Lecture 21: Correlated Errors (Continued)

Ailin Zhang

2023-07-11

Agenda

- Definition of Correlated Errors
- Detection of Correlated Errors
  - Time plot of residuals
  - Runs test
  - Durbin-Watson test
  - Lag plots
  - Autocorrelation function and autocorrelation plot
- Remedies to Correct for Autocorrelated Errors (Today)
- Autocorrelation and Seasonality (Today)

Autocorrelation

- The lag-k autocorrelation of the residuals $(e_1, ..., e_n)$ is defined as

$$\hat{\rho}_k = \frac{\sum_{t=k+1}^{n} e_t e_{t-k}}{\sum_{t=1}^{n} e_t^2}$$

- Autocorrelation is slightly different from the correlation of $(e_1, \ldots, e_{n-k})$ v.s. $(e_{1+k}, \ldots, e_n)$:

$$\frac{\sum_{t=k+1}^{n} e_t e_{t-k}}{\sqrt{\sum_{t=1}^{n-k} e_t^2 \sum_{t=k+1}^{n} e_t^2}}$$

- The R command `acf()` (autocorrelation function) in R can calculate lag-k autocorrelation.

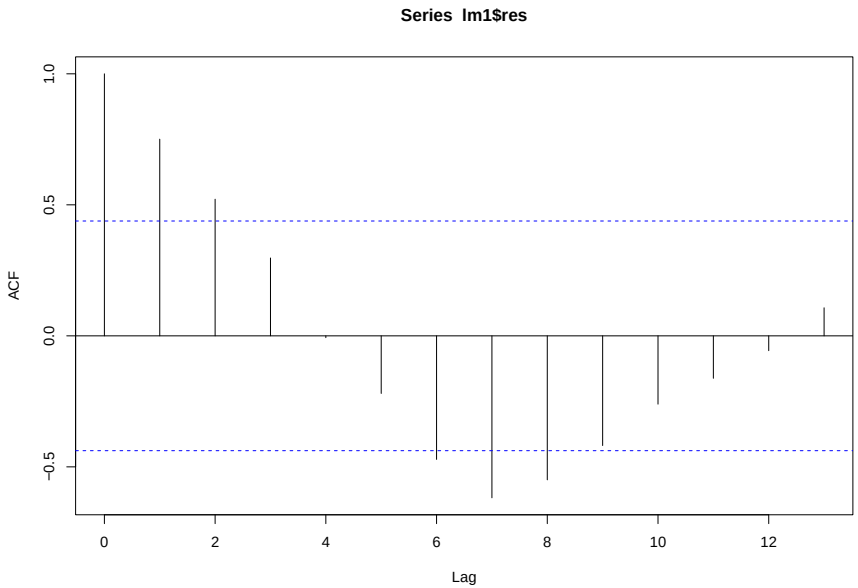
```
acf(lm1$res, lag.max =5, plot=FALSE)
```

```
##
## Autocorrelations of series 'lm1$res', by lag
##
##     0     1     2     3     4     5
## 1.000 0.751 0.521 0.297 -0.007 -0.220
```

Autocorrelation Function and the Plot

In time-series analysis, one often plot the lag-k autocorrelations against k to examine the autocorrelation structure of a variable. The `acf()` command can produce such autocorrelation plot.

```
acf(lm1$res)
```



The horizontal dash lines marks the levels autocorrelations to be significantly different from 0.

Remedies to Correct for Autocorrelated Errors: Transformation

Assume the errors ϵ_t 's of the linear model $y_t = \beta_0 + \beta_1 x_t + \epsilon_t$ have the first order autocorrelation AR(1) structure:

$$\epsilon_t = \rho\epsilon_{t-1} + \omega_t, \omega_t \sim N(0, \theta^2)$$

Then

$$y_t - \beta_0 - \beta_1 x_t = \rho(y_{t-1} - \beta_0 - \beta_1 x_{t-2}) + \omega_t$$

$$\underbrace{y_t - \rho y_{t-1}}_{y_t^*} = \beta_0(1-\rho) + \beta_1 \underbrace{(x_t - \rho x_{t-1})}_{x_t^*} + \omega_t$$

Hence the transformed variables x_t^* and y_t^* satisfy the SLR model:

$$y_t^* = \beta_0^* + \beta_1^* x_t^* + \omega_t$$

, where ω_t are indep. $\sim N(0, \theta^2)$

Interpreting the coefficients

- Transform back!
- The coefficients of the orignal and the transformed models are related as follows:

$$\beta_0(1-\rho) = \beta_0^*, \quad \beta_1 = \beta_1^*$$

- However we need to estimate ρ

Cochrane-Orcutt Method

1. Fit the OLS model and obtain the residuals $e_1, \ldots, e_n$
2. Use the residuals $e_1, \ldots, e_n$ to estimate ρ with

$$\hat{\rho} = \frac{\sum_{t=2}^{n} e_t e_{t-1}}{\sum_{t=1}^{n} e_t^2}$$

3. Compute OLS estimates of β_0^* and β_1^* by regressing $y^* = y_t - \hat{\rho} y_{t-1}$ on $x^* = x_t - \hat{\rho} x_{t-1}$ and use them to find coefficients for the original variables. $\hat{\beta}_0 = \dfrac{\hat{\beta}_0^*}{1 - \hat{\rho}}$ and $\hat{\beta}_1 = \hat{\beta}_1^*$
4. Use the new $\hat{\beta}_0$ and $\hat{\beta}_1$ to calulate the new residuals $e_1, \ldots, e_n$

and then go back to Step 2.

5. Iterate until the estimates $\hat{\beta}_0$ and $\hat{\beta}_1$ converge.

R code: first iteration

```
x = stock$Stock
y = stock$Expenditure
n = length(y)
fit1 = lm(y ~ x)
res = fit1$res
rho.hat = sum(res[1:(n-1)]*res[2:n]) / sum(res^2 )
rho.hat
```

```
## [1] 0.7506122
```

```
ystar = y[2:n] - rho.hat*y[1:(n-1)]
xstar = x[2:n] - rho.hat*x[1:(n-1)]
fit2 = lm(ystar ~ xstar)
b0.hat = fit2$coef[1]/(1-rho.hat)
b1.hat = fit2$coef[2]
c(b0.hat, b1.hat)
```

```
## (Intercept)       xstar
## -215.310969    2.643443
```

R code: second iteration

```
res = y - b0.hat - b1.hat*x
rho.hat = sum(res[1:(n-1)]*res[2:n]) / sum(res^2 )
rho.hat
```

```
## [1] 0.789962
```

```
ystar = y[2:n] - rho.hat*y[1:(n-1)]
xstar = x[2:n] - rho.hat*x[1:(n-1)]
fit2 = lm(ystar ~ xstar)
b0.hat = fit2$coef[1]/(1-rho.hat)
b1.hat = fit2$coef[2]
c(b0.hat, b1.hat)
```

```
## (Intercept)       xstar
## -225.600207    2.699873
```

R code: More iterations

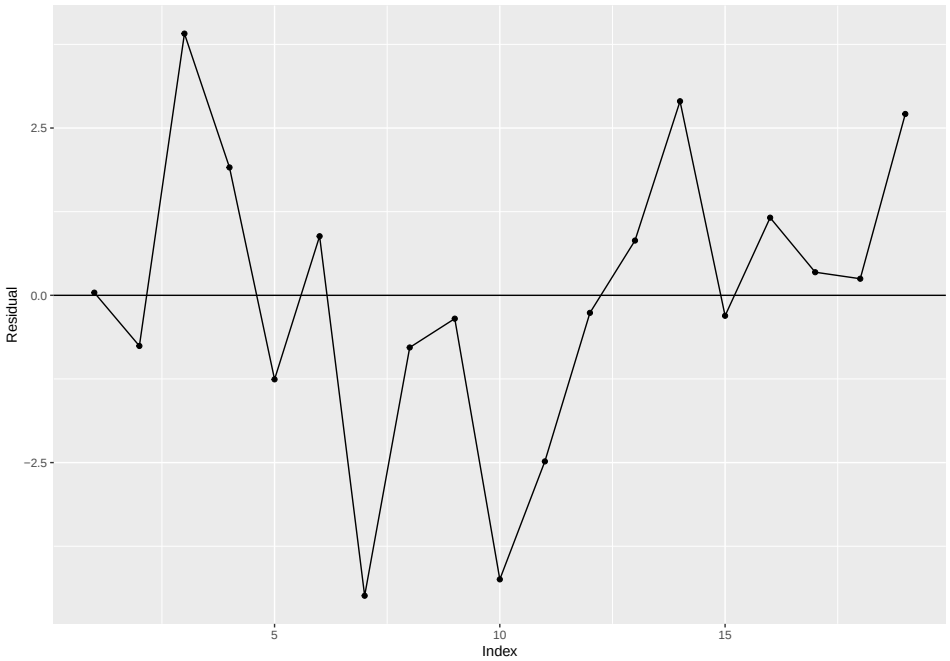
```
x = stock$Stock
y = stock$Expenditure
n = length(y)
n.iter = 15
rho.iter = vector("numeric", n.iter)
b0.iter = vector("numeric", n.iter)
b1.iter = vector("numeric", n.iter)
fit1 = lm(y ~ x)
res = fit1$res
rho.iter[1] = sum(res[1:(n-1)]*res[2:n]) / sum(res^2 )

for(i in 2:n.iter){
  rho.iter[i] = sum(res[1:(n-1)]*res[2:n]) / sum(res^2 )
  ystar = y[2:n] - rho.iter[i]*y[1:(n-1)]
  xstar = x[2:n] - rho.iter[i]*x[1:(n-1)]
  fit2 = lm(ystar ~ xstar)$coef
  b0.iter[i] = fit2[1]/(1-rho.iter[i])
  b1.iter[i] = fit2[2]
  res = y - b0.iter[i] - b1.iter[i]*x
}
```

R code: More iterations

```
##      rho.iter   b0.iter  b1.iter
## 1   0.7506122    0.0000 0.000000
## 2   0.7506122 -215.3110 2.643443
## 3   0.7899620 -225.6002 2.699873
## 4   0.7977275 -227.8166 2.711905
## 5   0.7995646 -228.3473 2.714777
## 6   0.8000161 -228.4780 2.715485
## 7   0.8001281 -228.5105 2.715660
## 8   0.8001560 -228.5185 2.715704
## 9   0.8001629 -228.5206 2.715715
## 10 0.8001646 -228.5211 2.715717
## 11 0.8001650 -228.5212 2.715718
## 12 0.8001651 -228.5212 2.715718
## 13 0.8001652 -228.5212 2.715718
## 14 0.8001652 -228.5212 2.715718
## 15 0.8001652 -228.5212 2.715718
```

Checking the Independence Assumption After Transformation

```
rho.hat = rho.iter[n.iter]
ystar = y[2:n] - rho.hat*y[1:(n-1)]
xstar = x[2:n] - rho.hat*x[1:(n-1)]
xystar = data.frame(xstar, ystar)
fit2 = lm (ystar ~ xstar)
ggplot(xystar, aes(x=1:(n-1), y = fit2$res)) +
  geom_point() + geom_line() +
  labs(x="Index", y="Residual") + geom_hline(yintercept=0)
```



Runs Test

```
library(tseries)
runs.test(factor(fit2$res > 0)) # two-sided by default
```

```
##
##  Runs Test
##
## data:  factor(fit2$res > 0)
## Standard Normal = -0.69782, p-value = 0.4853
## alternative hypothesis: two.sided
```

```
runs.test(factor(fit2$res > 0),alternative = "less")
```

```
##
##  Runs Test
##
## data:  factor(fit2$res > 0)
## Standard Normal = -0.69782, p-value = 0.2426
## alternative hypothesis: less
```

```
runs.test(factor(fit2$res > 0),alternative = "greater")
```

```
##
##  Runs Test
##
## data:  factor(fit2$res > 0)
## Standard Normal = -0.69782, p-value = 0.7574
## alternative hypothesis: greater
```

Durbin-Watson Test

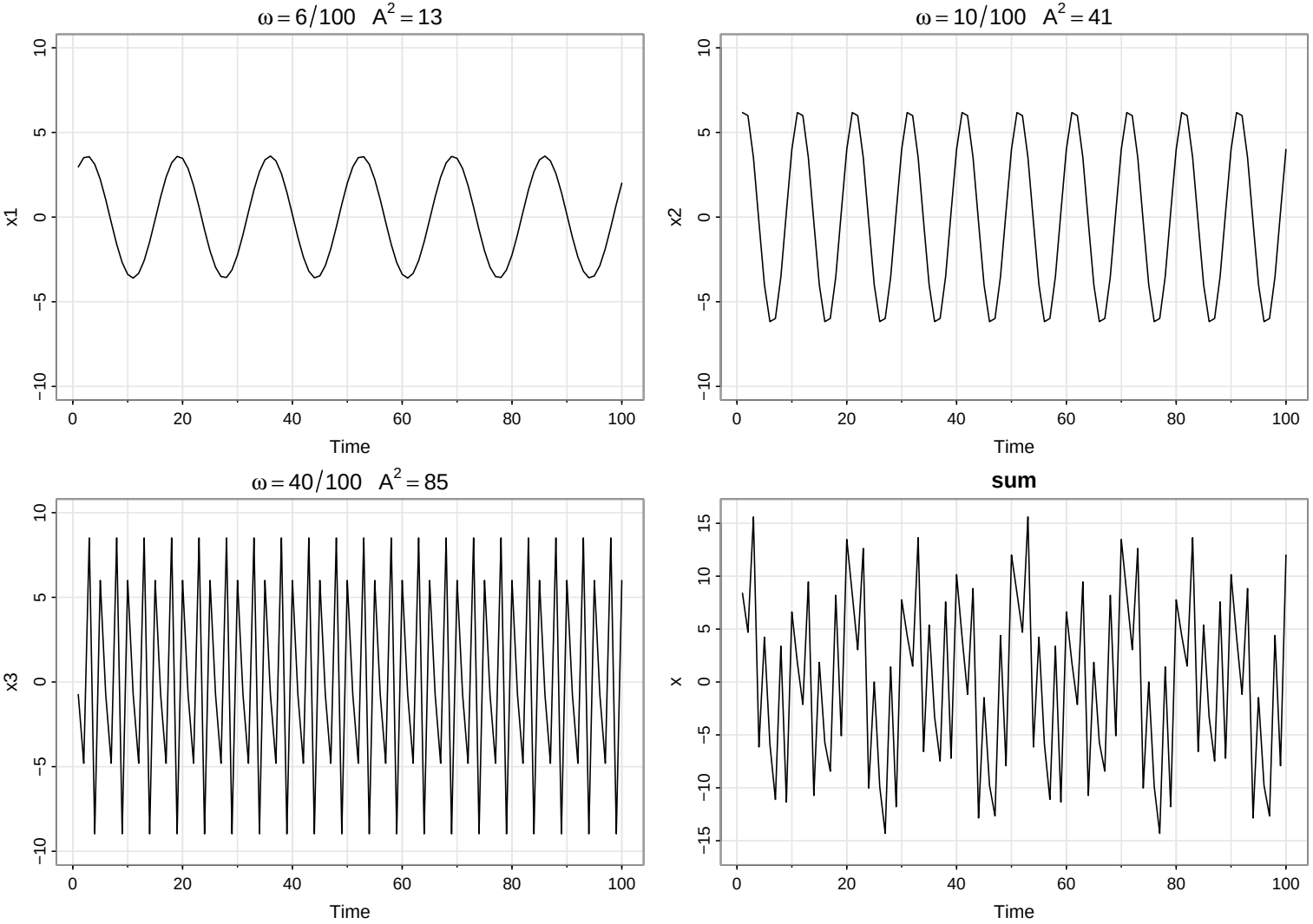
```
library(car)
durbinWatsonTest(fit2, alt="positive")
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.1824862      1.548671   0.109
##  Alternative hypothesis: rho > 0
```

Autocorrelation and Seasonality

```
x1 = 2*cos(2*pi*1:100*6/100)  + 3*sin(2*pi*1:100*6/100)
x2 = 4*cos(2*pi*1:100*10/100) + 5*sin(2*pi*1:100*10/100)
x3 = 6*cos(2*pi*1:100*40/100) + 7*sin(2*pi*1:100*40/100)
x = x1 + x2 + x3
```

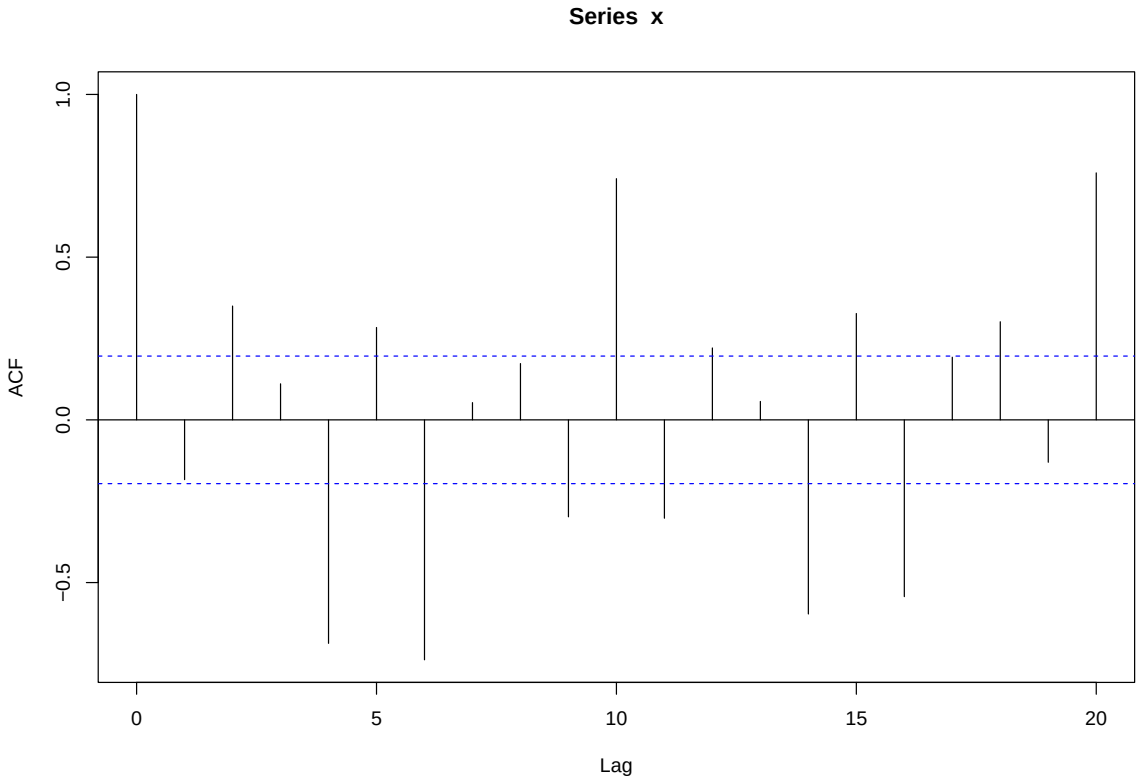
Autocorrelation and Seasonality



Autocorrelation

ACF plots can help detect higher-order dependence.

```
acf(x)
```



Durbin-Watson Test

```
fit_syn = lm(x~1)
durbinWatsonTest(fit_syn, alt="positive")
```

```
##  lag Autocorrelation D-W Statistic p-value
##    1      -0.1836366       2.336403   0.953
##  Alternative hypothesis: rho > 0
```

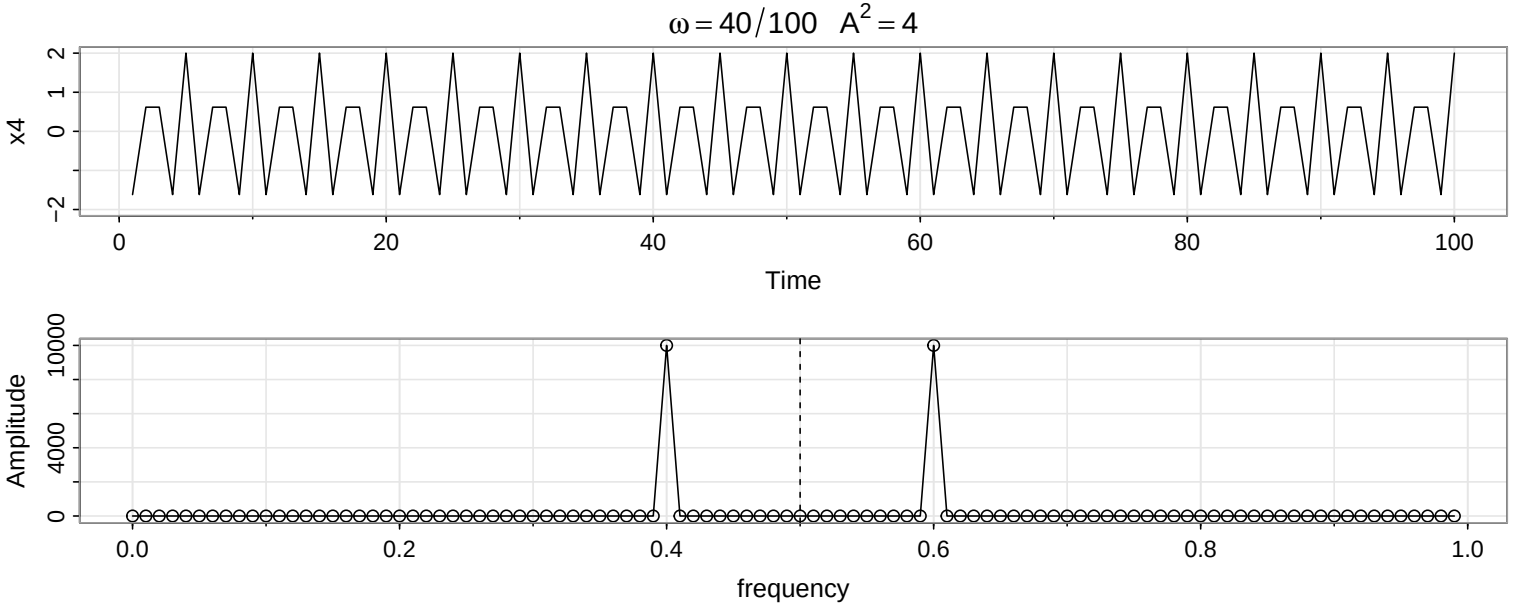
Durbin-Watson test give large P-values even though there exist significant lag-k autocorrelation (It failed!)

Seasonality (Time and frequency)

- The frequency is measured in cycles per unit time (ω)
- $\omega = 1.0$ the series makes one cycle per time unit
- $\omega = 0.5$ the series makes one cycle every two time units
- In general, for data that occur at discrete time points will need at least two points to determine a cycle, so the highest frequency of interest is 0.5 cycles per point.
- This frequency is called the folding frequency/ Nyquist frequency ($F = f_s/2$)
- Higher frequencies sampled this way will appear at lower frequencies, called aliases

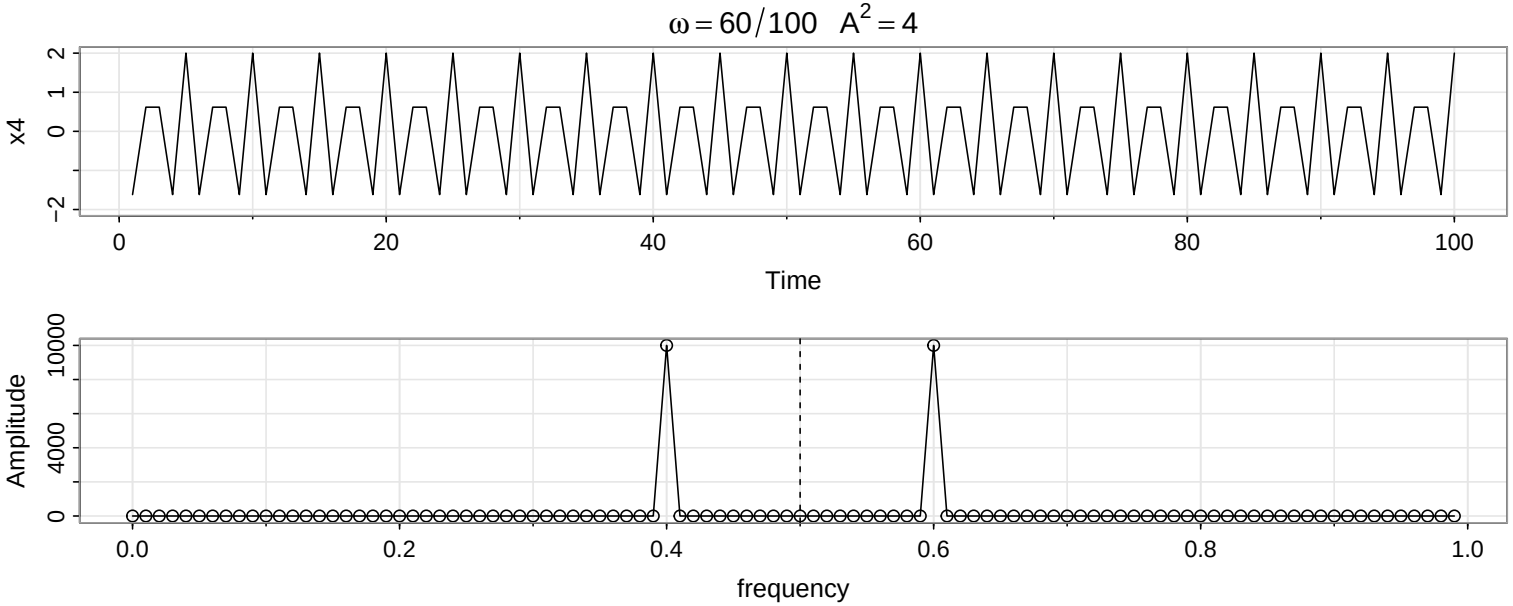
Alias and Folding Frequency (f=0.4Hz)

```
x4 = 2*cos(2*pi*1:100*40/100)
P = abs(fft(x4))^2
Fr = 0:99/100
par(mfrow=c(2,1))
tsplot(x4, ylim=c(-2,2), main = expression(omega==40/100~~~A^2
tsplot(Fr, P, type="o", xlab="frequency", ylab="Amplitude")
abline(v=.5, lty=2)
```



Alias and Folding Frequency (f=0.6Hz)

```
x4 = 2*cos(2*pi*1:100*60/100)
P = abs(fft(x4))^2
Fr = 0:99/100
par(mfrow=c(2,1))
tsplot(x4, ylim=c(-2,2), main = expression(omega==60/100~~~A^2
tsplot(Fr, P, type="o", xlab="frequency", ylab="Amplitude")
abline(v=.5, lty=2)
```



Seasonality: The Scaled Periodogram

It may be regarded as a measure of the squared correlation of the data

```
P = abs(2*fft(x)/100)^2
Fr = 0:99/100
tsplot(Fr, P, type="o", xlab="frequency", ylab="periodogram")
abline(v=.5, lty=2)
```