Runs Test Paper

Lindesay Scott-Hayward

2016-07-27

Contents

Simulated Data

Make some simulated toy data using the following equation:

$$y = \beta_0 \ + \ \beta_1 X_1$$

where $\beta_0 = 1$ and $\beta_1 = 0.3$

```r
dat<-makeToyData(200, length.panels=5)
head(dat)
```

```
         x evph       mu panelid
1 1.000000    0 3.669297       1
2 1.090909    0 3.770746       1
3 1.181818    0 3.874999       1
4 1.272727    0 3.982135       1
5 1.363636    0 4.092234       1
6 1.454545    0 4.205376       2
```

Uncorrelated data

1000 sets of noisy data are simulated from this truth using a Poisson distribution.

```r
newdat<-generateNoise(nsim, dat$mu, family='poisson', d=1)
```

To test, fit a glm to one of the sets of data.

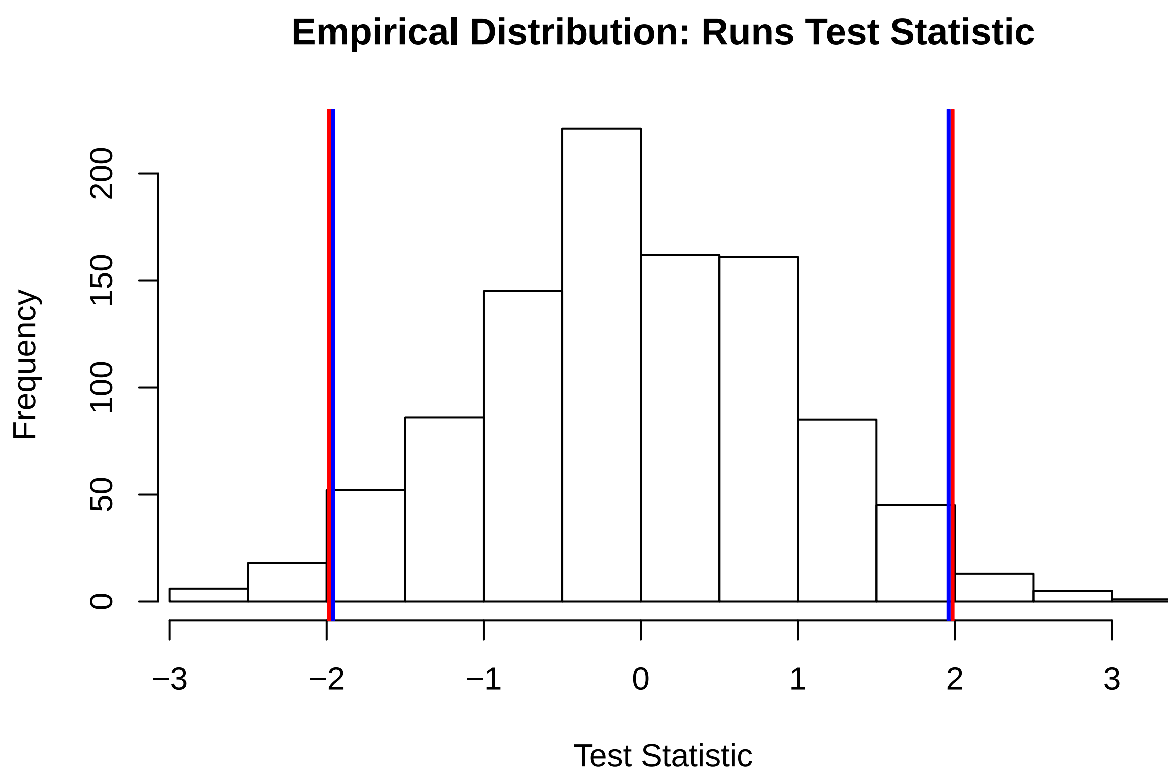
```r
init_glm<-glm(newdat[,1] ~ x, data=dat, family='quasipoisson')
summary(init_glm)
```

```

Call:
glm(formula = newdat[, 1] ~ x, family = "quasipoisson", data = dat)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-3.13421  -0.69426   0.02305   0.57711   2.06863

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.956845   0.057000   16.79   <2e-16 ***
x           0.306386   0.007401   41.40   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 1.013624)

    Null deviance: 2270.5  on 199  degrees of freedom
Residual deviance:  204.9  on 198  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 4
```

Models were fitted to all datasets generated above and the runs test was evaluated for each one. The distribution of the test statistics is returned. A plot is also produced, which shows the lower 2.5% and upper 97.5% critical values of the empirical distribution (red) and from the Normal ($N(0, 1)$) distribution.

```r
empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                 model = init_glm, data = dat, plot=TRUE,
                                 returnDist = TRUE)
```

```
.....................................................................................................
.....................................................................................................
```



Use the data generated to assess the 5% error rate for this test when using the Normal distribition or the empirical distribution generated above.

```
newdat<-generateNoise(nsim, dat$mu, family='poisson', d=1)
ps<-matrix(NA, nrow=nsim, ncol=2)

for(i in 1:nsim){
  sim_glm<-glm(newdat[,i] ~ x , data=dat, family='quasipoisson')
  # find both the empirical and Normal p-values
  ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
  ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
}
```

```
(length(which(ps[,1]<0.05))/nsim)*100
```

```
[1] 6.9
```

```
(length(which(ps[,2]<0.05))/nsim)*100
```

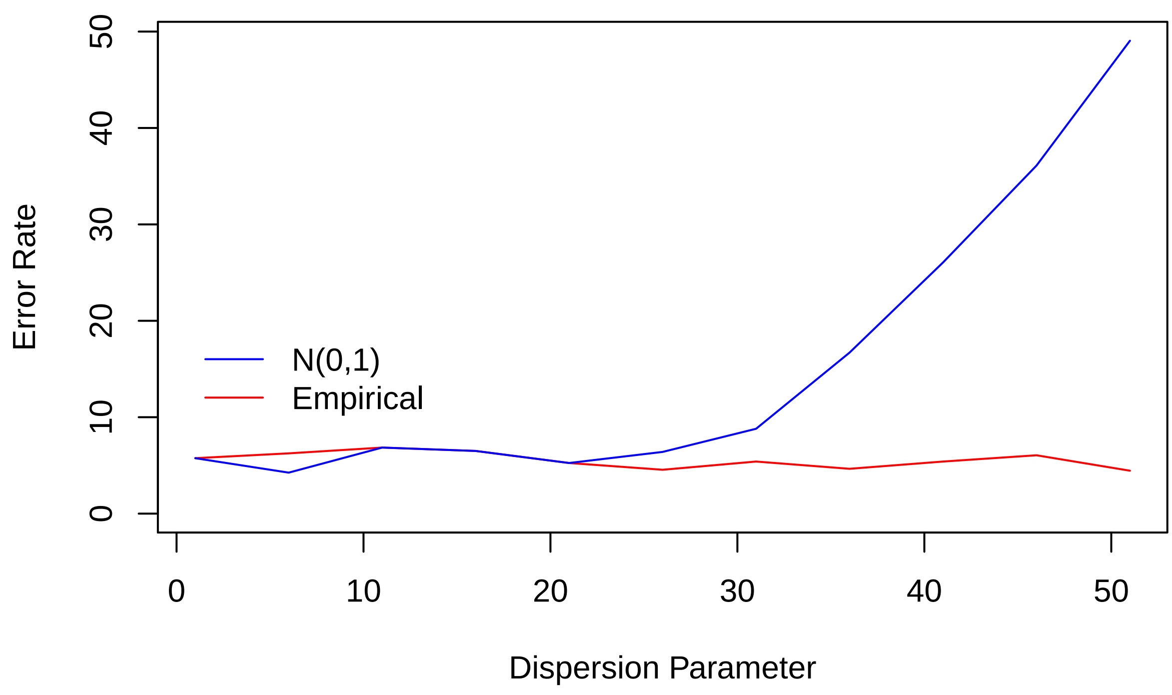
```
[1] 6.9
```

How does this change if we vary the dispersion parameter for the data?

```
nphi=seq(1,51, by=5)
errrate<-matrix(NA, nrow=length(nphi), ncol=2)
counter=1
nsim=2000
for(p in nphi){
  newdat<-generateNoise(nsim, dat$mu, family='poisson', d=p)
```

```r
  # for each change in phi, update the empirical distribution
  empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                   model = init_glm, data = dat, plot=FALSE,
                                   returnDist = TRUE, dots=FALSE)

  ps<-matrix(NA, nrow=nsim, ncol=2)
  for(i in 1:nsim){
    sim_glm<-glm(newdat[,i] ~ x + as.factor(evph), data=dat, family='quasipoisson')
    # find both the empirical and Normal p-values
    ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
    ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
  }

  errrate[counter,1]<-(length(which(ps[,1]<0.05))/nsim)*100
  errrate[counter,2]<-(length(which(ps[,2]<0.05))/nsim)*100
  counter=counter+1
}
```

```r
plot(nphi, errrate[,1], type='l', col='red', ylim=c(0, range(errrate)[2]), ylab='Error Rate', xlab='Dis
lines(nphi, errrate[,2], col='blue')
legend(0, 20, legend = c('N(0,1)', 'Empirical'), col = c('blue', 'red'), lty = c(1,1), bty = 'n')
```



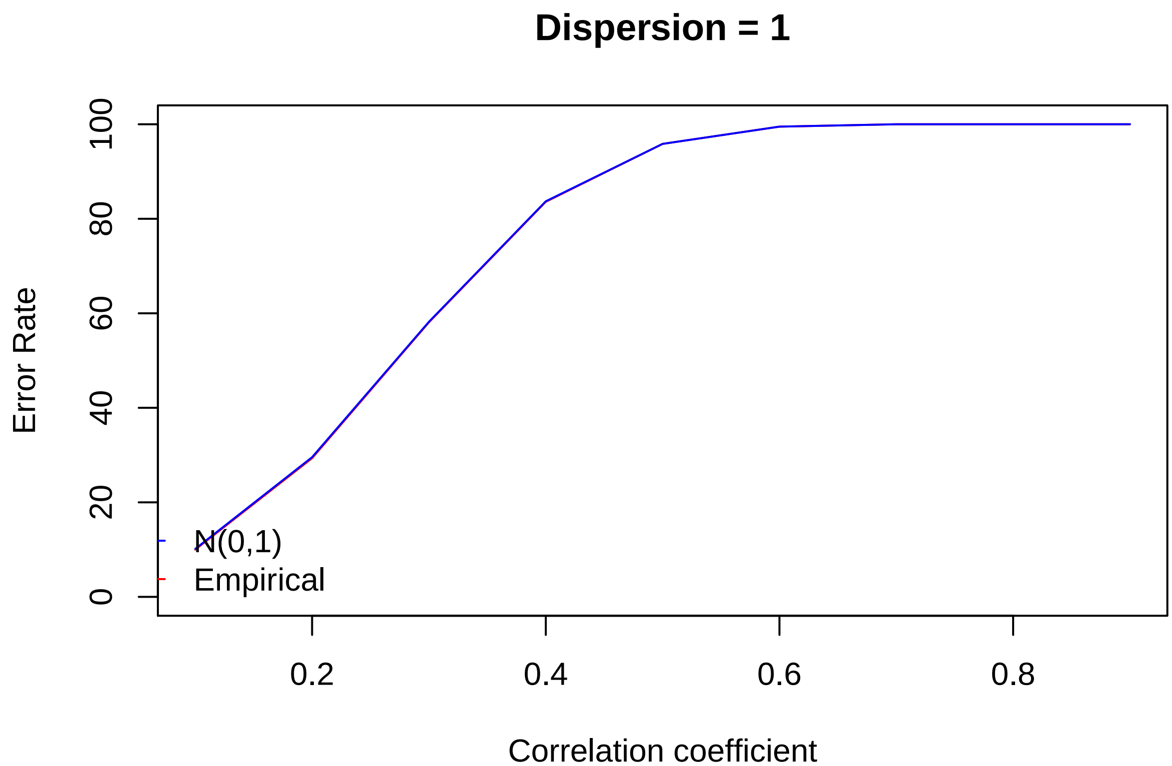
Correlated data

```
dat<-makeToyData(200, length.panels=5)
head(dat)
```

```
         x evph       mu panelid
1 1.000000    0 3.669297       1
2 1.090909    0 3.770746       1
3 1.181818    0 3.874999       1
4 1.272727    0 3.982135       1
5 1.363636    0 4.092234       1
6 1.454545    0 4.205376       2
```

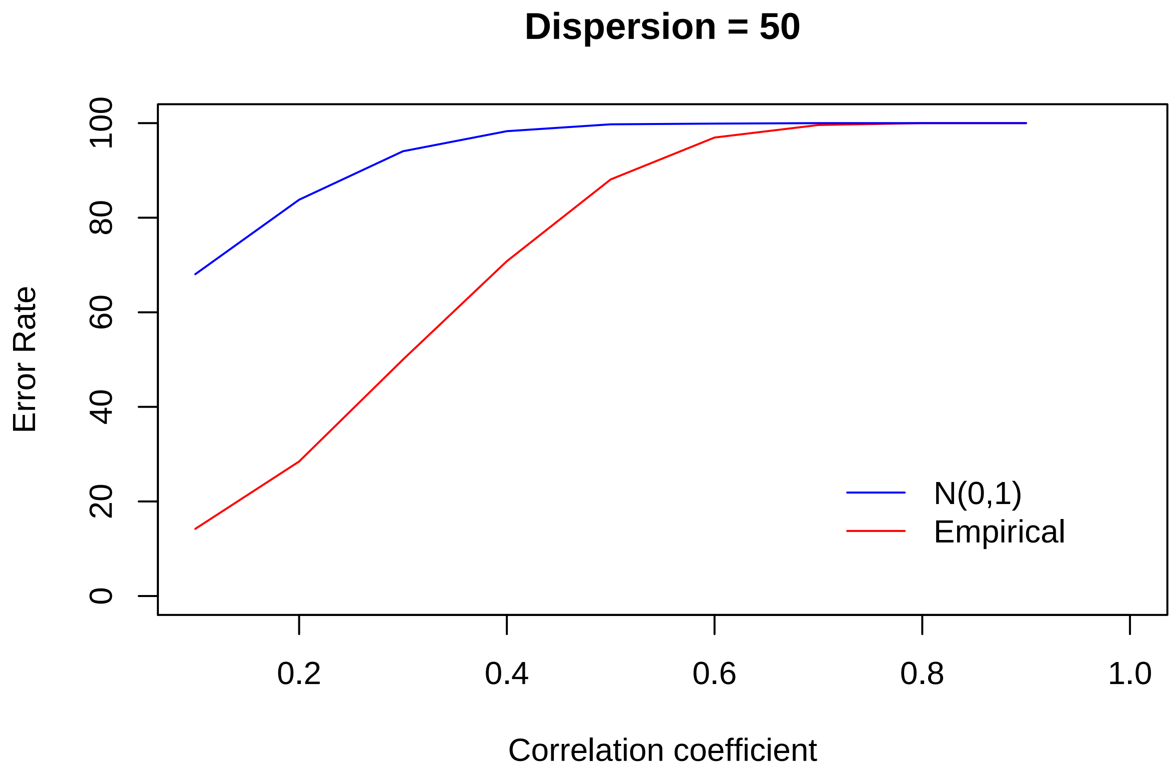
```
nsim=2000
newdat<-generateNoise(nsim, dat$mu, family='poisson', d=1)
rho=seq(0.1, 0.9, by=0.1)
errrate<-matrix(NA, nrow=length(rho), ncol=2)
counter=1

for(r in rho){
  newdatcorr<-generateIC.toy(dat, c(1, r, r^2, r^3, r^4), 'panelid', newdat, ncol(newdat), dots=FALSE)

  # for each change in phi, update the empirical distribution
  empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                   model = init_glm, data = dat, plot=FALSE,
                                   returnDist = TRUE, dots=FALSE)

  ps<-matrix(NA, nrow=nsim, ncol=2)
  for(i in 1:nsim){
    sim_glm<-glm(newdatcorr[,i] ~ x , data=dat, family='quasipoisson')
    # find both the empirical and Normal p-values
    ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
    ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
  }

  errrate[counter,1]<-(length(which(ps[,1]<0.05))/nsim)*100
  errrate[counter,2]<-(length(which(ps[,2]<0.05))/nsim)*100
  counter=counter+1
}
```

```
plot(rho, errrate[,1], type='l', col='red', ylim=c(0, range(errrate)[2]), ylab='Error Rate', xlab='Corr
lines(rho, errrate[,2], col='blue')
legend(0, 20, legend = c('N(0,1)', 'Empirical'), col = c('blue', 'red'), lty = c(1,1), bty = 'n')
```




```
nsim=2000
newdat<-generateNoise(nsim, dat$mu, family='poisson', d=50)
rho=seq(0.1, 0.9, by=0.1)
errrate<-matrix(NA, nrow=length(rho), ncol=2)
counter=1

for(r in rho){
  newdatcorr<-generateIC.toy(dat, c(1, r, r^2, r^3, r^4), 'panelid', newdat, ncol(newdat), dots=FALSE)

  # for each change in phi, update the empirical distribution
  empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                   model = init_glm, data = dat, plot=FALSE,
                                   returnDist = TRUE, dots=FALSE)

  ps<-matrix(NA, nrow=nsim, ncol=2)
  for(i in 1:nsim){
    sim_glm<-glm(newdatcorr[,i] ~ x , data=dat, family='quasipoisson')
    # find both the empirical and Normal p-values
    ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
    ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
  }

  errrate[counter,1]<-(length(which(ps[,1]<0.05))/nsim)*100
  errrate[counter,2]<-(length(which(ps[,2]<0.05))/nsim)*100
  counter=counter+1
}
```

```
plot(rho, errrate[,1], type='l', col='red', ylim=c(0, range(errrate)[2]), xlim=c(0.1,1), ylab='Error Ra
lines(rho, errrate[,2], col='blue')
legend(0.7, 30, legend = c('N(0,1)', 'Empirical'), col = c('blue', 'red'), lty = c(1,1), bty = 'n')
```



As expected, when the dispersion in the data is high, there is a greater discrepancy between the N(0,1) and empirical distributions.

What does all this mean for power analysis?

If data are assumed to be correlated, then a correlation structure is chosen and GEE - type model fitted. If the independent working correlation structure is specified, then robust standard errors are used for inference. Does this machinery still work when there is actually no correlation present in the data?

Question:

1. Are robust standard errors the same as raw standard errors when no correlation is present?
2. What happens when a blocking structure is chosen that is not necessary. i.e. does the answer to question 1 vary with differing block structures?
3. Is the power to detect change affected by using robust standard errors when not necessary?

Make data but add an impact. Here we repeat the data but have a 50% decline in the response for the second set.

50% decline, $\phi = 50$

```
datimp<-makeToyData(200, length.panels=5, changecoef.link = log(0.5))
head(datimp)
```

```
         x evph       mu panelid
1 1.000000    0 3.669297       1
2 1.090909    0 3.770746       1
3 1.181818    0 3.874999       1
4 1.272727    0 3.982135       1
5 1.363636    0 4.092234       1
6 1.454545    0 4.205376       2
```

Generate independent noisy data with a high dispersion, e.g. $\phi = 50$

```
nsim=2000
newdat<-generateNoise(nsim, datimp$mu, family='poisson', d=50)
```

```
init_impglm<-glm(newdat[,1] ~ x + as.factor(evph), data=datimp, family='quasipoisson')
summary(init_impglm)
```

```

Call:
glm(formula = newdat[, 1] ~ x + as.factor(evph), family = "quasipoisson",
    data = datimp)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-10.6337   -4.4825   -2.7733   -0.0955   26.7367

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)        0.9116     0.4549   2.004  0.04643 *
x                  0.3567     0.0567   6.290 2.01e-09 ***
as.factor(evph)1  -0.7526     0.2539  -2.964  0.00341 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 54.75764)

    Null deviance: 9692.4  on 199  degrees of freedom
Residual deviance: 6460.4  on 197  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 7
```

Get the empirical distribution critical values:

```
empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                 model = init_impglm, data = datimp, plot=FALSE,
                                 returnDist = TRUE, dots=FALSE)
```

```
runspowsim_out<-runsPowerSim(newdat, init_impglm, empdistribution, nsim, powercoefid = 3)
```

```
apply(runspowsim_out$rawrob, 2, sum)/nsim
```

```
   Emp   Norm
```

```
0.0665 0.8280
```

```
(length(which(runspowsim_out$imppvals[,1]<=0.05))/nsim)*100
```

```
[1] 73.1
```

```
(length(which(runspowsim_out$imppvals[,2]<=0.05))/nsim)*100
```

```
[1] 73.85
```

25% decline, $\phi = 50$

```
  Emp  Norm
0.066 0.609
```

```
[1] 23.45
```

```
[1] 26
```

0% decline, $\phi = 50$

```
   Emp   Norm
0.0510 0.4485
```

```
[1] 5.15
```

```
[1] 5.85
```

It seems that when the Normal distribution is used to test for correlation, consistently the power to detect change is higher than when the empirical distribution is used to choose which standard errors to use.

Test: Repeat for 50% decline to get a range of power values. Check to see if the discrepancy is consistent.

50% decline, $\phi = 50$, multiruns

```
wilcox.test(x = runspower[,1], runspower[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x = power[,1], power[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 12, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```
apply(power, 2, mean)
```

```
[1] 72.9140 74.0305
```

```r
apply(runspower, 2, mean)
```

```
[1] 0.059565 0.834885
```

Statistically we have a significant difference between the power calculated from raw/robust s.e. but the real world significance in this toy example is minimal. The mean difference is one percentage point (with raw giving a higher power).

25% decline, $\phi = 50$, multiruns

```

    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0


    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```
[1] 23.6045 25.3685
```

```
[1] 0.057965 0.620265
```

Toy data with smaller mean

The Falls of Warness data has a very small mean (1.01) compared with the toy data used so far. Here we reduce the mean of the toy data inline with that seen in the Falls of Warness data to see if we can replicate results

```r
datlow<-makeToyData(200, length.panels=5, b0=-2)
head(datlow)
```

```
         x evph        mu panelid
1 1.000000    0 0.1826835       1
2 1.090909    0 0.1877344       1
3 1.181818    0 0.1929248       1
4 1.272727    0 0.1982588       1
5 1.363636    0 0.2037403       1
6 1.454545    0 0.2093733       2
```

```r
mean(datlow$mu)
```

```
[1] 0.9442818
```

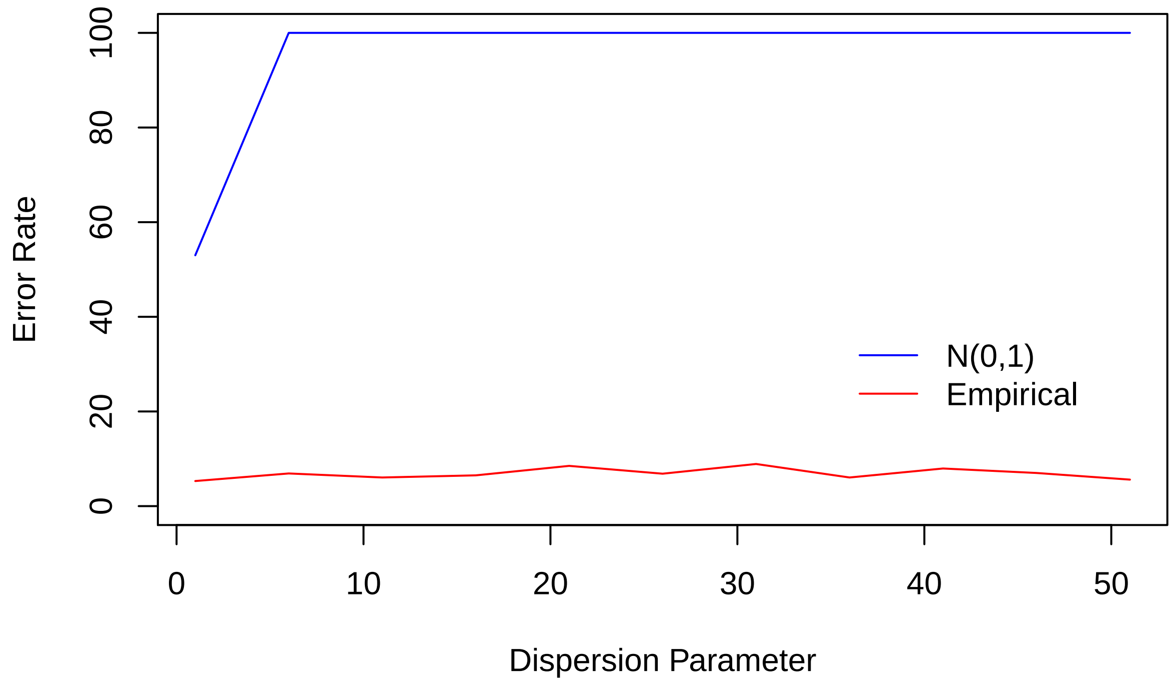
```r
init_glm<-glm(newdat[,1] ~ x, data=datlow, family='quasipoisson')
summary(init_glm)
```

```


Call:
glm(formula = newdat[, 1] ~ x, family = "quasipoisson", data = datlow)

Deviance Residuals:
```

```
    Min       1Q   Median       3Q      Max
-8.1861  -4.3997  -3.1320  -0.0988  26.8324

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  1.27496    0.42226   3.019  0.00287 **
x            0.22782    0.05738   3.971  0.00010 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 55.17751)

    Null deviance: 6956.7  on 199  degrees of freedom
Residual deviance: 5994.9  on 198  degrees of freedom
AIC: NA

Number of Fisher Scoring iterations: 6
```

```
nphi=seq(1,51, by=5)
errrate<-matrix(NA, nrow=length(nphi), ncol=2)
counter=1
nsim=2000
for(p in nphi){
  newdat<-generateNoise(nsim, datlow$mu, family='poisson', d=p)
  # for each change in phi, update the empirical distribution
  empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                  model = init_glm, data = datlow, plot=FALSE,
                                  returnDist = TRUE, dots=FALSE)

  ps<-matrix(NA, nrow=nsim, ncol=2)
  for(i in 1:nsim){
    sim_glm<-glm(newdat[,i] ~ x + as.factor(evph), data=datlow,
                 family='quasipoisson')
    # find both the empirical and Normal p-values
    ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
    ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
  }

  errrate[counter,1]<-(length(which(ps[,1]<0.05))/nsim)*100
  errrate[counter,2]<-(length(which(ps[,2]<0.05))/nsim)*100
  counter=counter+1
}
```

```
plot(nphi, errrate[,1], type='l', col='red', ylim=c(0, range(errrate)[2]), ylab='Error Rate', xlab='Dis
lines(nphi, errrate[,2], col='blue')
legend(35, 40, legend = c('N(0,1)', 'Empirical'), col = c('blue', 'red'), lty = c(1,1), bty = 'n')
```



25% decline, $\phi = 50$, multiruns

```
    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0

    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```
[1]  9.2585 22.5115
```

```
[1] 0.078355 1.000000
```

25% decline, $\phi = 1$, multiruns

```
    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
```

```
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0


    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0

[1] 44.5755 47.3495

[1] 0.056925 0.695025
```

Real Data: Falls of Warness

```
init_glm<-glm(response ~  TideState + WindStrength + SeaState + SimpPrecipitation + CloudCover, data=dat
```

```
nsim=500
newdat<-generateNoise(nsim, fitted(init_glm), family='poisson', d=1)
```

500 sets of noisy data are simulated from this truth using a Poisson distribution.

To test, fit a glm to one of the sets of data.

```
fowsim_glm<-glm(newdat[,1] ~  TideState + WindStrength + SeaState + SimpPrecipitation + CloudCover, data
summary(fowsim_glm)
```

```
Call:
glm(formula = newdat[, 1] ~ TideState + WindStrength + SeaState +
    SimpPrecipitation + CloudCover, family = quasipoisson, data = dat)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6583  -1.2134  -0.1304   0.5300   3.9991

Coefficients:
                          Estimate Std. Error t value Pr(>|t|)
(Intercept)              -0.758034   0.056030 -13.529  < 2e-16 ***
TideState                -0.118847   0.008593 -13.831  < 2e-16 ***
WindStrength             -0.355634   0.012438 -28.593  < 2e-16 ***
SeaState                  0.394730   0.012600  31.327  < 2e-16 ***
SimpPrecipitationHEAVY    0.167996   0.084613   1.985   0.0471 *
SimpPrecipitationLIGHT    0.880236   0.053269  16.524  < 2e-16 ***
SimpPrecipitationNONE     0.311551   0.048875   6.374 1.87e-10 ***
SimpPrecipitationSHOWERS  0.846511   0.049402  17.135  < 2e-16 ***
CloudCover                0.102456   0.002971  34.484  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 1.003368)

    Null deviance: 34848  on 26334  degrees of freedom
Residual deviance: 29566  on 26326  degrees of freedom
```

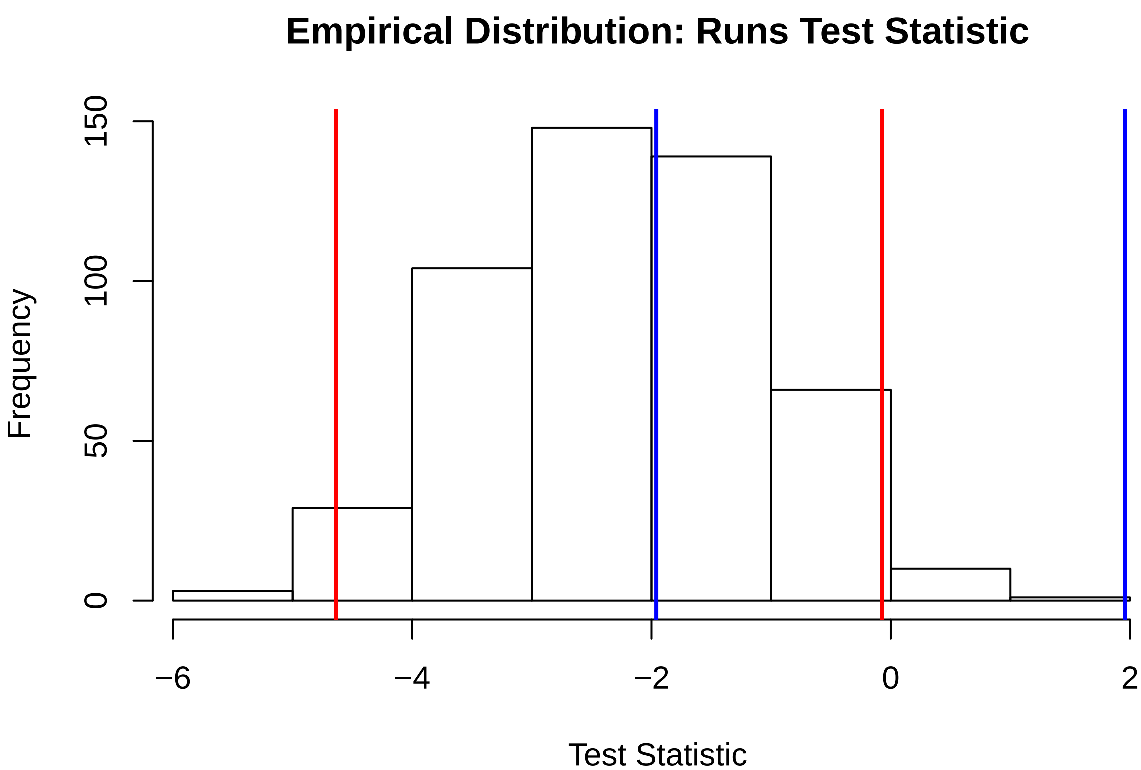
```
AIC: NA

Number of Fisher Scoring iterations: 5
```

Runs Test Check

Models were fitted to all datasets generated above and the runs test was evaluated for each one. The distribution of the test statistics is returned. A plot is also produced, which shows the lower 2.5% and upper 97.5% critical values of the empirical distribution (red) and from the Normal ($N(0, 1)$) distribution.

```
empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                 model = fowsim_glm, data = dat, plot=TRUE,
                                 returnDist = TRUE)
```

```
....................................................................................................
```



Use the data generated to assess the 5% error rate for this test when using the Normal distribition or the empirical distribution generated above.

```
newdat<-generateNoise(nsim, fitted(init_glm), family='poisson', d=1)
ps<-matrix(NA, nrow=nsim, ncol=2)

for(i in 1:nsim){
  sim_glm<-update(fowsim_glm, newdat[,i]~.)
  # find both the empirical and Normal p-values
  ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
  ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
```

```
}
```

```
(length(which(ps[,1]<0.05))/nsim)*100
```

```
[1] 5.6
```

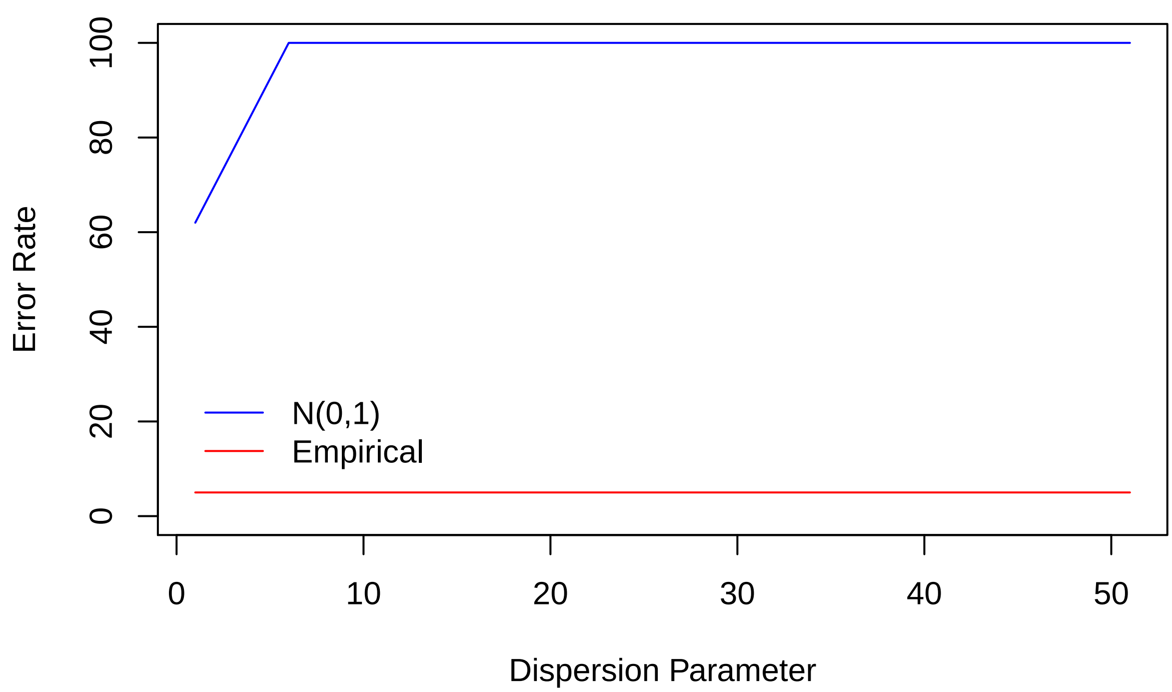
```
(length(which(ps[,2]<0.05))/nsim)*100
```

```
[1] 62.6
```

How does this change if we vary the dispersion parameter for the data?

```
nphi=seq(1,51, by=5)
errrate<-matrix(NA, nrow=length(nphi), ncol=2)
counter=1
nsim=500
for(p in nphi){
  newdat<-generateNoise(nsim, fitted(init_glm), family='poisson', d=p)
  # for each change in phi, update the empirical distribution
  empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                   model = init_glm, data = dat, plot=FALSE,
                                   returnDist = TRUE, dots=FALSE)

  ps<-matrix(NA, nrow=nsim, ncol=2)
  for(i in 1:nsim){
    sim_glm<-update(fowsim_glm, newdat[,i]~.)
    # find both the empirical and Normal p-values
    ps[i,1]<-runs.test(residuals(sim_glm, type='pearson'), critvals = empdistribution)$p.value
    ps[i,2]<-runs.test(residuals(sim_glm, type='pearson'))$p.value
  }

  errrate[counter,1]<-(length(which(ps[,1]<0.05))/nsim)*100
  errrate[counter,2]<-(length(which(ps[,2]<0.05))/nsim)*100
  counter=counter+1
}
```

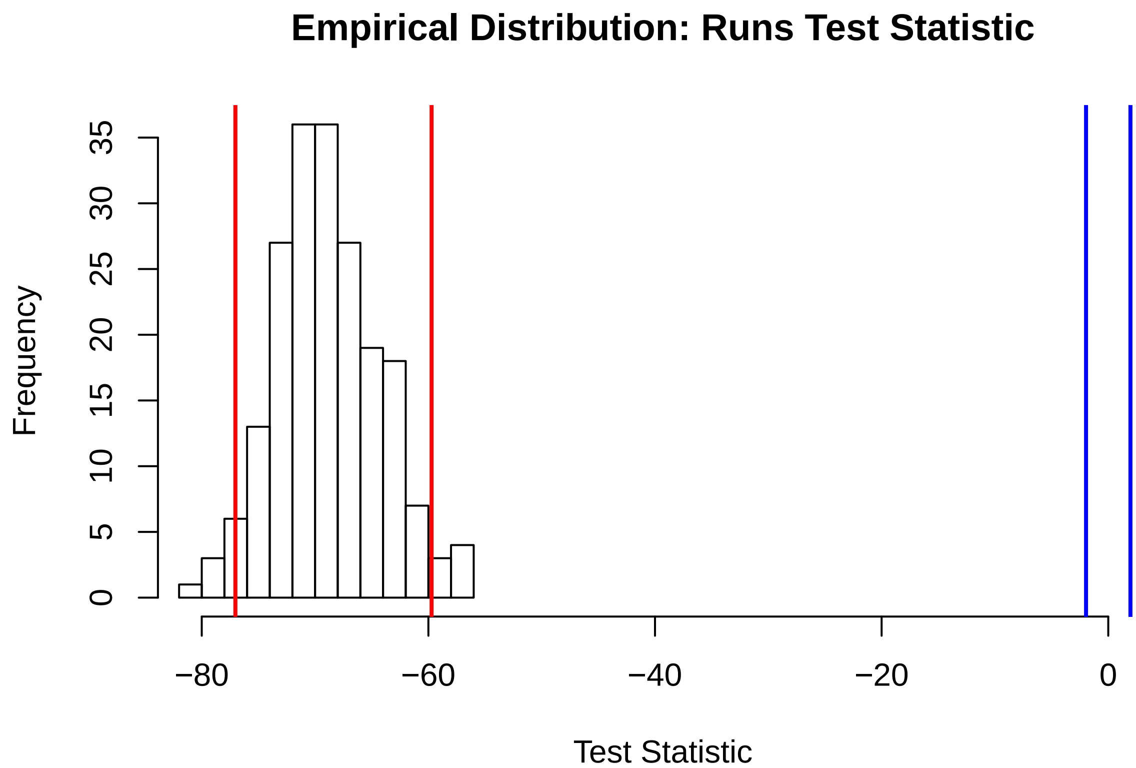
```
plot(nphi, errrate[,1], type='l', col='red', ylim=c(0, range(errrate)[2]), ylab='Error Rate', xlab='Disp
lines(nphi, errrate[,2], col='blue')
legend(0, 30, legend = c('N(0,1)', 'Empirical'), col = c('blue', 'red'), lty = c(1,1), bty = 'n')
```



Asessing the effect on Power

0% decline, dispersion = 50

```
nsim=200
impdata<-genOverallchangeData(log(1), init_glm, data = dat)
newdat<-generateNoise(nsim, impdata$truth, family='poisson', d=50)
fowsim_glm<-glm(newdat[,1] ~  TideState + WindStrength + SeaState + SimpPrecipitation + CloudCover + as
empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                 model = fowsim_glm, data = impdata, plot=TRUE,
                                 returnDist = TRUE, dots = FALSE)
```




```
runspowsim_out<-runsPowerSim(newdat, fowsim_glm, empdistribution, nsim, powercoefid = length(coef(fowsi
```

```
apply(runspowsim_out$rawrob, 2, sum)/nsim
```

```
  Emp  Norm
0.055 1.000
```

```
(length(which(runspowsim_out$imppvals[,1]<=0.05))/nsim)*100
```

```
[1] 5.5
```

```
(length(which(runspowsim_out$imppvals[,2]<=0.05))/nsim)*100
```

```
[1] 5.5
```

Multiple runs of power test: 0% decline, dispersion=50

```
wilcox.test(x = runspower[,1], runspower[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(x = power[,1], power[,2], paired = T)
```

```
    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 372, p-value = 0.00302
alternative hypothesis: true location shift is not equal to 0
```

```r
apply(power, 2, mean)
```

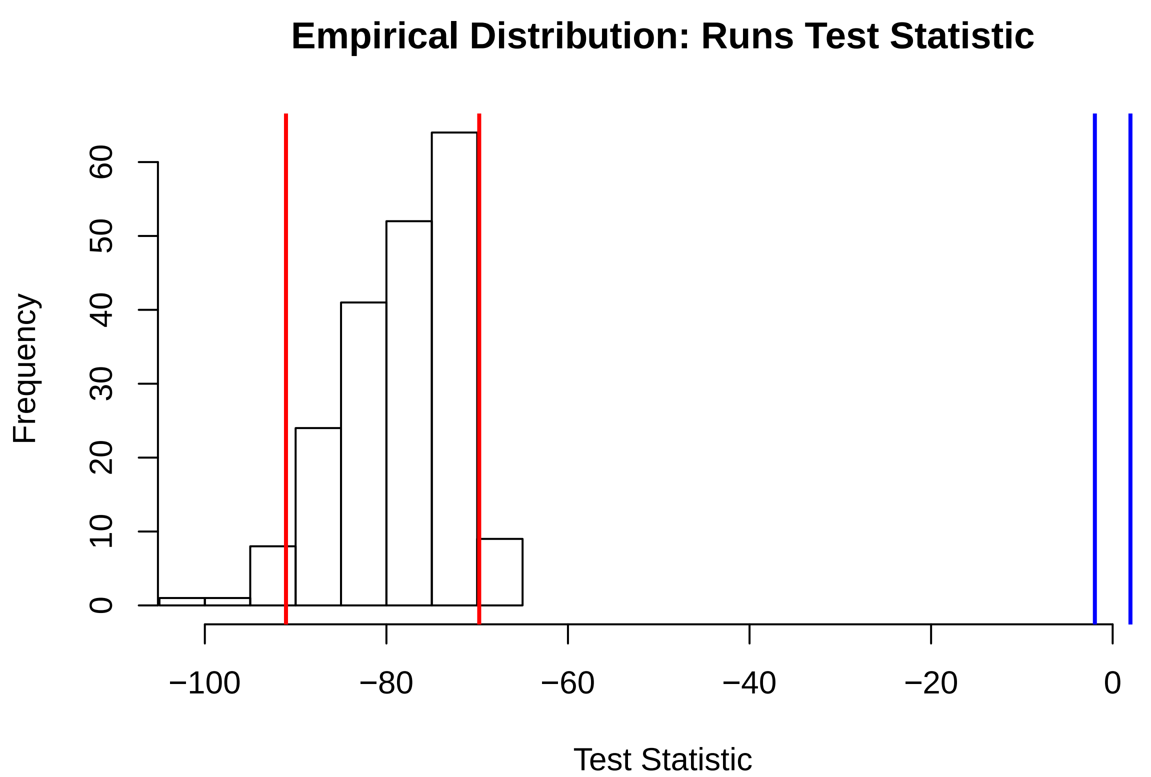
```
[1] 5.385 5.290
```

```r
apply(runspower, 2, mean)
```

```
[1] 0.055 1.000
```

25% decline, dispersion = 50

```r
nsim=200
impdata<-genOverallchangeData(log(0.75), init_glm, data = dat)
newdat<-generateNoise(nsim, impdata$truth, family='poisson', d=50)
fowsim_glm<-glm(newdat[,1] ~  TideState + WindStrength + SeaState + SimpPrecipitation + CloudCover + as
empdistribution<-getRunsCritVals(n.sim = nsim, simData=newdat,
                                 model = fowsim_glm, data = impdata, plot=TRUE,
                                 returnDist = TRUE, dots = FALSE)
```




```r
runspowsim_out<-runsPowerSim(newdat, fowsim_glm, empdistribution, nsim, powercoefid = length(coef(fowsi
```

```r
apply(runspowsim_out$rawrob, 2, sum)/nsim
```

```
  Emp  Norm 
0.055 1.000 
```

```r
(length(which(runspowsim_out$imppvals[,1]<=0.05))/nsim)*100
```

```
[1] 99
```

```r
(length(which(runspowsim_out$imppvals[,2]<=0.05))/nsim)*100
```

```
[1] 99
```

Multiple runs of power test: 25% decline, dispersion=50

```r
wilcox.test(x = runspower[,1], runspower[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

```r
wilcox.test(x = power[,1], power[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 36, p-value = 0.008334
alternative hypothesis: true location shift is not equal to 0
```

```r
apply(power, 2, mean)
```

```
[1] 99.580 99.535
```

```r
apply(runspower, 2, mean)
```

```
[1] 0.055 1.000
```

Multiple runs of power test: 10% decline, dispersion=50

```r
wilcox.test(x = runspower[,1], runspower[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value = 0.001904
alternative hypothesis: true location shift is not equal to 0
```

```r
wilcox.test(x = power[,1], power[,2], paired = T)
```

```

    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
```

```
V = 0, p-value = NA
alternative hypothesis: true location shift is not equal to 0
```

```r
apply(power, 2, mean)
```

```
[1] 36.4 36.4
```

```r
apply(runspower, 2, mean)
```

```
[1] 0.06 1.00
```

Multiple runs of power test: 10% decline, dispersion=80

```r
wilcox.test(x = runspower[,1], runspower[,2], paired = T)
```

```
    Wilcoxon signed rank test with continuity correction

data:  runspower[, 1] and runspower[, 2]
V = 0, p-value = 0.001904
alternative hypothesis: true location shift is not equal to 0
```

```r
wilcox.test(x = power[,1], power[,2], paired = T)
```

```
    Wilcoxon signed rank test with continuity correction

data:  power[, 1] and power[, 2]
V = 7.5, p-value = 0.4237
alternative hypothesis: true location shift is not equal to 0
```

```r
apply(power, 2, mean)
```

```
[1] 28.6 28.2
```

```r
apply(runspower, 2, mean)
```

```
[1] 0.06 1.00
```