A different approach

One of the problems with the way we have been adjusting the theta values (adding and subtracting a constant) is that as the adjustment gets larger, the distribution of abilities spreads out and departs more and more from the standard normal that we started with (it will still be normal - you can prove that linear combinations of independent normal random variables are still normal - it just won't be standard normal).

It would be nice if we could preserve the original standard normal distribution because most IRT models assume this.

To preserve the original distribution, if we "promote" a student to a higher theta value, we need to "demote" a specific student from the new theta value to the old.

In other words, to preserve the original distribution, we have to always exchange the theta values for a specific pair of students. That way, the overall distribution of theta values doesn't change.

The original theta samples were generated with `rnorm()` and have no particular spacing. To exchange with a student who is, say 10 percentile points higher, once we figure out the new theta value we would have to search through the 70,000 theta values until we found a suitable candidate to exchange. It is unlikely there would even be a student with that exact theta value, so we would probably have to search the entire sample to find the closest match. Doing this thousands of times would be expensive.

One solution would be to generate an array of 70,000 theta values at equally spaced *quantiles*. This guarantees that we can find students with a specific theta values (at least for the values we generated), and we can find the right student without having to search through the array if we generate them in order to begin with. I don't think the order matters to the SGP software, but we can verify this.

Basically, we would divide up the range of theta values into 70,000 subintervals in such a way that the probability of falling into each interval is the same, 1/70000, if we were to randomly select one of the 70,000 students.

This sounds like it would be hard but actually the `qnorm()` function will do it, all we have to do is supply it 70,000 equally spaced points in the open interval (0,1).

Recall that in R the `qxxx()` function for a distribution gives the inverse of the cumulative distribution function (CDF) F for distribution `xxx`, so it turns a probability value p, which will be between zero and one, into the x-coordinate that has $F(x) = p$.

In the case of the standard normal, given a probability p, `qnorm(p)` is the x value that satisfies the equation:

$$\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) dz = p$$

We need a computer and a function like `qnorm` to solve this, because as it turns

out it is impossible to find an antiderivative of

$$\exp\left(-\frac{z^2}{2}\right)$$

so the integration can't be done in closed form.

Start by dividing the open interval $(0, 1)$ into 70,000 equal subintervals (we don't include 0 or 1 because qnorm would give $-\infty$ and ∞, respectively, for those values).

Having excluded 0 and 1 from the list of p values, we'll just let the max and min fall where they may. The smallest value will be:

```
1/70001

## [1] 1.428551e-05

qnorm(1/70001)

## [1] -4.184567
```

and the largest will be:

```
70000/70001

## [1] 0.9999857

qnorm(70000/70001)

## [1] 4.184567
```

Now we'll generate the 70,000 p values in the open interval $(0, 1)$.

The i^{th} p value will be:

$$p_i = \frac{i}{70001}, \quad i = 1, 2, \ldots, 70000$$

and the corresponding θ_i value will be:

$$\theta_i = qnorm(p_i)$$

The fact that R is so fogiving of expressions that mix arrays and constants makes this very easy to code:

```
x=1:70000                          #generate 70,000 equally spaced numbers
p=x/70001                          #scale them into the open interval (0,1)
theta=qnorm(p)                     #compute the corresponding theta values
str(theta)                         #display a bit of the result

##  num [1:70000] -4.18 -4.02 -3.93 -3.86 -3.8 ...
```

We'll perform a couple of quick sanity checks on the generated values. They should have a mean and median very close to zero, a standard deviation very close to 1, and quartiles close to `qnorm(.25)` and `qnorm(.75)`:

```
summary(theta)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -4.1850 -0.6745  0.0000  0.0000  0.6745  4.1850

sd(theta)

## [1] 0.9998641

qnorm(.25)

## [1] -0.6744898

qnorm(.75)

## [1] 0.6744898
```

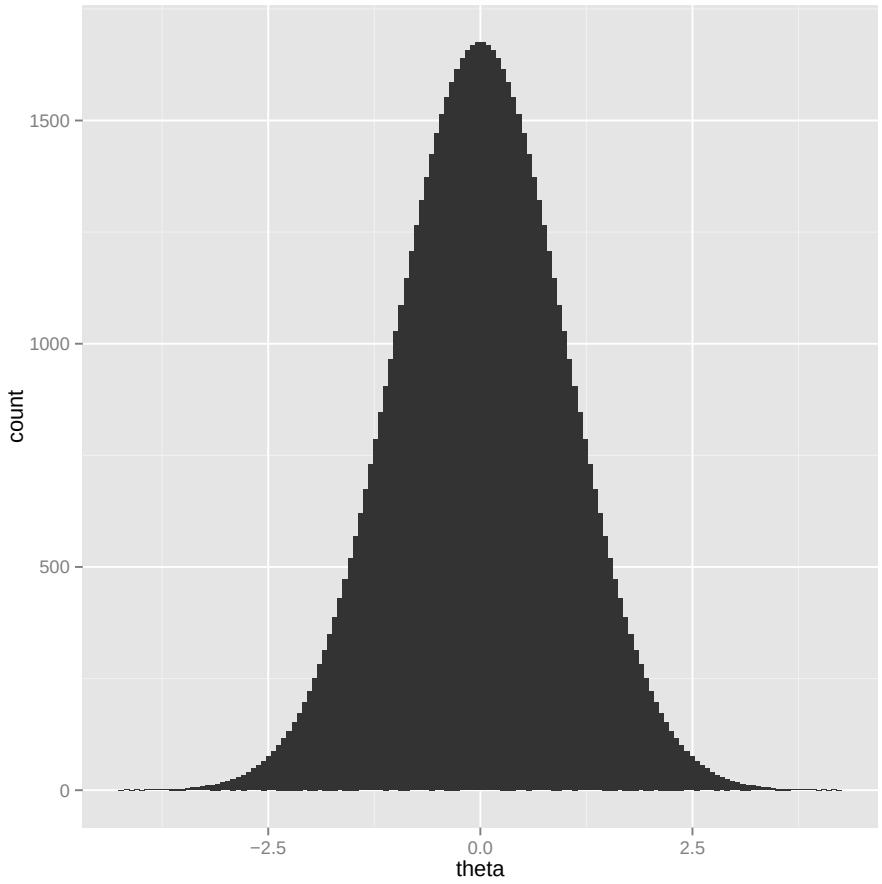
All that remains is to construct a data frame and save it in a file. We can avoid a lot of headaches by sticking with the naming convention we have been using, namely that the data frame is called `thetas` and the column name is `theta`. That way we don't need to change any of our R programs to use this set of θ values. We can call the file anything we want, but again we'll pick something similar to what we have been using: `samplen70.Rdata`. As before, we'll put it in the `Rdata` subdirectory:

```
thetas=data.frame(theta)                    #data frame named thetas with one column, theta
save(thetas,file="Rdata/samplen70.Rdata") #write it into a file called samplen70.Rdata in s
```

We can use `ggplot2` to graph the values, and we should get a nice bell curve. `ggplot2` has a rather cryptic syntax but it produces beautiful graphs if you can figure out how to code them (as usual, you can use your favorite search engine to find examples):

```
library(ggplot2)                                                    #make ggplot2 available
p<-ggplot(thetas,aes(theta))+geom_histogram(binwidth=0.06)    #shamelessly steal this code
p                                                                   #display the plot

## Warning in loop_apply(n, do.ply): position_stack requires constant
width: output may be incorrect
```


1500
1000
500
0
count
−2.5
0.0
2.5
theta