Monte Carlo Simulation Examples

A. Burnetas

Example 1: Estimating Excess

Let $X \sim N(\mu, \sigma^2)$ be a random variable representing the outcome of a biological measurement (e.c. cholesterol level) in a population with known parameters μ, σ. We want to compute the expected excess from a normal level d, i.e., $\theta_1 = E(\max(X-d, 0))$. This quantity can be computed numerically as follows: $\theta_1 = \int_d^\infty (x-d) f(x) dx$, where $f(x)$ the probability density function of the normal distribution with parameters μ, σ^2.

To approximate the value of θ_1 with simulation, we let $\theta_1 = E(Y)$, where $Y = h(X) = \max(X-d, 0)$. Then we can create a random sample $x_1, \ldots, x_N$ of size N from $N(\mu, \sigma^2)$, compute the quantities $y_1, \ldots, y_n$, where $y_j = h(x_j) = \max(x_j, 0)$ and estimate θ_1 from the sample mean

$$\hat{\theta_1} = \frac{\sum_{j=1}^N x_j}{N}.$$

In addition, by treating $y_1, \ldots, y_n$ as a random sample from the distribution of Y, we can create a $(1-\alpha)100\%$ confidence interval for $E(Y)$ using the central limit theorem approximation :

$$\hat{\theta_1} - t_{\alpha/2, N-1} \frac{s}{\sqrt{N}} \leq \theta_1 \leq \hat{\theta_1} + t_{\alpha/2, N-1} \frac{s}{\sqrt{N}}$$

where s is the sample standard deviation.

To implement the above, we create an R function with parameters μ, σ, d, N, a, where $1-a$ is the confidence level:

```
normexcessest=function(m, s, d, N, a)
{
  x=rnorm(N, m, s)
  y=rep(0,N)
  for (j in 1:N)
  {
    y[j]=max(x[j]-d,0)
  }
  theta1est=mean(y)
  sy=sd(y)
  halflength=qt(1-a/2, N-1)*s/sqrt(N)
  confint=c(theta1est-halflength, theta1est, theta1est+halflength)
  return(confint)
}
```

As an application, assume that the level of LDL cholesterol in the population follows normal distribution with mean 92 and standard deviation 15 and the critical level is set at $d = 100$. We can now estimate $\theta_1 = E(\max(X-100, 0)$ with a 95% confidence interval by simulating a pseudorandom sample of size $N = 10000$:

```
ldlexcess=normexcessest(92,15,100, 10000, 0.05)
ldlexcess
```

```
## [1] 2.471656 2.765686 3.059717
```

Example 2: Estimating Conditional Excess

In the Example θ_1 is the expected excess for the entire population, where any individual whose measurement is $X \leq d$ is counted as zero. For example if we consider a random sample of 20 measurements

```
x=rnorm(20, 92, 15)
x
```

```
##  [1] 114.59500  93.47105  86.68142  95.51268  79.83165  75.67152 101.36716
##  [8]  99.63922 105.11193  96.89739 118.19196  87.66091  94.63889 109.93221
## [15] 107.38426  74.43636  68.96959  67.97844  87.35727  88.21708
```

and compute the excess above 100 for each of them

```
y=rep(0,20)
for (j in 1:20)
{
  y[j]=max(x[j]-100,0)
}
y
```

```
##  [1] 14.595005  0.000000  0.000000  0.000000  0.000000  0.000000  1.367156
##  [8]  0.000000  5.111931  0.000000 18.191961  0.000000  0.000000  9.932214
## [15]  7.384256  0.000000  0.000000  0.000000  0.000000  0.000000
```

we see that there are 14 measurements below 100 and those are counted as zero excess. The mean excess of the sample in this case is

```
thest=sum(y)/20
thest
```

```
## [1] 2.829126
```

However it may also be of interest to estimate the mean excess only for those individuals whose measurement actually exceeds the level d. In the above example the mean excess would be the average excess of the 6 observations that are above 100:

```
y1=y[y>0]
mean(y1)
```

```
## [1] 9.430421
```

Mathematically, the new quantity is the conditional expectation $\theta_2 = E[\max(X-d,0)|X-d>0]$. We can see that

$$\begin{aligned}\theta_1 &= E[\max(X-d,0)] \\ &= P(X-d <= 0)\ E[\max(X-d,0)|X-d \leq 0] + P(X-d>0)\ E[\max(X-d,0)|X-d>0] \\ &= P(X-d>0)\ E[\max(X-d,0)|X-d>0] = P(X-d>0)\theta_2\end{aligned}$$

from which it follows that

$$\theta_2 = \frac{\theta_1}{P(X-d>0)}.$$

Therefore, one way to estimate θ_2 is to create a sample of size N from $X \sim N(\mu, \sigma^2)$, transform it to the vector of excess values $y_j = \max(x_j - d, 0)$ and compute the sum of y_j, $S_N = \sum_{j=1}^{N} y_j$ and the number of y_j that are strictly positive $N_d = \sum_{j=1}^{N} 1(y_j > 0)$. We can then compute estimates of θ_1 and p:

$$\hat{\theta_1} = \frac{S_N}{N}, \quad \hat{p} = \frac{N_2}{N}.$$

Now an estimate of θ_2 is

$$\hat{\theta_2} = \frac{\hat{\theta_1}}{\hat{p}} = \frac{S_N}{N_2}.$$

This is consistent with the previous discussion, where we estimate the conditional excess by averaging over only the positive measurements.

We can now create a function that implements the estimation of the conditional excess by simulating a sample of size N from X:

```
condnormexcess=function(m,s,d,N)
{
 x=rnorm(N, m, s)
  y=rep(0,N)
  for (j in 1:N)
  {
    y[j]=max(x[j]-d,0)
  }
  N2=length(y[y>0])
  thetaj2est=sum(y)/N2
}
```

For the LDL example, we now estimate the conditional excess of LDL above 100, given that an individual does have LDL>100 as

```
ldlcondexcess=condnormexcess(92,15,100,10000)
ldlcondexcess
```

```
## [1] 9.24575
```

Regarding bias and confidence intervals, things are not so simple. Parameter θ_2 has not been expressed as an expectation, but rather as the ratio of two expectations and it is estimated as the ration of the two respective estimates. However in this case the estimator is not unbiased. Thus, even if we obtain a confidence interval for its mean, this is not necessarily a confidence interval for θ_2.

One approach to deal with this problem is to express θ_2 as the expectation of $Y = X - d$, where X follows the conditional distribution $X|X > d$:

$$F(x|X > d) = P(X \leq x|X > d) = \frac{F(x)}{1 - F(d)}, \quad x > d.$$

If we generate N i.i.d. observations from this distribution, i.e., a random sample, then the sample mean will be un unbiased estimate of θ_2 and the usual approach for constructing a confidence interval can be applied. However the above conditional distribution is not available to simulate using a prebuilt generator and we must create an ad-hoc random number generator for it. An easy way to do this is by thinking as follows:

Assume we create successive i.i.d. observations from the $N(\mu, \sigma^2)$ distribution. As long as an observation is below d we reject it, and we accept the first value that is above d. Assume that $\tilde{X}$ is the first observation that is accepted. Then $\tilde{X}$ is a random variable and its distribution is

$$P(\tilde{X} \leq x) = P(X \leq x|X \text{is accepted}) = P(X \leq x|X > d),$$

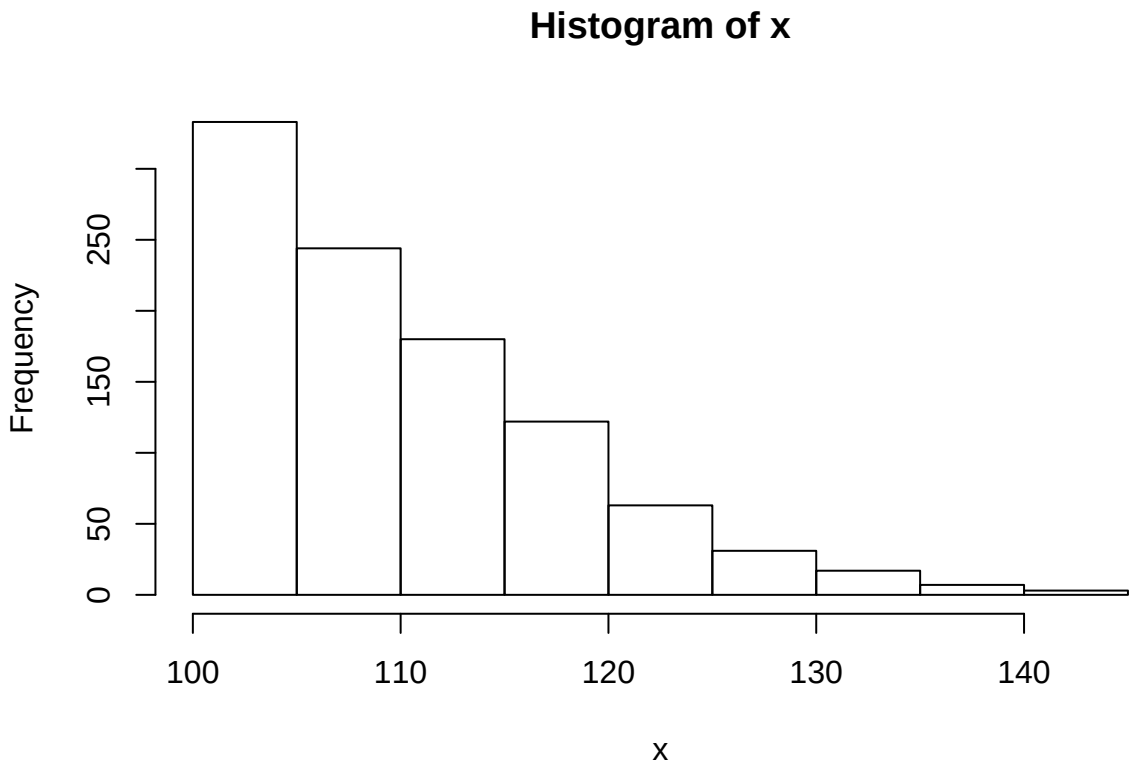
i.e., the conditional distribution of $X|X > d$. We can create this generator with the following function.

```
condexcgen=function(m,s,d)
{
  x=rnorm(1,m,s)
  while (x <=d)
    x=rnorm(1,m,s)
```

```
  return(x)
}
```

We can create a sample of size $N = 1000$ from the distribution for $\mu = 92, \sigma = 15, d = 100$ and plot the histogram:

```
x=rep(0,1000)
for (j in 1:1000)
  x[j]=condexcgen(92,15,100)
hist(x)
```



Now that we have a generator from this distribution, we can estimate its mean θ_2 using a method similar to that for θ_1. The only difference is that we use our generator instead of rnorm, and now $y_j = x_j - d$. Also, since we only need y we do not need to store all generated values of x:

```
condexcessest=function(m, s, d, N, a)
{
  y=rep(0,N)
  for (j in 1:N)
  {
    x = condexcgen(m,s,d)
    y[j]=x-d
  }
  theta2est=mean(y)
  sy=sd(y)
  halflength=qt(1-a/2, N-1)*s/sqrt(N)
  confint=c(theta2est-halflength, theta2est, theta2est+halflength)
```

```
  return(confint)
}
```

Applying the function in our example with $N = 10000$ replications,

```
condldlexcess=condexcessest(92,15,100,10000,0.05)
condldlexcess
```

```
## [1] 9.112945 9.406976 9.701006
```

Using this method, we can obtain a confidence interval for θ_2 in addition to the point estimate, which could also be derived with the simpler first method. The interpretation of this estimate is that the individuals whose LDL is above 100, are expected to have LDL on average 9.4069755 units above the acceptable limit of 100, i.e., 109.4069755 units.

Example 3: Assess estimator bias

Consider a random variable X following exponential distribution with parameter λ unknown. The mean of this distribution is $1/\lambda$ and we can easily obtain an unbiased estimator for it by collecting a sample of size n and computing the sample mean.

However for the rate itself things are not as simple. Assume that we have a sample $x_1, \ldots, x_n$ of size n from this distribution and we want to estimate λ. The maximum likelihood etimator for λ is known and is equal to $\hat{\lambda} = \frac{n}{\sum_{i=1}^{n} x_i} = \frac{1}{\bar{x}_n}$, i.e. equal to the inverse of the estimator of the mean. This estimator is biased, since $E(1/\bar{x}_n) \neq 1/E(\bar{x}_n) = 1/(1/\lambda) = \lambda$. The question is to estimate the bias of $\hat{\lambda}$ as a function of the sample size n and the true (unknown) value λ, i.e, $b(n, \lambda) = E_\lambda(1/\bar{X}_n) - \lambda$, where $\bar{X}_n = \frac{\sum_{i=1}^{n} X_i}{n}$ and $X_1, \ldots, X_n$ i.i.d. random variables with exponential distribution with rate λ.

To do this with simulation, we must simulate the estimation process a number N of times (replications). In each replication we generate a sample of size n from the distribution with the given λm compute the estimate $\hat{\lambda}$ and find the difference of the two. Repeating this N times and taking the sample mean of the N differences we can obtain an estimate of the bias for the given n, λ. Note the difference between N, which is the number of simulated replications (scenarios) and n, which is the sample size of the estimator $\hat{\lambda}$ whose bias we want to assess.

To do this we create a function that simulates the estimation once under given λ and n and computes the difference of this estimate from the true value:

```
simulexpbias=function(l, n)
{
  x=rexp(n,l)
  lhat=1/mean(x)
  return(lhat-l)
}
```

Function simulexpbias generates one replication of the estimation process, i.e., one simulated value of the bias. We next apply the to estimate the bias of the MLE under sample size $n = 10$ and $\lambda = 2$ based on $N = 10000$ replications

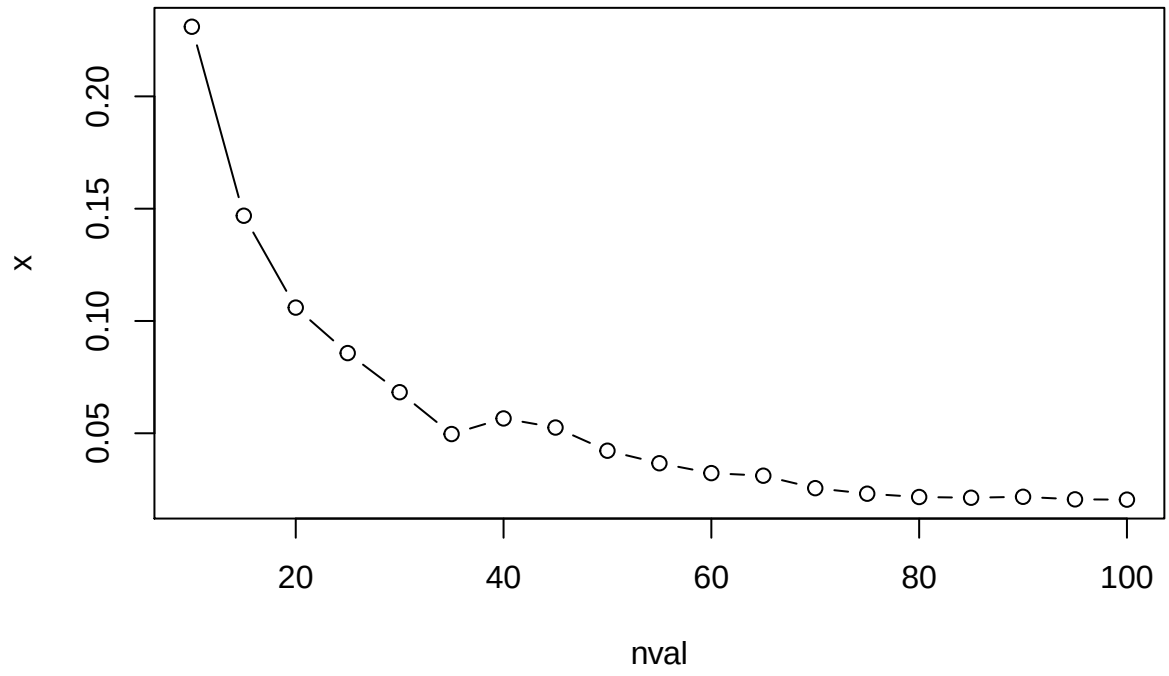
```
expbiasest=function(l,n,N)
{
  x=rep(0,N)
  for (i in 1:N)
  {
    x[i]=simulexpbias(l,n)
  }
  biasest=mean(x)
```

```
  return(biasest)
}
biasest=expbiasest(2,10,10000)
biasest
```

```
## [1] 0.2024302
```

We can also use this function to assess the behavior of the bias with respect to λ and n. For example, to see how the bias changes with the sample size, we fix $\lambda = 1$ and repeat the simulation for $n = 10, 15, \ldots, 100$.

```
N=10000
l=2
nval=seq(10, 100, by=5)
nn=length(nval)
x=rep(0,nn)
for (i in 1:nn)
{
  x[i]=expbiasest(l, nval[i], N)
}
plot(x~nval, type='b')
```



We see that the bias approaches zero as n increases.

Example 4: Power of t-test

Let $X \sim N(\mu, \sigma^2)$ with unknown μ, σ. We consider the two-sided hypothesis test

$$H_0 : \mu = \mu_0, \quad H_1 : \mu \neq \mu_0$$

Assume we obtain a sample $x_1, \ldots, x_n$ from this distribution. Based on this sample the decision rule of the test at significance level α is : Reject H_0 if $|t| > t_{\alpha_2, n-1}$, and do not reject H_0 otherwise, where s is the sample standard deviation and $t \equiv \frac{\bar{x}_n - \mu_0}{\frac{s}{\sqrt{n}}}$ is the test t-statistic. Equivalently, H_0 is rejected if $p < \alpha$, where p is the p-value of the test.

We want to estimate the power of the t-test, i.e., the probability $P_0 = 1 - \beta$, where β is the type-II error, i.e., the probability of incorrectly accepting the null hypothesis, given that the true value of the mean is $\mu \neq 0$, i.e., $\beta(\mu, \sigma) = P(accept H_0 | \mu, \sigma)$. Therefore, P_0 is the probability of correctly rejecting the null hypothesis if it is not true. Of course a test is more reliable when the power is higher under a given α level. The power is not a single value, but a function of the true parameter value μ. Intuitively we expect that it increases when μ is further from μ_0, since then it should be easier for the test to identify the correct answer. Finally, for $\mu = \mu_0$, in which case H_0 is true, $P_0 = \alpha$, i.e., equal to the type-I error.

For the t-test under a normal distribution the power can be computed analytically, with a rather complicated formula that involves the noncentral t distribution. The R function power.t.test can be used to compute the power exactly. For example for the two-sided test $H_0 : \mu = 11$, $H_1 : \mu \neq 11$, with sample size $n = 10$, $\alpha = 0.05$ and true distribution parameters $\mu = 10, s = 2$, the power can be found as follows (delta $= \mu - \mu_0$)

```
p1=power.t.test(n=10, delta=-1, sd=2, sig.level = 0.05, type="one.sample", alternative = "two.sided")
p1
```

```
##
##      One-sample t test power calculation
##
##               n = 10
##           delta = 1
##              sd = 2
##       sig.level = 0.05
##           power = 0.2928286
##     alternative = two.sided
```

```
p1$power
```

```
## [1] 0.2928286
```

Although there are are analytical expressions for the power of the t-test, we will estimate it using simulation, as another example of the use of simulation methods for evaluating statistical methodologies. Since the exact value is known, we can also evaluate the quality of the simulation approximation (in general the true value is not known and this is typically the reason we use simulation).

The building block of the simulation experiment will be a function that simulates one replication of the t-test under given $\mu, \sigma, \mu_0, \alpha, n$: It generates a random sample of size n from $N(\mu, \sigma_2)$, performs the two-sided t-test at significance level α, and returns a binary variable $R = 1$ if H_0 is rejected and $R = 0$ otherwise:

```
simulatettest=function(m,s,m0,a,n)
{
  x=rnorm(n, m, s)
  t=t.test(x, mu=m0, conf.level = 1-a, alternative = "two.sided")
  tp=t$p.value
  if (tp<a) return(1) else return(0)
}
```

Note that the t-test inside the function could also be implemented directly without calling the t.test function, by computing the t-statistic as above.

Assume we repeat the simulation N times with given values of n, μ, σ, α using different simulated samples at each replication. Let $R_1, \ldots, R_N$ be the outcomes of each replication. Then an estimate of the power is the proportion of replications in which H_0 was rejected i.e., the outcome was $R = 0$. Therefore the estimate of

the power is $\hat{P} = \bar{R}_N$.

For example, to estimate the power of a t-test with sample size $n = 10$, $\alpha = 0.05$, true distribution parameters $\mu = 10, \sigma = 2$ and hypothesized value $\mu_0 = 11$, we perform a simulation experiment with $N = 10000$ replications of the simulattetst function:

```
N=10000
n=10
a=0.05
m=10
s=2
m0=11
x=rep(0,N)
for (i in 1:N)
{
  x[i]=simulatettest(m,s,m0,a,n)
}
powerest=mean(x)
sx=sd(x)
hlength=qt(0.975, N-1)*sd(x)/sqrt(N)
powerconfint=c(powerest-hlength, powerest+hlength)
powerest
```

```
## [1] 0.2992
```

```
powerconfint
```

```
## [1] 0.2902236 0.3081764
```

We observe that the confidence interval contains the exact value, which we computed before for the same test.

Since we want to use these calculations repeatedly to compute the power function, we create a new function that simulates the t-test under given parameters with N replications and returns a point estimate and a 95% confidence interval for the power.

```
estpowerttest=function(m,s,m0,a,n,N)
{
  x=rep(0,N)
  for (i in 1:N)
  {
    x[i]=simulatettest(m,s,m0,a,n)
  }
  powerest=mean(x)
  sx=sd(x)
  hlength=qt(0.975, N-1)*sd(x)/sqrt(N)
  powerconfint=c(powerest-hlength, powerest+hlength)
  results=list(est=powerest, confint=powerconfint)
}
p1=estpowerttest(10,2,11,0.05,10,10000)
p1
```

```
## $est
## [1] 0.2906
##
## $confint
## [1] 0.2816995 0.2995005
```

```
p1$est
```

```
## [1] 0.2906
```

```
p1$confint
```

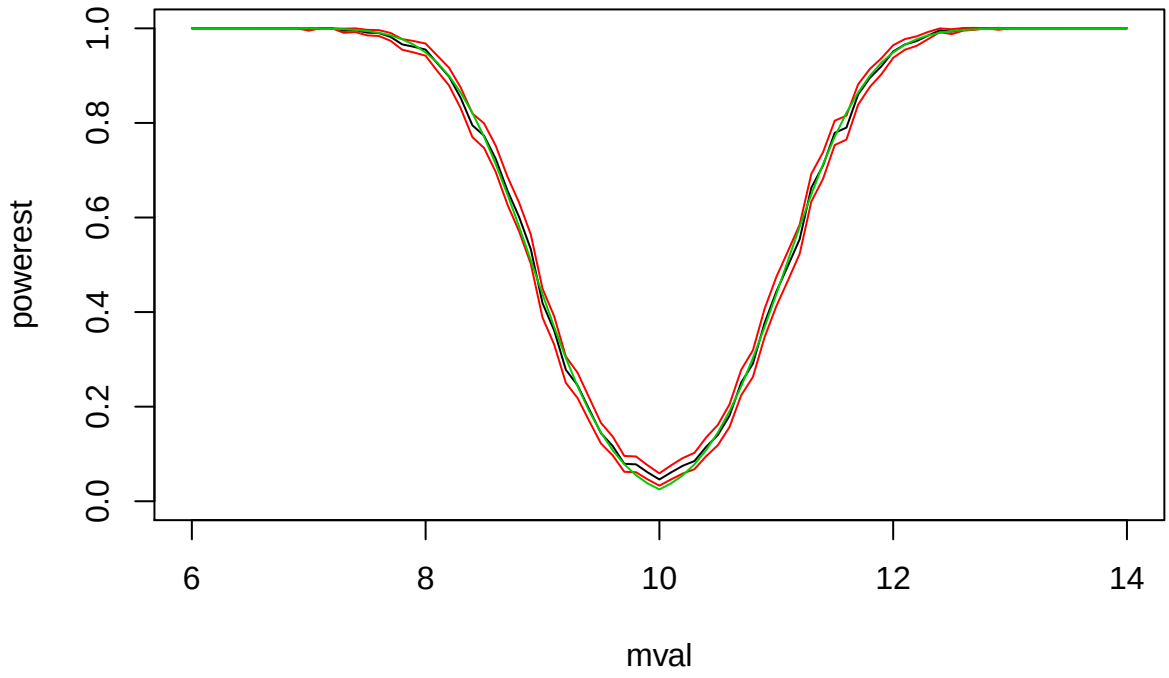
```
## [1] 0.2816995 0.2995005
```

In our first simulation study we will estimate the power function of the one-sample two-sided t test:

$$H_0 : \mu = 10 : \quad H_1 : \mu \neq 10$$

with sample size $n = 15$, significance level $a = 0.05$ population standard deviation $\sigma = 2$ (considered unknown) and population mean μ varying between 6 and 14 with a step size of 0.1. We will make a plot of the confidence interval of the power function, as well as the exact value for comparison:

```
N=1000
mval=seq(6,14,0.1)
s=2
a=0.05
n=15
m0=10
nm=length(mval)
powerest=rep(0,nm)
powerconfint=matrix(rep(0,2*nm), nm, 2)
powerexact=rep(0,nm)
for (i in 1:nm)
{
  m=mval[i]
  p1=estpowerttest(m,s,m0,a,n,N)
  powerest[i]=p1$est
  powerconfint[i,]=p1$confint
  pex=power.t.test(n, delta=m-m0, sd=2, sig.level = 0.05, type="one.sample", alternative = "two.sided")
  powerexact[i]=pex$power
}
plot(powerest~mval, type='l', ylim=c(0,1), col=1)
lines(powerconfint[,1]~mval, col=2)
lines(powerconfint[,2]~mval, col=2)
lines(powerexact~mval, col=3)
```



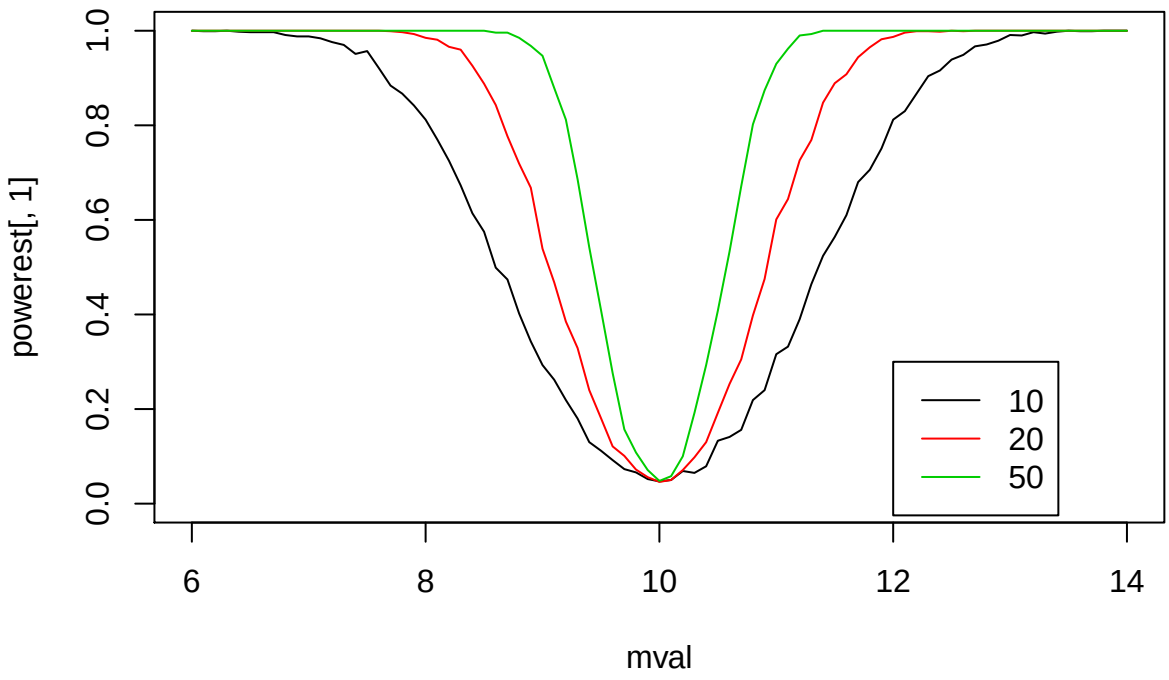
We observe that the true value is almost always inside the confidence interval.

In the second experiment we will plot the estimated power functions for the same parameters as above but with sample sizes $n = 10, 20, 50$.

```
N=1000
mval=seq(6,14,0.1)
s=2
a=0.05
nval=c(10,20,50)
m0=10
nm=length(mval)
nn=length(nval)
powerest=matrix(rep(0,nm*nn), nm, nn)

for (j in 1:nn)
  {
    for (i in 1:nm)
    {
      n=nval[j]
      m=mval[i]
      p1=estpowerttest(m,s,m0,a,n,N)
      powerest[i,j]=p1$est
    }
  }
plot(powerest[,1]~mval, type='l', ylim=c(0,1), col=1)
for (i in 2:nn)
```

```
  lines(powerest[,i]~mval, type='l', col=i)
legend(x=12, y=0.3, legend=nval, lty=1, col=1:nn)
```



The graphs show how the power of the t-test increases with the sample size, in particular for values of the true mean close to the hypothesized value.