

「データ解析」（下平英寿） 講義資料５

単回帰分析

- 目標： 単回帰分析を理解する．
  1. ２変量の統計量：共分散，相関係数など
  2. 回帰直線の推定：最小二乗法
  3. 推定量のバラツキ：ブートストラップ法
  4. ハズレ値の影響： 頑健な回帰分析

1 ２変量の統計量

- データ $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- （標本) 平均

$$\bar{x} = \frac{x_1 + \cdots + x_n}{n}, \quad \bar{y} = \frac{y_1 + \cdots + y_n}{n}$$

- (不偏標本) 分散

$$S_x^2 = \frac{(x_1 - \bar{x})^2 + \cdots + (x_n - \bar{x})^2}{n-1} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$S_y^2 = \frac{(y_1 - \bar{y})^2 + \cdots + (y_n - \bar{y})^2}{n-1} = \frac{1}{n-1}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

- (標本) 標準偏差

$$S_x = \sqrt{S_x^2}, \quad S_y = \sqrt{S_y^2}$$

- (不偏標本) 共分散

$$S_{xy} = \frac{(x_1 - \bar{x})(y_1 - \bar{y}) + \cdots + (x_n - \bar{x})(y_n - \bar{y})}{n-1} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

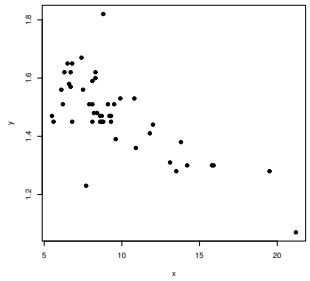
- （標本) 相関係数

$$r_{xy} = \frac{S_{xy}}{S_x S_y} = \frac{S_{xy}}{\sqrt{S_x^2 S_y^2}}$$

```
# run0031.R
# ２変量の統計量
dat <- read.table("dat0001.txt") # データの読み込み（47 x ２行列）
x <- dat[,1]; y <- dat[,2]
plot(x,y,pch=16) # 散布図 plot(dat,pch=16) でも良い．
dev.copy2eps(file="run0031-s.eps")
```



```
cat("# 平均の計算１: "); c(mean(x), mean(y)) # それぞれの平均
cat("# 平均の計算２: "); mymean <- function(v) sum(v)/length(v)
c(mymean(x), mymean(y)) # これでも同じ
cat("# 平均の計算３:\n"); mean(dat) # データフレームに適用

cat("# 分散の計算１: "); c(var(x), var(y)) # それぞれの分散
cat("# 分散の計算２: ");
myvar <- function(v) sum( (v-mymean(v))^2 )/(length(v)-1)
c(myvar(x), myvar(y)) # これでも同じ
cat("# 分散の計算３:\n"); var(dat) # データフレームに適用

cat("# 共分散の計算１: "); cov(x,y) # 共分散
cat("# 共分散の計算２: ")
mycov <- function(u,v) sum( (u-mymean(u))*(v-mymean(v)) )/(length(v)-1)
mycov(x,y) # これでも同じ

cat("# 相関係数の計算１: "); cov(x,y)/sqrt(var(x)*var(y)) # 相関係数
cat("# 相関係数の計算２: "); cor(x,y) # これでも同じ
cat("# 相関係数の計算３:\n"); cor(dat) # データフレームに適用
```



run0031-s

```
> source("run0031.R",print=T)
X11
  2
# 平均の計算１: [1] 9.540426 1.472979
# 平均の計算２: [1] 9.540426 1.472979
# 平均の計算３:
Gakureki  Shushou
```



```
9.540426 1.472979
# 分散の計算１: [1] 11.82637373  0.01772572
# 分散の計算２: [1] 11.82637373  0.01772572
# 分散の計算３:
           Gakureki     Shushou
Gakureki 11.8263737 -0.33407956
Shushou  -0.3340796  0.01772572
# 共分散の計算１: [1] -0.3340796
# 共分散の計算２: [1] -0.3340796
# 相関係数の計算１: [1] -0.7296628
# 相関係数の計算２: [1] -0.7296628
# 相関係数の計算３:
           Gakureki    Shushou
Gakureki  1.0000000 -0.7296628
Shushou  -0.7296628  1.0000000
```

2 単回帰分析

2.1 単回帰モデル

- x と y の関係を表現するモデル

$$y = \beta_0 + \beta_1 x + \epsilon$$

- x: 説明変数, 独立変数, 予測変数
- y: 目的変数, 従属変数, 応答変数
- ϵ: 誤差
- β_0, β_1: 回帰係数, 偏回帰係数

2.2 最小二乗法（その１）

- データへの当てはまりが最も良くなるように β_0, β_1 を調整する．
- 当てはめ： $i = 1, \ldots, n$ に対して

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

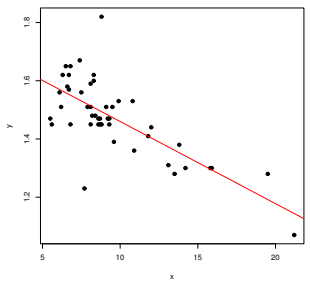
誤差の二乗和を最小にするように，β_0, β_1 を調整する．

$$\sum_{i=1}^{n} \epsilon_i^2 = \sum (\beta_0 + \beta_1 x_i - y_i)^2 \quad \Rightarrow \quad \text{最小化}$$

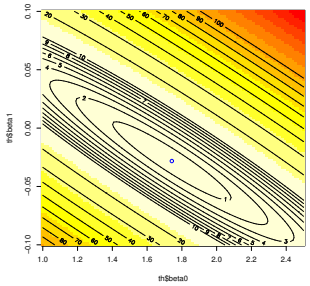
- まず optim 関数を用いて，数値的に最適解を求めてみる．

```
# run0032.R
# 最小二乗法（その１）
dat <- read.table("dat0001.txt") # データの読み込み（47 x ２行列）
x <- dat[,1]; y <- dat[,2]
rss <- function(be) sum( (be[1]+be[2]*x - y)^2 ) # 誤差の二乗和
be <- optim(c(0,0),rss)$par # 最小化
cat("# 回帰係数: "); print(be) # 最適値
cat("# 目的関数の最小値: "); print(rss(be)) # 最小値
plot(x,y,pch=16) # 散布図 plot(dat,pch=16) でも良い．
abline(be[1],be[2],col=2) # 回帰直線
dev.copy2eps(file="run0032-s.eps")

th <- list(beta0=seq(1,2.5,length=100),
           beta1=seq(-0.1,0.1,length=100)) # プロットする範囲
z <- matrix(apply(expand.grid(th),1,rss),
            length(th[[1]])) # すべての格子点で対数尤度を計算
image(th$beta0,th$beta1,-z) # ２次元プロット（赤＝高い，白＝低い）
contour(th$beta0,th$beta1,z,levels=c(0:9,(1:10)*10),add=T) # 等高線の表示
points(be[1],be[2],col=4)
dev.copy2eps(file="run0032-cp.eps")
```

```
> source("run0032.R")
# 回帰係数: [1]  1.74220470 -0.02822086
# 目的関数の最小値: [1] 0.3812672
```



run0032-s



run0032-cp

2.3 最小二乗法（その２）

- 最小二乗法には，数値的最適化は必要ない．解は次のように与えられる．

$$\hat{\beta}_1 = \frac{S_{xy}}{S_x^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

（解法）

$$S = \sum (\beta_0 + \beta_1 x_i - y_i)^2$$

を β_0 と β_1 に関して偏微分して

$$\frac{\partial S}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0$$

$$\frac{\partial S}{\partial \beta_1} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) x_i = 0$$

を β_0, β_1 に関して解けばよい．式を整理すると，

$$\bar{y} - \beta_0 - \beta_1 \bar{x} = 0, \quad \sum_{i=1}^{n} (y_i - \bar{y} - \beta_1 (x_i - \bar{x})) x_i = 0$$

最後の式は，

$$\sum_{i=1}^{n} \{(y_i - \bar{y})(x_i - \bar{x}) - \beta_1 (x_i - \bar{x})(x_i - \bar{x})\} = 0$$

つまり

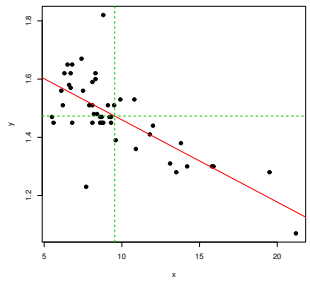
$$S_{xy} - \beta_1 S_x^2 = 0$$

- 回帰直線：$y = \hat{\beta}_0 + \hat{\beta}_1 x$
- $\bar{y} = \hat{\beta}_0 + \hat{\beta}_1 \bar{x}$ より，回帰直線は重心を通ることがわかる．回帰直線の傾きは，x と y の共分散を x の分散で割ったもの．

```
# run0033.R
# 最小二乗法（その２）
dat <- read.table("dat0001.txt") # データの読み込み（47 x 2行列）
x <- dat[,1]; y <- dat[,2]
rss <- function(be) sum( (be[1]+be[2]*x - y)^2 ) # 誤差の二乗和
beta1 <- cov(x,y)/var(x) # beta1の計算
beta0 <- mean(y) - beta1*mean(x) # beta0の計算
be <- c(beta0,beta1);
cat("# 回帰係数: "); print(be) # 最適値
cat("# 目的関数の最小値: "); print(rss(be)) # 最小値
plot(x,y,pch=16) # 散布図 plot(dat,pch=16)でも良い．
abline(be[1],be[2],col=2) # 回帰直線
abline(v=mean(x),col=3,lty=2) # 重心(x)
abline(h=mean(y),col=3,lty=2) # 重心(y)
dev.copy2eps(file="run0033-s.eps")
```



```
> source("run0033.R")
# 回帰係数: [1]  1.74248324 -0.02824869
# 目的関数の最小値: [1] 0.3812667
```



run0033-s

2.4 最小二乗法（その３）

- R に組み込みの関数を用いても良い．
- lsfit() は，最小二乗法の関数．
- lm() は，より一般的な線形モデルの当てはめの関数．R のオブジェクト指向プログラミングで設計されているので，実際には関数というよりメソッド．

```
# run0034.R
# 最小二乗法（その３）
dat <- read.table("dat0001.txt") # データの読み込み（47 x 2行列）
x <- dat[,1]; y <- dat[,2]
cat("# lsfitの実行\n")
fit1 <- lsfit(x,y) # 最小二乗法の計算（QR分解）
print(fit1$coef) # 回帰係数
cat("# lmの実行\n")
fit2 <- lm(y ~ x, data.frame(x,y)) # データフレームが必要
print(fit2$coef) # 回帰係数
fit3 <-  lm(Shushou ~ Gakureki,dat) # これでも良い
print(fit3$coef)
cat("# lsfitから得られる詳細な結果\n")
ls.print(fit1) # サマリー
```



```
cat("# lmから得られる詳細な結果\n")
print(summary(fit3)) # サマリー
```

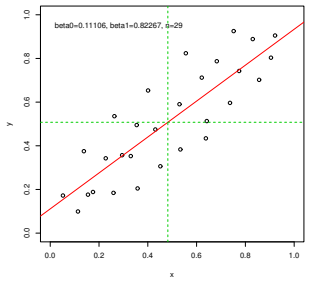
```
> source("run0034.R")
# lsfitの実行
  Intercept           X
 1.74248324 -0.02824869
# lmの実行
(Intercept)           x
 1.74248324 -0.02824869
(Intercept)    Gakureki
 1.74248324 -0.02824869
# lsfitから得られる詳細な結果
Residual Standard Error=0.092
R-Square=0.5324
F-statistic (df=1, 45)=51.2377
p-value=0

          Estimate Std.Err t-value Pr(>|t|)
Intercept   1.7425  0.0400 43.5916        0
X          -0.0282  0.0039 -7.1581        0


# lmから得られる詳細な結果

Call:
lm(formula = Shushou ~ Gakureki, data = dat)

Residuals:
      Min        1Q    Median        3Q       Max
-0.294968 -0.048132 -0.009319  0.045992  0.326105

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.742483   0.039973  43.592  < 2e-16 ***
Gakureki    -0.028249   0.003946  -7.158 5.94e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.09205 on 45 degrees of freedom
Multiple R-Squared: 0.5324,Adjusted R-squared: 0.522
F-statistic: 51.24 on 1 and 45 DF,  p-value: 5.943e-09
```


2.5 最小二乗法を実行してみる

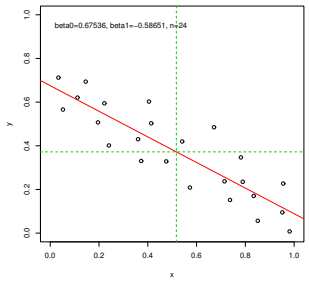
- マウスをクリックしてデータ点を自由に入力する．それに回帰直線を当てはめる．

```
# run0035.R
# 最小二乗法を実行してみる
func0035 <- function(na) {
  plot(0,0,xlab="x",ylab="y",xlim=c(0,1),ylim=c(0,1),type="n") # 枠を描く
  a <- locator(type="p") # 左ボタンクリックでデータ点の入力．他のボタンで終了
  x <- a$x; y <- a$y
  beta1 <- cov(x,y)/var(x) # beta1の計算
  beta0 <- mean(y) - beta1*mean(x) # beta0の計算
  be <- c(beta0,beta1); print(be) # 最適値
  abline(be[1],be[2],col=2) # 回帰直線
  abline(v=mean(x),col=3,lty=2) # 重心(x)
  abline(h=mean(y),col=3,lty=2) # 重心(y)
  text(0.0,0.95,pos=4,
       paste("beta0=",signif(be[1],5),", beta1=",signif(be[2],5),
             ", n=",length(x),sep="")) # betaとnをグラフに書き込む
  dev.copy2eps(file=paste("run0035-s",na,".eps",sep=""))
  invisible(a) # 関数値を返すがprintしない．
}
```

```
> source("run0035.R")
> func0035(1)
[1] 0.1110640 0.8226714
> func0035(2)
[1]  0.6753553 -0.5865131
```



run0035-s1



run0035-s2

2.6 最小二乗法と最尤法の関係

- 誤差 ϵ の従う確率分布が平均 0，分散 σ^2 の正規分布であると仮定する．

$$\epsilon \sim N(0, \sigma^2)$$

- データサイズが n なので，各点ごとに誤差がある．$\epsilon_1, \ldots, \epsilon_n$ が互いに独立に $(x_1, \ldots, x_n$ とも無関係に) $N(0, \sigma^2)$ に従うと仮定する．
- このとき，データ $(x_1, y_1), \ldots, (x_n, y_n)$ の確率密度関数は次式で与えられる．

$$f(x_1, y_1, \ldots, x_n, y_n | \beta_0, \beta_1, \sigma) = f(x_1, \ldots, x_n) f(y_1, \ldots, y_n | x_1, \ldots, x_n, \beta_0, \beta_1, \sigma)$$

最後の項は，$x_1, \ldots, x_n$ を与えたときの $y_1, \ldots, y_n$ の条件付確率密度関数．誤差が $\epsilon_i = y_i - \beta_0 - \beta_1 x_i$ であることに注意すれば，

$$f(y_1, \ldots, y_n | x_1, \ldots, x_n, \beta_0, \beta_1, \sigma) = \prod_{i=1}^{n} \left\{ \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2} \right) \right\}$$

- 最尤法では，この条件付確率密度を最大にするように，パラメタ $\theta = (\beta_0, \beta_1, \sigma)$ を決定する．条件付確率密度の対数を取れば，

$$\ell(\beta_0, \beta_1, \sigma) = -\frac{n}{2} \log(2\pi\sigma^2) - \sum_{i=1}^{n} \left(-\frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^2} \right)$$

この関数の形から，β_0 と β_1 について考えると，ℓ の最大化は最小二乗法になることがわかる．つまり，

$$\frac{\partial \ell}{\partial \beta_0} = 0, \quad \frac{\partial \ell}{\partial \beta_1} = 0$$

を解けば，

$$\hat{\beta}_1 = \frac{S_{xy}}{S_x^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

である．したがって，最尤法による回帰係数の推定は，最小二乗法に一致する．

- 一方誤差の分散 σ^2 の最尤推定に関しては，

$$\frac{\partial \ell}{\partial (\sigma^2)} = -\frac{n}{2\sigma^2} + \sum_{i=1}^{n} \left(-\frac{(y_i - \beta_0 - \beta_1 x_i)^2}{2\sigma^4} \right)$$

なのでこれを 0 と置けば，

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2$$

が得られる．

2.7 予測値と残差

- データ点 x_i における $\beta_0 + \beta_1 x_i$ の予測値

$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

- 現実の値 y と予測値との差を「残差」(residual) と呼ぶ

$$e_i = y_i - \hat{y}_i$$

- 誤差 ϵ の分散の不偏推定は

$$S_\epsilon^2 = \frac{1}{n-2} \sum_{i=1}^{n} e_i^2$$

分母の $n-2$ に注意．誤差の分散の最尤推定（正規分布を仮定）は

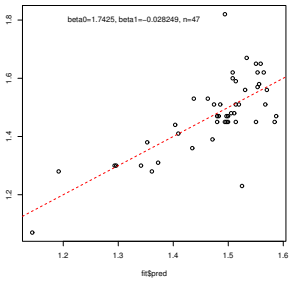
$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} e_i^2$$

であった．不偏分散の分母はデータサイズの n ではなく，「自由度」と呼ばれる量である．「自由度」= n −「推定した係数の個数」．n がある程度大きくなれば，n と $n-2$ の差は無視できる．

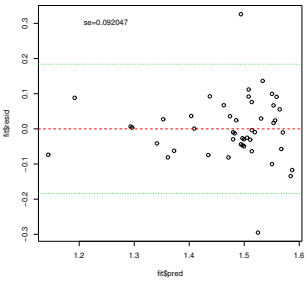
```
# run0036.R
# 単回帰分析の予測値と残差
func0036 <- function(x,y) {
  beta1 <- cov(x,y)/var(x) # beta1 の計算
  beta0 <- mean(y) - beta1*mean(x) # beta0 の計算
  beta <- c(beta0,beta1) # 回帰係数
  pred <- beta0 + beta1 * x # 予測値
  resid <- y - pred # 残差
  se <- sqrt(sum(resid^2)/(length(x)-2)) # 誤差分散の推定の平方根
  list(beta=beta,x=x,y=y,pred=pred,resid=resid,se=se)
}
dat <- read.table("dat0001.txt") # データの読み込み (47 x 2 行列)
fit <- func0036(dat[,1],dat[,2])
plot(fit$pred,fit$y) # 予測値と y
abline(0,1,col=2,lty=2)
text(1.2,1.8,pos=4,paste("beta0=",signif(fit$beta[1],5),
                ", beta1=",signif(fit$beta[2],5),", n=",length(fit$x),sep=""))
dev.copy2eps(file="run0036-s1.eps")
plot(fit$pred,fit$resid) # 予測値と残差
abline(h=0,col=2,lty=2) # e=0
abline(h=fit$se*2,col=3,lty=3) # e=2*se
abline(h=-fit$se*2,col=3,lty=3) # e=-2*se
```



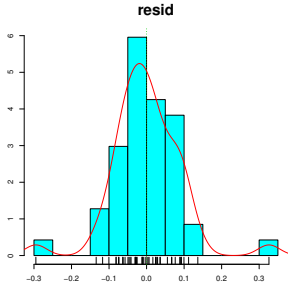
```
text(1.2,0.3,pos=4,paste("se=",signif(fit$se,5),sep=""))
dev.copy2eps(file="run0036-s2.eps")
source("run0044.R") # drawhist のロード
drawhist(fit$resid,20,"resid","run0036-") # 残差のヒストグラム
```



run0036-s1



run0036-s2



mean= −2.0787e−16 , sd= 0.091041
run0036-resid

3 推定量のバラツキ

3.1 ブートストラップ法（その１）

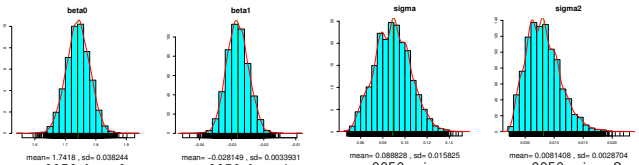
- 推定量 $(\hat{\beta}_0, \hat{\beta}_1, S_\epsilon)$ のバラツキを調べる．
- とりあえずブートストラップ法を実行してみる．
- データ $(x_1, y_1), \ldots, (x_n, y_n)$

- 各要素 (x_i, y_i) は 2 変量確率変数 (X, Y) の実現値であると考える．
- 各 (x_i, y_i) をひとつの要素とみなして，リサンプリングする．
  1. $(x_1, y_1), \ldots, (x_n, y_n)$ にたいしてブートストラップ法を適用し，ブートストラップ標本 $(x_1^*, y_1^*), \ldots, (x_n^*, y_n^*)$ を生成する．
  2. これにたいして回帰分析を適用し，ブートストラップ複製 $(\hat{\beta}_0^*, \hat{\beta}_1^*, S_\epsilon^*, S_\epsilon^{*2})$ を計算する．
  3. 上記 1,2 を多数回 (10000 回) くりかえす．
- このほかに，「残差のリサンプリング」という考え方もある．後で説明する．

```
# run0050.R
# 単回帰分析のブートストラップ法（その１）
# x と y にデータをあらかじめセットしておく．
fit <- func0036(x,y) # 単回帰分析
func0050 <- function(f) # fit から係数の取り出し
  unlist(list(beta0=f$beta[1],beta1=f$beta[2],sigma=f$se,sigma2=f$se^2))
cat("\n# 回帰係数，誤差の標準偏差，分散の推定\n"); print(func0050(fit))
boot1 <- function(i) { # i = サイズ n の添え字ベクトル
  a <- func0036(x[i],y[i]); # 単回帰分析
  func0050(a)
}
n <- length(y)
b <- 10000 # シミュレーションの繰り返し回数
cat("\n# 統計量をオリジナルデータへ適用\n"); print(boot1(1:n))
simi <- matrix(0,n,b) # ブートストラップ標本の添え字アレイを準備
for(j in 1:b) simi[,j] <- sample(1:n,replace=T) # ブートストラップ法
simt <- apply(simi,2,boot1) # 統計量を繰り返し適用
cat("\n# 統計量の平均，標準偏差\n")
a <- apply(simt,1,function(x) unlist(list(mean=mean(x),sd=sd(x))))
print(a)
for(k in rownames(simt)) drawhist(simt[k,],20,k,"run0050-") # ヒストグラム
cat("\n# lm() を利用してチェック\n")
print(summary(lm(y~x,data.frame(x,y)))$coef)
```

```
> library(MASS)
> source("run0044.R")
> dat <- read.table("dat0001.txt") # データの読み込み (47 x 2 行列)
> x <- dat[,1]; y <- dat[,2]
> source("run0050.R")

# 回帰係数，誤差の標準偏差，分散の推定
       beta0        beta1        sigma       sigma2
```

```
 1.742483239 -0.028248689  0.092046696  0.008472594

# 統計量をオリジナルデータへ適用
       beta0        beta1        sigma       sigma2
 1.742483239 -0.028248689  0.092046696  0.008472594

# 統計量の平均，標準偏差
           beta0        beta1       sigma       sigma2
mean 1.74175802 -0.028149266 0.08882796 0.008140813
sd   0.03824445  0.003393117 0.01582504 0.002870361

# lm() を利用してチェック
                Estimate  Std. Error   t value     Pr(>|t|)
(Intercept)  1.74248324 0.039972882 43.591634 1.875933e-38
x           -0.02824869 0.003946422 -7.158051 5.943343e-09
```



run0050-beta0 run0050-beta1 run0050-sigma run0050-sigma2

- 回帰係数の標準誤差：ブートストラップ推定と lm() の Std.Error は，だいたい同じ結果を与えている．
- lm() では，「$x_1, \ldots, x_n$ が定数，かつ誤差 ϵ は独立に正規モデルにしたがう」という仮定をして標準誤差を求めており，線形の単回帰分析では，lm() の結果で十分．この理論の導出は，後ほど説明する．
- しかし，ブートストラップ法は，非常に一般的な手法であり，他の複雑な問題でも，理論的な計算を知らなくても，このまま利用すれば，実用上十分な精度の標準誤差推定を与える．
- lm() の仮定により近づけたバージョンのブートストラップ法を次に示す．

3.2 ブートストラップ法（その２）

- 推定量 $(\hat{\beta}_0, \hat{\beta}_1, S_\epsilon, S_\epsilon^2)$ のバラツキを調べる．
- 「残差のリサンプリング」を実行する．

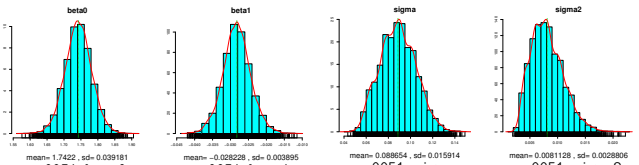
- データ $(x_1, y_1), \ldots, (x_n, y_n)$ で，各 (x_i, y_i) のうち，x_i は定数，y_i は確率変数 Y の実現値と考える．つまり，$x_1, \ldots, x_n$ は，あらかじめ与えられた定数なのでサンプリングしても不変，$y_1, \ldots, y_n$ が確率変数 Y の n 個の実現値なので，サンプリングすると変動する．
- この「モデル」を以下のブートストラップ法でシミュレートする．
  1. まず単回帰分析を行い，回帰係数 $\hat{\beta}_0, \hat{\beta}_1$ を推定する．
  2. 予測値 $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$ と残差 $e_i = y_i - \hat{y}_i$ を計算する．
  3. $e_1, \ldots, e_n$ に対してブートストラップ法を適用し，ブートストラップ標本 $e_1^*, \ldots, e_n^*$ を生成する．そして，y_i のブートストラップ標本を，$y_i^* = \hat{y}_i + e_i^*$ で定義する．
  4. データ全体のブートストラップ標本は，$(x_1, y_1^*), \ldots, (x_n, y_n^*)$ である．これにたいして回帰分析を適用し，ブートストラップ複製 $(\hat{\beta}_0^*, \hat{\beta}_1^*, S_\epsilon^*, S_\epsilon^{*2})$ を計算する．
  5. 上記 3,4 を多数回 (10000 回) くりかえす．

```
# run0051.R
# 単回帰分析のブートストラップ法（その２）
# x と y にデータをあらかじめセットしておく．
fit <- func0036(x,y) # 単回帰分析
cat("\n# 回帰係数，誤差の標準偏差，分散の推定\n"); print(func0050(fit))
pred <- fit$pred # 予測値
resid <- fit$resid # 残差
boot1 <- function(i) { # i = サイズ n の添え字ベクトル
  yi <- pred + resid[i] # ブートストラップ標本
  a <- func0036(x,yi); # 単回帰分析
  func0050(a)
}
n <- length(y)
b <- 10000 # シミュレーションの繰り返し回数
cat("\n# 統計量をオリジナルデータへ適用\n"); print(boot1(1:n))
simi <- matrix(0,n,b) # ブートストラップ標本の添え字アレイを準備
for(j in 1:b) simi[,j] <- sample(1:n,replace=T) # ブートストラップ法
simt <- apply(simi,2,boot1) # 統計量を繰り返し適用
cat("\n# 統計量の平均，標準偏差\n")
a <- apply(simt,1,function(x) unlist(list(mean=mean(x),sd=sd(x))))
print(a)
for(k in rownames(simt)) drawhist(simt[k,],20,k,"run0051-") # ヒストグラム
cat("\n# lm() を利用してチェック\n")
print(summary(lm(y~x,data.frame(x,y)))$coef)
```

```
> source("run0051.R")

# 回帰係数，誤差の標準偏差，分散の推定
```



```
       beta0        beta1        sigma       sigma2
 1.742483239 -0.028248689  0.092046696  0.008472594

# 統計量をオリジナルデータへ適用
       beta0        beta1        sigma       sigma2
 1.742483239 -0.028248689  0.092046696  0.008472594

# 統計量の平均，標準偏差
           beta0        beta1       sigma       sigma2
mean 1.74223477 -0.028227663 0.08865409 0.008112762
sd   0.03918107  0.003895007 0.01591353 0.002880596

# lm() を利用してチェック
                Estimate  Std. Error   t value     Pr(>|t|)
(Intercept)  1.74248324 0.039972882 43.591634 1.875933e-38
x           -0.02824869 0.003946422 -7.158051 5.943343e-09
```



run0051-beta0 run0051-beta1 run0051-sigma run0051-sigma2

- lm() でも，回帰係数の標準誤差（理論値のプラグイン推定量）を計算している．これを，標準誤差のブートストラップ推定と比較すると，両者は大体同じ結果である．
- $E(S_x^{*2}) < S_x^2$ であり，分散推定に多少バイアスがある．ところが，lm() の Std.Error と同じ仮定の下で，S_x^2 は不偏な推定量である（これの証明は後ほど）．したがって，この（わずかな）バイアスは推定量 S_x^2 に責任があるのではなく，ブートストラップ法（その２）が原因である．これの修正を次で考える．

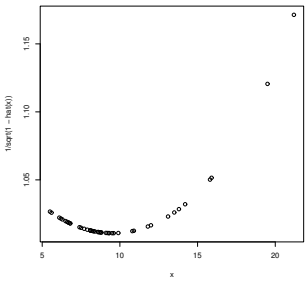
3.3 ブートストラップ法（その３）

- 推定量 $(\hat{\beta}_0, \hat{\beta}_1, S_\epsilon, S_\epsilon^2)$ のバラツキを調べる．
- 「残差のリサンプリング」を実行する．
- ただし，残差をあらかじめ修正しておく．hat(x) は「ハット行列」の対角成分 $h_{ii}, i = 1, \ldots, n$ を計算する関数.(この意味については，後ほど説明する.)．修正残差 $r_i, i = 1, \ldots, n$ を，$r_i = e_i / \sqrt{1 - h_{ii}}$ で定義し，これをリサンプリングする．

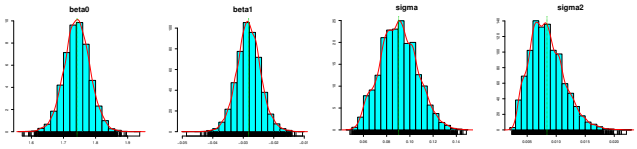

```
# run0052.R
# 単回帰分析のブートストラップ法（その３）
# x と y にデータをあらかじめセットしておく．
fit <- func0036(x,y) # 単回帰分析
cat("\n# 回帰係数，誤差の標準偏差，分散の推定\n"); print(func0050(fit))
pred <- fit$pred # 予測値
resid <- fit$resid/sqrt(1-hat(x)) # 修正残差
plot(x,1/sqrt(1-hat(x)))
dev.copy2eps(file="run0052-hat.eps")
boot1 <- function(i) { # i = サイズ n の添え字ベクトル
  yi <- pred + resid[i] # ブートストラップ標本
  a <- func0036(x,yi); # 単回帰分析
  func0050(a)
}
n <- length(y)
b <- 10000 # シミュレーションの繰り返し回数
cat("\n# 統計量をオリジナルデータへ適用\n"); print(boot1(1:n))
simi <- matrix(0,n,b) # ブートストラップ標本の添え字アレイを準備
for(j in 1:b) simi[,j] <- sample(1:n,replace=T) # ブートストラップ法
simt <- apply(simi,2,boot1) # 統計量を繰り返し適用
cat("\n# 統計量の平均，標準偏差\n")
a <- apply(simt,1,function(x) unlist(list(mean=mean(x),sd=sd(x))))
print(a)
for(k in rownames(simt)) drawhist(simt[k,],20,k,"run0052-") # ヒストグラム
cat("\n# lm() を利用してチェック\n")
print(summary(lm(y~x,data.frame(x,y)))$coef)
```

```
> source("run0052.R")

# 回帰係数，誤差の標準偏差，分散の推定
       beta0        beta1        sigma       sigma2
 1.742483239 -0.028248689  0.092046696  0.008472594

# 統計量をオリジナルデータへ適用
       beta0        beta1        sigma       sigma2
 1.74298852 -0.02831561  0.09385334  0.00880845

# 統計量の平均，標準偏差
           beta0        beta1       sigma       sigma2
mean 1.74224071 -0.028234465 0.09035270 0.008413201
sd   0.04026878  0.003981816 0.01579923 0.002921135
```

```
# lm() を利用してチェック
                Estimate  Std. Error   t value     Pr(>|t|)
(Intercept)  1.74248324 0.039972882 43.591634 1.875933e-38
x           -0.02824869 0.003946422 -7.158051 5.943343e-09
```



run0052-hat



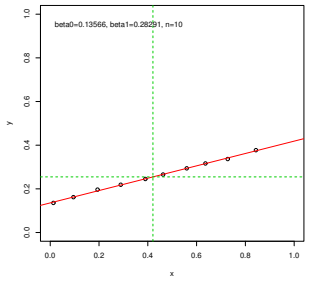
run0052-beta0 run0052-beta1 run0052-sigma run0052-sigma2

- 修正項 $1/\sqrt{1-h_{ii}}, i=1,\ldots,n$ のグラフをみると，修正項は ≥ 1 であるので，残差 e_i を多少増幅している．x が小さいか大きいところで，修正項が大きくなるので，その増幅率が大きい．じつは，修正をしない残差 e_i は，本来あるべき値よりも，x の両端で小さな値をとる傾向があるので，その分を補正項を掛けて元にもどしている．
- lm() でも，回帰係数の標準誤差（理論値のプラグイン推定量）を計算している．これを，標準誤差のブートストラップ推定と比較すると，両者は大体同じ結果である．
- ほぼ $E(S_x^{*2}) \approx S_x^2$ であり，バイアスがないことが分かる．
- ところが S_x のほうで見ると，まだわずかに $E(S_x^*) < S_x$ である．$S_x = \sqrt{S_x^2}$ である．一般に推定量の不偏性という性質は，その統計量を非線形変換 (ここでは平方根) ですることにより失われる．実用的には，この程度のわずかなバイアスに，そもそもこだわる必要はないだろう．理論統計では，しばしば不偏性を過度に重要視することがあるが，非線形変換で失われる程度の性質なので，それほどこだわる必要がないとも言える．(しかし，近似的な不偏性は，やはりあるほうがよい．厳密な不偏性に過度こだわる必要がないということ．)

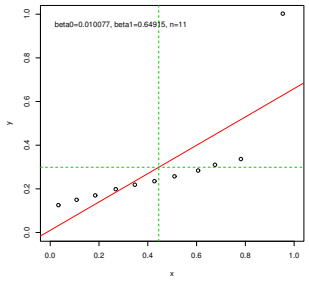
4 ハズレ値の影響

4.1 まず実験してみる

```
> func0035(3)
[1] 0.1356613 0.2829097
> func0035(4)
[1] 0.01007696 0.64914966
```



run0035-s3



run0035-s4

4.2 頑健な回帰分析

- 通常の最小二乗法

$$\sum (\beta_0 + \beta_1 x_i - y_i)^2 \quad \Rightarrow \quad \text{最小化}$$

- 重み付き最小二乗法

$$\sum w_i (\beta_0 + \beta_1 x_i - y_i)^2 \quad \Rightarrow \quad \text{最小化}$$

- 通常の最小二乗法は，$w_1 = \cdots = w_n = 1$ に相当．
- 「ハズレ値」である可能性が高い (x_i, y_i) に対しては，w_i を 0 に近づければよい．明らかにハズレ値なら $w_i = 0$ とすればよい．
- 頑健な回帰分析では，データから w_i を自動的に計算する．この計算のアルゴリズムに様々なバージョンがある．
- 基本的なアイデアは，次のとおり
  1. まず回帰係数 $(\hat{\beta}_0, \hat{\beta}_1)$，誤差のバラツキ $\hat{\sigma}$ の初期推定を行う．
  2. 予測値 $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i, i = 1, \ldots, n$ を計算する．

  3. 残差 $e_i = y_i - \hat{y}_i$ の大きさを $\hat{\sigma}$ と比較して，$|e_i|$ が大きいほど，w_i を 0 に近づける．$|e_i|$ が $\hat{\sigma}$ とあまり変わらなければ，$w_i \approx 1$ にする．(この部分の実装によってアルゴリズムのバージョンが様々にある．)
  4. 重み付き最小二乗法によって $\hat{\beta}_0, \hat{\beta}_1, \hat{\sigma}$ を推定する．
  5. 十分収束するまで，2,3,4 を繰り返す．
- 現実には，どのくらいズレていればハズレ値なのか，判断は難しい．したがって，どの手法がよいかは場合による．(理論的な比較研究は多数ある．)
- そもそも，直線当てはめの「モデル」が間違っている可能性もある．たとえば真実が 2 次曲線なのに，無理やり単回帰分析を実行すれば，両端がハズレ値と判断されるかもしれない．

4.3 頑健な回帰分析の実装

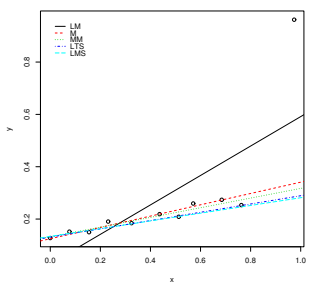
- R における頑健な回帰分析の実装：MASS ライブラリの rlm 関数と lqs 関数．どちらもオプションの指定により動作が変わる．Venables and Ripley "Modern Applied Statistics with S" fourth edition の 156 ページを参照．
- 以下では lm，rlm（2 通り），lqs（2 通り）の回帰分析を行う．

```
# run0053.R
# 頑健回帰
library(MASS) # rlm,lqs は MASS ライブラリで定義されている
rkaiki <- function(x,y,led=c(min(x),max(y)),pl=T) { # 各種の頑健回帰
  a <- data.frame(x,y) # データフレームにしておく
  fit <- list()
  fit[[1]] <- lm(y ~ x, a) # 最小二乗法
  fit[[2]] <- rlm(y ~ x, a) # M 推定量 (method="M")
  fit[[3]] <- rlm(y ~ x, a, method="MM") # MM 推定量
  fit[[4]] <- lqs(y ~ x, a) # 抵抗回帰 LTS 法 (method="lts")
  fit[[5]] <- lqs(y ~ x, a, method="lms") # 抵抗回帰 LMS 法
  names(fit) <- c("LM","M","MM","LTS","LMS")
  if(pl) {
    plot(x,y) # 散布図
    for(i in 1:5) abline(fit[[i]],lty=i,col=i) # 回帰直線
    legend(led[1],led[2],names(fit),lty=1:5,col=1:5,bty="n") # 凡例
  }
  invisible(fit) # 値を返すが表示しない
}
func0053 <- function(na) {
  plot(0,0,xlab="x",ylab="y",xlim=c(0,1),ylim=c(0,1),type="n") # 枠を描く
```

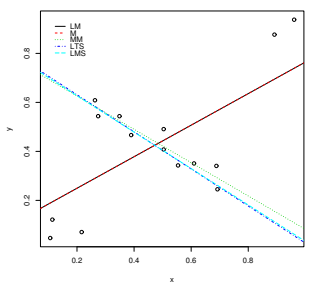


```
  a <- locator(type="p") # 左ボタンクリックでデータ点の入力．他のボタンで終了
  fit <- rkaiki(a$x,a$y)
  cat("\n# 回帰係数\n")
  print(t(sapply(fit,function(f) f$coef)))
  dev.copy2eps(file=paste("run0053-s",na,".eps",sep=""))
  invisible(a) # 関数値を返すが print しない．
}
```

```
> source("run0053.R")
> func0053(1)

# 回帰係数
    (Intercept)         x
LM    0.0291045 0.5624292
M     0.1248387 0.2161695
MM    0.1324150 0.1847028
LTS   0.1309441 0.1582447
LMS   0.1337484 0.1478169
> func0053(2)

# 回帰係数
    (Intercept)          x
LM    0.1207470  0.6437719
M     0.1207470  0.6437719
MM    0.7622259 -0.6802707
LTS   0.7825285 -0.7568227
LMS   0.7756168 -0.7428357
```



run0053-s1



run0053-s2

- rlm 関数：通常の最小二乗法 lm を頑健 (robust) にしたものなので，rlm という名前．以下の説明の詳細は，help(rlm) 参照．

  1. rlm(...,method="M") は M 推定量（rlm のデフォルト）．重み関数は Huber 法．スケール ($\hat{\sigma}$) を MAD で推定．
  2. rlm(...,method="MM") は MM 推定量．M 推定量の初期値設定，重み関数，スケールを改良したもの．重み関数は Tukey's biweight 法．理論的には，性能が高いことが示される．つまり，うまく回帰直線が推定できるためのハズレ値の比率の限界 (break-down point という) が高く 0.5 であり，また同じサンプルサイズのとき標準誤差が小さく（有効性が高い，efficient という），最適な場合の９５%の性能．一般にハズレ値に強いということは，回帰直線からのズレに敏感に反応して重みを小さくするということであり，結果としてデータを無駄に捨てるので，有効性とは互いにトレードオフの関係にある．

- lqs 関数：抵抗回帰直線

  1. lqs(...,method="lts") は LTS (least trimmed squares) 推定量（lqs のデフォルト）．

     $$e_1^2, \ldots, e_n^2 \quad \text{の刈り込み平均} \Rightarrow \text{最小}$$

     ハズレ値に強くブレークダウンポイントは約０．５，有効性も高いことが示されている．(ブレークダウンポイントは刈り込み率による．ここでは $\alpha \approx 0.25$ としている)
  2. lqs(...,method="lms") は LMS (least median of squares) 推定量．

     $$e_1^2, \ldots, e_n^2 \quad \text{のメディアン} \Rightarrow \text{最小}$$

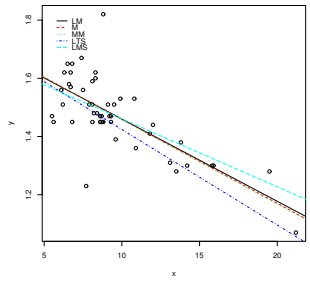
     ハズレ値に強くブレークダウンポイントは約０．５だが，有効性が低い（標準誤差が大きい）．

- 結論としては，MM か LTS が推薦？

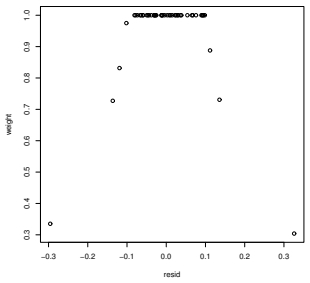
```
# run0054.R
# 頑健回帰（重み関数の表示）
# x と y にデータをあらかじめセットしておく．
fit <- rkaiki(x,y)
cat("\n# 回帰係数\n")
print(t(sapply(fit,function(f) f$coef)))
dev.copy2eps(file="run0054-s.eps")
for(i in c(2,3)) {
  resid <- fit[[i]]$resid; weight <- fit[[i]]$w
  plot(resid,weight)
  dev.copy2eps(file=paste("run0054-w",i,".eps",sep=""))
}
```



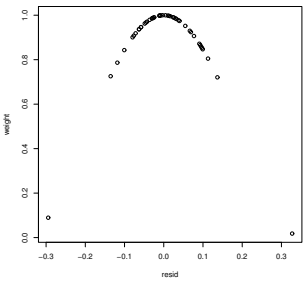
```
> dat <- read.table("dat0001.txt") # データの読み込み（47 x 2行列）
> x <- dat[,1]; y <- dat[,2]
> source("run0054.R")

# 回帰係数
    (Intercept)           x
LM     1.742483 -0.02824869
M      1.747932 -0.02886364
MM     1.747779 -0.02899027
LTS    1.752331 -0.03281250
LMS    1.694000 -0.02333333
```



run0054-s



run0054-w2



run0054-w3

4.4 推定量のバラツキのブートストラップ推定

- ブートストラップ法で，各種の頑健回帰による回帰係数 $\hat{\beta}_0, \hat{\beta}_1$ のバラツキを評価する．
- データ $(x_1, y_1), \ldots, (x_n, y_n)$．各要素 (x_i, y_i) は２変量確率変数 (X, Y) の実現値であると考える．各 (x_i, y_i) をひとつの要素とみなして，リサンプリングする．

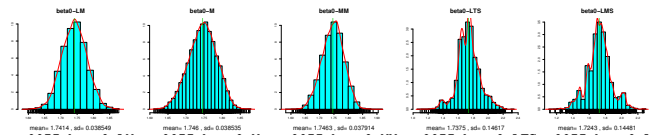
```
# run0055.R
# 頑健回帰のブートストラップ
# x と y にデータをあらかじめセットしておく．
fit <- rkaiki(x,y,pl=F) # 頑健回帰
func0055 <- function(fit) { # fitから係数の取り出し
  a <- sapply(fit,function(f) f$coef)
  na <- as.vector(outer(c("beta0","beta1"),
                        dimnames(a)[[2]],paste,sep="-"))
  a <- as.vector(a); names(a) <- na
  a
}
cat("\n# 回帰係数\n"); print(func0055(fit))
boot1 <- function(i) { # i = サイズ n の添え字ベクトル
  fit <- rkaiki(x[i],y[i],pl=F); # 単回帰分析
  func0055(fit)
}
n <- length(y)
b <- 10000 # シミュレーションの繰り返し回数
cat("\n# 統計量をオリジナルデータへ適用\n"); print(boot1(1:n))
simi <- matrix(0,n,b) # ブートストラップ標本の添え字アレイを準備
print(date())
for(j in 1:b) simi[,j] <- sample(1:n,replace=T) # ブートストラップ法
print(date())
simt <- apply(simi,2,boot1) # 統計量を繰り返し適用
print(date())
cat("\n# 統計量の平均，標準偏差\n")
a <- apply(simt,1,function(x) unlist(list(mean=mean(x),sd=sd(x))))
print(a)
for(k in rownames(simt)) drawhist(simt[k,],20,k,"run0055-") # ヒストグラム
cat("\n# lm() を利用してチェック\n")
print(summary(lm(y~x,data.frame(x,y)))$coef)
```

```
# run0055bat.R
source("run0044.R")
source("run0053.R")
```

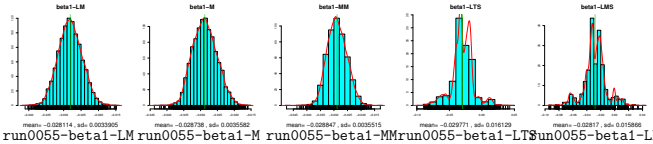


```
dat <- read.table("dat0001.txt") # データの読み込み（47 x 2行列）
x <- dat[,1]; y <- dat[,2]
source("run0055.R")
```

```
> source("run0055bat.R")

# 回帰係数
   beta0-LM    beta1-LM     beta0-M     beta1-M    beta0-MM    beta1-MM
 1.74248324 -0.02824869  1.74793182 -0.02886364  1.74777887 -0.02899027
  beta0-LTS   beta1-LTS   beta0-LMS   beta1-LMS
 1.75233073 -0.03281250  1.69400000 -0.02333333

# 統計量をオリジナルデータへ適用
   beta0-LM    beta1-LM     beta0-M     beta1-M    beta0-MM    beta1-MM
 1.74248324 -0.02824869  1.74793182 -0.02886364  1.74777887 -0.02899027
  beta0-LTS   beta1-LTS   beta0-LMS   beta1-LMS
 1.75233073 -0.03281250  1.69400000 -0.02333333
[1] "Tue Sep 28 17:14:29 2004"
[1] "Tue Sep 28 17:14:30 2004"
[1] "Tue Sep 28 17:34:36 2004"

# 統計量の平均，標準偏差
       beta0-LM     beta1-LM     beta0-M      beta1-M   beta0-MM     beta1-MM
mean 1.74142433 -0.028113793 1.74596525 -0.028737512 1.74627196 -0.028847435
sd   0.03854868  0.003390508 0.03853532  0.003558198 0.03791426  0.003551456
     beta0-LTS   beta1-LTS beta0-LMS   beta1-LMS
mean 1.7374880 -0.02977136 1.7243136 -0.02817012
sd   0.1461708  0.01612913 0.1448134  0.01586646

# lm() を利用してチェック
               Estimate  Std. Error   t value     Pr(>|t|)
(Intercept)  1.74248324 0.039972882 43.591634 1.875933e-38
x           -0.02824869 0.003946422 -7.158051 5.943343e-09
There were 18 warnings (use warnings() to see them)
```



run0055-beta0-LM run0055-beta0-M run0055-beta0-MM run0055-beta0-LTS run0055-beta0-LMS



run0055-beta1-LM run0055-beta1-M run0055-beta1-MM run0055-beta1-LTS run0055-beta1-LMS

- 実行に 20 分くらいかかった.（ほとんどの時間は，回帰分析を繰り返し実行するところに費やされている.）
- 標準誤差をみると，LM ≈ M ≈ MM << LTS ≈ LMS となっている.
  1. `lm` はハズレ値に弱いが，標準誤差が小さい
  2. `rlm` はハズレ値に強く，標準誤差も小さい.
  3. `lqs` はハズレ値に強いが，標準誤差が大きい.
- 結局 MM が推薦？

5 課題

5.1 課題 5-1

4.3 節を参考にして，以下を実行せよ．自分で書いたプログラムとその実行結果も示せ．（講義資料に示されているプログラムは自由に利用してよい．）

- マウスクリックでデータサイズ $n = 20 \sim 50$ 程度の 2 変量データを作成する．背後にある真の回帰直線やハズレ値を意識して作成すると良い.
- 単回帰分析（LM, M, MM, LTS, LMS の各手法）を実行し，データの散布図に回帰直線を書き込む．それぞれの手法について，回帰係数 $(\hat{\beta}_0, \hat{\beta}_1)$ を示す.
- LM について，回帰係数の標準誤差を示せ.
- データ作成時に意識した点，および，それがどのように推定された回帰直線に反映されたかを述べよ.

5.2 課題 5-2

上記の作成したデータについて，ブートストラップ法を適用し，LM, M, MM, LTS, LMS から得られた回帰係数の標準誤差を示せ.（利用する計算機の性能によっては，計算時間がかかりすぎるかもしれない．その場合は，ブートストラップ法の繰り返し数を 10000 から 1000 程度に減らしても良い．）

5.3 課題 5-3 ∗

与えられたデータベクトル `x` と `y` から，LMS 法の単回帰係数 $\hat{\beta}_0, \hat{\beta}_1$ を計算する関数を作成せよ．その実行例を示し，`lqs(...,method = "lms")` と比較して動作を確認せよ．なお，メディアンの計算は `median()` 関数を利用してよい．また最小化のアルゴリズムは `optim()` を利用しても良いが，いろいろ工夫する必要があるかもしれない.（メディアンは滑らかな関数では無いことに注意．）