クレジット:
UTokyo Online Education 統計データ解析 Ⅱ 2018 小池祐太

東京大学 THE UNIVERSITY OF TOKYO

CC BY NC ND

統計データ解析 II (平成３０年度)

東京大学大学院数理科学研究科
統計データ解析教育研究グループ

村田　昇 (早稲田大学, 東京大学)
吉田朋広 (東京大学)
小池祐太 (東京大学)

第 8 章 時系列解析

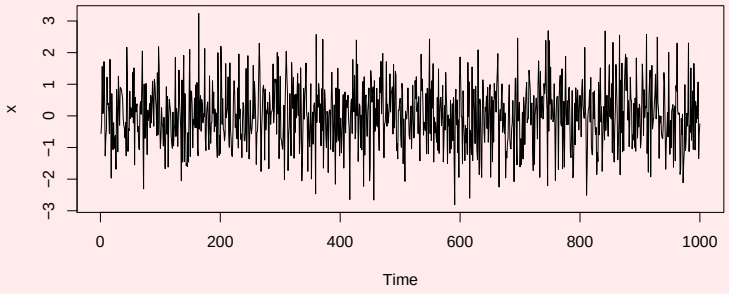
8.1. 時系列のモデル

時間軸に沿って観測されたデータを**時系列データ** *(time series data)* と呼ぶ. 時系列データの特徴は, 観測の順序に意味があることや, 異なる時点間での観測データの間の従属関係が重要であることが挙げられる. 時系列解析ではそのような時系列データの特徴を効果的に記述することが目的である.

統計学では, 時系列データは**確率過程** *(stochastic process)* (時間を添え字として持つ確率変数列: $X_t, t = 0, 1, \ldots, T$ (あるいは $t = 1, \ldots, T$) によってモデル化する. 以下に時系列分析で利用されるいくつかの代表的な確率過程およびその R でのシミュレーション法について述べる. なお, R には時系列データを表すためのクラス `ts` が用意されており, 例えば関数 `plot()` を適用した際の挙動が通常のベクトルと異なる (`ts` クラスのデータのプロットはデフォルトで折れ線プロットとなる). 通常のベクトル `x` は関数 `ts()` を適用することで `ts` クラスに変換できる.

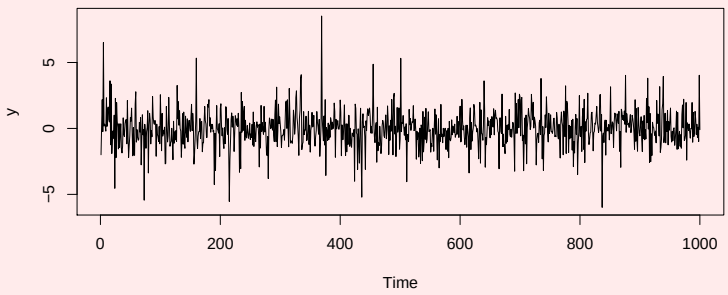
8.1.1. ホワイトノイズ. 平均 0, 分散 σ^2 で互いに無相関な確率変数列からなる確率過程を**ホワイトノイズ** *(white noise)*$\mathrm{WN}(0, \sigma^2)$ と呼ぶ. 平均 0 で有限の分散を持つ同一の分布に従う独立な確率変数列がホワイトノイズの典型的な例である.[1] このことから, ホワイトノイズのシミュレーションには乱数発生器 (`rnorm()` や適当な自由度の `rt()` など) で行うことが多い.

```
> set.seed(123)
> n <- 1000 # 時系列の長さ
> x <- ts(rnorm(n)) # 正規分布の場合. ts() で ts クラスのオブジェクトを作る
> plot(x)
```




```
> y <- ts(rt(n, df = 4)) # 自由度 4 の t 分布の場合
> plot(y)
```

[1] ただし, ホワイトノイズは独立な確率変数列になるとは限らない.



(wn.r)

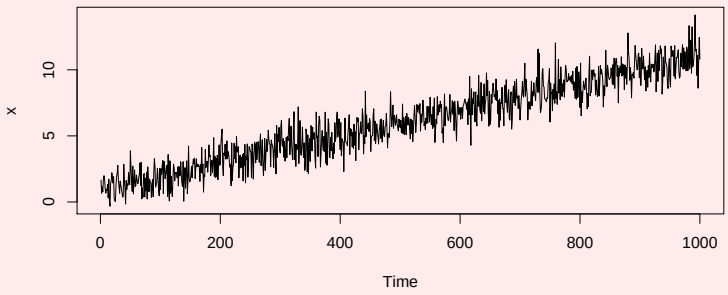
8.1.2. トレンドのあるホワイトノイズ. 確率過程 ϵ_t $(t = 1, \ldots, T)$ はホワイトノイズを表すとする. μ, α を定数とするとき,

$$X_t = \mu + \alpha t + \epsilon_t$$

で与えられる時系列データ $X_t,\ t = 1, \ldots, T$ を**トレンドのあるホワイトノイズ** *(white noise with a trend)* と呼ぶ. 項 "$\mu + \alpha t$" は**トレンド** *(trend)* と呼ばれる. トレンドのあるホワイトノイズは, 時間とともに平均が変動する時系列モデルの一つである.

トレンド項としてここでは t の 1 次式を考えているが, 高次の多項式や非線形関数 (指数関数, 三角関数など) を考えることもある.

```
> n <- 1000
> mu <- 1
> alpha <- 0.01
> x <- ts(mu + alpha * (1:n) + rnorm(n))
> plot(x)
```



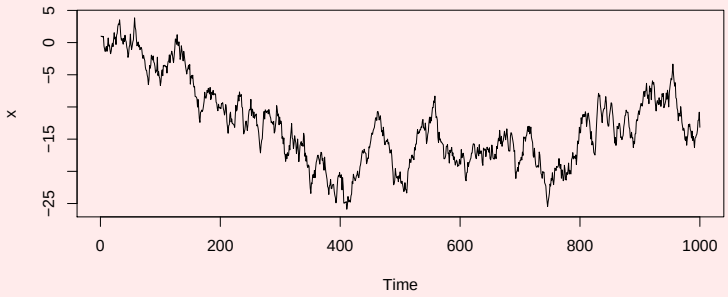
(trend.r)

8.1.3. ランダムウォーク. X_1 を定数もしくは確率変数として, 式

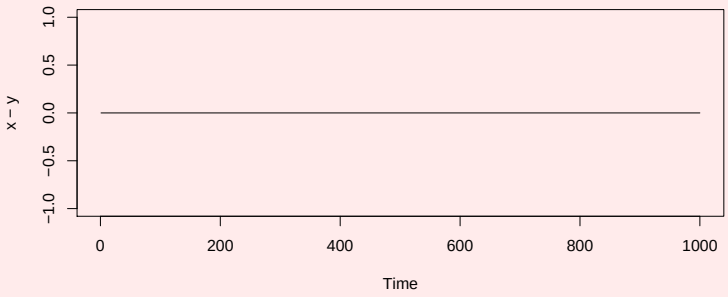
$$X_t = X_{t-1} + \epsilon_t, \qquad (t = 2, \ldots, T)$$

で帰納的に定義される確率過程 $X_t,\ t = 1, \ldots, T$ を**ランダムウォーク** *(random walk)* と呼ぶ. ここで $\epsilon_t,\ t = 2, \ldots, T$ は同一の分布に従う独立な確率変数列 (i.i.d.) である. ϵ_t が分散を持つとき, ランダムウォークは分散が時間とともに増加する時系列モデルの一つである.

```
> n <- 1000
> x0 <- 1 # 初期値
> epsilon <- rnorm(n-1)
> ## for 文によって生成する方法
> x <- ts(double(n)) # ts オブジェクトを作成
> x[1] <- x0
> for(i in 2:n) x[i] <- x[i-1] + epsilon[i-1]
> plot(x)
```




```
> ## 関数 diffinv を使う方法 (こちらの方が速い)
> y <- ts(diffinv(epsilon, xi = x0))
> plot(x - y) # x と y は全く同じ系列
> # 注. diff は階差（差分），diffinv はその逆関数（和分）である.
> (0:10)^2
 [1]   0   1   4   9  16  25  36  49  64  81 100
> diff((0:10)^2)
 [1]  1  3  5  7  9 11 13 15 17 19
> diffinv(diff((0:10)^2))
 [1]   0   1   4   9  16  25  36  49  64  81 100
```



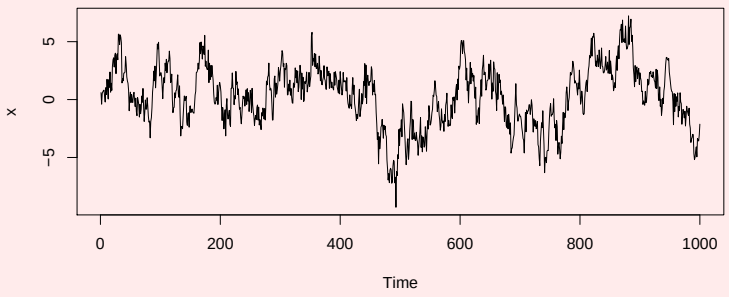
(rw2.r)

8.1.4. 自己回帰モデル (AR モデル). $\epsilon_t,\ t = p+1, \ldots, T$ を $\mathrm{WN}(0, \sigma^2)$ とする. $a_1, \ldots, a_p$ を定数とする. $X_1, \ldots, X_p$ が初期値として与えられたとき,

$$X_t = a_1 X_{t-1} + \cdots + a_p X_{t-p} + \epsilon_t, \qquad t = p+1, \ldots, T$$

で帰納的に定まる確率過程を**次数 p の自己回帰過程** *(autoregressive process of order p)* と呼び，このモデルを**次数 p の自己回帰モデル**もしくは **AR(p) モデル**と呼ぶ (AR は autoregressive の略). とくに $p = 1$, $a_1 = 1$ でさらに $\epsilon_t,\ t = 2, ..., T$ が独立で同一の分布に従う場合，ランダムウォークとなるため, AR 過程はランダムウォークの一般化と考えられる.

```
> ## AR(2) モデルのシミュレーション
> n <- 1000
> a <- c(0.669, 0.263) # AR の係数
> epsilon <- rnorm(n - 2)
> x0 <- rnorm(2) # 初期値を乱数で設定
> # for 文で生成する方法
> x <- ts(double(n))
> x[1:2] <- x0
> for(i in 3:n) x[i] <- a %*% x[i - 1:2] + epsilon[i-2]
> plot(x)
> #
> # 関数 filter を使う方法 (こちらの方が速い)
> y <- ts(c(x0, filter(epsilon, filter = a, method = "r", init = rev(x0))))
> sum(abs(x - y)) # ほぼ 0(数値計算上の問題からちょうど 0 でない)
[1] 3.703982e-13
> #
> # 注. 関数 filter は時系列に対する線形フィルタである.
> # method で"c"(convolution) を選ぶと moving average,
> # "r"(recursive) を選ぶと autoregression の変換になる.
> # それぞれの変換を行うときの係数を filter で指定する:
> a <- c(-2,1) # a_1, a_2
> x.init <- c(0,1) # x_2, x_1
> ep <- c(5,6,7)
> (x <- filter(ep,filter = a, method = "r", init = x.init))
Time Series:
Start = 1
End = 3
Frequency = 1
[1]  6 -6 25
> # filter の出力結果の確認:
> a[1]*x.init[1]+a[2]*x.init[2]+ep[1] # x_3
[1] 6
> a[1]*x[1]+a[2]*x.init[1]+ep[2] # x_4
[1] -6
> a[1]*x[2]+a[2]*x[1]+ep[3]  # x_5
[1] 25
```



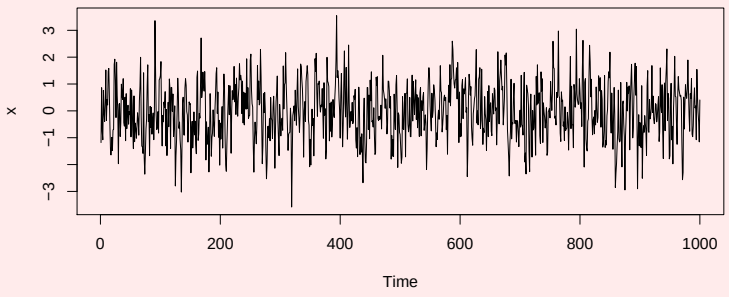
(ar2.r)

8.1.5. 移動平均モデル (MA モデル). $\epsilon_t,\ t = q+1,\dots,T$ を $\mathrm{WN}(0,\sigma^2)$ とする. $b_1,\dots,b_q$ を定数とする. $X_1,\dots,X_q$ が初期値として与えられたとき,

$$X_t = \epsilon_t + b_1\epsilon_{t-1} + \cdots + b_q\epsilon_{t-q}, \qquad t = q+1,\dots,T$$

で定まる確率過程を**次数 q の移動平均過程** *(moving average process of order q)* と呼び，このモデルを**次数 q の移動平均モデル**もしくは **MA(q) モデル**と呼ぶ (MA は moving average の略).

```
> ## MA(2) モデルのシミュレーション
> n <- 1000
> b <- c(0.438, 0.078) # MA の係数
> epsilon <- rnorm(n)
> x0 <- epsilon[1:2] # 初期値は epsilon1, epsilon2 とする
> # for 文で生成する方法
> x <- ts(double(n))
> x[1:2] <- x0
> for(i in 3:n) x[i] <- b %*% epsilon[i - 1:2] + epsilon[i]
> plot(x)
> # 関数 filter を使う方法 (こちらの方が速い)
> y <- ts(filter(epsilon, filter = c(1, b), method = "c", sides = 1))
> # sides=1 は moving average を過去の方向にのみ行うことを意味する.
> y[1:2] <- epsilon[1:2]
> sum(abs(x - y)) # ほぼ 0(数値計算上の問題からちょうど 0 でない)
[1] 3.734339e-14
```



(ma2.r)

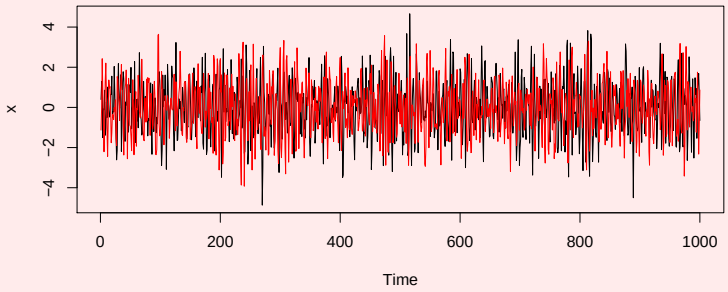
8.1.6. 自己回帰平均移動モデル (ARMA モデル). $a_1, \ldots, a_p, b_1, \ldots, b_q$ を定数とする. $X_1, \ldots, X_{\max\{p,q\}}$ が初期値として与えられたとき,

$$X_t = a_1 X_{t-1} + \cdots + a_p X_{t-p} + b_1 \epsilon_{t-1} + \cdots + b_q \epsilon_{t-q} + \epsilon_t, \qquad t = \max\{p, q\} + 1, \ldots, T$$

で帰納的に定まる確率過程を**次数 (p, q) の自己回帰平均移動モデル**もしくは **ARMA(p, q) モデル**と呼ぶ. ARMA$(p, 0)$ モデルは AR(p) モデルであり, ARMA$(0, q)$ モデルは MA(q) モデルであるから, ARMA モデルは AR モデルと MA モデルの一般化である. ARMA モデルは単純な形ながら異なる時点間の観測データの従属構造を柔軟に記述できるため, 基本的な時系列モデルとして広く利用されている.

```
> ## ARMA(2,1) モデルのシミュレーション
> n <- 1000
> a <- c(0.8, -0.64) # AR の係数
> b <- -0.5 # MA の係数
> # for 文で生成する方法
> epsilon <- rnorm(n)
> x0 <- rnorm(2) # 初期値を乱数で設定
> x <- ts(double(n))
> x[1:2] <- x0
> for(i in 3:n) x[i] <- a %*% x[i - 1:2] + b * epsilon[i - 1] + epsilon[i]
> plot(x)
```

```
> # arima.sim で生成する方法 (初期値の指定はできない)
> # 関数 arima.sim のノイズはデフォルトでは標準正規列である
> y <- arima.sim(list(ar = a, ma = b), n)
> lines(y, col = "red")
```



(arma3.r)

8.2. (弱) 定常性と自己共分散・自己相関

8.2.1. (弱) 定常性. 確率過程 $X_t,\ t = 1, \ldots, T$ が次の 2 つの性質をもつとき, **(弱) 定常** *((weakly) stationary)* であるという:

(i) X_t の平均は時点 t によらない.
(ii) X_t と X_{t+h} の共分散は時点 t によらず時差 h のみで定まる. とくに, X_t の分散は時点 t によらない ($h = 0$ の場合を考えればよい).

定常でない確率過程は**非定常** *(non-stationary)* であるという. 前節で説明した確率過程の定常性は以下のようにまとめられる.

- **定常過程** ホワイトノイズ, MA モデル
- **非定常過程** トレンドのあるホワイトノイズ, ランダムウォーク
- **定常にも非定常にもなりうる確率過程** AR モデル, ARMA モデル

非定常過程は平均や分散といった基本的な統計量が時間によって変動してしまうため扱いが難しい. そのため, 非定常過程の分析の際には対数変換や階差をとる変換等によって定常過程とみなせるように変換したあと分析を実行することが多い (例えばランダムウォークは階差をとればホワイトノイズとなって定常過程となる).

8.2.2. 自己共分散・自己相関. $X_t,\ t = 1, \ldots, T$ が定常過程の場合, その定義から X_t と X_{t+h} の共分散は時点 t によらずラグ (時差) $h \geq 0$ のみで定まる. この共分散をラグ h での**自己共分散** *(autocovariance)* と呼ぶ. また, 分散も時点によらないので, X_t と X_{t+h} の相関も時点 t によらずラグ $h \geq 0$ のみで定まる. この相関をラグ h での**自己相関** *(autocorrelation)* と呼ぶ. 通常, ラグ h での自己共分散を観測データ $X_1, \ldots, X_T$ から推定するには, 標本自己共分散

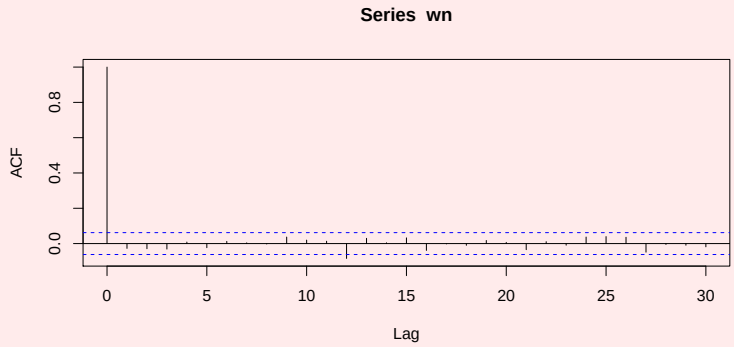
$$\frac{1}{T}\sum_{t=1}^{T-h}(X_t - \bar{X})(X_{t+h} - \bar{X})$$

を用いる. ただし, $\bar{X} = \frac{1}{T}\sum_{t=1}^{T} X_t$ は標本平均である. 同様に, ラグ h での自己相関を観測データ $X_1, \ldots, X_T$ から推定するには, 標本自己相関

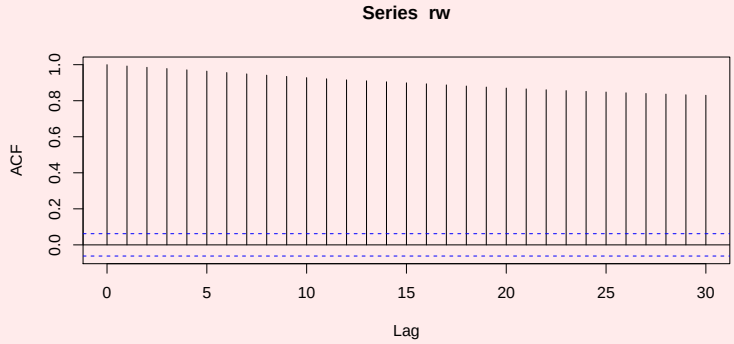
$$\frac{\sum_{t=1}^{T-h}(X_t - \bar{X})(X_{t+h} - \bar{X})}{\sum_{t=1}^{T}(X_t - \bar{X})^2}$$

を用いる. 自己共分散および自己相関は, 異なる時点間での観測データの従属関係を要約するための最も基本的な統計量である. R では関数 `acf()` によって計算できる.

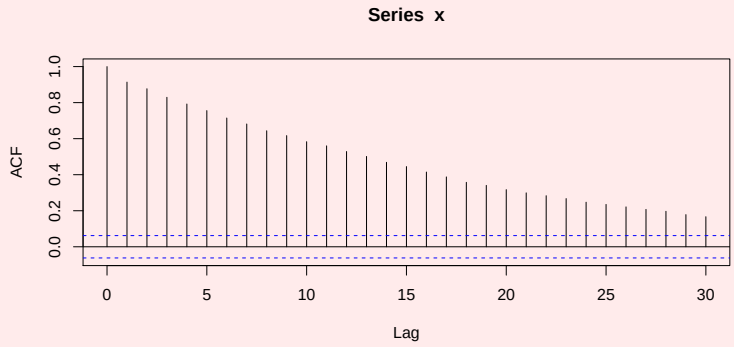
```
> ## acf の実行例
> ## 人工データによる例
> set.seed(123)
> n <- 1000 # サンプル数
> #1 ホワイトノイズ
> wn <- ts(rnorm(n))
> acf(wn) # 正のラグでほぼ 0
```



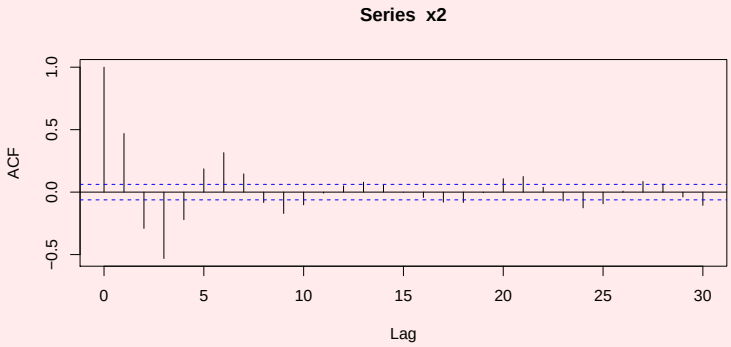

```
> #2 ランダムウォーク
> rw <- diffinv(rnorm(n-1)) # 初期値 0. diffinv は階差をとる関数 diff の逆関数
> acf(rw) # 自己相関がなかなか減衰しない
```



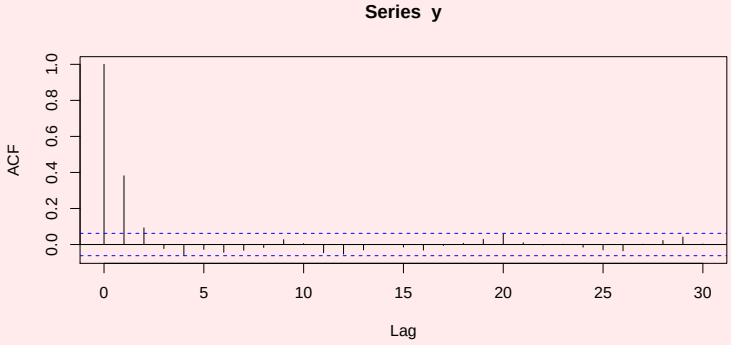

```
> #3 AR(2) モデル
> a <- c(0.669, 0.263) # AR の係数
> x <- arima.sim(list(ar = a), n)
> acf(x) # 自己相関は指数関数的に減衰する
```



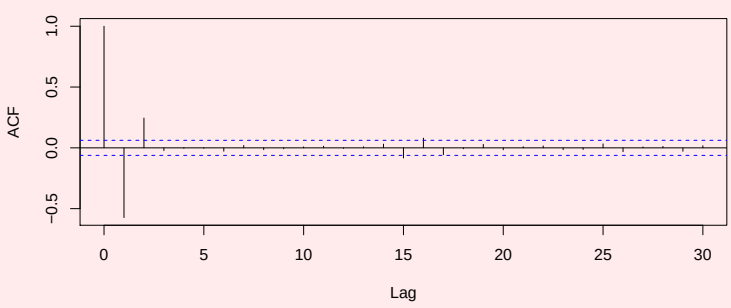
```
> a2 <- c(0.8, -0.64) # 別の係数も試してみる
> x2 <- arima.sim(list(ar = a2), n)
> acf(x2)
```



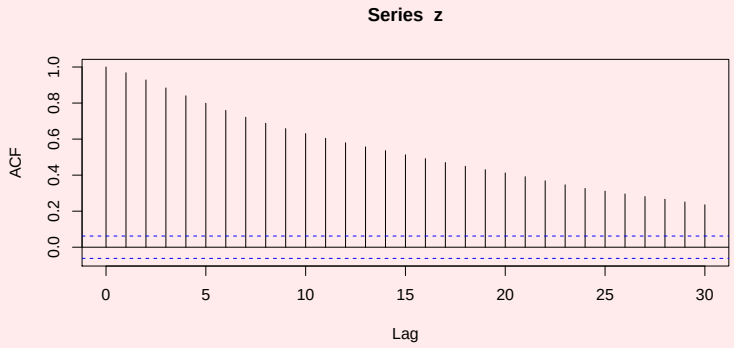

```
> #4 MA(2) モデル
> b <- c(0.438, 0.078) # MA の係数
> y <- arima.sim(list(ma = b), n)
> acf(y) # 次数より大きいラグの自己相関はほぼ 0
```



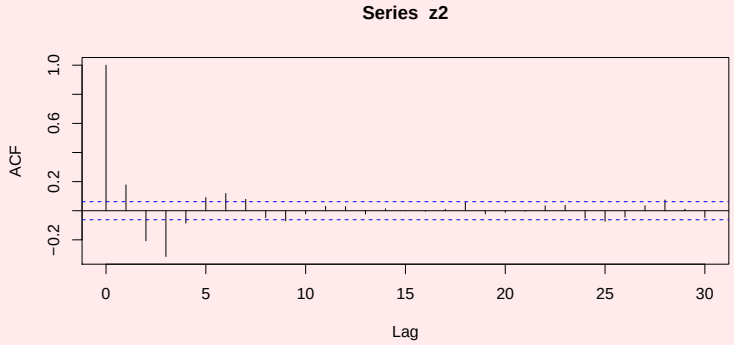

```
> b2 <- c(-0.6, 0.3) # 別の係数も試してみる
> y2 <- arima.sim(list(ma = b2), n)
> acf(y2)
```




```
> #5 ARMA(2,2) モデル
> z <- arima.sim(list(ar = a, ma = b), n)
> acf(z) # 自己相関は指数関数的に減衰する
```

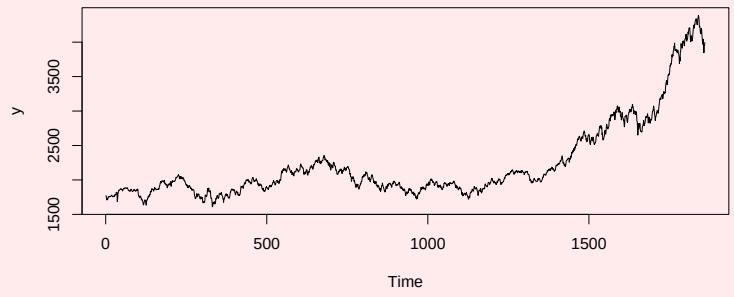



```
> z2 <- arima.sim(list(ar = a2, ma = b2), n) # 別の係数
> acf(z2)
```

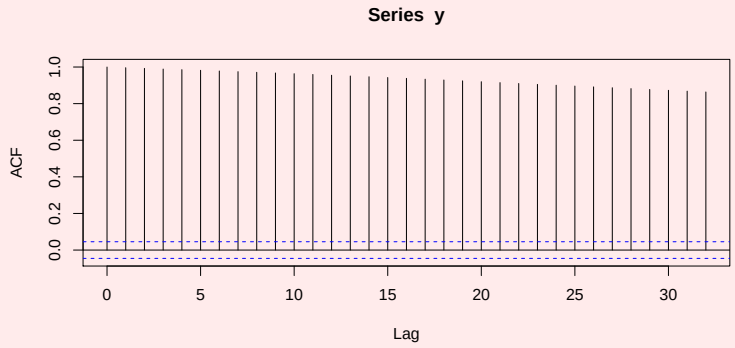


(acf2.r)

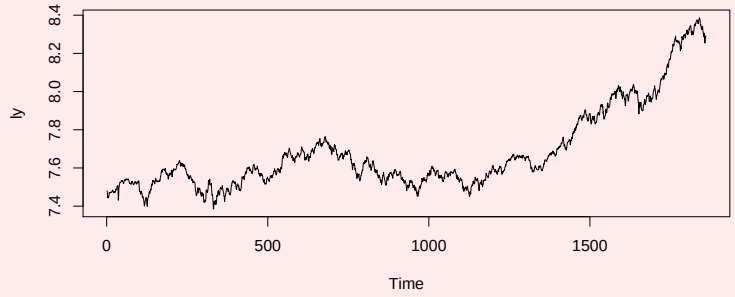
```
> ## データセット EuStockMarkets による例
> ## ヨーロッパ各国の株価指数の日次時系列データを集めたデータ
> y <- ts(EuStockMarkets[,3]) # France CAC を分析する
> plot(y)
```



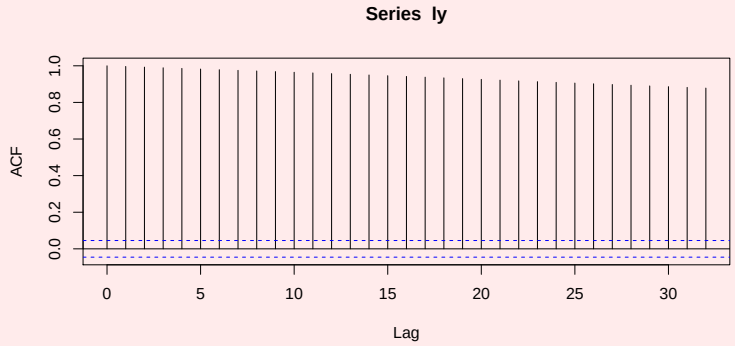

```
> acf(y) # 自己相関の計算
```



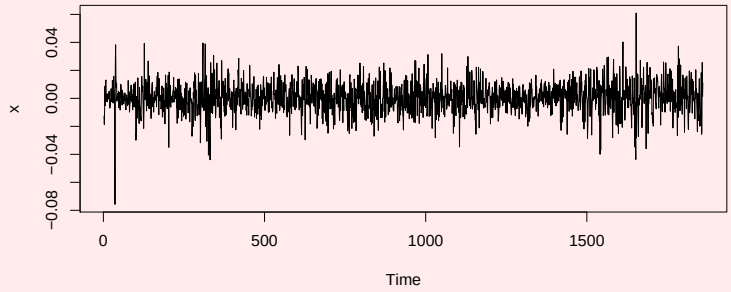

```
> ### y は指数関数のように増えているので log をとってみる
> ly <- log(y)
> plot(ly)
```



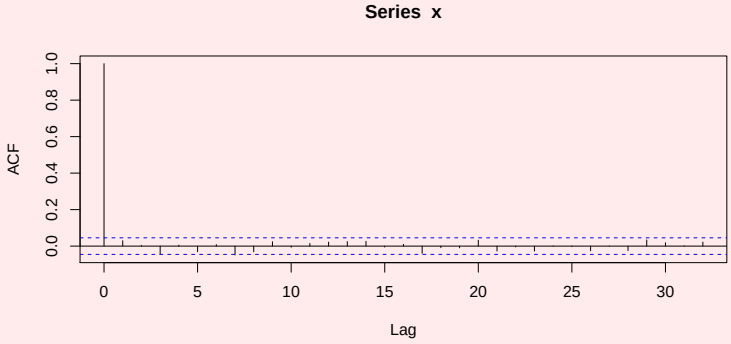

```
> acf(ly) # 自己相関の計算
```




```
> # 階差をとってみる
> x <- diff(ly)
> plot(x)
```




```
> acf(x) # 自己相関の計算
```




```
> ### ラグ 1 以上の自己相関がほぼなくなっており，ランダムウォーク的な
> ### 動きの時系列データであると予想される
> acf(x,plot=FALSE)$acf[,,1] # 自己相関の計算値の出力
 [1]  1.0000000000  0.0296846513  0.0033649283 -0.0454564783  0.0058038429
 [6] -0.0309941937  0.0082795741 -0.0487419482 -0.0313195150  0.0238469922
[11] -0.0064111378  0.0145865478  0.0220925246  0.0247725391  0.0263304230
[16] -0.0054466533  0.0097560901 -0.0397751661 -0.0086775234 -0.0095492995
[21]  0.0324935476 -0.0270460521 -0.0045434243 -0.0272122540  0.0006845900
[26] -0.0026809536 -0.0260768140 -0.0006207087 -0.0247566070  0.0338711657
[31]  0.0186643921  0.0015907985  0.0223137502
```

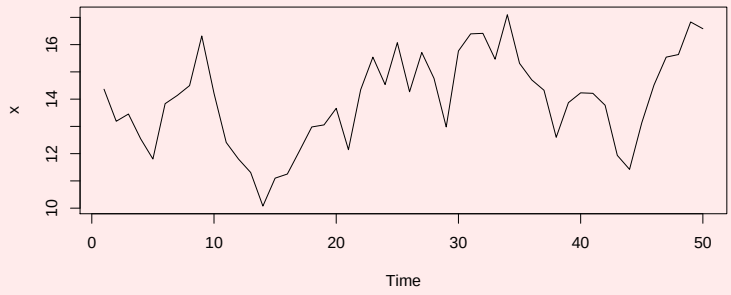
(eustock.r)

8.3. AR モデルのあてはめ

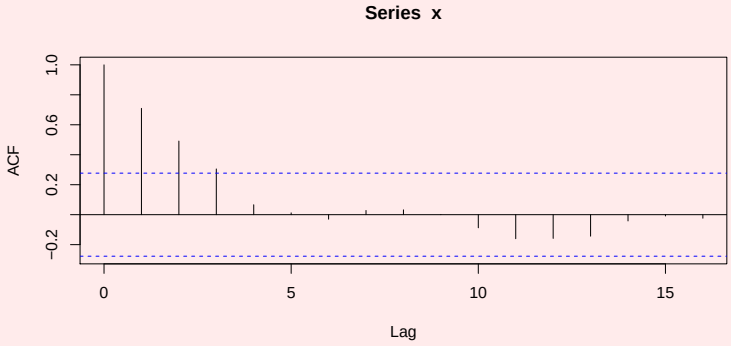
R には, 与えられた時系列データに適切な定常 AR モデルをあてはめるための関数 ar( ) が用意されている. 使い方は以下の実行例を参照すること.

```
> ## 関数 ar の使い方
> ## 実行例 1： 人工データ
> ## 前々節でシミュレートした AR モデルを利用する
> set.seed(123)
> n <- 1000 # データ数
> a <- c(0.669, 0.263) # AR の係数
> x <- arima.sim(list(ar = a), n)
> p.max <- 10 # あてはめの候補とする AR モデルの最大次数
> ar(x, aic=TRUE, order.max=p.max, method="yule-walker")
Call:
ar(x = x, aic = TRUE, order.max = p.max, method = "yule-walker")
```

```
Coefficients:
     1       2
0.6420  0.2812

Order selected 2  sigma^2 estimated as  1.016
> ### 推定されたモデル: AR(2), a1=0.6420, a2=0.2812
>
>
> # 実行例２：Monthly Lake Erie Levels 1921 - 1970
> # Hyndman, R.J. “Time Series Data Library”, http://data.is/TSDLdemo.
> # Accessed on <monthly-lake-erie-levels-1921-1970>
> ldata <- read.csv(file="monthly-lake-erie-levels-1921-1970.csv",row.names=1)
> ldata1 <- ldata$ Monthly.Lake.Erie.Levels.1921...1970.
> length(ldata1)
[1] 602
> x <-  ts(ldata1[seq(12,600, by =12)]) # data of December
> plot(x)
```




```
> acf(x)
> # pacf(x)
```




```
> (arfit <- ar(x, aic=TRUE, order.max=10, method=c("yule-walker")))
Call:
ar(x = x, aic = TRUE, order.max = 10, method = c("yule-walker"))

Coefficients:
     1
0.7097

Order selected 1  sigma^2 estimated as  1.479
> (ar(x, aic=TRUE, order.max=10, method=c("mle"))) # 結果がすこし異なる．推定法が異なる．
```

```
Call:
ar(x = x, aic = TRUE, order.max = 10, method = c("mle"))

Coefficients:
     1
0.7314

Order selected 1  sigma^2 estimated as  1.344
```
(ar-estimation.r)

詳細は割愛するが, 関数 ar() において, AR モデルの次数は AIC(Akaike Information Criterion, 赤池情報量規準) と呼ばれるモデルのあてはまりの良さを表す指標が最も小さくなるように選ばれている.

8.4. ARMA モデルのあてはめ

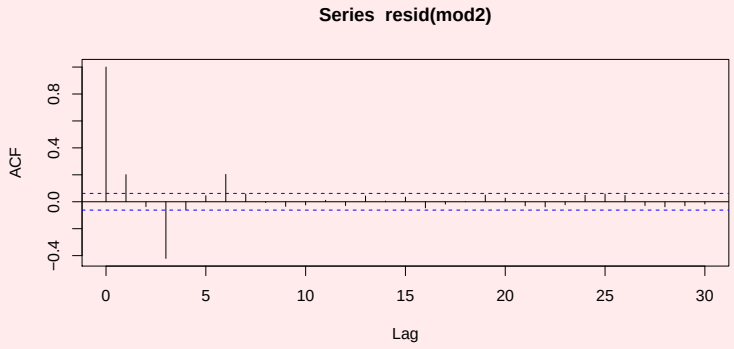
指定された次数の定常 ARMA モデルを与えられた時系列データにあてはめるための関数として arima() が用意されている. ただし, 関数 ar() と異なり関数 arima() には適切な次数を決定する機能は備わっていない. そのため, 次数の決定は試行錯誤で行うか, パッケージ forecast の funauto.arima を利用するとよい.

```
> ## 関数 arima の使い方
> ## 実行例 1 : 人工データ
> ## 前々節でシミュレートした ARMA モデルを利用する
> set.seed(123)
> n <- 1000 # データ数
> a <- c(0.8, -0.64) # AR の係数
> b <- -0.5 # MA の係数
> y <- arima.sim(list(ar = a, ma = b), n)
> (mod1 <- arima(y, order = c(2, 0, 1))) # ARMA(2,1) モデルのあてはめ
Call:
arima(x = y, order = c(2, 0, 1))

Coefficients:
         ar1      ar2      ma1  intercept
      0.7988  -0.6403  -0.5316     0.0157
s.e.  0.0345   0.0243   0.0423     0.0177

sigma^2 estimated as 1.002:  log likelihood = -1420.56,  aic = 2851.13
> # モデルのあてはまり具合を確認するため残差の自己相関をみる
> # あてはまりがよければ残差はほぼホワイトノイズになるはず
> acf(resid(mod1))
```


```
> (mod2 <- arima(y, order = c(0, 0, 2))) # MA(2) モデルのあてはめ
```

```
Call:
arima(x = y, order = c(0, 0, 2))

Coefficients:
         ma1      ma2  intercept
      0.0452  -0.5171     0.0159
s.e.  0.0303   0.0297     0.0190

sigma^2 estimated as 1.295:  log likelihood = -1548.62,  aic = 3105.24
```

```
> acf(resid(mod2)) # あてはまりはよくない
```

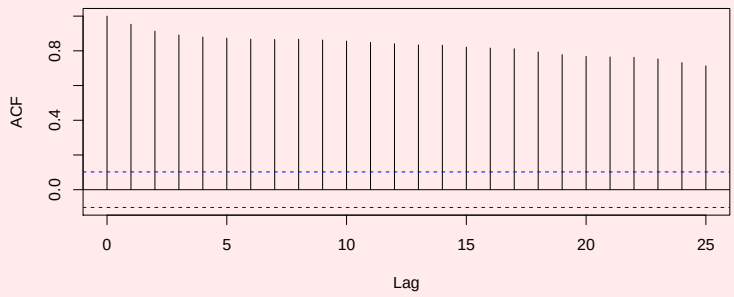



```
> # AIC最小化によって次数選択をする方法
> # パッケージ forecast の関数 auto.arima を使う
> # install.packages("forecast") # パッケージのインストール (必要な場合)
> library(forecast) # パッケージのロード
> (mod3 <- auto.arima(y, d = 0, D = 0))
```

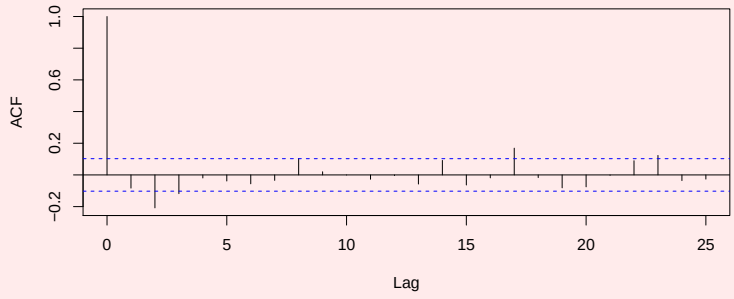
```
Series: y
ARIMA(3,0,1) with zero mean

Coefficients:
         ar1      ar2     ar3      ma1
      0.8604  -0.6779  0.0495  -0.5827
s.e.  0.0770   0.0499  0.0568   0.0691

sigma^2 estimated as 1.006:  log likelihood=-1420.58
AIC=2851.17   AICc=2851.23   BIC=2875.71
```

```
> # d は原系列の差分を何回とるか決めるパラメーター
> # D は原系列の周期性 (季節性) を取り除くためにとる差分の回数
> # ARMA(3,1) モデルが選ばれる
> acf(resid(mod3))
> # あてはまりは悪くない
> # ARMA(3,1) は真のモデル ARMA(2,1) を含むモデル (過剰なモデル)
> # なので, これはおかしくない
```


```
> ## 実行例２：気候データ
> # データの読み込み
> kikou <- read.csv("kikou2016.csv", fileEncoding = "sjis")
> x <- ts(kikou$気温)
> acf(x) # 減衰が遅いので，差分をとった方が良さそう
```

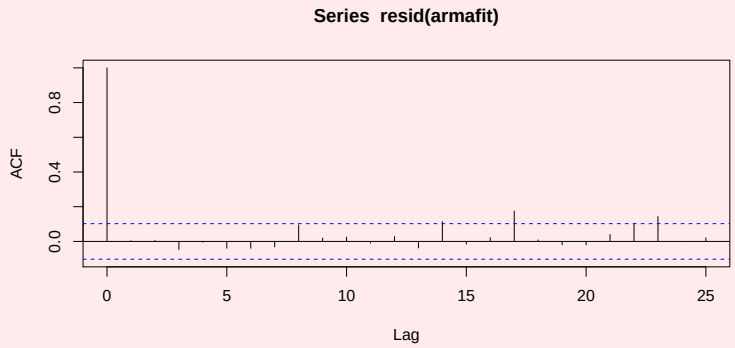



```
> acf(diff(x)) # 定常に見える
```




```
> # 差分系列に ARMA モデルをあてはめる (ARMA(1,2) が選ばれる)
> (armafit <- auto.arima(x, d = 1, D = 0))
Series: x
ARIMA(1,1,2)

Coefficients:
         ar1      ma1      ma2
      0.3732  -0.5901  -0.2147
s.e.  0.1003   0.0996   0.0699
```

```
sigma^2 estimated as 4.364:  log likelihood=-785.56
AIC=1579.13   AICc=1579.24   BIC=1594.73
> acf(resid(armafit)) # そこそこあてはまりはよさそう
```



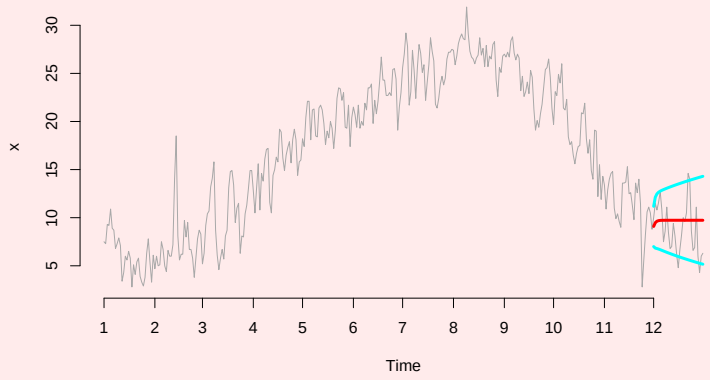
(arma-estimation.r)

8.5. 予測

推定されたモデルを使って数期先の時系列データの値を予測するには関数 predict() を使う. n 期先までのデータを予測する場合, オプション n.ahead に n を指定すればよい.

```
> ## 気候データによる例
> ## 11 月までの気温から 12 月の気温を予測する
> kikou <- read.csv("kikou2016.csv", fileEncoding = "sjis")
> x <- ts(kikou$気温) # 気温データを時系列データに変換
> x.in <- x[kikou$月%in%1:11] # 訓練データ
> x.out <- x[kikou$月==12] # テストデータ
> (armafit <- auto.arima(x.in, d = 1, D = 0)) # 差分系列に ARMA モデルをあてはめる
Series: x.in
ARIMA(1,1,2)

Coefficients:
         ar1      ma1      ma2
      0.3914  -0.6192  -0.1868
s.e.  0.1060   0.1063   0.0744

sigma^2 estimated as 4.38:  log likelihood=-719.33
AIC=1446.65   AICc=1446.77   BIC=1461.9
> (p <- predict(armafit, n.ahead=length(x.out))) # 12 月の予測
$pred
Time Series:
Start = 336
End = 366
Frequency = 1
 [1] 9.089694 9.480413 9.633345 9.693205 9.716635 9.725806 9.729395 9.730800
 [9] 9.731350 9.731566 9.731650 9.731683 9.731696 9.731701 9.731703 9.731704
[17] 9.731704 9.731704 9.731704 9.731704 9.731704 9.731704 9.731704 9.731704
[25] 9.731704 9.731704 9.731704 9.731704 9.731704 9.731704 9.731704

$se
Time Series:
Start = 336
End = 366
Frequency = 1
```

```
 [1] 2.092874 2.644295 2.840959 2.954876 3.042294 3.119433 3.191812 3.261490
 [9] 3.329293 3.395582 3.460540 3.524278 3.586874 3.648393 3.708890 3.768416
[17] 3.827016 3.884731 3.941602 3.997664 4.052950 4.107492 4.161320 4.214460
[25] 4.266938 4.318779 4.370005 4.420637 4.470695 4.520200 4.569168
> signif(p$pred, digits = 2) # 予測値 (小数第二位以下は四捨五入)
Time Series:
Start = 336
End = 366
Frequency = 1
 [1] 9.1 9.5 9.6 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7
[20] 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7 9.7
> x.out # 実績値
 [1] 10.0 11.5 10.8 11.6 12.7 10.8  7.5  8.6 11.1  8.5  6.8  7.0  9.4  8.1  6.2
[16]  4.8  6.7  8.2 10.0  9.6 10.5 14.6 14.0  8.6  6.6  6.9 11.1  6.0  4.3  6.0
[31]  6.3
> # 可視化
> plot(x, col = "darkgray", axes = FALSE)
> axis(1, which(kikou$日==1), 1:12) # x 軸の作成
> axis(2) # y 軸の作成
> lines(p$pred, col = "red", lwd = 3)
> lines(p$pred+p$se,col="cyan", lwd = 3) # 予測の誤差 (1 sigma)
> lines(p$pred-p$se,col="cyan", lwd = 3)
```



(ts-predict.r)

8.6. 参考文献

1. 田中勝人著「現代時系列分析」, 岩波書店 (2006 年).