Autocorrelation

Peter Claussen

9/5/2017

Suppose we have a series of measurements, $y_1, ..., y_n$ and we wish to know if the measurements are independent. A simple test is the sample lag-1 autocorrelation coefficient r_1, given by (http://www.itl.nist.gov/div898/strd/univ/certmethdef/michelso.html)

$$r_1 = \frac{\Sigma_{i=2}^{n}(y_i - \bar{y})(y_{i-1} - \bar{y})}{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}$$

and

$$\bar{y} = (\Sigma y_i)/n$$

is the sample mean.

(For reference, we also write sample variance as

$$\frac{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}{(n-1)}$$

and two sample covariance as

$$\frac{\Sigma_{i=1}^{n}(y_i - \bar{y})(x_i - \bar{x})}{\sqrt{\Sigma_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}}$$

)

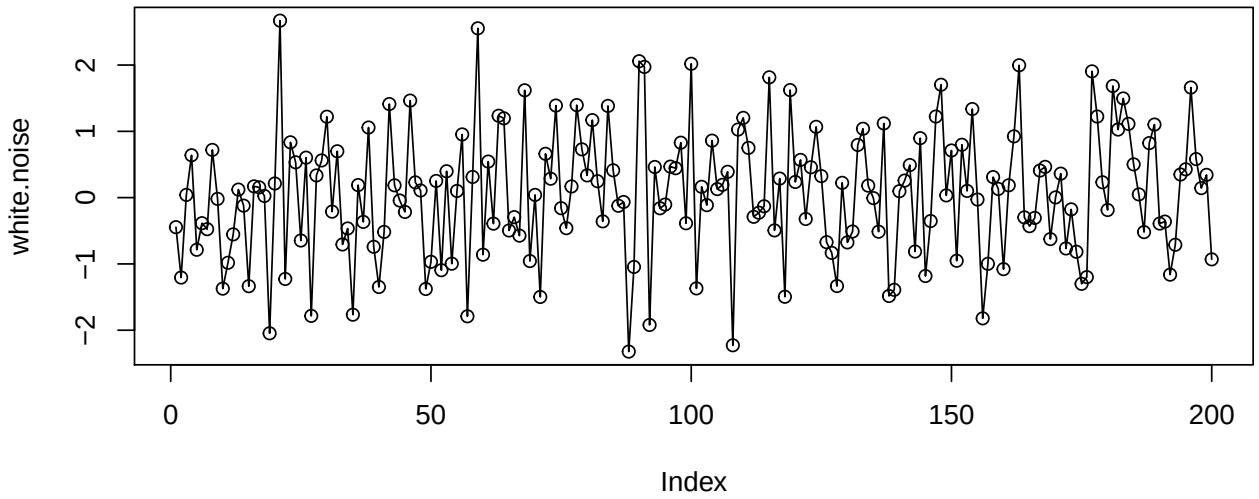
Example (Simulated) Data

Before we start computing autocorrelation coefficients, we first consider the types of processes that create sequences of observations.

White Noise

A white noise process is the type of process commonly associated with experimental error. This is random process that produces values that are identically distributed and independent. We simulate a series of 200 observations, from a normal distribution with mean 0 and standard deviation 1 by

```
set.seed(1000)
white.noise <- rnorm(200)
plot(white.noise);lines(white.noise)
```



Effectively, this is statistical model with no effects, only the error term,

$$Y_i = e_i$$

We can compute r_1 by

```
print(mean.white.noise <- mean(white.noise))
```

```
## [1] 0.058402
```

```
print(ss.white.noise <- sum((white.noise-mean.white.noise)^2))
```

```
## [1] 181.4588
```

```
print(n <- length(white.noise))
```

```
## [1] 200
```

```
ss.white.noise/(n-1)
```

```
## [1] 0.9118534
```

```
print(lag.white.noise <- sum((white.noise[2:n]-mean.white.noise) * (white.noise[1:(n-1)]-mean.white.noi
```

```
## [1] -2.90349
```

```
print(r.white.noise <- lag.white.noise/ss.white.noise)
```

```
## [1] -0.01600082
```

We'll compute this value again, so we can write a simple function.

```
auto.correlation <- function(univariate) {
  x.bar <- mean(univariate)
  ss <- sum((univariate-x.bar)^2)
  n <- length(univariate)
  lag.ss <- sum((univariate[(2):n]-x.bar)*(univariate[1:(n-1)]-x.bar))
  return(lag.ss/ss)
}
auto.correlation(white.noise)
```

```
## [1] -0.01600082
```

r_1 for these data is very close to 0, which is what we expect for uncorrelated values.

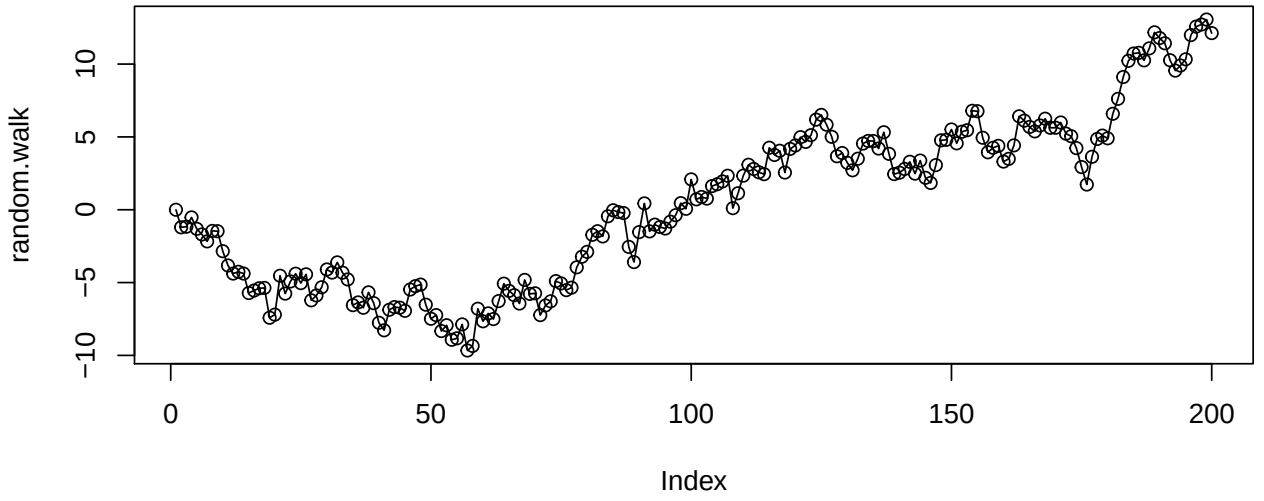
Random Walk

Suppose, in a given sequence of values, the next value in the sequence is created by adding a random value to the current value in the sequence. This type of sequence is commonly called a random (or drunkards) walk. We can write this mathematically as

$$Y_i = Y_{i-1} + e_i$$

In order to understand the relationship among the different types of autocorrelated models, we will generate these models using the same `white-noise` error, $e_1, ...e_n$. We generate a random walk by

```
random.walk <- rep(0,200)
for(i in 2:200) {
  random.walk[i] <- random.walk[i-1]+white.noise[i]
}
plot(random.walk);lines(random.walk)
```




```
auto.correlation(random.walk)
```

```
## [1] 0.9754526
```

Since values are created via summation, this sequence has a r_1 near 1. Note that a perfectly correlated sequence would have $r_1 = 1$; it would also not be calculable, since $\Sigma_{i=1}^{n}(y_i - \bar{y})^2 = 0$

```
auto.correlation(rep(1,100))
```

```
## [1] NaN
```

We can also note that a sequence of alternative values would produce an r_1 near -1.

```
auto.correlation(rep(c(1,0),100))
```

```
## [1] -0.995
```

Autoregressive

Now, suppose we have a random walk, but the next value is proportional to the current value and not simply additive. This is known as an autoregression model. We write this as

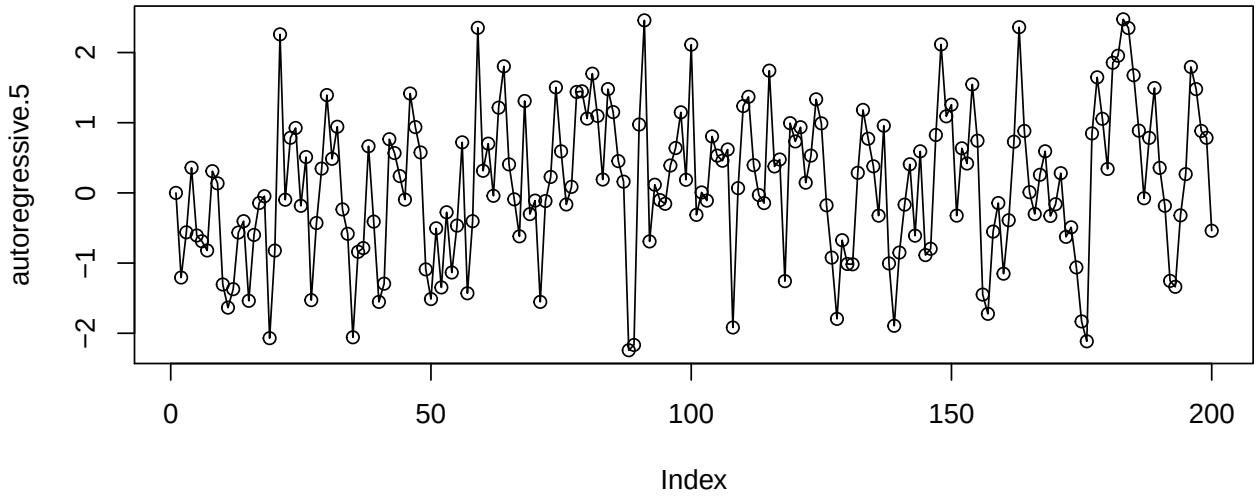
$$Y_i = \alpha Y_{i-1} + e_i$$

When $\alpha = 1$, then we have a random walk, and when $\alpha = 0$, we have a white-noise process.

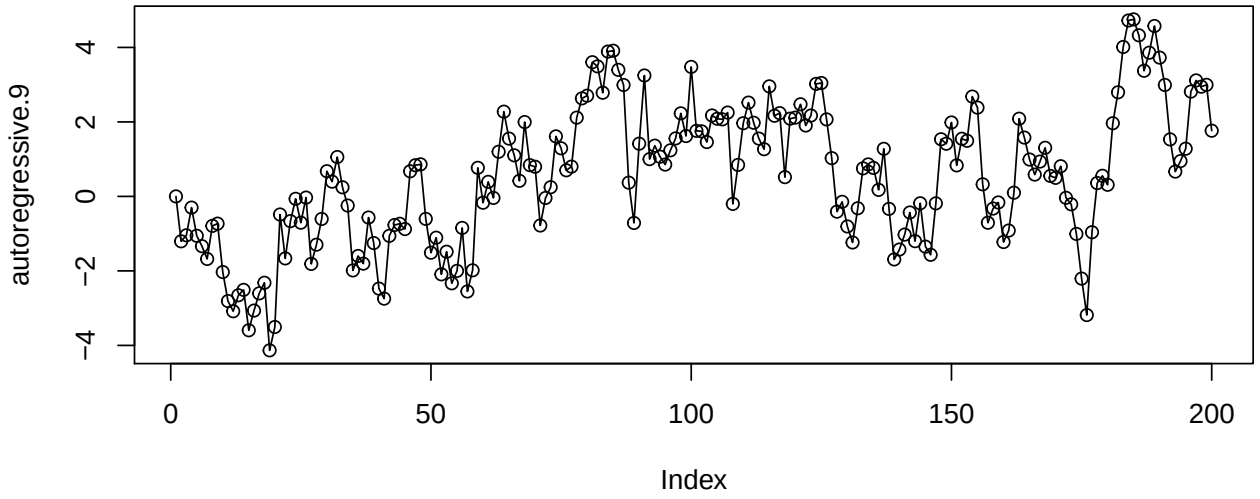
We'll create a set of processes using different α to help visualize how this parameter affects the process.

```
autoregressive.5 <- rep(0,200)
autoregressive.9 <- rep(0,200)
autoregressive.99 <- rep(0,200)
autoregressive.101 <- rep(0,200)
for(i in 2:200) {
  autoregressive.5[i] <- 0.5*autoregressive.5[i-1]+white.noise[i]
  autoregressive.9[i] <- 0.9*autoregressive.9[i-1]+white.noise[i]
  autoregressive.99[i] <- 0.99*autoregressive.99[i-1]+white.noise[i]
  autoregressive.101[i] <- 1.02*autoregressive.101[i-1]+white.noise[i]
}
```

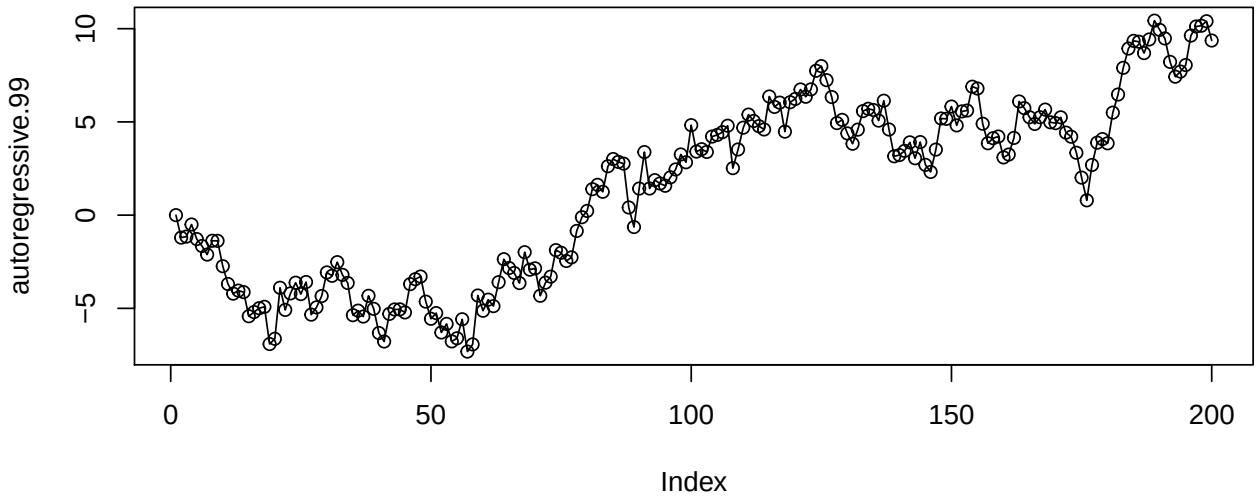
```
plot(autoregressive.5);lines(autoregressive.5)
```



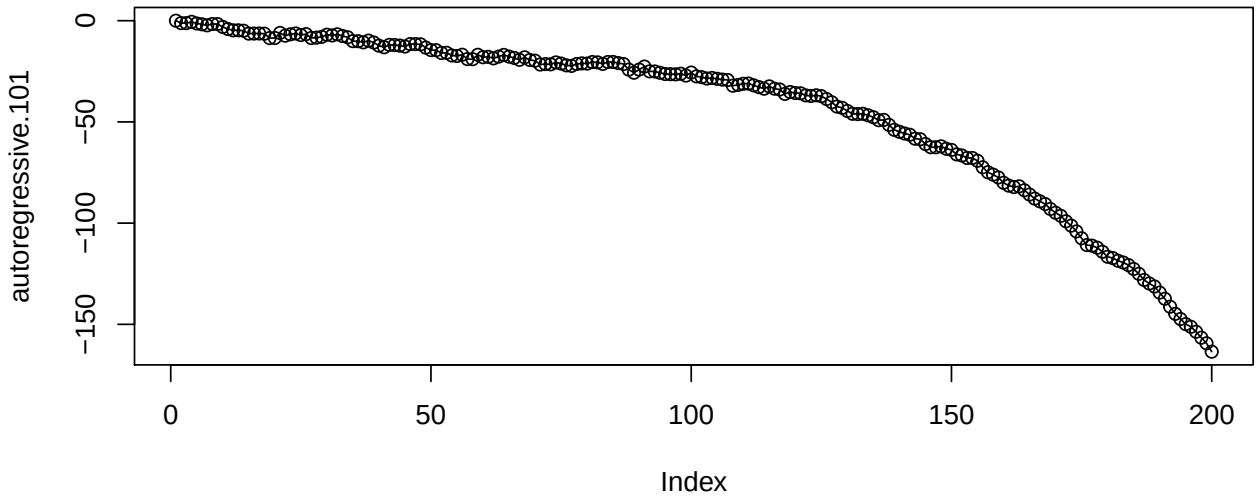

```
plot(autoregressive.9);lines(autoregressive.9)
```




```
plot(autoregressive.99);lines(autoregressive.99)
```




```
plot(autoregressive.101);lines(autoregressive.101)
```



As we might expect, r_1 increases with α

```
auto.correlation(autoregressive.5)
```

```
## [1] 0.4279652
```

```
auto.correlation(autoregressive.9)
```

```
## [1] 0.8574139
```

```
auto.correlation(autoregressive.99)
```

```
## [1] 0.973654
```

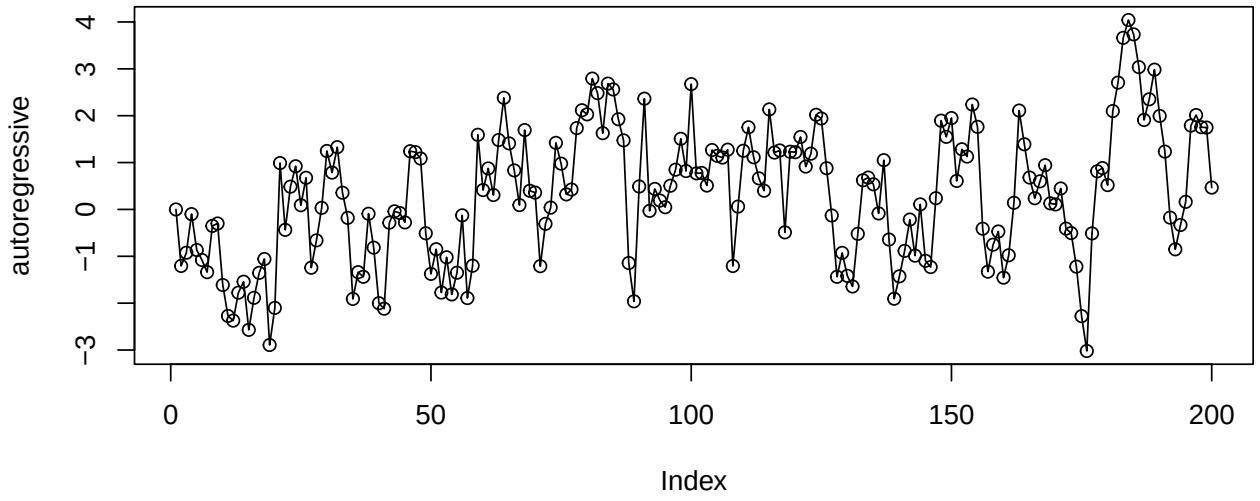
```
auto.correlation(autoregressive.101)
```

```
## [1] 0.9762879
```

For further analysis, we use a modestly autoregressive sequence with $\alpha = 0.8$

```
autoregressive <- rep(0,200)
alpha <- 0.8
for(i in 2:200) {
  autoregressive[i] <- alpha*autoregressive[i-1]+white.noise[i]
```

```
}
plot(autoregressive);lines(autoregressive)
```



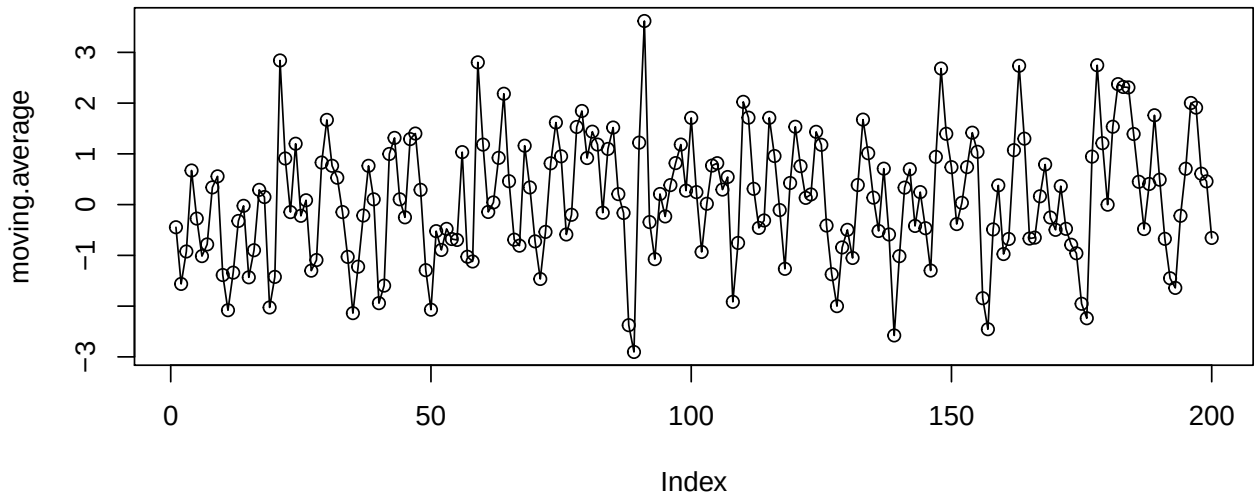
Moving Average

Now suppose we have a sequence that is not dependent on the preceding value in the sequence, but only on the preceding random variable generated in the sequence. Mathematicall, we write

$$Y_i = \beta e_{i-1} + e_i$$

We generate this process by

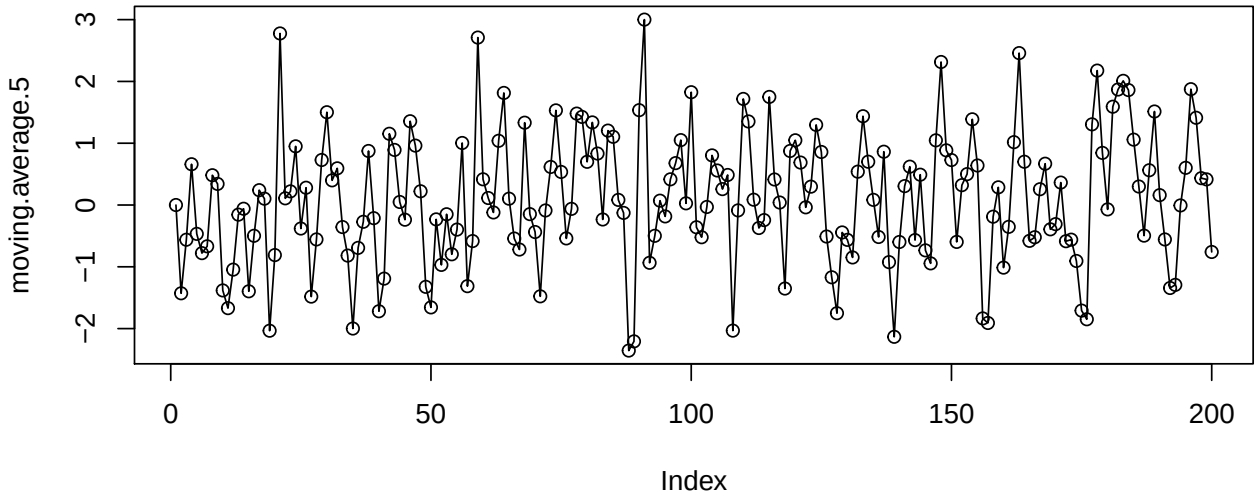
```
moving.average <- rep(0,200)
moving.average[1] <- white.noise[1]
beta <- 0.8
for(i in 2:200) {
  moving.average[i] <- beta*white.noise[i-1]+white.noise[i]
}
plot(moving.average);lines(moving.average)
```



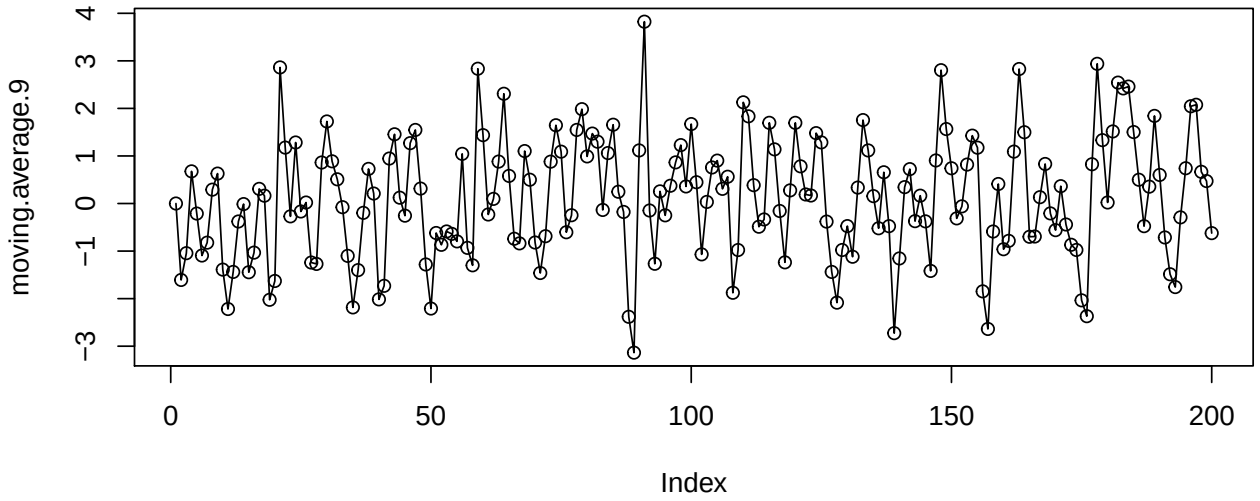
Again, we can compare different β values.

```
moving.average.5 <- rep(0,200)
moving.average.9 <- rep(0,200)
moving.average.99 <- rep(0,200)
moving.average.101 <- rep(0,200)
for(i in 2:200) {
  moving.average.5[i] <- 0.5*white.noise[i-1]+white.noise[i]
  moving.average.9[i] <- 0.9*white.noise[i-1]+white.noise[i]
  moving.average.99[i] <- 0.99*white.noise[i-1]+white.noise[i]
  moving.average.101[i] <- 1.02*white.noise[i-1]+white.noise[i]
}
```

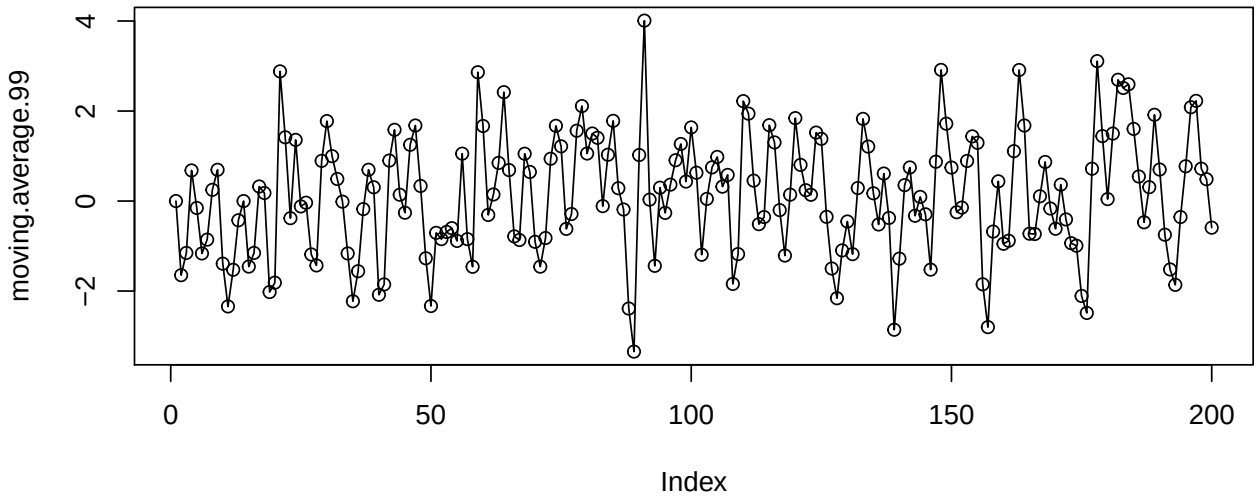
```
plot(moving.average.5);lines(moving.average.5)
```



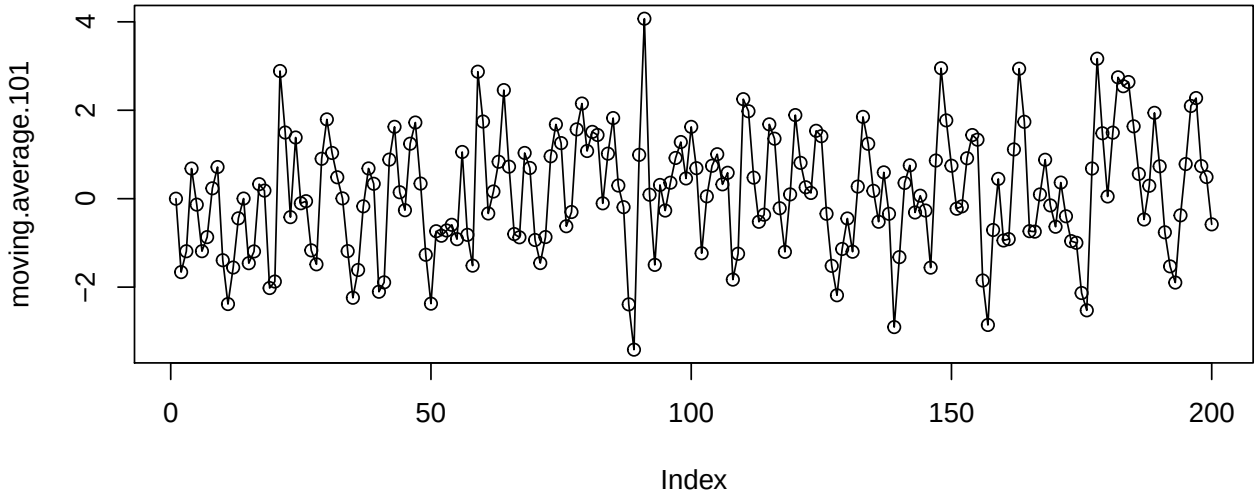

```
plot(moving.average.9);lines(moving.average.9)
```




```
plot(moving.average.99);lines(moving.average.99)
```




```
plot(moving.average.101);lines(moving.average.101)
```



Unlike α, β is not captured by r_1.

```
auto.correlation(moving.average.5)
```

```
## [1] 0.344721
```

```
auto.correlation(moving.average.9)
```

```
## [1] 0.4360312
```

```
auto.correlation(moving.average.99)
```

```
## [1] 0.4388845
```

```
auto.correlation(moving.average.101)
```

```
## [1] 0.4389174
```

Trend plus Error

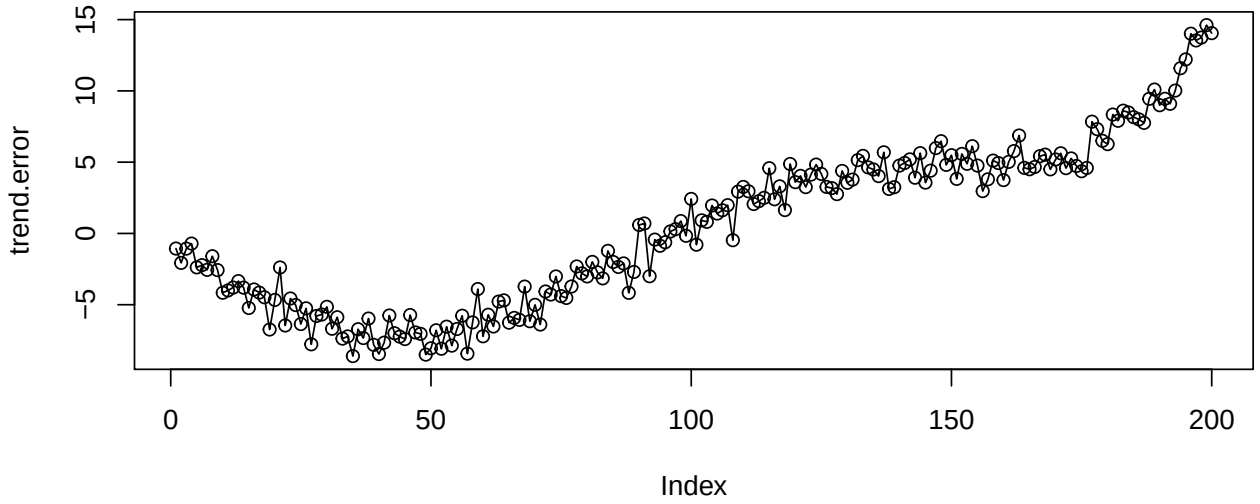
Previous models have been correlated via a random process, in most cases with preceding values. We should also consider a fixed effects model, where values are determine by an independent variable. In this case, we

let the independent variable be represented by the position in the sequence, so $x_i = i = 1, ..., n$. We could simulate this as a simple linear model, $y_i = \beta_0 + \beta_1 x_i + e_i$, but, instead, we'll approximate the random walk with a 5^{th} degree polynomial:

```
x <- 1:200
coefs.walk <- coef(lm(random.walk ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5)))
coefs.walk
```

```
##   (Intercept)             x        I(x^2)        I(x^3)        I(x^4)
## -3.693175e-01 -2.448578e-01 -4.782134e-04  8.676540e-05 -7.528478e-07
##        I(x^5)
##  1.855891e-09
```

```
trend.error <- coefs.walk[1] + coefs.walk[2]*x + coefs.walk[3]*I(x^2)+ coefs.walk[4]*I(x^3) + coefs.wall
plot(trend.error);lines(trend.error)
```




```
auto.correlation(trend.error)
```

```
## [1] 0.9565492
```

We'll reuse these same examples, so save for later,

```
autocorrelation.dat <- data.frame(
  white.noise = white.noise,
  random.walk = random.walk,
  autoregressive=autoregressive,
  moving.average=moving.average,
  trend.error=trend.error
)
save(autocorrelation.dat,file="autocorrelation.Rda")
write.csv(autocorrelation.dat, file = "autocorrelation.csv")
write.table(autocorrelation.dat, file = "autocorrelation.tab")
```

Lag

Before we continue, we should consider lag. So far, we've used lag-1 only models - our sequences have depending only on a single preceding value. We can generalize our models to include longer-range random correlations by

Name	General formula
order-k autocorrelation r_k	$r_n = \frac{\Sigma_{i=1}^{n-k}(y_i - \bar{y})(y_{i+k} - \bar{y})}{\Sigma_{i=1}^{n}(y_i - \bar{y})^2}$
order-k autoregression	$Y_i = \alpha_1 Y_{i-1} + ... + \alpha_k Y_{i-k} + e_i$
order-k moving average	$Y_i = \beta_1 e_{i-1} + ... + \beta_q e_{i-k}$

Fitting Autocorrelated Data

While there are many functions for fitting the various models, the most general method is to fit to ARMA - AutoRegressive Moving Average. This is a model of the form

$$Y_i = \alpha_1 Y_{i-1} + ... + \alpha_p Y_{i-p} + e_i + \beta_1 e_{i-1} + ... + \beta_q e_{i-q}$$

for AR models of order p and MA models of order q. We use the R function `arima` and specify order of the models using the parameter list of the form `(p,d,q)`, where `d` is a differencing parameter we can ignore for now.

So, consider fitting to our simulated sequences:

```
arima(autoregressive,c(1,0,0))
```

```
##
## Call:
## arima(x = autoregressive, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.7307     0.2922
## s.e.  0.0476     0.2454
##
## sigma^2 estimated as 0.8978:  log likelihood = -273.39,  aic = 552.78
```

```
arima(moving.average,c(0,0,1))
```

```
##
## Call:
## arima(x = moving.average, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       0.7827     0.1045
## s.e.  0.0435     0.1198
```

```
##
## sigma^2 estimated as 0.9069:  log likelihood = -274.49,  aic = 554.97
```

This seems to recover our simulated parameters well; we started with

$$\alpha = 0.8, \beta = 0.8, \sigma^2 = 1$$

and estimated

$$\hat{\alpha} = 0.7376, \hat{\beta} = 0.0.7827, \hat{\sigma}^2_{AR} = 0.8958, \hat{\sigma}^2_{MA} = 0.9069$$

If we fit a first order AR MA model, we find this also finds reasonable parameters

```
arima(autoregressive,c(1,0,1))
```

```
##
## Call:
## arima(x = autoregressive, order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.6962  0.0747     0.2924
## s.e.  0.0691  0.0980     0.2339
##
## sigma^2 estimated as 0.8953:  log likelihood = -273.11,  aic = 554.22
```

```
arima(moving.average,c(1,0,1))
```

```
##
## Call:
## arima(x = moving.average, order = c(1, 0, 1))
##
## Coefficients:
##           ar1     ma1  intercept
##       -0.0035  0.7840     0.1046
## s.e.   0.0891  0.0545     0.1195
##
## sigma^2 estimated as 0.9069:  log likelihood = -274.48,  aic = 556.97
```

What we should also note from this is that the AIC values suggest that in each case, the smaller model (AR or MA only) is a better fit than the combined ARMA model for our simulated data.

We also consider the other artificial series.

```
arima(white.noise,c(1,0,1))
```

```
##
## Call:
## arima(x = white.noise, order = c(1, 0, 1))
##
## Coefficients:
##          ar1      ma1  intercept
##       0.9731  -1.0000     0.0694
## s.e.  0.0258   0.0156     0.0284
##
## sigma^2 estimated as 0.8962:  log likelihood = -273.49,  aic = 554.98
```

The white noise data set finds α and β at the extremes.

```
arima(random.walk,c(1,0,1))
```

```
##
## Call:
## arima(x = random.walk, order = c(1, 0, 1))
##
## Coefficients:
##          ar1      ma1  intercept
##       0.9892  -0.0103     3.1070
## s.e.  0.0092   0.0801     4.6385
##
## sigma^2 estimated as 0.9094:  log likelihood = -276.2,  aic = 560.4
```

Remember, a random walk is equivalent to an AR model with $\alpha = 1$.

Most interestling, we have an error fitting the trend with error modeling. We discuss this error more when we consider stationarity.

```
#Error in arima(trend.error, c(1, 0, 1)) : non-stationary AR part from CSS
#arima(trend.error,c(1,0,1))
```

Autocorrelated Series from Yield Monitor Data

Before we consider the strictly spatial components of yield monitor data, we should first consider the temporal component. Remember, as the combined is traveling in a straight line in space, it is also moving in time, and there is a temporal mixing process involved, as grain moves through the thresher. How much of yield monitor data can be described using only autocorrelation in a single direction?

We'll use just single pass from a sample data set. We'll start with a short section.

Portion of a single pass

```
load(file="sample.dat.Rda")
sample.pass13.dat <- sample.dat[sample.dat$PassNum==13,]
sample.pass14.dat <- sample.dat[sample.dat$PassNum==14,]
sample.pass15.dat <- sample.dat[sample.dat$PassNum==15,]
head(sample.pass14.dat)
```

```
##                    Group.1 Longitude Latitude   ObjId Distance Swath
## 2633 10/12/2015 6:26:07 PM -97.59259 44.09689 15294.5    6.358     5
## 2634 10/12/2015 6:26:08 PM -97.59259 44.09691 15300.5    6.358     5
## 2635 10/12/2015 6:26:09 PM -97.59259 44.09692 15306.5    6.390     5
## 2636 10/12/2015 6:26:10 PM -97.59259 44.09694 15312.5    6.554     5
## 2637 10/12/2015 6:26:11 PM -97.59259 44.09696 15318.5    6.784     5
## 2638 10/12/2015 6:26:12 PM -97.59259 44.09698 15324.5    6.882     5
##       Yield MarkID YldMassWet Moisture Heading     LonM     LatM
## 2633 122.86  235.5     6908.2    15.35  359.20 89.46710 47.09577
## 2634 150.89  235.5     8484.2    15.35  358.75 89.39681 49.02334
## 2635 166.68  235.5     9372.4    15.35  358.67 89.33536 50.96417
## 2636 201.94  235.5    11355.0    15.35  358.75 89.27684 52.94814
## 2637 188.56  235.5    10603.0    15.35  358.99 89.24593 55.00822
## 2638 200.84  235.5    11293.0    15.35  359.63 89.26010 57.09909
```

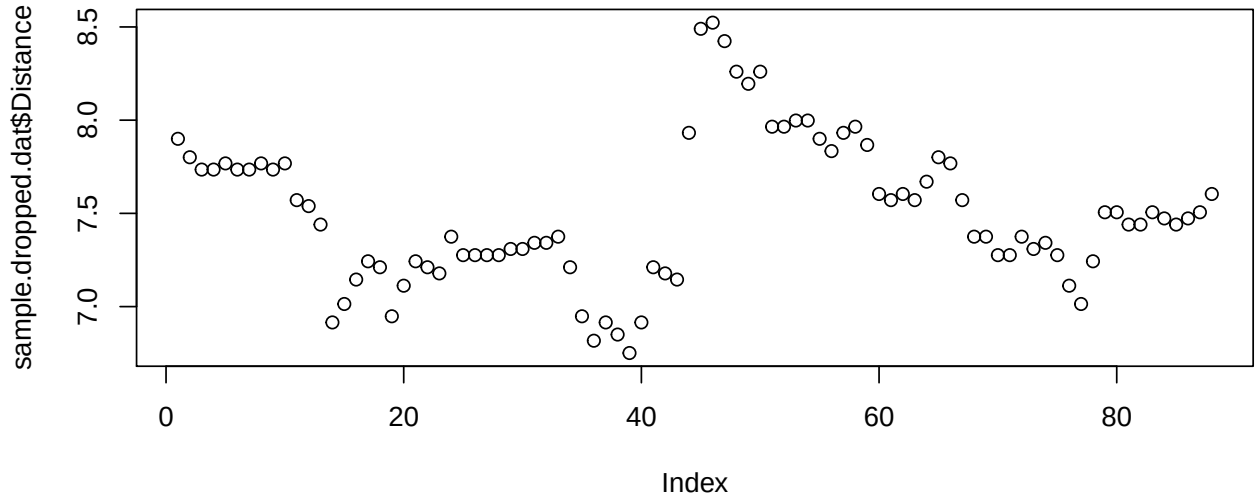
```
##               Product            DateTime   Seconds Outlier PassNum Pass
## 2633 5893VT2RIB 98day 2015-10-12 18:26:07 6055 secs   FALSE      14   14
## 2634 5893VT2RIB 98day 2015-10-12 18:26:08 6056 secs   FALSE      14   14
## 2635 5893VT2RIB 98day 2015-10-12 18:26:09 6057 secs   FALSE      14   14
## 2636 5893VT2RIB 98day 2015-10-12 18:26:10 6058 secs   FALSE      14   14
## 2637 5893VT2RIB 98day 2015-10-12 18:26:11 6059 secs   FALSE      14   14
## 2638 5893VT2RIB 98day 2015-10-12 18:26:12 6060 secs   FALSE      14   14
```

```
sample.dropped.dat <- sample.pass15.dat
```

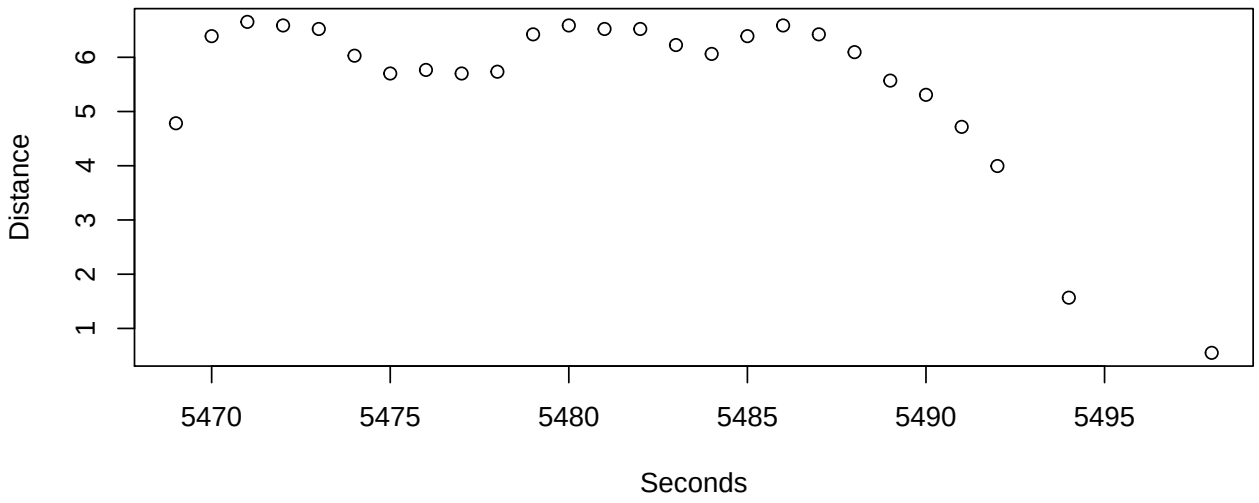
ARMA model fit

Distance

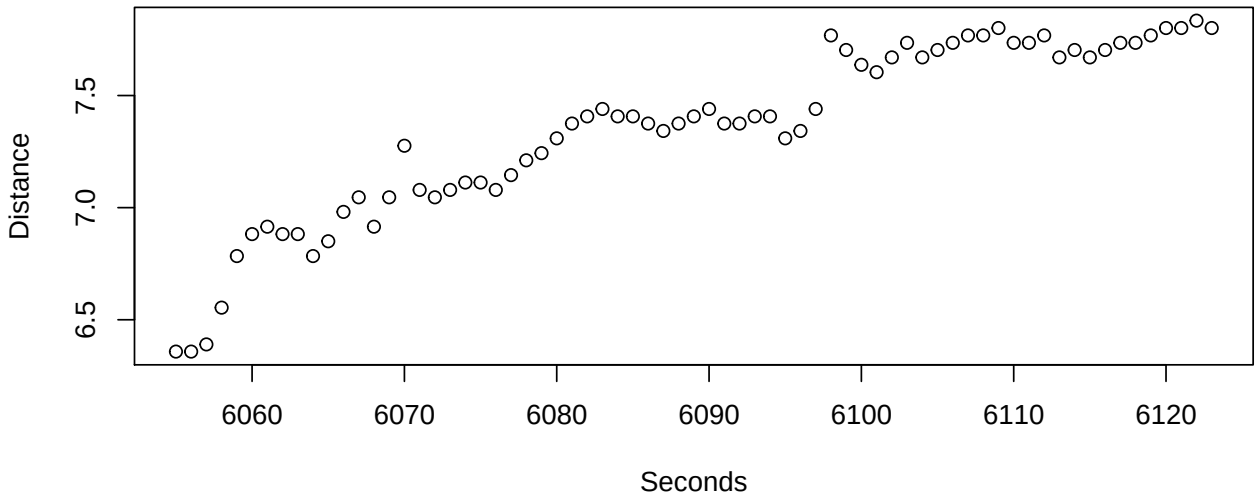
```
plot(sample.dropped.dat$Distance)
```




```
plot(Distance ~ Seconds, data=sample.pass13.dat)
```




```
plot(Distance ~ Seconds, data=sample.pass14.dat)
```




```
arima(sample.pass14.dat$Distance, order = c(1,0,0))
```

```
##
## Call:
## arima(x = sample.pass14.dat$Distance, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.9913     7.1410
## s.e.  0.0110     0.5577
##
## sigma^2 estimated as 0.006882:  log likelihood = 71.84,  aic = -137.67
```

```
arima(sample.pass14.dat$Distance, order = c(0,0,1))
```

```
##
## Call:
## arima(x = sample.pass14.dat$Distance, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       0.9176     7.3432
## s.e.  0.0379     0.0495
##
## sigma^2 estimated as 0.04664:  log likelihood = 6.92,  aic = -7.84
```

```
arima(sample.pass14.dat$Distance, order = c(1,0,1))
```

```
##
## Call:
## arima(x = sample.pass14.dat$Distance, order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.9859  0.2154     7.1665
## s.e.  0.0173  0.1398     0.4916
##
## sigma^2 estimated as 0.00668:  log likelihood = 72.89,  aic = -137.78
```

```
arima(sample.pass14.dat$Distance, order = c(2,0,2))
```

```
##
## Call:
## arima(x = sample.pass14.dat$Distance, order = c(2, 0, 2))
##
## Coefficients:
##          ar1      ar2     ma1      ma2  intercept
##       1.0950  -0.1027  0.0641  -0.1577     7.1392
## s.e.  0.5754   0.5693  0.5634   0.1458     0.5541
##
## sigma^2 estimated as 0.006525:  log likelihood = 73.62,  aic = -135.23
```

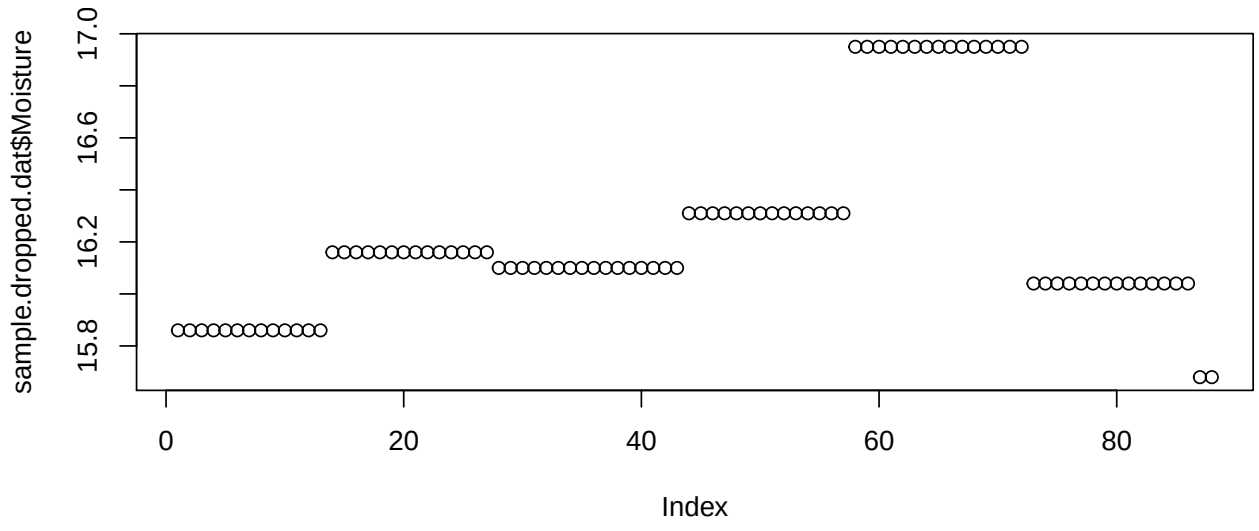
```
arima(sample.pass14.dat$Distance, order = c(3,0,3))
```

```
##
## Call:
## arima(x = sample.pass14.dat$Distance, order = c(3, 0, 3))
##
## Coefficients:
##          ar1     ar2      ar3     ma1      ma2      ma3  intercept
##       0.2896  0.8214  -0.1245  0.8718  -0.1498  -0.1251     7.1377
## s.e.  0.7073  0.3961   0.5054  0.6971   0.6115   0.1422     0.5553
##
## sigma^2 estimated as 0.006512:  log likelihood = 73.67,  aic = -131.34
```

Distance seems to be simply represented by a first order moving average process, AIC for c(0,0,1) smallest at -142.51.

Moisture

```
plot(sample.dropped.dat$Moisture)
```




```
arima(sample.dropped.dat$Moisture, order = c(1,0,0))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Moisture, order = c(1, 0, 0))
```

```
##
## Coefficients:
##          ar1  intercept
##       0.9374    16.1157
## s.e.  0.0357     0.1977
##
## sigma^2 estimated as 0.01675:  log likelihood = 54.01,  aic = -102.02
```

```
arima(sample.dropped.dat$Moisture, order = c(0,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Moisture, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       0.7703    16.2298
## s.e.  0.0463     0.0424
##
## sigma^2 estimated as 0.05092:  log likelihood = 5.69,  aic = -5.39
```

```
arima(sample.dropped.dat$Moisture, order = c(1,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Moisture, order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.9332  0.0326    16.1215
## s.e.  0.0399  0.1114     0.1934
##
## sigma^2 estimated as 0.01673:  log likelihood = 54.05,  aic = -100.11
```

```
arima(sample.dropped.dat$Moisture, order = c(2,0,2))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Moisture, order = c(2, 0, 2))
##
## Coefficients:
##          ar1      ar2      ma1     ma2  intercept
##       1.9357  -0.9430  -1.0297  0.0297    16.2111
## s.e.  0.0351   0.0355   0.1236  0.1194     0.0665
##
## sigma^2 estimated as 0.01575:  log likelihood = 55.75,  aic = -99.5
```

```
arima(sample.dropped.dat$Moisture, order = c(3,0,3))
```

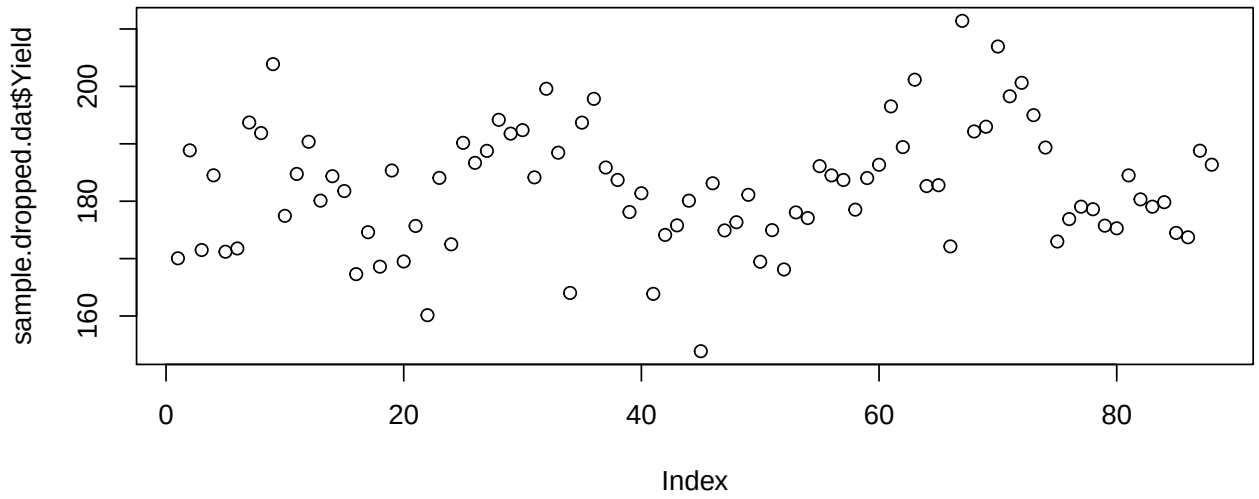
```
##
## Call:
## arima(x = sample.dropped.dat$Moisture, order = c(3, 0, 3))
##
## Coefficients:
##           ar1     ar2     ar3     ma1     ma2     ma3  intercept
##       -0.1939  0.1215  0.8511  1.2126  1.1250  0.1138    16.1241
## s.e.   0.1076  0.0686  0.0742  0.1632  0.1652  0.1407     0.1887
```

```
##
## sigma^2 estimated as 0.01537:  log likelihood = 56.2,  aic = -96.39
```

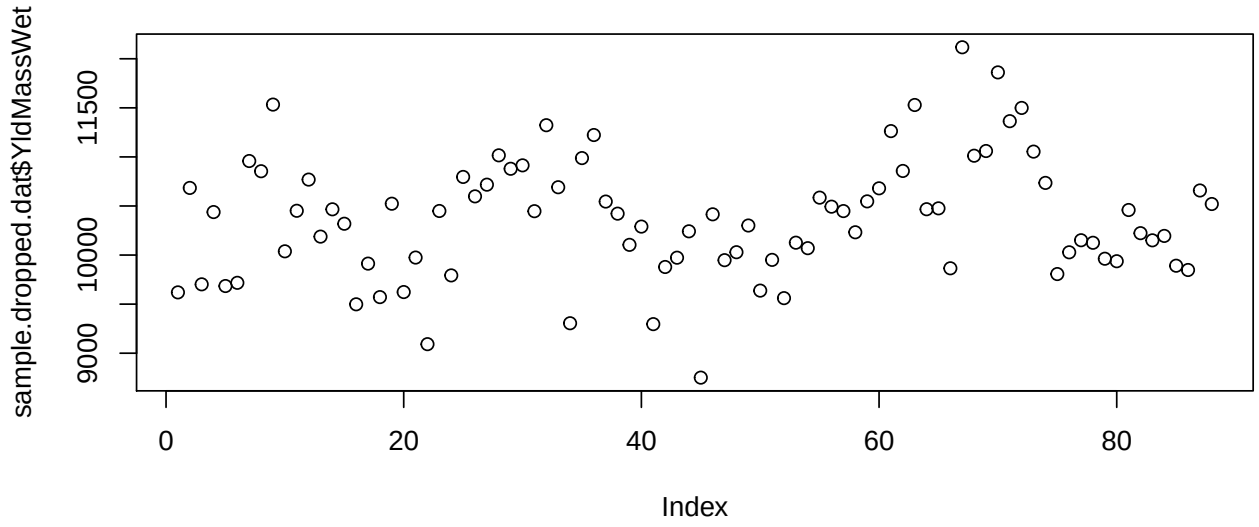
Yield

We have raw yield measurements, and yield adjusted for moisture.

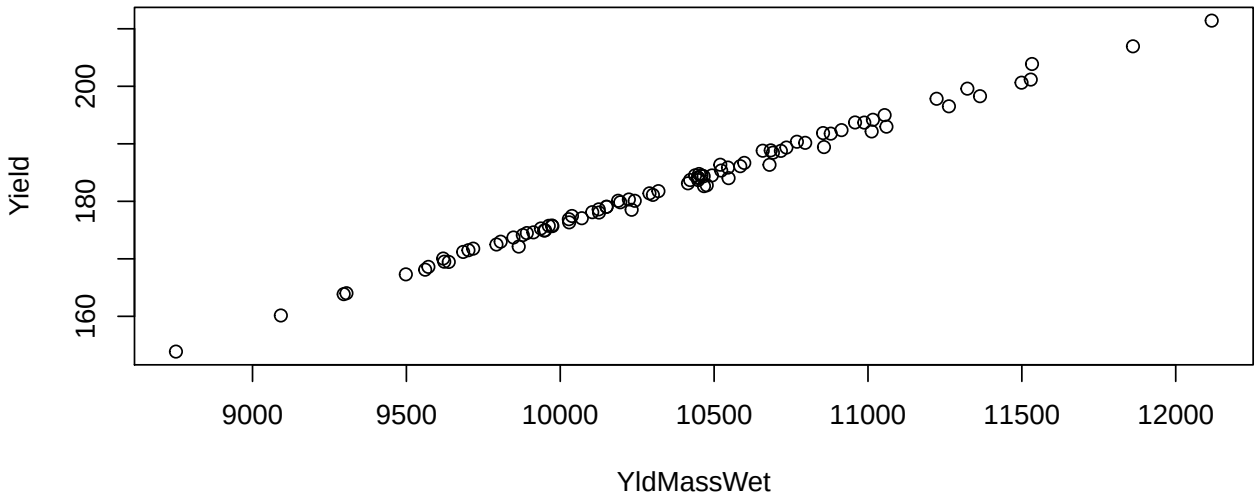
```
plot(sample.dropped.dat$Yield)
```




```
plot(sample.dropped.dat$YldMassWet)
```




```
plot(Yield ~ YldMassWet,data=sample.dropped.dat)
```



YldVolDry vs YldWet

```
arima(sample.dropped.dat$Yield, order = c(1,0,0))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Yield, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.3184   182.4071
## s.e.  0.1012     1.5447
##
## sigma^2 estimated as 98.58:  log likelihood = -326.92,  aic = 659.84
```

```
arima(sample.dropped.dat$Yield, order = c(0,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Yield, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       0.2009   182.4248
## s.e.  0.0827     1.2960
##
## sigma^2 estimated as 102.9:  log likelihood = -328.76,  aic = 663.52
```

```
arima(sample.dropped.dat$Yield, order = c(1,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Yield, order = c(1, 0, 1))
##
## Coefficients:
##          ar1      ma1  intercept
##       0.7987  -0.5323   182.3189
## s.e.  0.1055   0.1373     2.2942
```

```
##
## sigma^2 estimated as 91.16:  log likelihood = -323.54,  aic = 655.08
```

```
arima(sample.dropped.dat$YldMassWet, order = c(1,0,0))
```

```
##
## Call:
## arima(x = sample.dropped.dat$YldMassWet, order = c(1, 0, 0))
##
## Coefficients:
##          ar1   intercept
##       0.3506  10364.4918
## s.e.  0.0999     93.2519
##
## sigma^2 estimated as 326585:  log likelihood = -683.58,  aic = 1373.15
```

```
arima(sample.dropped.dat$YldMassWet, order = c(0,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$YldMassWet, order = c(0, 0, 1))
##
## Coefficients:
##          ma1   intercept
##       0.2174  10366.1857
## s.e.  0.0816     76.0602
##
## sigma^2 estimated as 344850:  log likelihood = -685.93,  aic = 1377.86
```

```
arima(sample.dropped.dat$YldMassWet, order = c(1,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$YldMassWet, order = c(1, 0, 1))
##
## Coefficients:
##          ar1      ma1   intercept
##       0.8134  -0.5301  10355.9756
## s.e.  0.0974   0.1309    141.8562
##
## sigma^2 estimated as 298849:  log likelihood = -679.75,  aic = 1367.5
```

```
arima(sample.dropped.dat$Yield, order = c(2,0,2))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Yield, order = c(2, 0, 2))
##
## Coefficients:
##          ar1     ar2     ma1      ma2  intercept
##       0.1636  0.4525  0.0414  -0.1594   182.3977
## s.e.  0.3980  0.3234  0.4219   0.2704     2.2477
##
## sigma^2 estimated as 88.83:  log likelihood = -322.44,  aic = 656.88
```

```
arima(sample.dropped.dat$YldMassWet, order = c(2,0,2))
```

```
##
```

```
## Call:
## arima(x = sample.dropped.dat$YldMassWet, order = c(2, 0, 2))
##
## Coefficients:
##          ar1     ar2     ma1      ma2   intercept
##       0.1899  0.4539  0.0293  -0.1480  10361.5590
## s.e.  0.3925  0.3237  0.4158   0.2641    138.0908
##
## sigma^2 estimated as 290679:  log likelihood = -678.56,  aic = 1369.13
```

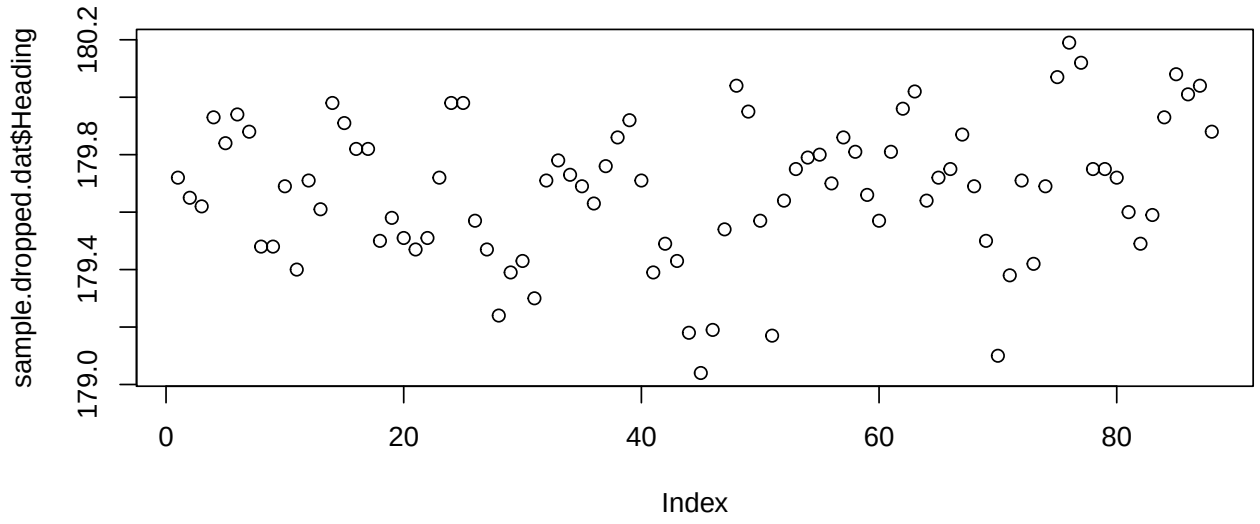
```
arima(sample.dropped.dat$Yield, order = c(3,0,3))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Yield, order = c(3, 0, 3))
##
## Coefficients:
##           ar1     ar2     ar3     ma1     ma2      ma3  intercept
##       -0.8714  0.3997  0.7038  1.1829  0.1794  -0.4904   182.4110
## s.e.   0.1460  0.1920  0.1112  0.1716  0.2670   0.1616     2.2641
##
## sigma^2 estimated as 80.41:  log likelihood = -320.05,  aic = 656.11
```

```
arima(sample.dropped.dat$YldMassWet, order = c(3,0,3))
```

```
##
## Call:
## arima(x = sample.dropped.dat$YldMassWet, order = c(3, 0, 3))
##
## Coefficients:
##           ar1     ar2     ar3     ma1     ma2      ma3   intercept
##       -0.8608  0.4246  0.7204  1.1879  0.1864  -0.4860  10361.8937
## s.e.   0.1370  0.1775  0.1044  0.1646  0.2550   0.1544    139.6292
##
## sigma^2 estimated as 262891:  log likelihood = -676.16,  aic = 1368.32
```

```
plot(sample.dropped.dat$Heading)
```




```
arima(sample.dropped.dat$Heading, order = c(1,0,0))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Heading, order = c(1, 0, 0))
##
## Coefficients:
##          ar1  intercept
##       0.6094   179.6855
## s.e.  0.0834     0.0512
##
## sigma^2 estimated as 0.03648:  log likelihood = 20.58,  aic = -35.17
```

```
arima(sample.dropped.dat$Heading, order = c(0,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Heading, order = c(0, 0, 1))
##
## Coefficients:
##          ma1  intercept
##       0.6175   179.6817
## s.e.  0.0722     0.0330
##
## sigma^2 estimated as 0.03685:  log likelihood = 20.13,  aic = -34.26
```

```
arima(sample.dropped.dat$Heading, order = c(1,0,1))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Heading, order = c(1, 0, 1))
##
## Coefficients:
##          ar1     ma1  intercept
##       0.4156  0.3466   179.6834
## s.e.  0.1362  0.1342     0.0448
##
## sigma^2 estimated as 0.03405:  log likelihood = 23.55,  aic = -39.1
```

```
arima(sample.dropped.dat$Heading, order = c(2,0,2))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Heading, order = c(2, 0, 2))
##
## Coefficients:
##          ar1      ar2     ma1     ma2  intercept
##       0.5052  -0.2999  0.2571  0.3133    179.683
## s.e.  0.3056   0.1909  0.2852  0.2460      0.038
##
## sigma^2 estimated as 0.03288:  log likelihood = 25.03,  aic = -38.07
```

```
arima(sample.dropped.dat$Heading, order = c(3,0,3))
```

```
##
## Call:
## arima(x = sample.dropped.dat$Heading, order = c(3, 0, 3))
##
## Coefficients:
```

```
##          ar1     ar2      ar3     ma1      ma2      ma3  intercept
##       0.2795  0.6132  -0.4851  0.5078  -0.5304  -0.0382   179.6779
## s.e.  0.2528  0.1695   0.1352  0.2743   0.3241   0.1700     0.0306
##
## sigma^2 estimated as 0.03178:  log likelihood = 25.83,  aic = -35.67
```