Problem Set 2

Klint Kanopka
EDUC 252L

January 28, 2018

1 Breaking the Classical Test Theory Model

1.1 Coin Flips

Coin flips should not be reliable data - they're random! To look at this a little more analytically:

$$\alpha = \frac{K}{K-1}\left(1 - \frac{\sum_{i=1}^{K} p_i(1-p_i)}{\sigma_X^2}\right)$$

The interesting thing to note here is that the probability of flipping heads is:

$$p_i = 0.5$$

And the variance on the sum of K coin flips will be:

$$\sigma_X = 0.25K$$

Substituting in the formula for Cronbach's Alpha:

$$\alpha = \frac{K}{K-1}\left(1 - \frac{\sum_{i=1}^{K} (0.5)(1-0.5)}{0.25K}\right)$$

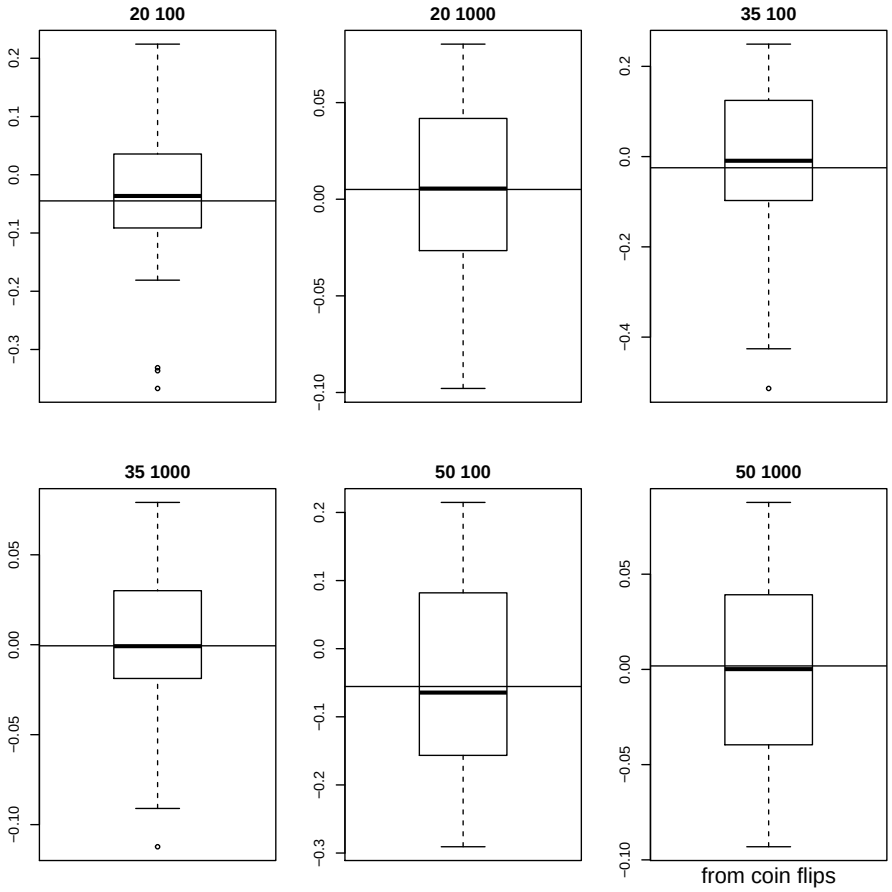
Cleaning up:

$$\alpha = \frac{K}{K-1}\left(1 - \frac{0.25K}{0.25K}\right)$$

$$\alpha = \frac{K}{K-1}(1-1)$$

$$\alpha = 0$$

The expectation, then, is that α shoudl be zero for each situation.



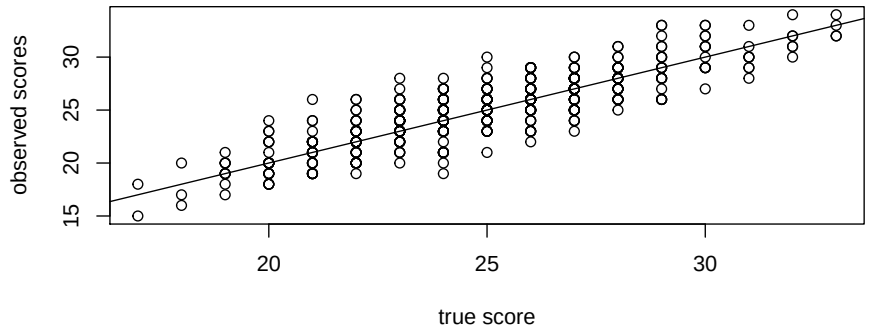
These α plots make sense - they are centered around zero, as predicted.

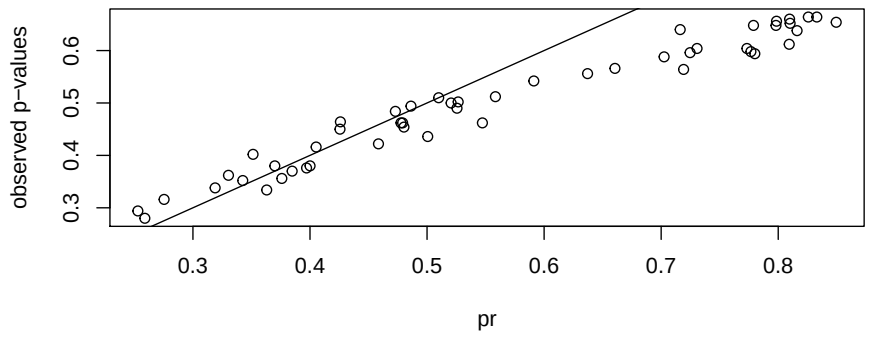
1.2 Simulating Item Response Data

```
##########################################################################################
##now, we were kind of cheating with the coins. we didn't really impose any structure on the

##let's give ourselves a stiffer challenge. let's generate item response data that has
##A. a pre-specified reliability
##B. and yet weird inter-workings such that reliability estimates fall apart completely
##C. [i don't really emphasize this in the below, but you can also pick (roughly) the distr
##below i'm going to do just that using this function.
sim_ctt<-function(N.items, #N.items needs to be even
                  N.ppl,
                  s2.true,
                  s2.error,
```

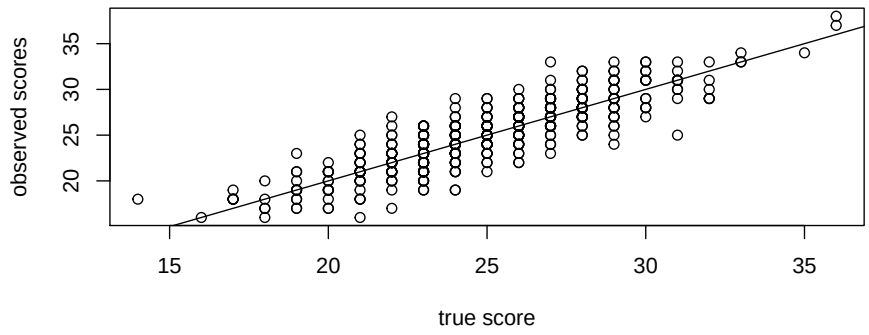
```
                    check=FALSE #this will produce a figure that you can use to check that th
                    ) {
    ##desired reliability
    reliability<-(s2.true)/(s2.error+s2.true)
    ##sample (unobserved) true abilities & errors
    T<-round(rnorm(N.ppl,mean=round(N.items/2),sd=sqrt(s2.true)))
    E<-round(rnorm(N.ppl,mean=0,sd=sqrt(s2.error)))
    O<-T+E
    ifelse(O<0,0,O)->O
    ifelse(O>N.items,N.items,O)->O
    ##note that we already know how many items a person got right (O)
    ##ctt doesn't specify how individual item responses are generated.
    ##so we have carte blance in terms of coming up with item responses. here's how i'm goi
    pr<-runif(N.items,min=.25,max=.85)
    pr.interval<-c(0,cumsum(pr))/sum(pr)
    resp<-list()
    for (i in 1:N.ppl) { #for each person
        resp.i<-rep(0,N.items) #start them with all 0s
        if (O[i]>0) {
            for (j in 1:O[i]) { #now for each of the observed correct responses (for those
                init.val<-1
                while (init.val==1) {
                    index<-cut(runif(1),pr.interval,labels=FALSE) #here i pick the item whe
                    init.val<-resp.i[index]
                }
                resp.i[index]<-1
            }
        }
        resp[[i]]<-resp.i
    }
    do.call("rbind",resp)->resp
    ##note that we are getting both the p-values and the true score/observed score relation
    if (check) {
        par(mfrow=c(2,1))
        plot(T,rowSums(resp),xlab="true score",ylab="observed scores"); abline(0,1)
        plot(pr,colMeans(resp),xlab="pr",ylab="observed p-values"); abline(0,1)
    }
    obs.rel<-cor(T,O)^2 #we can use this to make sure that the true and observed score vari
    print(c(reliability,obs.rel)) ##for real time checking, these are the pre-specified rel
    kr<-kr20(resp)
    c(obs.rel,kr)
}

##to illustrate the basic idea here, we'll generate 10 sets of item response data.
##for each iteration, we'll use the same parameters, these are below.
```

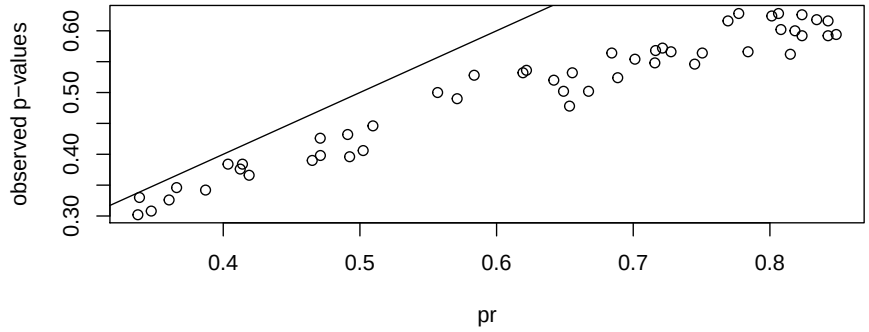
```r
N.items<-50
N.ppl<-500
s2.true<-10
s2.error<-3
alph<-list()
for (i in 1:10) {
    alph[[i]]<-sim_ctt(N.items=N.items,N.ppl=N.ppl,s2.true=s2.true,s2.error=s2.error,check=T
}
```



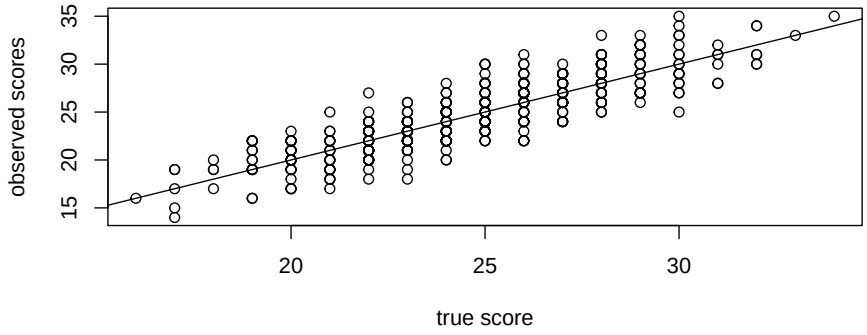


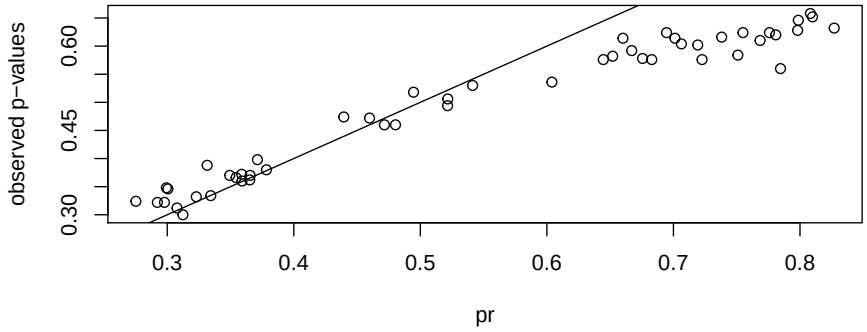

```
## [1] 0.7692308 0.7519898
```



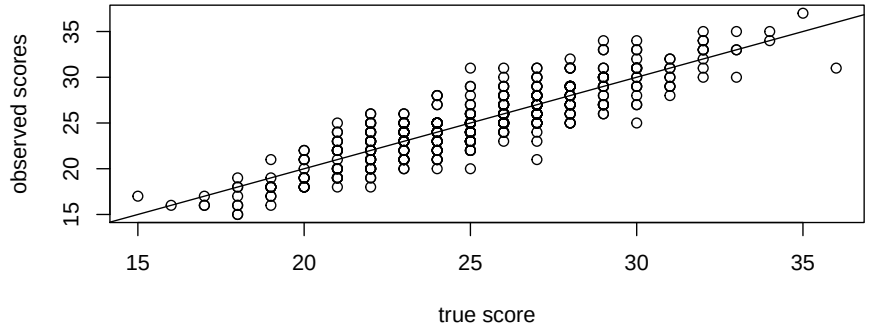


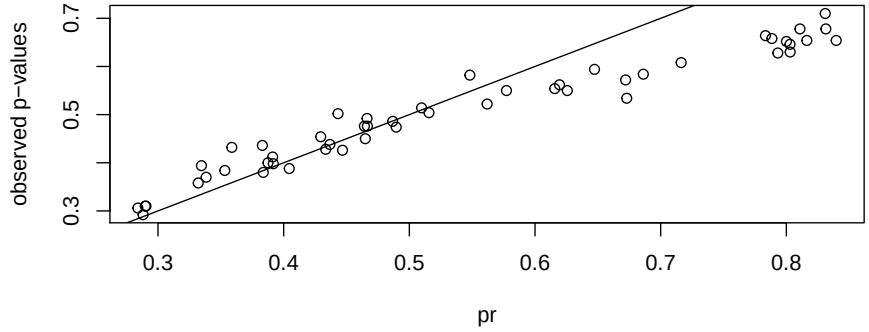

```
## [1] 0.7692308 0.7586454
```



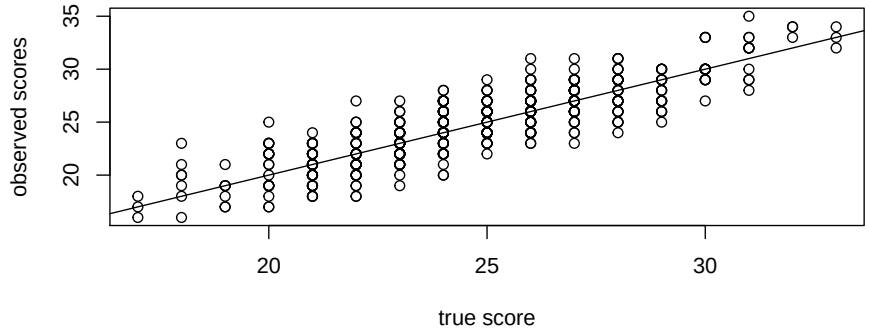


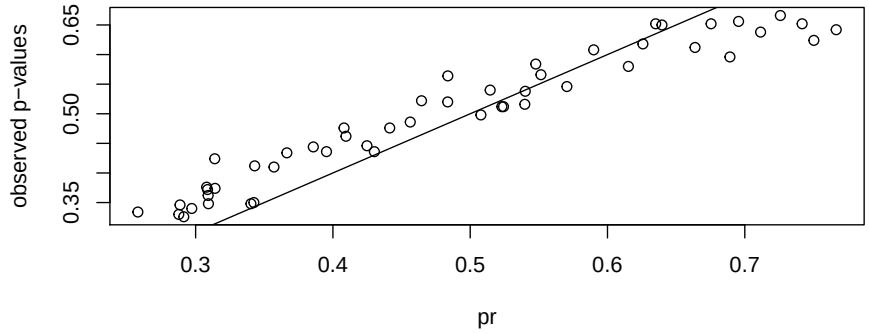

```
## [1] 0.7692308 0.7453799
```



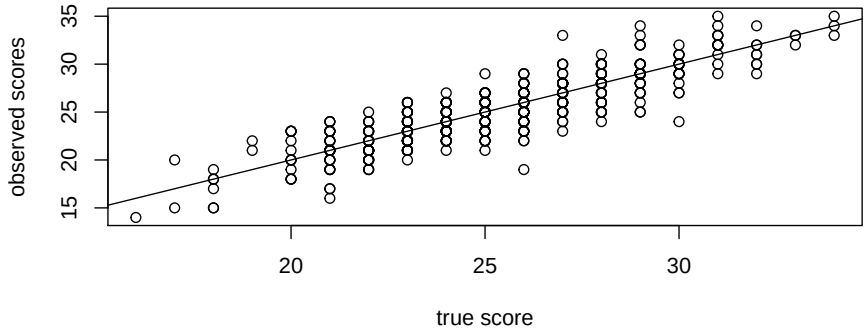


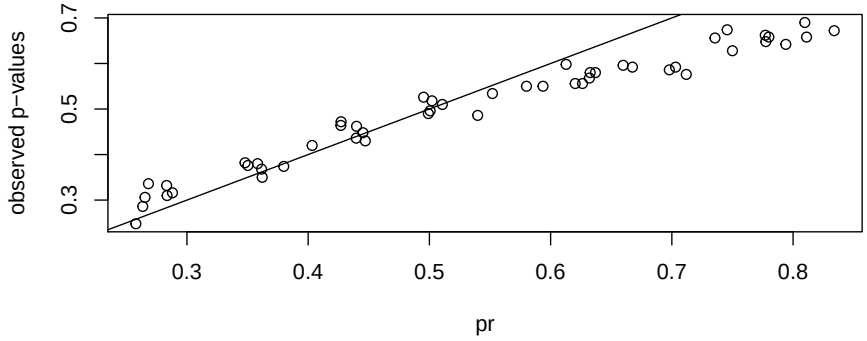

```
## [1] 0.7692308 0.7847733
```



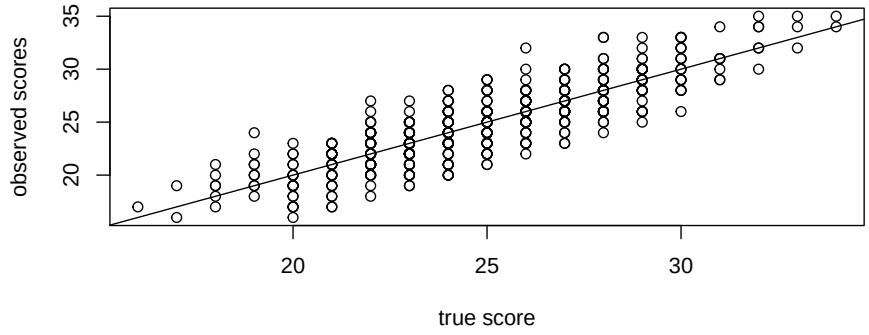


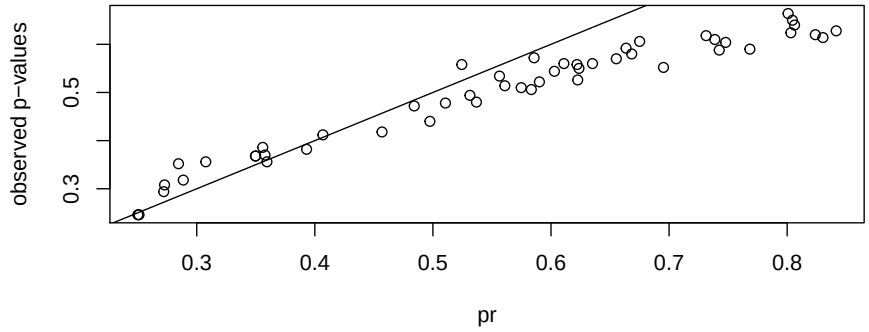

```
## [1] 0.7692308 0.7415703
```



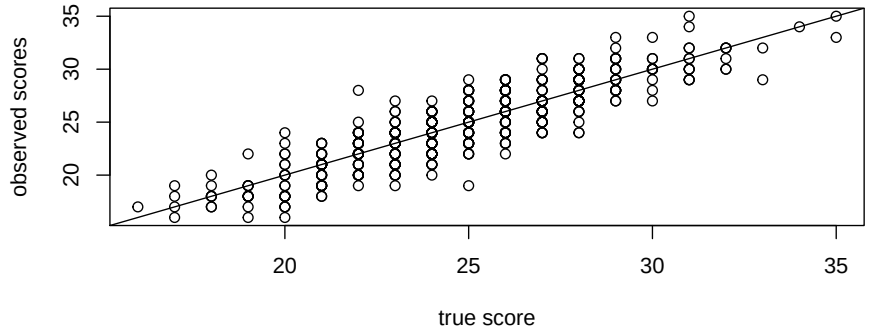


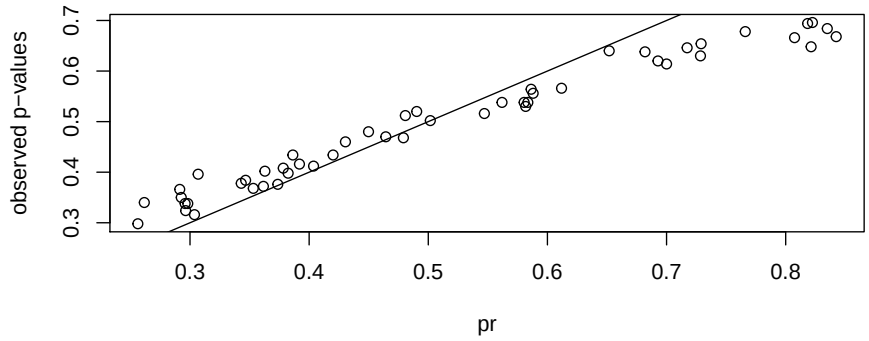

```
## [1] 0.7692308 0.7627290
```



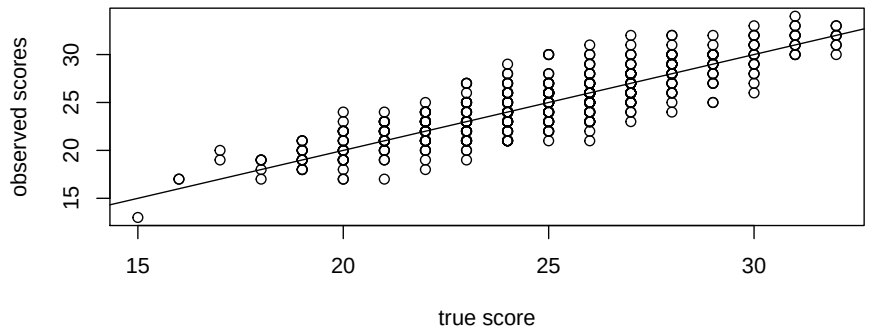


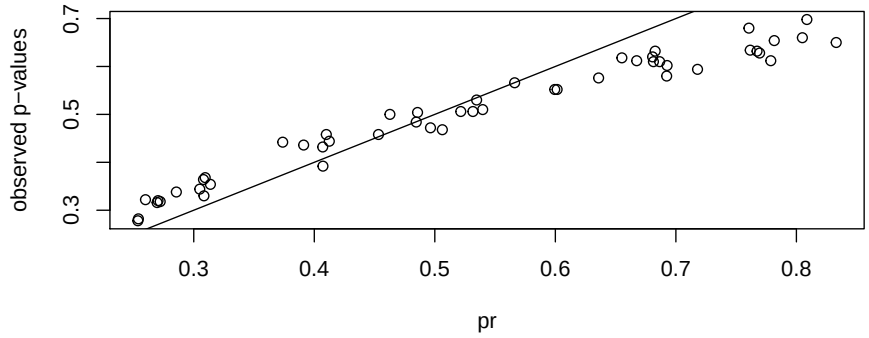

```
## [1] 0.7692308 0.7436331
```



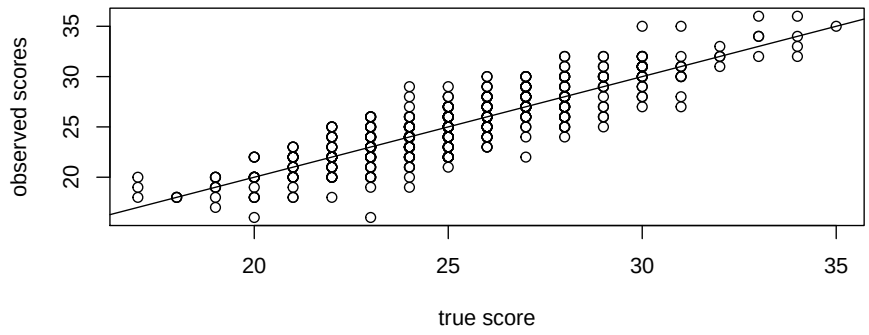


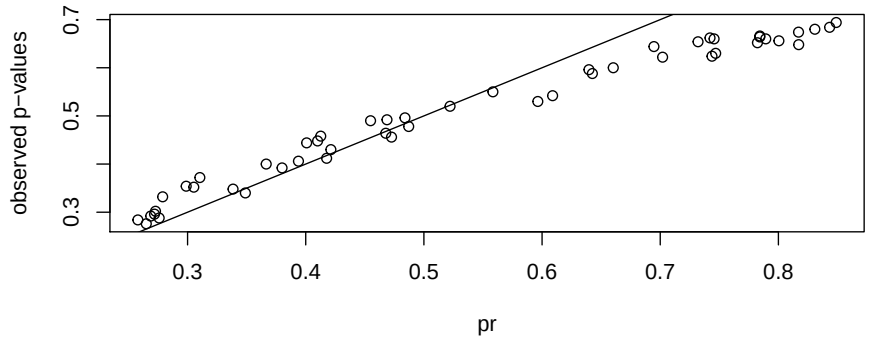

```
## [1] 0.7692308 0.7870532
```



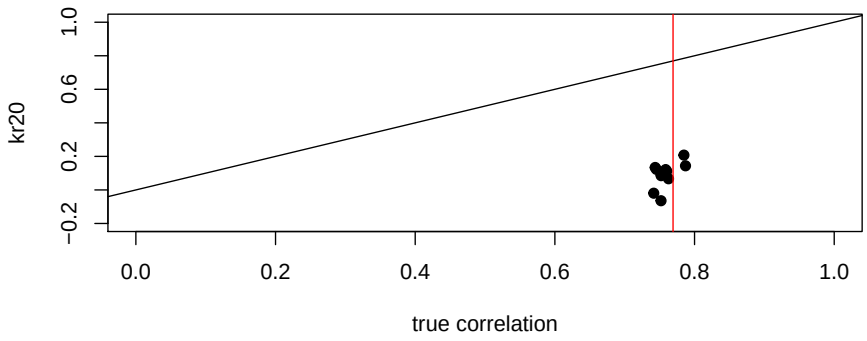



```
## [1] 0.7692308 0.7521453
```





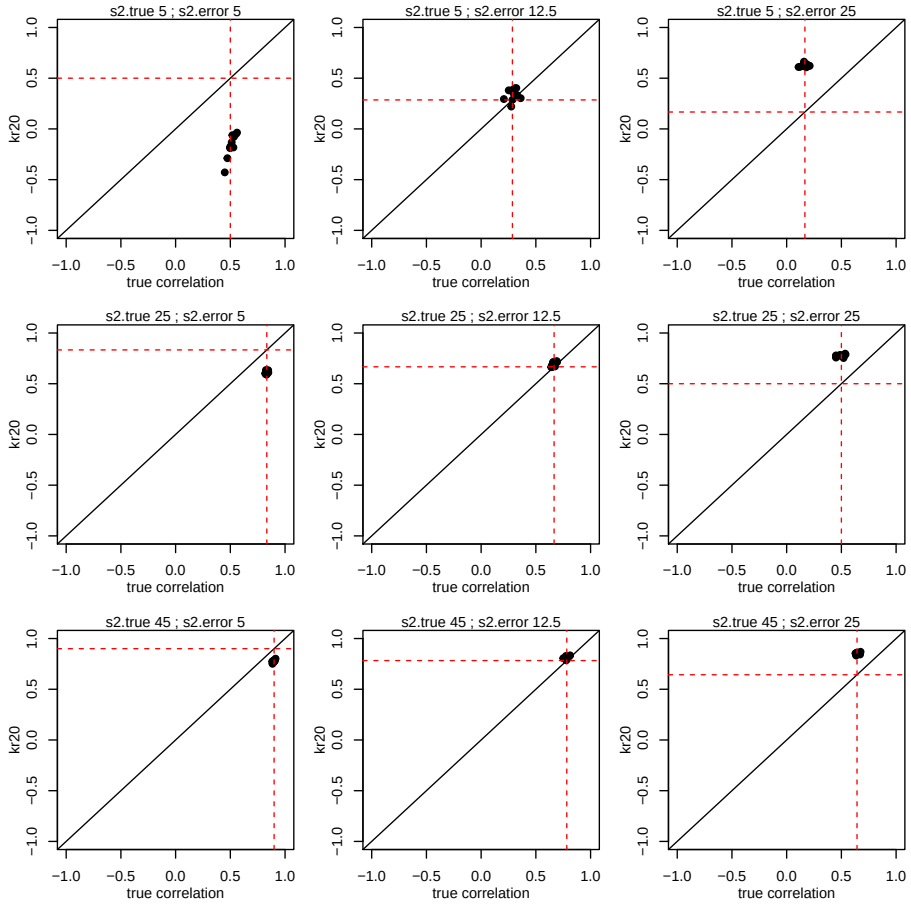

```
## [1] 0.7692308 0.7601565

do.call("rbind",alph)->alph
plot(alph,xlim=c(0,1),ylim=c(-.2,1),pch=19,xlab="true correlation",ylab="kr20"); abline(0,1)
abline(v=s2.true/(s2.true+s2.error),col="red")
##what you can see here is that the item response datsets have the right property when it co
##but, kr20 is producing reliability estimates that are *way* low
##something is going disastrously awry.
##q what is it? make sure you see that the true/error variances are getting handled correctl
##q: any clues as to why this is happening? what is the feature of my data generating mechan

par(mfrow=c(3,3),mgp=c(2,1,0),mar=c(3,3,2,1))
```




```
##now let's look at reliabilities for many different datasets generated by the above wherein
N.ppl<-500
N.items<-50
for (s2.true in N.items*c(.1,.5,.9)) {
    for (s2.error in N.items*c(.1,.25,.5)) {
        alph<-list()
        for (i in 1:10) {
            alph[[i]]<-sim_ctt(N.items=N.items,N.ppl=N.ppl,s2.true=s2.true,s2.error=s2.error
        }
        plot(do.call("rbind",alph),xlim=c(-1,1),ylim=c(-1,1),pch=19,xlab="true correlation",
        mtext(side=3,line=0,paste("s2.true",round(s2.true,2),"; s2.error",round(s2.error,2))
        abline(0,1)
        abline(h=(s2.true)/(s2.error+s2.true),col="red",lty=2)
        abline(v=(s2.true)/(s2.error+s2.true),col="red",lty=2)
    }
}
```

```
## [1] 0.5000000 0.5128796
## [1] 0.5000000 0.5519223
## [1] 0.5000000 0.4977091
## [1] 0.5000000 0.5004587
## [1] 0.5000000 0.5330445
## [1] 0.500000 0.561126
## [1] 0.500000 0.449229
## [1] 0.5000000 0.4732625
## [1] 0.5000000 0.5281862
## [1] 0.5000000 0.5188388
## [1] 0.2857143 0.2960561
## [1] 0.2857143 0.2077476
## [1] 0.2857143 0.2754577
## [1] 0.2857143 0.3299369
## [1] 0.2857143 0.3615758
## [1] 0.2857143 0.2869029
## [1] 0.2857143 0.2856972
## [1] 0.2857143 0.3096434
## [1] 0.2857143 0.3215014
## [1] 0.2857143 0.2522629
## [1] 0.1666667 0.1409280
## [1] 0.1666667 0.1221363
## [1] 0.1666667 0.2104736
## [1] 0.1666667 0.1698770
## [1] 0.1666667 0.1679034
## [1] 0.1666667 0.1089484
## [1] 0.1666667 0.1798455
## [1] 0.1666667 0.2042687
## [1] 0.1666667 0.1583881
## [1] 0.1666667 0.1830265
## [1] 0.8333333 0.8251025
## [1] 0.8333333 0.8256566
## [1] 0.8333333 0.8443499
## [1] 0.8333333 0.8254703
## [1] 0.8333333 0.8473939
## [1] 0.8333333 0.8289805
## [1] 0.8333333 0.8364622
## [1] 0.8333333 0.8457272
## [1] 0.8333333 0.8187131
## [1] 0.8333333 0.8309489
## [1] 0.6666667 0.6557792
## [1] 0.6666667 0.6408120
## [1] 0.6666667 0.6942931
## [1] 0.6666667 0.6449742
## [1] 0.6666667 0.6748290
```

```
## [1] 0.6666667 0.6474504
## [1] 0.6666667 0.6460956
## [1] 0.6666667 0.6579885
## [1] 0.6666667 0.6462138
## [1] 0.6666667 0.6905044
## [1] 0.5000000 0.5198967
## [1] 0.5000000 0.5169372
## [1] 0.5000000 0.4533354
## [1] 0.50000 0.49399
## [1] 0.5000000 0.4506438
## [1] 0.5000000 0.5217988
## [1] 0.5000000 0.5380321
## [1] 0.5000000 0.4873121
## [1] 0.5000000 0.4506171
## [1] 0.500000 0.534523
## [1] 0.9000000 0.8853601
## [1] 0.9000000 0.9063467
## [1] 0.9000000 0.8928527
## [1] 0.9000000 0.8900774
## [1] 0.9000000 0.8951238
## [1] 0.9000000 0.8968953
## [1] 0.9000000 0.8827859
## [1] 0.9000000 0.9042878
## [1] 0.9000000 0.9132436
## [1] 0.9000000 0.9029843
## [1] 0.7826087 0.7955050
## [1] 0.7826087 0.7781482
## [1] 0.7826087 0.7762094
## [1] 0.7826087 0.8135696
## [1] 0.7826087 0.7506937
## [1] 0.7826087 0.8071848
## [1] 0.7826087 0.7963201
## [1] 0.7826087 0.7705015
## [1] 0.7826087 0.7784723
## [1] 0.7826087 0.7781468
## [1] 0.6428571 0.6429260
## [1] 0.6428571 0.6569876
## [1] 0.6428571 0.6295471
## [1] 0.6428571 0.6312202
## [1] 0.6428571 0.6329505
## [1] 0.6428571 0.6556899
## [1] 0.6428571 0.6709553
## [1] 0.6428571 0.6345450
## [1] 0.6428571 0.6440468
## [1] 0.6428571 0.6752339
```




```
##q. how does the kr20 estimate of reliability behave as a function of the true and error v
##q. what does this make you think of the CTT model?
```