでの基本多項式$l_i(x_k)$は, デルタ関数δ_{ik}となる

$$l_i(x_k) = \prod_{j=0\sim n, j\neq i} \frac{x_k - x_j}{x_i - x_j} = \delta_{ik} = \begin{cases} 1 & (k = i) \\ 0 & (k \neq i) \end{cases}$$

($k = i$では$\frac{x_k - x_j}{x_i - x_j}$がすべて1になり, $k \neq i$では$k = j$の項が存在して0になる)

よって$L_n(x)$で$x = x_k$の場合は:

$$L_n(x_k) = \sum_{i=0}^{n} f_i\, l_i(x_k) = f_k \qquad (k = 0, \ldots, n)$$

($i \neq k$で$l_i = 0$なので$i = k$の項だけ残る)

これでラグランジュ補間式が**$n+1$個の点を通ることが証明できた!**

ラグランジュ補間

補間式を具体的に求めてみよう

$$L_n(x) = \sum_{i=0}^{n} f_i\, l_i(x)$$

$$l_i(x) = \prod_{j=0\sim n, j\neq i} \frac{x - x_j}{x_i - x_j}$$

$n = 1$の場合(2点):

$$l_0(x) = \frac{x - x_1}{x_0 - x_1} \qquad l_1(x) = \frac{x - x_0}{x_1 - x_0}$$

$$L_n(x) = f_0 l_0(x) + f_1 l_1(x) = f_0 \frac{x - x_1}{x_0 - x_1} + f_1 \frac{x - x_0}{x_1 - x_0}$$

1次のラグランジュ補間多項式

線形補間式も変形すると:

$$f(x) = \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) + f_0 = \frac{f_1(x - x_0) - f_0(x - x_1)}{x_1 - x_0} = f_1 \frac{x - x_0}{x_1 - x_0} + f_0 \frac{x - x_1}{x_0 - x_1}$$

となり, 結局1次のラグランジュ補間式と同じ

ラグランジュ補間

補間式を具体的に求めてみよう

$$L_n(x) = \sum_{i=0}^{n} f_i\, l_i(x)$$

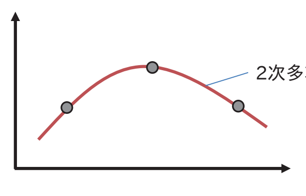
$$l_i(x) = \prod_{j=0\sim n, j\neq i} \frac{x - x_j}{x_i - x_j}$$

$n = 2$の場合(3点):

$$l_0(x) = \frac{x - x_1}{x_0 - x_1}\frac{x - x_2}{x_0 - x_2}, \quad l_1(x) = \frac{x - x_0}{x_1 - x_0}\frac{x - x_2}{x_1 - x_2}, \quad l_2(x) = \frac{x - x_0}{x_2 - x_0}\frac{x - x_1}{x_2 - x_1}$$

$$L_n(x) = f_0 \frac{x - x_1}{x_0 - x_1}\frac{x - x_2}{x_0 - x_2} + f_1 \frac{x - x_0}{x_1 - x_0}\frac{x - x_2}{x_1 - x_2} + f_2 \frac{x - x_0}{x_2 - x_0}\frac{x - x_1}{x_2 - x_1}$$

2次のラグランジュ補間多項式



⇒ 3次, 4次, . . . , n次も同様にして計算できる

ラグランジュ補間

$n+1$点を用いたラグランジュ補間のコード例(次数はn次)
(式をそのまま計算するだけなので計算手順はなしでコード例のみ)

```
double Ln = 0.0;
for(int i = 0; i <= n; ++i){
    // 補間係数(Π(x-xj)/(xi-xj) (j!=i) の計算)
    double l = 1;
    for(int j = 0; j <= n; ++j){
        if(j == i) continue;
        l *= (x-xi[j])/(xi[i]-xi[j]);
    }
    // 補間値
    Ln += fi[i]*l;
}
```

$i \neq j$の条件を忘れないように($i=j$だとゼロ割になってしまう)

ラグランジュ基底多項式の計算：

$$l_i(x) = \prod_{j=0\sim n, j\neq i} \frac{x - x_j}{x_i - x_j}$$

今回の講義のプログラム例では次数をn, データ数をmとしている

ラグランジュ補間

ラグランジュ補間の実行結果例

誤差評価のために, $f(x) = e^x$と設定して, サンプリング点を取って補間してみる(補間位置は$x = 0.5$)

サンプリング点が2点の場合(1次のラグランジュ補間=線形補間)

```
sampling points : (0, 1), (1, 2.718281)
f_lagrangian(0.5) = 1.859140914,  error = 0.2104196435
ground truth = 1.648721271
```

サンプリング点が4点の場合(3次のラグランジュ補間)

```
sampling points : (0, 1), (1, 2.718281), (0.33, 1.390968), (0.66,
1.934792)
f_lagrangian(0.5) = 1.648250737,  error = 0.0004705340683
ground truth = 1.648721271
```

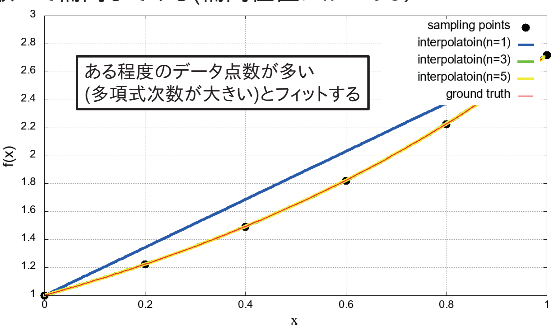
4点の方がより精度が向上している

この例では真値の式が分かっているので精度評価できるが実際には評価が難しい. そのため, **計算方法**や**結果のグラフ**を評価してみよう

ラグランジュ補間

ラグランジュ補間の実行結果例

誤差評価のために, $f(x) = e^x$と設定して, サンプリング点を取って補間してみる(補間位置は$x = 0.5$)



ニュートン補間

$$l_i(x) = \prod_{j=0\sim n, j\neq i} \frac{x - x_j}{x_i - x_j}$$

ラグランジュ補間の問題点1:

すべての基底多項式$l_i\ (i = 0, \ldots, n)$がすべてのサンプル点x_iを含んでいる

⇒ 新しい点が追加されたらl_0から**全部最初から計算し直し**

どうすれば新しい点が追加されたときに**部分計算で済ませられるか?**

$P_{n+1}(x)$が$P_n(x) + \alpha$の形で表されれば良い
(P_nとP_{n+1}の関係を明らかにする)

ニュートン補間

ニュートン補間

$n+1$個の点$(x_0, f_0), \ldots (x_n, f_n)$が与えられたとき,
これらすべての点を通る多項式を以下のように定義する

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

(x_0, f_0)について$f(x_0) = f_0$より

$$f(x_0) = a_0 = f_0$$ (a_1以降の項が$(x - x_0) \to 0$になって消える)

(x_1, f_1)について$f(x_1) = f_1$より

$$f(x_1) = a_0 + a_1(x_1 - x_0) = f_1 \quad \to \quad a_1 = \frac{f_1 - a_0}{x_1 - x_0}$$

(x_2, f_2)について$f(x_2) = f_2$より

$$f(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) = f_2$$

$$\to \quad a_2 = \frac{f_2 - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

ニュートン補間

前ページの手順をまとめると:

(x_0, f_0)からa_0が計算される(0次補間式)
↳ a_0と(x_1, f_1)からa_1が計算される(1次補間式)
　↳ a_0, a_1と(x_2, f_2)からa_2が計算される(2次補間式)
　　↳ a_0, a_1, a_2と(x_3, f_3)から$\cdots$

この方法なら新しい点x_iとそこでの観測値f_iが追加されても,
新しい項a_iを一つ追加で計算するだけで済みそう
⇒ これを公式化しよう

ニュートン補間

a_1の式で$a_0 = f_0 = f(x_0)$なので

$$a_1 = \frac{f_1 - a_0}{x_1 - x_0} = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

$x_1 - x_0 \to 0$の極限を取ると微分になる

⇒ 実際には極限は取っていないので**差分**となる

a_2の式も同様に変形すると

$$a_2 = \frac{f_2 - a_0 - a_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \frac{\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0}$$

$f[x_i, x_j] = \frac{f(x_i) - f(x_j)}{x_i - x_j}$となるような新たなオペレータ$[\cdot]$を定義すると

$$a_2 = \frac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$$

と書ける

⇒ a_1と**同様の形**になった!

ニュートン補間

n番目の係数a_nは

$$a_n = f[x_n, \dots, x_1, x_0] = \frac{f[x_n, \dots, x_2, x_1] - f[x_{n-1}, \dots, x_1, x_0]}{x_n - x_0}$$

ここで, $f[x_0] = f(x_0)$, $f[x_1, x_0] = \frac{f[x_1] - f[x_1]}{x_1 - x_0}$, … を

$f(x)$の**差分商**という

ニュートンの補間公式

$$P_n(x) = f[x_0] + f[x_1, x_0](x - x_0) + f[x_2, x_1, x_0](x - x_0)(x - x_1) + \cdots$$
$$+ f[x_n, \dots, x_1, x_0](x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

$$\boxed{P_n(x) = P_{n-1}(x) + f[x_n, \dots, x_1, x_0] \prod_{i=0}^{n-1} (x - x_i)}$$

ニュートン補間

ラグランジュ補間の問題点1:

すべての基底多項式l_i $(i = 0, \dots, n)$がすべてのサンプル点x_iを含んでいる

⇒ 新しい点が追加されたら**全部最初から計算し直し**

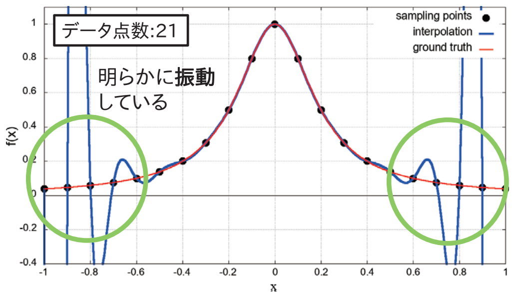
ニュートンの補間公式なら$P_{n-1}(x)$から$P_n(x)$が計算可能

- 新しい点が追加されたら, 次数を1つあげて, **項を一つ追加計算して足すだけ**
- 計算し直しではないので**誤差が蓄積**していき, nが大きくなると**精度に問題が出る可能性あり**
  (精度重視ならラグランジュ補間公式がよい場合もあるということ)

振動の問題

ラグランジュ補間の問題点2:

ルンゲ関数$f(x) = \frac{1}{1+25x^2}$の例



振動の問題

ラグランジュ補間の問題点2:

補間点数が増える(次数が増える)と特に端の方のサンプリング点付近で**大きく振動する(ルンゲ現象)**

解決策

・**チェビシェフ節点**: サンプル点を等間隔ではなく, **端の方ほど間隔を小さくする**

・**スプライン補間**: **導関数**を用いた区分的な多項式補間

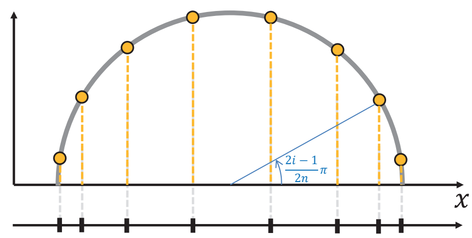
チェビシェフ節点

チェビシェフ節点(Chebyshev nodes)

サンプル点を等間隔ではなく, 端の方ほど間隔を小さくするために三角関数を使う(第一種チェビシェフ多項式)

範囲$[a, b]$のチェビシェフ接点の座標値

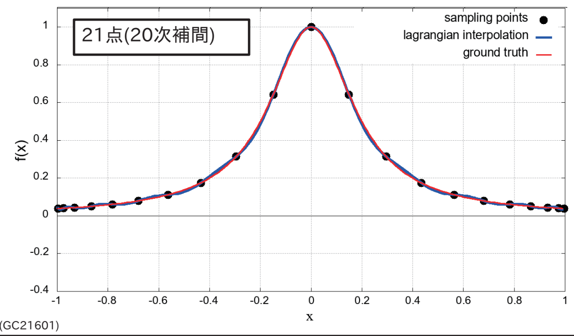
$$x_i = \frac{1}{2}(a + b) + \frac{1}{2}(b - a)\cos\left(\frac{2i - 1}{2n}\pi\right) \quad (i = 1, \dots, n)$$



チェビシェフ節点

チェビシェフ節点を使った結果

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i-1}{2n}\pi\right) \qquad (i = 1, \dots, n)$$



振動の問題

ラグランジュ補間の問題点2:
補間点数が増える(次数が増える)と特に端の方のサンプリング点付近で**大きく振動する(ルンゲ現象)**

解決策

・**チェビシェフ節点**：サンプル点を等間隔ではなく，**端の方ほど間隔を小さくする**

・**スプライン補間**: **導関数**を用いた区分的な多項式補間

スプライン補間

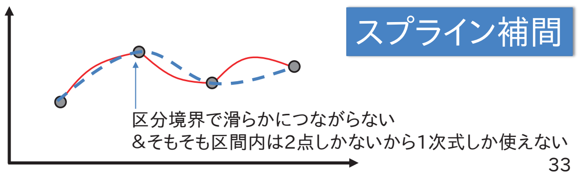
サンプリング点が多い場合に，**全体で1つの多項式**で近似すると振動が発生する

⇒ データ領域を区切って，**各区分毎に多項式近似**すれば良いのでは？

問題点 区分の境界で滑らかにならないのでは？

解決策 滑らかになるように**導関数**を使おう

スプライン補間



スプライン補間

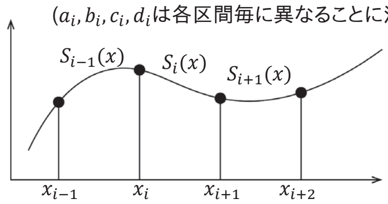
3次スプライン補間*

区間$[x_i, x_{i+1}]$ を**3次多項式で近似**する

$$S_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i$$

⇒ 他の区間も同様にそれぞれ3次多項式で近似する
(例えば,区間$[x_{i-1}, x_i]$を$S_{i-1}(x)$, $[x_{i+1}, x_{i+2}]$を$S_{i+1}(x)$で近似)

3次多項式の**係数をどのようにして求めるのか**？
(a_i, b_i, c_i, d_iは各区間毎に異なることに注意)



データ数を$n+1$とすると区間数はn, つまり**未知数は全体で$4n$**

*区間毎に多項式を定義するのでデータ点数と次数は無関係となることに注意($n = m-1$とは限らない)

スプライン補間

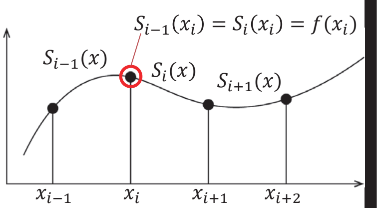
$4n$個の係数を求めるために**条件をつける**

[条件1] 区間の境界点x_iにおいて多項式の値$S_{i-1}(x)$と$S_i(x)$は$f(x)$の値に一致する($S_i(x)$がデータ点を通る)

$$S_i(x_i) = f_i, \ S_i(x_{i+1}) = f_{i+1}$$

⇒ 各区間毎に式が2個, つまり全体で**$2n$個の式**

$4n$個の**未知数に対して式が少ない**ので, 多項式を定義できない！
⇒ 別の条件を追加

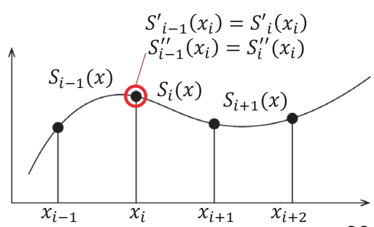


スプライン補間

$4n$個の係数を求めるために**条件を更に追加する**

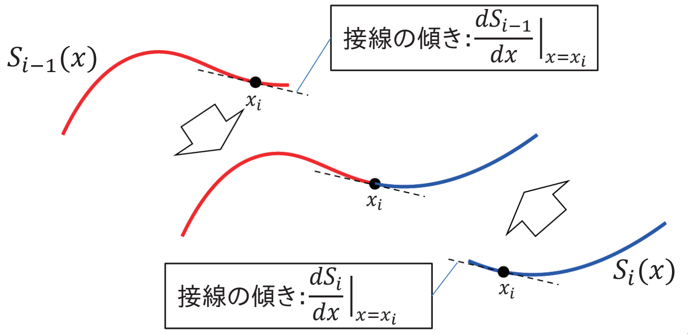
[条件2] 各区間が**滑らかに接続**する = 各点x_iで**1次導関数と2次導関数が一致**する

$$S'_{i-1}(x_i) = S'_i(x_i), \quad S''_{i-1}(x_i) = S''_i(x_i)$$



スプライン補間

導関数が等しい($\frac{dS_{i-1}}{dx} = \frac{dS_i}{dx}$)と**何故滑らかにつながるのか?** ⇒ **導関数=接線の傾き**ということが重要



スプライン補間

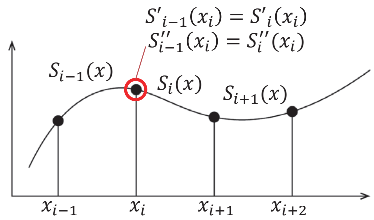
$4n$個の係数を求めるために**条件を更に追加する**

[条件2] 各区間が**滑らかに接続**する = 各点x_iで**1次導関数と2次導関数*が一致**する

$$S'_{i-1}(x_i) = S'_i(x_i), \quad S''_{i-1}(x_i) = S''_i(x_i)$$

⇒ 区間の境界毎に式が2個, つまり全体で**$2(n-1)$個の式**

条件1,2で合計$4n-2$
あと残り2個!



*2次多項式なら1次導関数のみでOK. 実際にTrueTypeフォントは2次スプライン(2次ベジエ)曲線を使っている(PostScriptは3次).

スプライン補間

$4n$個の係数を求めるために**条件を更に追加する**

[条件3] **両端点**(x_0とx_n)の**2次導関数が0**

(両端点の外にはつながる区間がないので傾きが変化しないと仮定, 紐の両端のイメージ)

$$S''(x_0) = 0, \quad S''(x_n) = 0$$

3つの条件で**$4n$個の式**ができたので**係数を求めていこう**

条件3を使った3次スプライン補間を**自然3次スプライン**と呼ぶ. 他にもx_0とx_nが滑らかにつながるということを条件3とした**周期3次スプライン**などもある.

スプライン補間

補間係数の式を求めるために先に2次導関数を求めておく

$$S''_i(x) = 6a_i(x - x_i) + 2b_i \quad (i = 0, 1, \ldots, n)$$

2次導関数を$u_i = S''_j(x_i) \quad (j = i-1, i)$として,
まずは4つの係数(a_i, b_i, c_i, d_i)をu_iとx_i, f_iを使って表してみる
(条件2から$u_i = S''_{i-1}(x_i) = S''_i(x_i)$であることに注意)

- 区間の左端$x = x_i$で$u_i = S''_i(x_i) = 2b_i$となるので,

$$\boxed{b_i = \frac{u_i}{2}}$$

- 区間の右端$x = x_{i+1}$で$u_{i+1} = S''_i(x_{i+1}) = 6a_i(x_{i+1} - x_i) + 2b_i$より

$$\boxed{a_i = \frac{u_{i+1} - u_i}{6(x_{i+1} - x_i)}}$$

スプライン補間

条件1より

- 区間の左端$x = x_i$で$S_i(x_i) = d_i = f_i$となるので,

$$\boxed{d_i = f_i}$$

- 区間の右端$x = x_{i+1}$で
$S_i(x_{i+1}) = a_i(x_{i+1} - x_i)^3 + b_i(x_{i+1} - x_i)^2 + c_i(x_{i+1} - x_i) + d_i = f_{i+1}$
より

$$c_i = \frac{f_{i+1} - a_i(x_{i+1} - x_i)^3 - b_i(x_{i+1} - x_i) - d_i}{x_{i+1} - x_i}$$

ここまでで求めたa_i, b_i, d_iの式を代入して整理

$$\boxed{c_i = \frac{f_{i+1} - f_i}{x_{i+1} - x_i} - \frac{1}{6}(x_{i+1} - x_i)(2u_i + u_{i+1})}$$

スプライン補間

係数(a_i, b_i, c_i, d_i)をu_iとx_i, f_iを使って表せた

⇒ (x_i, f_i)はデータ点なので**既知**. よって後は**u_iを求めれば良い**

条件2より

境界点x_{i+1}で両区間の**1次導関数は等しい**ので

$$S'_i(x_{i+1}) = S'_{i+1}(x_{i+1})$$

$S'_{i+1}(x) = 3a_{i+1}(x - x_{i+1})^2 + 2b_{i+1}(x - x_{i+1}) + c_{i+1}$なので

$$3a_i(x_{i+1} - x_i)^2 + 2b_i(x_{i+1} - x_i) + c_i = c_{i+1}$$

a_i, b_i, c_iの式を(c_{i+1}に注意しながら)代入して,
$h_i = x_{i+1} - x_i$として整理すると

$$\boxed{h_i u_i + 2(h_i + h_{i+1})u_{i+1} + h_{i+1}u_{i+2} = 6\left(\frac{f_{i+2} - f_{i+1}}{h_{i+1}} - \frac{f_{i+1} - f_i}{h_i}\right)}$$

スプライン補間

$$h_i u_i + 2(h_i + h_{i+1})u_{i+1} + h_{i+1}u_{i+2} = 6\left(\frac{f_{i+2} - f_{i+1}}{h_{i+1}} - \frac{f_{i+1} - f_i}{h_i}\right)$$

全体の両端点(x_0とx_n)だと条件2が成り立たない
⇒ この**式の個数は$n-1$個**($i = 0,1,\dots,n-2$)

条件3から両端点では$u_0 = u_n = 0$となるので,
未知のu_iは$n-1$個($i = 1,2,\dots,n-1$)

⇩

線形システム$H\boldsymbol{u} = \boldsymbol{v}$として解けば
u_iをすべて計算可能

スプライン補間

u_iを求めるための$(n-1)\times(n-1)$の線形システム$H\boldsymbol{u} = \boldsymbol{v}$

$$H = \begin{pmatrix} 2(h_0+h_1) & h_1 & 0 & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \cdots & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & \vdots \\ \vdots & \vdots & h_3 & & 0 \\ 0 & & & \ddots & h_{n-2} \\ 0 & 0 & \cdots & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{pmatrix}$$

$$\boldsymbol{u} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{pmatrix} \quad \boldsymbol{v} = 6\begin{pmatrix} \frac{f_2-f_1}{h_1} - \frac{f_1-f_0}{h_0} \\ \frac{f_3-f_2}{h_2} - \frac{f_2-f_1}{h_1} \\ \frac{f_4-f_3}{h_3} - \frac{f_3-f_2}{h_2} \\ \vdots \\ \frac{f_{n-1}-f_{n-2}}{h_{n-2}} - \frac{f_{n-2}-f_{n-3}}{h_{n-3}} \\ \frac{f_n-f_{n-1}}{h_{n-1}} - \frac{f_{n-1}-f_{n-2}}{h_{n-2}} \end{pmatrix}$$

Hは**対称疎行列**なので**CG法などで解ける**
⇒ u_iが分かったら, u_iから係数a_i, b_i, c_i, d_iを計算できる

スプライン補間

スプライン補間の計算手順

1. $h_i = x_{i+1} - x_i$を計算($i = 0,\dots,n-1$)
2. 係数行列Hと右辺項ベクトル$\boldsymbol{v}$を計算
3. 線形システムソルバ(CG法など)で$H\boldsymbol{u} = \boldsymbol{v}$を解いて, u_iを計算($i = 1,\dots,n-1$)
4. $u_0 = u_n = 0$とし, 係数a_i, b_i, c_i, d_iを計算($i = 0,\dots,n-1$)

以下, 計算位置xが与えられたときの処理

5. xが含まれる区間kの計算($x \geq x_k$かつ$x < x_{k+1}$となる区間)
   ⇒ 区間幅がすべて同じなら$k = \mathrm{floor}\left(\frac{x-x_0}{h}\right)$ *
6. $S_k(x) = a_k(x-x_k)^3 + b_k(x-x_k)^2 + c_k(x-x_k) + d_k$を計算

*floor(x)は床関数でx以下の最大の整数を表す
(要は小数点以下切り捨て)

スプライン補間

スプライン補間のコード例(線形システムを解くところまで)

```
vector<double> a(n), b(n), c(n), d(n); // 各区間における補間係数
vector<double> h(n); // 各補間区間の幅
for(int i = 0; i < n; ++i){
    h[i] = xi[i+1]-xi[i];
}

// 位置x_iでの多項式の2階微分u_iの計算
// 線形システムの係数行列Hと右辺項ベクトルbの計算
vector< vector<double> > H(n-1);
vector<double> tmp(n-1, 0.0), v(n-1, 0.0);
for(int i = 0; i < n-1; ++i){
    H[i].resize(n-1, 0.0); // 係数行列の各行を0で初期化
    // 係数行列と右辺項ベクトルの要素の計算
    if(i != 0) H[i][i-1] = h[i];
    H[i][i] = 2*(h[i]+h[i+1]);
    if(i != n-2) H[i][i+1] = h[i+1];
    v[i] = 6*((f[i+2]-f[i+1])/h[i+1]-(f[i+1]-f[i])/h[i]);
}
// CG法で線形システムを解く(Hは対称疎行列)
int max_iter = 100;
double eps = 1e-6;
cg_solver(H, v, tmp, n-1, max_iter, eps);
```

区間幅h_iの計算

係数行列Hと右辺項ベクトルvの要素の計算.
係数行列の要素は
$i = 0, n-2$の行では2つになることに注意

共役勾配法で線形システム$H\boldsymbol{u} = \boldsymbol{v}$を解く

スプライン補間

スプライン補間のコード例(係数計算と関数値計算)

```
// 係数a,b,c,dの計算
vector<double> u(n+1, 0.0);
for(int i = 0; i < n-1; ++i) u[i+1] = tmp[i];
for(int i = 0; i < n; ++i){
    a[i] = (u[i+1]-u[i])/(6.0*h[i]);
    b[i] = u[i]/2.0;
    c[i] = (f[i+1]-f[i])/h[i]-h[i]*(2*u[i]+u[i+1])/6.0;
    d[i] = f[i];
}

// xが含まれる区間の探索(区間幅がすべて同じならint(x/h)で求められる)
int k = 0;
for(int i = 0; i < n; ++i){
    if(x >= xi[i] && x < xi[i+1]){ k = i; break; }
}

// 計算済みの係数を使って3次多項式で補間値を計算
double dx = x-xi[k];
ans = a[k]*dx*dx*dx + b[k]*dx*dx + c[k]*dx + d[k];
```

$u_0 = u_n = 0$ を考慮して,
u_i用の配列を作り,
係数a_i, b_i, c_i, d_iを計算

関数値を計算したい位置xを含む区間kの探索

3次多項式$S_k(x)$を計算

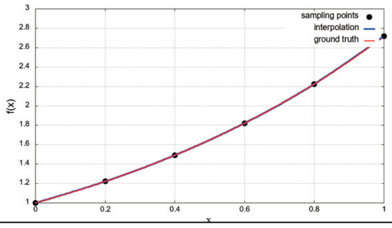
スプライン補間

スプライン補間の実行結果例

誤差評価のために, $f(x) = e^x$と設定して, サンプリング点を取って補間してみる(補間位置は$x = 0.5$)

サンプリング点数:6 ($x = 0, 0.2, 0.4, 0.6, 0.8, 1$)

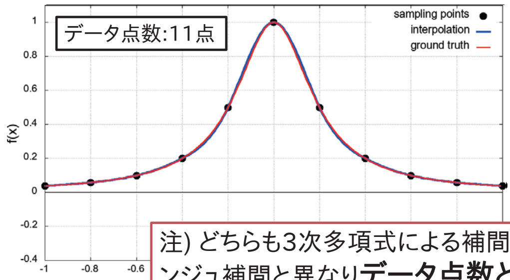
```
sampling points : (0, 1),  (0.2, 1.2214),  (0.4, 1.49182),  (0.6,
1.82212),  (0.8, 2.22554),  (1, 2.71828)
f_spline(0.5) = 1.649202088,  error = 0.0004808177638
ground truth = 1.648721271
```



スプライン補間

3次スプライン補間の実行結果例

ルンゲ関数$f(x) = \frac{1}{1+25x^2}$の補間結果



注) どちらも3次多項式による補間. ラグランジュ補間と異なり**データ点数と次数は独立して設定できる**ことに注意

今回の講義内容

- 今日の問題
- ラグランジュ補間とニュートン補間
- スプライン補間
- **最小2乗法による関数近似**

最小2乗近似

ラグランジュ補間ではデータ点数が増えると多項式の次数が増え, 振動してしまう

⇒ スプライン補間は区分ごとに多項式を定義するが, **全体で1つの多項式**にする方法はないのか?

⇩

データ点数を増やしてもラグランジュ補間の**次数を増やさなければ**いいのでは?

⇒ 線形システムが成り立たなくなる

$P_n(x)$をただ一つに定めるためには$n+1$個のデータ点が必要ということを思いだそう

$$\begin{pmatrix}1 & x_0 & x_0^2 & \cdots & x_0^n\\ 1 & x_1 & x_1^2 & & x_1^n \\ \vdots & & & \ddots & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n\end{pmatrix}\begin{pmatrix}a_0\\a_1\\ \vdots\\a_n\end{pmatrix}=\begin{pmatrix}f_0\\f_1\\ \vdots\\f_m\end{pmatrix}$$

$m > n$だと正方行列ではなくなり, $(a_0, a_1, \ldots, a_n)$がただ1つに定まらない!

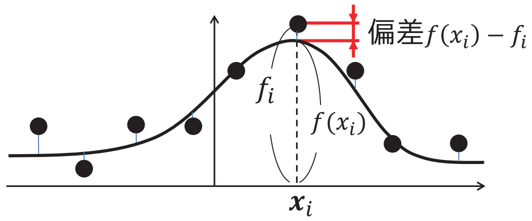
最小2乗近似

ラグランジュ補間(&スプライン補間)では, **「すべてのデータ点を通る」**関数を求めた

⇩

この考えをやめて, **「データ点と関数の偏差がなるべく小さくなるようにする」**にしてはどうか?

(つまり, 必ずその点上を通るのではなく近くを通ればOKとする)



最小2乗近似

n次の多項式$f(x)$で近似する(ここでのnはデータ点数とは無関係)

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

m個のデータ点(x_i, f_i)が与えられたときの**誤差関数**は:

偏差の和 $E = \sum_{i=0}^{m-1} f(x_i) - f_i$

⇩ これだと+と−で打ち消し合う可能性があるので2乗しよう*

偏差の2乗和 $E = \sum_{i=0}^{m-1} (f(x_i) - f_i)^2$

*2乗するのはこの理由だけでなく, Eが前回やった凸関数になるようにするという理由もある

最小2乗近似

Eを最小にするような係数a_iを求めたいのだから

$$\frac{\partial E}{\partial a_i} = 0 \qquad (i = 0, \ldots, n)$$

を解けばよい.

未知数a_iは$n+1$個で式も$n+1$個 ⇒ **線形システムとして解ける!**

最小2乗法(最小自乗法)

⇨ 最小2乗法を具体的に定式化していこう

最小2乗近似

多項式$f(x)$を**係数ベクトル$\boldsymbol{c}$と基底ベクトル$\boldsymbol{b}(x)$の積**で表現

$$f(\boldsymbol{x}) = \boldsymbol{b}(x)^T\boldsymbol{c} \quad \Rightarrow \quad E = \sum(\boldsymbol{b}(x_i)^T\boldsymbol{c} - f_i)^2$$

$$\boldsymbol{c} = (a_0, a_1, a_2, \dots, a_n)^T,\ \boldsymbol{b} = (1, x, x^2, \dots, x^n)^T$$

係数ベクトル$\boldsymbol{c} = (a_0, \dots, a_n)^T$をどうやって求めるのか？

⇨ Eは$\boldsymbol{c}$についての2次関数なので**微分値が0となる$\boldsymbol{c}$を求めればよい**

$$\frac{\partial E}{\partial a_k} = \sum_i 2b_k(\boldsymbol{x}_i)\,(\boldsymbol{b}(x_i)^T\boldsymbol{c} - f_i) = 0\,, \qquad k = 0, \dots, n$$

$\boldsymbol{b}(x_i) = (b_0, b_1, \dots, b_n)^T$で$n+1$個の式を全部まとめてベクトル表記すると

$$2\sum_i(\boldsymbol{b}(x_i)\boldsymbol{b}(x_i)^T\boldsymbol{c} - \boldsymbol{b}(x_i)f_i) = \boldsymbol{0}$$

最小2乗近似

前ページの式を係数ベクトル$\boldsymbol{c}$について解くと：

$$\boldsymbol{c} = \left(\sum_i \boldsymbol{b}(x_i)\boldsymbol{b}(x_i)^T\right)^{-1}\sum_i \boldsymbol{b}(x_i)f_i$$

$A = \sum_i \boldsymbol{b}(x_i)\boldsymbol{b}(x_i)^T\,, \boldsymbol{y} = \sum_i \boldsymbol{b}(x_i)f_i$ とすると

$A\boldsymbol{c} = \boldsymbol{y}$の線形システムを解くのと同じ

例) 2次多項式$a_0 + a_1x + a_2x^2$による近似(データ点数m)

$$\boldsymbol{b}(x) = (1, x, x^2)^T, \quad \boldsymbol{c} = (a_0, a_1, a_2)^T$$

$$\sum_{i=0}^{m-1}\begin{pmatrix} 1 & x_i & x_i^2 \\ x_i & x_i^2 & x_i^3 \\ x_i^2 & x_i^3 & x_i^4 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \sum_{i=0}^{m-1}\begin{pmatrix} 1 \\ x_i \\ x_i^2 \end{pmatrix} f_i$$

最小2乗近似

基底ベクトルを変えるだけで**多次元でも同じ方法で解ける**

例) 2次元空間で2次の多項式で近似する場合

$$f(\boldsymbol{x}) = a_0 + a_1x + a_2y + a_3x^2 + a_4xy + a_5y^2$$

$$\sum_i\begin{pmatrix} 1 & x_i & y_i & x_i^2 & x_iy_i & y_i^2 \\ x_i & x_i^2 & x_iy_i & x_i^3 & x_i^2y_i & x_iy_i^2 \\ y_i & x_iy_i & y_i^2 & x_i^2y_i & x_iy_i^2 & y_i^3 \\ x_i^2 & x_i^3 & x_i^2y_i & x_i^4 & x_i^3y_i & x_i^2y_i^2 \\ x_iy_i & x_i^2y_i & x_iy_i^2 & x_i^3y_i & x_i^2y_i^2 & x_iy_i^3 \\ y_i^2 & x_iy_i^2 & y_i^3 & x_i^2y_i^2 & x_iy_i^3 & y_i^4 \end{pmatrix}\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{pmatrix} = \sum_i\begin{pmatrix} 1 \\ x_i \\ y_i \\ x_i^2 \\ x_iy_i \\ y_i^2 \end{pmatrix} f_i$$

$\boxed{\boldsymbol{b}(\boldsymbol{x}) = (1, x, y, x^2, xy, y^2)^T,}\quad \boldsymbol{c} = (a_0, a_1, a_2, a_3, a_4, a_5)^T$

最小2乗近似

線形システム$A\boldsymbol{c} = \boldsymbol{y}$について

- $A\boldsymbol{c} = \boldsymbol{y}$を**正規方程式**と呼ぶ
- Aは**対称行列**. ただし, 基底ベクトル次第ではあるが, 単純な多項式だと**疎行列にはならず条件数も大きい**(そのため共役勾配法は使いづらい)*

  ⇒ **ガウスの消去法やLU分解**を使う必要がある

最小2乗近似

最小2乗近似の計算手順

1. 基底ベクトル$\boldsymbol{b}(\boldsymbol{x})$を設定, 係数行列$A$と右辺項ベクトル$\boldsymbol{y}$の全要素を0で初期化
2. 各データ点$\boldsymbol{x}_i$について以下の処理を反復($k = 0,1,\dots,n-1$)
   a. $\boldsymbol{x}_i$での基底ベクトル$\boldsymbol{b}(\boldsymbol{x}_i)$を計算
   b. 係数行列Aと右辺項ベクトル$\boldsymbol{y}$を更新:
      $A \leftarrow A + \boldsymbol{b}(\boldsymbol{x}_i)\boldsymbol{b}(\boldsymbol{x}_i)^T, \quad \boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{b}(\boldsymbol{x}_i)f(\boldsymbol{x}_i)$
3. $A\boldsymbol{c} = \boldsymbol{y}$をLU分解などで解いて係数ベクトル$\boldsymbol{c}$を求める

以下, 計算位置xが与えられたときの処理

4. $f(\boldsymbol{x}) = \boldsymbol{b}(x)^T\boldsymbol{c}$を計算して関数値を求める

最小2乗近似

最小2乗近似のコード例(線形システム設定まで)

データ数nでm次元多項式で近似, 結果を返すために係数用配列cは引数で与えている

```
// 多項式の次数から行列のサイズ(=基底ベクトルの次元数)を計算
int dim_b = m+1; // 1次元の場合

// Ac=y, bは基底ベクトル
c.resize(dim_b, 0.0);// 結果の係数ベクトル
vector<double> b(dim_b, 0.0), y(dim_b, 0.0);
vector< vector<double> > A(dim_b, b);
for(int k = 0; k < n; ++k){ // データ分だけ反復
    // 多項式の基底ベクトルの計算
    double xb = 1;
    for(int i = 0; i < dim_b; ++i){
        b[i] = xb; xb *= xi[k];
    }
    // 係数行列Aと右辺項bの計算
    for(int i = 0; i < dim_b; ++i){
        for(int j = 0; j < dim_b; ++j){
            A[i][j] += b[i]*b[j];
        }
        y[i] += b[i]*fi[k];
    }
}
```

基底ベクトルの大きさは1次元ならば多項式の次元mに1を足したもの(0次項分を足す)

データ点x_iでの基底ベクトルの計算. ここでは多項式を使っているがそれ以外の基底を使う場合はここを変えれば良い

係数行列Aと右辺項ベクトル$\boldsymbol{y}$の要素の計算. 外側のkでのループがそれぞれの式の$\sum$に相当する

最小2乗近似

最小2乗補間のコード例(線形システムを解く&関数値計算)

```
// Ac=yをLU分解で解く(疎行列とは限らないのでCGソルバは使わない)
LUDecomp(A, dim_b);
LUSolver(A, y, c, dim_b);

// 基底ベクトルに求めた係数を掛けて行くことで位置xにおける値yを計算
double fx = 0;
double xb = 1;
for(int i = 0; i < dim_b; ++i){
    fx += c[i]*xb; xb *= x;
}
```

位置xにおける基底ベクトルと係数ベクトルの内積を取ることで関数値を計算

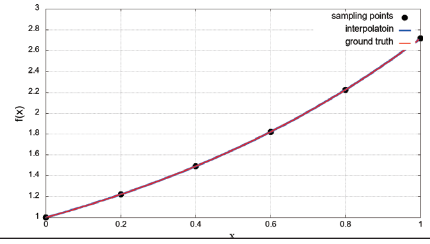
最小2乗近似

最小2乗近似(3次多項式)の実行結果例

誤差評価のために, $f(x) = e^x$と設定して, サンプリング点を取って補間してみる(補間位置は$x = 0.5$)

サンプリング点数:6 ($x = 0, 0.2, 0.4, 0.6, 0.8, 1$), 3次多項式近似

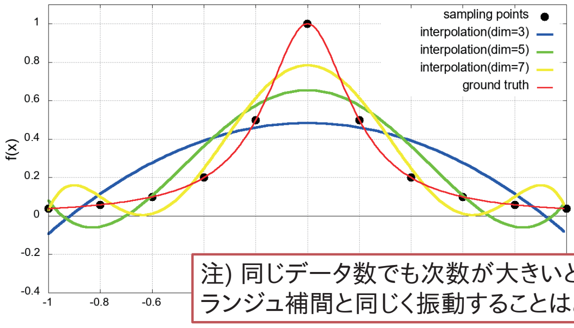
```
sampling points : (0, 1),  (0.2, 1.2214),  (0.4, 1.49182),  (0.6, 1.82212),  (0.8, 2.22554),  (1, 2.71828)
f_ls3(0.5) = 1.64816,  error = 0.000562449
ground truth = 1.64872
```



最小2乗近似

最小2乗近似で次数を変えた時の実行結果例

ルンゲ関数$f(x) = \frac{1}{1+25x^2}$の補間結果(データ点数:11)

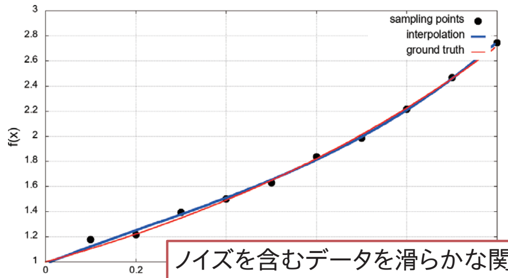


注) 同じデータ数でも次数が大きいとラグランジュ補間と同じく振動することはある

最小2乗近似

最小2乗近似は***ノイズに強い***

$f(x) = e^x$に$f(x)$の大きさの±7.5%の乱数(ホワイトノイズ)を加えたデータ点に対する3次多項式近似結果 (データ点数:11)



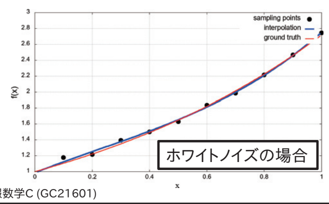
ノイズを含むデータを滑らかな関数で近似したい場合に最小2乗近似は有効

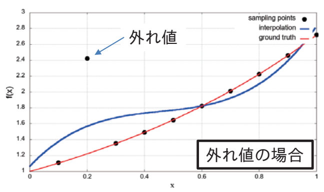
最小2乗近似

最小2乗法に限らず, データ点x_iを何らかの関数$f(x)$で近似することを**回帰分析**と呼ぶ

最小2乗法はどのようなノイズに対しても強いというわけではない

- ホワイトノイズのような**全体的に分布するような*ノイズには強い***が, 1点だけ大きく外れるような**外れ値(outlier)に弱い ⇒ ロバスト推定**





最小2乗近似

ロバスト推定 (ノイズや外れ値に対応できる手法)

- **最小メディアン法(LMedS*)** : 2乗和の代わりに**中央値(メディアン)を誤差関数**にする(中央値は少数の外れ値に影響されない)
- **RANSAC** : **ランダムサンプリング**したデータ点でパラメータ推定する処理を繰り返し, **最も誤差の小さかったパラメータ**を使う
- **M推定** : データ点に**重みを設定**する方法. 重み一定で一度最小2乗法で推定後, **誤差の大きいデータの重みを小さく**して, またパラメータ推定を繰り返す.

*Least Median of Squares

最小2乗近似

線形システム$A\boldsymbol{c} = \boldsymbol{y}$について

- $A\boldsymbol{c} = \boldsymbol{y}$を**正規方程式**と呼ぶ
- Aは**対称行列**. ただし, 基底ベクトル次第ではあるが, 単純な多項式だと**疎行列にはならず条件数も大きい**(そのため共役勾配法は使いづらい)

⇩

もしAが**対角行列**ならば$A\boldsymbol{c} = \boldsymbol{y}$は直接解ける

Aが対角行列となる条件：
基底ベクトルの要素(関数)が**直交**している

最小2乗近似

近似関数$f(x)$をn個の関数$\phi_k(x)$の**線形結合**とする

$$f(x) = \sum_{k=1}^{n} a_k \phi_k(x) = a_1\phi_1(x) + a_2\phi_2(x) + \cdots + a_n\phi_n(x)$$

係数a_kを最小2乗法で求めるとすると正規方程式は：

$$\begin{pmatrix} \int_\alpha^\beta \phi_1(x)^2 dx & \int_\alpha^\beta \phi_1(x)\phi_2(x)dx & \cdots & \int_\alpha^\beta \phi_1(x)\phi_n(x)dx \\ \int_\alpha^\beta \phi_2(x)\phi_1(x)dx & \int_\alpha^\beta \phi_2(x)^2 dx & \cdots & \int_\alpha^\beta \phi_2(x)\phi_n(x)dx \\ \vdots & \vdots & \ddots & \vdots \\ \int_\alpha^\beta \phi_n(x)\phi_1(x)dx & \int_\alpha^\beta \phi_n(x)\phi_2(x)dx & \cdots & \int_\alpha^\beta \phi_n(x)^2 dx \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \int_\alpha^\beta \phi_1(x)f(x)dx \\ \int_\alpha^\beta \phi_2(x)f(x)dx \\ \vdots \\ \int_\alpha^\beta \phi_n(x)f(x)dx \end{pmatrix}$$

ここで, $A = \sum_i \boldsymbol{b}(\boldsymbol{x}_i)\boldsymbol{b}(\boldsymbol{x}_i)^T$なのでその要素は**関数同士の内積**となっていることに注意.
関数同士の内積は範囲を$[\alpha, \beta]$とすると, $\int_\alpha^\beta \phi_i(x)\phi_j(x)dx$で計算される.

最小2乗近似

もし, $\int_\alpha^\beta \phi_i(x)\phi_j(x)dx = 0 \ (i \neq j)$ となるように$\phi_i(x)$を設定すれば, 正規方程式のAは**対角行列**になる

$$\begin{pmatrix} \int_\alpha^\beta \phi_1(x)^2 dx & 0 & \cdots & 0 \\ 0 & \int_\alpha^\beta \phi_2(x)^2 dx & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \int_\alpha^\beta \phi_n(x)^2 dx \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} \int_\alpha^\beta \phi_1(x)f(x)dx \\ \int_\alpha^\beta \phi_2(x)f(x)dx \\ \vdots \\ \int_\alpha^\beta \phi_n(x)f(x)dx \end{pmatrix}$$

係数は $a_k = \dfrac{\int_\alpha^\beta \phi_k(x)f(x)dx}{\int_\alpha^\beta \phi_k(x)^2 dx}$ と簡単に計算できる!

$\int_\alpha^\beta \phi_i(x)\phi_j(x)dx = 0 \ (i \neq j)$ を満たす関数列を**直交関数系**と呼ぶ*

*$\int_\alpha^\beta \phi_i(x)^2 dx \neq 0$でなければならないことにも注意. 更に$\int_\alpha^\beta \phi_i(x)^2 dx = 1$ならば正規直交関数系と呼ぶ

最小2乗近似

直交関数系の例*

- 三角関数系($x \in [-\pi, \ \pi]$) ⇨ **フーリエ変換**

$$\{\phi_k\} = \{1, \cos x, \sin x, \cos 2x, \sin 2x, \ldots, \cos mx, \sin nx\}$$
$(m, n = 1, 2, \ldots)$

- **ルジャンドル多項式**($x \in [-1, 1]$)

$$\{\phi_k\} = \left\{1, x, \frac{1}{2}(3x^2 - 1), \frac{1}{2}(5x^3 - 3x), \ldots, \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2 - 1)^n\right\}$$
$(n = 0, 1, \ldots)$

- **エルミート多項式**($x \in [-\infty, \infty]$)

$$\{\phi_k\} = \left\{1, 2x, 4x^2 - 2, 8x^3 - 12x, \ldots, (-1)^n e^{x^2}\frac{d^n}{dx^n}e^{-x^2}\right\}$$
$(n = 0, 1, \ldots)$

*他にもラゲール多項式やチェビシェフ多項式などがある

講義内容のまとめ

- 今日の問題：
  データ(x_i, f_i)にフィットする多項式$f(x)$を求める
- ラグランジュ補間とニュートン補間
  - データ点を通る多項式近似
- スプライン補間
  - 導関数を用いた区分毎の多項式近似
- 最小2乗法による関数近似
  - 任意関数のフィッティング
  - ロバスト推定(外れ値に頑強な推定), 直交関数

Appendix

(以降のページは補足資料です)

ヴァンデルモンドの行列式

行列式の性質から$x_0 = x_1$のとき(1行目と2行目が全く同じになる)，$\det(V) = 0$となる.

$$\det(V) = \begin{vmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & & x_1^n \\ & \vdots & & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{vmatrix}$$

⇒ $\det(V)$は多項式になるので，$(x_0 - x_1)$が因数になる(因数定理)
　他の行についても同様に考えていくと:

$$\det(V) = C(x) \prod_{0 \le i < j \le n} (x_j - x_i)$$

$\det(V)$は行列式の定義から$\frac{1}{2}n(n-1)$次の多項式であり，

$\prod_{0 \le i < j \le n}(x_j - x_i)$は$i < j$の条件から$\frac{1}{2}n(n-1)$個の項の掛け合わせとなる.

⇒ $C(x)$は定数(0次)

対角成分を掛け合わせた項$(x_1 x_2^2 \cdots x_n^n)$の係数から$C(x) = 1$となるので，

$$\det(V) = \prod_{0 \le i < j \le n} (x_j - x_i)$$

ディラックのデルタ関数

ある集合に対して$i = j$で1，そうでない場合に0となることを表したい場合:

$$\delta_{ij} = \begin{cases} 1 & (i = j) \\ 0 & (i \neq j) \end{cases}$$

と書くことができる(**クロネッカーのデルタ**).
これを拡張して，連続量に対して$x = 0$で∞，それ以外のところで0となる**関数として定義**したのが**ディラックのデルタ関数**である*.

$$\delta(x) = \begin{cases} \infty & (x = 0) \\ 0 & (x \neq 0) \end{cases}$$

ただし，このままだと積分すると∞に発散してしまうので数学的には良くない. そのため，以下の性質も持つ関数と定義されている.

$$\int_{-\infty}^{\infty} f(x)\delta(x)dx = f(0) \quad \xRightarrow{f(x)=1\text{なら}} \quad \int_{-\infty}^{\infty} \delta(x)dx = 1$$

*関数(function)といっているが数学での分類上では超関数(distribution)
超関数:関数の概念を一般化したもので，不連続関数を滑らかな関数として扱う

チェビシェフ多項式

第一種チェビシェフ多項式

$P_n(x) = \cos(nt)$　ただし $x = \cos(t)$

$P_n(x)$を求めていくと(倍角公式$\cos(2t) = 2\cos^2(t) - 1$ から)，

$P_0(x) = 1$,
$P_1(x) = x$,
$P_2(x) = 2x^2 - 1, \ldots,$

⇨ $P_{n+1}(x) = 2xP_n(x) - P_{n-1}(x)$ (漸化式)

第二種チェビシェフ多項式

$P_n(x) = \sin(nt)/\sin(t)$　ただし $x = \cos(t)$

$P_n(x)$を求めていくと

$P_0(x) = 1$,
$P_1(x) = 2x$,
$P_2(x) = 4x^2 - 1, \ldots,$

⇨ $P_{n+1}(x) = 2xP_n(x) - P_{n-1}(x)$ (漸化式)

三角関数の直交性

三角関数系$\{1, \cos(mx), \sin(nx)\}$　$(m, n = 1,2, \ldots)$の直交性を確かめる.

三角関数の積和公式 $\sin(nx)\sin(mx) = \frac{1}{2}\cos((m-n)x) - \frac{1}{2}\cos((m+n)x)$
より

$$\int_{-\pi}^{\pi} \sin(nx)\sin(mx)\,dx = \int_{-\pi}^{\pi} \left\{\frac{1}{2}\cos((m-n)x) - \frac{1}{2}\cos((m+n)x)\right\} dx$$
$$= \left[\frac{1}{2(m-n)}\sin((m-n)x) - \frac{1}{2(m+n)}\sin((m+n)x)\right]_{-\pi}^{\pi}$$

[$m \neq n$の場合] $x = -\pi\ or\ \pi$で$\sin((m \pm n)\pi)$は0, つまり,

$$\int_{-\pi}^{\pi} \sin(nx)\sin(mx)\,dx = 0 \qquad (m \neq n)$$

[$m = n$の場合] 積分前の時点で$\cos((m-n)x) = 1$となるので,

$$\int_{-\pi}^{\pi} \sin(nx)\sin(mx)\,dx = \left[\frac{x}{2} - \frac{1}{2(m+n)}\sin((m+n)x)\right]_{-\pi}^{\pi} = \pi \qquad (m = n)$$

$\int_{-\pi}^{\pi}\cos(nx)\sin(mx)\,dx$や$\int_{-\pi}^{\pi}\cos(nx)\cos(mx)\,dx$についても同様