# 1　Rによるノンパラメトリック回帰

## 1.1　はじめに

Rはインターネット上で無料で入手できるプログラミング言語である。統計計算のための様々なオブジェクトが搭載されている。また、機能を拡張するためのパッケージも無料で利用することができる。また、Rを利用するパッケージやオブジェクトは、世界中で日々新たなものが製作され、その多くがインターネット上で無料で公開されている。これらを利用すれば、いろいろな統計計算を容易に実行できる。

ここでは、「gss」というパッケージを利用してノンパラメトリック回帰を行うための方法を紹介する。「gss」はパデュー大学の Gu 教授が作製したもので、Rを配布しているサイトで入手できる。「gss」の詳細な内容や依って立つ数学的な内容については、以下の本を参照していただきたい。

Chong Gu(2002): Smoothing spline ANOVA models. Springer Verlag. ISBN: 0-387-95353-1

## 1.2　予測変数が１つのときの平滑化スプライン

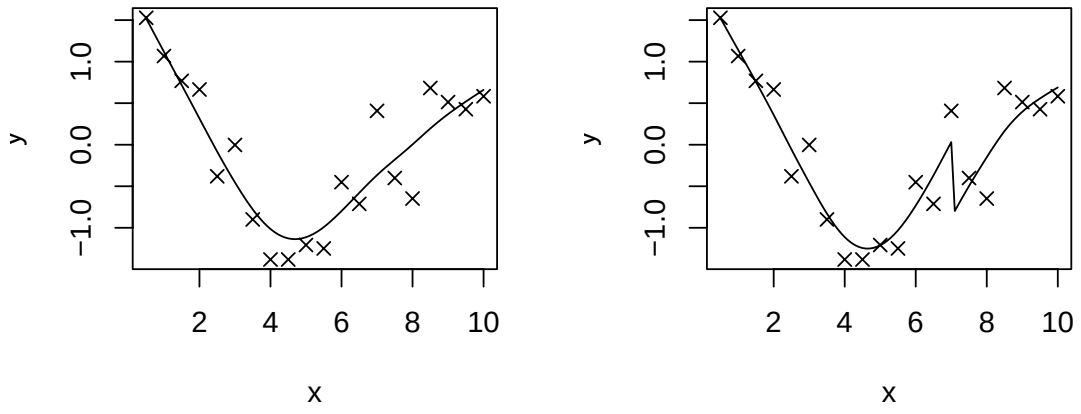


図 1.1: 平滑化スプラインによる平滑化。$GCV$ によって平滑化パラメータの値を最適化した。３次のスプラインを利用（左)。３次の自然スプラインを利用し、$x = 7$ のところを不連続点にする（右)。

予測変数が１つのとき、平滑化スプラインによる平滑化を行い、平滑化パラメータを $GCV$ を用いて最適化した結果が、図1.1である。図1.1（左）は、通常の３次の平滑化スプラインによる平滑化を行った結果である。$x = 7$ のところに不連続点を設けると、図1.1（右）が得られる。図1.1を得るためのオブジェクトの一例が以下のものである。

```
function ()
{
# (1)
    library(gss)
```

Rプログラミングの要点 (1)

**オブジェクト の実行とコマンド 行の実行では要領が少し違う。**

Rによる計算を行うための方法には２つある。１つは、function() で始まるオブジェクト（CやFortran におけるプログラムに相当する）を作製し、それを実行する方法である。もう一つは、コンソールから１つ１つのコマンド行を実行する方法である。実際に、キーボードから打ち込むのではなく、一続きのコマンド行を記録してから実行することもできる。この両方でほぼ同じコマンドが利用できることが、Rの特長の１つである。しかし、両者ではいくらか使い方が違う。例えば、attach() によるデータの設定は後者では推奨されるけれども、前者では推奨されない。

```
# (2)
    set.seed(101)
    nd <- 20
    xx <- seq(from = 0.5, to =10, length = nd)
    yy <- cos(pi*xx*0.2) + rnorm(nd, mean = 0, sd = 0.5)
    datag <- data.frame(x = xx, y= yy)
# (3)
    ex_seq(from = 0.5, to = 10, by = 0.1)
    data1 <- data.frame(x = ex)
    data2 <- data.frame(x = ex, partial = as.numeric(ex>7))
# (4)
    fit1 <- ssanova(y~x, data = datag, type = "cubic", method = "v")
    ey1 <- predict(fit1, newdata = data1)
# (5)
    fit2 <- ssanova(y~x, data = datag, type = "cubic", method = "v",
     partial = as.numeric(xx>7))
    ey2 <- predict(fit2, newdata = data2)
#  (6)
    par(mfrow = c(1, 2), mai = c(0.8, 0.8, 0.3, 0.3), oma = c(7, 3, 7, 3))
    plot(xx, yy, type = "n",  xlab = "x", ylab = "y")
    points(xx,yy, pch=4)
    lines(ex, ey1)
    plot(xx, yy, type = "n",  xlab = "x", ylab = "y")
    points(xx,yy, pch=4)
    lines(ex, ey2)
}
```

(1) ライブラリー「gss」を使用できるようにする。このオブジェクトを使用する前に、Rがインストールされているフォルダの中にある「library」というフォルダの中に「gss」というフォルダとその中身を入れておく必要がある。
(2) シミュレーションデータを作製し、それを datag というデータフレームにまとめる。
(3) 予測変数を求める点の値を与える。data1 は通常の平滑化スプラインのためのもので、data2 は、$x = 7$ のところに不連続点を設けるときのためのものである。
(4) ssanova()（「gss」に所収されているオブジェクト）を用いて、３次の平滑化スプラインによる平滑化を実行する。type = "linear"とすると、１次の平滑化スプラインが用いられる。平滑化パ

ラメータの最適化のためには $GCV$ を使う。`method = "m"`とすると、$GML$(Generalized Maximum likelihood) が利用される。
(5) $x = 7$ のところに不連続点を設けて、3次の平滑化スプラインを求める。
(6) (4)(5) で得られた結果をグラフに描く。

Rプログラミングの要点 (2)

**オブジェクトにデータを入れるときには、その形式に注意する。**

Rには統計計算や行列計算を行うための優れたオブジェクトがたくさん用意されている。それらを使う前に、データを適切な形式のものに加工する必要がある。`ssanova()`には、データフレームの形式をしたデータを入れる。そして、そのデータフレームの名前を`data=`で指定し、データフレームの中のデータセットの名前を使って回帰関数の形を指定する。統計計算を行うためのオブジェクトにはこの形式のものが多い。しかし、行列の形式をしたデータを必要とするものなどもある。

## 1.3 平滑化スプラインによる加法モデル

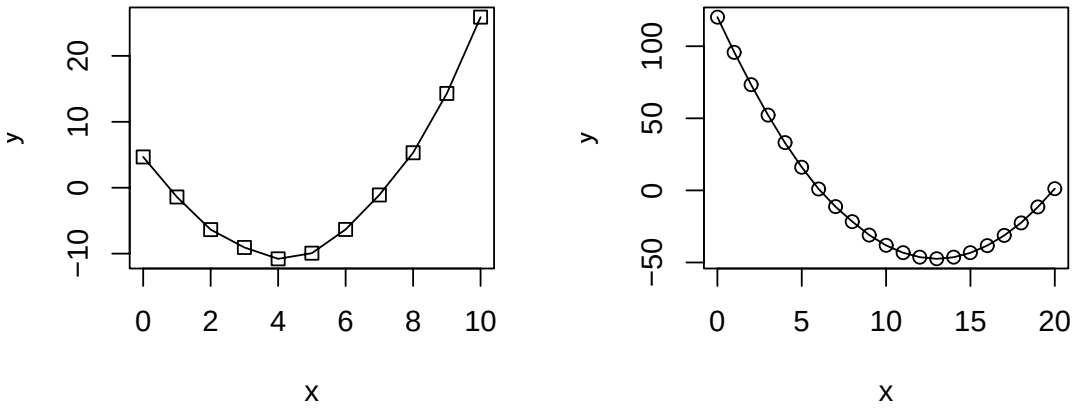


図 1.2: 平滑化スプラインによる加法モデルを用いて、$y = f_1(x_1) + f_2(x_2)$ という式に回帰したときに得られる $f_1(x_1)$（左）と $f_2(x_2)$（右)。

平滑化スプラインによる加法モデルとは、例えば以下のような回帰式を導くものである。

$$y = f_1(x_1) + f_2(x_2) \tag{1.1}$$

ここで、$x_1$ と $x_2$ が予測変数で、$y$ が目的変数である。$f_1(x_1)$ と $f_2(x_2)$ が平滑化スプラインによって作製された自然スプラインである。それぞれの平滑化スプラインにおける平滑化パラメータは $GCV$ を用いて最適化される。

この方法によって得られた $f_1(x_1)$ と $f_2(x_2)$ の例が、ぞれぞれ、図 1.2（左）と図 1.2（右）である。図 1.2 を得るためのオブジェクトの一例が以下のものである。

```
function()
{
```

Rプログラミングの要点 (3)

**rep() をうまく使えば規則的な数列が簡単に作製できる。**

rep() の代表的な使い方は以下のようなものである。

```
xx1 <- 1.7
yy1 <- rep(xx1, length = 4)
print(yy1)
xx2 <- c(2.5, 4.1)
yy2 <- rep(xx2, times = 3)
print(yy2)
```

以下が結果である。

```
1.7 1.7 1.7 1.7
2.5 4.1 2.5 4.1 2.5 4.1
```

以下のような使い方も様々な場面で利用できる。

```
xx3 <- c(7.2, 9.5)
yy3 <- rep(xx3, rep(3, length=2))
print(yy3)
```

以下が得られる。

```
7.2 7.2 7.2 9.5 9.5 9.5
```

```
# (1)
    library(gss)
    nd <- 100
    set.seed(2525)
    xx1 <- runif(nd, min = 0, max = 10)
    xx2 <- runif(nd, min = 0, max = 20)
    yy <- (xx1 - 4)^2 +(xx2 - 13)^2
     + rnorm(nd, mean = 0, sd = 1)
    data1 <- data.frame(x1=xx1, x2=xx2, y=yy)
# (2)
    nx1 <- 11
    nx2 <- 21
    grix1 <- seq(from = 0, to = 10, length = nx1)
    grix2 <- seq(from = 0, to = 20, length = nx2)
    data2 <- data.frame(x1 = rep(grix1, nx2),
     x2 = rep(grix2, rep(nx1, nx2) ) )
# (3)
    fit1 <- ssanova(y~x1+x2, data = data1, type="cubic", method="v", ext=0.2)
# (4)
    fx1 <- predict(fit1, newdata = data2, include="x1")
    fx1 <- fx1[1:nx1]
    fx2 <- predict(fit1, newdata = data2, include="x2")
    fx2 <- matrix(fx2, ncol=nx2)
    fx2 <- fx2[1,]
```

Rプログラミングの要点 (4)

**行列とは、列ベクトルを結合したもの、と理解する。**

行列を作成するためのコマンド matrix() は以下のように使う。

```
xx1 <- c(1, 2, 3, 4, 5, 6)
xx1mat <- matrix(xx1 , nrow=3)
print(xx1mat)
```

以下が結果である。

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

ベクトルの形で与えられたデータを使って、3つの要素からなる列ベクトルを2つ作製し、それらを繋ぎ合わせて行列を作成している。Rにおけるいろいろな操作において、ベクトルと行列の間での対応関係はこのようなものである。回帰式を作るときの計画行列を作るときには、それぞれのデータが行ベクトルになるので注意を要する。

```
# (5)
    par(mfrow = c(1, 2), mai = c(0.8, 0.8, 0.3, 0.3), oma = c(7, 3, 7, 3))
    plot(grix1, fx1, type = "n",  xlab = "x", ylab = "y")
    points(grix1, fx1, pch=0)
    lines(grix1, fx1)
    plot(grix2, fx2, type = "n",  xlab = "x", ylab = "y")
    points(grix2, fx2, pch=1)
    lines(grix2, fx2)
}
```

(1) ライブラリー「fields」を利用できるようにした後、シミュレーションデータを作製する。
(2) 推定値を求める格子点の、$x_1$ 座標を grix1、$x_2$ 座標を grix2 とする。格子点を表すデータフレームを data2 とする。data2 においては、$x_1$ 座標の値の入れ方と $x_2$ 座標の値の入れ方が異なる。
(3) ssanova() を用いて加法モデルを作製する。ext = 0.2 は、データにおける予測変数が存在する範囲の両側を左右に 20%ずつ拡張した領域における推定値を求めることが可能になるようにすることを指定している。
(4) predict() を用いてそれぞれの変換関数を求める。predict() からの出力は、それぞれの格子点における値を表す行列をベクトルに変換したものになる。
(5) (4) で得られた結果をグラフに描く。

## 1.4 薄板平滑化スプライン

薄板平滑化スプラインとは、例えば以下のような回帰式を導くものである。

$$y = f(x_1, x_2) \tag{1.2}$$

ここで、$x_1$ と $x_2$ が予測変数で、$y$ が目的変数である。$f(x_1, x_2)$ は、薄板平滑化スプラインによって得られた、薄板スプラインと呼ばれる回帰関数である。薄板スプラインとは自然スプラインを予測変数が2つ以上の場合に拡張した回帰関数である。加法モデルよりも一般性が高い。平滑化スプライン

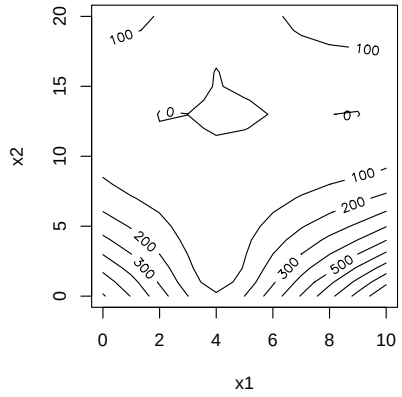


図 1.3: 薄板平滑化スプラインを用いて得られた回帰曲面

と同様に、平滑化パラメータによって推定値の滑らかさを調整する。ここでは、その値を $GCV$ を用いて最適化する。

マニュアルには、ssanova()を薄板平滑化スプラインを用いた平滑化だけを実行することができなので、ここでは「fields」というパッケージ（これも、Rの本体が配布されているサイトで無料で配布されている）を利用した。この方法によって得られた $f_1(x_1, x_2)$ の例が、図 1.3 である。図 1.3 を得るためのオブジェクトの一例が以下のものである。

```
function()
{
# (1)
    library(fields)
    nd <- 100
    set.seed(2525)
    xx1 <- runif(nd, min = 0, max = 10)
    xx2 <- runif(nd, min = 0, max = 20)
    yy <- abs(xx1 - 4) *(xx2 - 13)^2 + rnorm(nd, mean = 0, sd = 1)
    data1_data.frame(x1=xx1, x2=xx2,  y=yy)
# (2)
    nx1_11
    nx2_21
    grix1_seq(from = 0, to = 10, length = nx1)
    grix2_seq(from = 0, to = 20, length = nx2)
    grix12_expand.grid(grix1, grix2)
# (3)
    fit1_Tps(cbind(xx1, xx2), yy)
    fx12_predict(fit1, grix12 )
    fx12_matrix(fx12, ncol = nx2)
# (4)
    par(mfrow = c(1, 1), mai = c(0.5, 0.5, 0.5, 0.5), oma = c(8, 8, 8, 8))
    contour(grix1, grix2, fx12, xlab = "x1", ylab = "x2")
}
```

Rプログラミングの要点 (5)

**格子点の座標を求めるときは、`expand.grid()` を使う。**

`expand.grid()` を利用すれば以下のようにして格子点の値が得られる。

```
xx1 <- c(11, 22)
yy1 <- c(55, 66, 77)
grid1 <- expand.grid(xx1, yy1)
print(grid1)
```

以下が結果である。

```
  Var1 Var2
1   11   55
2   22   55
3   11   66
4   22   66
5   11   77
6   22   77
```

`grid1` はデータフレームの形式をしている。その中のデータセットの名前を代えるには以下のようにする。

```
colnames(grid1) <- c("x-axis", "y-axis")
print(grid1)
```

以下のものが得られる。

```
  x-axis  y-axis
1     11     55
2     22     55
3     11     66
4     22     66
5     11     77
6     22     77
```

(1) ライブラリー「fields」を利用できるようにした後、シミュレーションデータを作製する。
(2) 推定値を求める格子点の座標を求める。
(3) `Tps()` を用いて薄板平滑化スプラインによる平滑化を実行する。`Tps()` の中で `lambda=`とすると、平滑化パラメータの値を指定することができる。ここでは、`lambda=`が書かれていないので、$GCV$ を用いた最適化が行われる。その結果を用いて、格子点における推定値を求める。
(4) 推定値の等高線を描く。

## 1.5 Smoothing Spline ANOVA

加法モデルや薄板平滑化スプラインよりも柔軟な形式の回帰式として以下のようなものが考えられる。

$$y = f_1(x_1) + f_2(x_2) + f_3(x_1, x_2) \tag{1.3}$$

ここで、$f_1(x_1)$ と $f_2(x_2)$ は平滑化スプラインによって作製された自然スプラインで、$f_3(x_1, x_2)$ は薄板平滑化スプラインである。この方法は、古典的な回帰分析における分散分析 (ANOVA, ANalysis Of

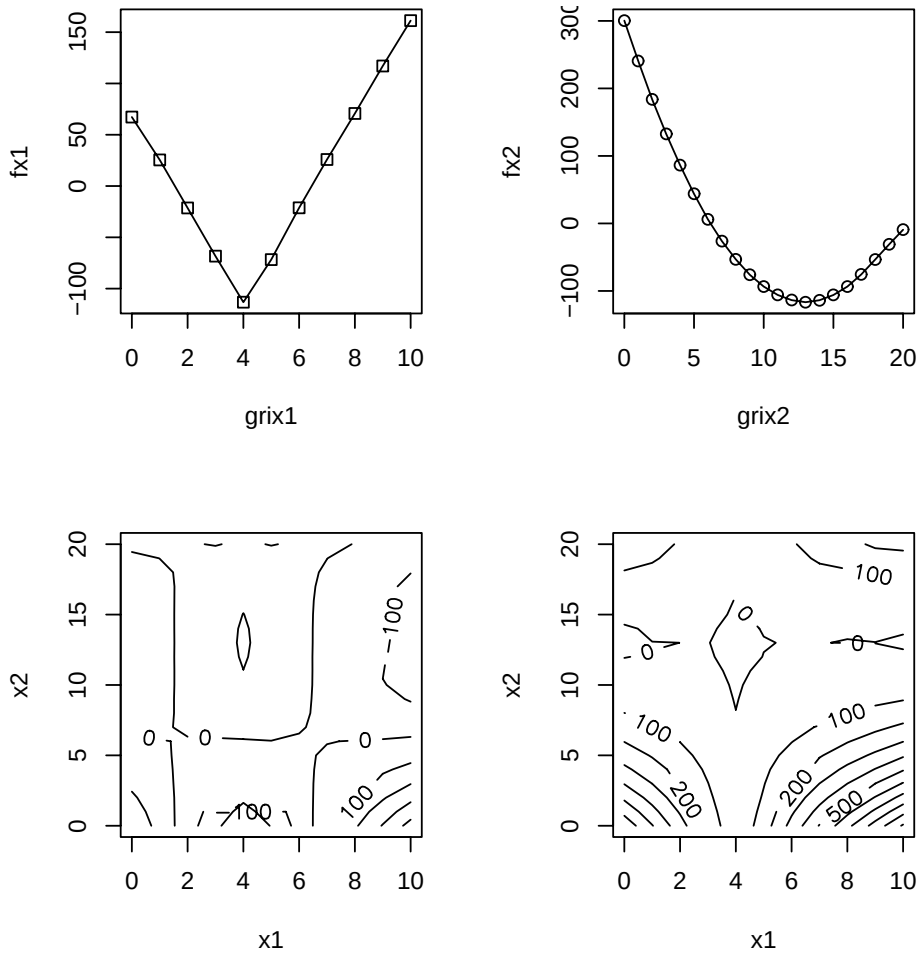


図 1.4: $y = f_1(x_1) + f_2(x_2) + f_3(x_1, x_2)$ という形の回帰式を用いた結果。$f_1(x_1)$（左上）。$f_2(x_2)$（右上）。$f_3(x_1, x_2)$（左下）。$f_1(x_1) + f_2(x_2) + f_3(x_1, x_2)$（右下）。

Rプログラミングの要点 (6)

**contour()（等高線の作製）のデータには行列を使う。**

contour() のデータは、第１予測変数の座標（ベクトル）、第２予測変数の座標（ベクトル）、目的変数の値（行列）の順に列挙する。目的変数の値を示す行列の列ベクトルが第１予測変数に対応する。すなわち、以下のように用いる。

```
xx1 <- c(11, 32)
yy1 <- c(55, 76, 108, 149)
zz1 <- matrix(c(1, 2.2, 3, 4.5, 9.9, 12.3, 22.3, 15.8), nrow=2)
contour(xx1, yy1, zz1)
```

目的変数の値を表す行列（上の例では zz1）の値を表示すると、第１予測変数が縦方向に相当する。一方、等高線においては、第１予測変数が横軸に相当する。

Variance）を平滑化スプラインと薄板平滑化スプラインを用いて拡張したものと見なすことができることから、Smoothing Spline ANOVA と呼ばれる。

この方法を用いて得られた回帰式の一例が図 1.4 である。そのために用いたオブジェクトが以下のものである。

```
function()
{
#  (1)
    library(gss)
    nd <- 100
    set.seed(2525)
    xx1 <- runif(nd, min = 0, max = 10)
    xx2 <- runif(nd, min = 0, max = 20)
    yy <- abs(xx1 - 4) *(xx2 - 13)^2 + rnorm(nd, mean = 0, sd = 1)
    data1 <- data.frame(x1=xx1, x2=xx2, y=yy)
#  (2)
    nx1 <- 11
    nx2 <- 21
    grix1 <- seq(from = 0, to = 10, length = nx1)
    grix2 <- seq(from = 0, to = 20, length = nx2)
    data2 <- data.frame(x1 = rep(grix1, nx2), x2 = rep(grix2, rep(nx1, nx2) ) )
#  (3)
    fit1 <- ssanova(y~x1*x2, data = data1, type="tp", method="v")
    fx1 <- predict(fit1, newdata = data2, include="x1")
    fx1 <- fx1[1:11]
    fx2 <- predict(fit1, newdata = data2, include="x2")
    fx2 <- matrix(fx2, ncol=nx2)
    fx2 <- fx2[1, ]
    fx3 <- predict(fit1, newdata = data2, include="x1:x2")
    fx3 <- matrix(fx3, ncol=nx2)
    fx4 <- predict(fit1, newdata = data2, include=c("1", "x1", "x2", "x1:x2"))
    fx4 <- matrix(fx4, ncol=nx2)
```

```
# (4)
    par(mfrow = c(2, 2), mai = c(0.5, 0.5, 0.3, 0.3), oma = c(1, 1, 1, 1))
    plot(grix1, fx1, type="n")
    points(grix1, fx1, pch=0)
    lines(grix1, fx1)
    plot(grix2, fx2, type="n")
    points(grix2, fx2, pch=1)
    lines(grix2, fx2)
    contour(grix1, grix2, fx3, xlab = "x1", ylab = "x2")
    contour(grix1, grix2, fx4, xlab = "x1", ylab = "x2")
}
```

(1) ライブラリー「gss」を利用できるようにした後、シミュレーションデータを作製する。
(2) 推定値を求める格子点の座標を求める。
(3) ssanova() を用いて、Smoothing Spline ANOVA による回帰を行う。その結果を用いて、格子点における、$f_1(x_1)$、$f_2(x_2)$、$f_3(x_1, x_2)$ 、$f_1(x_1) + f_2(x_2) + f_3(x_1, x_2)$ の値を求める。
(4) (3) で求めた推定値を図示する。

## 1.6 ポアソン回帰を用いた、ヒストグラムの平滑化

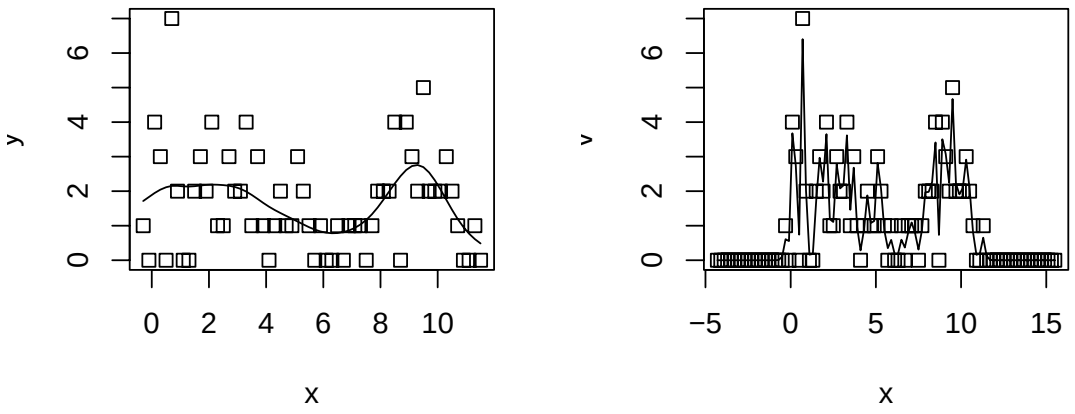


図 1.5: ポアソン分布を仮定した平滑化スプラインによる、ヒストグラムの平滑化。平滑化パラメータを最適化するために $GCV$ を用いた。データが存在する領域とヒストグラムを作製する領域をほぼ一致させたとき（左)。データが存在する領域よりもヒストグラムを作製する領域をかなり広く設定した場合（右)。

ヒストグラムは、端点の位置と区切り幅の大きさによって形状が大きく異なってしまうことがある。むしろ、区切り幅が小さいヒストグラムを作製して、それを平滑化した方が、分布の様子を把握するために有効である。ヒストグラムの平滑化のための方法の一つとして、ポアソン分布を仮定した平滑化スプラインを用いることができる。ポアソン分布を仮定した回帰をポアソン回帰と総称することがある。この方法でヒストグラムを行った結果の例を図 1.5 に示す。平滑化パラメータを最適化するた

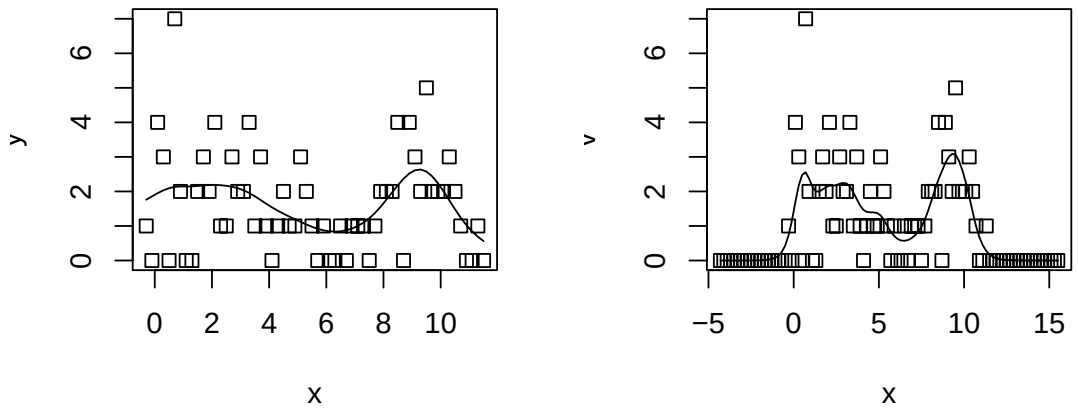


図 1.6: ポアソン分布を仮定した平滑化スプラインによる、ヒストグラムの平滑化。平滑化パラメータを最適化するために $GML$ を用いた。データが存在する領域とヒストグラムを作製する領域をほぼ一致させたとき（左)。データが存在する領域よりもヒストグラムを作製する領域をかなり広く設定した場合（右)。

めには、$GCV$ を用いている。図 1.5（左）は、妥当な平滑化が行われた結果と見なすことができる。しかし、$GCV$ を用いて平滑化パラメータを最適化した場合、ヒストグラムを作製する領域をデータが存在する領域よりもかなり広くとると、凸凹が大きすぎる推定値が得られてしまうことが多い（図 1.5（右))。そのときは、平滑化パラメータを最適化するために、$GCV$ に代えて、$GML$ を用いると、この傾向が軽減されることがある（図 1.6)。

図 1.5（左）を作製するために作製したオブジェクトは以下のものである。

```
function()
{
# (1)
    library(gss)
    set.seed(2523)
    yy <- c(rnorm(60, mean=3, sd=2), rnorm(40 , mean=9, sd=1))
    nd <- length(yy)
# (2)
    br1 <- seq(from = floor(min(yy)*5)/5, to = ceiling(max(yy)*5)/5+0.2, by = 0.2)
    dbr1 <- br1[2] - br1[1]
    yyhist <- floor(yy/dbr1)*dbr1 + dbr1*0.5
    hist1_hist(yyhist, breaks = br1, include.lowest = T, plot=F)
    counts1_hist1$counts
# (3)
    midp1 <- br1[1:(length(br1)-1)]+(br1[2]-br1[1])*0.5
    data1 <- data.frame(x=midp1, y=counts1)
    fit1 <- gssanova(y~x, data=data1, family="poisson", method="v")
    data2 <- data.frame(x=midp1)
    ey <- exp(predict(fit1, newdata=data2))
```

Rプログラミングの要点 (7)

**普通の意味でのヒストグラムを作製するためには、hist() の前に準備が必要になる。**

ヒストグラムの頻度の計算とグラフの描画のために hist() が用意されている。しかし、hist() における頻度の定義を通常のものと比べると、等号の扱い方が異なっている。そのため、以下の例のように、yy というデータのヒストグラムを求める際に、yy を離散化して yyhist を作って、それを hist() に入れる。

```
    yy <- c(0.4999999999999, 0.5, 0.54, 2, 2, 2.1, 3, 3.5)
    br1 <- seq(from = 0, to = 4, by = 0.5)
    dbr1 <- br1[2]-br1[1]
    yyhist <- floor(yy/dbr1)*dbr1 + dbr1*0.5
    print(yyhist)
    hist1 <- hist(yyhist, breaks = br1, plot = F)
    print(hist1$counts)
```

以下が結果である。

```
    0.25 0.75 0.75 2.25 2.25 2.25 3.25 3.75
    1 2 0 0 3 0 1 1
```

```
# (4)
    par(mfrow = c(1, 2), mai = c(0.8, 0.8, 0.3, 0.3), oma = c(7, 3, 7, 3))
    plot(midp1, counts1, type = "n",  xlab = "x", ylab = "y")
    points(midp1, counts1, pch=0)
    lines(midp1, ey)
}
```

(1) ライブラリー「gss」を利用できるようにした後、シミュレーションデータを作製する。
(2) 通常のヒストグラムの定義に従ってヒストグラムを作製し、頻度を counts1 とする。
(3) midp1 を予測変数、counts1 を目的変数とする、ポアソン分布を仮定した平滑化スプラインの計算を行う。そして、midp1 における推定値を求める。predict() で得られる値は、推定値を連結関数を使って変換したものなので、連結関数の逆関数を用いて推定値を求めている。
(4) (3) で求めた推定値を図示する。

## 1.7 平滑化スプラインによるロジスティック曲線

スカラーの予測変数に対して、「勝ち」「負け」のような２値データが目的変数として与えられているとき、通常のロジスティック回帰とは、２項分布を仮定してロジスティック曲線をあてはめるものである。これに対して、２項分布を仮定して、平滑化スプラインによる自然スプラインをあてはめる方が、より多様なデータに対して優れた結果を生み出すことが期待できる。そうした方法で、行った回帰の結果が一例が図 1.7 である。図 1.7 を得るために以下のオブジェクトを利用した。

```
function()
{
# (1)
    library(gss)
```

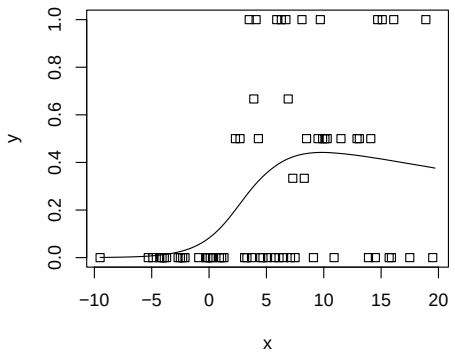


図 1.7: ２項分布を仮定した、平滑化スプラインによる平滑化。平滑化パラメータは $GCV$ を用いて最適化した。

```
    set.seed(2523)
    yya <- rnorm(30, mean=9.3, sd=4.8)
    nda <- length(yya)
    yyb <- c(rnorm(60, mean=3.2, sd=5), rnorm(10, mean=14.2, sd=3.1))
    ndb <- length(yyb)
# (2)
    br1 <- seq(from = floor(min(yya,yyb)*5)/5,
     to = ceiling(max(yya,yyb)*5)/5+0.2, by = 0.2)
    dbr1 <- br1[2] - br1[1]
    yyhist <- floor(yya/dbr1)*dbr1 + dbr1*0.5
    hist1 <- hist(yyhist, breaks = br1, include.lowest = T, plot=F)
    counts1a <- hist1$counts
    yyhist <- floor(yyb/dbr1)*dbr1 + dbr1*0.5
    hist1 <- hist(yyhist, breaks = br1, include.lowest = T, plot=F)
    counts1b <- hist1$counts
# (3)
    midp1 <- br1[1:(length(br1)-1)]+(br1[2]-br1[1])*0.5
    ratio1 <- counts1a/(counts1a+counts1b)
    data1 <- data.frame(x=midp1, y=ratio1 )
    fit1 <- gssanova(y~x, data=data1, family="binomial",
     weights=counts1a+counts1b, method="v")
    data2 <- data.frame(x=midp1)
    ey <- 1-1/(1+exp(predict(fit1, newdata=data2)))
# (4)
   par(mfrow = c(1, 1), mai = c(0.8, 0.8, 0.3, 0.3), oma = c(7, 3, 7, 3))
    plot(midp1, ratio1, type = "n",  xlab = "x", ylab = "y")
    points(midp1, ratio1, pch=0)
    lines(midp1, ey)
}
```

Rプログラミングの要点 (8)

**切りのいい数値を得るためには、floor()、ceiling()、pretty() を使う。**

数値を要素とするベクトルから切りのいい数値を得るためには、以下のような方法がある。

```
xx <- c(-81.2, -0.011, 0, 0.499, 0.5, 0.54, 2, 2.1, 3, 192.22)
print(ceiling(xx))
print(floor(xx))
print(pretty(xx))
```

以下が結果である。

```
-81   0   0   1   1   1   2   3   3 193
-82  -1   0   0   0   0   2   2   3 192
-100  -50    0   50  100  150  200
```

5 刻みの値を値を得るには以下のようにする。

```
print(floor(xx*0.2)*5)
print(ceiling(xx*0.2)*5)
```

以下のものが得られる。

```
-85  -5   0   0   0   0   0   0   0 190
-80   0   0   5   5   5   5   5   5 195
```

(1) ライブラリー「gss」を利用できるようにした後、シミュレーションデータを作製する。例えば、yya が、勝ったときの予測変数の値を示し、yya が、負けたときの予測変数の値を示す。
(2) yya と yyb のヒストグラムの頻度を、それぞれ、counts1a、counts1b とする。
(3) gssanova() を使って、counts1a と counts1b の和に対する counts1a の割合を表す、スプライン関数を求める。そして、predict() を使って、それぞれの区切りの中央における推定値を求める。predict() で得られる値は、推定値を連結関数を使って変換したものなので、連結関数の逆関数を用いて推定値を求めている。
(4) (3) で求めた推定値を図示する。

## 1.8 ノンパラメトリック確率密度関数の推定

Rプログラミングの要点 (9)

**入力引数を省略したときに設定される値や設定に注意する。**

RやS言語におけるオブジェクトにおいては、入力引数を省略した場合に特定の値や設定を与えるようにすることができる。そうしたオブジェクトを利用する場合には、省略した場合の値や設定がどのようなものでどのような意味を持つものであるかを知っておく必要がある。例えば、「gss」に所収されている ssden() においえては alpha=を設定しないことは、alpha=1.4 と設定することと同じ意味を持つ。その場合は、1.4 という値を利用した modified Cross-Validation を実行する。通常の Cross-Validation を実行するためには、alpha=1 としなければならない。alpha=を与えなければ通常の Cross-Validation が行われると考えてしまうと、思わく通りの計算が行われない。

平滑化スプラインを用いてノンパラメトリック確率密度関数の推定を行うこともできる。その例が、図 1.8 である。平滑化パラメータを用いるための手段として、modified Cross-Validation を用いている。このときは、推定値を求める範囲を広げても推定値の様子は殆ど変化していない。この点で、図

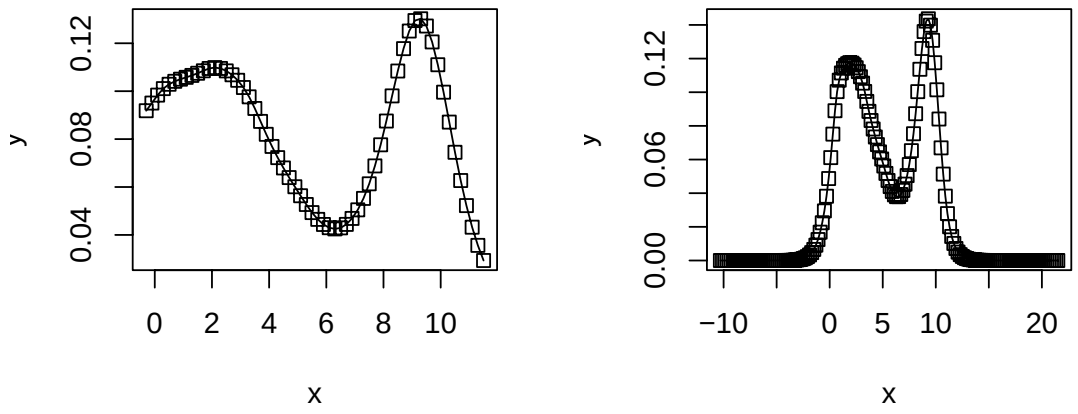


図 1.8: 平滑化スプラインを用いてノンパラメトリック確率密度関数の推定。推定値を求める範囲がデータの存在する範囲とほぼ同じ（左)。推定値を求める範囲を広げたとき（右)。

1.6 とは大きく異なる。図 1.8（左）を得るために用いたオブジェクトは以下のものである。

```
function()
{
# (1)
    library(gss)
    set.seed(2523)
    xx <- c(rnorm(60, mean=3, sd=2), rnorm(40 , mean=9, sd=1))
    nd <- length(xx)
# (2)
    br1 <- seq(from = floor(min(xx)*5)/5, to = ceiling(max(xx)*5)/5+0.2, by = 0.2)
    data1 <- data.frame(x=xx)
    fit1 <- ssden(~x, data = data1, alpha=1.4, domain=
     data.frame( x = c(br1[1], br1[length(br1)] )  ) )
# (3)
    midp1 <- br1[1:(length(br1)-1)]+(br1[2]-br1[1])*0.5
    data2 <- data.frame(x = midp1)
    ey <- dssden(fit1, data2)
# (4)
    par(mfrow = c(1, 2), mai = c(0.8, 0.8, 0.3, 0.3), oma = c(7, 3, 7, 3))
    plot(midp1, ey, type = "n",  xlab = "x", ylab = "y")
    points(midp1, ey, pch=0)
    lines(midp1, ey)
}
```

(1) ライブラリー「gss」を利用できるようにした後、シミュレーションデータを作製する。
(2) ssden()（「gss」に所収されているオブジェクト）を用いてノンパラメトリック確率密度関数を推定する。domain=によって推定値を求める領域を指定する。alpha=1 とすると、通常のクロスバリ

デーションによる平滑化パラメータの最適化を行われる。それ以外の値を指定すると、modified Cross-Validation が利用される。`alpha=1.4` くらいの値を用いることが推奨されているため、`alpha=`を指定しない場合には、`alpha=1.4` を指定したことになる。自然スプラインの節点は、`ssden()` の中に特に指定しない場合、データ点の中から $max(30, 10n^{2/9})$（$n$ はデータ数）の数のものが選択される。
(3) `dssden()`（「gss」に所収されているオブジェクト）を使って、`midp1` のそれぞれの点における推定値を求める。
(4) (3) で求めた推定値を図示する。

## 1.9 加法モデルの形式のノンパラメトリック確率密度関数の推定

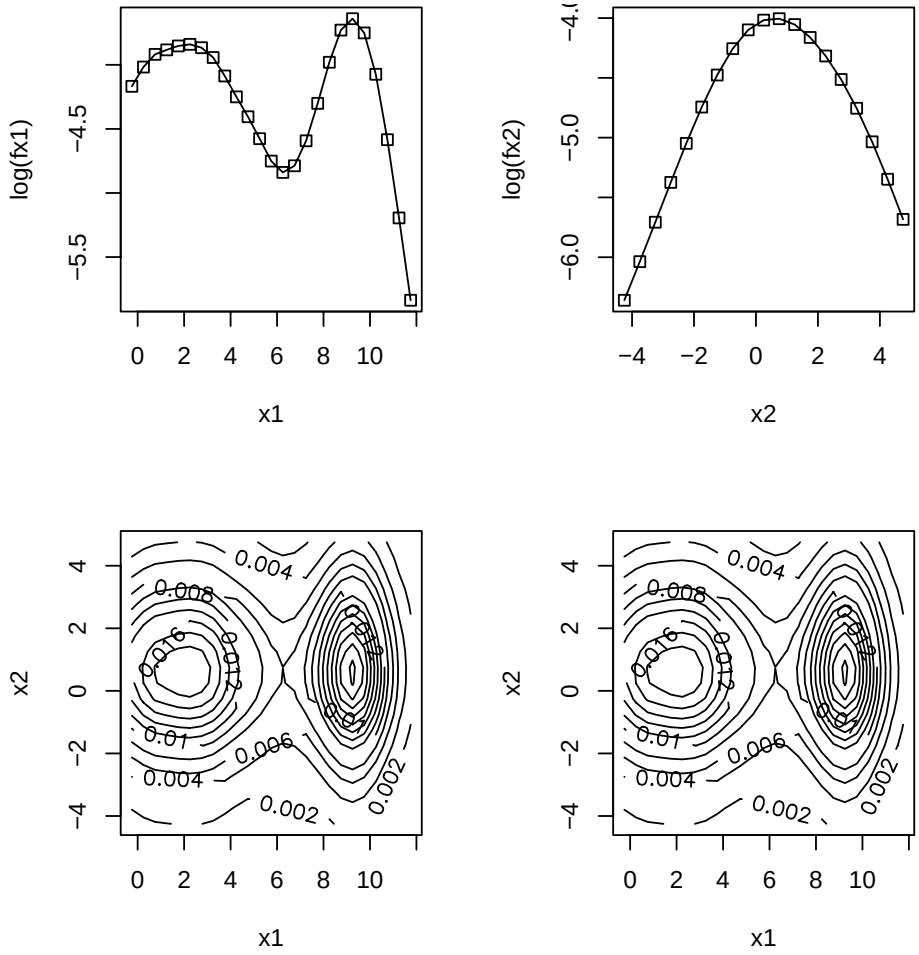


図 1.9: 式 (1.4) の形の回帰式を用いて求めた、変量のノンパラメトリック確率密度関数。$f_1(x_1)$（実際にはこの値に定数を加えたものが用いられる）（左上）。$f_2(x_2)$（右上）。$\exp(f_1(x_1) + f_2(x_2))$（左下）。$f_1(x_1)$ と $f_2(x_2)$ を使って求めた、$\exp(f_1(x_1) + f_2(x_2))$（右下）。

2 変量のノンパラメトリック確率密度関数として以下のような回帰式を用いることが考えられる。

$$\log(y) = f_1(x_1) + f_2(x_2) \tag{1.4}$$

ここで、$y$ がノンパラメトリック確率密度関数の値の推定値で、$f_1(x_1)$ と $f_2(x_2)$ が、それぞれ、$x_1$ と

Rプログラミングの要点 (10)

**加法モデルのそれぞれの項の値を使って格子点での値を求めるとき、sapply()と<<-が利用できる。**

２つの予測変数を用いた加法モデルにおいて、$x_1$ として４つの値を与えたときの $f_1(x_1)$ の値が $-2, 5, 7, 9$ で、$x_2$ として３つの値を与えたときの特定の値を与えたときの $f_2(x_2)$ の値が $-0.004, -0.00003, 0.009$ とする。これらの予測変数の値の組み合わせからなる１２の格子点における値を求めるには、以下のようにすると簡単である。

```
fx1 <- c(-2, 5, 7, 9)
fx2 <- c(-0.004, -0.00003, 0.009)
yy <- matrix(rep(0, length=12), nrow=4)
sapply(c(1:3), function(i) yy[,i] <<- yy[,i] + fx1)
sapply(c(1:4), function(i) yy[i,] <<- yy[i,] + fx2)
print(yy)
```

以下のようになる。

```
       [,1]     [,2]   [,3]
[1,] -2.004  -2.00003  -1.991
[2,]  4.996  4.99997   5.009
[3,]  6.996  6.99997   7.009
[4,]  8.996  8.99997   9.009
```

ただし、S言語においては「<<-」の定義が異なるため、この方法は利用できない。

$x_2$ を予測変数とするスプライン関数である。しかし、実際には、$y$ の値をこの領域で積分したものが 1 になるように $f_1(x_1)$ と $f_2(x_2)$ の何れかに定数を加えたものが使われている。この形式のノンパラメトリック確率密度関数を求めた例が、図 1.9 である。図 1.9（左上）が $f_1(x_1)$ で、図 1.9（右上）が $f_2(x_2)$、である。図 1.9（左下）と図 1.9（右下）はいずれも $y$ で、同じものである。ただし、図 1.9（左下）は、ssden()からの出力をそのまま使って求めた推定値で、図 1.9（右下）は、図 1.9（左下）の値の一部分を使って $f_1(x_1)$ と $f_2(x_2)$ を求め、式 (1.4) に代入して得られた $y$ である。図 1.9（左下）と図 1.9（右下）が一致することで、ssden()からの出力が式 (1.4) に従ったものであることが確認できた。

```
function()
{
# (1)
    library(gss)
    set.seed(2523)
    xx1 <- c(rnorm(60, mean=3, sd=2), rnorm(40 , mean=9, sd=1))
    xx2 <- c(rnorm(100, mean=1, sd=2))
# (2)
    br1 <- seq(from = floor(min(xx1)*2)/2, to = ceiling(max(xx1)*2)/2+0.5, by = 0.5)
    br2 <- seq(from = floor(min(xx2)*2)/2, to = ceiling(max(xx2)*2)/2+0.5, by = 0.5)
    data1 <- data.frame(x1 = xx1, x2 = xx2)
    fit1 <- ssden(~x1+x2, data=data1, type="cubic", alpha=1.4,
     domain=data.frame( x1=c(br1[1], br1[length(br1)] ),
```

```
     x2=c(br2[1], br2[length(br2)] )) )
# (3)
    midp1 <- br1[1:(length(br1)-1)]+(br1[2]-br1[1])*0.5
    midp2 <- br2[1:(length(br2)-1)]+(br2[2]-br2[1])*0.5
    midpg <- expand.grid(midp1, midp2)
    data2 <- data.frame(x1=midpg[,1], x2=midpg[,2])
    ey <- dssden(fit1, data2)
    ey <- matrix(ey, ncol=length(midp2))
# (4)
    nx1 <- length(midp1)
    nx2 <- length(midp2)
    ey2 <- matrix( rep(0, length=nx1*nx2), ncol=nx2  )
    sapply(c(1:nx2), function(i) ey2[,i]<<-ey2[,i]+log(ey[,10]) )
    sapply(c(1:nx1), function(i) ey2[i,]<<-ey2[i,]+log(ey[2,]) )
    ey2 <- ey2- -4.017161
    ey2 <- exp(ey2)
# (5)
    par(mfrow = c(2, 2), mai = c(0.5, 0.5, 0.5, 0.5),oma = c(1, 1, 1, 1))
    plot(midp1, log(ey[,10]), type = "n",  xlab = "x1", ylab = "log(fx1)")
    points(midp1, log(ey[,10]),   pch=0)
    lines(midp1, log(ey[,10]))
    plot(midp2, log(ey[2,]), type = "n",  xlab = "x2", ylab = "log(fx2)")
    points(midp2, log(ey[2,]),  pch=0)
    lines(midp2, log(ey[2,]))
    contour(midp1, midp2, ey,  xlab = "x1", ylab = "x2")
# (6)
    contour(midp1, midp2, ey2, xlab = "x1", ylab = "x2")
}
```

(1) ライブラリー「gss」を利用できるようにした後、シミュレーションデータを作製する。
(2) `ssden()` を使って、式 (1.4) の形式のノンパラメトリック確率密度関数を求める計算を行う。
(3) (2) の結果（`fit1`）を `dssden()` に入れることによって、ノンパラメトリック確率密度関数の推定値を求め、`ey` とする。
(4) 式 (1.4) における、$f_1(x_1)$ と $f_2(x_2)$ を求めて、それを用いて、$y$ の値を計算した結果を `ey2` とする。`ey[,10]` は $f_1(0)$ の値で、`ey[2,]` は $f_2(0)$ の値である。すると、$f_1(0)+f_2(0)$ の値が 4.017161 になったのである。

オワリ