スプライン・
一般化加法モデル

非線形回帰

- 多項式回帰

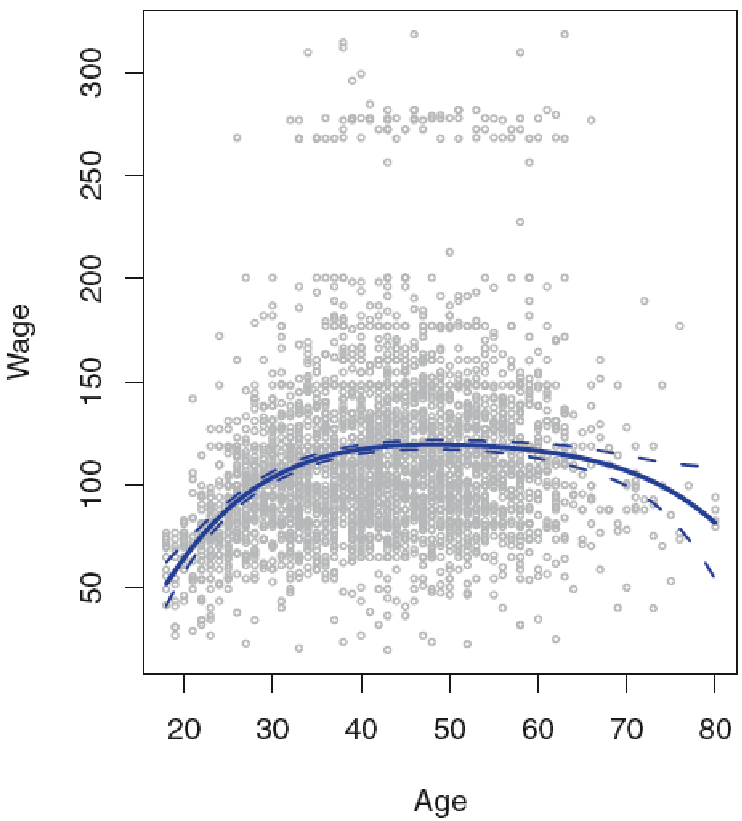
$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \cdots + \beta_d x_i^d + \epsilon_i$$

  - 高次の多項式を用いることで非線形性の強い入出力関係をモデル化できる
  - パラメータの推定法は線形回帰と全く同じ

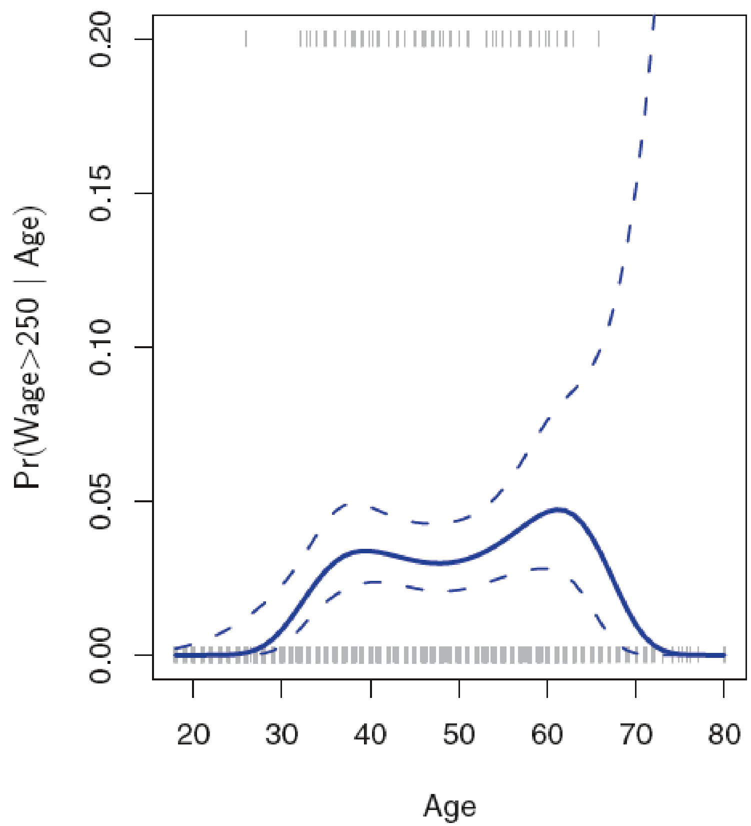
- 問題点
  - 次数を大きくすると(特に入力変数の値の最小・最大付近で)モデルの出力値が不安定に

- 例）Wage データセット
  - 3000人の賃金データ

4次多項式による回帰
（点線は$\hat{f}(x)$の95%信頼区間）



4次多項式によるロジスティック回帰
（点線は$\hat{f}(x)$の95%信頼区間）



→ Age の値が大きいところでの信頼区間が大きくなってしまっている

ステップ関数による方法

- 多項式回帰では入力値の区間全体をひとつの多項式でモデル化
- 区間を分割し、個々の区間を（より単純な）モデルで近似できないか

- ステップ関数による方法
  - 入力値の区間の分割点: $c_1, c_2, \ldots, c_K$
  - ダミー変数

$$
\begin{aligned}
C_0(X) &= I(X < c_1) \\
C_1(X) &= I(c_1 \le X < c_2) \\
C_2(X) &= I(c_2 \le X < c_3) \\
&\vdots \\
C_{K-1}(X) &= I(c_{K-1} \le X < c_K) \\
C_K(X) &= I(c_K \le X)
\end{aligned}
$$

$I(\cdot)$: 指示関数（indicator function）
条件が成立すれば 1 、そうでなければ 0 を返す関数

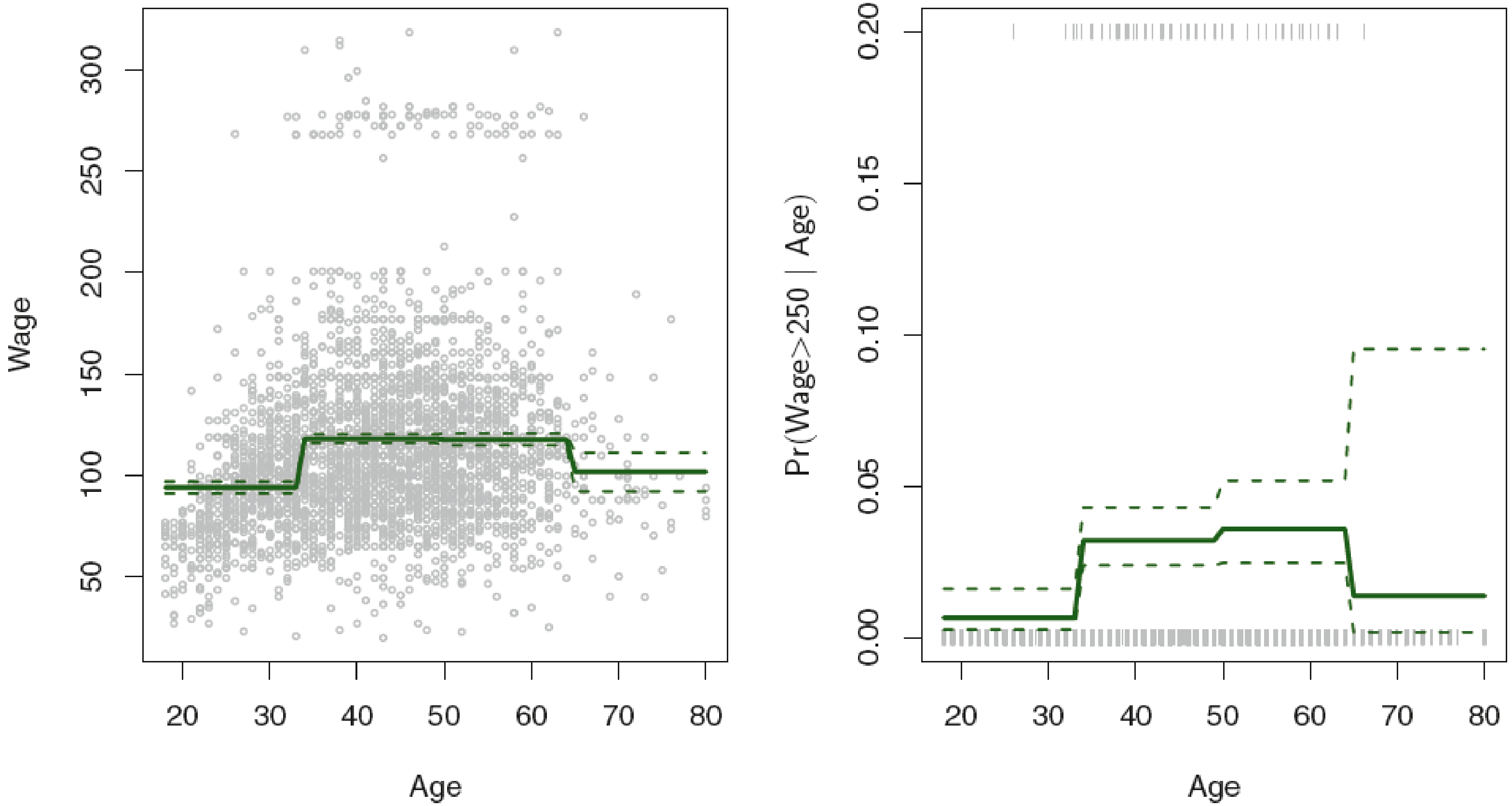
$C_0(X) + C_1(X) + \cdots + C_K(X) = 1$ が常に成立

  - 線形モデル

$$
y_i = \beta_0 + \beta_1 C_1(x_i) + \beta_2 C_2(x_i) + \cdots + \beta_K C_K(x_i) + \epsilon_i
$$

ステップ関数による回帰・分類

- 入力値の区間を4分割



基底関数

- 基底関数（basis functions）による表現

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \cdots + \beta_K b_K(x_i) + \epsilon_i$$

  – 基底関数 $b_1(\cdot), b_2(\cdot), \ldots, b_K(\cdot)$
    - 多項式の場合: $b_j(x_i) = x_i^j$
    - ステップ関数の場合: $b_j(x_i) = I\left(c_j \leq x_i < c_{j+1}\right)$

  – パラメータ推定法は線形回帰と同様

回帰スプライン

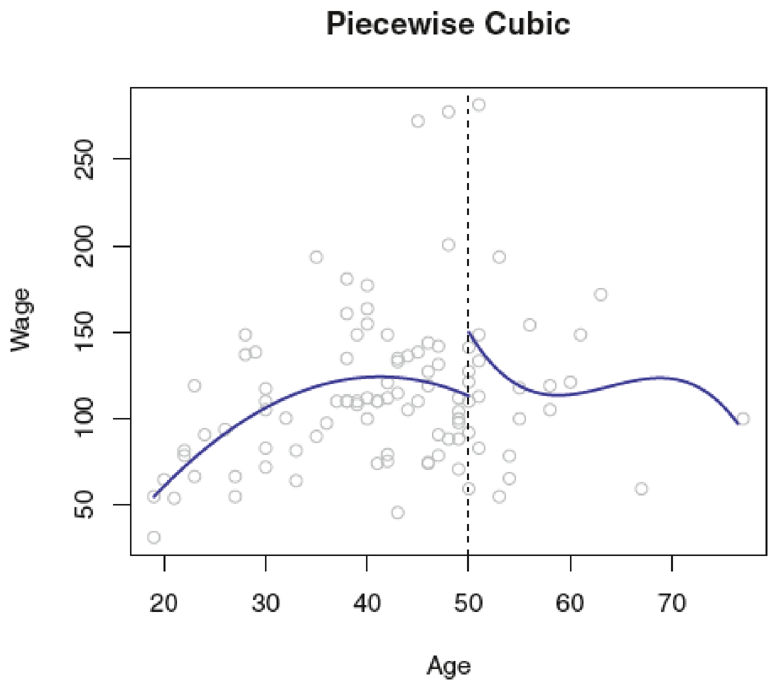
- 区分多項式回帰（piecewise polynomial regression）
  - 区間ごとに低次の多項式モデル
    $$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$
    を適用
  - パラメータ $\beta_0, \beta_1, \beta_2, \beta_3$ は区間ごとに異なる
  - 区間の境目は節点（knot）と呼ばれる

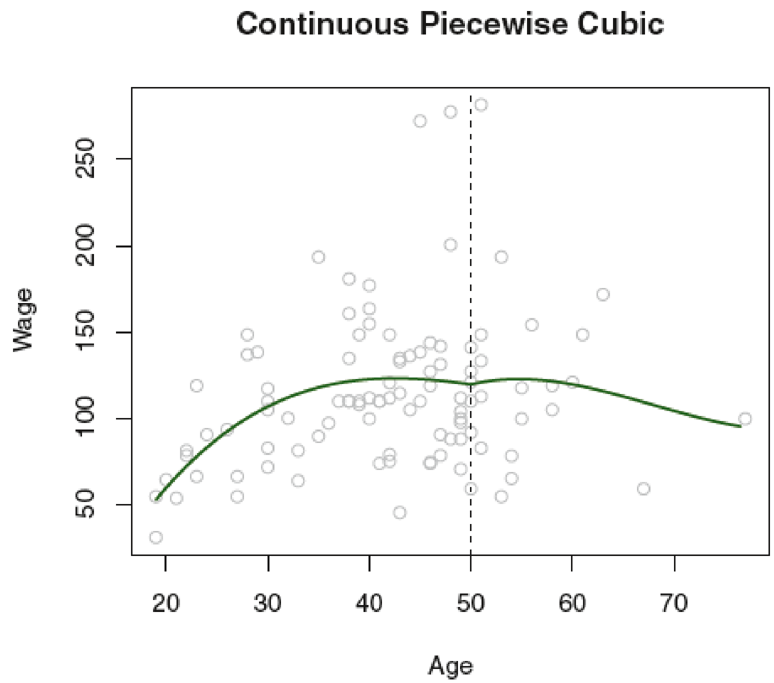
  - 例
$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \epsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \epsilon_i & \text{if } x_i \geq c \end{cases}$$

回帰スプライン

- 例
  - $c = 50$

age = 50 で連続という制約を追加





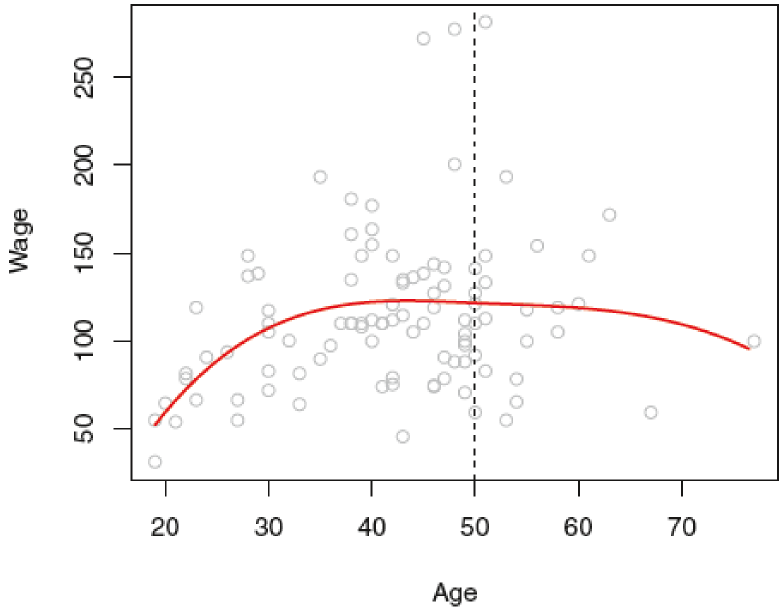
自由度 ＝ 8

自由度 ＝ 7

回帰スプライン

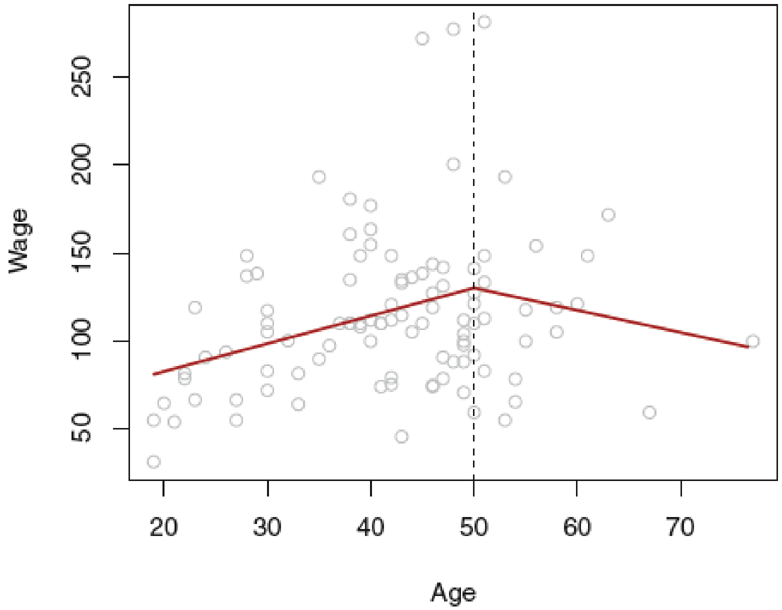
一階微分と二階微分も
age = 50 で連続

3次スプライン（cubic spline）



区分線形で連続

線形スプライン（linear spline）
（1次スプライン）



自由度 ＝ 5

スプライン基底関数

- K 個の節点を持つ3次スプライン
  - 自由度: $4(K+1)-3K=4+K$

$$y_i=\beta_0+\beta_1 b_1(x_i)+\beta_2 b_2(x_i)+\cdots+\beta_{K+3} b_{K+3}(x_i)+\epsilon_i$$

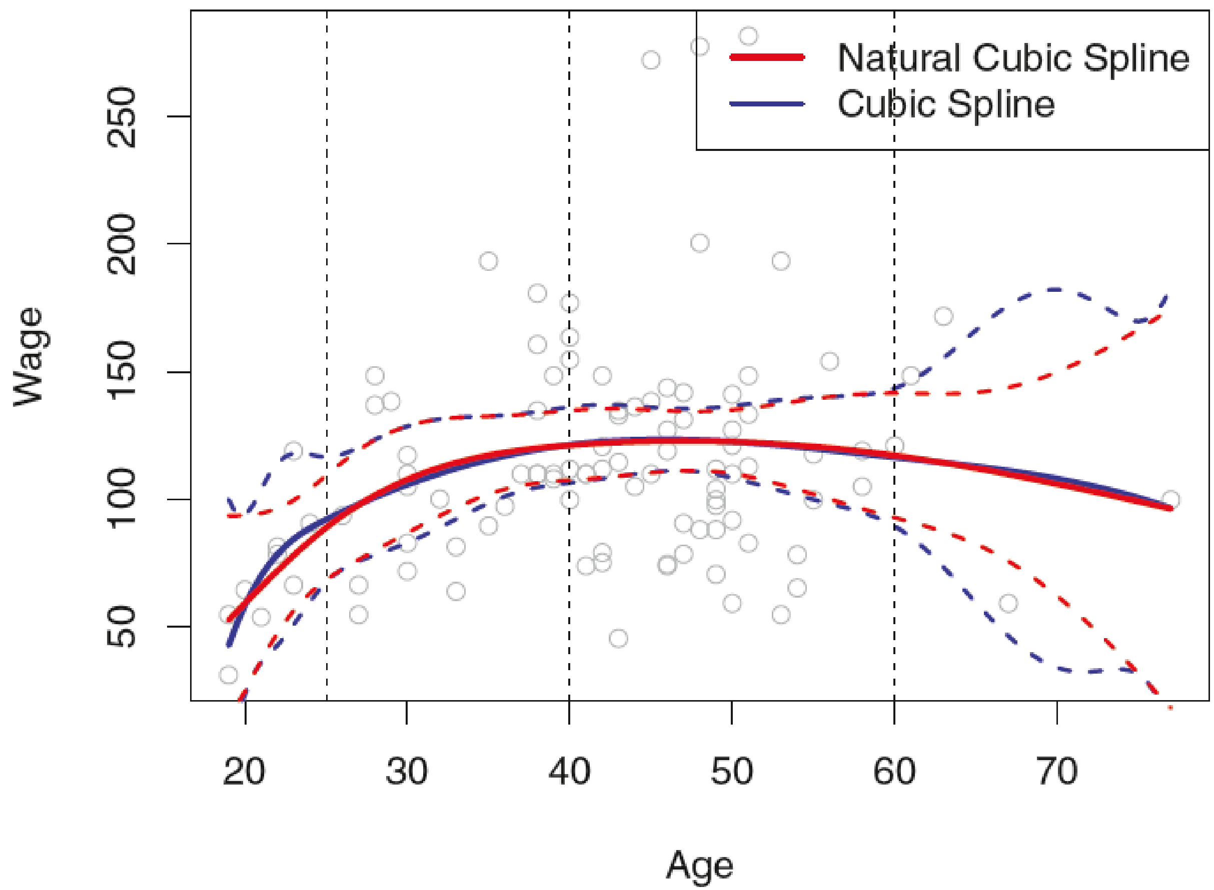
- 基底関数
  - 3次多項式: $b_1(x)=x,\ b_2(x)=x^2,\ b_3(x)=x^3$
  - 切断べき基底関数（truncated power basis function）
    - 各節点 ξ ごとに

$$h(x,\xi)=(x-\xi)_+^3=\begin{cases}(x-\xi)^3 & \text{if } x>\xi \\ 0 & \text{otherwise}\end{cases}$$

→ 節点で二階微分まで連続

自然スプライン

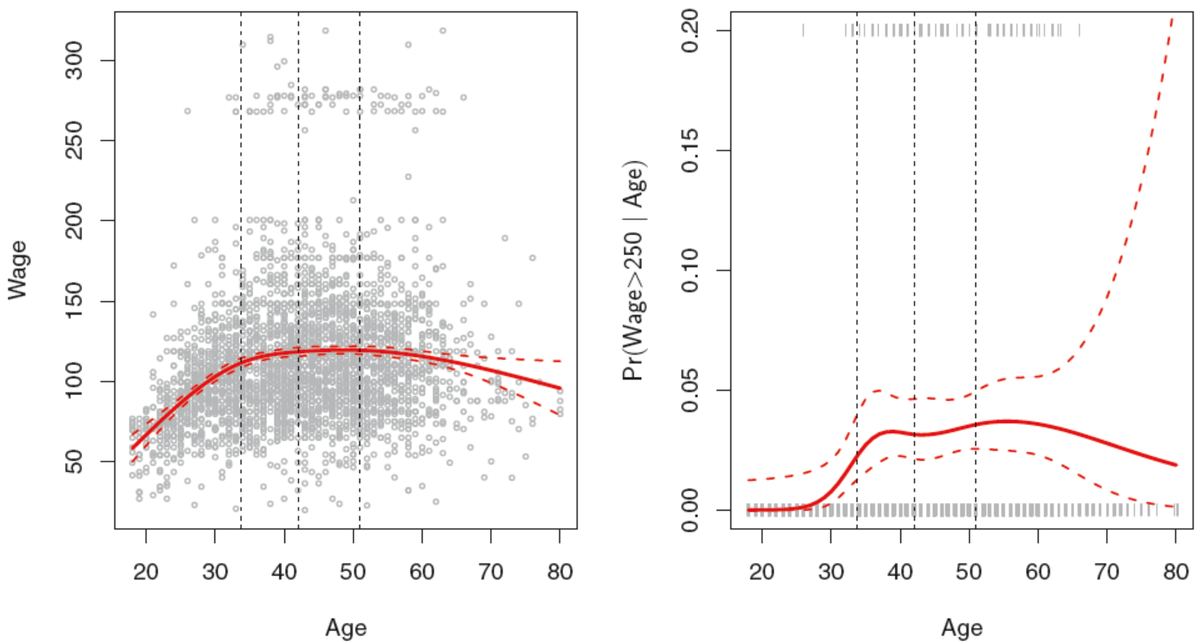
- 自然スプライン（natural spline）
  - 制約の追加：
    - 境界節点（Rのデフォルトではデータの両端）の外側で線形
  - 両端付近での推定値が安定



節点の数と位置

- 節点の位置の決め方の例
  - 節点の数（または自由度）は所与
  - 節点の位置はパーセンタイルで均等割り

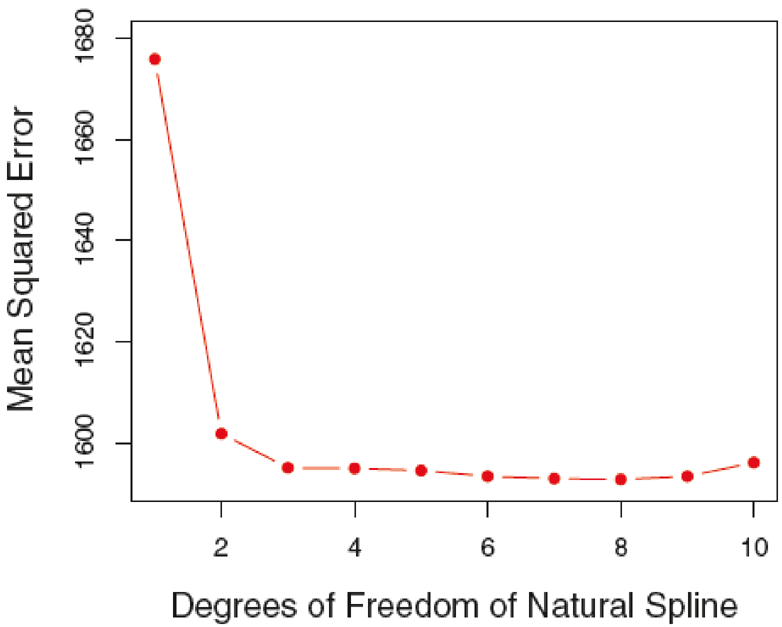
スプライン　　　ロジスティック回帰



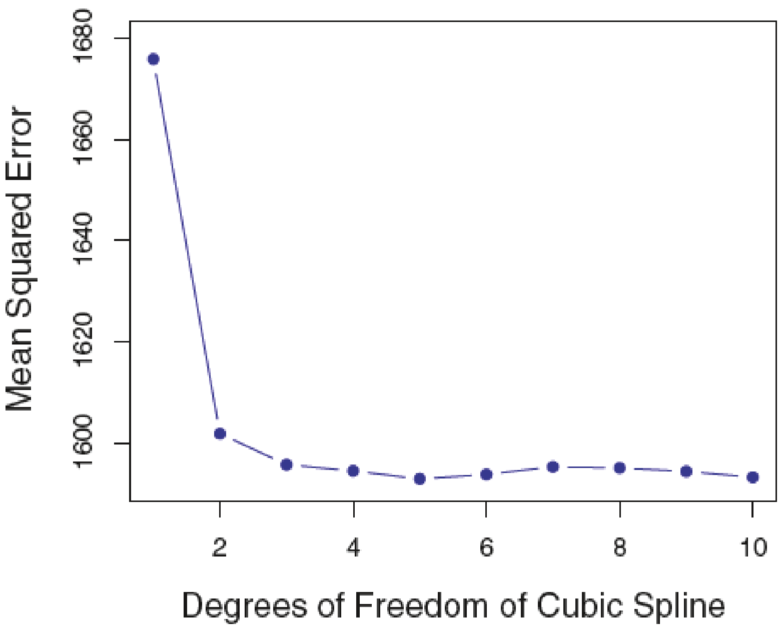
節点の位置：25, 50, 75パーセンタイル

節点の数と位置

- 交差検証による節点数の決定



自然スプラインの自由度

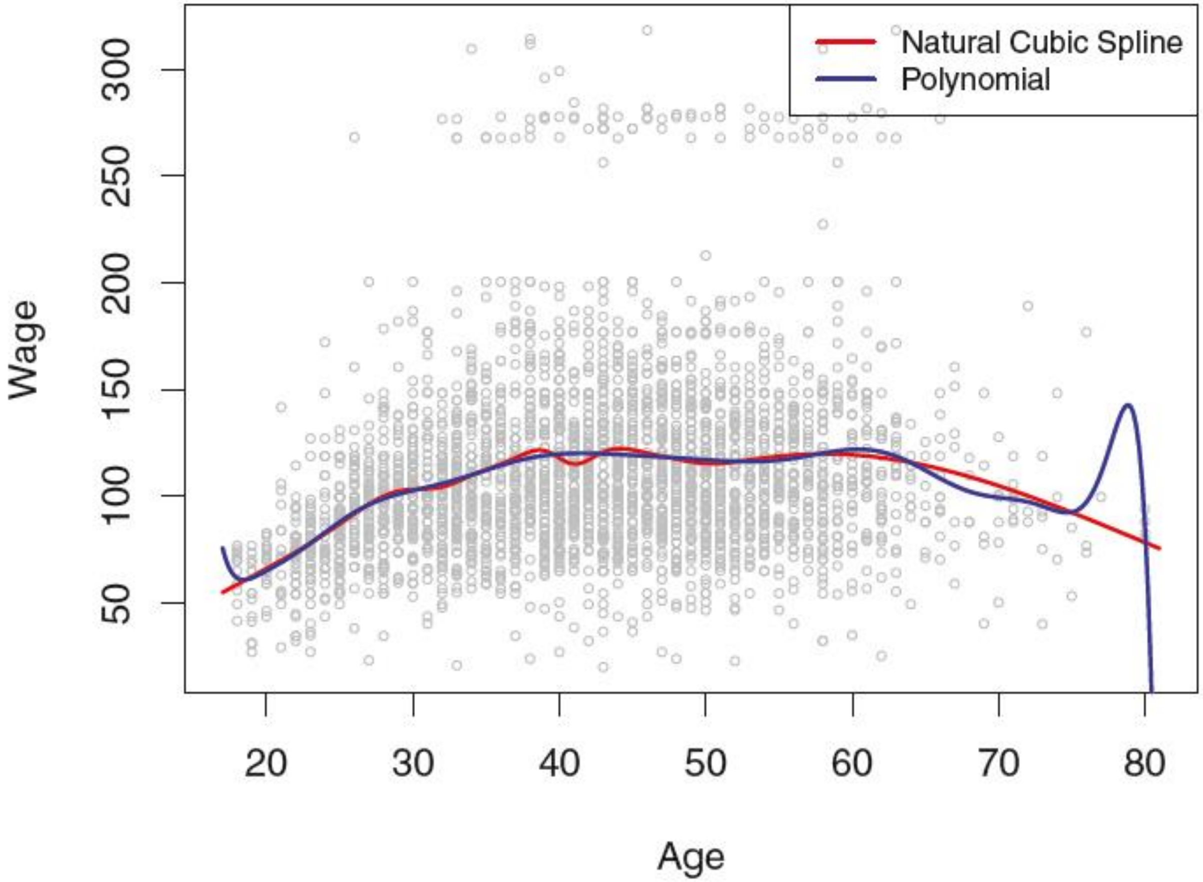


３次スプラインの自由度

スプライン vs 多項式回帰

- 例

自由度 = 15



高次の多項式回帰はデータの両端付近で不安定になりがち

Python実習

- 準備
  - Wage.csv をダウンロード
    - http://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/sml/Wage.csv

```
>>> import os
>>> os.chdir(os.path.expanduser("~"))
>>> import pandas as pd
>>> Wage = pd.read_csv('Wage.csv', header=0)
>>> Wage.shape
...
>>> Wage.head(10)
```

  - Wage
    - アメリカ Washington, D.C. 周辺地域の賃金データ

Wage データ

year	age	sex	maritl	race	education	region	jobclass	health	health_ins	logwage	wage
2006	18	1. Male	1. Never Married	1. White	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.318063	75.04315
2004	24	1. Male	1. Never Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	2. No	4.255273	70.47602
2003	45	1. Male	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.875061	130.9822
2003	43	1. Male	2. Married	3. Asian	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	5.041393	154.6853
2005	50	1. Male	4. Divorced	1. White	2. HS Grad	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.318063	75.04315
2008	54	1. Male	2. Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	4.845098	127.1157
2009	44	1. Male	2. Married	4. Other	3. Some College	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	5.133021	169.5285
2008	30	1. Male	1. Never Married	3. Asian	3. Some College	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.716003	111.7208
2006	41	1. Male	1. Never Married	2. Black	3. Some College	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	4.778151	118.8844
2004	52	1. Male	2. Married	1. White	2. HS Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	4.857332	128.6805
2007	45	1. Male	4. Divorced	1. White	3. Some College	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.763428	117.1468
2007	34	1. Male	2. Married	1. White	2. HS Grad	2. Middle Atlantic	1. Industrial	2. >=Very Good	2. No	4.39794	81.28325
2005	35	1. Male	1. Never Married	1. White	2. HS Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	4.494155	89.49248
2003	39	1. Male	2. Married	1. White	4. College Grad	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	4.90309	134.7054
2009	54	1. Male	2. Married	1. White	2. HS Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	4.90309	134.7054
2009	51	1. Male	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	2. >=Very Good	1. Yes	4.50515	90.48191
2003	37	1. Male	1. Never Married	3. Asian	4. College Grad	2. Middle Atlantic	1. Industrial	2. >=Very Good	2. No	4.414973	82.67964

Python実習

- スプライン回帰
  - bs()
    - スプラインの基底関数の値の行列を作成
    - デフォルトでは3次スプライン

```
>>> import numpy as np
>>> from patsy import dmatrix
>>> basis1 = dmatrix("bs(Wage.age, knots=(25,40,60), degree=3)",
{"Wage.age": Wage.age}, return_type='dataframe')
>>> import statsmodels.api as sm
>>> fit1 = sm.GLM(Wage.wage, basis1).fit()
>>> fit1.summary()
...
```

Python実習

- 基底関数の値の生成
  - 自由度(df)を指定

```
>>> basis2 = dmatrix("bs(Wage.age, df=6)", {"Wage.age":
Wage.age}, return_type='dataframe')
>>> fit2 = sm.GLM(Wage.wage, basis2).fit()
>>> fit2.summary()
...
```

Python実習

- 自然スプライン
  - `cr()`
    - 自然スプラインの基底行列を生成
    - 自由度(df)を指定

```
>>> basis3 = dmatrix("cr(Wage.age, df=4)", {"Wage.age":
Wage.age}, return_type='dataframe')
>>> fit3 = sm.GLM(Wage.wage, basis3).fit()
>>> fit3.summary()
...
```

Python実習

- 予測

```
>>> ag = np.arange(Wage.age.min(), Wage.age.max()).reshape(-1,1)
>>> pred1 = fit1.predict(dmatrix("bs(ag, knots=(25,40,60))",
{"age_grid": ag}, return_type='dataframe'))
>>> pred2 = fit2.predict(dmatrix("bs(ag, df=6)", {"age_grid": ag},
return_type='dataframe'))
>>> pred3 = fit3.predict(dmatrix("cr(ag, df=4)", {"age_grid": ag},
return_type='dataframe'))
```

Python実習

- プロット

```
>>> import matplotlib.pyplot as plt
>>> plt.scatter(Wage.age, Wage.wage, facecolor='None',
edgecolor='k', alpha=0.1)
>>> plt.plot(ag, pred1, color='r', label='Cubic spine with
knots at [25, 40, 60]')
>>> plt.plot(ag, pred2, color='g', label='Cubic spine with
df=6')
>>> plt.plot(ag, pred3, color='b', label='Natural spline
df=4')
>>> plt.legend()
>>> plt.xlabel('age')
>>> plt.ylabel('wage')
>>> plt.show()
```

平滑化スプライン

- 学習データをうまく近似する関数 $g(x)$ を求めたい
- 単に各データ点での誤差 $\sum_{i=1}^{n}(y_i - g(x_i))^2$ を最小化しようとするとデータ点をつなぐだけの関数になってしまう
- 関数は滑らかであってほしい

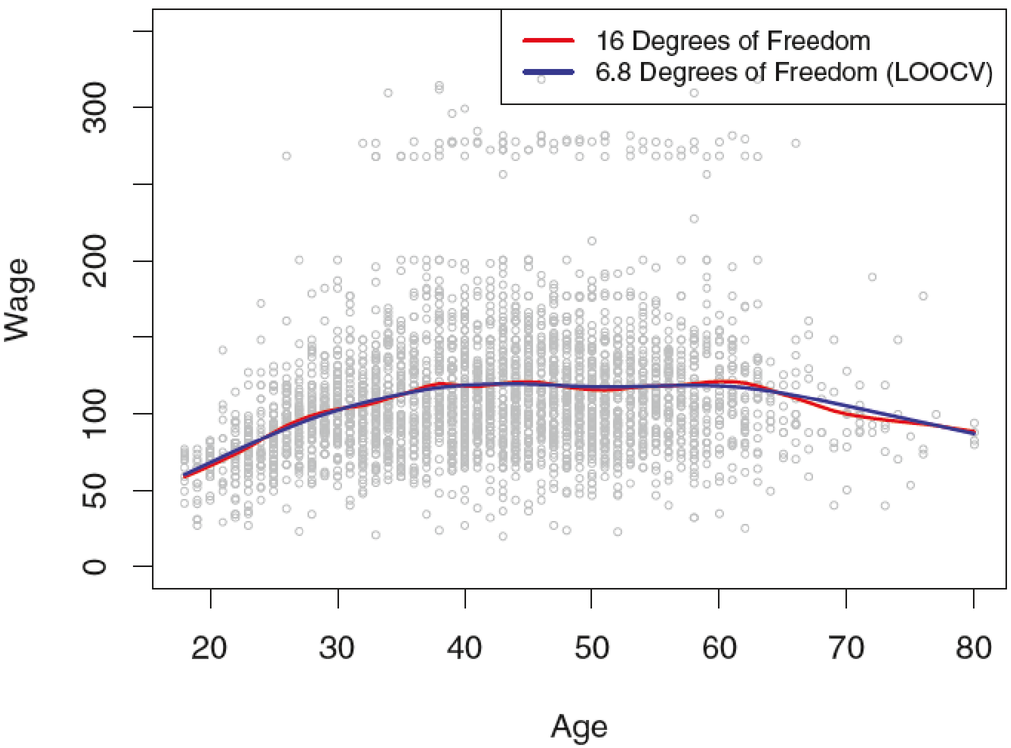
- 以下の目的関数を最小化する関数 $g(x)$ を求める

$$\underbrace{\sum_{i=1}^{n}\left(y_i - g(x_i)\right)^2}_{\text{各データ点でのずれ}} + \underbrace{\lambda \int g''(t)^2 dt}_{\text{滑らかでないことに対するペナルティ}}$$

- この最適化問題の解は $x_1, x_2, \ldots, x_n$ を節点とする（ペナルティ付きの）自然3次スプラインとして得られる

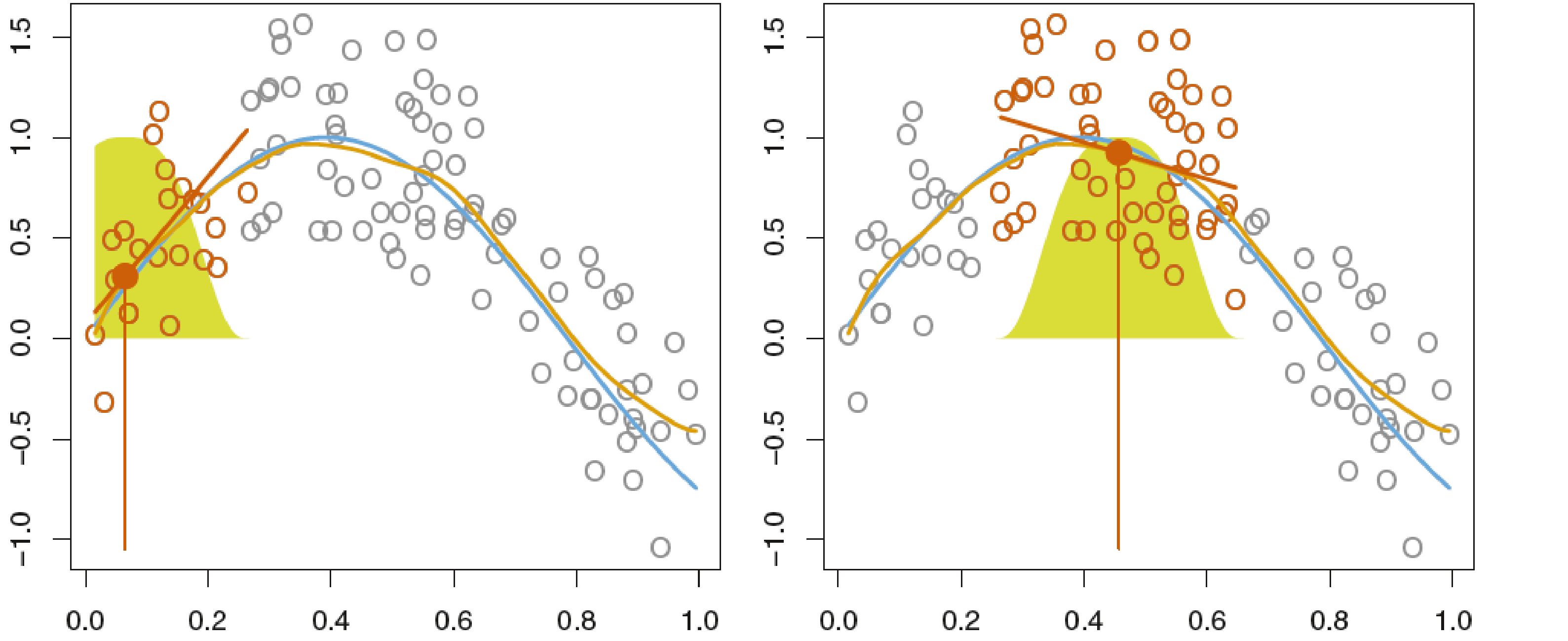
平滑化スプライン

- ハイパーパラメータ λ
  - 関数 $g(x)$ が滑らかでないことに対するペナルティの大きさ
  - 有効自由度（effective degrees of freedom）df_λを制御
    - λ がゼロ → $df_\lambda = n$
    - λ が無限大→ $df_\lambda = 2$



局所回帰

- 局所回帰（local regression）モデル
  - 対象となる点の近傍のデータ点のみを用いて回帰



青い曲線: 真の f(x)
オレンジ色の曲線: 局所回帰によって得られた曲線

局所回帰

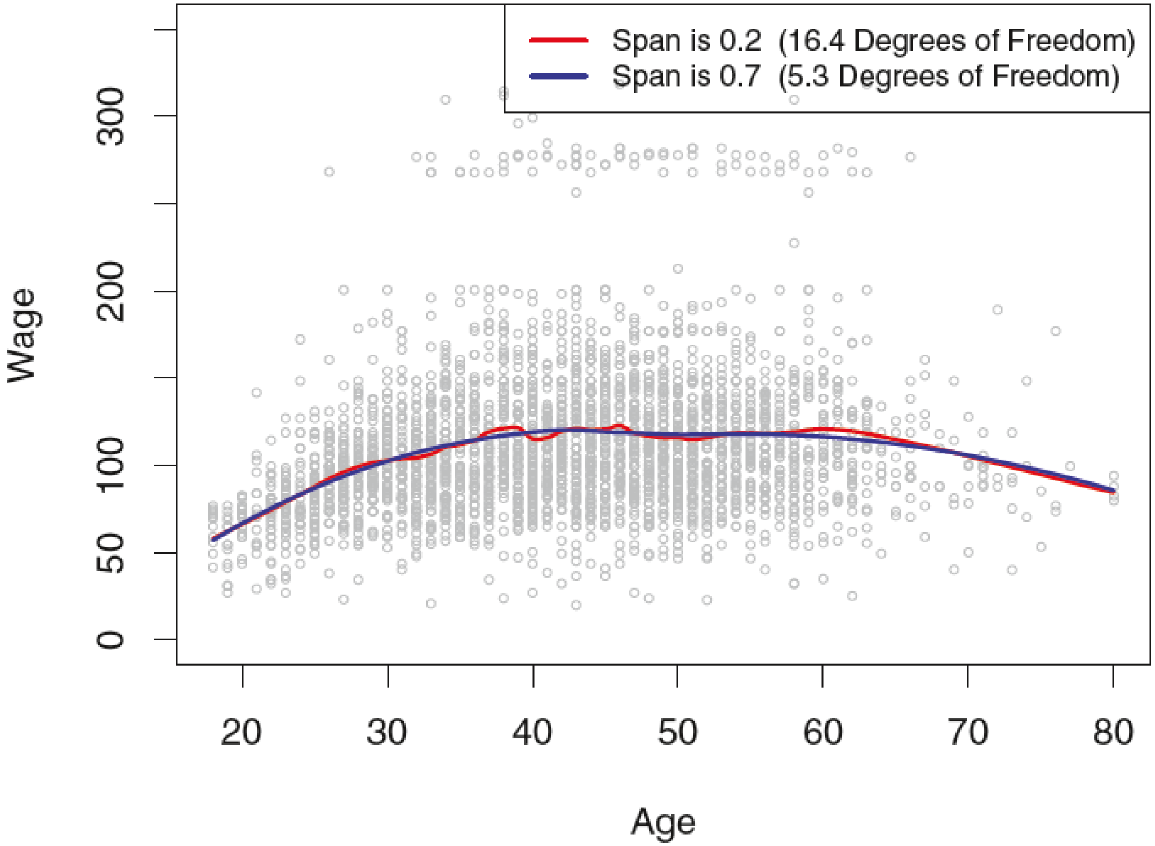
- アルゴリズム: $X = x_0$ での局所回帰
  1. 学習データから x_0 の最近傍のデータ点を k 個選択
  2. 各データ点に x_0 からの距離に応じて重み $K_{i0} = K(x_i, x_0)$ を割り当て
  3. 重み付き最小二乗法により回帰モデルを計算

     目的関数 $$\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$$

  4. 値 x_0 での推定値は $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$

局所回帰

- ハイパーパラメータ $s = k/n$（span）で有効自由度をコントロール



一般化加法モデル

- 多重線形回帰

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

各線形要素 $\beta_j x_{ij}$ を非線形関数 $f_j(x_{ij})$ で置き換え

- 一般化加法モデル（generalized additive models, GAMs）

$$y_i = \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \epsilon_i$$

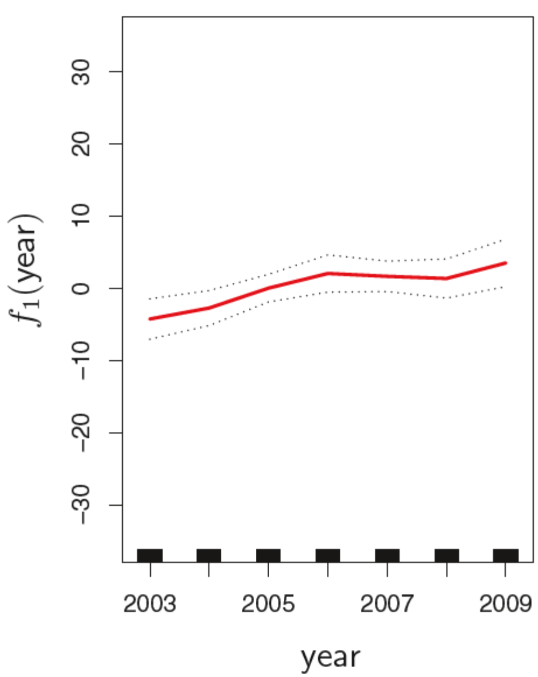
$$= \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i$$

一般化加法モデル

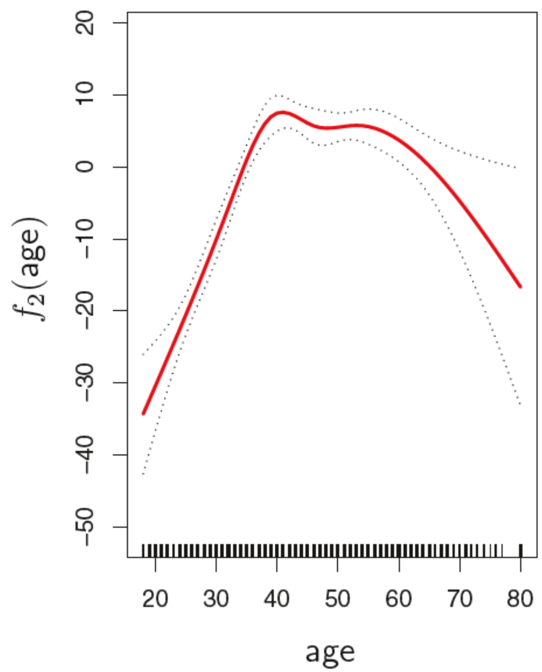
- 非線形関数 $f_j(\cdot)$ としてスプラインを含む様々な関数が使える

例）Wage データ

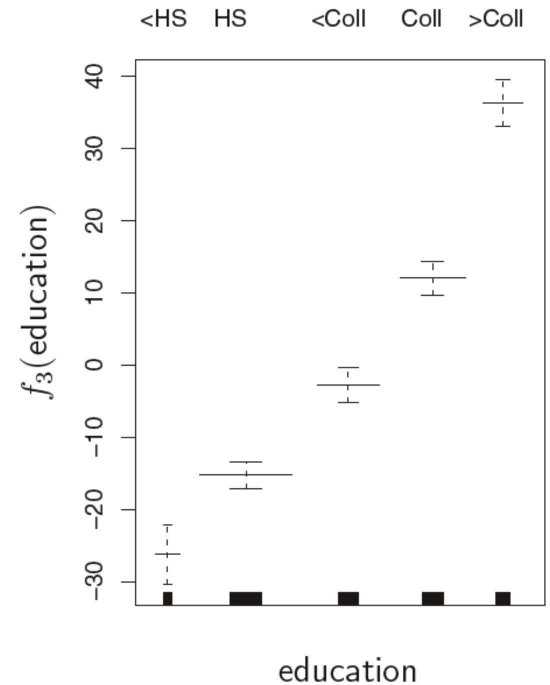
自然スプライン
（自由度＝4）



自然スプライン
（自由度＝5）



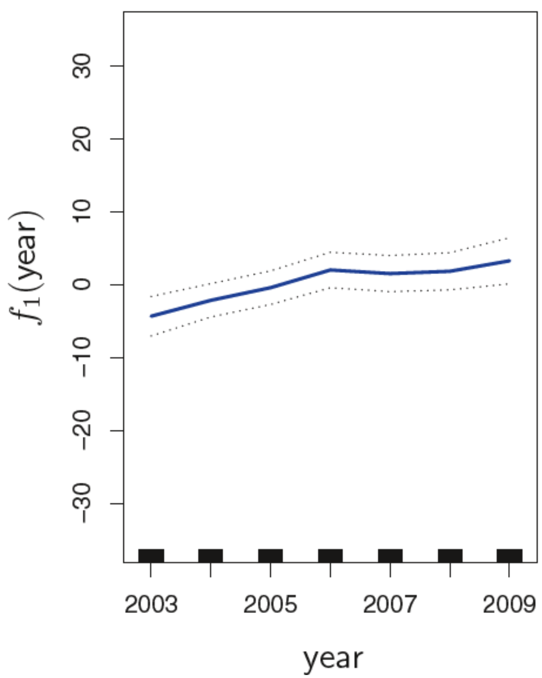
ステップ関数



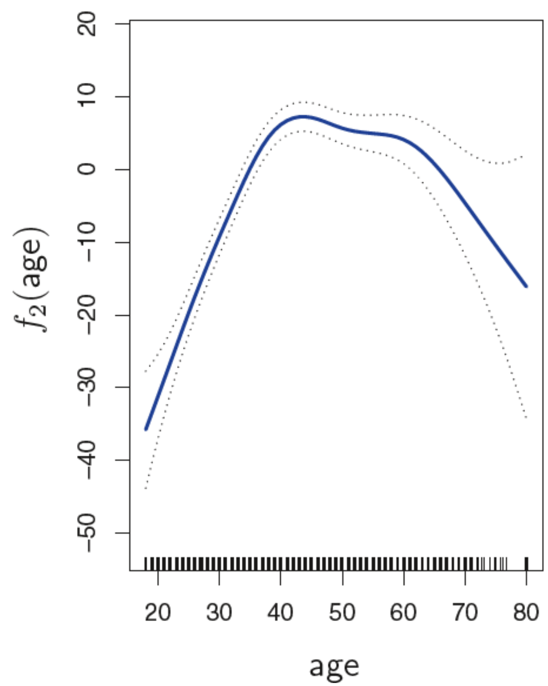
一般化加法モデル

例）Wage データ

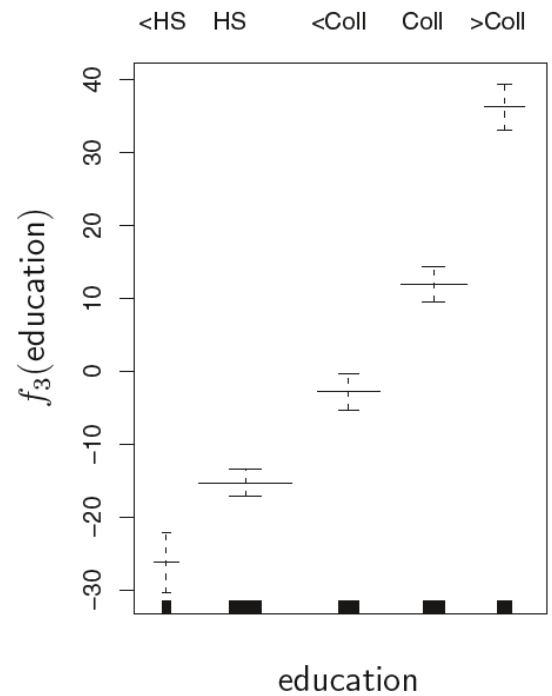
平滑化スプライン
（自由度＝4）



平滑化スプライン
（自由度＝5）



ステップ関数



一般化加法モデルの特徴

- 長所
  - 各入力変数に関して非線形なモデル化が可能
  - 出力の予測精度が高くなる可能性
  - 各変数の出力に対する影響の解釈が容易
- 短所
  - モデルが加法的であることによる表現力の限界

一般化加法モデルによる分類

- ロジスティック回帰

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p$$

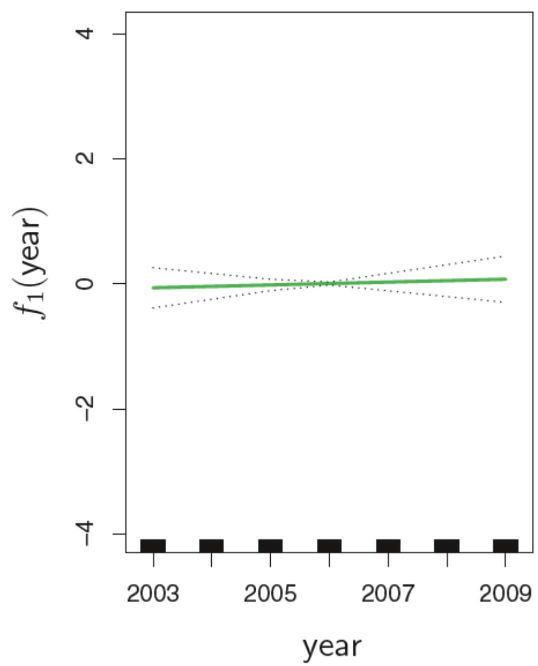
- 一般化加法モデル

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + f_1(X_1) + f_2(X_2) + \cdots + f_p\left(X_p\right)$$

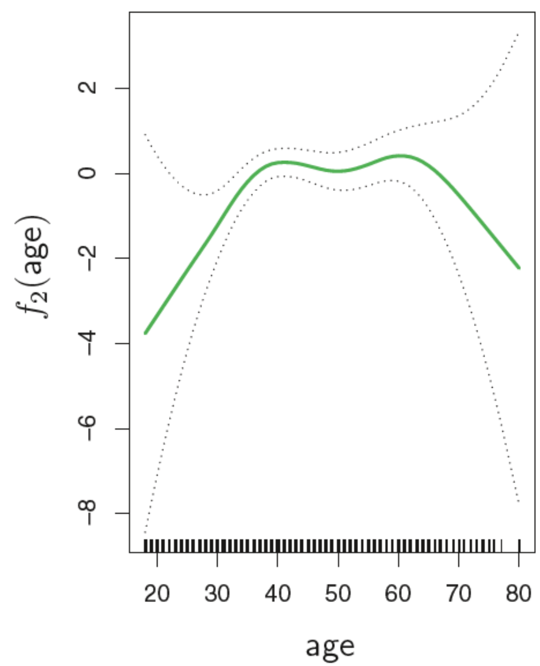
一般化加法モデルによる分類

- 例）Wage データ
  - wage > 250 であるかどうかを予測

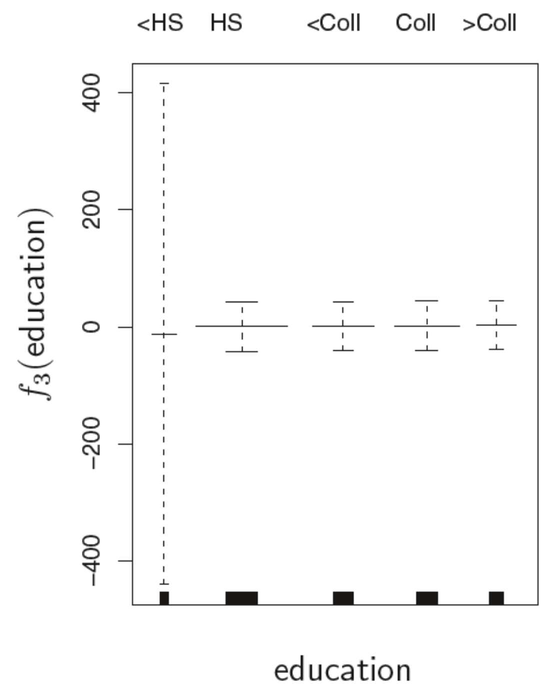
線形モデル



平滑化スプライン



ステップ関数



Python実習

- 一般化加法モデル
  - 自然スプラインの場合は基底関数行列を与えるだけ

```
>>> age_basis = dmatrix("cr(Wage.age, df=5)", {"Wage.age": Wage.age}, 
return_type='dataframe')
>>> year_basis = dmatrix("cr(Wage.year, df=4)", {"Wage.year": 
Wage.year}, return_type='dataframe').drop (['Intercept'], axis = 1)
>>> dummies = pd.get_dummies(Wage.education)
>>> dummies = dummies.drop(dummies.columns[0], axis = 1)
>>> x_all = pd.concat([age_basis, year_basis, dummies], axis = 1)
```

```
>>> gam1_fit = sm.OLS(Wage.wage, x_all).fit()
>>> gam1_fit.summary()
```

→　予測に使う変数を減らしたらどうなるか？