Working with R lists and and vectors

Initial abilities vector and vector of adjustments

Start with a vector of standard normal random variables:

```
N=30
theta=rnorm(N)
theta

##  [1]  0.90805207 -0.68670354 -1.66366200  0.76429616  0.81408520
##  [6] -0.55112782  0.43594958  0.12619570 -1.14718626  0.47101088
## [11]  1.29104086 -1.28832704  0.39049722  0.62466909 -0.18435028
## [16] -2.15690469  0.23971444  0.09871848 -0.69634262 -2.19911687
## [21]  0.16425505  0.59060146  0.18023639 -0.12365305  0.17069830
## [26]  0.24359931  0.67288627  0.83575232 -1.14437950 -0.76567367
```

Now we want to construct a vector of effect size adjustments for each group of three consecutive theta values:

- Increase the first theta by the effect size
- Leave the second theta value of the second unchanges
- Decrease the third theta by the effect size

We will use 0.2 for a small effect size, 0.5 for medium, and 0.8 for large.

```
effect_size=0.2              #small effect size
```

Constructing a list with the concatenation operator

R has a handy constructor for lists which behave like arrays. The syntax is:

`c(item_1,item_2,...,item_n)`

Because R is object-oriented, the items don't have to be numbers or characters, they can be any object. In this case, we will use numbers.

The list we want to construct is `[effect_size,0,-effect_size]`, and the code to do that is:

```
c(effect_size,0,-effect_size)

## [1]  0.2  0.0 -0.2
```

Because we didn't assign it to a variable name, R will just compute it and print it.

Repeating a pattern

Another very useful function in R is the replication function `rep`. The syntax is:

`rep(something,count)`

which produces a list with "something" replicated "count" times. For example,

```
rep(5,7)

## [1] 5 5 5 5 5 5 5
```

Combining the concatenation and repetition functions

We can use the `rep()` and `c()` functions together to generate our vector of effect size adjustments.

We want the same pattern of three values to repeat for a list whose length equals that of `theta`.

The code for this is quite simple:

```
rep(c(effect_size,0,-effect_size),N/3)

##  [1]  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0
## [15] -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2
## [29]  0.0 -0.2
```

One of the nice things about the R language is that you can combine list, vectors, and scalars in a single expression. If an expression contains some scalars and some vectors, R assumes you want to evaluate it for every element of the vector.

This eliminates the need to write tedious and error-prone `for` loops in many cases.

```
adj=rep(c(effect_size,0,-effect_size),N/3)
adj

##  [1]  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0
## [15] -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2  0.0 -0.2  0.2
## [29]  0.0 -0.2
```

Now compute the probabilities of a correct answer for a dichotomous question from its IRT parameters:

```r
a=1.1
b=0.3
c=0.4
D=1.1701
p=c+(1-c)*exp(D*a*(theta-b))/(1+exp(D*a*(theta-b)))
p
```

```
##  [1] 0.8117480 0.5315548 0.4443737 0.7870643 0.7957832 0.5503510 0.7261805
##  [8] 0.6665835 0.4806340 0.7328838 0.8690177 0.4687746 0.7174523 0.7617862
## [15] 0.6094035 0.4243655 0.6883667 0.6613552 0.5302849 0.4231266 0.6738587
## [22] 0.7554603 0.6769233 0.6201750 0.6750936 0.6891157 0.7706410 0.7995230
## [29] 0.4808865 0.5214131
```

Finally compute the adjusted probabilities, where we adjust the ability parameters using the vector of adjustments called `adj`:

```r
padj=c+(1-c)*exp(D*a*(theta+adj-b))/(1+exp(D*a*(theta+adj-b)))
padj
```

```
##  [1] 0.8433160 0.5315548 0.4348883 0.8209718 0.7957832 0.5232363 0.7638686
##  [8] 0.6665835 0.4642945 0.7702989 0.8690177 0.4545857 0.7554409 0.7617862
## [15] 0.5758030 0.4311476 0.6883667 0.6242035 0.5584350 0.4231266 0.6361692
## [22] 0.7916920 0.6769233 0.5856677 0.7136401 0.6891157 0.7332415 0.8323062
## [29] 0.4808865 0.4983754
```