PS 2707 HW1

Orly Olbum

Question 1

Please do Question 12.8 from ISL. However, use the selected_ANES data used in lecture on 24 Jan.

In Section 12.2.3, a formula for calculating PVE was given in Equation 12.10. We also saw that the PVE can be obtained using the sdev output of the prcomp() function.

Calculate PVE in two ways:
** Using the sdev output of the prcomp() function, as was done in Section 12.2.3.*
** By applying Equation 12.10 directly. That is, use the prcomp() function to compute the principal component loadings. Then, use those loadings in Equation 12.10 to obtain the PVE.*

These two approaches should give the same results.

Hint: You will only obtain the same results in (a) and (b) if the same data is used in both cases. For instance, if in (a) you performed prcomp() using centered and scaled variables, then you must center and scale the variables before applying Equation 12.10 in (b).

```
# scale
std_ANES = apply(selected_ANES, MARGIN = 2, scale)

# first way
fit1 = prcomp(std_ANES, scale = TRUE)
sdev1 = summary(fit1)$importance[2,]
round(sdev1, 4)
```

```
##    PC1    PC2    PC3    PC4    PC5    PC6    PC7    PC8    PC9   PC10   PC11
## 0.2821 0.1045 0.0959 0.0640 0.0481 0.0432 0.0418 0.0368 0.0360 0.0324 0.0297
##   PC12   PC13   PC14   PC15   PC16   PC17   PC18   PC19   PC20
## 0.0295 0.0282 0.0260 0.0244 0.0235 0.0208 0.0158 0.0153 0.0021
```

```
# second way
fit2 = prcomp(std_ANES, scale = TRUE)

# find total variance explained by each principal component
pve = fit2$sdev^2 / sum(fit2$sdev^2)
round(pve, 4)
```

```
##  [1] 0.2821 0.1045 0.0959 0.0640 0.0481 0.0433 0.0418 0.0368 0.0360 0.0325
## [11] 0.0297 0.0295 0.0282 0.0260 0.0244 0.0235 0.0208 0.0158 0.0153 0.0021
```

Are they equal?

```
first = as.data.frame(round(sdev1, 4))[,1]
second = round(pve, 4)
combo = as.data.frame(cbind(first, second))
combo$equal = ifelse(combo$first == combo$second, TRUE, FALSE)
summary(combo$equal)
```

```
##    Mode   FALSE    TRUE
## logical       2      18
```

The FALSE's are rounding errors. They are equal!

Question 2

- *Using the selected_ANES data, please manually estimate the first principal component using the alternating regression method discussed in lecture (see also Section 12.2.2 of ISL)*
  - *Hint: Use lm and a for loop. Remember to respect the constraints on β. You can estimate β without the constraints, and then scale it to satisfy them.*
  - *The answer can be done in fewer than 10 lines of code.*
- *Confirm that your answer is correct by comparing it against the results of some other method of estimating principal components.*

```
# alternating regression method
# reg_pc = lm(as.matrix(fit1) ~ factor(selected_ANES$party_id))
# sapply(summary(reg_pc), FUN = function(i){i$r.squared})
```

```
# trying something with SVD???

# means of each question
means = map_dbl(selected_ANES, mean)
# subtract on corresponding row
data_c = map2(selected_ANES, means, .f = function(x, mean) x - mean)
data_c = data.frame(data_c)
# now perform svd
svd_main = svd(data_c)
# components
u = svd_main$u; v = svd_main$v; d = svd_main$d
# u; v; d
egv = d ^ 2 / (nrow(selected_ANES) - 1)
egv
```

```
##  [1] 8.57230312 3.87809554 2.30577044 1.75058325 1.35573835 0.98096939
##  [7] 0.91044752 0.88140918 0.84763273 0.59982124 0.58382330 0.44516389
## [13] 0.37627597 0.32739288 0.30228061 0.29070881 0.28499541 0.22697148
## [19] 0.22034458 0.08600453
```

We like eigenvalues over 1.

```r
# compare
pc = prcomp(selected_ANES, scale = TRUE)
summary(pc)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     2.3753 1.4456 1.38484 1.13104 0.98047 0.93009 0.91457
## Proportion of Variance 0.2821 0.1045 0.09589 0.06396 0.04807 0.04325 0.04182
## Cumulative Proportion  0.2821 0.3866 0.48246 0.54642 0.59449 0.63774 0.67957
##                            PC8    PC9    PC10    PC11    PC12    PC13   PC14
## Standard deviation     0.85759 0.8485 0.80564 0.77115 0.76775 0.75077 0.7211
## Proportion of Variance 0.03677 0.0360 0.03245 0.02973 0.02947 0.02818 0.0260
## Cumulative Proportion  0.71634 0.7523 0.78479 0.81452 0.84399 0.87218 0.8982
##                           PC15    PC16    PC17    PC18    PC19   PC20
## Standard deviation     0.69811 0.68527 0.64489 0.56144 0.55357 0.2049
## Proportion of Variance 0.02437 0.02348 0.02079 0.01576 0.01532 0.0021
## Cumulative Proportion  0.92255 0.94602 0.96682 0.98258 0.99790 1.0000
```

Disclaimer - I recognize the answer given here does not satisfy the question. I understand that the goal of PCA in a regression context is to find a linear function of the vector of random variables and the sum of constants*those random variables with maximum variance, find another that is uncorrelated, and continue to iterate - the desired outcome is most variation is accounted for with some number of principal components less than the amount of constants. Just couldn't quite figure out how to execute.

Question 3

We will use data on the US Supreme Court for the remaining questions. The data can be loaded as follows:

```r
suppressPackageStartupMessages(library(MCMCpack))
data(SupremeCourt)
# Remove one case with missing data
SupremeCourt <- na.omit(SupremeCourt)
```

Please fit (traditional) factor analysis model. Note: This will treat the number of justices as "questions" and cases as "individuals". Justify your choice of the number of factors.

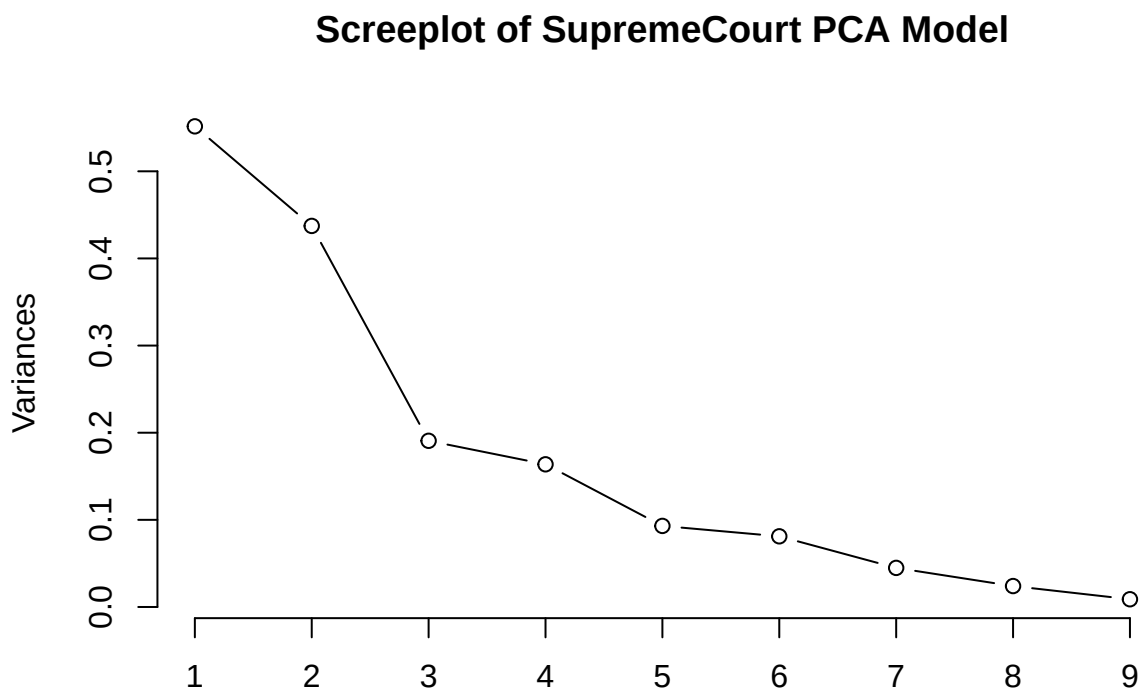
```r
fit_fa = factanal(x = SupremeCourt, factors = 2, scores = 'regression')
fit_fa
```

```
##
## Call:
## factanal(x = SupremeCourt, factors = 2, scores = "regression")
##
## Uniquenesses:
## Rehnquist   Stevens  O'Connor    Scalia   Kennedy    Souter    Thomas  Ginsburg
##     0.634     0.776     0.856     0.127     0.851     0.274     0.043     0.228
##    Breyer
##     0.548
##
## Loadings:
```

```
##           Factor1 Factor2
## Rehnquist   0.602
## Stevens    -0.446   0.159
## O'Connor    0.178   0.335
## Scalia      0.925  -0.131
## Kennedy     0.358   0.142
## Souter              0.852
## Thomas      0.976
## Ginsburg   -0.227   0.849
## Breyer     -0.344   0.578
##
##                Factor1 Factor2
## SS loadings        2.7   1.962
## Proportion Var     0.3   0.218
## Cumulative Var     0.3   0.518
##
## Test of the hypothesis that 2 factors are sufficient.
## The chi square statistic is 50.3 on 19 degrees of freedom.
## The p-value is 0.000119
```

After 2 factors, the p-value of the chi-square test goes above the significance threshold. The chi-square test is testing the hypothesis that the designated number of factors are sufficient, and because 3 puts the p-value too high, 2 should be accurate. We can also look at the screeplot:

```
fit_pc = prcomp(SupremeCourt)
screeplot(fit_pc, type = 'line', main = 'Screeplot of SupremeCourt PCA Model')
```



Sure enough, the elbow at 3 tells us what the chi-square tests confirms (if we were interested in 95% confidence - but if we want 90% confidence, 3 would suffice).

Question 4

Fit a one dimensional ideal point model on the dataset using one of the software packages discussed in lecture.
* *Produce some simple tests for whether the model has converged.*
* *Hint: Use the code from lecture or MCMCpack's documentation.*

```
# using mcmcpack
fit_mcmc = MCMCirt1d(t(SupremeCourt),
                     burnin = 4000,
                     mcmc = 8000,
                     thin = 10) # excluding verbose for purposes of cleanliness
```

```
# do all converge?
# check geweke
geweke.diag(fit_mcmc)$z
```

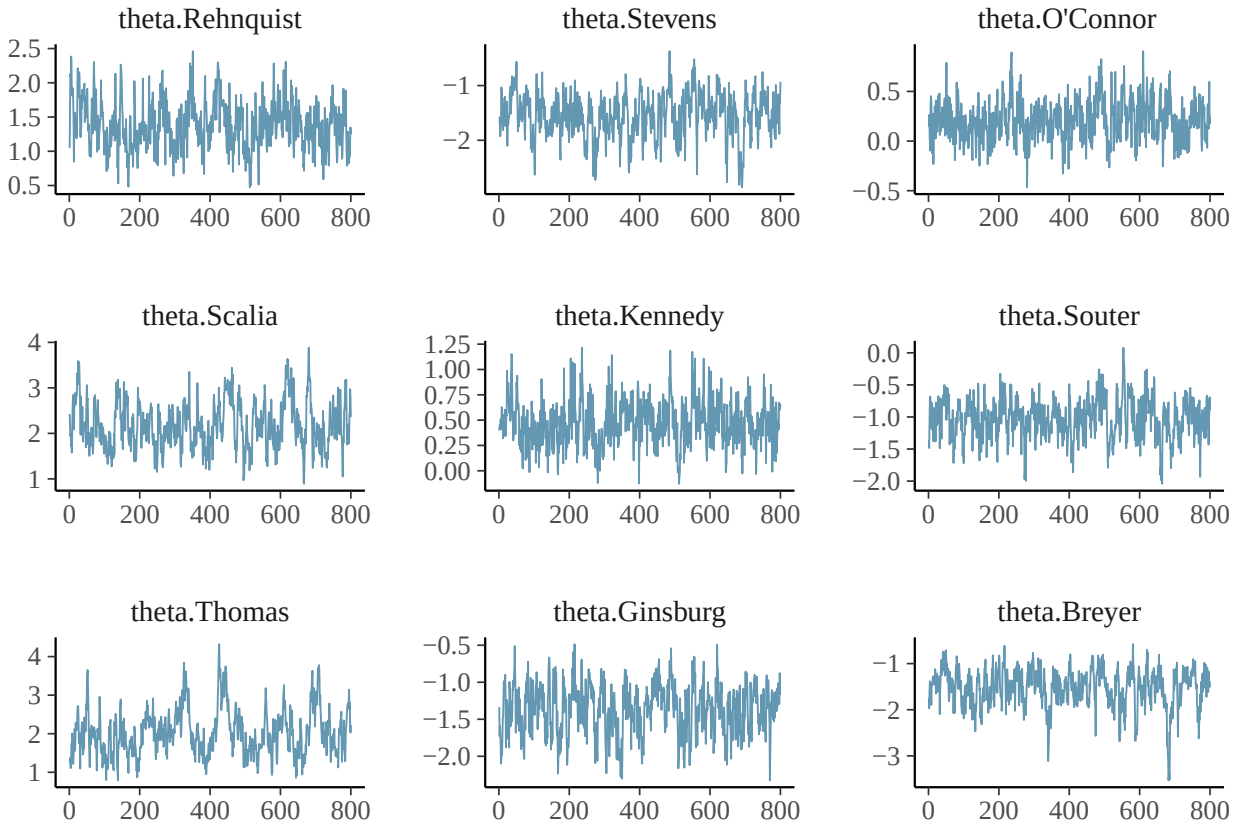
```
## theta.Rehnquist   theta.Stevens  theta.O'Connor    theta.Scalia   theta.Kennedy
##       2.1684150       1.4886998      -0.7417315       0.5098682       0.5958606
##    theta.Souter    theta.Thomas  theta.Ginsburg    theta.Breyer
##       0.1494533      -0.6508868      -0.6655625       1.7839058
```

```
# check rhat
summary_probit_mcmc = summarize_draws(fit_mcmc)
summary_probit_mcmc$rhat
```

```
## [1] 1.000006 1.011817 1.019633 1.013634 1.000791 1.023972 1.010738 1.006904
## [9] 1.001899
```

The geweke test has some that are significant-looking, but since the r-hats are all under 1.1, we should be set! We want the r-hat's to be close to 1 and we are even happier if they are below 1.05. But let's check a plot to be sure.

```
mcmc_trace(fit_mcmc)
```



These look pretty good. No really obvious trends, lots of "noisy" looking plots. But we might like to see even less of a trend, and since the geweke test wasn't as promising, let's try again.

```
fit_mcmc2 = MCMCirt1d(t(SupremeCourt),
                      burnin = 1000,
                      mcmc = 100000,
                      thin = 5)

geweke.diag(fit_mcmc2)$z
```

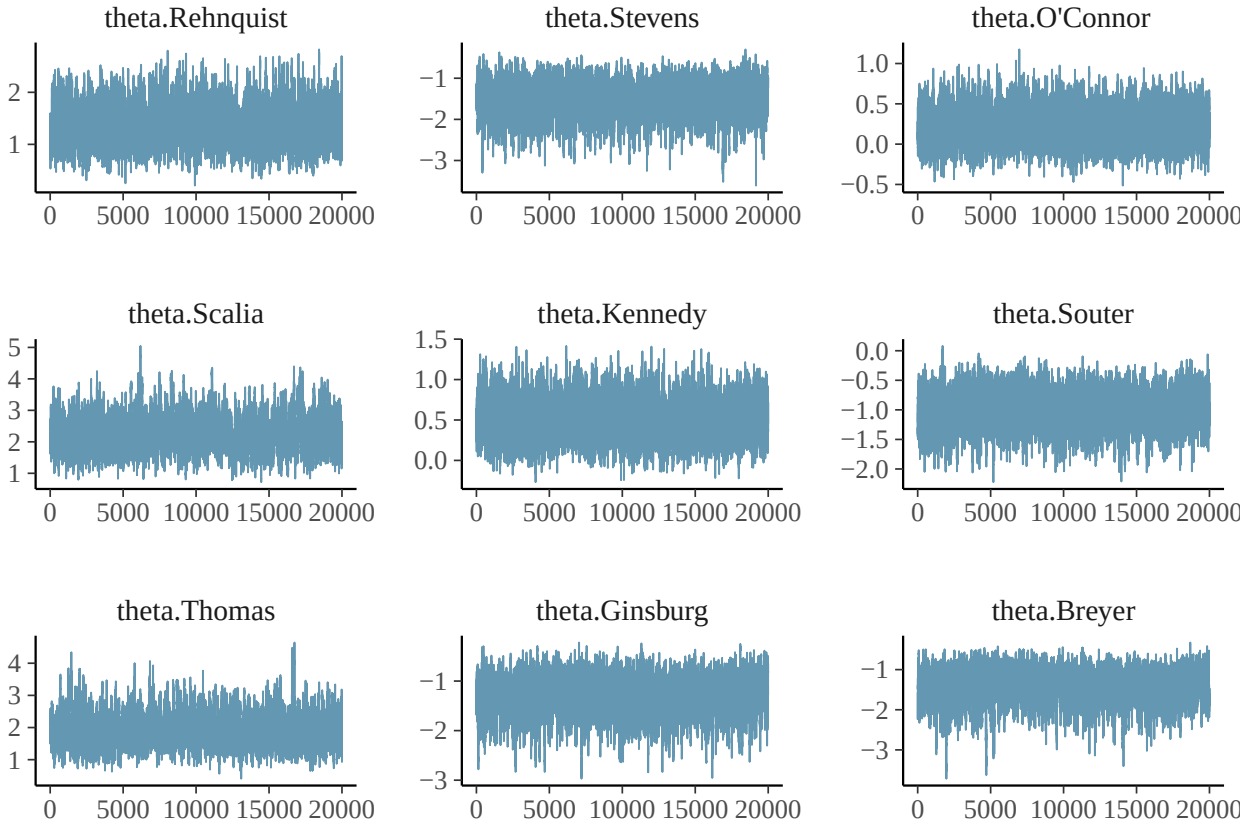
```
## theta.Rehnquist   theta.Stevens  theta.O'Connor    theta.Scalia   theta.Kennedy
##      0.68249755     -0.72711737     -0.03753253     -0.83316572     -0.05082264
##    theta.Souter    theta.Thomas  theta.Ginsburg    theta.Breyer
##     -1.81304284      0.33251863     -2.04040753     -1.13547812
```

```
summary_probit_mcmc2 = summarize_draws(fit_mcmc2)
summary_probit_mcmc2$rhat
```

```
## [1] 1.000513 1.000113 1.000610 1.000038 1.000780 1.000648 1.001579 1.000887
## [9] 1.000656
```

Even better!

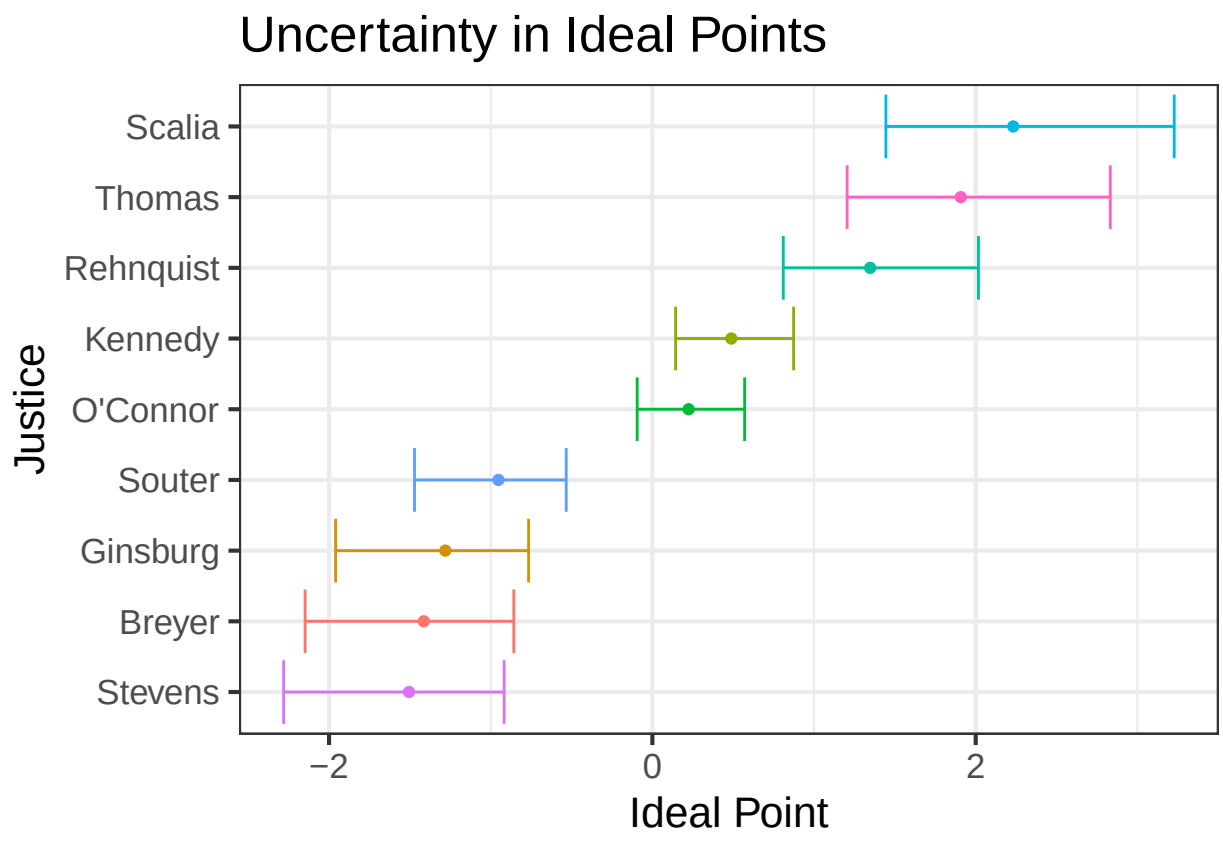
```
mcmc_trace(fit_mcmc2)
```



Super noisy, so I am happy. We have no evidence to suggest they have *not* converged.

Produce the following plots:
* *Create a well-labeled and informative plot that ranks justices from highest to lowest scores and shows the uncertainty in the ideal points.*
* *What is the correlation between this ideal point and the score estimated in Question 3?*

```
# plot uncertainty
summary_probit_mcmc2$variable = c("Rehnquist", "Stevens", "O'Connor", "Scalia", "Kennedy",
                                  "Souter", "Thomas", "Ginsburg", "Breyer")
ggplot() +
  geom_point(aes(x = reorder(variable, median), y = median, colour =  factor(variable)),
             data = summary_probit_mcmc2) +
  geom_errorbar(aes(x = variable, ymin = q5, ymax = q95, colour =  factor(variable)),
                data = summary_probit_mcmc2) +
  theme_bw(base_size = 16) +
  coord_flip() +
  labs(y = 'Ideal Point', x = 'Justice') +
  ggtitle('Uncertainty in Ideal Points') +
  theme(legend.position = "none")
```



The plot shows less uncertainty for the middle scores and more uncertainty for the higher and lower scores - not necessarily what I would have expected, but it might make sense.

```
# scores from q3
scores_pc = as.matrix(SupremeCourt) %*% fit_pc$rotation - fit_pc$x
scores_pc[1,]
```

```
##         PC1         PC2         PC3         PC4         PC5         PC6
## -0.46817001 -1.40170943  0.07141677 -0.51634517 -0.45421699  0.04903721
##         PC7         PC8         PC9
##  0.11618383  0.37914344 -0.16079285
```

```
# scores now
summary_probit_mcmc2$mean
```

```
## [1]  1.3720969 -1.5413388  0.2297528  2.2751447  0.4972458 -0.9705330  1.9458984
## [8] -1.3099371 -1.4470210
```

```
# correlation
cor(scores_pc[1,], summary_probit_mcmc2$mean)
```

```
## [1] 0.05158283
```

Very low correlation - or maybe I am doing something wrong.

Question 5

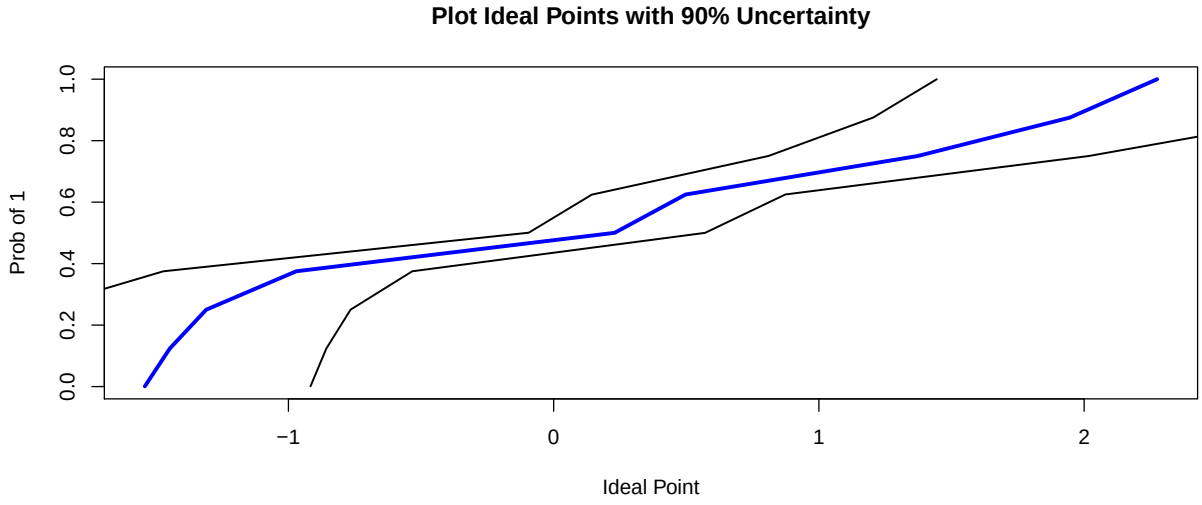
*Using the model fit in Question 4, choose one question and create a plot that shows the probability of responding "1" as the ideal point varies. * Please incorporate the uncertainty in this curve.*
** Hint: Loop over the posterior draws and then find the credible interval over the predicted probabilities at each point on the curve.*

```
# we have the draws
draws = summary_probit_mcmc2

# from draws we have mean, q5, and q95 to draw the curve
# just need corresponding probabilities

yaxis = c(0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1)
names = as.data.frame(draws$variable); means = as.data.frame(draws$mean)
q5 = as.data.frame(draws$q5); q95 = as.data.frame(draws$q95)
all = cbind(names, means, q5, q95)
colnames(all) = c('justice', 'means', 'q5', 'q95')
all = all %>%
  arrange(means)
all = cbind(all, yaxis)

# plot
plot(all$means, all$yaxis, type = 'n',
     main = 'Plot Ideal Points with 90% Uncertainty',
     xlab = 'Ideal Point', ylab = 'Prob of 1')
smoother = smooth.spline(all$means, all$yaxis, spar = 0.1)
lines(smoother, lwd = 3, col = 'blue')
smoother2 = smooth.spline(all$q5, all$yaxis, spar = 0.1)
lines(smoother2, lwd = 1.5)
smoother3 = smooth.spline(all$q95, all$yaxis, spar = 0.1)
lines(smoother3, lwd = 1.5)
```



I know this is not nearly as smooth as what you were expecting but it's what I can produce at the moment. Again, conceptually I have it - we create probabilities as you showed in class and then we'd loop over the draws for each justice with the uncertainty to plot the actual points.