# Time Series: A First Course with Bootstrap Starter

## Contents

# Lesson 11-1: Nonlinear Processes

- We discuss various types of nonlinear processes.

## Paradigm 11.1.1. Linear, Causal, and Gaussian Processes

- $\{X_t\}$ is Gaussian if all its finite-dimensional marginal distributions are multivariate Gaussian.
- The best one-step ahead predictor for a Gaussian time series is a *linear* function of the sample.
- More broadly, a time series is *linear* if it can be written

$$X_t = \sum_{j=-\infty}^{\infty} \psi_j \epsilon_{t-j}$$

  for $\{\epsilon_t\}$ i.i.d.
- Some linear time series are *causal*, where $\psi(z)$ is a power series.
- Some causal linear time series are *invertible*, where roots of $\psi(z)$ are outside the unit circle.
- A stationary Gaussian time series with positive spectral density is linear, causal, and invertible.

## Example 11.1.2. Linear but Non-Gaussian AR($p$)

- Consider a non-Gaussian AR($p$) process:

$$X_t = \sum_{j=1}^{p} \pi_j X_{t-j} + \epsilon_t$$

  for $\{\epsilon_t\}$ i.i.d. (but not Gaussian).
- The best one-step ahead predictor is still linear: supposing that $t > p$,

$$\mathbb{E}[X_t | \underline{X}_{t-1}] = \sum_{j=1}^{p} \pi_j \mathbb{E}[X_{t-j} | \underline{X}_{t-1}] + \mathbb{E}[\epsilon_t | \underline{X}_{t-1}] = \sum_{j=1}^{p} \pi_j X_{t-j}.$$

## Example 11.1.5. Nonlinear Autoregression

- We can generalize the regression in Example 11.1.2 to be a nonlinear function of past data.
- An order one nonlinear autoregression has the form

$$X_t = g(X_{t-1}) + Z_t,$$

  where $\{Z_t\}$ is i.i.d.
- E.g., $g(x) = \exp(x)$.
- The best one-step ahead predictor is

$$\mathbb{E}[X_t | \underline{X}_{t-1}] = g(X_{t-1}).$$

## Example 11.1.6. Autoregressive Conditional Heteroscedasticity

- $\{X_t\}$ is an AutoRegressive Conditional Heteroscedastic (ARCH) process of order $p$ if

$$X_t = \sigma_t Z_t$$
$$\sigma_t^2 = a_0 + \sum_{j=1}^{p} a_j X_{t-j}^2,$$

  where $\{Z_t\}$ is i.i.d. $(0, 1)$, and $a_j \geq 0$.
- Then $Z_t$ is independent of past values of the process, and $\{X_t^2\}$ is a nonlinear process whose best predictor is linear (shown in Theorem 11.3.6 below).

## Example 11.1.7. Prediction of Lognormal Time Series

- The lognormal times series $\{X_t\}$ is defined by $X_t = \exp Y_t$, where $\{Y_t\}$ is Gaussian.
- Say $\{Y_t\}$ is mean zero with ACVF $\gamma_Y(h)$. Then $\mathbb{E}[X_t] = \exp\{\gamma_Y(0)/2\}$ and

$$\gamma_X(h) = \exp\{\gamma_Y(0)\}\,(\exp\{\gamma_Y(h)\} - 1).$$

- Suppose that $\sigma^2$ is the asymptotic prediction variance of $Y_t$, and $\widehat{Y}_{t+1}$ is the best predictor of $Y_{t+1}$. Then

$$\widehat{X}_{t+1} = \exp\{\widehat{Y}_{t+1} + \sigma^2/2\},$$

and the nonlinear prediction error variance is

$$\exp\{2\gamma_Y(0)\} \cdot (1 - \exp\{-\sigma^2\}).$$

## Exercise 11.3. Lognormal Time Series Prediction Error

- We numerically compute the linear and nonlinear prediction error variances.
- The spectral density of $\{X_t\}$ is

$$f(\lambda) = \exp\{\gamma_Y(0)\} \sum_{h=-\infty}^{\infty} (\exp\{\gamma_Y(h)\} - 1)\, e^{-i\lambda h}.$$

- So the best linear prediction error variance is $\exp\{\langle \log f \rangle_0\}$.
- We consider the case where $\{Y_t\}$ is a Gaussian AR(1) of parameter $\phi = .8$ and $\sigma^2 = 1$.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

ARMAauto <- function(phi,theta,maxlag)
{
    p <- length(phi)
    q <- length(theta)
    gamMA <- polymul(c(1,theta),rev(c(1,theta)))
    gamMA <- gamMA[(q+1):(2*q+1)]
    if (p > 0)
    {
        Amat <- matrix(0,nrow=(p+1),ncol=(2*p+1))
        for(i in 1:(p+1))
        {
            Amat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
        Amat <- cbind(Amat[,(p+1)],as.matrix(Amat[,(p+2):(2*p+1)]) +
            t(matrix(apply(t(matrix(Amat[,1:p],p+1,p)),2,rev),p,p+1)))
        Bmat <- matrix(0,nrow=(q+1),ncol=(p+q+1))
        for(i in 1:(q+1))
        {
            Bmat[i,i:(i+p)] <- c(-1*rev(phi),1)
        }
```

```
        Bmat <- t(matrix(apply(t(Bmat),2,rev),p+q+1,q+1))
        Bmat <- matrix(apply(Bmat,2,rev),q+1,p+q+1)
        Bmat <- Bmat[,1:(q+1)]
        Binv <- solve(Bmat)
        gamMix <- Binv %*% gamMA
        if (p <= q) { gamMix <- matrix(gamMix[1:(p+1),],p+1,1)
            } else gamMix <- matrix(c(gamMix,rep(0,(p-q))),p+1,1)
        gamARMA <- solve(Amat) %*% gamMix
    } else gamARMA <- gamMA[1]

    gamMA <- as.vector(gamMA)
    if (maxlag <= q) gamMA <- gamMA[1:(maxlag+1)] else gamMA <- c(gamMA,rep(0,(maxlag-q)))
    gamARMA <- as.vector(gamARMA)
    if (maxlag <= p) gamARMA <- gamARMA[1:(maxlag+1)] else {
    for(k in 1:(maxlag-p))
    {
        len <- length(gamARMA)
        acf <- gamMA[p+1+k]
        if (p > 0) acf <- acf + sum(phi*rev(gamARMA[(len-p+1):len]))
        gamARMA <- c(gamARMA,acf)
    } }
    return(gamARMA)
}

phi <- .8
sigma <- 1
gamma <- ARMAauto(phi,NULL,100)*sigma^2
mesh <- 1000
lambda <- pi*seq(0,mesh)/mesh
f.spec <- (exp(gamma[1])-1)*cos(0*lambda)
for(h in 1:100)
{
    f.spec <- f.spec + 2*(exp(gamma[h+1])-1)*cos(h*lambda)
}
f.spec <- exp( gamma[1] ) * f.spec
lin.pred.mse <- exp( mean(log(f.spec)) )
nonlin.pred.mse <- exp(2*gamma[1])*(1- exp(-sigma^2))
print(c(lin.pred.mse,nonlin.pred.mse))
```

```
## [1] 170.0569 163.5110
```

- Notice that the nonlinear predictor has lower MSE than the linear predictor.

# Lesson 11-3: ARCH Process

- For financial data, we need models that explain *fat tails* and *volatility clustering*.
- The ARCH model is a popular nonlinear time series model for financial data.

### Remark 11.3.1. Volatility Clustering

- Financial time series exhibit periods of high variability followed by low variability.
- In intra-day trading data, volatility is higher at the beginning and end of the day.

### Proposition 11.3.2.

- Let $\{X_t\}$ be an ARCH($p$) process.
- It has mean zero, and conditional variance

$$\mathrm{Var}[X_t|X_{t-1:}] = \sigma_t^2,$$

  where $X_{t-1:}$ is shorthand for $X_{t-1}, X_{t-2}, \ldots$.
- The conditional variance $\sigma_t^2$ is called the *volatility*.

### Theorem 11.3.6.

- Let $\{X_t\}$ be an ARCH($p$) process such that $\delta = \sum_{j=1}^{p} a_j < 1$.
- Then $\{X_t\}$ is white noise with variance $a_0/(1-\delta)$.
- If $\mathbb{E}[Z_t^4] < \infty$ and $\delta < \mathbb{E}[Z_t^4]^{-1/2}$, then $\{X_t^2\}$ is a causal AR($p$) with respect to white noise inputs $\epsilon_t$.
- Since the innovations $\epsilon_t$ are a dependent white noise, it can be shown that $\{X_t^2\}$ is a nonlinear process.

### Remark 11.3.8. The Market Efficiency Axiom

- We can predict $X_t^2$ using its AR representation.
- But $\mathbb{E}[X_t|X_{t-1:}] = 0$.
- So we can say something about the magnitude of future returns, but not the direction.
- Knowing future values of log returns would permit an arbitrage opportunity.
- The *market efficiency axiom* states that there are no arbitrage opportunities in an efficient market.

### Paradigm 11.3.9. Fitting an ARCH($p$) Model

- We construct the likelihood of the ARCH model, based on the marginal distribution of $Z_t$.
- Given the past data $x_{t-1}, x_{t-2}, \ldots$, we compute

$$s_t = \sqrt{a_0 + \sum_{j=1}^{p} a_j x_{t-j}^2}.$$

- Hence $p_Z(x_t/s_t)/s_t$ is the density for $X_t$ given $X_{t-1:}$.
- The *pseudo-likelihood* is obtained by omitting some initial values. Its log is

$$\sum_{t=p+1}^{n} \left[\log p_Z(x_t/s_t) - \log s_t\right].$$

- This is a function of the parameters $a_0, a_1, \ldots, a_p$, so we can numerically optimize.

### Example 11.3.10. Fitting a Gaussian ARCH($p$) to Dow Log Returns

- We illustrate the fitting of an ARCH model to the Dow log returns data.
- We first load the data and fit an AR($p$) to the squares via Ordinary Least Squares, using AIC to select model order $p$.

```
dow <- read.table("dow.dat")
dow <- diff(log(dow[,1]))
dow <- ts(dow,start=c(2008,164),frequency=260)
dow.ols <- ar.ols(dow^2)
p <- dow.ols$order
print(p)
```

```
## [1] 33
```

- Then we utilize the pseudo-likelihood based on a Gaussian marginal distribution.
- This uses a reparameterization of the ARCH parameters, to ensure we get values in $[0, 1]$ that satisfy the constraint that $\delta < 1$.

```
psi2arch <- function(psi)
{
    p <- length(psi)-1
    a.0 <- exp(psi[1])
    if(p > 0)
    {
        r <- (1 + exp(-psi[2]))^(-1)
        if(p > 1)
        {
            a.1 <- (1 + sum(exp(-psi[3:(p+1)])))^(-1)
            a.j <- a.1
            for(j in 2:p)
            {
                a.j <- c(a.j,exp(-psi[j+1])*a.1)
            }
            a.j <- r*a.j
        } else { a.j <- r }
        a.0 <- c(a.0,a.j)
    }
    return(a.0)
}

lik.arch <- function(psi,data,df)
{
    p <- length(psi)-1
    T <- length(data)
    arch <- psi2arch(psi)

    lik <- 0
    for(t in (p+1):T)
    {
        arch.sd <- sqrt(arch[1] + sum(arch[-1]*data[(t-1):(t-p)]^2))
        if(df==Inf) { lik <- lik + (-2)*log(dnorm(data[t],sd=arch.sd)) } else {
            lik <- lik + (-2)*log(dt(data[t]/arch.sd,df=df)/arch.sd) }
    }
    return(lik)
}
```

- We initialize and run the optimization.

```
psi.init <- rep(0,p+1)
arch.fit <- optim(psi.init,lik.arch,data=dow,df=Inf,method="BFGS")
print(arch.fit)
```

```
## $par
##  [1] -2033.2920745    63.3655168    -1.0359247    -0.9982537    -0.6378827
##  [6]    -0.8151557    -0.8403049    -0.4084173    -0.3525442    -0.1679610
## [11]    -0.2513546     2.2654488     9.8935678     2.6956507     0.8310226
## [16]    10.4503981     6.3194832     2.5990406     1.7512673     9.4247571
## [21]    10.7428642     3.0778047     0.1038050     0.4003737    11.7614792
## [26]    11.8914237     2.1830580     9.0794927     0.3500038     0.2858920
```

```
## [31]     0.4896008    11.3106561    10.6978804    -0.2668674
##
## $value
## [1] -12517.6
##
## $counts
## function gradient
##       79       47
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
arch.par <- psi2arch(arch.fit$par)
print(round(arch.par,digits=3))
```

```
##  [1] 0.000 0.041 0.116 0.111 0.078 0.093 0.095 0.062 0.058 0.049 0.053 0.004
## [13] 0.000 0.003 0.018 0.000 0.000 0.003 0.007 0.000 0.000 0.002 0.037 0.028
## [25] 0.000 0.000 0.005 0.000 0.029 0.031 0.025 0.000 0.000 0.054
```

# Lesson 11-4: GARCH Processes

- We explore the GARCH process, a generalization of the ARCH process.

### Definition 11.4.1.

- The *Generalized Autoregressive Conditionally Heteroscedastic* process of order $p, q$, or GARCH(p,q), is a process $\{X_t\}$ defined via

$$X_t = \sigma_t Z_t$$
$$\sigma_t^2 = a_0 + \sum_{j=1}^{p} a_j X_{t-j}^2 + \sum_{k=1}^{q} b_k \sigma_{t-k}^2.$$

- The $\{Z_t\}$ inputs are i.i.d. mean zero, variance 1.
- The parameters satisfy $a_j \geq 0$, $b_k \geq 0$.

### Theorem 11.4.3.

- Let $\{X_t\}$ be a GARCH(p,q) process, and set $\delta = \sum_{j=1}^{p} a_j + \sum_{k=1}^{q} b_k$.
- If $\delta < 1$, then $\{X_t\}$ is strictly stationary with finite variance and mean zero.
- Also $\{X_t\}$ is a white noise with variance $a_0/(1-\delta)$.
- If $\mathbb{E}[Z_t^4] < \infty$ and $\delta < \mathbb{E}[Z_t^4]^{-1/2}$, then $\{X_t^2\}$ is an ARMA($p*$,$q$) with respect to white noise inputs $\epsilon_t$, and $p* = \max\{p, q\}$.

### Example 11.4.4. GARCH(1,1)

- Set $p = q = 1$ for the most popular GARCH specification.

$$\sigma_t^2 = a_0 + a_1 X_{t-1}^2 + b_1 \sigma_{t-1}^2.$$

- We can solve recursively for the volatility $\sigma_t^2$:

$$\sigma_t^2 = \frac{a_0}{1-b_1} + a_1 \sum_{j=0}^{\infty} b_1^j X_{t-1-j}^2.$$

- This is called the ARCH($\infty$) representation of the GARCH(1,1).

### Corollary 11.4.5.

- Let $\{X_t\}$ be a GARCH(p,q) process with $\delta < 1$, such that $\mathbb{E}[Z_t^4] < \infty$ and $\delta < \mathbb{E}[Z_t^4]^{-1/2}$.
- Let $\theta(z) = 1 - \sum_{k=1}^{q} b_k z^k$ and $\phi(z) = 1 - \sum_{j=1}^{p*}(a_j + b_j)z^j$.
- Then the best predictor of $X_t^2$ is linear:

$$\mathbb{E}[X_t^2 | X_{t-1:}] = \frac{a_0}{\theta(1)} + \sum_{k=1}^{\infty} \pi_k X_{t-k}^2,$$

where $\pi(z) = \phi(z)/\theta(z) = 1 - \sum_{k=1}^{\infty} \pi_k z^k$.

### Remark 11.4.6. Fitting a GARCH Model

- Note that $Z_t = X_t/\sigma_t$, which is i.i.d. with some given probability density function $p_Z$.
- From a sample, we have realization $x_t$ and compute

$$s_t^2 = \frac{a_0}{1 - b_1} + a_1 \sum_{j=0}^{\infty} b_1^j x_{t-1-j}^2,$$

truncating the sum where the sample begins.
- Then the *pseudo-likelihood* is defined as

$$\sum_{t=2}^{n} \left( \log p_Z(x_t/s_t) - \log s_t \right),$$

where we truncate the $s_t$ calculation as needed.

- This pseudo-likelihood is a function of $a_0, a_1, b_1$, and can be maximized.

### Example 11.4.7. Fitting a Gaussian GARCH(1,1) to Dow Log Returns

- We consider the Dow Log Returns, and fit a Gaussian GARCH(1,1) model.

```
dow <- read.table("dow.dat")
dow <- diff(log(dow[,1]))
dow <- ts(dow,start=c(2008,164),frequency=252)
```

- Then we utilize the pseudo-likelihood based on a Gaussian marginal distribution.
- This uses a reparameterization of the GARCH parameters, to ensure we get values in $[0, 1]$ that satisfy the constraint that $\delta < 1$.

```
polymul <- function(a,b)
{
    bb <- c(b,rep(0,length(a)-1))
    B <- toeplitz(bb)
    B[lower.tri(B)] <- 0
    aa <- rev(c(a,rep(0,length(b)-1)))
    prod <- B %*% matrix(aa,length(aa),1)
    return(rev(prod[,1]))
}

psi2arch <- function(psi)
{
```

```
    p <- length(psi)-1
    a.0 <- exp(psi[1])
    if(p > 0)
    {
        r <- (1 + exp(-psi[2]))^(-1)
        if(p > 1)
        {
            a.1 <- (1 + sum(exp(-psi[3:(p+1)])))^(-1)
            a.j <- a.1
            for(j in 2:p)
            {
                a.j <- c(a.j,exp(-psi[j+1])*a.1)
            }
            a.j <- r*a.j
        } else { a.j <- r }
        a.0 <- c(a.0,a.j)
    }
    return(a.0)
}

lik.garch <- function(psi,data,p.order,df,fitting=TRUE)
{

    p <- p.order
    q <- length(psi)-p-1
    T <- length(data)
    coef <- psi2arch(psi)
    acoef <- coef[1:(p+1)]
    bcoef <- NULL
    if(q > 0) { bcoef <- coef[(p+2):(p+q+1)] }
    theta <- bcoef
    phi <- acoef[-1]
    psi <- c(1,ARMAtoMA(ar=theta,ma=NULL,lag.max=T))
    psi <- polymul(psi,phi)

    lik <- 0
    resids <- NULL
    for(t in (p+1):T)
    {
        new.sigt <- sqrt(acoef[1]/(1-sum(theta)) + sum(psi[1:(t-1)]*data[(t-1):1]^2))
        if(df==Inf) { lik <- lik + (-2)*log(dnorm(data[t],sd=new.sigt)) } else {
            lik <- lik + (-2)*log(dt(data[t]/new.sigt,df=df)/new.sigt) }
        resids <- c(resids,data[t]/new.sigt)
    }
    if(fitting) { return(lik) } else { return(resids) }
}
```

- We initialize and run the optimization.

```
p <- 1
q <- 1
psi.init <- rep(0,p+q+1)
garch.fit <- optim(psi.init,fn=lik.garch,data=dow,p.order=p,df=Inf,method="BFGS")
print(garch.fit)
```

```
## $par
## [1] -60.497149  13.003395  -2.306336
##
## $value
## [1] -12716.67
##
## $counts
## function gradient
##       48       22
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
garch.par <- psi2arch(garch.fit$par)
print(round(garch.par,digits=3))
```

```
## [1] 0.000 0.091 0.909
```

- The estimated parameters are: $a_0$ is $5.3262535 \times 10^{-27}$, $a_1$ is 0.0905994, and $b_1$ is 0.9093984.
- There is a high degree of persistence in volatility due to the high value of $b_1$.

```
z <- lik.garch(garch.fit$par,dow,p,Inf,FALSE)
m <- length(z)
kurt <- m*sum((z - mean(z))^4)/(sum((z - mean(z))^2))^2 - 3
```

- Also $a_1 + b_1 \approx 1$, indicating high variance. We can check the residuals' kurtosis: it is 1.3230191. This high value indicates that the residuals have fat tails.

### Example 11.4.8. Fat-Tailed GARCH(1,1) for Dow Log Returns

- The above fitted model assumes a Gaussian input, but the variance was large.
- So we should try a fat-tailed distribution. Consider Student t.

```
lik.tgarch <- function(psi,data,p.order)
{

    p <- p.order
    q <- length(psi)-p-2
    T <- length(data)
    df <- 2 + exp(psi[p+q+2])
    coef <- psi2arch(psi[1:(p+q+1)])
    acoef <- coef[1:(p+1)]
    bcoef <- NULL
    if(q > 0) { bcoef <- coef[(p+2):(p+q+1)] }
    theta <- bcoef
    phi <- acoef[-1]
    psi <- c(1,ARMAtoMA(ar=theta,ma=NULL,lag.max=T))
    psi <- polymul(psi,phi)

    lik <- 0
    for(t in (p+1):T)
    {
        new.sigt <- sqrt(acoef[1]/(1-sum(theta)) + sum(psi[1:(t-1)]*data[(t-1):1]^2))
        new.sigt <- new.sigt/sqrt(df/(df-2))
```

```
        lik <- lik + (-2)*log(dt(data[t]/new.sigt,df=df)/new.sigt)
    }
    return(lik)
}
```

- We initialize and run the optimization.

```
p <- 1
q <- 1
# initialize with values close to those of the Gaussian GARCH
psi.init <- c(garch.fit$par,1)
tgarch.fit <- optim(psi.init,fn=lik.tgarch,data=dow,p.order=p,method="BFGS")
print(tgarch.fit)
```

```
## $par
## [1] -60.497149  13.020642  -2.167761   1.630332
##
## $value
## [1] -12816.1
##
## $counts
## function gradient
##       23        5
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```
tgarch.par <- c(psi2arch(tgarch.fit$par[1:(p+q+1)]),2+exp(tgarch.fit$par[p+q+2]))
print(round(tgarch.par,digits=3))
```

```
## [1] 0.000 0.103 0.897 7.106
```

- The estimated parameters are: $a_0$ is $5.3262535 \times 10^{-27}$, $a_1$ is 0.102683, and $b_1$ is 0.8973148. The degrees of freedom is 7.1055719.
- We still have $a_1 + b_1 \approx 1$, indicating high variance.

# Lesson 11-5: Bi-spectral Density

- Any Gaussian process is linear, and the 1st and 2nd cumulants (mean and covariance) summarize it completely.
- We can measure nonlinearity of a process through its 3rd cumulants, and the associated bi-spectral density.

## Definition 11.5.1.

- Suppose $\{X_t\}$ is a strictly stationary time series with finite absolute third moment.
- Then its third *cumulant function* is defined to be

$$\gamma(k_1, k_2) = \mathbb{E}[(X_t - \mu)(X_{t+k_1} - \mu)(X_{t+k_2} - \mu)],$$

for any integer $k_1$, $k_2$.

### Remark 11.5.2. Cumulant Functions

- Cumulant functions generalize cumulants for random variables.
- The mean is the first cumulant function.
- The ACVF is the second cumulant function.
- The third cumulant function measures skewness across lags.

### Fact 11.5.3. Symmetries of the Third Cumulant

- There are symmetries:

$$\gamma(k_1, k_2) = \gamma(k_2, k_1) = \gamma(k_1 - k_2, -k_2) = \gamma(-k_1, k_2 - k_1).$$

### Definition 11.5.6.

The *bi-spectral density* function of a strictly stationary time series is the Fourier transform of the third cumulant function:

$$f(\lambda_1, \lambda_2) = \sum_{k_1, k_2 = -\infty}^{\infty} \gamma(k_1, k_2) \exp\{-i\lambda_1 k_1 - i\lambda_2 k_2\},$$

where $\lambda_2, \lambda_2 \in [-\pi, \pi]$.

### Example 11.5.7. Bi-spectral Density of an ARCH(1) Process

- Consider the ARCH(1) process $X_t = Z_t \sqrt{a_0 + a_1 X_{t-1}^2}$.
- The bi-spectral density is

$$f(\lambda_1, \lambda_2) = \mu_3 \left( (1 - a_1 e^{-i\lambda_1 - i\lambda_2})^{-1} + (1 - a_1 e^{i\lambda_1})^{-1} + (1 - a_1 e^{i\lambda_2})^{-1} - 2 \right),$$

  where $\mu_3 = \mathbb{E}[X_t^3]$.

### Theorem 11.5.8.

- Suppose that $\{X_t\}$ is a stationary linear time series such that $X_t = \mu + \psi(B)\epsilon_t$, with $\{\epsilon_t\}$ i.i.d. $(0, \sigma^2)$ and $\psi(z) = \sum_{j=-\infty}^{\infty} \psi_j z^j$.
- Then the bi-spectral density is

$$f(\lambda_1, \lambda_2) = \kappa_3 \, \psi(e^{-i\lambda_1}) \psi(e^{-i\lambda_2}) \psi(e^{i\lambda_1 + i\lambda_2}),$$

  where $\kappa_3 = \mathbb{E}[\epsilon_t^3]$.

## Lesson 11-6: Volatility Filtering

- ARCH and GARCH models do not *fully* capture volatility clustering and *fat tails* (i.e., extreme observations).
- We can use a model-free approach to describe $\sigma_t$, the volatility.
- In this lesson we view $\sigma_t$ as deterministic, and defined via $\sigma_t = \mathbb{E}[X_t^2]$.

### Definition 11.6.2

- Consider a moving average of squared returns:

$$\Pi(B) X_t^2 = \sum_{j=-\infty}^{\infty} \pi_j \, X_{t-j}^2.$$

  This is a *volatility filter*, if all $\pi_j \geq 0$.

- A volatility filter can be causal, symmetric, etc.

### Paradigm 11.6.3. Unbiased Volatility Filtering

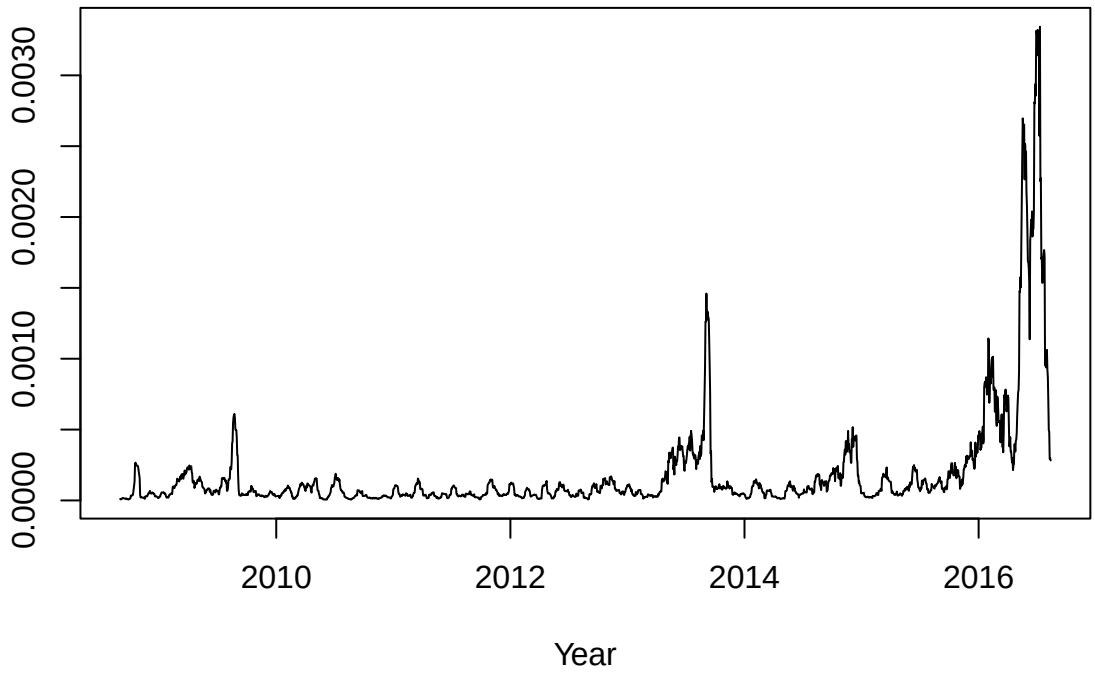
- We consider estimating volatility $\sigma_t^2 = \mathbb{E}[X_t^2]$ by a volatility filter, just like we did for estimating the mean $\mu_t = \mathbb{E}[X_t]$:
$$\widehat{\sigma}_t^2 = \Pi(B)X_t^2.$$
- The mean is $\Pi(B)\sigma_t^2$, which can be compared to the target $\sigma_t^2$; if it differs, there is bias.
- The same bias-variance trade-off applies as with mean estimation.

### Example 11.6.4. Volatility Filtering of Dow Jones Log Returns

- Consider the volatility filter $\pi_j = 1/11$ for $|j| \leq 5$.
- Observe the increased volatility in more recent years.

```
dow <- read.table("dow.dat")
dow <- diff(log(dow[,1]))
dow <- ts(dow,start=c(2008,164),frequency=252)

p <- 5
dow.vol <- filter(dow^2,rep(1,2*p+1)/(2*p+1),method="convolution",sides=2)
plot(dow.vol,xlab="Year",ylab="",lwd=1)
```




### Definition 11.6.5.

- When $\Pi(B)$ is causal, then $\widehat{\sigma}_t^2$ depends only on present and past data.
- Then we can divide out by the (square root) volatility, getting a *pseudo-residual*:
$$W_t = \frac{X_t}{\widehat{\sigma}_t}.$$

This is called the *Normalizing and Variance Stabilizing* (NoVaS) transform.
- The NoVaS denominator $\widehat{\sigma}_t$ is unbiased if $\Pi(1) = 1$, i.e., the sum of all coefficients equals one.

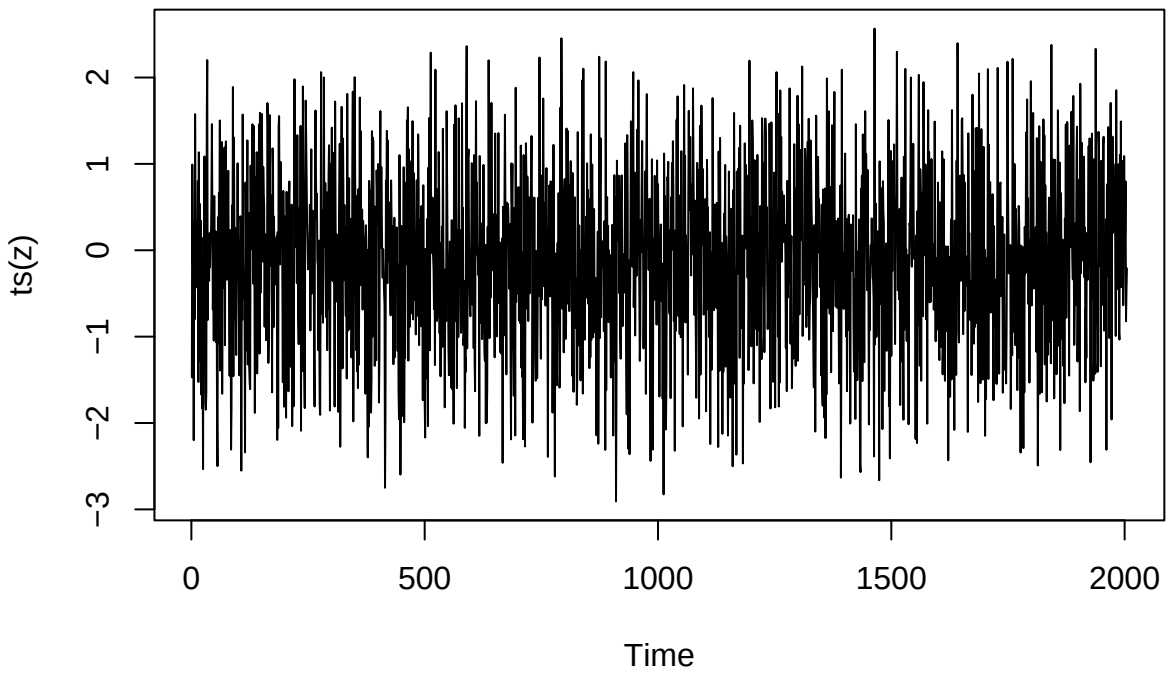
### Paradigm 11.6.6. NoVaS with Simple Weights

- The idea is to choose a NoVaS transform $\Pi(B)$ such that $\{W_t\}$ is close to i.i.d. Gaussian.
- We want light tails and no serial correlation.
- We can search over volatility filters to find pseudo-residuals with this property.
- NoVaS with simple weights: take $\Pi(B)$ causal with $\pi_0 = \pi_1 = \ldots = \pi_p = 1/(p+1)$.
- We just need to choose $p$, which is done by making the kurtosis low (close to that of a Gaussian).

### Example 11.6.7. Simple NoVaS Applied to Dow Log Returns

- Test out Simple NoVaS on the Dow log returns time series.

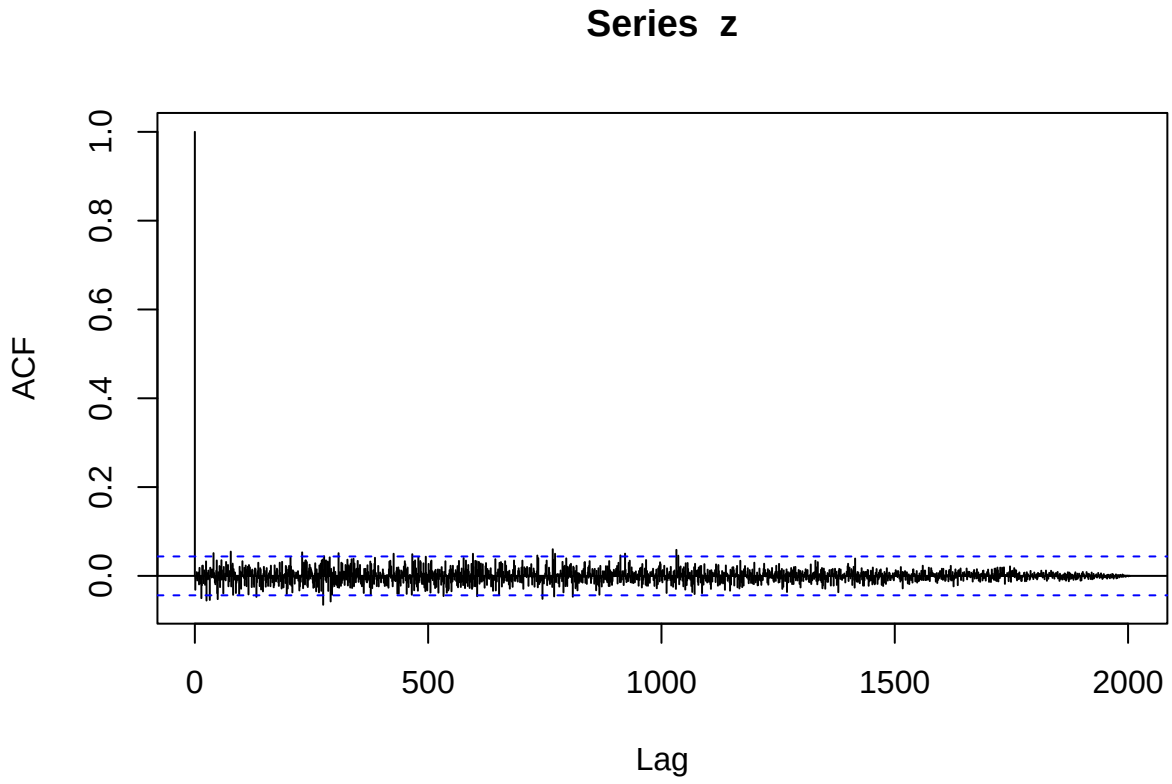
```
p <- 9
psi <- rep(1,p+1)/(p+1)
n <- length(dow)
dow.vol <- filter(dow^2,psi,method="convolution",sides=1)[(p+1):n]
z <- dow[(p+1):n]/sqrt(dow.vol)
m <- length(z)
plot(ts(z))
```




- It turns out, the serial correlation is low and the kurtosis is close to three (the Gaussian case)!
- We print out the excess kurtosis and a standardized excess kurtosis.

```
kurt <- m*sum((z - mean(z))^4)/(sum((z - mean(z))^2))^2 - 3
kurt.std <- sqrt(m)*kurt/sqrt(mean(z^8)-mean(z^4)^2)
```

```r
z.acf <- acf(z,plot=TRUE,lag.max=m-1,type="correlation")$acf
```




```r
print(c(kurt,kurt.std))
```

```
## [1] -0.2743338 -1.8086432
```