

# STAT 812: Computational Statistics

## Simulation for Studying Point Estimation and Hypothesis Testing Procedures

Longhai Li

2024-09-26

### Contents

1	Using simulation to study two estimators of proportions	1
2	Using simulation to study t-test on non-normal observations	3
3	Using simulation to study a permutation test	7
4	Using simulation to study a power regression test	9

## 1 Using simulation to study two estimators of proportions

```
## Estimating Functions
p_est1 <- function(x) mean(x)

p_est2 <- function(x) {
  n <- length(x)
  (sum(x) + 1) / (n+2)
}

## Functions for estimating MSE
mse_est_mc <- function(n,p,no_sim,p_est)
{
  sq_error <- rep(0, no_sim)
  for(i_sim in 1:no_sim)
  { sq_error[i_sim] <- (p_est( rbinom(n,1,p) ) - p)^2
  }
  list(mse=mean(sq_error), sd = sqrt(var(sq_error)/no_sim) )
}

mse_p1 <- function(n,p,no_sim)
{
  sq_error <- rep(0, no_sim)
  for(i_sim in 1:no_sim)
  { sq_error[i_sim] <- (p1_est( rbinom(n,1,p) ) - p)^2
  }
  list(mse=mean(sq_error), sd = sqrt(var(sq_error)/no_sim) )
}
```

```

mse_p2 <- function(n,p,no_sim)
{
  sq_error <- rep(0, no_sim)
  for(i_sim in 1:no_sim)
  { sq_error[i_sim] <- (p2_est( rbinom(n,1,p) ) - p)^2
  }
  list(mse=mean(sq_error), sd = sqrt(var(sq_error)/no_sim) )
}

## Simulation
par( mfrow=c(2,2), mar=c(4,4,2,1) )

no_sim <- 10000

p_set <- seq(0,1,by = 0.05)

risk_est1 <- risk_est2 <- rep(0, length(p_set))
sd_risk_est1 <- sd_risk_est2 <- rep(0, length(p_set))

n_set <- c(2,10,50,100)

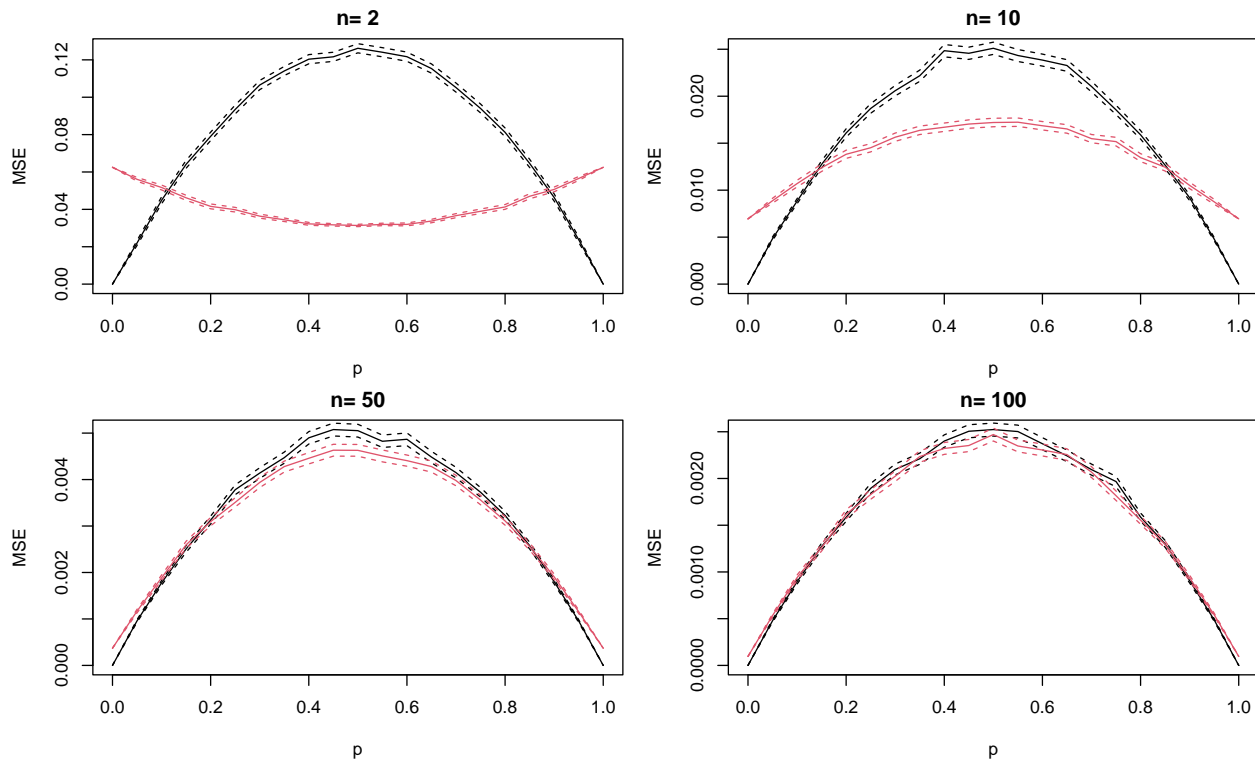
for( n in n_set )
{
  for(i in 1:length(p_set) )
  {
    output_est <- mse_est_mc(n=n,p=p_set[i],no_sim=no_sim,p_est1)
    risk_est1[i] <- output_est$mse
    sd_risk_est1[i] <- output_est$sd

    output_est <- mse_est_mc(n=n,p=p_set[i],no_sim=no_sim,p_est2)
    risk_est2[i] <- output_est$mse
    sd_risk_est2[i] <- output_est$sd
  }

  plot(p_set, risk_est1,type="l",
       xlab="p",ylab="MSE",main=paste("n=",n))
  points(p_set, risk_est1 + 1.96*sd_risk_est1,type="l",lty=2)
  points(p_set, risk_est1 - 1.96*sd_risk_est1,type="l",lty=2)

  points(p_set, risk_est2,type="l",col=2)
  points(p_set, risk_est2 + 1.96*sd_risk_est2,type="l",lty=2,col=2)
  points(p_set, risk_est2 - 1.96*sd_risk_est2,type="l",lty=2,col=2)
}

```



## 2 Using simulation to study t-test on non-normal observations

```
# SIMULATE T TESTS ON DATA FROM A T DISTRIBUTION. Simulates k data
# sets, each consisting of n data points that are drawn independently
# from the t distribution with df degrees of freedom. For each data
# set, the p-value for a two-sided t test of the null hypothesis that
# the mean is mu (default 0) is computed. The value returned by this
# function is the vector of these k p-values.

t.test.sim <- function (N, n, df, mu.true =0, mu=0)
{
  pvalues <- numeric(N)

  for (i in 1:N)
  { x <- rt (n, df) + mu.true
    pvalues[i] <- t.test.pvalue(x,mu)
  }

  pvalues
}

# FIND THE P-VALUE FOR A TWO-SIDED T TEST. The data is given by the first
# argument, x, which must be a numeric vector. The mean under the null
# hypothesis is given by the second argument, mu, which defaults to zero.
#
# Note: This function is just for illustrative purposes. The p-value
# can be obtained using the built-in t.test function with the expression:
```

```

#
#      t.test(x,mu=mu)$p.value

t.test.pvalue <- function (x, mu=0)
{
  if (!is.numeric(x) || !is.numeric(mu) || length(mu)!=1)
  { stop("Invalid argument")
  }

  n <- length(x)

  if (n<2)
  { stop("Can't do a t test with less than two data points")
  }

  t <- (mean(x)-mu) / sqrt(var(x)/n)

  2 * pt (-abs(t), n-1)
}

par (mfrow=c(3,2))

# Tests with 100 degrees of freedom, with 2 and 20 data points.

k <- 2000

bins <- 20

hist (t.test.sim(k,2,100), nclass=bins, prob=T,
      main="", xlab=paste(k,"pvalues for n=2, df=100"))

hist (t.test.sim(k,20,100), nclass=bins, prob=T,
      main="", xlab=paste(k,"pvalues for n=20, df=100"))

# Tests with 3 degrees of freedom, with 2 and 20 data points.

hist (t.test.sim(k,2,3), nclass=bins, prob=T,
      main="", xlab=paste(k,"pvalues for n=2, df=3"))

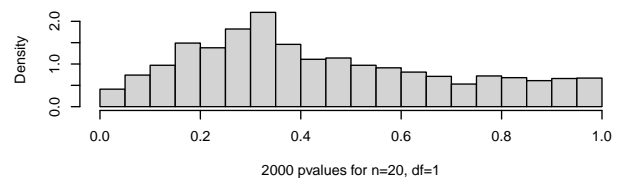
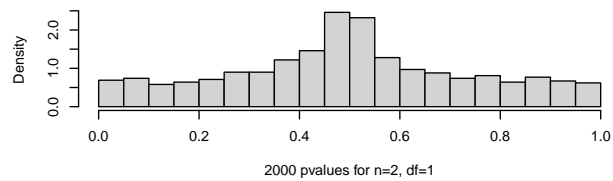
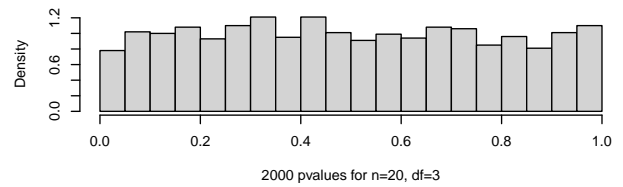
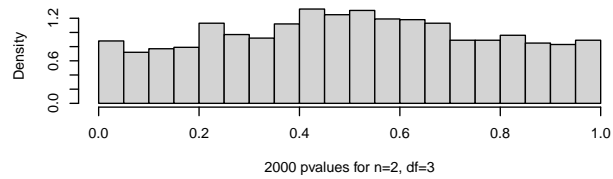
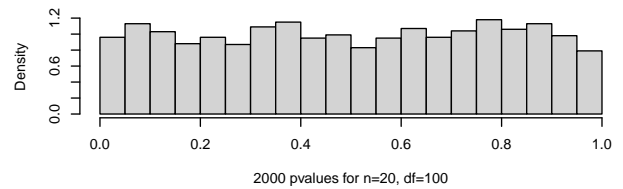
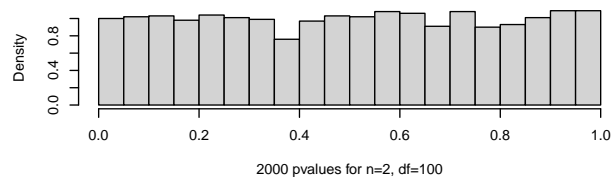
hist (t.test.sim(k,20,3), nclass=bins, prob=T,
      main="", xlab=paste(k,"pvalues for n=20, df=3"))

# Tests with 1 degree of freedom, with 2 and 20 data points.

hist (t.test.sim(k,2,1), nclass=bins, prob=T,
      main="", xlab=paste(k,"pvalues for n=2, df=1"))

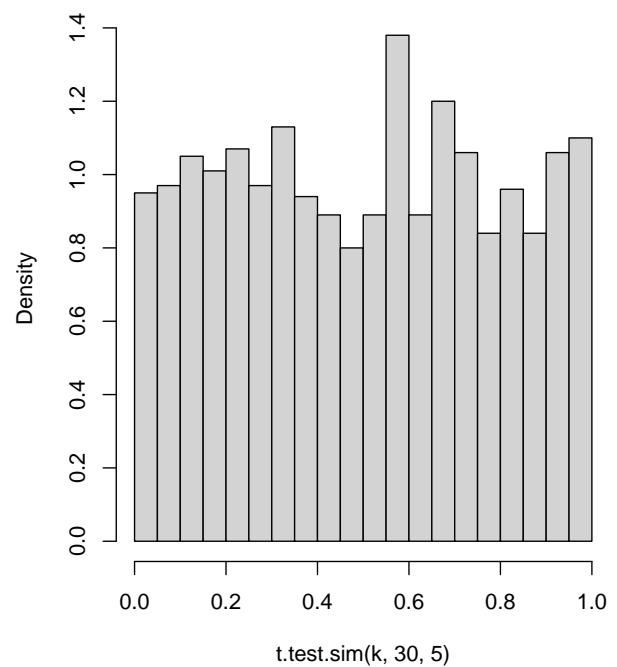
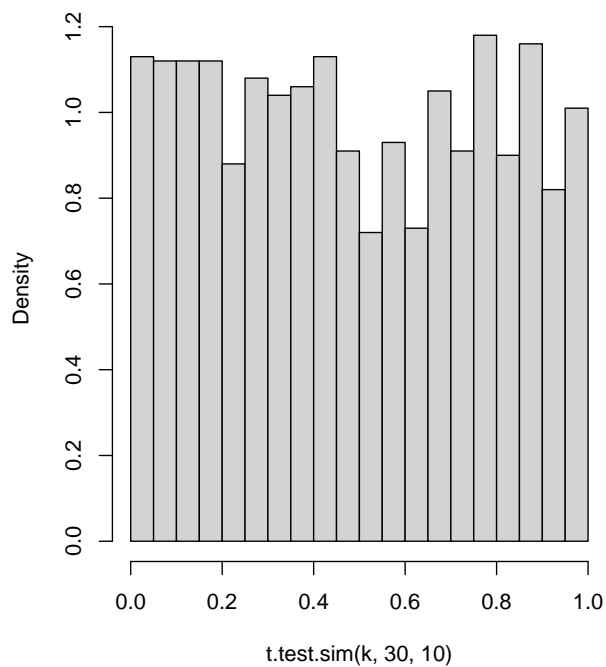
hist (t.test.sim(k,20,1), nclass=bins, prob=T,
      main="", xlab=paste(k,"pvalues for n=20, df=1"))

```



```
par (mfrow = c(1,2))
hist (t.test.sim(k,30,10), nclass=bins, prob=T,
      main="")

hist (t.test.sim(k,30,5), nclass=bins, prob=T,
      main="")
```

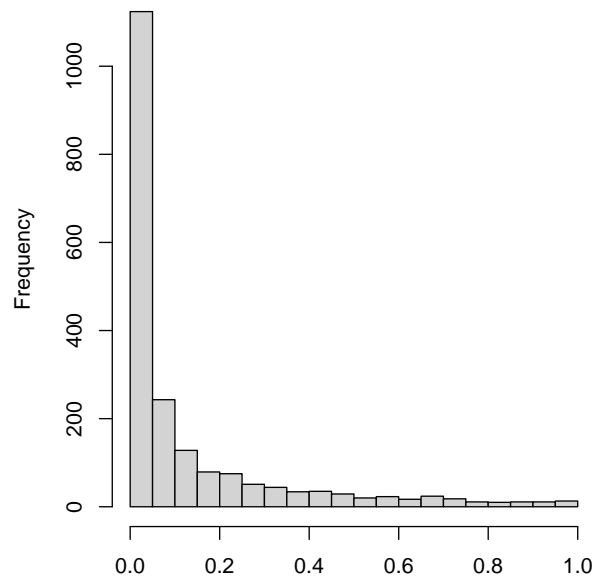


```
par (mfrow = c(1,2))
## find power of a test
hist(pvaues_H1 <- t.test.sim(k,30,5, mu.true = 0.5, mu = 0 ), nclass = 20, main = "")
```

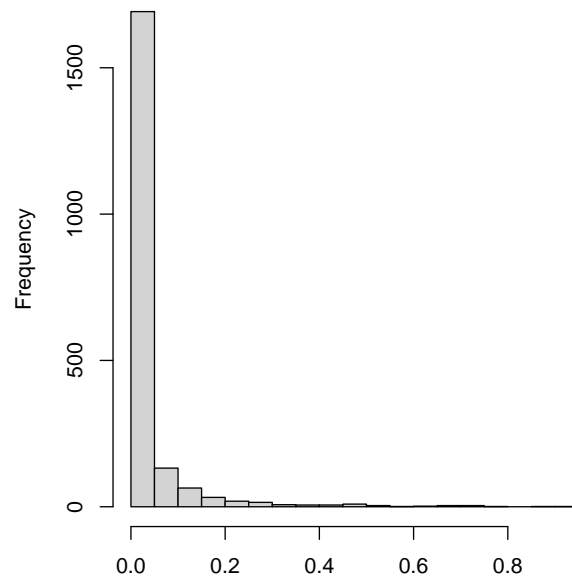
```
mean (pvaues_H1 < 0.05)
```

```
## [1] 0.562
```

```
hist(pvaues_H1 <- t.test.sim(k,60,5, mu.true = 0.5, mu = 0 ), nclass = 20, main = "")
```



pvaues\_H1 <- t.test.sim(k, 30, 5, mu.true = 0.5, mu = 0)



pvaues\_H1 <- t.test.sim(k, 60, 5, mu.true = 0.5, mu = 0)

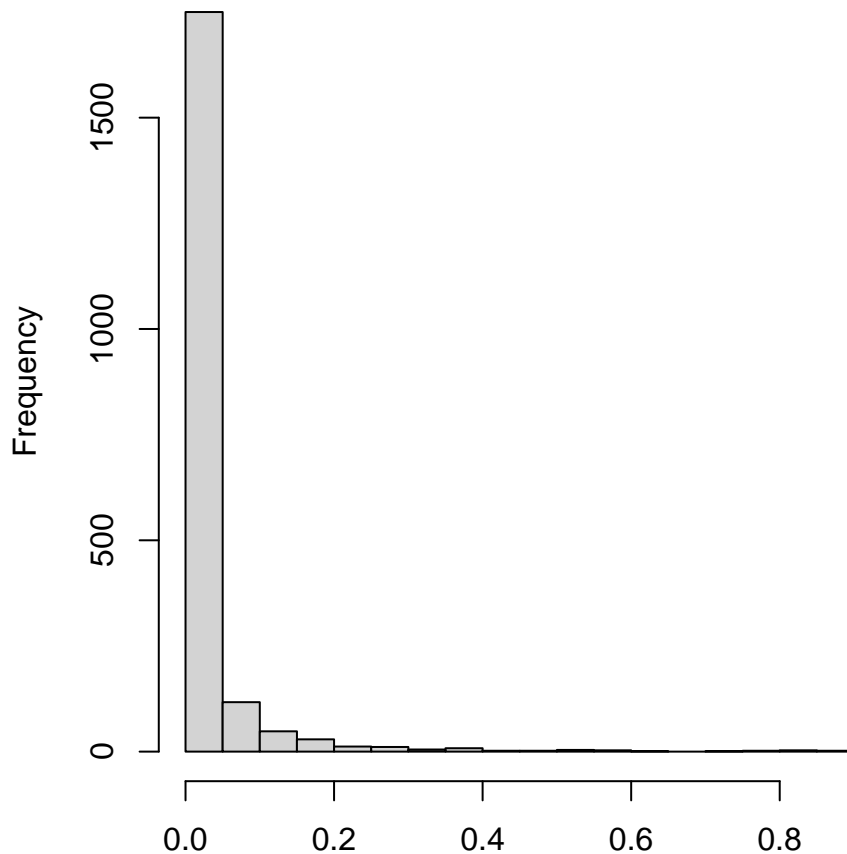
```
mean (pvaues_H1 < 0.05)
```

```
## [1] 0.846
```

```
hist(pvaues_H1 <- t.test.sim(k,70,5, mu.true = 0.5, mu = 0 ), nclass = 20, main = "")
```

```
mean (pvaues_H1 < 0.05)
```

```
## [1] 0.875
```



```
pvalues_H1 <- t.test.sim(k, 70, 5, mu.true = 0.5, mu = 0)
```

### 3 Using simulation to study a permutation test

```
# COMPUTE THE P-VALUE FOR A PERMUTATION TEST OF CORRELATION. Tests the null
# hypothesis that the the vectors x and y are independent, versus the
# alternative that they are correlated (either positively or negatively).
# The vectors x and y are given as the first and second arguments; they
# must be of equal length.
#
# The p-value returned is computed by simulating permutations of how the
# elements of the vectors are paired up, with the simulation sample
# size being given as the third argument, n, which defaults to 999. The
# p-values returned are integer multiples of 1/(n+1), and have the property
# that if the null hypothesis is true, the probability of obtaining a p-value
# of k/(n+1) or smaller is equal to k/(n+1), unless there is exact equality
# for the correlations obtained with different permutations, in which case the
# probability may differ slightly from this.

perm.cor.test <- function (x, y, n=999)
{
  real.abs.cor <- abs(cor(x,y))

  number.as.big <- 0
  for (i in 1:n)
```

```

{ if (abs(cor(x,sample(y))) >= real.abs.cor)
{ number.as.big <- number.as.big + 1
}
}

(number.as.big + 1) / (n + 1)
}

# TEST ON NORMALLY-DISTRIBUTED DATA, COMPARED TO TEST BASED ON NORMAL DIST.

test.perm.norm <- function(no.sim,no.perm)
{
  pvalue <- rep(0,no.sim)

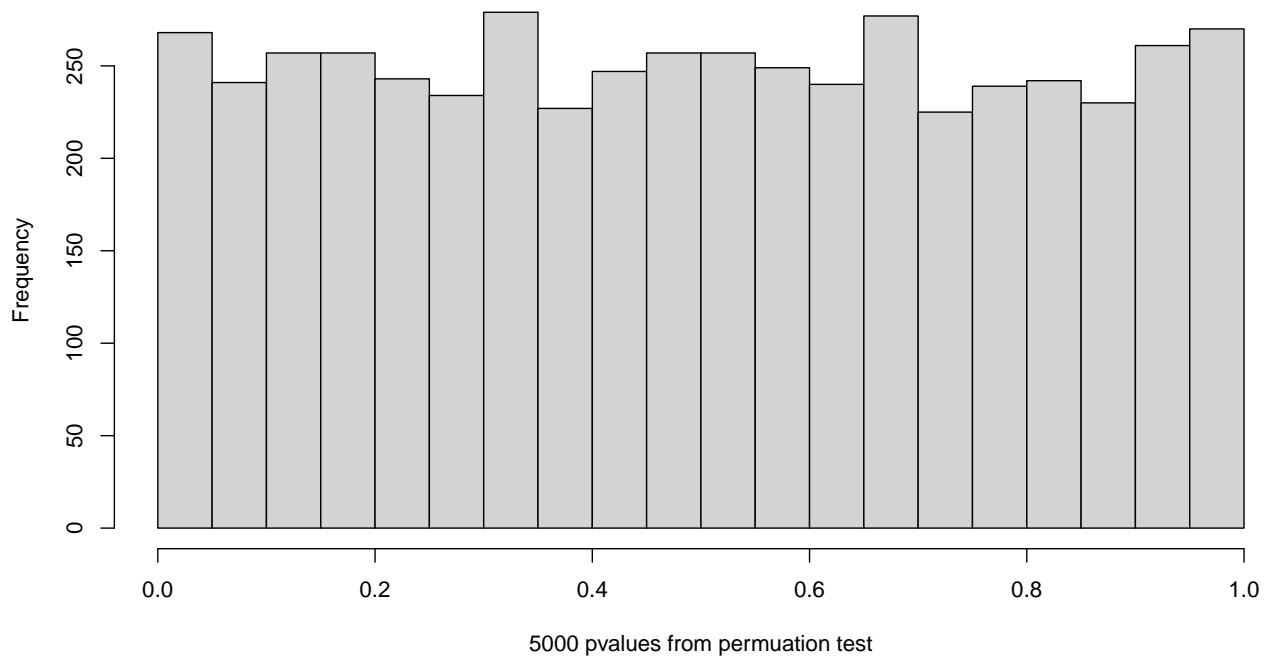
  for(i in 1:no.sim)
  {
    x <- rnorm(20)
    y <- rnorm(20)

    pvalue[i] <- perm.cor.test(x,y,no.perm)
  }

  pvalue
}

pv <- test.perm.norm(5000,100)
par (mfrow = c(1,1))
hist(pv,20, xlab="5000 pvalues from permuation test",main="")

```





## 4 Using simulation to study a power regression test

```
# REGRESSION USING BEST POWER TRANSFORMATION. Takes as arguments a
# vector of predictor values and a vector of response values (of equal
# length). The predictor values must be non-negative. Finds the
# power transformation for the predictors that produces the highest
# correlation (in absolute value) with the response, choosing the power
# from the powers argument (which defaults to seq(0.1,2.0,by=0.1).
# Returns the pvalue

pvalue.lm.pow <- function (x, y, powers=seq(0.1,4.0,by=0.1))
{
  if (!is.vector(x,"numeric") || !is.vector(y,"numeric")
      || length(x)!=length(y))
  { stop("Arguments should be numeric vectors of equal length")
  }

  if (any(x<0))
  { stop("Predictors must be non-negative")
  }

  n <- length(x)

  # Find the power that produces the highest correlation with the response.

  best.r <- 0

  for (p in powers)
  { r <- cor(x^p,y)
    if (abs(r)>=abs(best.r))
    { power <- p
      best.r <- r
    }
  }

  # Return the best power and the linear model using that power.

  xp <- x^power
  #return naive pvalue
  coef(summary(lm(y~xp)))[2,4]
}

# TEST VALIDITY OF THE NAIVE P-VALUES. Simulates the results of
# naively interpreting pvalues for the regression on the best
# power of the predictor variable (from pvalue.lm.pow) as a real pvalue.
# The arguments of this function are the vector of predictor variables
# to use (x), the number of datasets to simulate (N), and possible
# further arguments that are passed on to pvalue.lm.pow. N datasets of n
# cases are generated in which x is as specified and the corresponding
# y values are generated independently from the standard normal
# distribution (without reference to x). The result is the vector of
# N p-values obtained from the models that pvalue.lm.pow chooses for these
# datasets. Since the null hypothesis of no relationship is true,
# these p-values should be uniformly distributed between 0 and 1, if
```

```

# they are valid.

test.lm.power <- function (x, N, ...)
{
  n <- length(x)

  # Simulate N datasets and record the naive p-value found for each.

  pvalues <- rep(0,N)

  for (k in 1:N)
  {
    y <- rnorm(n)

    pvalues[k] <- pvalue.lm.pow(x,y,...)
  }

  # Return the vector of N p-values that were obtained.

  pvalues
}

# FIND PERMUTATION PVALUE FOR REGRESSION USING BEST POWER TRANSFORM.
# Takes as arguments the vectors of predictors (x) and responses (y),
# as for pvalue.lm.pow, the number of permutations to use in finding the
# p-value (default is 999), and possible further arguments that are
# passed on to pvalue.lm.pow. Returns the p-value from the permutation
# test, which is (roughly) the fraction of times that reg.pow applied
# to a randomly shuffled data sets gives a smaller p-value than reg.pow
# gives when applied to the actual dataset. In detail, the pvalue is
# (count+1)/(perms+1), with count being the number of smaller p-values
# obtained from permuted datasets. If the null hypothesis of no
# relationship is true, these p-values will be uniformly distributed
# over the possible values (ie, roughly uniform between 0 and 1).

pvalue.perm.lm.pow <- function (x, y, perms=999, ...)
{
  # Find the naive p-value using pvalue.lm.pow for the actual data.

  actual <- pvalue.lm.pow(x,y,...)

  # Count how many times pvalue.lm.pow applied to a random permutation finds
  # a model with as small a naive p-value as for the actual data.

  count <- 0

  for (k in 1:perms)
  { pvalue <- pvalue.lm.pow(x,sample(y),...)
    if (pvalue<=actual)
    { count <- count + 1
    }
  }
}

```

```

    # Return the p-value from the permutation test, based on the count found.

    (count+1) / (perms+2)
}

# TEST VALIDITY OF THE PERMUTATION PVALUES. The method is the same as for
#test.lm.power. When power.sim is not 0, it can be used to calculate the power
#of the test

test.perm.power <- function (x, power.sim, N=100, resid.std.dev=1, ...)
{
  n <- length(x)

  pvalues <- rep(0,N)

  for (k in 1:N)
  { y <- x^power.sim + rnorm(n,0,resid.std.dev)
    pvalues[k] <- pvalue.perm.lm.pow(x,y,...)
  }
  pvalues
}

par(mfrow=c(2,2),mar=c(4,4,3,1))

x <- exp(rnorm(20))
naive.pv <- test.lm.power(x, 1000)

hist(naive.pv,nclass=20,xlab="1000 naive pvalues from power regression",
     main="null hypothesis is true, naive test")

perm.pv <- test.perm.power(x,0,1000,perms=99)

hist(perm.pv,nclass=20,
     xlab="1000 permutation pvalues from power regression",
     main="null hypothesis is true,permutation test")

#p-value when the true power in function is 1.5
perm.pv.alt1 <- test.perm.power(x,1.5,1000,perms=49)

hist(perm.pv.alt1,nclass=20,
     xlab="1000 permutation pvalues from power regression",
     main="true relationship: y = x^1.5 + e,permutation test")

#p-value when the true power in function is 0.5
perm.pv.alt2 <- test.perm.power(x,0.5,1000,perms=49)

hist(perm.pv.alt2,nclass=20,
     xlab="1000 permutation pvalues from power regression",
     main="true relationship: y = x^0.5 + e, permutation test")

```

