CSSS 510: Lab 2

Introduction to Maximum Likelihood Estimation

2017-9-29

0. Agenda

1. Simulating heteroskedastic normal data
2. Fitting a model using the simulated data
3. Calculating predicted values
4. Fitting the heteroskedastic normal model using ML
5. Simulating predicted values and confidence intervals

1. Simulating heteroskedastic normal data

- Set the number of observations to 1500
- Set a parameter vector for the mean (assume 2 covariates plus the constant)
- Set a parameter vector for the variance (assume heteroskedasticity)
- Generate the constant and the covariates, length 1500 for each (draw from a uniform distribution)
- Create the systematic component for the mean
- Create the systematic component for the variance (the same covariates affect mu and sigma)
- Generate the response variable
- Save the data to a data frame
- Plot the data

Stochastic component:

$$y \sim N(\mu_i, \sigma_i^2)$$

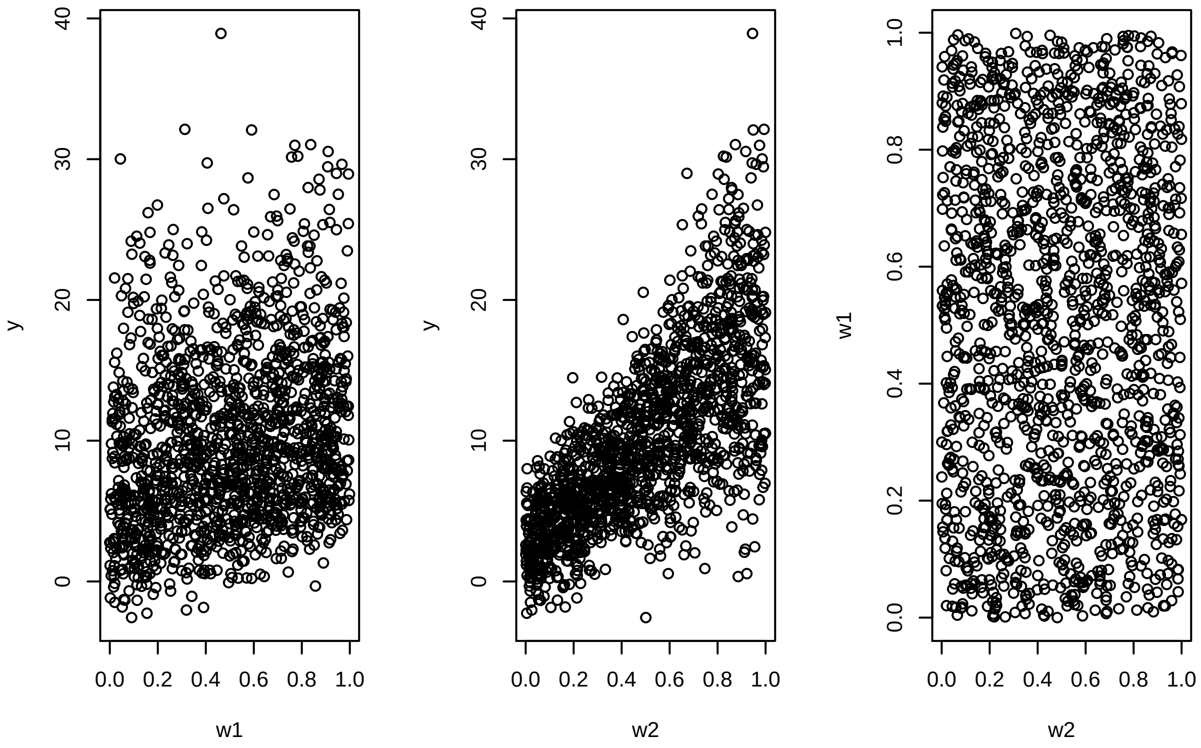
Systematic components:

$$\mu = \boldsymbol{x}_i \boldsymbol{\beta}$$

$$\sigma_i^2 = \exp(\boldsymbol{z}_i \boldsymbol{\gamma})$$

```r
rm(list=ls()) # Clear memory

set.seed(123456) # To reproduce random numbers

library(MASS) # Load packages
library(simcf)

n <- 1500 # Generate 1500 observations
```

```r
beta <- c(0, 5, 15) # Set a parameter vector for the mean
# One for constant, one for covariate 1, one for covariate 2.

gamma <- c(1, 0, 3) # Set a parameter vector for the variance
# Note that gamma estimate for covariate 2 is set to be 3, creating heteroskedasticity

w0 <- rep(1, n) # Create the constant and covariates, length of each vector is 1500
w1 <- runif(n)
w2 <- runif(n)

x <- cbind(w0, w1, w2) # Create a matrix of the covariates

mu <- x%*%beta # Create the systemtic component for the mean

z <- x # i.e., same covariates affect mu and sigma
sigma2 <- exp(x%*%gamma) # Create the systematic component for the variance

# z is 1500 by 3 matrix, gamma is 3 by 1 matrix
#ith row of sigma 2 thus equals exp(1+0+w2_i*3). i.e., it is a function of w2

y <- mu + rnorm(n)*sqrt(sigma2) # Create the response variable

data <- cbind(y,w1,w2) # Save the data to a data frame
data <- as.data.frame(data)
names(data) <- c("y","w1","w2")

par(mfrow=c(1,3)) #Plot the data

#pdf("YvsW1.pdf")
plot(y=y,x=w1)
#dev.off()

#pdf("YvsW2.pdf")
plot(y=y,x=w2)
#dev.off()

#pdf("W1vsW2.pdf")
plot(y=w1,x=w2)
```



```
#dev.off()
```

2. Fitting a model using the simulated data

- Assume we don't know the true value of the parameters and fit a model using least squares (use the lm() function and regress the response variable on the two covariates)
- Calculate and print the AIC

```
############

#Now let's try fitting a linear model using the simulated data

ls.result <- lm(y ~ w1 + w2)
print(summary(ls.result))
```

```
##
## Call:
## lm(formula = y ~ w1 + w2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.5710  -2.3157  -0.0219   2.2124  21.9345
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.04867    0.28221  -0.172    0.863
## w1           4.78421    0.37699  12.690   <2e-16 ***
## w2          15.68283    0.37112  42.258   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.17 on 1497 degrees of freedom
## Multiple R-squared:  0.5678, Adjusted R-squared:  0.5672
## F-statistic: 983.3 on 2 and 1497 DF,  p-value: < 2.2e-16
```

```
# Calculate and print the AIC
ls.aic <- AIC(ls.result)
print(ls.aic)
```

```
## [1] 8545.903
```

3. Calculating predicted values

Scenario 1: Vary covariate 1

- Create a data frame with a set of hypothetical scenarios for covariate 1 while keeping covariate 2 at its mean
- Calculate the predicted values using the predict() function Plot the predicted values

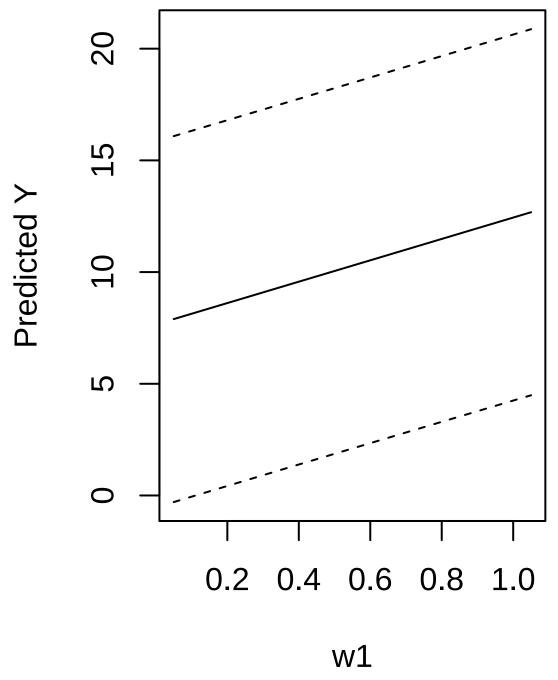
```
############


# Calculate predicted values using predict()

# Start by calculating P(Y|w1) for different w1 values
w1range <- seq(0:20)/20                                            # Set as necessary

# Set up a dataframe with the hypothetical scenarios (varied w1, all else equal)
baseline <- c(mean(w1), mean(w2))                                  # Set as necessary
xhypo <- matrix(baseline, nrow=length(w1range), ncol=2, byrow= TRUE) # Set ncol to # of x's
# same as: xhypo <- matrix(rep(baseline,21), nrow=length(w1range), ncol=2, byrow= TRUE)

xhypo <- as.data.frame(xhypo)
names(xhypo) <- c("w1","w2")
xhypo[,1] <- w1range # Scenarios: Changing values in the first column, keeping second column values at
xhypo
```

```
##      w1        w2
## 1  0.05 0.4912698
## 2  0.10 0.4912698
## 3  0.15 0.4912698
## 4  0.20 0.4912698
## 5  0.25 0.4912698
## 6  0.30 0.4912698
## 7  0.35 0.4912698
## 8  0.40 0.4912698
## 9  0.45 0.4912698
## 10 0.50 0.4912698
## 11 0.55 0.4912698
## 12 0.60 0.4912698
## 13 0.65 0.4912698
## 14 0.70 0.4912698
## 15 0.75 0.4912698
## 16 0.80 0.4912698
## 17 0.85 0.4912698
## 18 0.90 0.4912698
## 19 0.95 0.4912698
## 20 1.00 0.4912698
## 21 1.05 0.4912698
```

```
# Calculate Predicted Y using predict()

simls.w1 <- predict(ls.result, newdata = xhypo, interval = "prediction", level = 0.95)

simls.w1
```

```
##          fit         lwr      upr
## 1   7.895048 -0.29514691 16.08524
## 2   8.134259 -0.05450529 16.32302
## 3   8.373470  0.18596961 16.56097
## 4   8.612680  0.42627769 16.79908
## 5   8.851891  0.66641891 17.03736
## 6   9.091101  0.90639320 17.27581
## 7   9.330312  1.14620051 17.51442
## 8   9.569523  1.38584081 17.75320
## 9   9.808733  1.62531406 17.99215
## 10 10.047944  1.86462027 18.23127
## 11 10.287154  2.10375941 18.47055
## 12 10.526365  2.34273150 18.71000
## 13 10.765575  2.58153654 18.94961
## 14 11.004786  2.82017457 19.18940
## 15 11.243997  3.05864562 19.42935
## 16 11.483207  3.29694973 19.66946
## 17 11.722418  3.53508696 19.90975
## 18 11.961628  3.77305737 20.15020
## 19 12.200839  4.01086104 20.39082
## 20 12.440050  4.24849806 20.63160
## 21 12.679260  4.48596852 20.87255
```

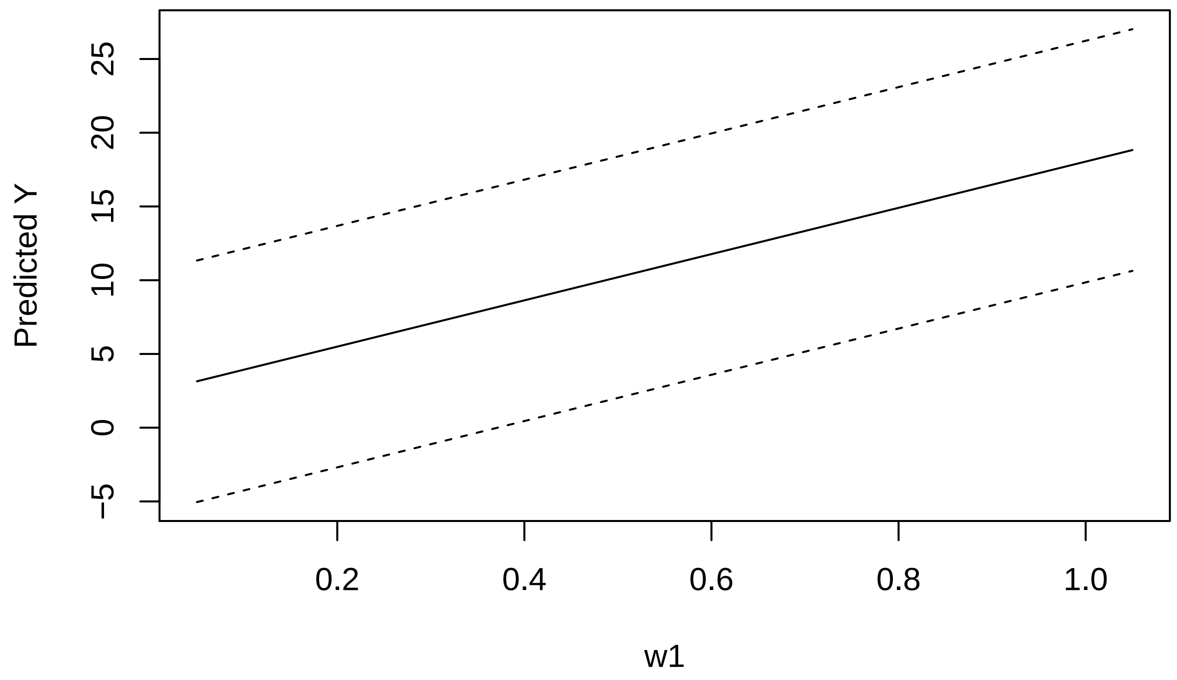
```
# Plot them
yplot <- simls.w1
xplot <- cbind(w1range,w1range,w1range) # need to have the same dimension [21,3]
#pdf("homoYvsW1.pdf")

# matplot is useful for plotting matrices of data

par(mfrow=c(1,2))

matplot(y=yplot,
        x=xplot,
        type="l",
        lty=c("solid","dashed","dashed"),
        col=c("black"),
        xlab = "w1",
        ylab = "Predicted Y")
#dev.off()
```



## Scenario 2: Vary covariate 2

- Create a data frame with a set of hypothetical scenarios for covariate 2 while keeping covariate 1 at its mean
- Calculate the predicted values using the predict() function Plot the predicted values

```
# Calculate predicted values using predict()

# Start by calculating P(Y|w1) for different w1 values
w2range <- seq(0:20)/20                                                    # Set as necessary

# Set up a dataframe with the hypothetical scenarios (varied w1, all else equal)
baseline <- c(mean(w1), mean(w2))                                          # Set as necessary
xhypo <- matrix(baseline, nrow=length(w2range), ncol=2, byrow= TRUE)  # Set ncol to # of x's
xhypo <- as.data.frame(xhypo)
names(xhypo) <- c("w1","w2")                                               # Set by user
xhypo[,2] <- w1range                                                       # Change as necessary

# Calculate Predicted Y using predict()
simls.w2 <- predict(ls.result, newdata = xhypo, interval = "prediction", level = 0.95)

# Plot them
yplot <- simls.w2
xplot <- cbind(w2range,w2range,w2range)
#pdf("homoYvsW2.pdf")
matplot(y=yplot,
        x=xplot,
        type="l",
        lty=c("solid","dashed","dashed"),
        col=c("black"),
        xlab = "w1",
        ylab = "Predicted Y")
```



```
#dev.off()
```

4. Fitting the heteroskedastic normal model using ML

- Create the input matrices (the two covariates)
- Write a likelihood function for the heteroskedastic normal model Find the MLEs using the optim() function
- Extract the point estimates
- Compute the standard errors
- Compare with the least squares estimates
- Find the log likelihood at its maximum
- Compute the AIC
- Simulate the results by drawing from the model's predictive distribution Separate the simulated betas from the simulated gammas

```
#####################################################
# Fit ML heteroskedastic normal model using optim()


# A likelihood function for ML heteroskedastic Normal
llk.hetnormlin <- function(param,y,x,z) {
  x <- as.matrix(x)      #x as a matrix
  z <- as.matrix(z)      #z as a matrix
  os <- rep(1,nrow(x))  #1 for the intercept
  x <- cbind(os,x)       #combine
  z <- cbind(os,z)
  b <- param[ 1 : ncol(x) ] # i.e., the first three spaces in the param vector
  g <- param[ (ncol(x)+1) : (ncol(x) + ncol(z)) ] # i.e., the three remaining spaces
  xb <- x%*%b # systematic components for the mean
  s2 <- exp(z%*%g) # systematic components for the variance

  sum(0.5*(log(s2)+(y-xb)^2/s2)) # "optim" command minimizes a function by default. Minimalization of -

  #-sum(0.5*(log(s2)+(y-xb)^2/s2)) # Alternativly, you can use lnL(param|y) and set optim to be a maxim

  }

# Create input matrices
xcovariates <- cbind(w1,w2)
zcovariates <- cbind(w1,w2)

# initial guesses of beta0, beta1, ..., gamma0, gamma1, ...
# we need one entry per parameter, in order!
stval <- c(0,0,0,0,0,0) # also include beta and gamma estiamtes for constants


help(optim)

# Run ML, get the output we need
hetnorm.result <- optim(stval,llk.hetnormlin,method="BFGS",hessian=T,y=y,x=xcovariates,z=zcovariates) #

pe <- hetnorm.result$par   # point estimates
pe
```

```
## [1] -0.1672393  5.0074031 15.6981262  0.9103126  0.2238205  2.9937686
```

```
vc <- solve(hetnorm.result$hessian)  # 6x6 var-cov matrix (allows to compute standard errors)
round(vc,5)
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
## [1,]  0.02909 -0.03265 -0.02810  0.00059 -0.00087 -0.00032
## [2,] -0.03265  0.06787 -0.00025 -0.00091  0.00099  0.00084
## [3,] -0.02810 -0.00025  0.10331 -0.00056  0.00143 -0.00033
## [4,]  0.00059 -0.00091 -0.00056  0.00920 -0.00832 -0.00749
## [5,] -0.00087  0.00099  0.00143 -0.00832  0.01693 -0.00042
## [6,] -0.00032  0.00084 -0.00033 -0.00749 -0.00042  0.01568
```

```
se <- sqrt(diag(vc))    # standard errors - the ML standard errors are the square roots of the diagonal
se
```

```
## [1] 0.17056695 0.26050990 0.32141276 0.09594005 0.13010517 0.12523484
```

```
mle.result<-round(cbind(pe[1:3], se[1:3]),2) # see pe and se
colnames(mle.result)<-c("Estimate", "Std.Error")
rownames(mle.result)<-c("(Intercept)", "w1","w2" )
mle.result
```

```
##             Estimate Std.Error
## (Intercept)    -0.17      0.17
## w1              5.01      0.26
## w2             15.70      0.32
```

```
round(summary(ls.result)$coefficients[,c(1,2)],2) #compare with the ls result
```

```
##             Estimate Std. Error
## (Intercept)    -0.05       0.28
## w1              4.78       0.38
## w2             15.68       0.37
```

```
ll <- -hetnorm.result$value  # likelihood at maximum, no need to  have a negative sign if you set optim
ll
```

```
## [1] -2620.334
```

```
#The AIC is the deviance or -2*ll at its max plus 2*number of parameters or the dimension
hetnorm.aic <- 2*length(stval) - 2*ll  # first component to penalizing the number of parameters (i.e.,

print(hetnorm.aic)
```

```
## [1] 5252.668
```

```
# remember AIC from LS fit?
print(ls.aic)
```

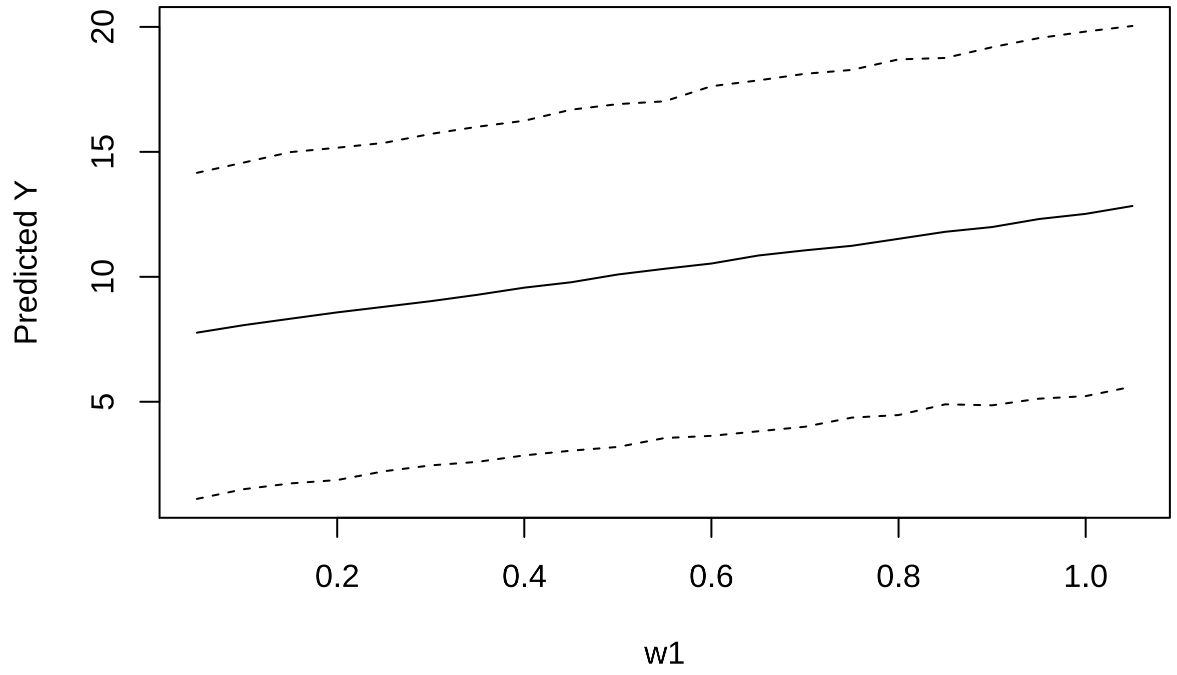
```
## [1] 8545.903
```

5. Simulating predicted values and confidence intervals

Scenario 1: Vary covariate 1

- Create a data frame with a set of hypothetical scenarios for covariate 1 while keeping covariate 2 at its mean
- Simulate the predicted values and the confidence intervals using simcf Plot the results

```
#########################################

# Simulate results by drawing from the model predictive distribution
sims <- 10000
simparam <- mvrnorm(sims,pe,vc) # draw parameters store them in 10000x6 matrix. We assume that paramete

# Separate into the simulated betas and simulated gammas
simbetas <- simparam[,1:(ncol(xcovariates)+1)] # first three columns store simulated beta coefficients
simgammas <- simparam[,(ncol(simbetas)+1):ncol(simparam)] # then simulated gamma coefficients


# Put our models in "formula" form
model <- (y ~ w1 + w2)
varmodel <- (y ~ w1 + w2)


# Scenario 1:  Vary w1

# Start by calculating P(Y|w1) for different w1 values
w1range <- seq(0:20)/20

# Set up a matrix with the hypothetical scenarios (varied w1, all else equal)
xhypo <- cfMake(model, data, nscen = length(w1range)) # creating a set of scenarios


for (i in 1:length(w1range)) {
  xhypo <- cfChange(xhypo, "w1", x=w1range[i], scen=i) # change the values of the variables of your int
}

zhypo <- cfMake(varmodel, data, nscen = length(w1range))

for (i in 1:length(w1range)) {
  zhypo <- cfChange(zhypo, "w1", x=w1range[i], scen=i)
}

# Simulate the predicted Y's and CI's
simres.w1 <- hetnormsimpv(xhypo,simbetas,
                          zhypo,simgammas,
                          ci=0.95,
                          constant=1,varconstant=1)

#simy<-rnorm(sims)*sqrt(simsigma2)+ simmu

# Plot them
yplot <- cbind(simres.w1$pe, simres.w1$lower, simres.w1$upper)
xplot <- cbind(w1range,w1range,w1range)
#pdf("heteroYvsW1.pdf")
matplot(y=yplot,
        x=xplot,
        type="l",
        lty=c("solid","dashed","dashed"),
        col=c("black"),
        xlab = "w1",
        ylab = "Predicted Y")
```

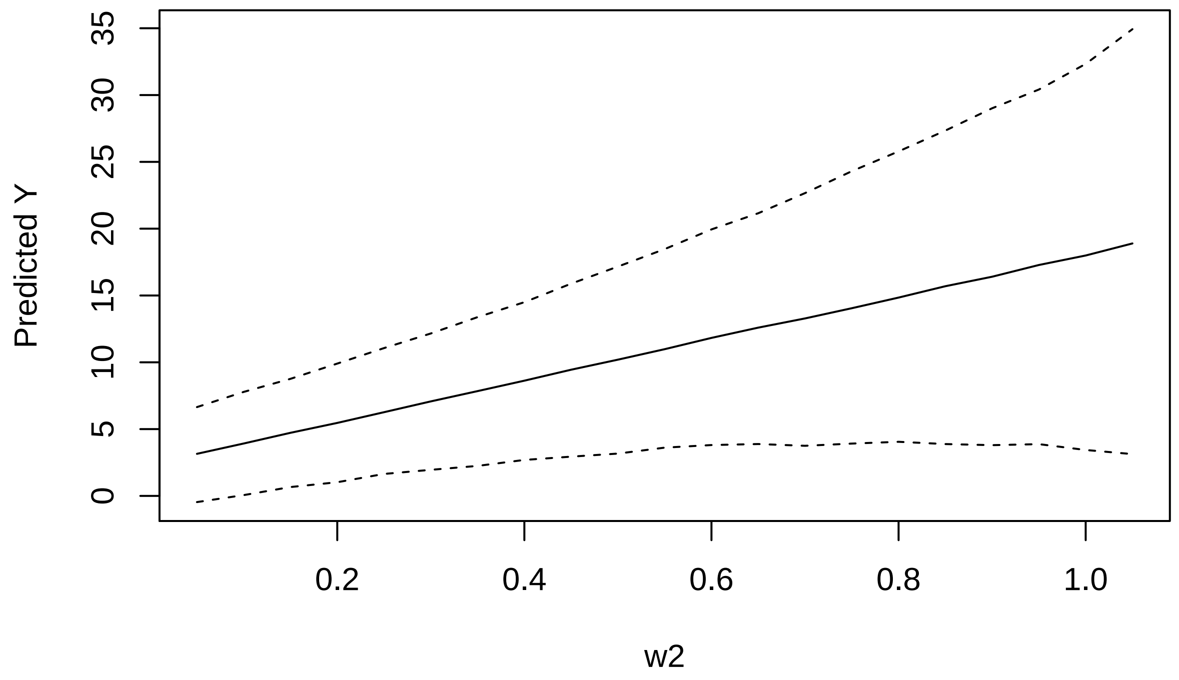


```
#dev.off()
```

Scenario 2: Vary covariate 2

- Create a data frame with a set of hypothetical scenarios for covariate 2 while keeping covariate 1 at its mean
- Simulate the predicted values and the confidence intervals using simcf Plot the results

```
########################################
# Scenario 2:  Vary w2

# Start by calculating P(Y|w2) for different w1 values
w2range <- seq(0:20)/20

# Set up a matrix with the hypothetical scenarios (varied w1, all else equal)
xhypo <- cfMake(model, data, nscen = length(w2range))
for (i in 1:length(w2range)) {
  xhypo <- cfChange(xhypo, "w2", x=w2range[i], scen=i)
}

zhypo <- cfMake(varmodel, data, nscen = length(w2range))
for (i in 1:length(w2range)) {
  zhypo <- cfChange(zhypo, "w2", x=w2range[i], scen=i)
}

# Simulate the predicted Y's and CI's
simres.w2 <- hetnormsimpv(xhypo,simbetas,
                          zhypo,simgammas,
                          ci=0.95,
                          constant=1,varconstant=1)

# Plot them
yplot <- cbind(simres.w2$pe, simres.w2$lower, simres.w2$upper)
xplot <- cbind(w1range,w1range,w1range)
#pdf("heteroYvsW2.pdf")
matplot(y=yplot,
        x=xplot,
        type="l",
        lty=c("solid","dashed","dashed"),
        col=c("black"),
        xlab = "w2",
        ylab = "Predicted Y")
```



```
# since we set the
#dev.off()
```