R, Reserving, Linear Regression & MRMR

Brian A. Fannin

September 2, 2013

Agenda

- Introducing MRMR
- Data visualization
- Linear modeling
- Fit diagnostics
- Projection
- Another view of regression
- Further

Introducing MRMR

MRMR is another R package for use in analyzing reserves.
MRMR was heavily influenced by the following:

- Andrew Gelman and Jennifer Hill, "Data Analysis Using Regression and Multilevel/Hierarchical Models"
- ggplot2 and Hadley Wickham
- Leigh Halliwell and Judge et al

MRMR Structure

MRMR supports three S4 classes: Triangle, TriangleModel and TriangleProjection. These have a rough correspondence to the behavior of functions lm, glm and lme4.

	R	MRMR
Data storage	Data frame	Triangle
Model	Function lm (S3 object)	TriangleModel
Project	Function predict (vector)	TriangleProjection

Startup MRMR

```
library(MRMR)
`?`(MRMR)
```

Basic requirements

A triangle object must possess the following data elements:

- Temporal dimensions for origin period, development lag and evaluation date. These are stored as lubridate objects.
- Measures
  - Stochastic - Loss, claim, etc. These are time series variables and candidates for prediction. MRMR will adjust these so that incremental, cumulative and prior cumulative columns are formed.
  - Static - Typically exposure variables. These will not be adjusted. These are very good candidates for predictors.
- One or more grouping elements. This is currently not implemented, but is reserved for future use.

A brief word about lubridate

lubridate is a package with many routines to aid in working with dates.

lubridate examples

```
aDate = mdy("06-30-2012")
day(aDate) = 6
aDate + years(1)

## [1] "2013-06-06 UTC"

myPeriod = months(6)
myPeriod/months(1)

## estimate only:  convert to intervals for accuracy

## [1] 6
```

Very quick lubridate exercise

How would you use lubridate to generate a sequence of the 15th of every month for the year of 2010?

Result

```
aDate = mdy("1-15-2010")
someDates = aDate + months(0:11)
someDates
```

```
##  [1] "2010-01-15 UTC" "2010-02-15 UTC"
##  [3] "2010-03-15 UTC" "2010-04-15 UTC"
##  [5] "2010-05-15 UTC" "2010-06-15 UTC"
##  [7] "2010-07-15 UTC" "2010-08-15 UTC"
##  [9] "2010-09-15 UTC" "2010-10-15 UTC"
## [11] "2010-11-15 UTC" "2010-12-15 UTC"
```

Very basic reserving data

```
AccidentYear = c(2002, 2002, 2002, 2003,
    2003, 2004)

Month = c(12, 24, 36, 12, 24, 12)

Paid = c(2318, 7932, 13822, 1743, 6240, 2221)

EP = c(61183, 61183, 61183, 69175, 69175,
    99322)

df = data.frame(AccidentYear = AccidentYear,
    Month = Month, Paid = Paid, EP = EP)
head(df)
```

Moving the data into a Triangle object

```
myTriangle = newTriangle(TriangleData = df,
    OriginPeriods = AccidentYear, DevelopmentLags = Month,
    Cumulative = TRUE, StochasticMeasures = c("Paid"),
    StaticMeasures = c("EP"), Verbose = FALSE)
```

What's in a Triangle object?

One may identify the components of a list object by using the name function. For an S4 object, use the function slotNames.

```
slotNames(myTriangle)

## [1] "TriangleData"
## [2] "TriangleName"
## [3] "OriginPeriodType"
## [4] "DevelopmentInterval"
## [5] "StaticMeasures"
## [6] "StochasticMeasures"
## [7] "Groups"
```

To access a slot, use the commercial a operator

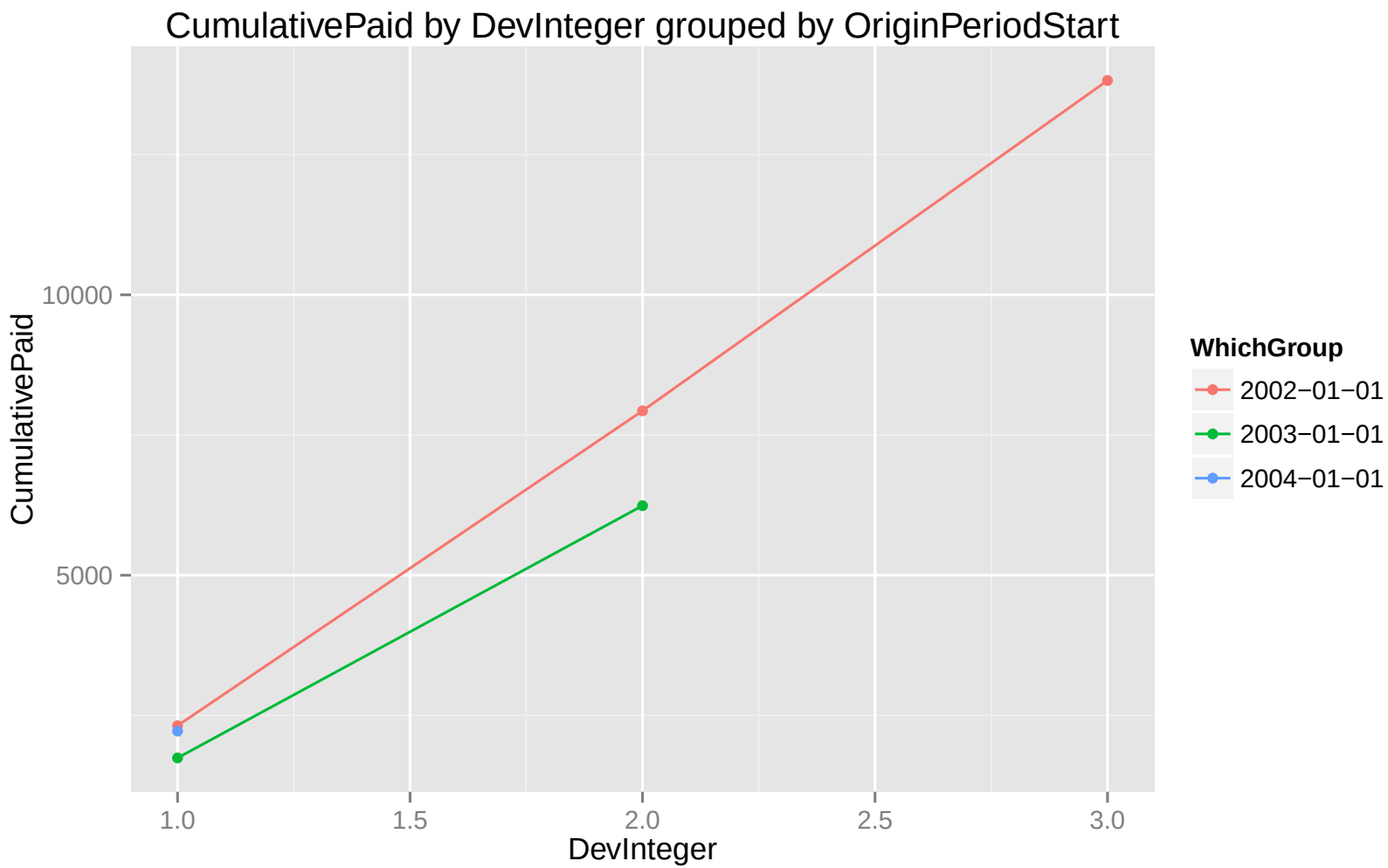
What sort of data frame have I created?

```
names(myTriangle@TriangleData)
```

```
##  [1] "OriginPeriod"
##  [2] "DevelopmentLag"
##  [3] "EvaluationDate"
##  [4] "DevInteger"
##  [5] "OriginPeriodStart"
##  [6] "OriginPeriodEnd"
##  [7] "CalendarPeriodStart"
##  [8] "CalendarPeriodEnd"
##  [9] "CalendarPeriod"
## [10] "Group"
## [11] "EP"
## [12] "CumulativePaid"
## [13] "IncrementalPaid"
## [14] "PriorPaid"
```

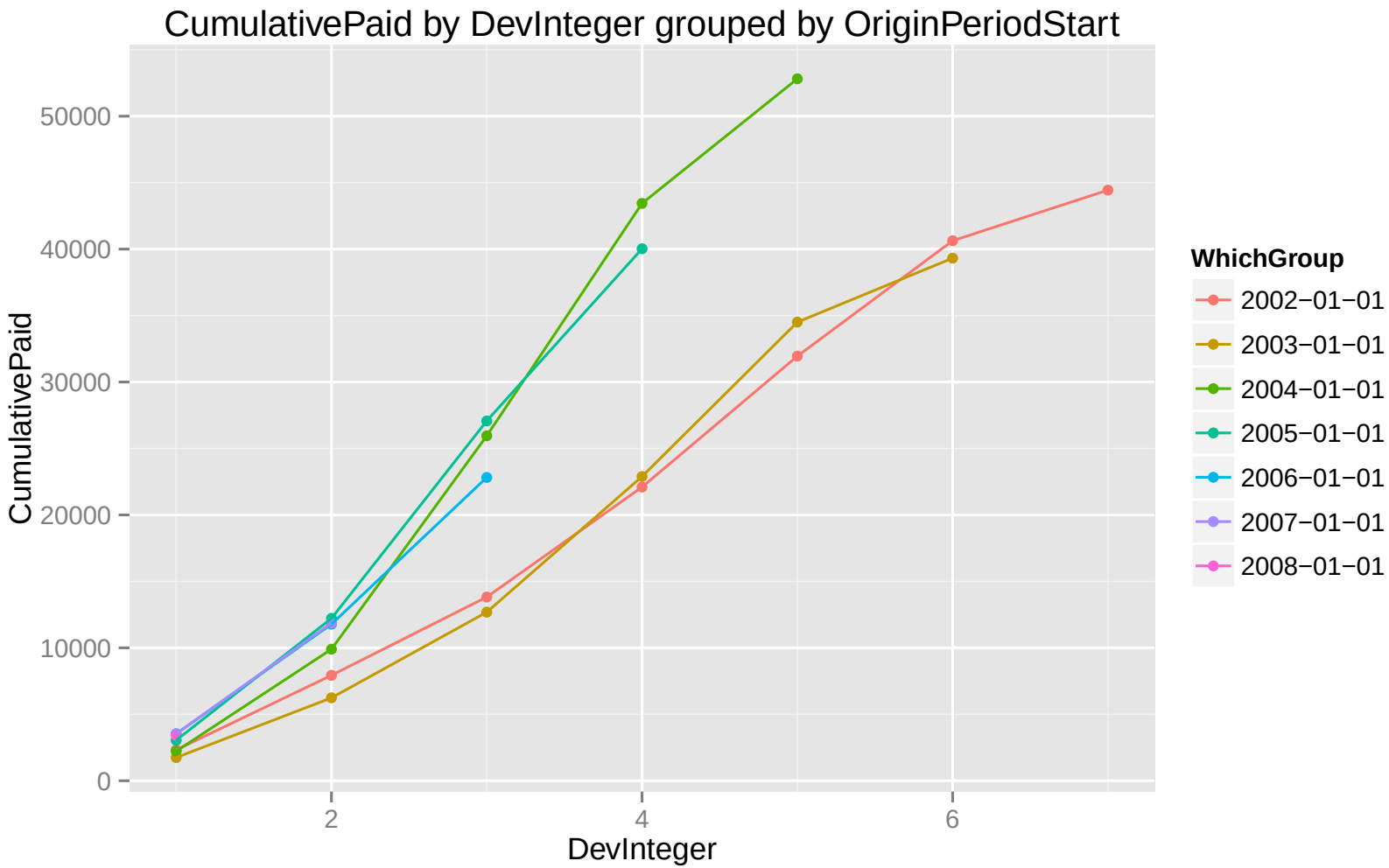
A very basic plot

```
plotTriangle(myTriangle, Predictor = "DevInteger",
    Response = "CumulativePaid")
```



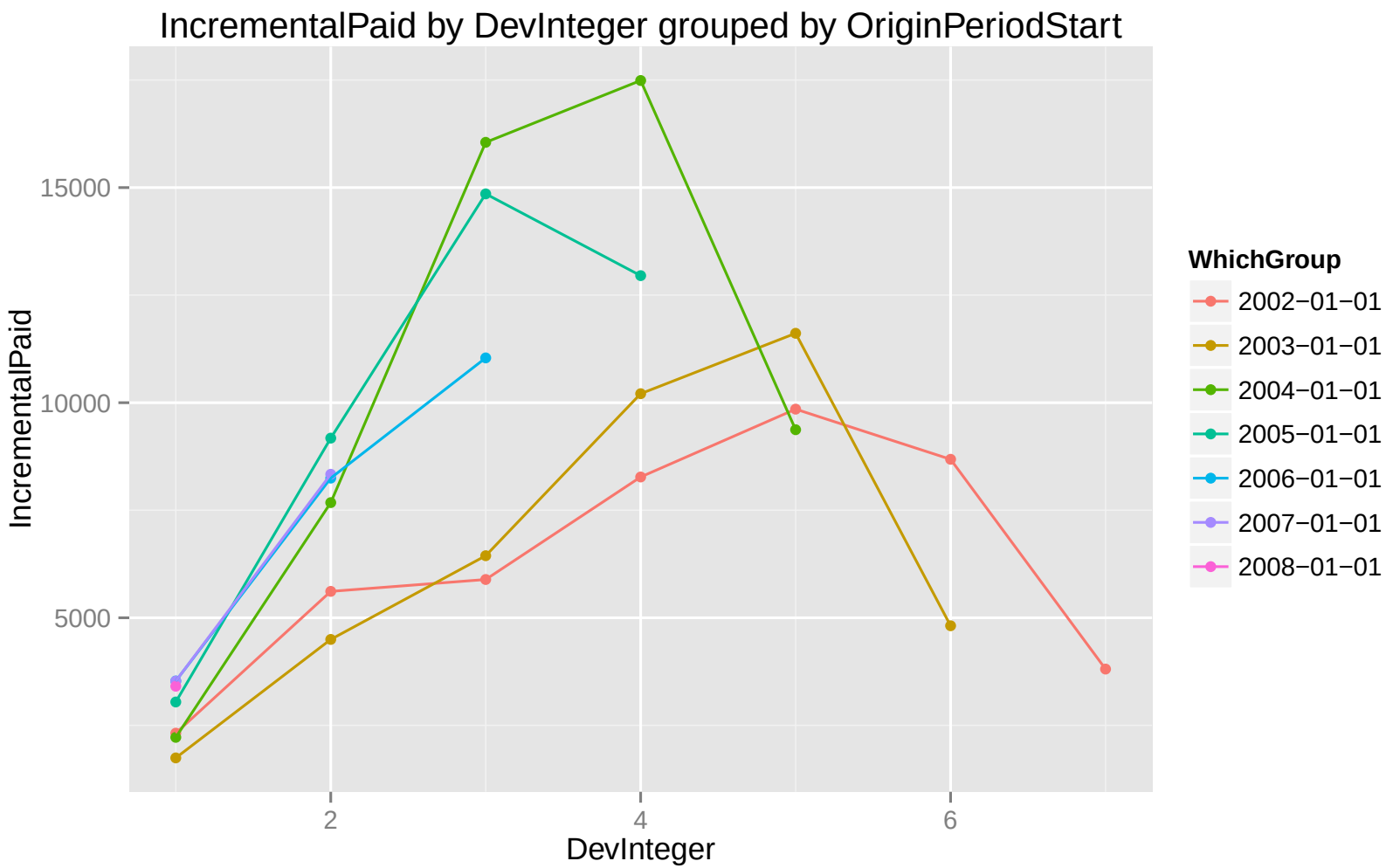
Something more complex

```
data(Friedland)
plotTriangle(Friedland, Predictor = "DevInteger",
    Response = "CumulativePaid")
```



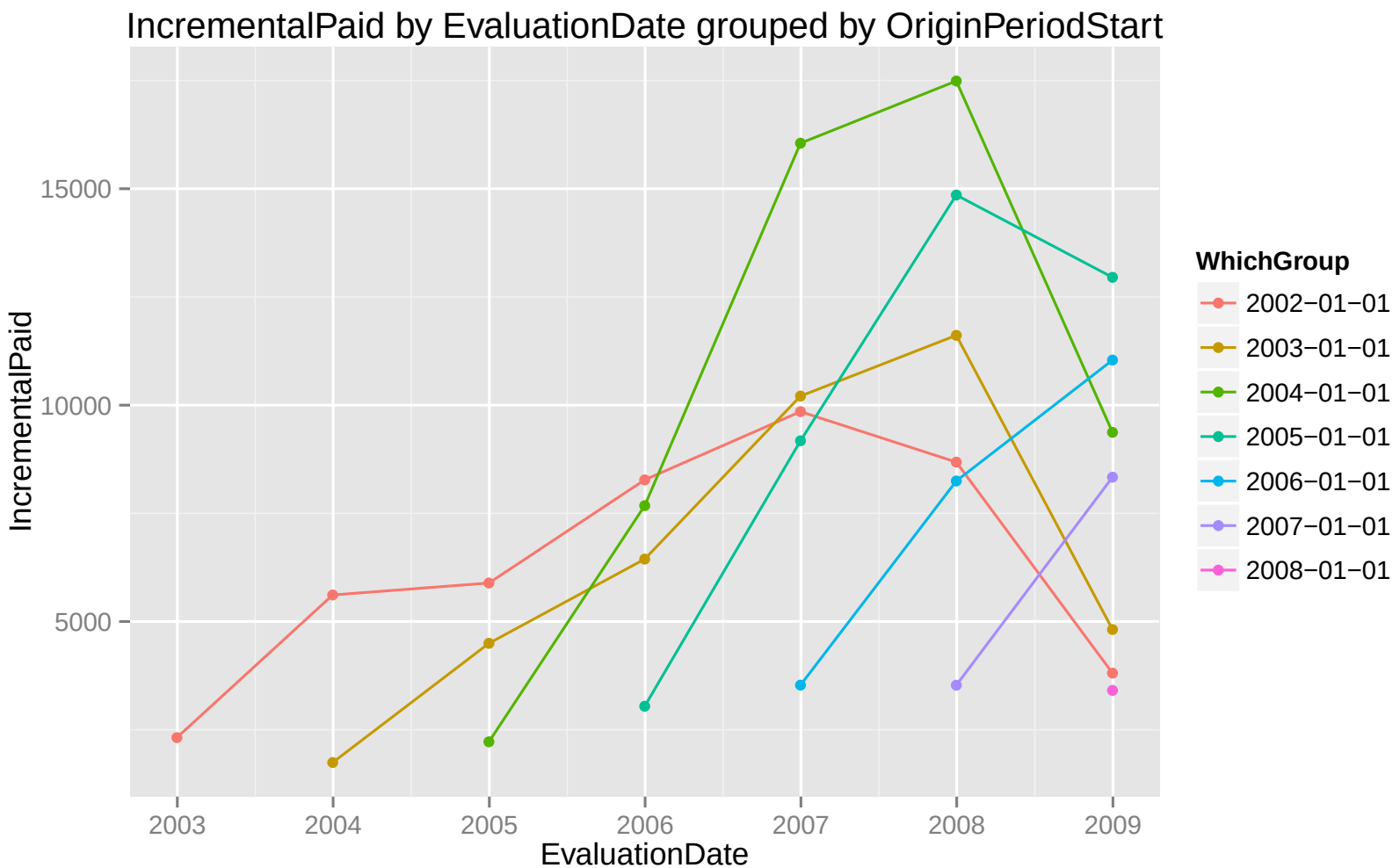
Change the response term

```
plotTriangle(Friedland, Predictor = "DevInteger",
    Response = "IncrementalPaid")
```



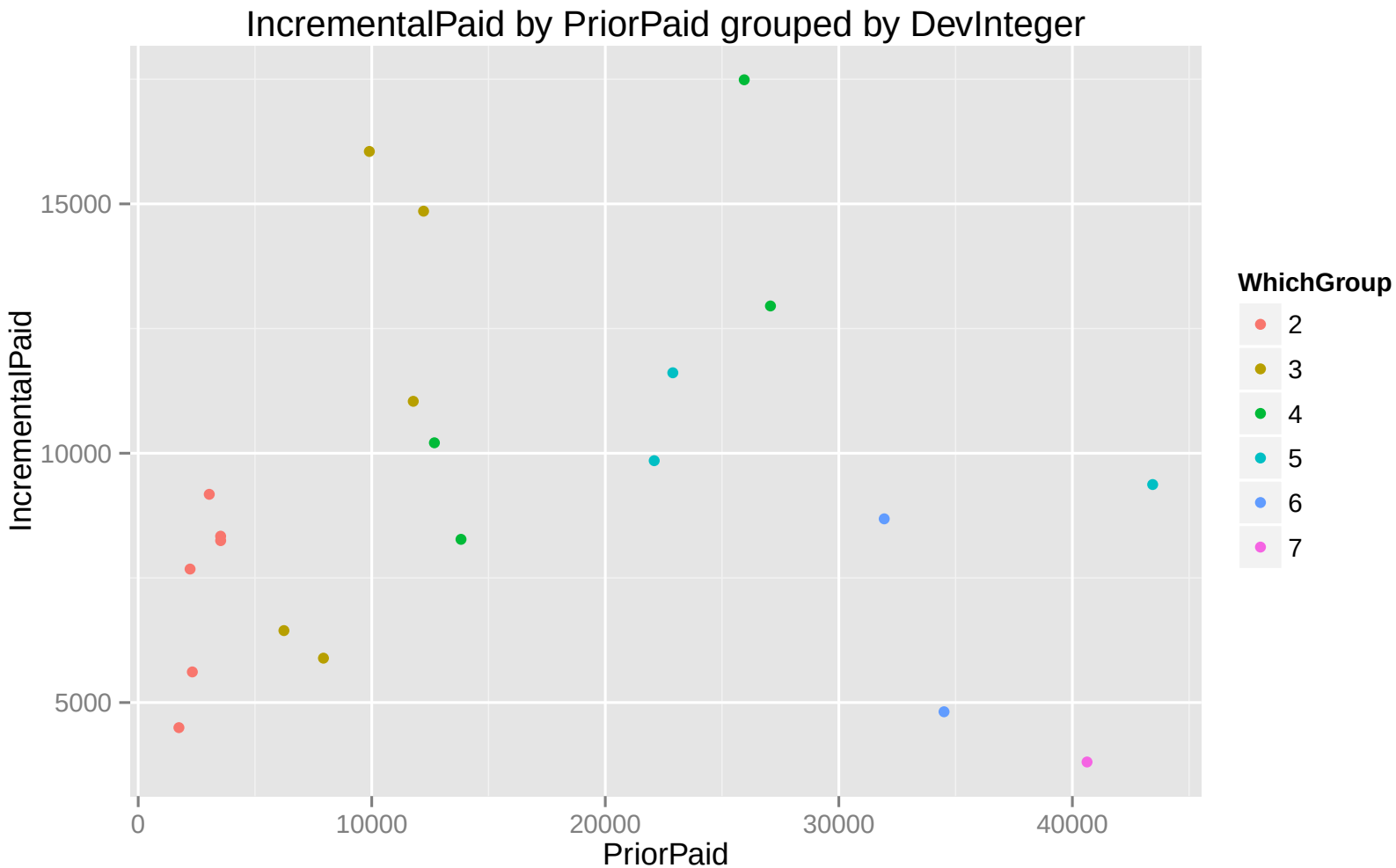
Change the time axis

```
plotTriangle(Friedland, Predictor = "EvaluationDate",
    Response = "IncrementalPaid")
```



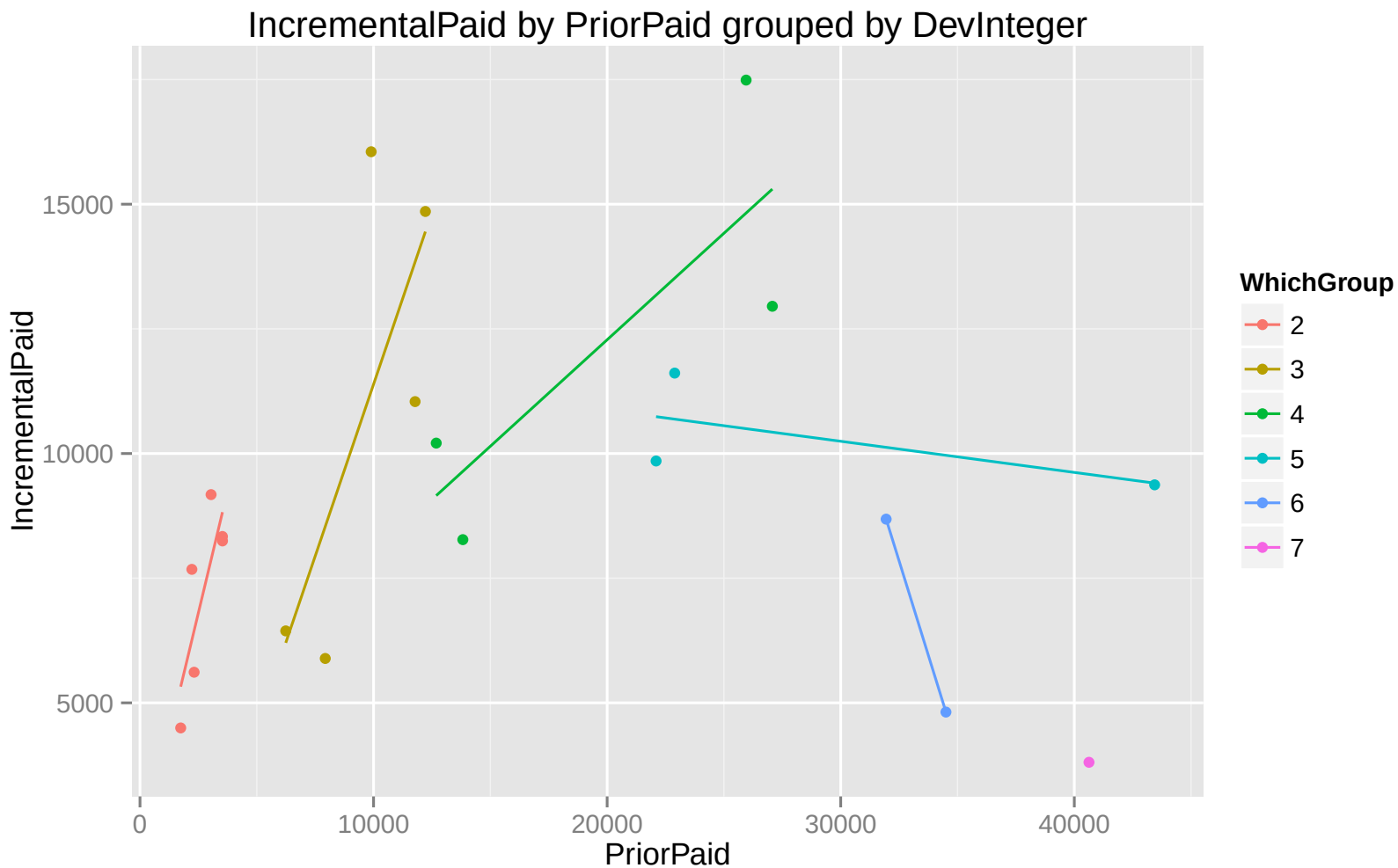
Change the grouping dimension

```
plotTriangle(Friedland, Predictor = "PriorPaid",
    Response = "IncrementalPaid", Group = "DevInteger",
    Lines = FALSE)
```



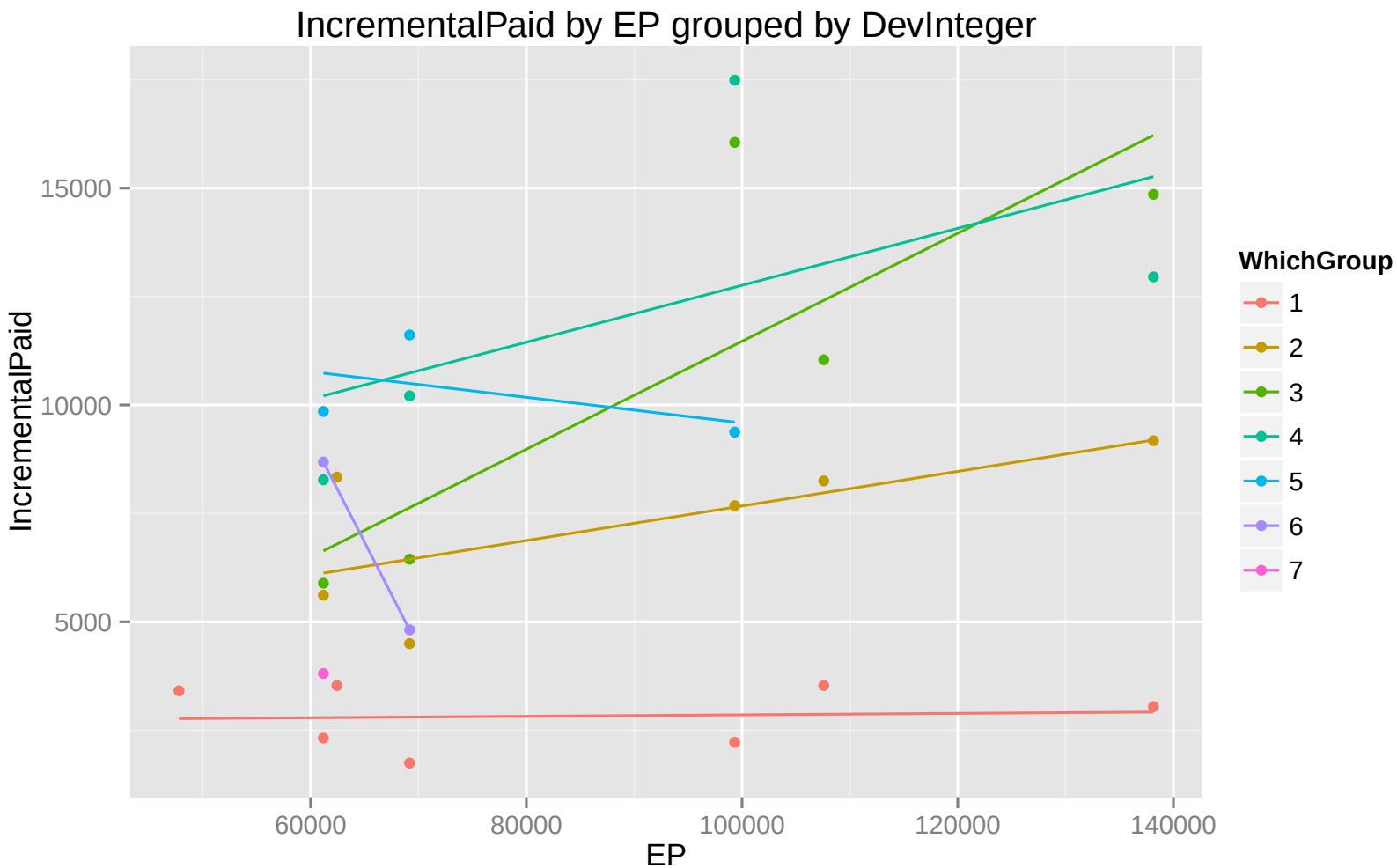
Add fit lines

```
plotTriangle(Friedland, Predictor = "PriorPaid",
    Response = "IncrementalPaid", Group = "DevInteger",
    Lines = FALSE, FitLines = TRUE)
```



Change the predictor variable

```
plotTriangle(Friedland, Response = "IncrementalPaid",
    Predictor = "EP", Group = "DevInteger",
    Lines = FALSE, FitLines = TRUE)
```



Fit a model

```
PaidAM = newTriangleModel(Triangle = Friedland,
    Response = "IncrementalPaid", Predictor = "EP",
    FitCategory = "DevInteger", Tail = 6)
```

Visualization is closely related to a model

```
plotTriangle(Friedland, Response = "IncrementalPaid",
    Predictor = "EP", Group = "DevInteger",
    Lines = FALSE, FitLines = TRUE)
PaidAM = newTriangleModel(Friedland, Response = "Incrementa
    Predictor = "EP", FitCategory = "DevInteger",
    Tail = 6)
```

Linear regression in R

```
set.seed(1234)
N = 100
e = rnorm(N, mean = 0, sd = 1)
B0 = 5
B1 = 1.5

X1 = rep(seq(1, 10), 10)
Y = B0 + B1 * X1 + e

df = data.frame(Y = Y, X1 = X1, e = e)
```

Fitting a linear model

```
myFit = lm(Y ~ X1, data = df)
```

Diagnostic output

```
summary(myFit)


##
## Call:
## lm(formula = Y ~ X1, data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.188 -0.742 -0.228  0.629  2.709
##
## Coefficients:
##             Estimate Std. Error t value
## (Intercept)   4.8383     0.2181    22.2
## X1            1.5009     0.0351    42.7
##             Pr(>|t|)
## (Intercept)   <2e-16 ***
## X1            <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.01 on 98 degrees of freedom
## Multiple R-squared:  0.949, Adjusted R-squared:  0.948
## F-statistic: 1.82e+03 on 1 and 98 DF,  p-value: <2e-16
```

Formulas in R

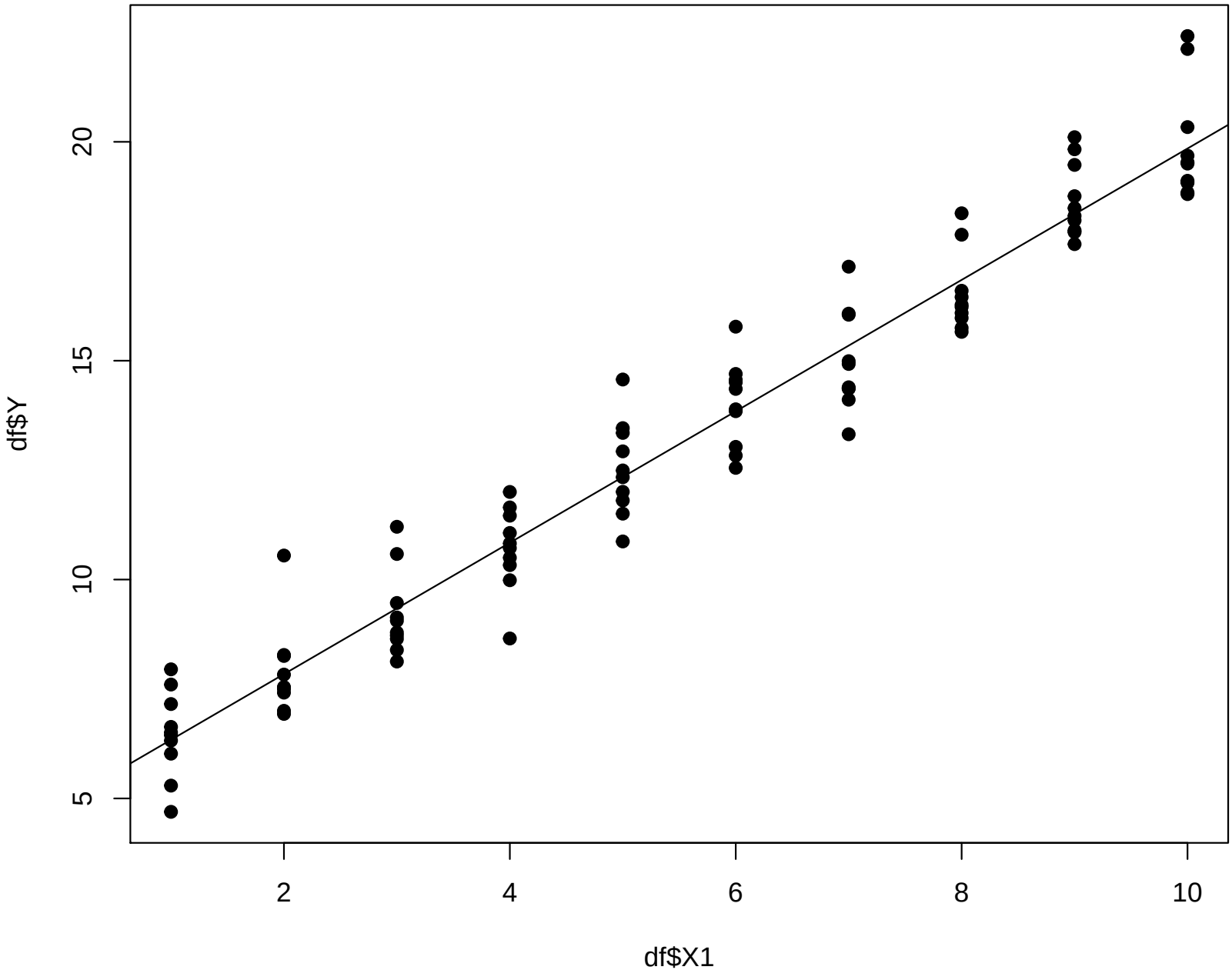
- The '~' is typically read "is modeled as"
- The '+' operator adds new predictor variables to the model
- To use operators normally, enclose them in I()
- An intercept is always assumed. To remove it, add '+ 0' or '- 1' to the formula
- The ':' operator controls interactions between variables.

Some examples

```
# The 1 is not necessary
lm(Y ~ 1 + X1, data = df)

# This is the same as above
lm(Y ~ X1, data = df)

lm(Y ~ 0 + X1, data = df)  #No intercept

lm(Y ~ X1 + X2, data = df)  #Two predictors

# Two predictors and an interaction
lm(Y ~ X1 + X2 + X1:X2, data = df)

# Use the operators normally
lm(Y ~ I(X1/X2), data = df)
```

Plot the data

```
plot(df$X1, df$Y, pch = 19)

# To plot the fit line we can type
# either this:
abline(myFit$coefficients[[1]], myFit$coefficients[[2]])

# Or this:
lines(df$X1, predict(myFit))
```


df$Y
df$X1
5
10
15
20
2
4
6
8
10

More than one variable

```
B2 = -3
df$X2 = rep(seq(-20, -11), 10)
df$Y = with(df, B0 + B1 * X1 + B2 * X2)
myFit2 = lm(Y ~ X1 + X2, data = df)
```

How would we plot this data?

A spurious variable

```
df$spurious = runif(N, min = -5, max = 5)
myFit3 = lm(Y ~ X1 + spurious, data = df)
```

How would you determine whether or not to include the spurious predictor?

Linear regression assumptions

1. Linear model with specified parameters
   - Significance of individual model factors
   - Significance of model
2. Functional form of errors
3. Independence of errors
   - (Serial) correlation of errors
   - Homoskedasticity

Linear regression assumptions

1. Linear model with specified parameters
   - Significance of individual model factors
   - Significance of model
2. Functional form of errors
3. Independence of errors
   - (Serial) correlation of errors
   - Homoskedasticity

Significance of model factors

Each model factor follows a t distribution, whose parameters depends on the underlying data. The "t value" reported by R is the ratio of the mean to the standard error. As a simple rule of thumb, any time a t-stat divided by its standard error is less than 2, one should reevaluate whether that factor improves the overall model. Put differently, such a low "t value" makes it difficult to reject the null hypothesis that the mean of the model factor is zero.

Significance of model factors (cont'd)

```
dfCoef = summary(myFit)$coefficients
dfCoef
```

```
##             Estimate Std. Error t value
## (Intercept)    4.838    0.21808   22.19
## X1             1.501    0.03515   42.70
##               Pr(>|t|)
## (Intercept) 5.423e-40
## X1          3.863e-65
```

Significance of model factors (cont'd)

```
spuriousFit = lm(Y ~ spurious, data = df)
dfCoef = summary(spuriousFit)$coefficients
dfCoef
```

```
##             Estimate Std. Error t value
## (Intercept)  59.7520     0.4328 138.052
## spurious     -0.1712     0.1640  -1.044
##                Pr(>|t|)
## (Intercept) 4.395e-114
## spurious     2.992e-01
```

Two different reserving models

```
PaidCL = newTriangleModel(Friedland, Response = "Incrementa
    Predictor = "PriorPaid", FitCategory = "DevInteger",
    Tail = 6)
PaidAM = newTriangleModel(Triangle = Friedland,
    Response = "IncrementalPaid", Predictor = "EP",
    FitCategory = "DevInteger", Tail = 6)
```

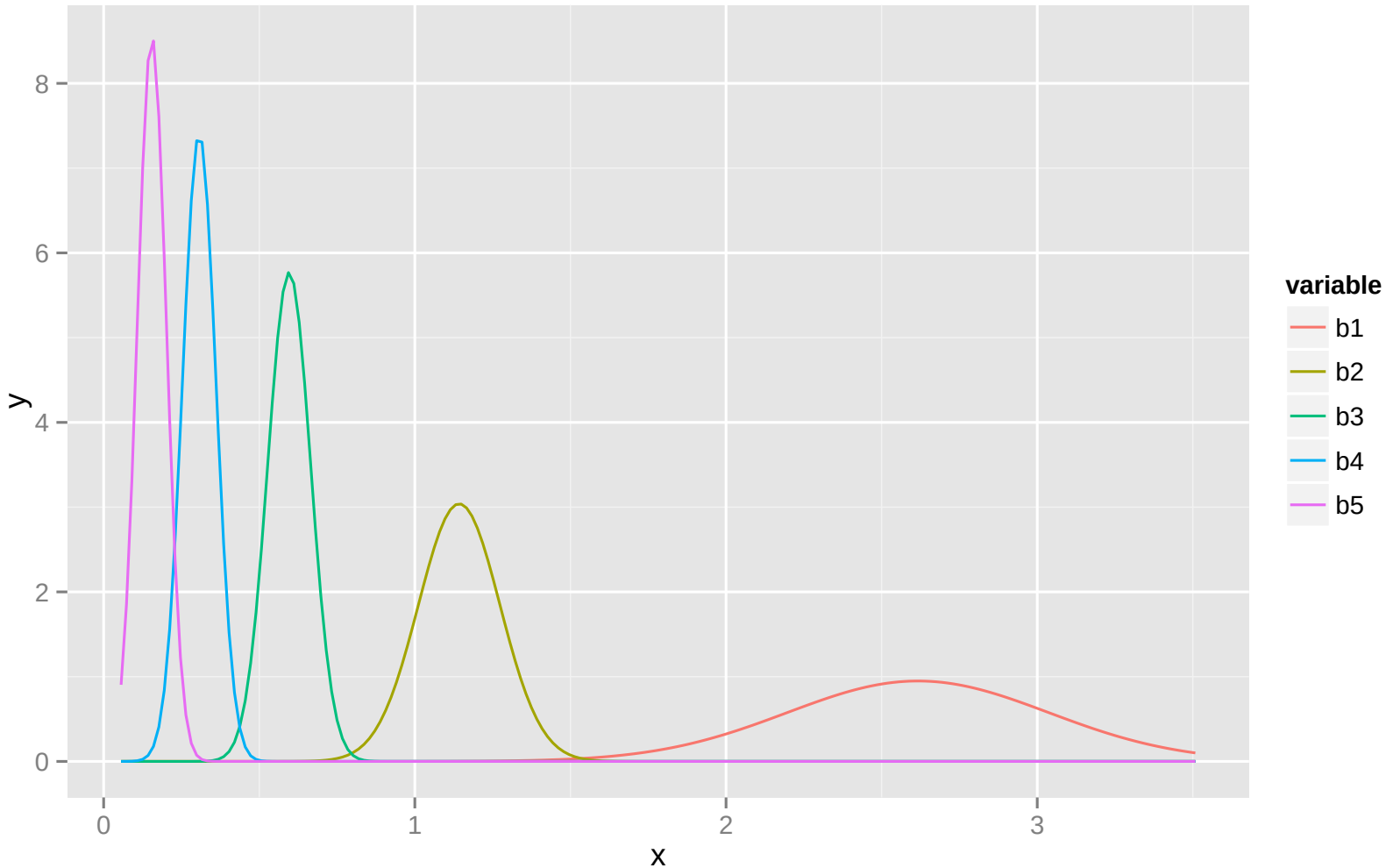
The first model corresponds to the traditional multiplicative chain ladder as applied to paid losses. The default is to have no weighting.
The second model is the additive model. This is described in Stanard and elsewhere.

```
summary(PaidCL@Fit)$coefficients[, 1:2]
```

```
##                             Estimate
## PriorPaid:FitCategory2        2.6167
## PriorPaid:FitCategory3        1.1425
## PriorPaid:FitCategory4        0.5962
## PriorPaid:FitCategory5        0.3072
## PriorPaid:FitCategoryTail     0.1549
##                             Std. Error
## PriorPaid:FitCategory2         0.42044
## PriorPaid:FitCategory3         0.13123
## PriorPaid:FitCategory4         0.06912
## PriorPaid:FitCategory5         0.05383
## PriorPaid:FitCategoryTail      0.04664
```

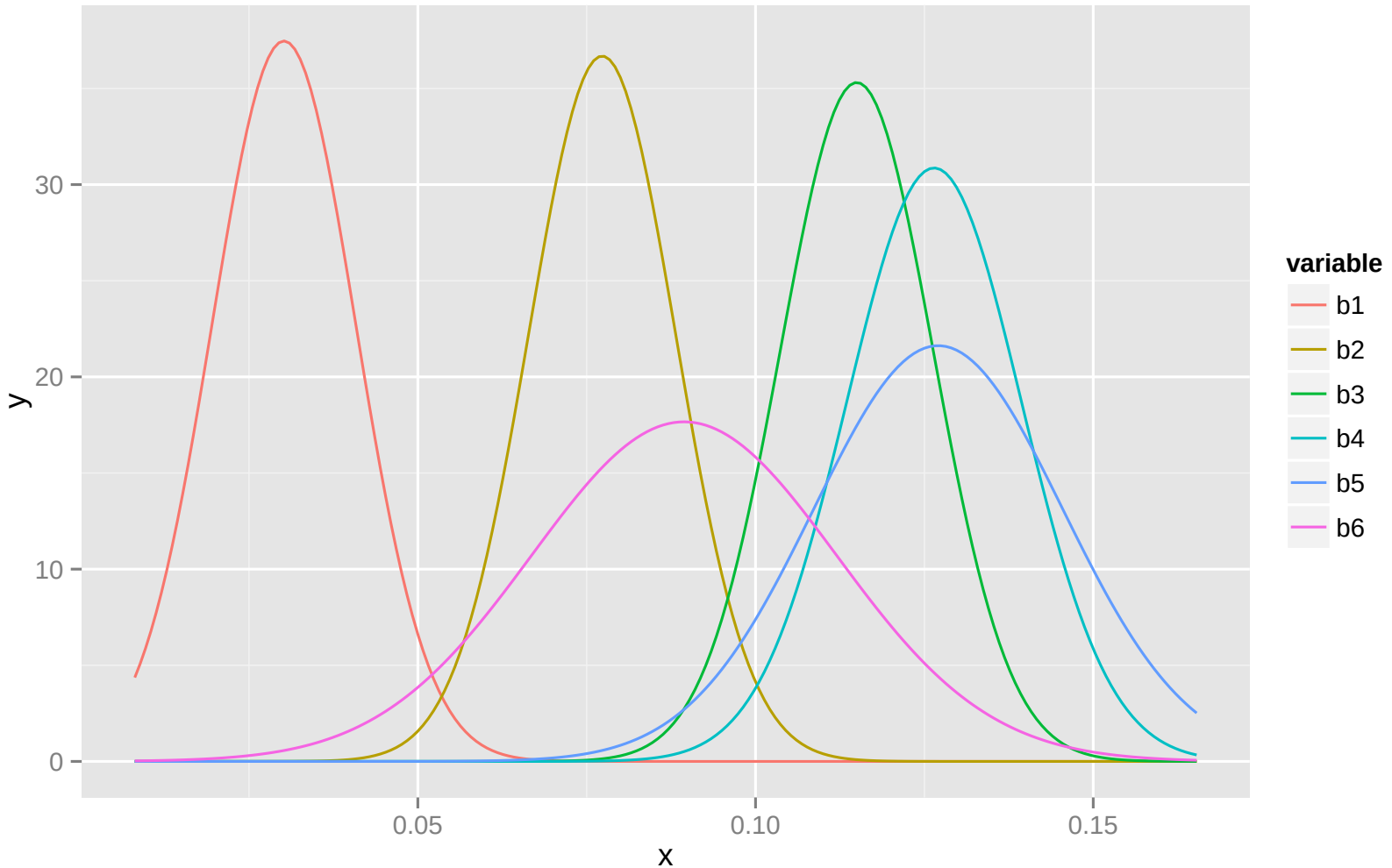
Observe the model factors - Chain Ladder

```
PlotModelFactors(PaidCL)
```



Observe the model factors - Additive

```
PlotModelFactors(PaidAM)
```



Linear regression assumptions

1. Linear model with specified parameters
   - Significance of individual model factors
   - Significance of model
2. Functional form of errors
3. Independence of errors
   - (Serial) correlation of errors
   - Homoskedasticity

Significance of model

Several metrics

1. R-squared
2. F-statistic
3. AIC
4. Penalized measures

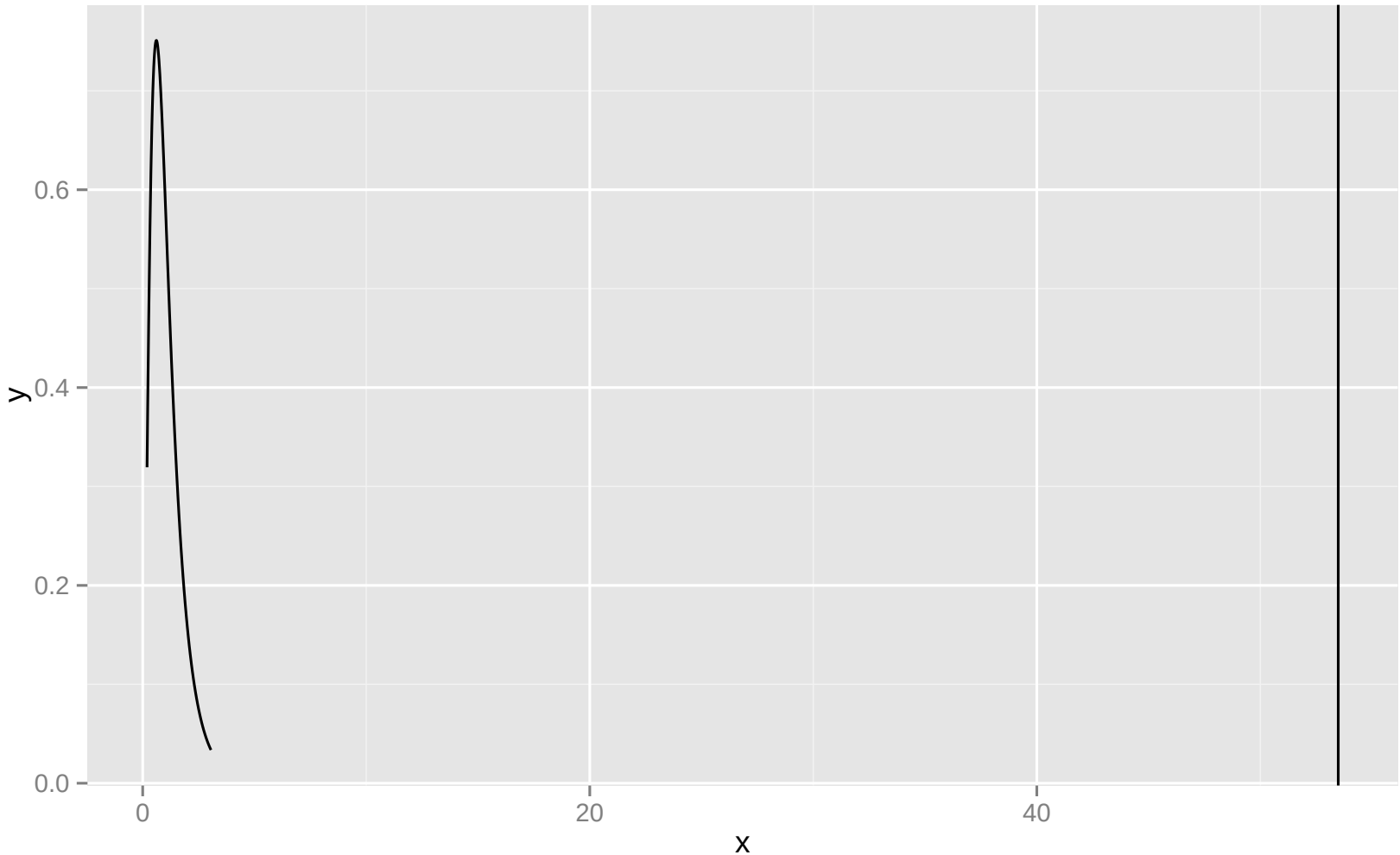
Diagnostics

```
summary(myFit)$r.squared

## [1] 0.949

summary(myFit)$fstatistic

## value numdf dendf
##  1824     1    98
```

Be careful of both of these statistics. Always visualize your data!

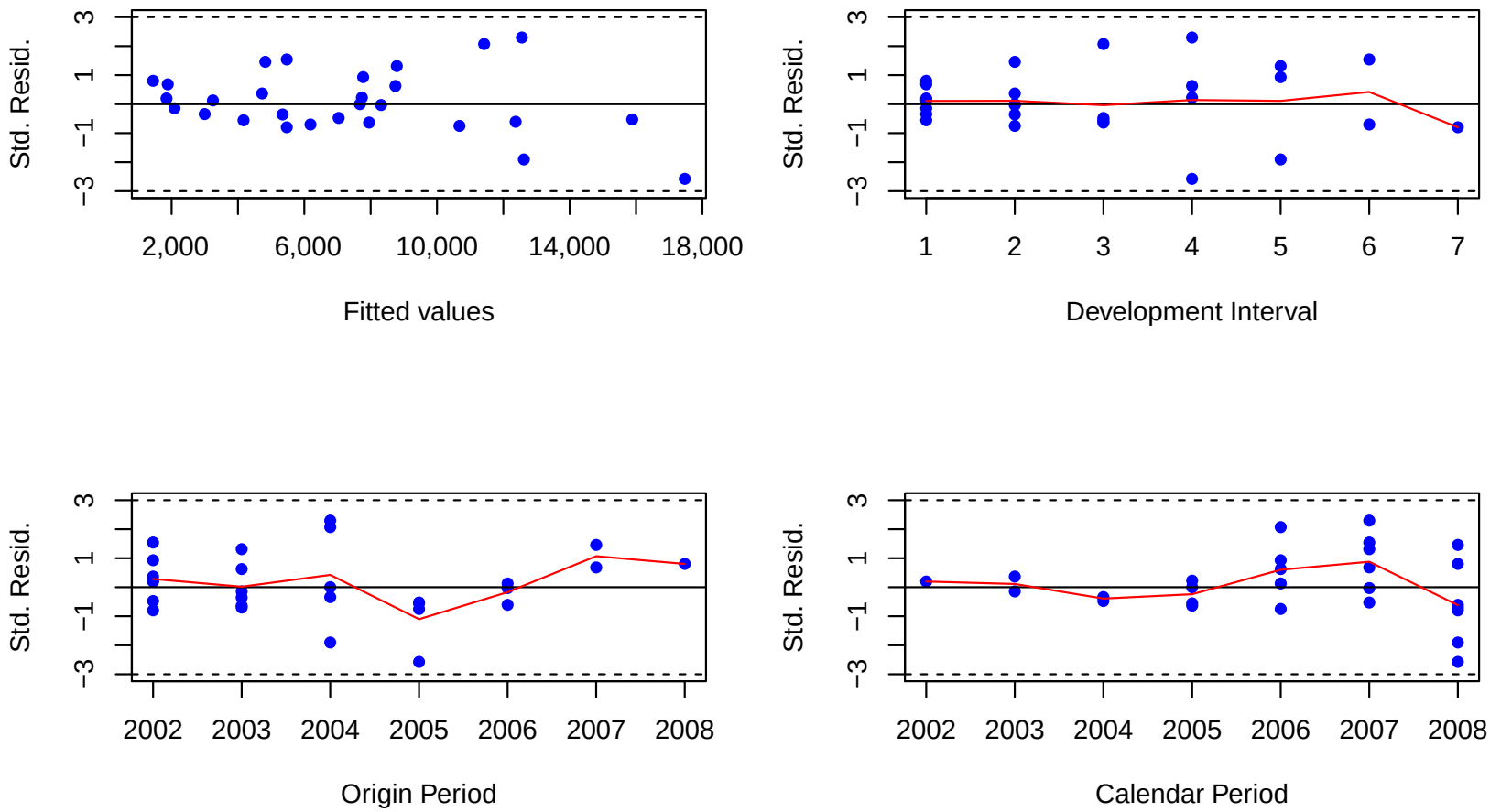
The F stat distribution looks good

```
PlotModelGoF(PaidAM)
```



But always observe the residual plots!

```
PlotResiduals(PaidAM)
```



Your turn

```
head(df)
```

```
##      Y X1       e  X2 spurious
## 1 66.5  1 -1.2071 -20   1.6075
## 2 65.0  2  0.2774 -19   0.2836
## 3 63.5  3  1.0844 -18  -1.8251
## 4 62.0  4 -2.3457 -17   2.6786
## 5 60.5  5  0.4291 -16   0.2631
## 6 59.0  6  0.5061 -15   2.3230
```

Fit a linear model with and without X1 and X2. Which model fits better? How would you determine whether to include the spurious parameter?

Linear regression assumptions

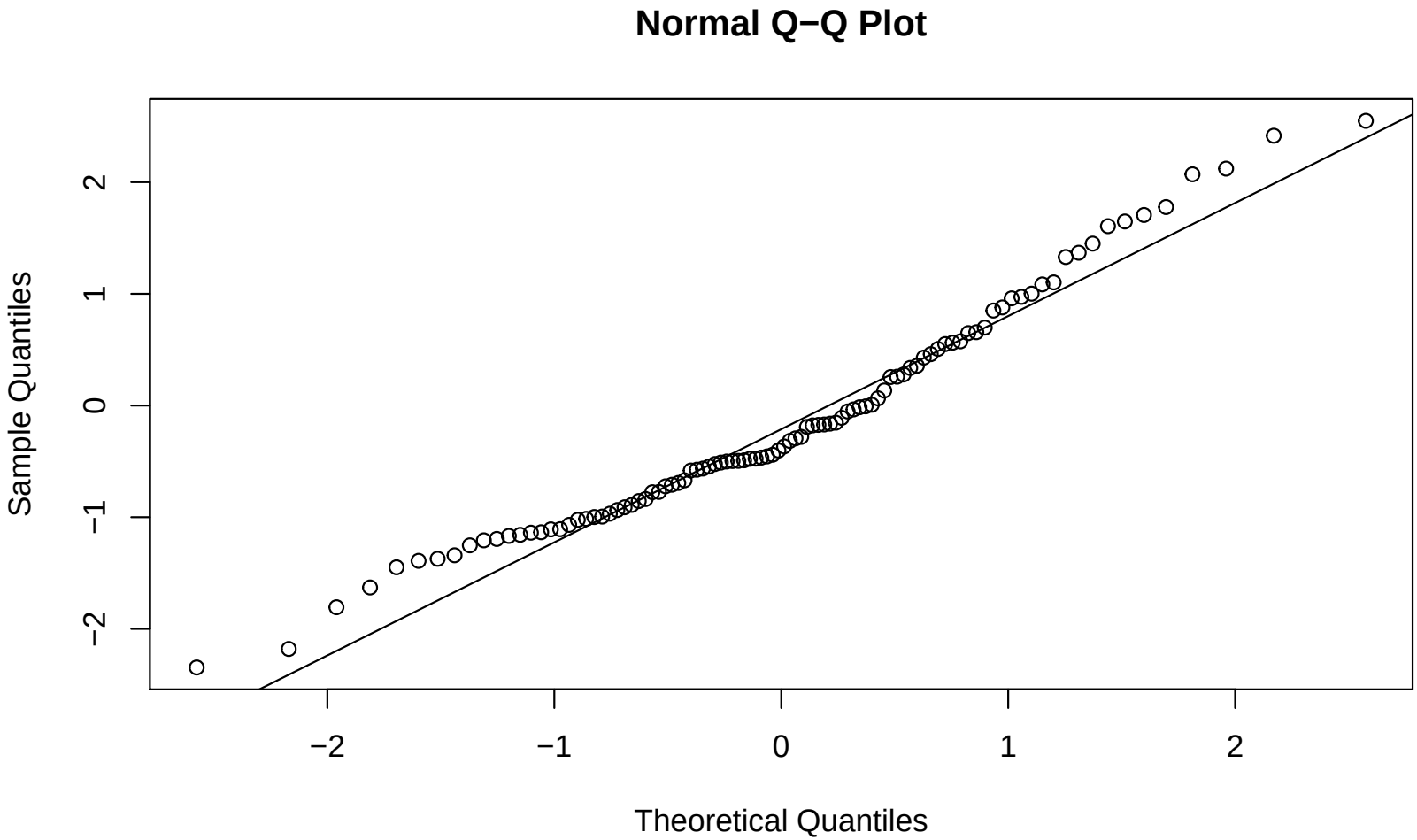
1. Linear model with specified parameters
   - Significance of individual model factors
   - Significance of model
2. Functional form of errors
3. Independence of errors
   - (Serial) correlation of errors
   - Homoskedasticity

Test for normalcy

```
shapiro.test(e)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  e
## W = 0.9659, p-value = 0.01078
```

```
qqnorm(e)
qqline(e)
```

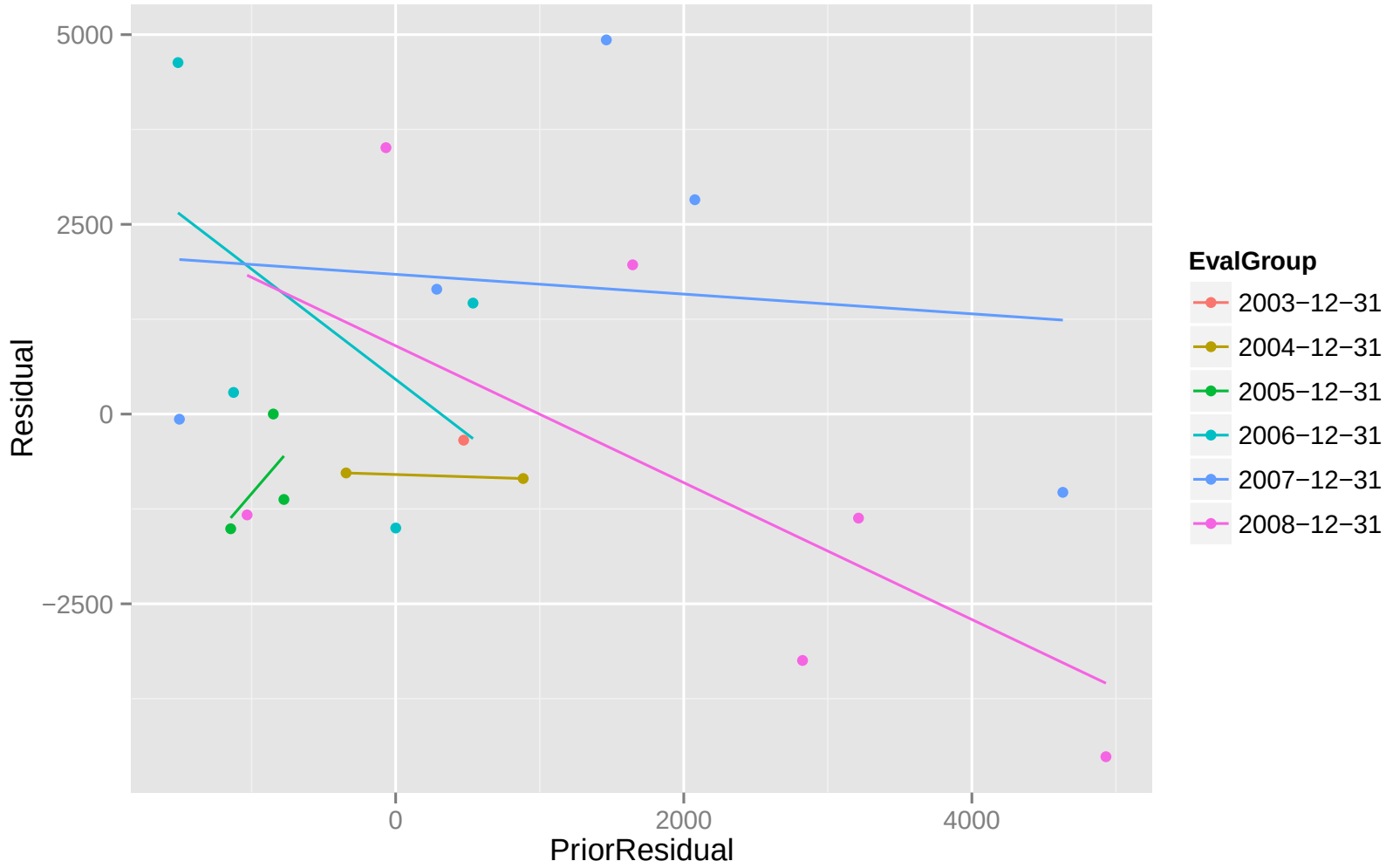

Normal Q−Q Plot
Sample Quantiles
Theoretical Quantiles
−2
−1
0
1
2

Linear regression assumptions

1. Linear model with specified parameters
   - Significance of individual model factors
   - Significance of model
2. Functional form of errors
3. Independence of errors
   - (Serial) correlation of errors
   - Homoskedasticity

Error correlation

```
lstFitResults = plotSerialCorrelation(PaidAM)
```



Error summary

```
summary(lstFitResults$fit)$coefficients
```

```
##                                     Estimate
## PriorResidual:EvalGroup2003-12-31  -0.7290
## PriorResidual:EvalGroup2004-12-31  -0.5376
## PriorResidual:EvalGroup2005-12-31   0.9879
## PriorResidual:EvalGroup2006-12-31  -1.7029
## PriorResidual:EvalGroup2007-12-31   0.2934
## PriorResidual:EvalGroup2008-12-31  -0.6783
##                                    Std. Error
## PriorResidual:EvalGroup2003-12-31     4.8838
## PriorResidual:EvalGroup2004-12-31     2.4270
## PriorResidual:EvalGroup2005-12-31     1.4203
## PriorResidual:EvalGroup2006-12-31     1.1767
## PriorResidual:EvalGroup2007-12-31     0.4192
## PriorResidual:EvalGroup2008-12-31     0.3385
##                                    t value
## PriorResidual:EvalGroup2003-12-31 -0.1493
## PriorResidual:EvalGroup2004-12-31 -0.2215
## PriorResidual:EvalGroup2005-12-31  0.6956
## PriorResidual:EvalGroup2006-12-31 -1.4471
## PriorResidual:EvalGroup2007-12-31  0.6998
## PriorResidual:EvalGroup2008-12-31 -2.0041
##                                    Pr(>|t|)
## PriorResidual:EvalGroup2003-12-31  0.88333
## PriorResidual:EvalGroup2004-12-31  0.82768
## PriorResidual:EvalGroup2005-12-31  0.49733
## PriorResidual:EvalGroup2006-12-31  0.16845
## PriorResidual:EvalGroup2007-12-31  0.49475
## PriorResidual:EvalGroup2008-12-31  0.06346
```

Introduce correlation

```
df = Friedland@TriangleData
cy = (year(df$EvaluationDate) == 2008)
df$IncrementalPaid[cy] = df$IncrementalPaid[cy] *
    (2/3)

myTriangle = newTriangle(df, OriginPeriods = OriginPeriod,
    DevelopmentLags = DevelopmentLag, StaticMeasures = "EP"
    StochasticMeasures = "IncrementalPaid",
    Cumulative = FALSE)

## estimate only:  convert to intervals for accuracy


myModel = newTriangleModel(myTriangle, "IncrementalPaid",
    "EP", "DevInteger")
```

Error summary

```
lstResult = FitSerialCorrelation(myModel)
summary(lstResult$fit)$coefficients
```

```
##                                    Estimate
## PriorResidual:EvalGroup2003-12-31  -0.5188
## PriorResidual:EvalGroup2004-12-31  -0.3895
## PriorResidual:EvalGroup2005-12-31   0.8558
## PriorResidual:EvalGroup2006-12-31  -0.4754
## PriorResidual:EvalGroup2007-12-31   0.6366
## PriorResidual:EvalGroup2008-12-31  -0.8712
##                                    Std. Error
## PriorResidual:EvalGroup2003-12-31     5.0708
## PriorResidual:EvalGroup2004-12-31     2.4087
## PriorResidual:EvalGroup2005-12-31     2.3931
## PriorResidual:EvalGroup2006-12-31     1.3052
## PriorResidual:EvalGroup2007-12-31     0.3936
## PriorResidual:EvalGroup2008-12-31     0.3044
##                                    t value
## PriorResidual:EvalGroup2003-12-31 -0.1023
## PriorResidual:EvalGroup2004-12-31 -0.1617
## PriorResidual:EvalGroup2005-12-31  0.3576
## PriorResidual:EvalGroup2006-12-31 -0.3643
## PriorResidual:EvalGroup2007-12-31  1.6173
## PriorResidual:EvalGroup2008-12-31 -2.8618
##                                    Pr(>|t|)
## PriorResidual:EvalGroup2003-12-31  0.91987
## PriorResidual:EvalGroup2004-12-31  0.87369
## PriorResidual:EvalGroup2005-12-31  0.72562
## PriorResidual:EvalGroup2006-12-31  0.72074
## PriorResidual:EvalGroup2007-12-31  0.12664
## PriorResidual:EvalGroup2008-12-31  0.01188
```

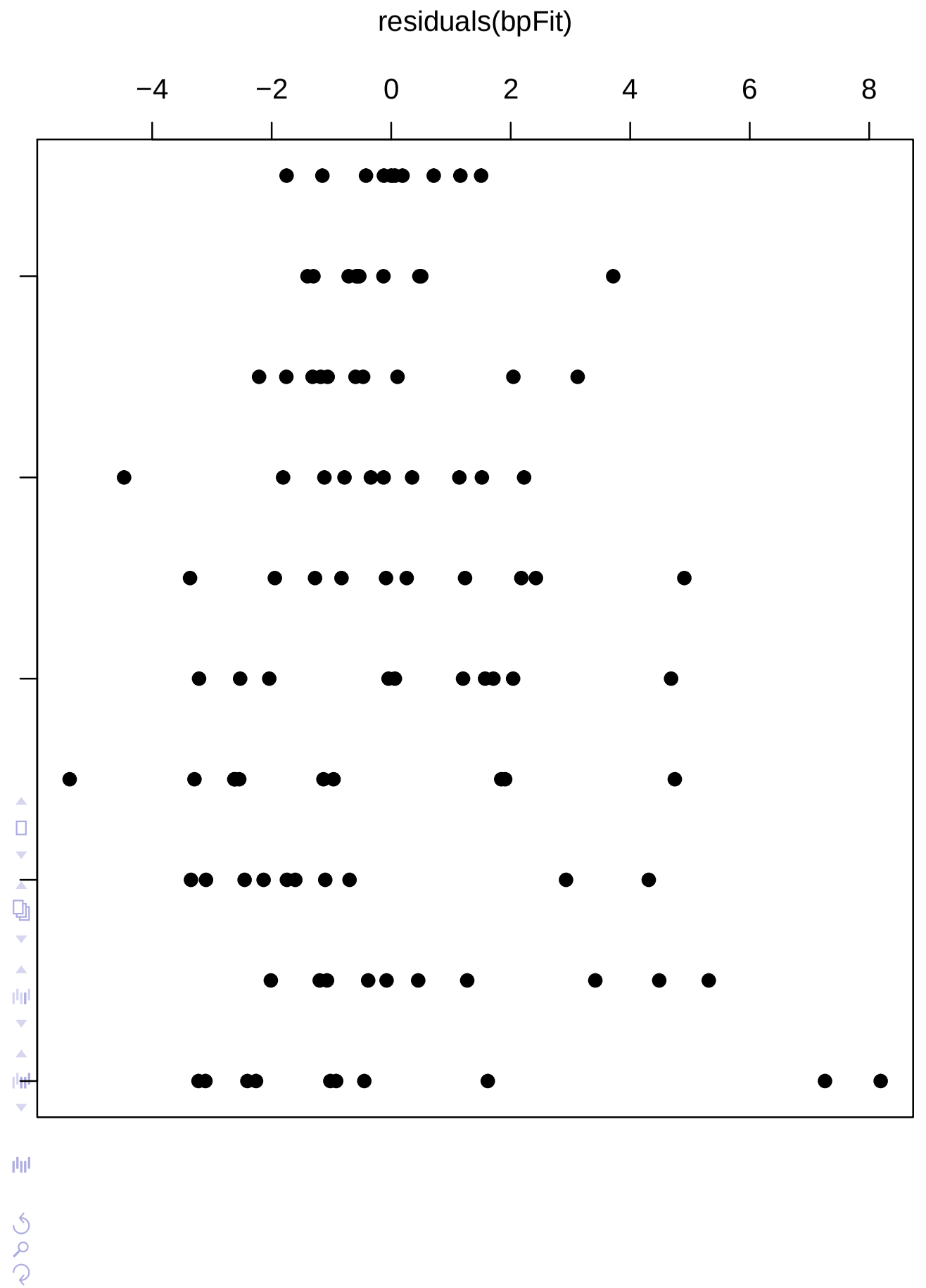
Linear regression assumptions

1. Linear model with specified parameters
   - Significance of individual model factors
   - Significance of model
2. Functional form of errors
3. Independence of errors
   - (Serial) correlation of errors
   - Homoskedasticity

Breusch-Pagan

Heteroskedasticity is most often detected by observing the residuals. Adjusting the weights of the regression is an implicit assumption about the variance of the response variable. Changing the alpha parameter allows one to adjust for presumed heteroskedasticity. Read papers by Dan Murphy (or just ask him, he's probably standing somewhere close by) to learn more.
There is a formal test from Breusch and Pagan, which is available from the lmtest package.

```r
set.seed(1234)
N = 100
e = rnorm(N, mean = 0, sd = 1)
B1 = 1.5
X1 = rep(seq(1, 10), 10)
Y = B1 * X1 + sqrt(X1) * e

bpFit = lm(Y ~ 0 + X1)
bptest(bpFit)

##
##  studentized Breusch-Pagan test
##
## data:  bpFit
## BP = 2.64, df = 0, p-value <
## 2.2e-16

coef(bpFit)

##    X1
```


residuals(bpFit)
−4
−2
0
2
4
6
8

```r
alpha = seq(-10, 10, by = 0.05)
slope = sapply(alpha, function(x) {
    w = X1^x
    fit = lm(Y ~ 0 + X1, weight = w)
    coef(fit)
})
max(Y/X1)
```

```
## [1] 3.302
```

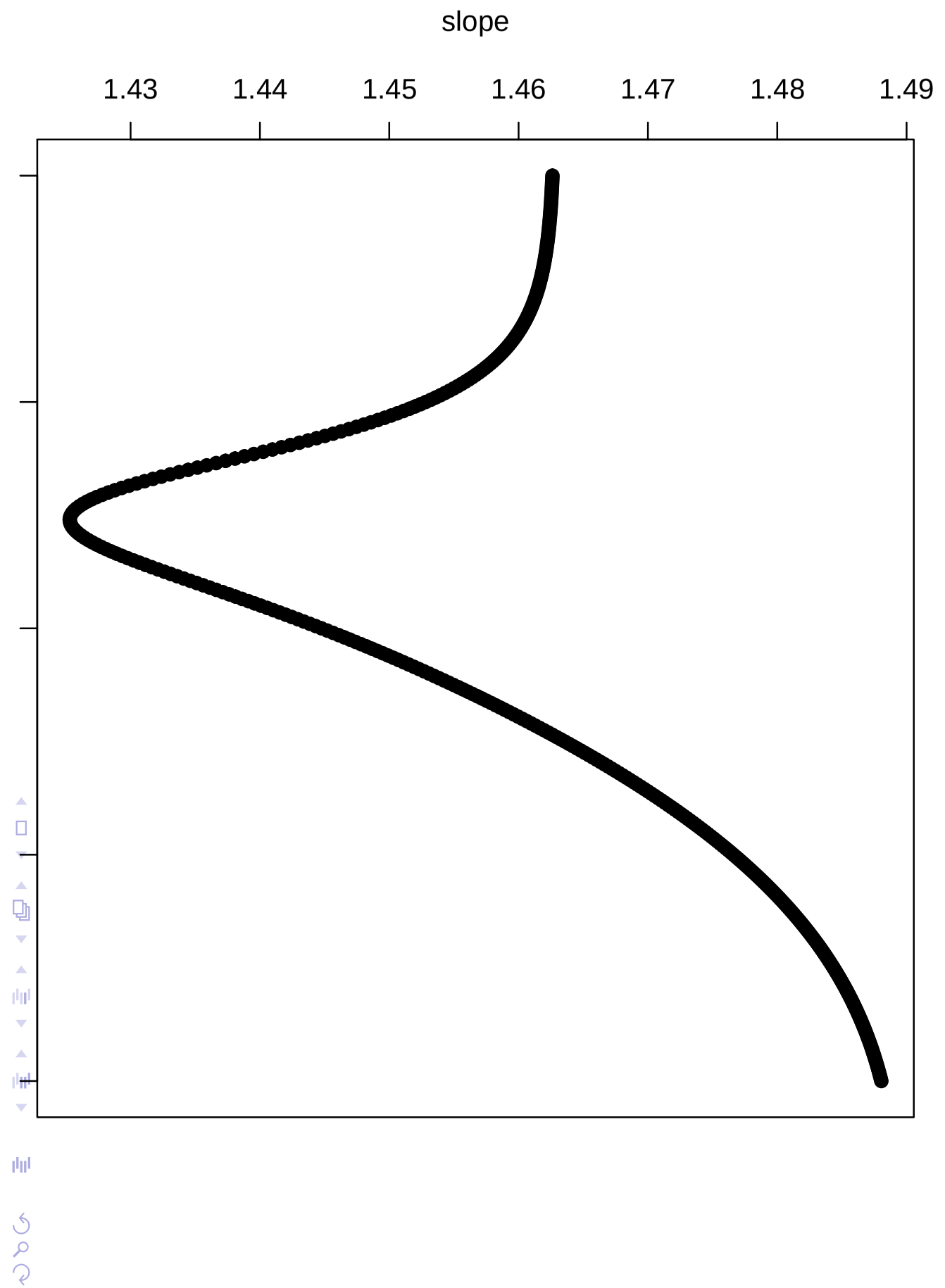
```r
min(Y/X1)
```

```
## [1] -0.306
```

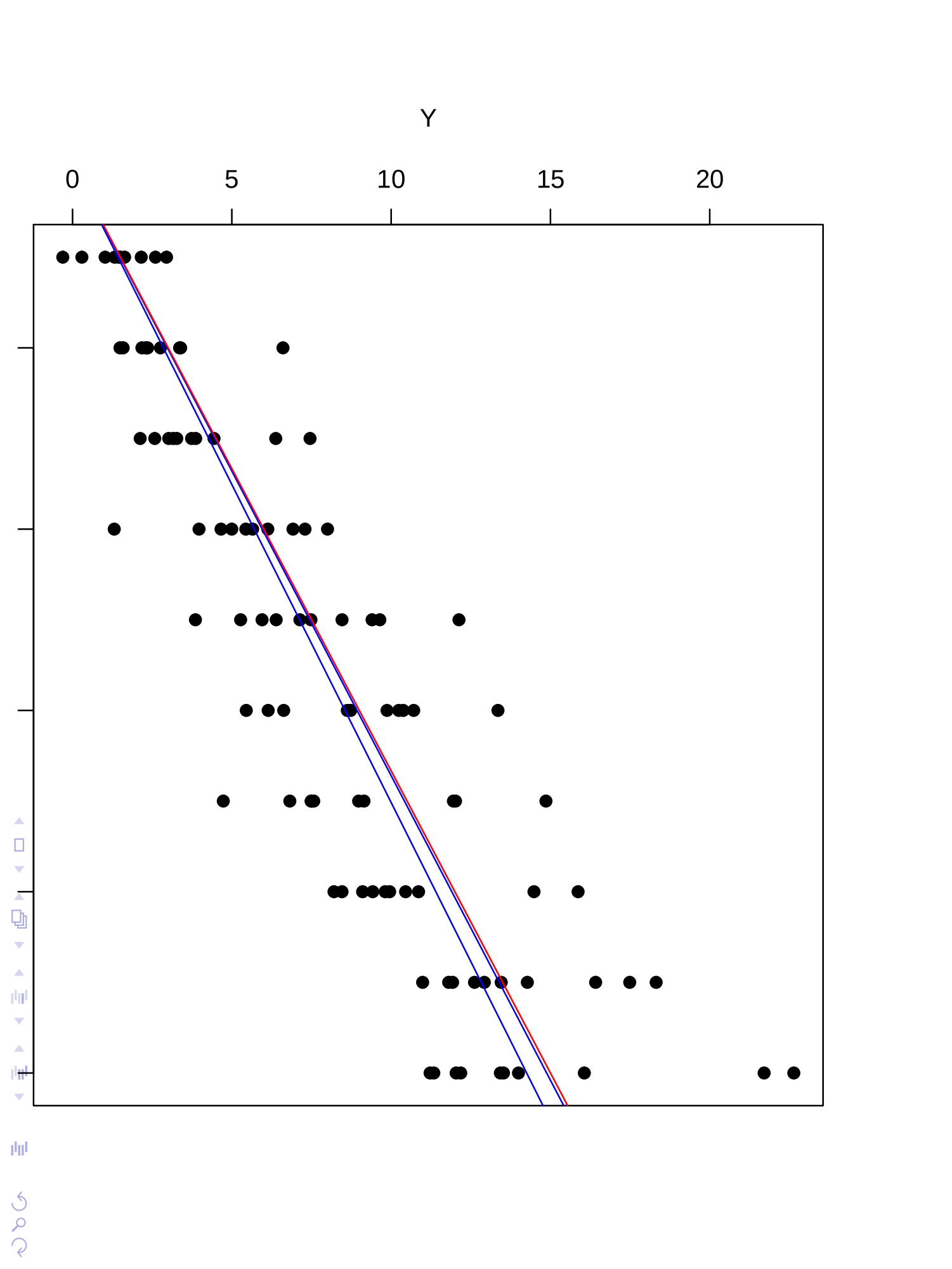
```r
max(slope)
```

```
## [1] 1.488
```

```r
min(slope)
```


slope
1.43
1.44
1.45
1.46
1.47
1.48
1.49

```r
plot(X1, Y, pch = 19)
abline(0, min(slope), col = "blue")
abline(0, max(slope), col = "blue")
abline(0, B1, col = "red")
```


Y
0
5
10
15
20

Heteroskedasticity is controlled through the alpha parameter

```
PaidAM0 = newTriangleModel(Friedland, Response = "Increment
    Predictor = "EP", FitCategory = "DevInteger",
    Tail = 6, Alpha = 1)
PaidAM0 = newTriangleModel(Friedland, Response = "Increment
    Predictor = "EP", FitCategory = "DevInteger",
    Tail = 6, Alpha = 2)
```

Diagnostics pitfalls

Following is an example, created by Francis Anscombe, of the difficulty in interpreting diagnostics

Anscombe pt.2

```
fit1 = lm(y1 ~ x1)
fit2 = lm(y2 ~ x2)
fit3 = lm(y3 ~ x3)
fit4 = lm(y4 ~ x4)
```

```r
summary(fit1)$r.squared
```

```
## [1] 0.6665
```

```r
summary(fit2)$r.squared
```

```
## [1] 0.6662
```

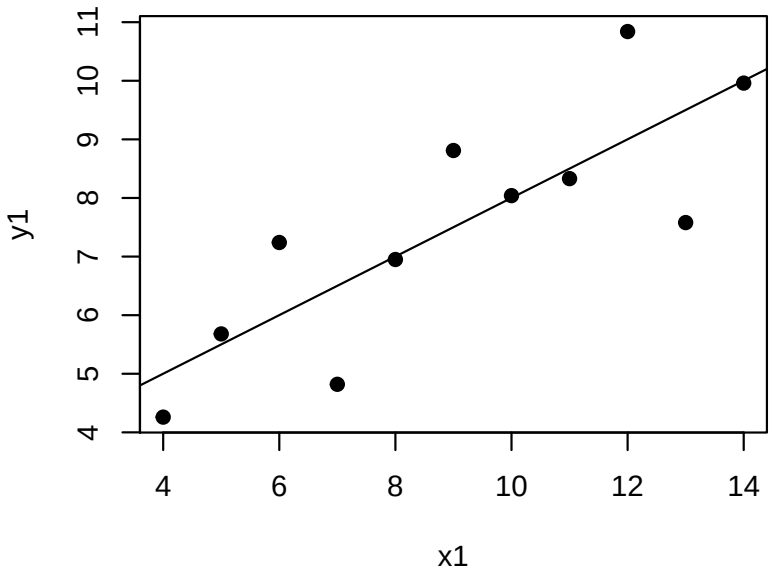
```r
summary(fit3)$r.squared
```

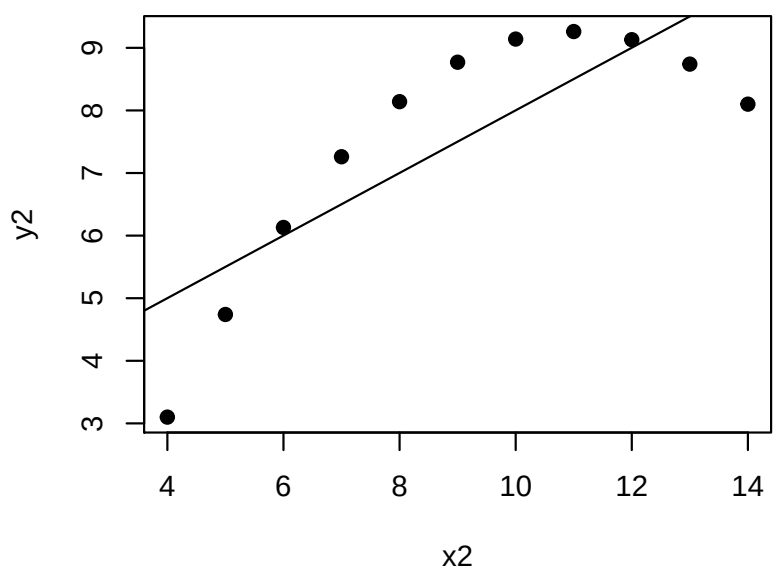
```
## [1] 0.6663
```

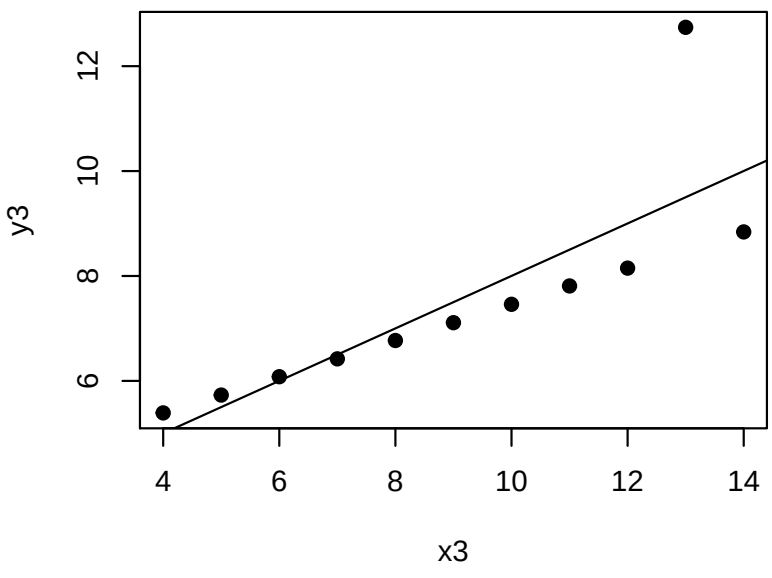
```r
summary(fit4)$r.squared
```

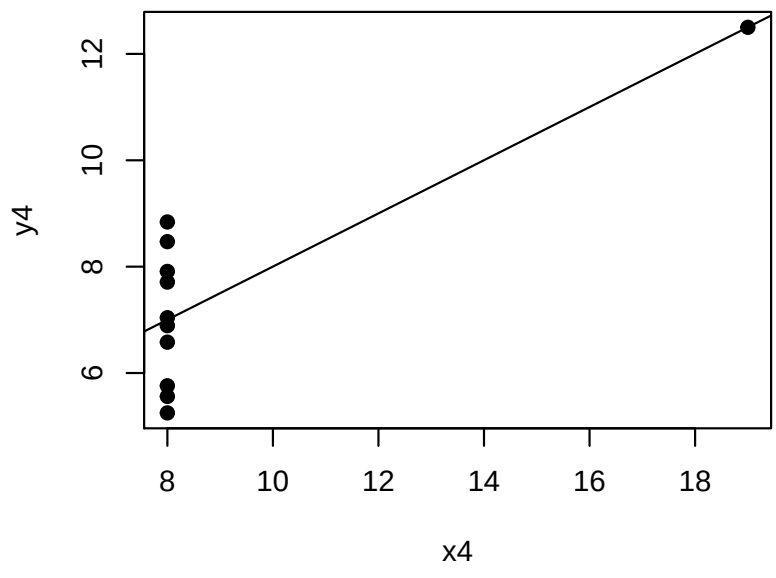
```
## [1] 0.6667
```

```
op = par(mfrow = c(2, 2))
plot(y1 ~ x1, pch = 19)
abline(fit1$coefficients[[1]], fit1$coefficients[[2]])

plot(y2 ~ x2, pch = 19)
abline(fit1$coefficients[[1]], fit1$coefficients[[2]])

plot(y3 ~ x3, pch = 19)
abline(fit1$coefficients[[1]], fit1$coefficients[[2]])

plot(y4 ~ x4, pch = 19)
abline(fit1$coefficients[[1]], fit1$coefficients[[2]])

par(op)
```


y1
x1
4
5
6
7
8
9
10
11
4
6
8
10
12
14

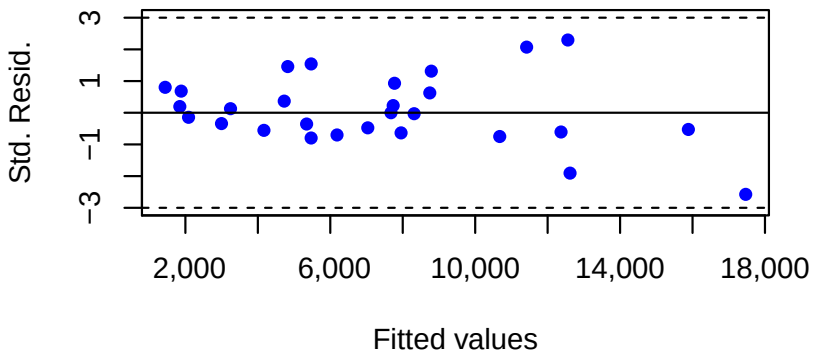

y2
x2
3
4
5
6
7
8
9
4
6
8
10
12
14

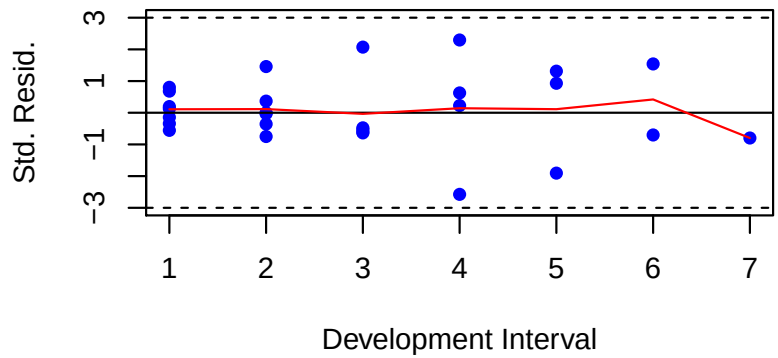

y3
x3
6
8
10
12
4
6
8
10
12
14

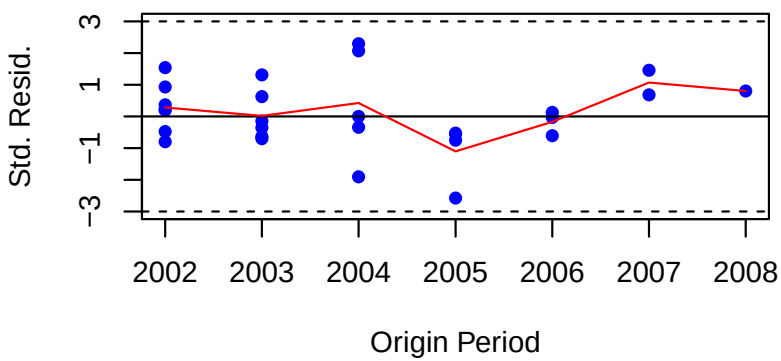

y4
x4
6
8
10
12
8
10
12
14
16
18

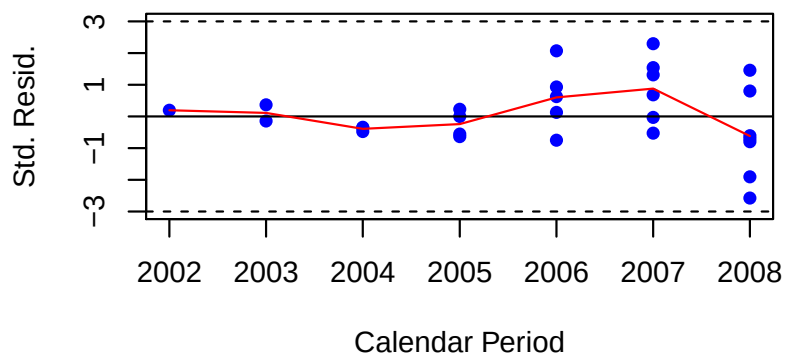
Observe the residual plots

```
PlotResiduals(PaidAM)
```



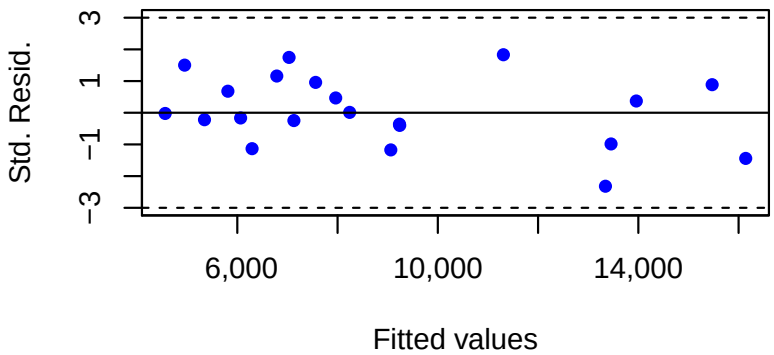


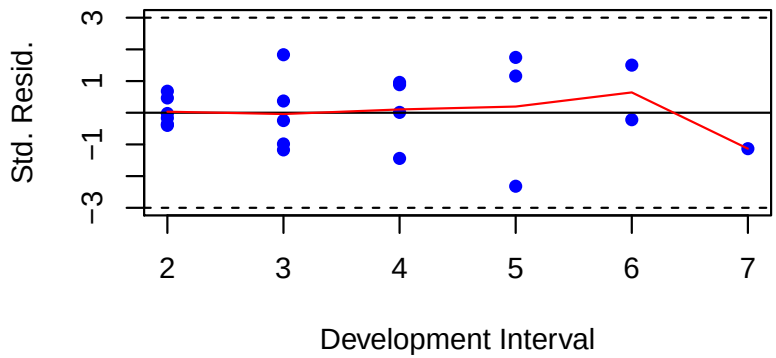


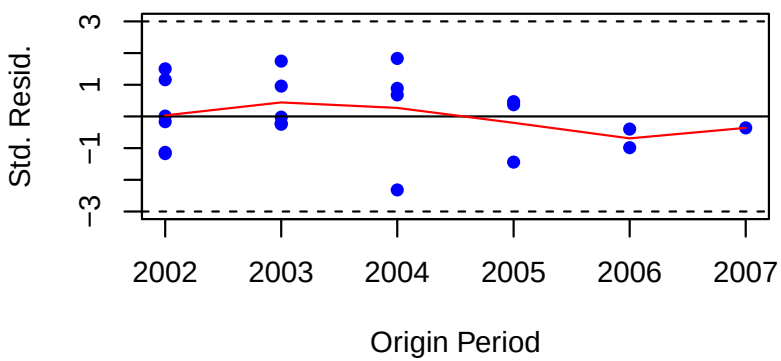


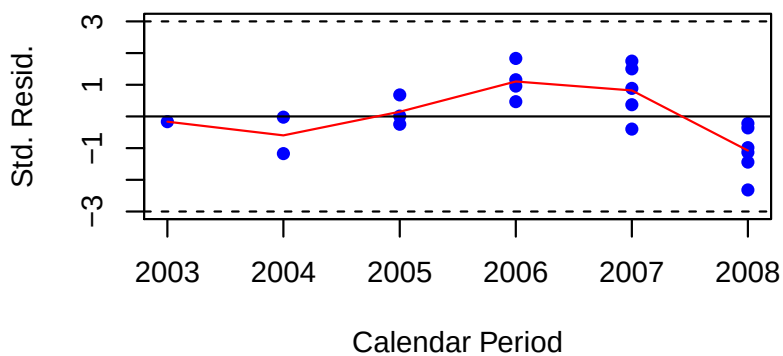
Observe the residual plots

```
PlotResiduals(PaidCL)
```









Projection to as-of date

Once a model has been checked and selected, projection of losses is trivial. MRMR will either project to a specific date or a specific development lag.

```
PaidAM_Projection = TriangleProjection(PaidAM,
    ProjectToDev = FALSE, AsOfDate = mdy("12/31/2010"))

## estimate only:  convert to intervals for accuracy

df = PaidAM_Projection@ProjectionData
```

Projection to development age

```
PaidAM_Projection = TriangleProjection(PaidAM,
    ProjectToDev = TRUE, MaxDev = 10)

## estimate only:  convert to intervals for accuracy

df = PaidAM_Projection@ProjectionData
```

Another view of linear regression

The ordinarly least squares (OLS) regression procedure began by minimizing the sum of squared errors. Assuming homoskedastic normal error terms, this produces the same model factors as maximizing the likelihood function.

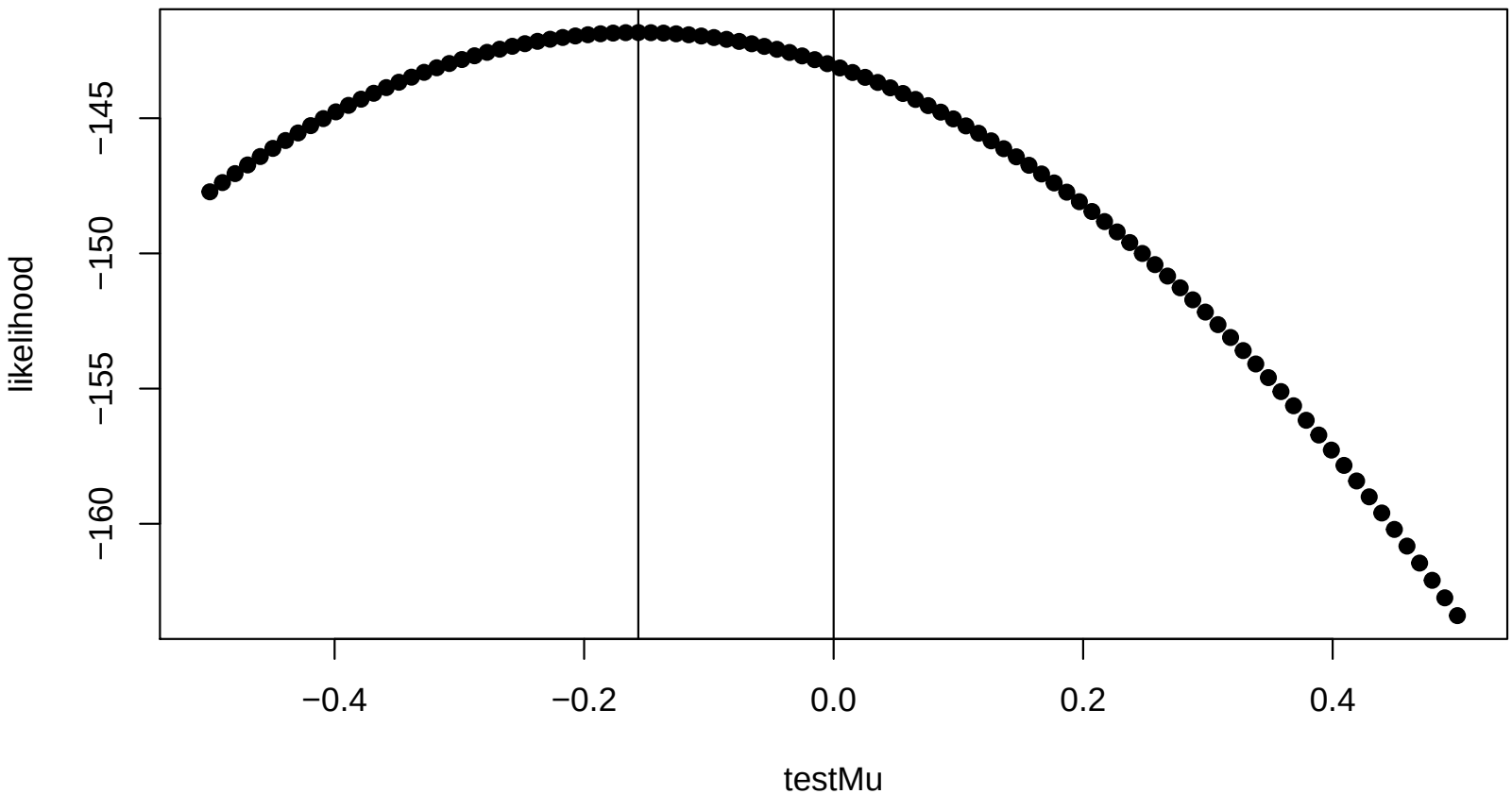
This idea allows us to extend the model so that functional forms other than homoskedastic normal may be contemplated. Generalized linear models are one such example.

Another view of linear regression

```
set.seed(1234)
N = 100
e = rnorm(N, mean = 0, sd = 1)

lnLike = function(x, mu, sigma) {
    n = length(x)
    lnLike = -n/2 * log(2 * pi)
    lnLike = lnLike - n/2 * log(sigma^2)
    lnLike = lnLike - 1/(2 * sigma^2) * sum((x -
        mu)^2)
    lnLike
}
```

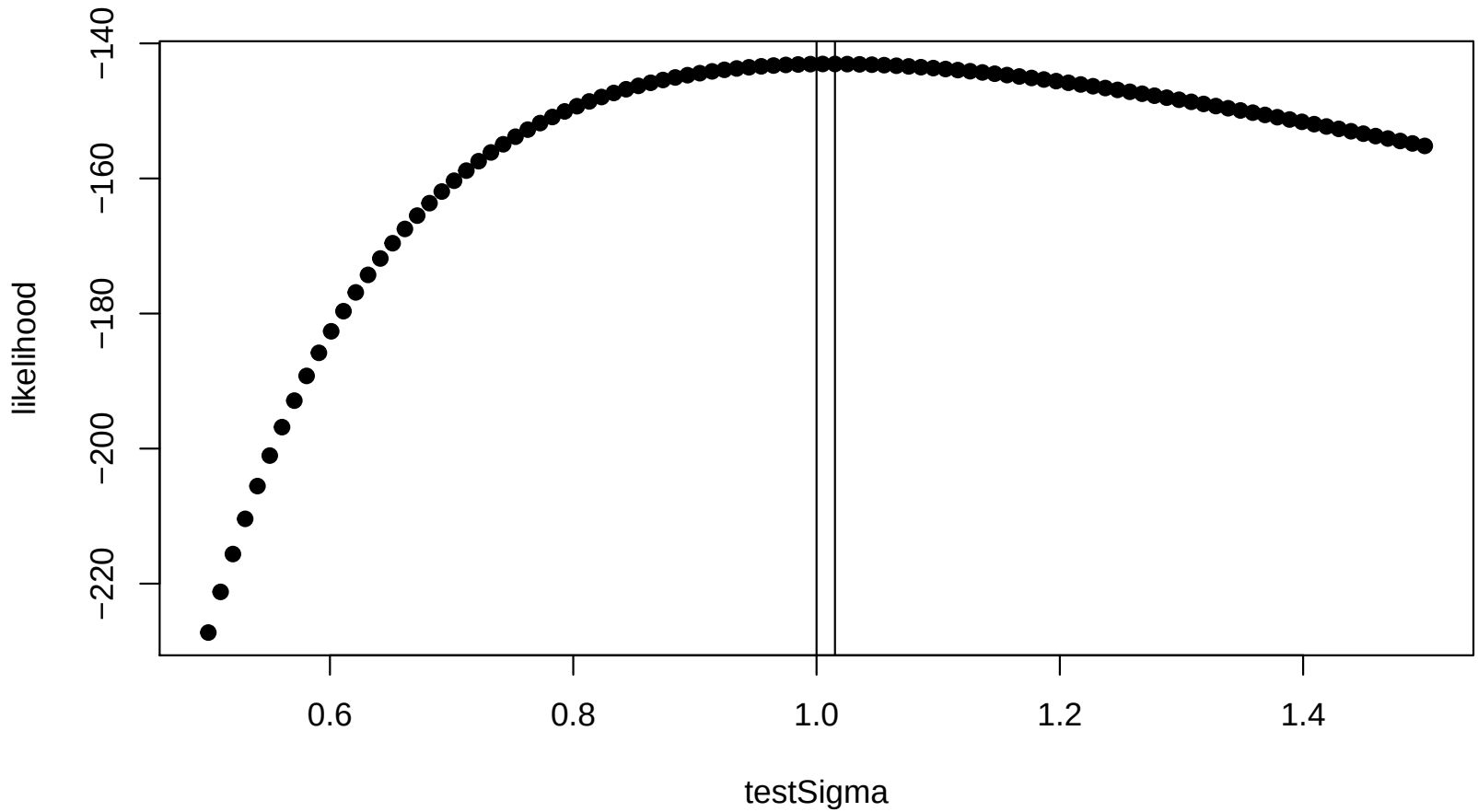
```
testMu = seq(-0.5, 0.5, length.out = 100)
likelihood = sapply(testMu, lnLike, x = e,
    sigma = 1)
testMu[likelihood == max(likelihood)]

## [1] -0.1566
```

```
plot(likelihood ~ testMu, pch = 19)
abline(v = 0)
abline(v = testMu[likelihood == max(likelihood)])
```



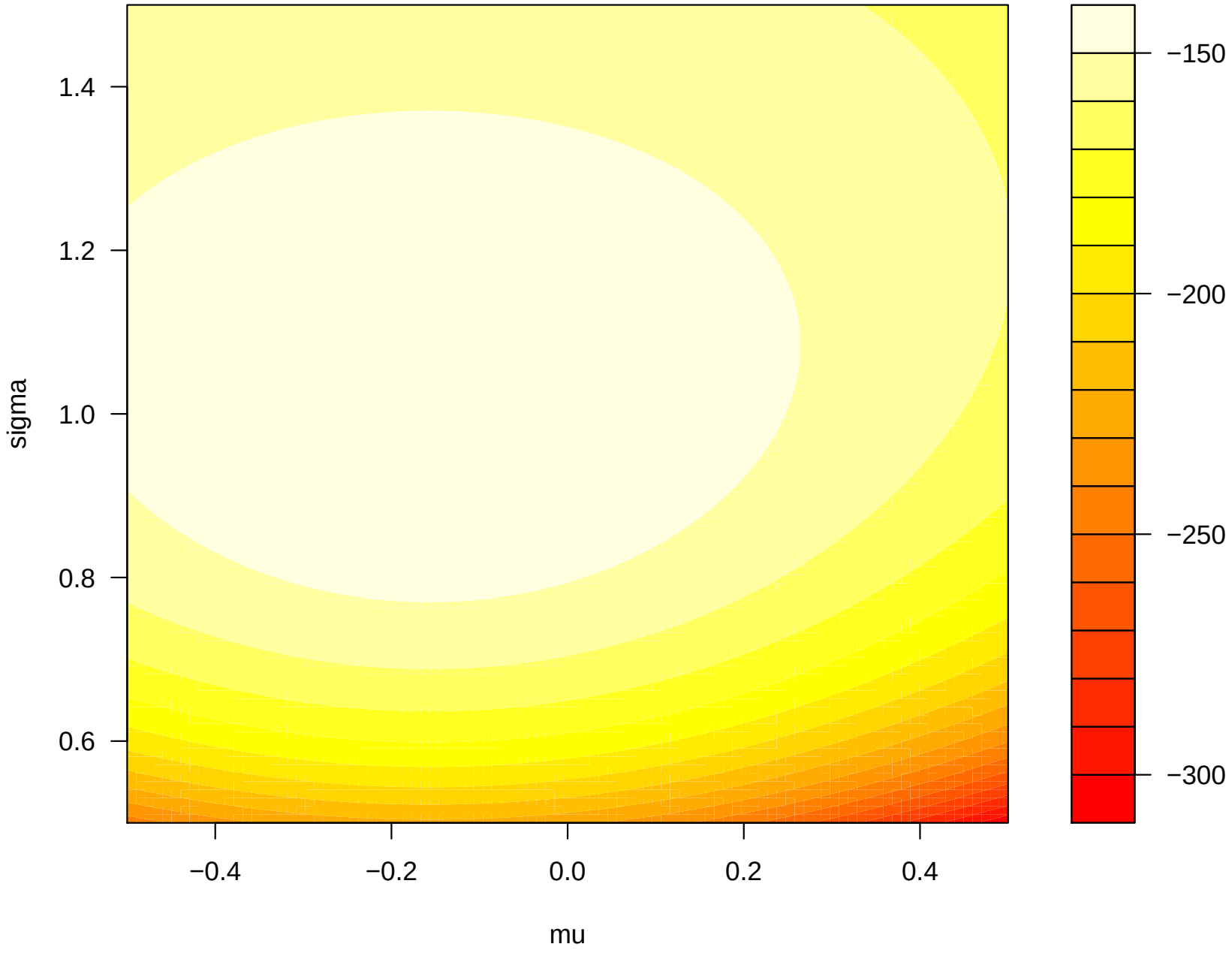
```
testSigma = seq(0.5, 1.5, length.out = 100)
likelihood = sapply(testSigma, lnLike, x = e,
    mu = 0)
testSigma[likelihood == max(likelihood)]

## [1] 1.015
```

```
plot(likelihood ~ testSigma, pch = 19)
abline(v = 1)
abline(v = testSigma[likelihood == max(likelihood)])
```



```
params = expand.grid(mu = testMu, sigma = testSigma)
params$Likelihood = mapply(lnLike, params$mu,
    params$sigma, MoreArgs = list(x = e))
z = matrix(params$Likelihood, length(testMu),
    length(testSigma))
```

```
filled.contour(x = testMu, y = testSigma,
    z = z, color.palette = heat.colors, xlab = "mu",
    ylab = "sigma")
```


sigma
1.4
1.2
1.0
0.8
0.6
−0.4
−0.2
0.0
0.2
0.4
mu
−150
−200
−250
−300

Optimize for both parameters

```
lnLike2 = function(x, par) {
    mu = par[1]
    sigma = par[2]

    lnLike(x, mu, sigma)
}

optimFit = optim(par = c(-1, 4), fn = lnLike2,
    control = list(fnscale = -1), x = e)
optimFit$par

## [1] -0.1566  0.9994
```

Add a constant term to the normal variable e

```
B0 = 5
Y = B0 + e
```

This is equivalent to lm

```
optimFit = optim(par = c(-1, 4), fn = lnLike2,
    control = list(fnscale = -1), x = Y)
optimFit$par[[1]]

## [1] 4.843


lmFit = lm(Y ~ 1)
lmFit$coefficients[[1]]

## [1] 4.843
```

Now add a slope

```
X = as.double(1:length(e))
B1 = 1.5
Y = B0 + B1 * X + e

lnLike3 = function(par, Y, X) {
    B0 = par[1]
    B1 = par[2]
    sigma = par[3]

    x = Y - B0 - B1 * X
    mu = 0

    lnLike(x, mu, sigma)
}
```

```
optimFit = optim(par = c(4, 1, 1), fn = lnLike3,
    control = list(fnscale = -1), Y = Y,
    X = X)
optimFit$par[1:2]

## [1] 4.404 1.509


lmFit = lm(Y ~ 1 + X)
lmFit$coefficients

## (Intercept)           X
##       4.406       1.509
```

Further

- Coursera
- Meetup.com
- R-bloggers.com
  - PirateGrunt.com
  - MagesBlog.com
- Github
- Books!
  - Machine Learning for Hackers by Conway & White
  - Software for Data Analysis by Chambers
  - Data Analysis Using Regression and Multilevel/Hierarchical Models by Gelman & Hille
  - Introduction to the Theory and Practice of Econometrics by Judge, Griffiths & Hill
- Other languages
  - Python
  - D3