Untitled

untilted

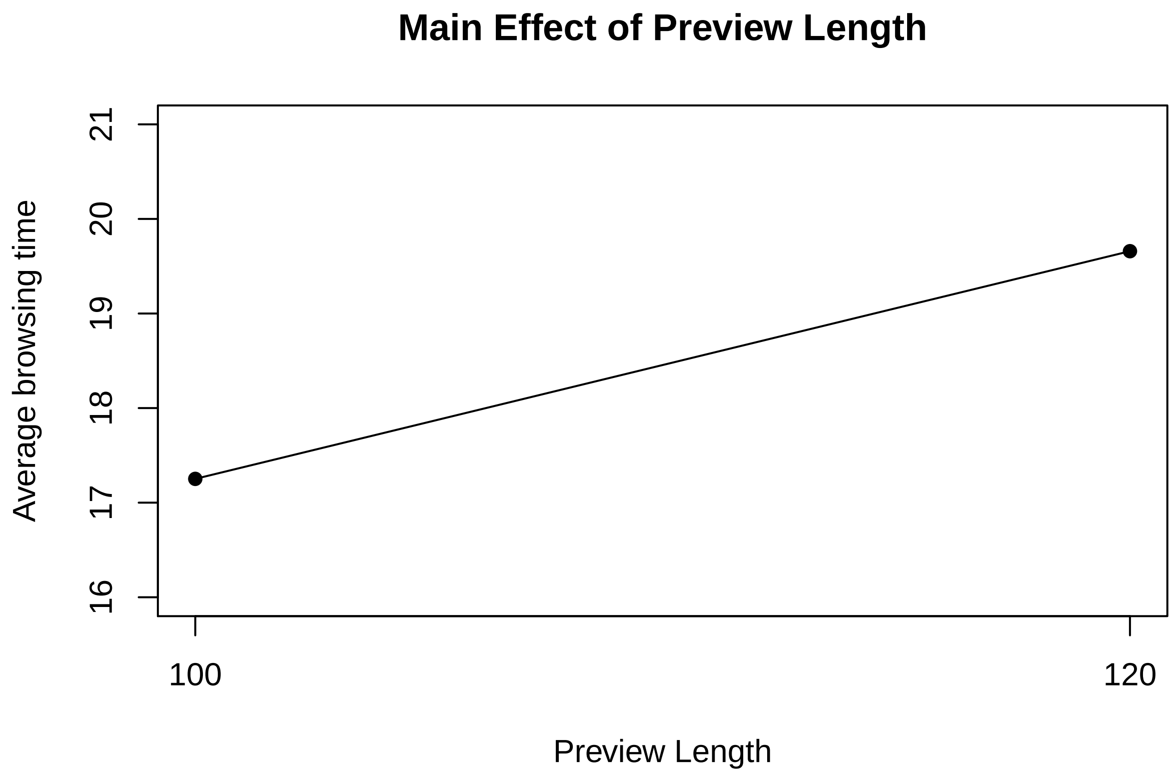
8/13/2021

```r
#Preparation
# Function for converting from natural units to coded units
convert.N.to.C <- function(U,UH,UL){
  x <- (U - (UH+UL)/2) / ((UH-UL)/2)
  return(x)
}

# Function for converting from coded units to natural units
convert.C.to.N <- function(x,UH,UL){
  U <- x*((UH-UL)/2) + (UH+UL)/2
  return(U)
}
```

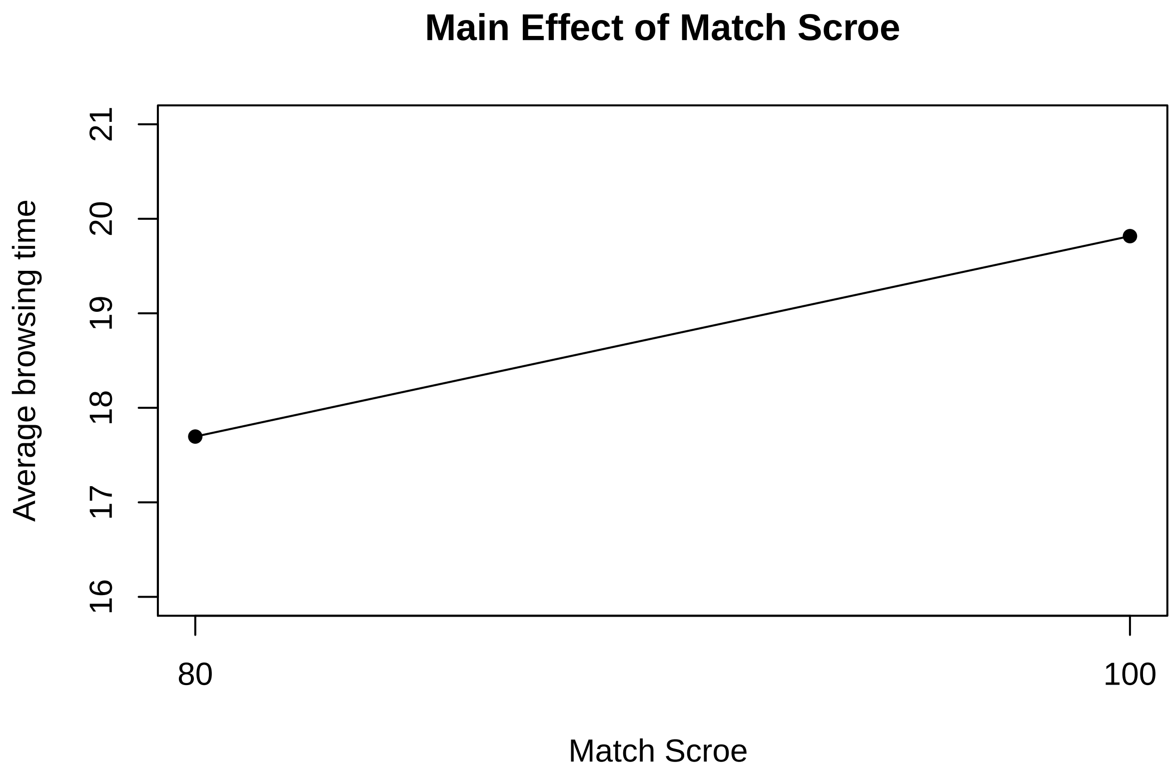
```r
#Phase I
data=read.csv("RESULTS_20756841_2021-08-14.csv",header=T)
data$Prev.Length <- convert.N.to.C(data$Prev.Length, 120, 100)
data$Match.Score <- convert.N.to.C(data$Match.Score, 100, 80)
data$Tile.Size <- convert.N.to.C(data$Tile.Size, 0.3,0.1)
y=data$Browse.Time
model=lm(y~(Prev.Length+Match.Score+Tile.Size)^2,data=data)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ (Prev.Length + Match.Score + Tile.Size)^2, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9713 -0.7254  0.0110  0.6473  3.9904
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             18.680247   0.046970 397.707   <2e-16 ***
## Prev.Length              0.896579   0.050213  17.856   <2e-16 ***
## Match.Score              0.984471   0.046970  20.960   <2e-16 ***
## Tile.Size                0.046902   0.046970   0.999    0.318
## Prev.Length:Match.Score -0.588317   0.050213 -11.716   <2e-16 ***
## Prev.Length:Tile.Size   -0.004575   0.050213  -0.091    0.927
## Match.Score:Tile.Size   -0.027946   0.046970  -0.595    0.552
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.004 on 793 degrees of freedom
## Multiple R-squared:  0.6941, Adjusted R-squared:  0.6918
## F-statistic: 299.9 on 6 and 793 DF,  p-value: < 2.2e-16
```

```r
agg.PL <- aggregate(data$Browse.Time, by = list(data$Prev.Length), FUN = mean)
plot(x = 1:2, y = agg.PL$x,
     pch = 16, ylim = c(16, 21), xaxt = "n", xlab = "Preview Length",
     ylab = "Average browsing time", main = "Main Effect of Preview Length")
lines(x = 1:2, y = agg.PL$x)
axis(side = 1, at = c(1,2), labels = c(100,120))
```




```
agg.PL <- aggregate(data$Browse.Time, by = list(data$Match.Score), FUN = mean)
plot(x = 1:2, y = agg.PL$x,
     pch = 16, ylim = c(16, 21), xaxt = "n", xlab = "Match Scroe ",
     ylab = "Average browsing time", main = "Main Effect of Match Scroe")
lines(x = 1:2, y = agg.PL$x)
axis(side = 1, at = c(1,2), labels = c(80,100))
```



drop tile.size

```r
#Phase II -2.1 Curvature Test

# Function to create blues
blue_palette <- colorRampPalette(c(rgb(247,251,255,maxColorValue = 255), rgb(8,48,107,maxColorValue = 25

#read data
data2=read.csv("2.0.csv",header=T)
table(data2$Prev.Length,data2$Match.Score)
```

```
##
##         80  90 100
##   100 100   0 100
##   110   0 100   0
##   120 100   0 100
```

```r
## Determine whether we're close to the optimum to begin with
## (i.e, check whether the pure quadratic effect is significant)
ph1 <- data.frame(y = data2$Browse.Time,
                  x1 = convert.N.to.C(U = data2$Prev.Length, UH = 120, UL = 100),
                  x2 = convert.N.to.C(U = data2$Match.Score, UH = 100, UL = 80))
ph1$xPQ <- (ph1$x1^2 + ph1$x2^2)/2

## Check the average browsing time in each condition:
aggregate(ph1$y, by = list(x1 = ph1$x1, x2 = ph1$x2), FUN = mean)
```

```
##   x1 x2        x
## 1 -1 -1 16.19875
## 2  1 -1 19.22809
## 3  0  0 19.19671
## 4 -1  1 19.19198
## 5  1  1 19.90541
```

```r
## The difference in average browsing time in factorial conditions vs. the center
## point condition
mean(ph1$y[ph1$xPQ != 0]) - mean(ph1$y[ph1$xPQ == 0])
```

```
## [1] -0.5656482
```

```r
## Check to see if that's significant
m <- lm(y~x1+x2+x1*x2+xPQ, data = ph1)
summary(m)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x1 * x2 + xPQ, data = ph1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.5463 -0.6435 -0.0156  0.7055  3.5585
##
## Coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 19.19671    0.09911 193.690  < 2e-16 ***
## x1           0.93569    0.04956  18.882  < 2e-16 ***
## x2           0.91764    0.04956  18.517  < 2e-16 ***
## xPQ         -0.56565    0.11081  -5.105 4.73e-07 ***
## x1:x2       -0.57898    0.04956 -11.683  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9911 on 495 degrees of freedom
## Multiple R-squared:  0.6352, Adjusted R-squared:  0.6323
## F-statistic: 215.5 on 4 and 495 DF,  p-value: < 2.2e-16
```

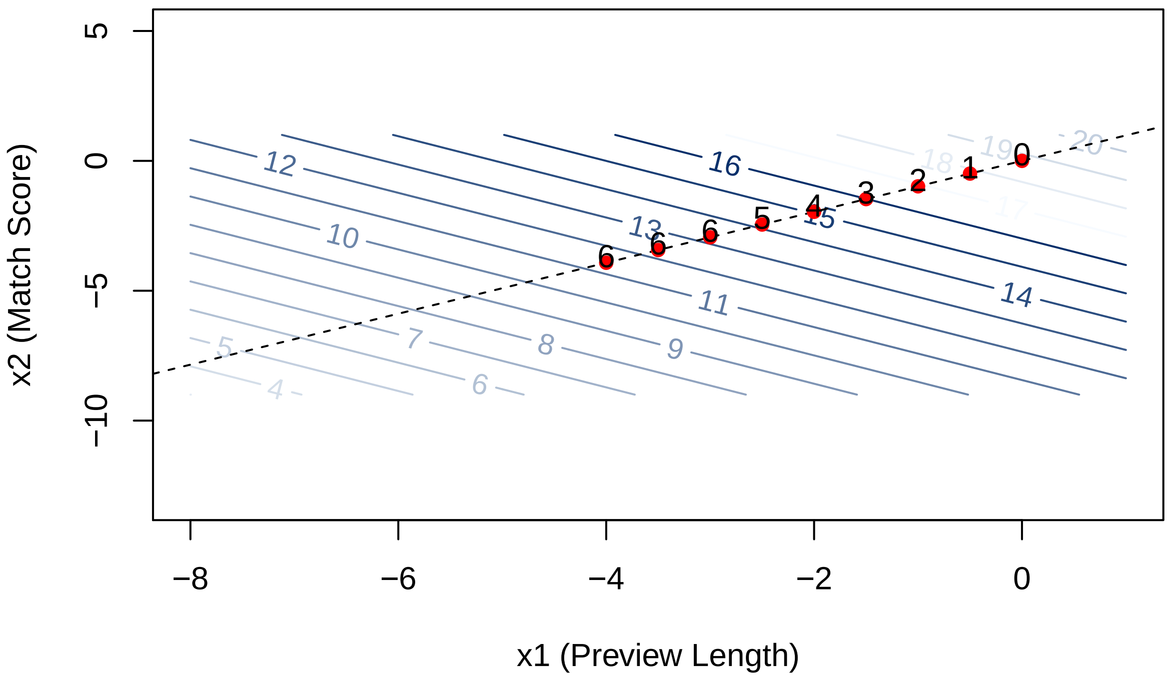
```
## steepest descent
library("plot3D")
```

```
## Warning: package 'plot3D' was built under R version 4.0.5
```

```
m.fo <- lm(y~x1+x2, data = ph1)
beta0 <- coef(m.fo)[1]
beta1 <- coef(m.fo)[2]
beta2 <- coef(m.fo)[3]
grd <- mesh(x = seq(convert.N.to.C(U = 30, UH = 120, UL = 100),
                    convert.N.to.C(U = 120, UH = 120, UL = 100),
                    length.out = 100),
            y = seq(convert.N.to.C(U = 0, UH = 100, UL = 80),
                    convert.N.to.C(U = 100, UH = 100, UL = 80),
                    length.out = 100))
x1 <- grd$x
x2 <- grd$y
eta.fo <- beta0 + beta1*x1 + beta2*x2
# 2D contour plot
contour(x = seq(convert.N.to.C(U = 30, UH = 120, UL =100),
                convert.N.to.C(U = 120, UH = 120, UL = 100),
                length.out = 100),
        y = seq(convert.N.to.C(U = 0, UH = 100, UL = 80),
                convert.N.to.C(U = 100, UH = 100, UL = 80),
                length.out = 100),
        z = eta.fo, xlab = "x1 (Preview Length)", ylab = "x2 (Match Score)",
        nlevels = 15, col = blue_palette(15), labcex = 0.9, asp=0.25)
abline(a = 0, b = beta2/beta1, lty = 2)
points(x = 0, y = 0, col = "red", pch = 16)


# The gradient vector
g <- matrix(c(beta1, beta2), nrow = 1)

# We will take steps of size 5 seconds in preview length. In coded units this is
PL.step <- convert.N.to.C(U = 110 + 5, UH = 120, UL = 100)
lamda <- PL.step/abs(beta1)

## Step 0: The center point we've already observed
x.old <- matrix(0, nrow=1, ncol=2)
```

```r
text(x = 0, y = 0+0.25, labels = "0")
step0 <- data.frame(Prev.Length = convert.C.to.N(x = 0, UH = 120, UL = 100),
                 Match.Score = convert.C.to.N(x = 0, UH = 100, UL = 80))

## Step 1:
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "1")
step1 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 2:
x.old <- x.new
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "2")
step2 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 3:
x.old <- x.new
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "3")
step3 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 4:
x.old <- x.new
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "4")
step4 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 5:
x.old <- x.new
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "5")
step5 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 6:
x.old <- x.new
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "6")
step6 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 7:
x.old <- x.new
```

```
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "6")
step7 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))

## Step 8:
x.old <- x.new
x.new <- x.old - lamda*g
points(x = x.new[1,1], y = x.new[1,2], col = "red", pch = 16)
text(x = x.new[1,1], y = x.new[1,2]+0.25, labels = "6")
```

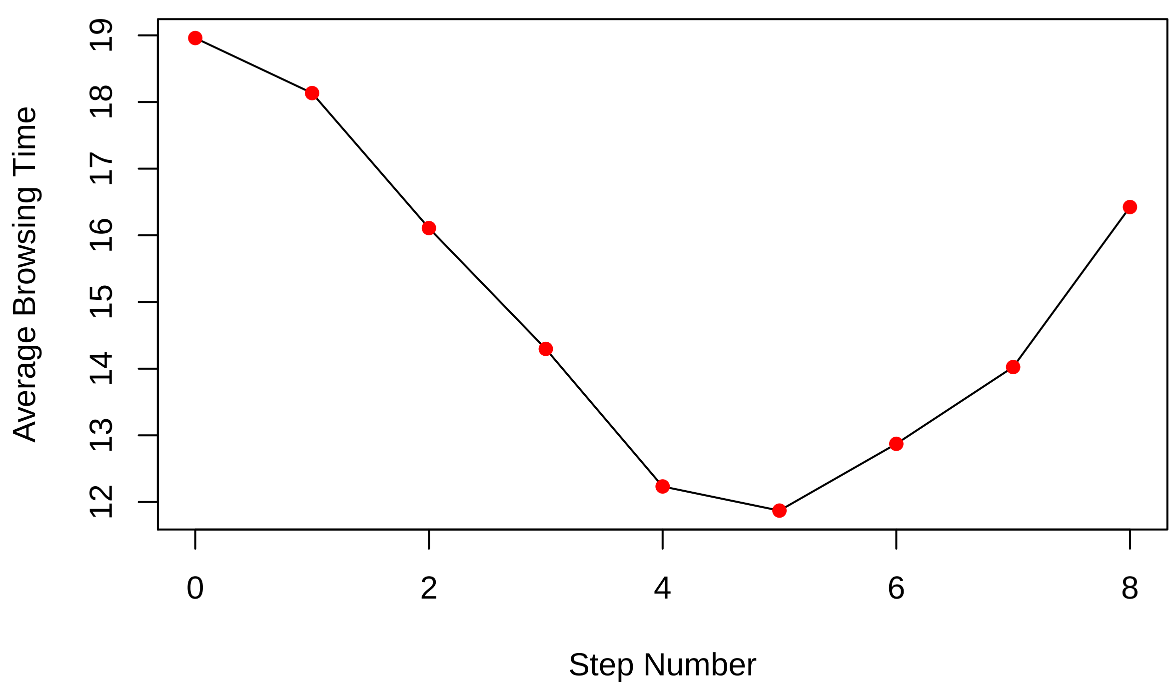



```
step8 <- data.frame(Prev.Length = convert.C.to.N(x = x.new[1,1], UH = 120, UL = 100),
                    Match.Score = convert.C.to.N(x = x.new[1,2], UH = 100, UL = 80))


## The following is a list of the conditions along the path of steepest descent
pstd.cond <- data.frame(Step = 0:8, rbind(step0, step1, step2, step3, step4, step5, step6, step7, step8
pstd.cond
```

```
##   Step Prev.Length Match.Score
## 1    0         110    90.00000
## 2    1         105    85.09648
## 3    2         100    80.19297
## 4    3          95    75.28945
```

```
## 5    4          90    70.38593
## 6    5          85    65.48241
## 7    6          80    60.57890
## 8    7          75    55.67538
## 9    8          70    50.77186
```

```
##########Find MOI for each
netflix.ph2 <- read.csv("3.0.csv",header=T)

## Calculate the average browsing time in each of these conditions and find the
## condition that minimizes it
pstd.means <- aggregate(netflix.ph2$Browse.Time,
                        by = list(Prev.Length = netflix.ph2$Prev.Length,
                                  Match.Score = netflix.ph2$Match.Score),
                        FUN = mean)

plot(x = 0:8, y = pstd.means$x[9:1],
     type = "l", xlab = "Step Number", ylab = "Average Browsing Time")
points(x = 0:8, y = pstd.means$x[9:1],
       col = "red", pch = 16)
```




```
pstd.cond
```

```
##   Step Prev.Length Match.Score
## 1    0         110    90.00000
```

```
## 2    1         105    85.09648
## 3    2         100    80.19297
## 4    3          95    75.28945
## 5    4          90    70.38593
## 6    5          85    65.48241
## 7    6          80    60.57890
## 8    7          75    55.67538
## 9    8          70    50.77186
```

```
pstd.means
```

```
##   Prev.Length Match.Score        x
## 1          70          51 16.42463
## 2          75          56 14.02431
## 3          80          60 12.87261
## 4          85          65 11.87106
## 5          90          70 12.23298
## 6          95          75 14.29648
## 7         100          80 16.10895
## 8         105          85 18.13371
## 9         110          90 18.95946
```

```
convert.C.to.N(sqrt(2),95,75)
```

```
## [1] 99.14214
```

```
convert.C.to.N(-sqrt(2),95,75)
```

```
## [1] 70.85786
```

```
convert.C.to.N(sqrt(2),75,55)
```

```
## [1] 79.14214
```

```
convert.C.to.N(-sqrt(2),75,55)
```

```
## [1] 50.85786
```

```
#Curvature Test for Final Region
## (i.e, check whether the pure quadratic effect is significant)
data3=read.csv("3.0_3.csv",header=T)
ph2 <- data.frame(y = data3$Browse.Time,
                  x1 = convert.N.to.C(U = data3$Prev.Length, UH = 95, UL = 75),
                  x2 = convert.N.to.C(U = data3$Match.Score, UH = 75, UL = 55))
ph2$xPQ <- (ph2$x1^2 + ph2$x2^2)/2

## Check the average browsing time in each condition:
aggregate(ph2$y, by = list(x1 = ph2$x1, x2 = ph2$x2), FUN = mean)
```

```
##   x1 x2        x
## 1  0 -1 14.38685
## 2 -1  0 11.02837
## 3  0  0 11.75241
## 4  1  0 13.35166
## 5  0  1 11.67958
```

```
## The difference in average browsing time in factorial conditions vs. the center
## point condition
mean(ph2$y[ph2$xPQ != 0]) - mean(ph2$y[ph2$xPQ == 0])
```

```
## [1] 0.8592075
```

```
## Check to see if that's significant
m1 <- lm(y~x1+x2+x1*x2+xPQ, data = ph2)
summary(m1)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x1 * x2 + xPQ, data = ph2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.92324 -0.69237  0.09461  0.76124  2.70243
##
## Coefficients: (1 not defined because of singularities)
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 11.75241    0.10320 113.884  < 2e-16 ***
## x1           1.16165    0.07297  15.919  < 2e-16 ***
## x2          -1.35364    0.07297 -18.550  < 2e-16 ***
## xPQ          1.71841    0.23075   7.447 4.26e-13 ***
## x1:x2             NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.032 on 496 degrees of freedom
## Multiple R-squared:  0.5683, Adjusted R-squared:  0.5657
## F-statistic: 217.7 on 3 and 496 DF,  p-value: < 2.2e-16
```

```
# phase III
netflix <- read.csv("4.0.csv",header=T)
table(netflix$Prev.Length,netflix$Match.Score)
```

```
##
##       51  55  65  75  79  85
##   70   0   0   0   0   0 100
##   71   0   0 100   0   0   0
##   75   0 100   0 100   0   0
##   80   0   0   0   0   0 100
##   85 100   0 100   0 100   0
##   90   0   0   0   0   0 100
##   95   0 100   0 100   0   0
##   99   0   0 100   0   0   0
```

```r
netflix$Prev.Length <- convert.N.to.C(netflix$Prev.Length,75, 55)
netflix$Match.Score <- convert.N.to.C(netflix$Match.Score,83, 63)
## We then fit the full 2nd-order response surface
model <- lm(Browse.Time ~ Prev.Length + Match.Score + Prev.Length*Match.Score + I(Prev.Length^2) + I(Ma
summary(model)
```

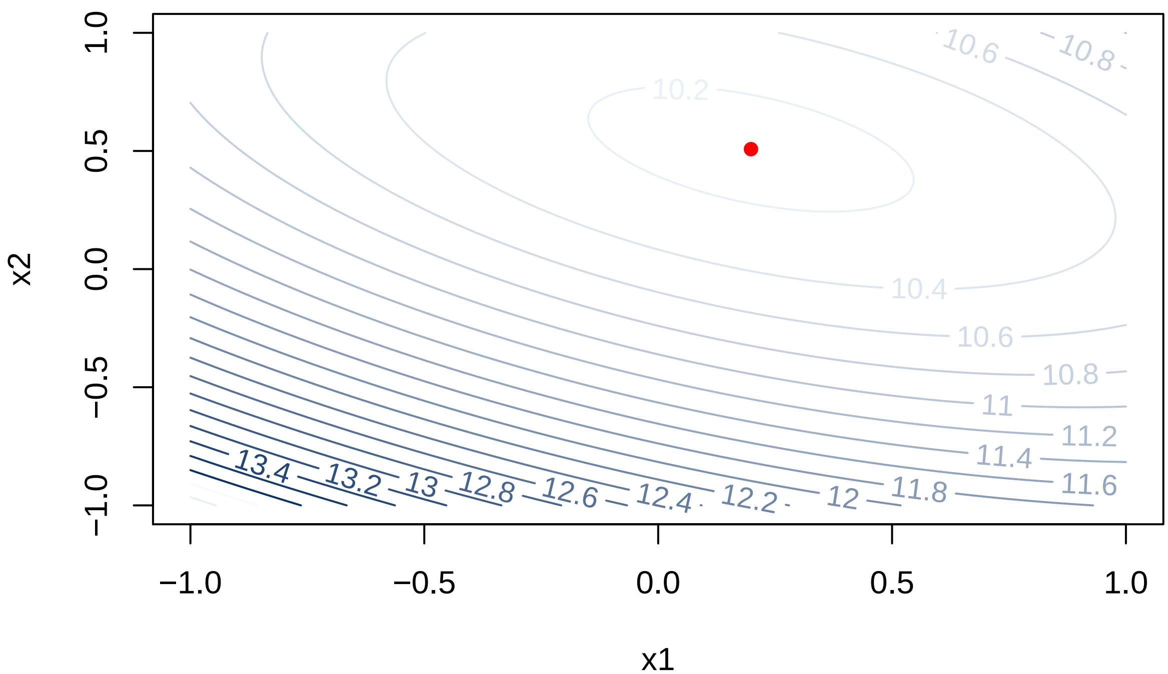
```
##
## Call:
## lm(formula = Browse.Time ~ Prev.Length + Match.Score + Prev.Length *
##     Match.Score + I(Prev.Length^2) + I(Match.Score^2), data = netflix)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.4955 -0.6306 -0.0220  0.6975  3.5006
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             10.48377    0.12407  84.501  < 2e-16 ***
## Prev.Length             -0.57059    0.16465  -3.466 0.000548 ***
## Match.Score             -1.09138    0.06220 -17.546  < 2e-16 ***
## I(Prev.Length^2)         0.54196    0.04437  12.215  < 2e-16 ***
## I(Match.Score^2)         0.93800    0.02914  32.184  < 2e-16 ***
## Prev.Length:Match.Score  0.70038    0.03420  20.479  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.993 on 1194 degrees of freedom
## Multiple R-squared:  0.7346, Adjusted R-squared:  0.7335
## F-statistic: 660.9 on 5 and 1194 DF,  p-value: < 2.2e-16
```

```r
anova(model)
```

```
## Analysis of Variance Table
##
## Response: Browse.Time
##                           Df  Sum Sq Mean Sq  F value    Pr(>F)
## Prev.Length                1 1548.65 1548.65 1570.624 < 2.2e-16 ***
## Match.Score                1  566.54  566.54  574.582 < 2.2e-16 ***
## I(Prev.Length^2)           1   27.85   27.85   28.247 1.273e-07 ***
## I(Match.Score^2)           1  701.66  701.66  711.615 < 2.2e-16 ***
## Prev.Length:Match.Score    1  413.54  413.54  419.409 < 2.2e-16 ***
## Residuals               1194 1177.30    0.99
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
## Let's visualize this surface:
beta0 <- coef(model)[1]
beta1 <- coef(model)[2]
beta2 <- coef(model)[3]
beta12 <- coef(model)[6]
beta11 <- coef(model)[4]
beta22 <- coef(model)[5]
```

```r
grd <- mesh(x = seq(convert.N.to.C(U = 61, UH = 81, UL = 61),
                    convert.N.to.C(U = 81, UH = 81, UL = 61),
                    length.out = 100),
            y = seq(convert.N.to.C(U = 63, UH = 83, UL = 63),
                    convert.N.to.C(U = 83, UH = 83, UL = 63),
                    length.out = 100))
x1 <- grd$x
x2 <- grd$y
eta.so <- beta0 + beta1*x1 + beta2*x2 + beta12*x1*x2 + beta11*x1^2 + beta22*x2^2

# 2D contour plot (coded units)
contour(x = seq(convert.N.to.C(U = 61, UH = 81, UL = 61),
                    convert.N.to.C(U = 81, UH = 81, UL = 61),
                length.out = 100),
        y = seq(convert.N.to.C(U = 63, UH = 83, UL = 63),
                    convert.N.to.C(U = 83, UH = 83, UL = 63),
                length.out = 100),
        z = eta.so, xlab = "x1", ylab = "x2",
        nlevels = 20, col = blue_palette(20), labcex = 0.9)

## Let's find the maximum of this surface and the corresponding factor levels
## at which this is achieved
b <- matrix(c(beta1,beta2), ncol = 1)
B <- matrix(c(beta11, 0.5*beta12, 0.5*beta12, beta22), nrow = 2, ncol = 2)
x.s <- -0.5*solve(B) %*% b
points(x = x.s[1], y = x.s[2], col = "red", pch = 16)
```




```
# The predicted book rate at this configuration is:
eta.so.opt=beta0+beta1*x.s[1]+beta2*x.s[2]+beta12*x.s[1]*x.s[2]+beta11*x.s[1]^2+beta22*x.s[2]^2
eta.so.opt
```

```
## (Intercept)
##    10.15013
```

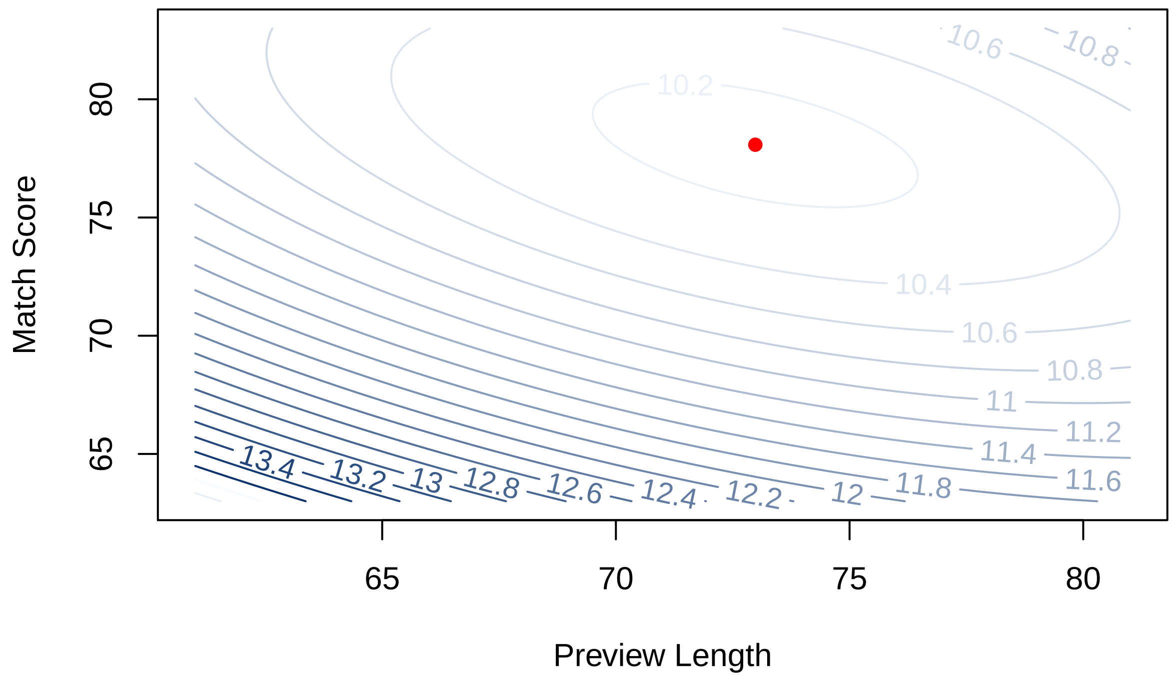
```
# In natural units this optimum is located at
convert.C.to.N(x = x.s[1,1], UH = 81, UL = 61)
```

```
## [1] 72.98363
```

```
convert.C.to.N(x = x.s[2,1], UH = 83, UL = 63)
```

```
## [1] 78.07703
```

```
# Remake the contour plot but in natural units
contour(x = seq(61, 81, length.out = 100),
        y = seq(63, 83, length.out = 100),
        z = eta.so, xlab = "Preview Length", ylab = "Match Score",
        nlevels = 20, col = blue_palette(20), labcex = 0.9)

points(x = convert.C.to.N(x = x.s[1,1], UH = 81, UL = 61),
       y = convert.C.to.N(x = x.s[2,1], UH = 83, UL = 63),
       col = "red", pch = 16)
```




```
## 95% prediction interval at this optimum:
pred <- predict(model, newdata = data.frame(Prev.Length=x.s[1,1], Match.Score=x.s[2,1]), type = "respons
pred
```

```
## $fit
##        1
## 10.15013
##
## $se.fit
## [1] 0.09763311
##
## $df
## [1] 1194
##
## $residual.scale
## [1] 0.9929812
```

```
print(paste("Prediction: ", pred$fit, sep = ""))
```

```
## [1] "Prediction: 10.1501251643416"
```

```
print(paste("95% Prediction interval: (", pred$fit-qnorm(0.975)*pred$se.fit, ",", pred$fit+qnorm(0.975)*
```

```
## [1] "95% Prediction interval: (9.95876778309328,10.3414825455898)"
```