A Vignette for the R package: ezsim

TszKin Julian Chan <ctszkin@gmail.com>

June 25, 2014

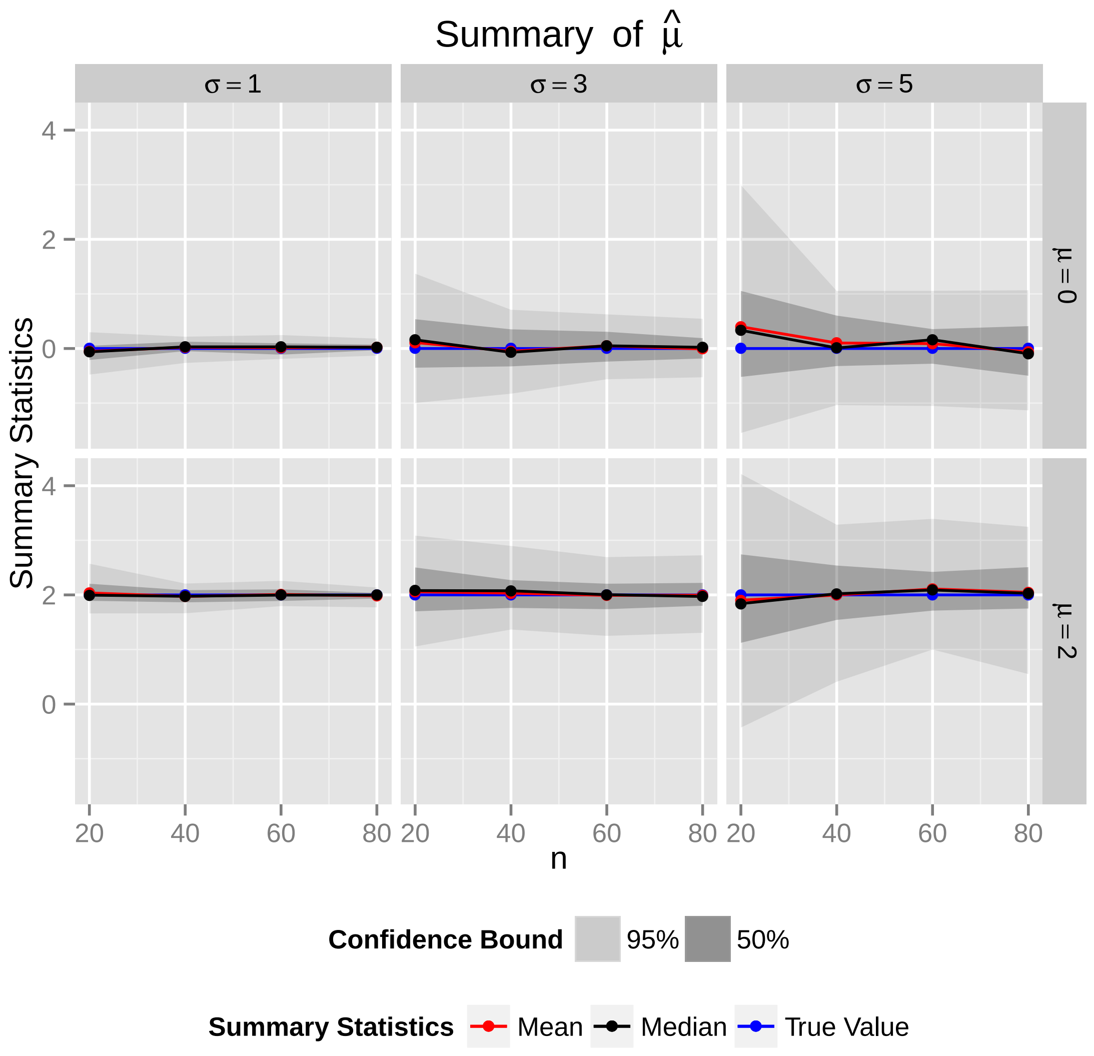
Contents

1 Introduction

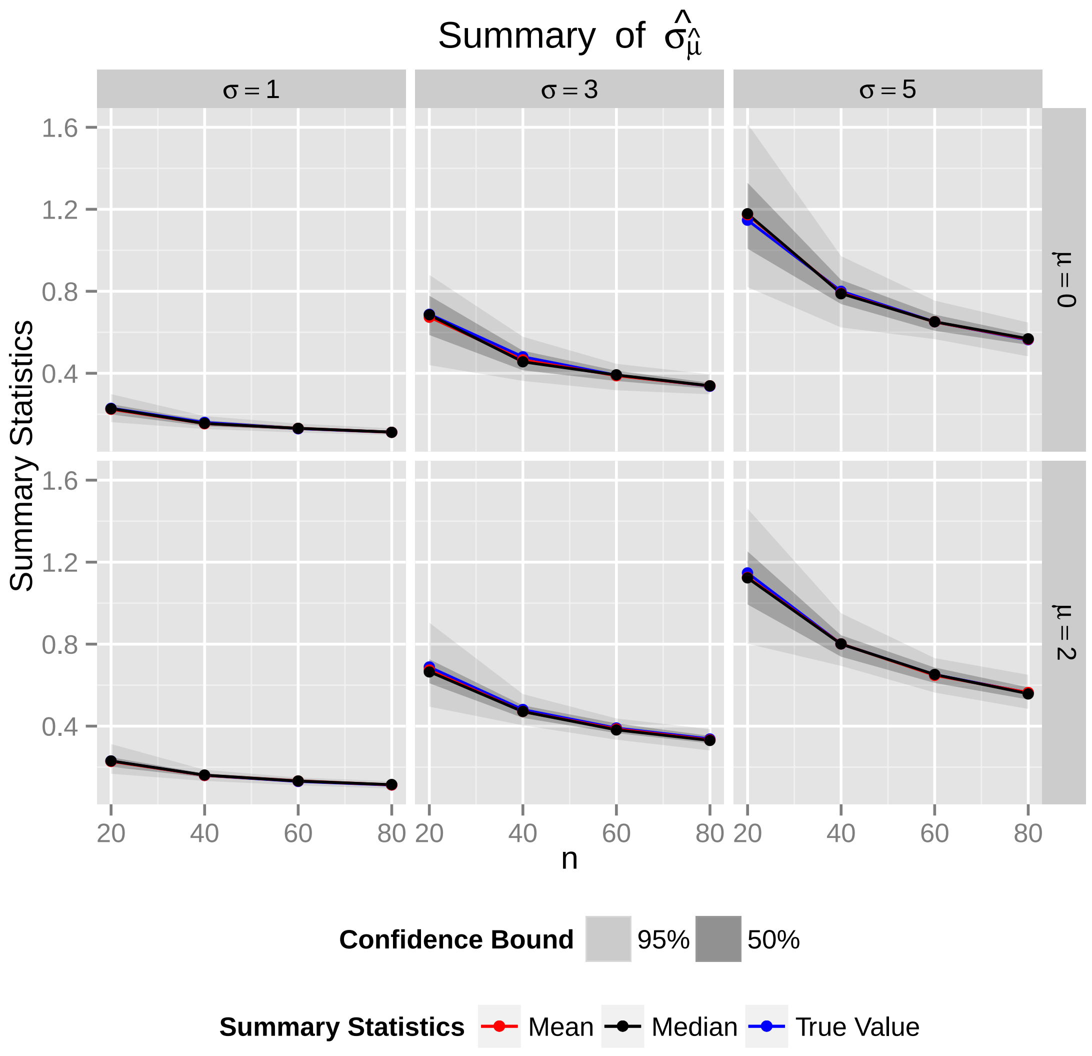
ezsim provides a handy way to run simulation and examine its results. Users dont have to work on those tedious jobs such as loop over several set of parameters, organize and summarize the simulation results,etc. Those tedious jobs are completed by ezsim. Users are only required to define some necessary information, such as data generating process, parameters and estimators. In addition, ezsim provides a flexible way to visualize the simulation results and support parallel computing. In this vignette, several examples are used to demonstrate how to create a simulation with ezsim. Our first example will give you a first glance of what ezsim can do for you. Section 2 and 3 will tell you how to use ezsim.

Suppose $x_i \dots x_n$ are drawn independently from a normal distribution with mean μ and standard deviation σ. We want to know how the sample size n, mean μ and standard deviation σ would affect the behavior of the sample mean.
We would like to replicate the simulation for 200 times. n takes value from 20,40,60,80 . μ takes value from 0,2. σ takes value from 1,3,5.

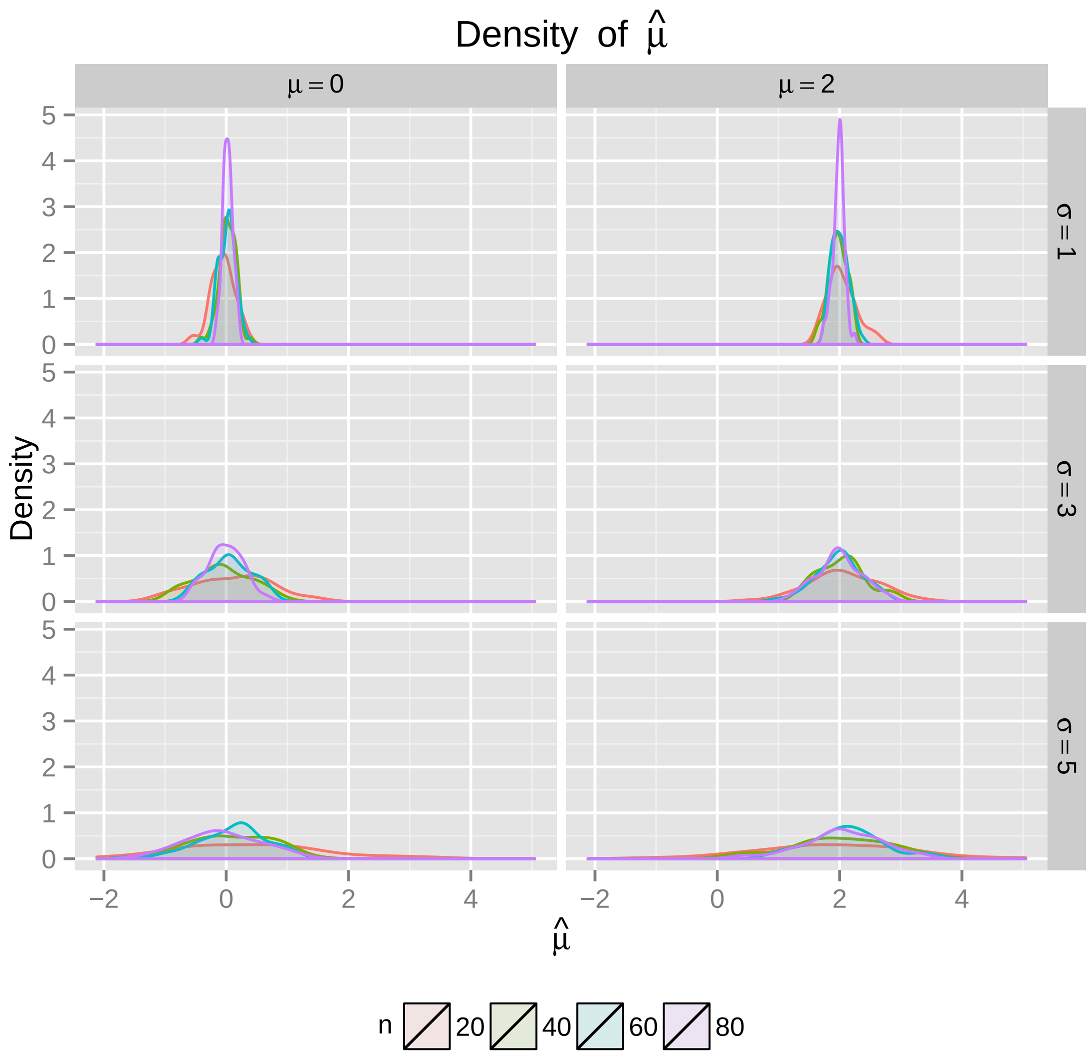
```
> library(ezsim)
> ezsim_basic<-ezsim(
+     m             = 50,
+     run           = TRUE,
+     display_name  = c(mean_hat="hat(mu)",sd_mean_hat="hat(sigma[hat(mu)])"),
+     parameter_def = createParDef(list(n=seq(20,80,20),mu=c(0,2),sigma=c(1,3,5))),
+     dgp           = function() rnorm(n,mu,sigma),
+     estimator     = function(x) c(mean_hat = mean(x),
+                                sd_mean_hat=sd(x)/sqrt(length(x)-1)),
+     true_value    = function() c(mu, sigma / sqrt(n-1))
+ )

> summary_ezsim_basic<-summary(ezsim_basic)
> head(summary_ezsim_basic,16)

   estimator  n sigma mu    Mean TV    Bias     SD   rmse BiasPercentage
1    hat(mu) 20     1  0 -0.0526  0 -0.0526 0.1970 0.2039           -Inf
2    hat(mu) 20     1  2  2.0363  2  0.0363 0.2408 0.2436         0.0181
3    hat(mu) 20     3  0  0.1089  0  0.1089 0.6329 0.6422            Inf
4    hat(mu) 20     3  2  2.0565  2  0.0565 0.5789 0.5817         0.0282
5    hat(mu) 20     5  0  0.3988  0  0.3988 1.1877 1.2528            Inf
6    hat(mu) 20     5  2  1.9030  2 -0.0970 1.2284 1.2323        -0.0485
7    hat(mu) 40     1  0  0.0232  0  0.0232 0.1438 0.1456            Inf
8    hat(mu) 40     1  2  1.9721  2 -0.0279 0.1482 0.1508        -0.0139
9    hat(mu) 40     3  0 -0.0496  0 -0.0496 0.4721 0.4747           -Inf
10   hat(mu) 40     3  2  2.0327  2  0.0327 0.4010 0.4023         0.0164
11   hat(mu) 40     5  0  0.1011  0  0.1011 0.6588 0.6665            Inf
12   hat(mu) 40     5  2  1.9972  2 -0.0028 0.8176 0.8176        -0.0014
13   hat(mu) 60     1  0  0.0139  0  0.0139 0.1395 0.1402            Inf
14   hat(mu) 60     1  2  2.0077  2  0.0077 0.1396 0.1398         0.0039
15   hat(mu) 60     3  0  0.0486  0  0.0486 0.3651 0.3683            Inf
16   hat(mu) 60     3  2  1.9917  2 -0.0083 0.3823 0.3824        -0.0042

> plot(ezsim_basic)
> plot(ezsim_basic,"density")
```



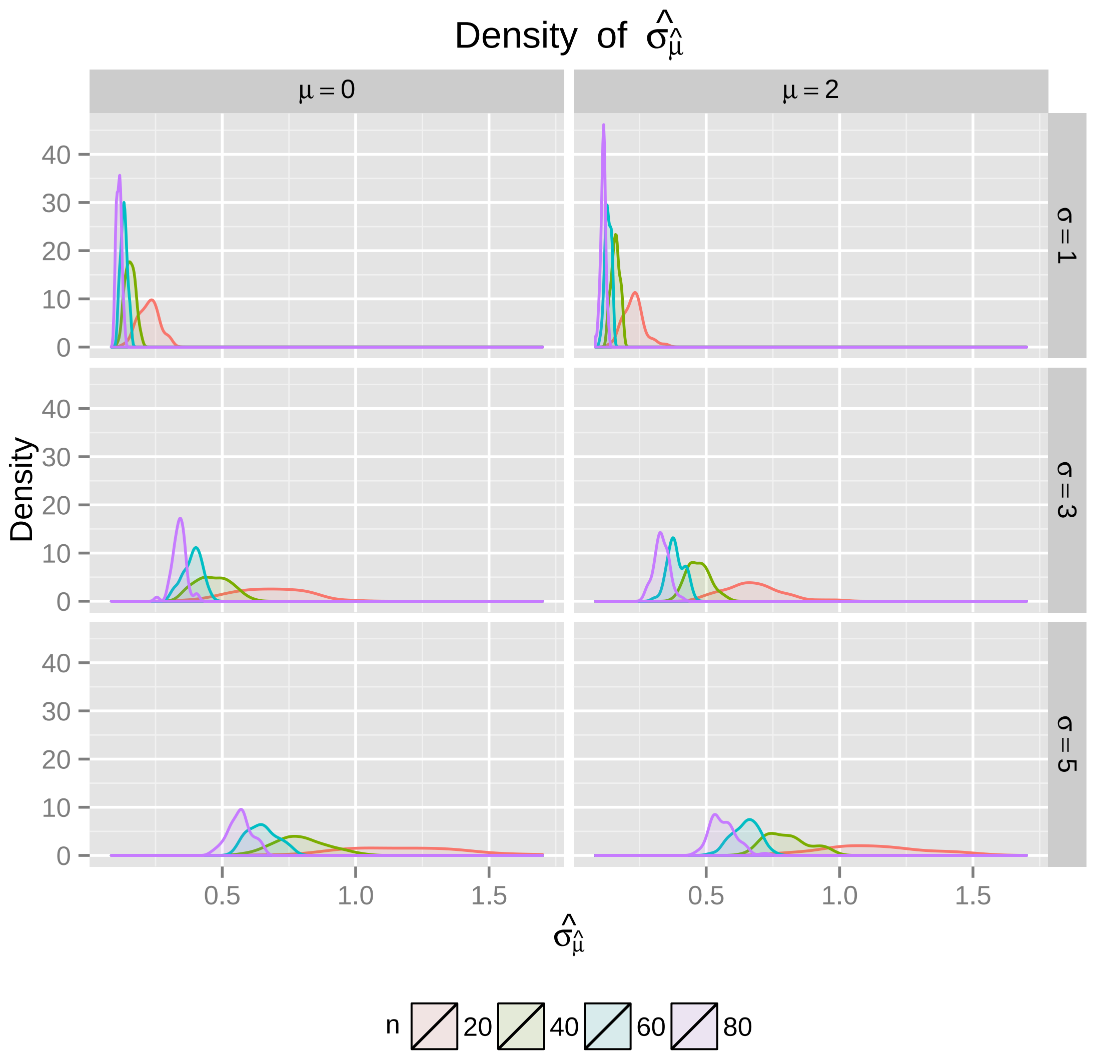
Summary of $\hat{\mu}$
$\sigma = 1$
$\sigma = 3$
$\sigma = 5$
$\mu = 0$
$\mu = 2$
Summary Statistics
n
Confidence Bound
95%
50%
Summary Statistics
Mean
Median
True Value



Summary of $\hat{\sigma}_{\hat{\mu}}$
$\sigma = 1$
$\sigma = 3$
$\sigma = 5$
$\mu = 0$
$\mu = 2$
Summary Statistics
1.6
1.2
0.8
0.4
20
40
60
80
n
Confidence Bound
95%
50%
Summary Statistics
Mean
Median
True Value



Density of $\hat{\mu}$
$\mu = 0$
$\mu = 2$
$\sigma = 1$
$\sigma = 3$
$\sigma = 5$
Density
5
4
3
2
1
0
−2
0
2
4
$\hat{\mu}$
n
20
40
60
80



2 Setup of a simulation

There are four essential components to build an ezsim object. You must specify each of them to create an `ezsim` object.

1. Number of Replication m
2. Data Generating Process (dgp) : Function for generating data.
3. Parameters : dgp takes the value of parameters to generate data.
4. Estimator : A function to perform estimation using the data generated by dgp.

Also there are optional components:

1. True Value (TV) : It computes the true value of estimates from dgp.
2. Display Name : It defines the display format of the name of estimators and parameters. See `plotmath` in R manual.
3. estimator_parser : Parse the value of the estimator.

4. run : If it is true, then the simulation will be ran right after the ezsim object is created. Otherwise, you can run it manually by `run(ezsim_basic)`. Default is `TRUE`.
5. run_test : Whether to perform a test before the simulation
6. auto_save : Number of auto save during the simulation, default is 0.
7. use_seed : The seed to be used in the simulation. If `use_core=1`, `set.seed(use_seed)` will be called. If `use_core=>1` and `cluster=NULL`, `clusterSetRNGStream(cluster,use_seed)` will be used. Ignored if `use_core=>1` and `cluster` is provided.
8. use_core : The number of CPU core to be used in the simulation.
9. cluster_packages : Names of the packages to be loaded in the cluster.
10. cluster : cluster for parallelization. If it is NULL, a cluster with `use_code` cores will be created automatically (will be removed after the simulation)

If you dont specify the value of `True Value`, the value of `bias` and `rmse` will also be `NA`.

2.1 Parameters

In ezsim, parameters are generated by parameterDef object. To create a parameterDef object, we can use the function createParDef. It takes 2 auguments, `selection`(the first argument) and `banker`. `selection` are parameters may vary in the parameters set. Any vectors or matrix are regarded as a sequence of the same parameter. `banker` are fixed parameters in the parameters set. It can be any data type.
In our example, all parameters are scalars. We can create a parameterDef object by:

```
> par_def<-createParDef(selection=list(n=seq(20,80,20),mu=c(0,2),sigma=c(1,3,5)))
> par_def

Selection Parameters:
$n
[1] 20 40 60 80

$mu
[1] 0 2

$sigma
[1] 1 3 5



Banker Parameters:
list()
```

Since we have 4 different values of n, 2 different values of μ and 3 different values of σ, there is total of $4 \times 3 \times 2 = 24$ possible combination of parameter sets. If we want to have a look of the generated parameters, we can use the function `generate`. It will return a list of parameter sets. (Only the first three will be shown in the example)

```
> generate(par_def)[1:3]

[[1]]
n=20, mu=0, sigma=1

[[2]]
n=40, mu=0, sigma=1

[[3]]
n=60, mu=0, sigma=1
```

`setSelection` and `setBanker` change the value of a parameterDef object. Different from `createParDef`, the parameters dont have to be store in a list.
Example: Suppose we want to generate n sample from a bivariate normal distribution with parameter μ_1, μ_2 and a variance-covraiance matrix Σ.

```
> par_def2<-createParDef(selection=list(mu1=5,mu2=3,n=c(10,20)),
+     banker=list(Sigma=matrix(c(1,.4,.4,1),nrow=2)))
> generate(par_def2)

[[1]]
Sigma  :
     [,1] [,2]
[1,]  1.0  0.4
[2,]  0.4  1.0

mu1=5, mu2=3, n=10

[[2]]
Sigma  :
     [,1] [,2]
[1,]  1.0  0.4
[2,]  0.4  1.0

mu1=5, mu2=3, n=20
```

2.2 Data Generating Process

The Data Generating Process generates the simulated data for `estimator` to compute the estimates. Inside this function, you can call any parameters directly. It must be a function.
In our example, the data generating process very is simple. It generate a vector of normal random variables with length n, mean μ and sd σ.

```
> dgp<-function(){
+         rnorm(n,mu,sigma)
+ }

> evalFunctionOnParameterDef(par_def,dgp,index=1)

 [1] -1.42948004  0.81997891 -1.13389009  1.11679822 -0.20629776 -1.03542477
 [7] -0.35962843 -0.35871067  0.99472509  0.17643160  2.09609300 -0.61966120
[13]  1.44681161  0.03750141 -0.67544571 -0.50322569 -0.04233521 -1.44715001
[19]  0.63792223 -0.44650014

> evalFunctionOnParameterDef(par_def,dgp,index=2)

 [1] -0.71822821 -0.28196851 -0.68754567 -0.68598706 -0.17958990  0.01389804
 [7]  0.95961557  0.37254580 -0.49191604 -0.58761360  0.61430317  0.34793092
[13]  0.69975428  0.77448065  0.92064450 -0.16118671  0.29057362 -0.24011791
[19] -0.39990023  1.37094890 -0.12504134  0.91386535 -0.60162751 -0.32373655
[25]  0.33775312  1.41183313  2.76016473 -0.58319669 -0.68167304  0.71624888
[31] -0.61333658  0.22946517  0.68528414  0.77646478 -0.27939038  0.17440808
[37] -1.09853606  0.59943025  0.64008191 -1.55105907

>

> dgp_2<-function(){
+     z1<-rnorm(n)
+     z2<-rnorm(n)
+     cbind(x1=mu1+z1*Sigma[1,1],
+           x2=mu2+ Sigma[2,2]*(Sigma[1,2]*z1+ sqrt(1-Sigma[1,2]^2)*z2 ))
+ }
> evalFunctionOnParameterDef(par_def2,dgp_2)
```

```
            x1       x2
 [1,] 5.561348 3.855970
 [2,] 4.878589 3.506899
 [3,] 4.845081 3.347011
 [4,] 4.935306 1.201829
 [5,] 4.713639 2.475914
 [6,] 5.526948 2.452217
 [7,] 5.005070 3.293616
 [8,] 6.067454 4.311427
 [9,] 6.739886 5.392706
[10,] 4.411689 2.261345
```

```
>
```

2.3 Estimators

It computes the estimates from the data generated by `dgp`. The return value of estimators must be a numeric vector. Dont forget to specify the name of estimators. You can use the `evalFunctionOnParameterDef` function to test whether the function work properly. It must be a function.

```
> estimator<-function(x){
+     c(mean_hat = mean(x), sd_mean_hat=sd(x)/sqrt(length(x)-1))
+ }
> estimator(evalFunctionOnParameterDef(par_def,dgp,index=1))
```

```
   mean_hat sd_mean_hat
 -0.3744250   0.2099226
```

2.4 True Value

It computes the true value of estimates from dgp. The return value should have same length as the estimators. Also, the position of return value should match with estimators. Similar to `dgp`, You can call any parameters within this function. It can be a function or `NA`(bias and rmse will also be `NA`).

```
> true<-function(){
+     c(mu, sigma / sqrt(n-1))
+ }
> evalFunctionOnParameterDef(par_def,true)
```

```
[1] 0.0000000 0.2294157
```

2.5 Display Name

It defines the display format of the name of estimators and parameters. For example, you can set the display name of ”mean_hat” to ”hat(mu)”. See `plotmath` for details.

```
> display_name<-c(mean_hat="hat(mu)",sd_mean_hat="hat(sigma[hat(mu)])")
```

2.6 estimator_parser

Sometimes the estimator may return an object instead of a vector, estimator_parser can be specified to extract relevant information from from the return value of estimator. It took the return value of `estimator` as argument and return a new value. For example, if the estimator returns an `lm` object, then estimator_parser can be `coef`.

```
> estimator_lm <- function(x) {
+     lm(y~x1+x2, data=x)
+ }
> estimator_parser_lm <- function(x){
+     coef(x)
+ }
```

if `estimator_parser` is not a function, it will be replaced by *"function(x) x"* which mean it is not going to change anything.

3 Run a simulation

3.1 Run it now or run it later

You dont have run the simulation right after the creation of an ezsim object. You can set `run=FALSE` to avoid running the simulation in ezsim. You can call `run()` to run the simulation later. For example,

```
> library(ezsim)
> ezsim_basic<-ezsim(
+     m             = 50,
+     run           = FALSE,
+     display_name  = c(mean_hat="hat(mu)",sd_mean_hat="hat(sigma[hat(mu)])"),
+     parameter_def = createParDef(list(n=seq(20,80,20),mu=c(0,2),sigma=c(1,3,5))),
+     dgp           = function() rnorm(n,mu,sigma),
+     estimator     = function(x) c(mean_hat = mean(x),
+                                  sd_mean_hat=sd(x)/sqrt(length(x)-1)),
+     true_value    = function() c(mu, sigma / sqrt(n-1))
+ )
> ezsim_basic = run(ezsim_basic)
```

3.2 Run a test

if `run_test=TRUE`, then ezsim will run the `estimator` and `dgp` once for each combination of parameters. You dont want to see an error message pop up after 1 hour of simulation because some of parameters are problematic. It is TRUE by default.

3.3 autosave

if autosave > 0, the corresponding number of temporary object of ezsim will be stored in current path. It aims to avoid losting everything if the program is halted before the simulation is completed (e.g. computer crashed). However, if this happened, we wont able to replicate the same simulation using the same seed.

3.4 Parallel Computing

3.4.1 Automatically generated cluster

`ezsim` implements Parallelization by using functionality of `snow` in `parallel`. You can simply set the value of `use_core` to use more than 1 CPU cores. `ezsim` will create clusters with `use_core` cores and distribute the simulation to elements of cluster. If the simulation require some packages, you have to specify them in `cluster_packages`. The cluster will be deleted after the simulation. It is as simple as:

```
> library(ezsim)
> ezsim_basic<-ezsim(
+     m             = 50,
+     run           = TRUE,
+     use_core      = 4,
+     use_seed        = 123,
+     cluster_packages = "Jmisc",
+     display_name  = c(mean_hat="hat(mu)",sd_mean_hat="hat(sigma[hat(mu)])"),
+     parameter_def = createParDef(list(n=seq(20,80,20),mu=c(0,2),sigma=c(1,3,5))),
+     dgp           = function() rnorm(n,mu,sigma),
+     estimator     = function(x) c(mean_hat = mean(x),
+                                  sd_mean_hat=sd(x)/sqrt(length(x)-1)),
+     true_value    = function() c(mu, sigma / sqrt(n-1))
+ )
> summary(ezsim_basic)
```

If the simulation require some packages, you have to specify them in `cluster_packages`. Notice that it took a fixed amount of time to create the cluster, simulation may take longer time if `m` is too small.

3.4.2 Customized cluster

If you want to customize the cluster, you can make your own cluster and give it to ezsim. **ezsim will not change the RNG for the cluster or delete the cluster before or after the simulation. You will have to do it by yourself**

```
> library(parallel)
> my_cluster = makeCluster(4)
> clusterSetRNGStream(my_cluster, 123)
> library(ezsim)
> ezsim_basic<-ezsim(
+     m             = 50,
+     run           = TRUE,
+     use_core      = 4,
+     cluster       = my_cluster,
+     display_name  = c(mean_hat="hat(mu)",sd_mean_hat="hat(sigma[hat(mu)])"),
+     parameter_def = createParDef(list(n=seq(20,80,20),mu=c(0,2),sigma=c(1,3,5))),
+     dgp           = function() rnorm(n,mu,sigma),
+     estimator     = function(x) c(mean_hat = mean(x),
+                                  sd_mean_hat=sd(x)/sqrt(length(x)-1)),
+     true_value    = function() c(mu, sigma / sqrt(n-1))
+ )
> stopCluster(my_cluster)
> summary(ezsim_basic)
```

3.5 use_seed

To ensure the simulation is reproduciable, you can specify a seed for the simulation. There are three possible scenarios for the usage:

1. If use_core=1, `set.seed(use_seed)` will be called right before the simulation start.
2. If `use_core=>1` and `cluster=NULL`, `clusterSetRNGStream(cluster,use_seed)` will be called.
3. If `use_core=>1` and `cluster`, it will be ignored. Probably you would like to control the RNG for the cluster by yourself

4 Summary of the simulation result

4.1 Summary Table

You can create a summary table by `summary` . The default summary statistics include mean, true value, bias, standard deviation, root mean square error and p-value of Jarque-Bera test. See section 1 for example.

4.1.1 Subset of the Summary Table

You can select a subset of parameters and estimators to compute the summary statistics.

```
> summary(ezsim_basic,subset=list(estimator="mean_hat",n=c(20,40),sigma=c(1,3)))

  estimator  n sigma mu    Mean TV    Bias     SD   rmse BiasPercentage
1   hat(mu) 20     1  0 -0.0526  0 -0.0526 0.1970 0.2039           -Inf
2   hat(mu) 20     1  2  2.0363  2  0.0363 0.2408 0.2436         0.0181
3   hat(mu) 20     3  0  0.1089  0  0.1089 0.6329 0.6422            Inf
4   hat(mu) 20     3  2  2.0565  2  0.0565 0.5789 0.5817         0.0282
5   hat(mu) 40     1  0  0.0232  0  0.0232 0.1438 0.1456            Inf
6   hat(mu) 40     1  2  1.9721  2 -0.0279 0.1482 0.1508        -0.0139
7   hat(mu) 40     3  0 -0.0496  0 -0.0496 0.4721 0.4747           -Inf
8   hat(mu) 40     3  2  2.0327  2  0.0327 0.4010 0.4023         0.0164
```

4.1.2 More Summary Statistics

If you want to have more summary statistics, you can set `simple=FALSE` in the argument. Then the summary statistics will also include: percentage of bias, minimum, first quartile, median, third quartile and maximum.

```
> summary(ezsim_basic,simple=FALSE,
+         subset=list(estimator="mean_hat",n=c(20,40),sigma=c(1,3)))

  estimator  n sigma mu    Mean TV    Bias BiasPercentage     SD   rmse     Min
1   hat(mu) 20     1  0 -0.0526  0 -0.0526           -Inf 0.1970 0.2039 -0.5711
2   hat(mu) 20     1  2  2.0363  2  0.0363         0.0181 0.2408 0.2436  1.6266
3   hat(mu) 20     3  0  0.1089  0  0.1089            Inf 0.6329 0.6422 -1.1185
4   hat(mu) 20     3  2  2.0565  2  0.0565         0.0282 0.5789 0.5817  0.5337
5   hat(mu) 40     1  0  0.0232  0  0.0232            Inf 0.1438 0.1456 -0.4157
6   hat(mu) 40     1  2  1.9721  2 -0.0279        -0.0139 0.1482 0.1508  1.6497
7   hat(mu) 40     3  0 -0.0496  0 -0.0496           -Inf 0.4721 0.4747 -0.9369
8   hat(mu) 40     3  2  2.0327  2  0.0327         0.0164 0.4010 0.4023  1.3168
      Q25  Median    Q75    Max JB_test
1 -0.2090 -0.0616 0.0500 0.3737  0.8335
2  1.8915  1.9957 2.2043 2.6512  0.2405
3 -0.3511  0.1583 0.5363 1.4314  0.6781
4  1.6990  2.0786 2.5023 3.3891  0.9556
5 -0.0483  0.0302 0.1249 0.3977  0.1825
6  1.8636  1.9750 2.0865 2.2185  0.6087
7 -0.3275 -0.0700 0.3511 0.9602  0.5667
8  1.7617  2.0758 2.2705 2.9215  0.5566
```

4.1.3 Customize the Summary Statistics

You can choose a subset of summary statistics by specifying value in `stat`. Also you can define your own summary statistics. `value_of_estimator` is the value of estimator and `value_of_TV` is the value of true value.

```
> summary(ezsim_basic,stat=c("q25","median","q75"),
+         Q025=quantile(value_of_estimator,0.025),
+         Q975=quantile(value_of_estimator,0.975),
+         subset=list(estimator="mean_hat",n=c(20,40),sigma=c(1,3)))

  estimator  n sigma mu     Q25  Median    Q75    Q025   Q975
1   hat(mu) 20     1  0 -0.2090 -0.0616 0.0500 -0.4778 0.2981
2   hat(mu) 20     1  2  1.8915  1.9957 2.2043  1.6610 2.5713
3   hat(mu) 20     3  0 -0.3511  0.1583 0.5363 -0.9977 1.3709
```

```
4    hat(mu) 20      3  2  1.6990  2.0786 2.5023  1.0549 3.0859
5    hat(mu) 40      1  0 -0.0483  0.0302 0.1249 -0.2644 0.2168
6    hat(mu) 40      1  2  1.8636  1.9750 2.0865  1.6630 2.2082
7    hat(mu) 40      3  0 -0.3275 -0.0700 0.3511 -0.8282 0.7103
8    hat(mu) 40      3  2  1.7617  2.0758 2.2705  1.3656 2.8976
```

4.2 Plotting the simulation

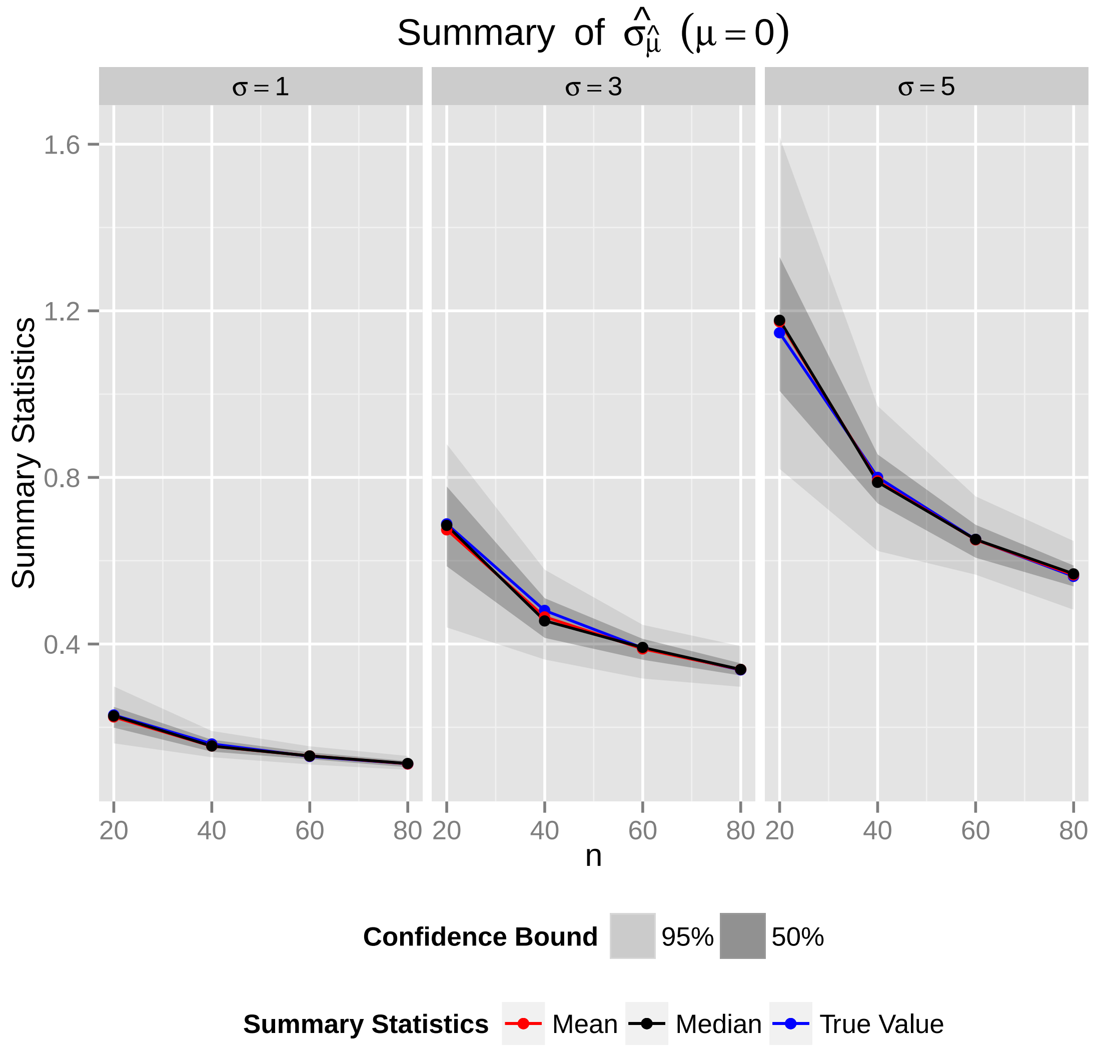
4.2.1 Plotting an ezsim object

The plot contains the mean, median, true value , 2.5th, 25th, 75th and 97.5th percentile of the estimator. The mean, median, true value are plotted as black, blue and red line respectively. 2.5th and 97.5th percentile form a 95% confidence bound and 25th and 75th percentile form a 50% confidence bound. x-axis of the plot will be the parameter take the most number of value (`n` in our example). Rest of them will be facets of the plot. Each estimator will occupy one plot. See section 1 for examples.

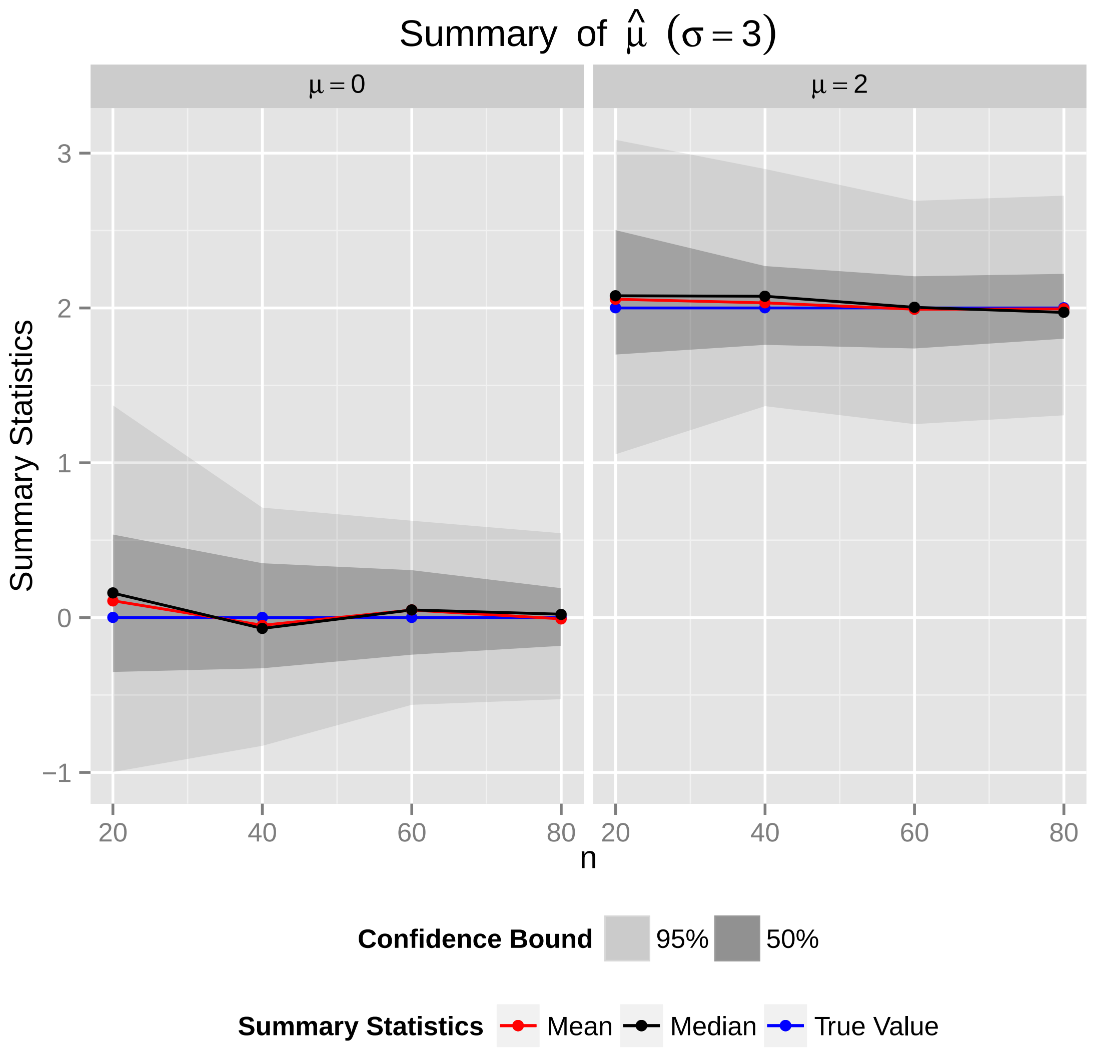
4.2.2 Subset of the Plot

The usage of `subset` is similar to `summary`. You can select a subset of `estimators` and \or `parameters`.

```
> plot(ezsim_basic,subset=list(estimator="sd_mean_hat",mu=3))
```



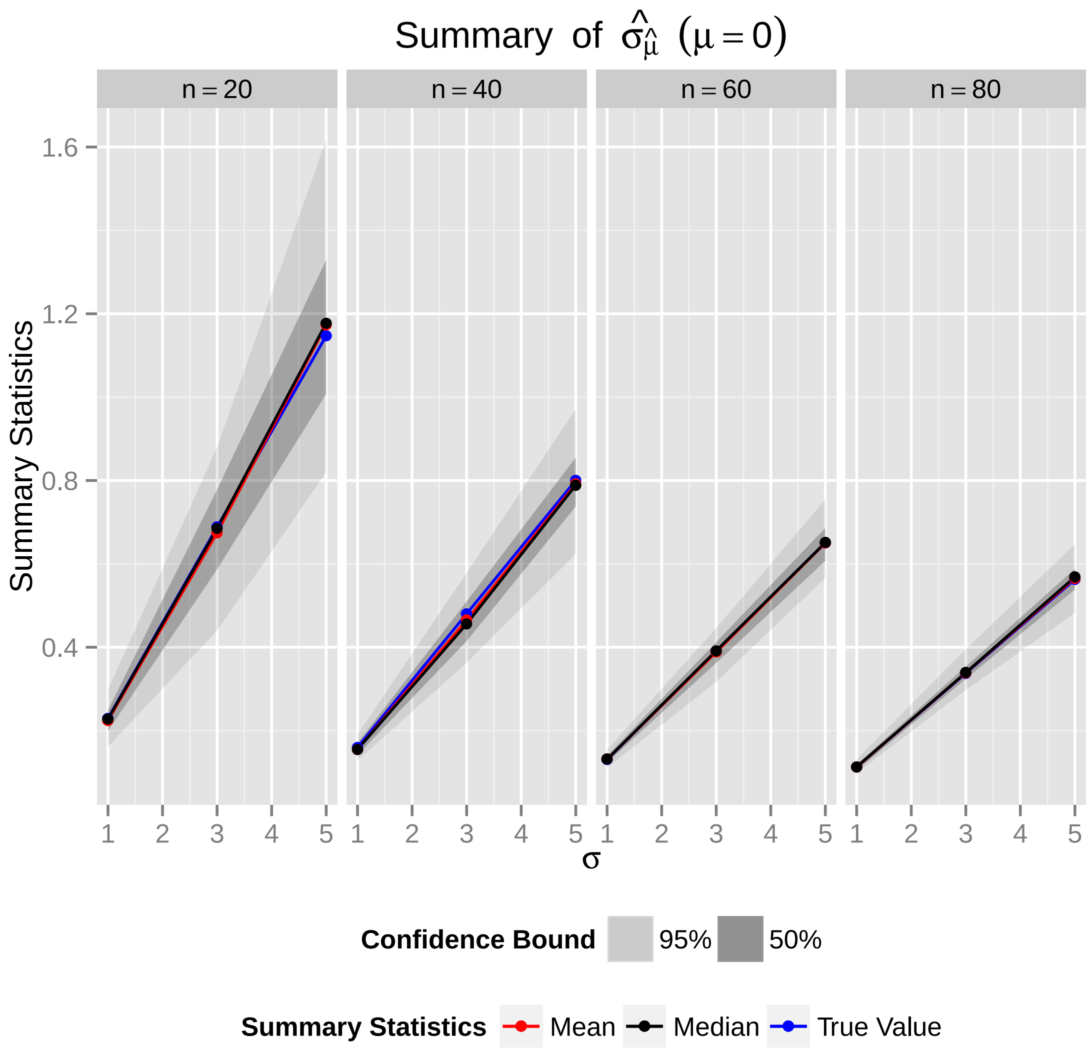
```
> plot(ezsim_basic,subset=list(estimator="mean_hat",sigma=3))
```



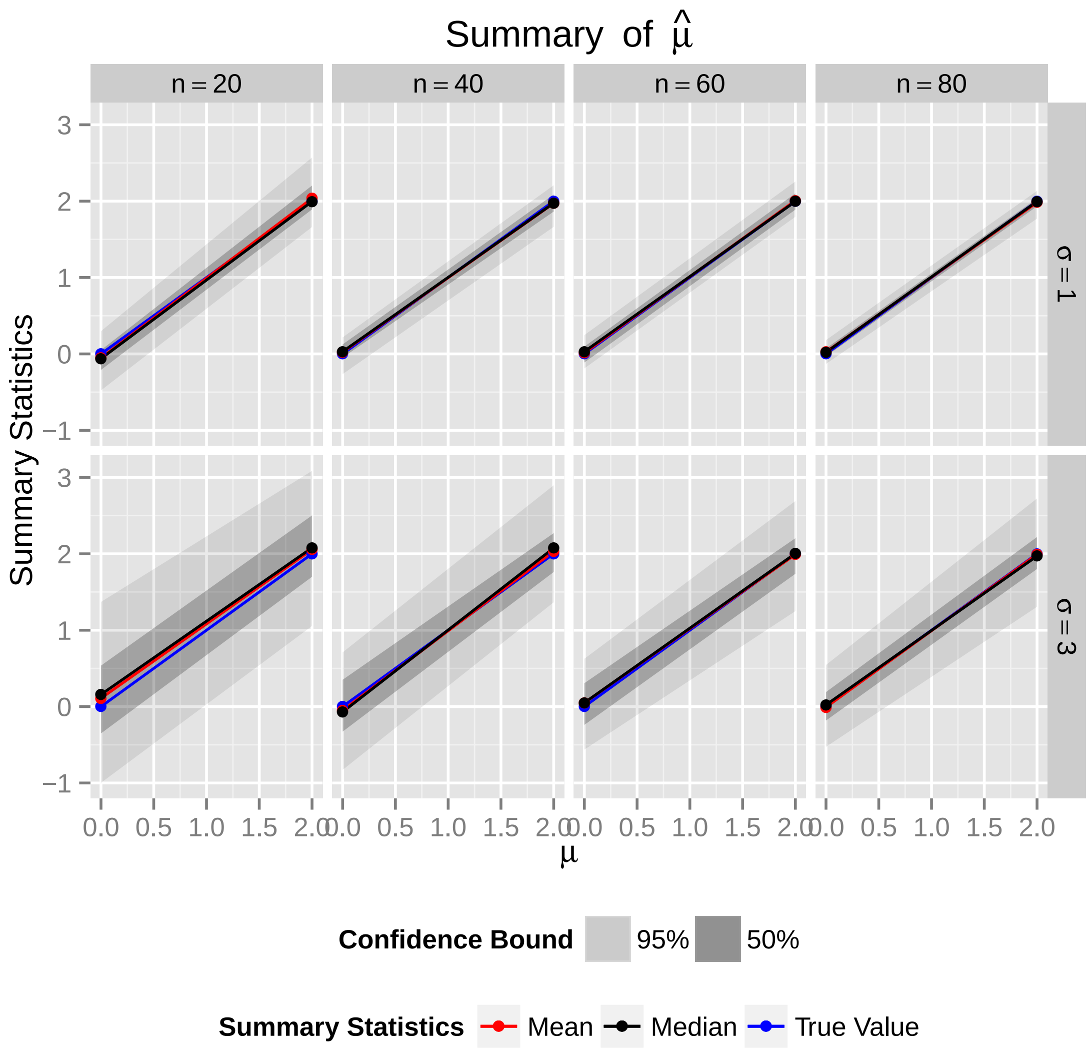
4.2.3 Parameters Priority of the Plot

The default priority of parameters is sorted by the number of value of each parameter(more to less). You can reset it by `parameter_priority`. The first parameter will have the highest priority(shown in the x-axis). You dont have to specify all parameters, the rest of them are sorted by the number of value of each of them.

```
> plot(ezsim_basic,subset=list(estimator="sd_mean_hat",mu=0),
+ parameters_priority=c("sigma","n"))
```



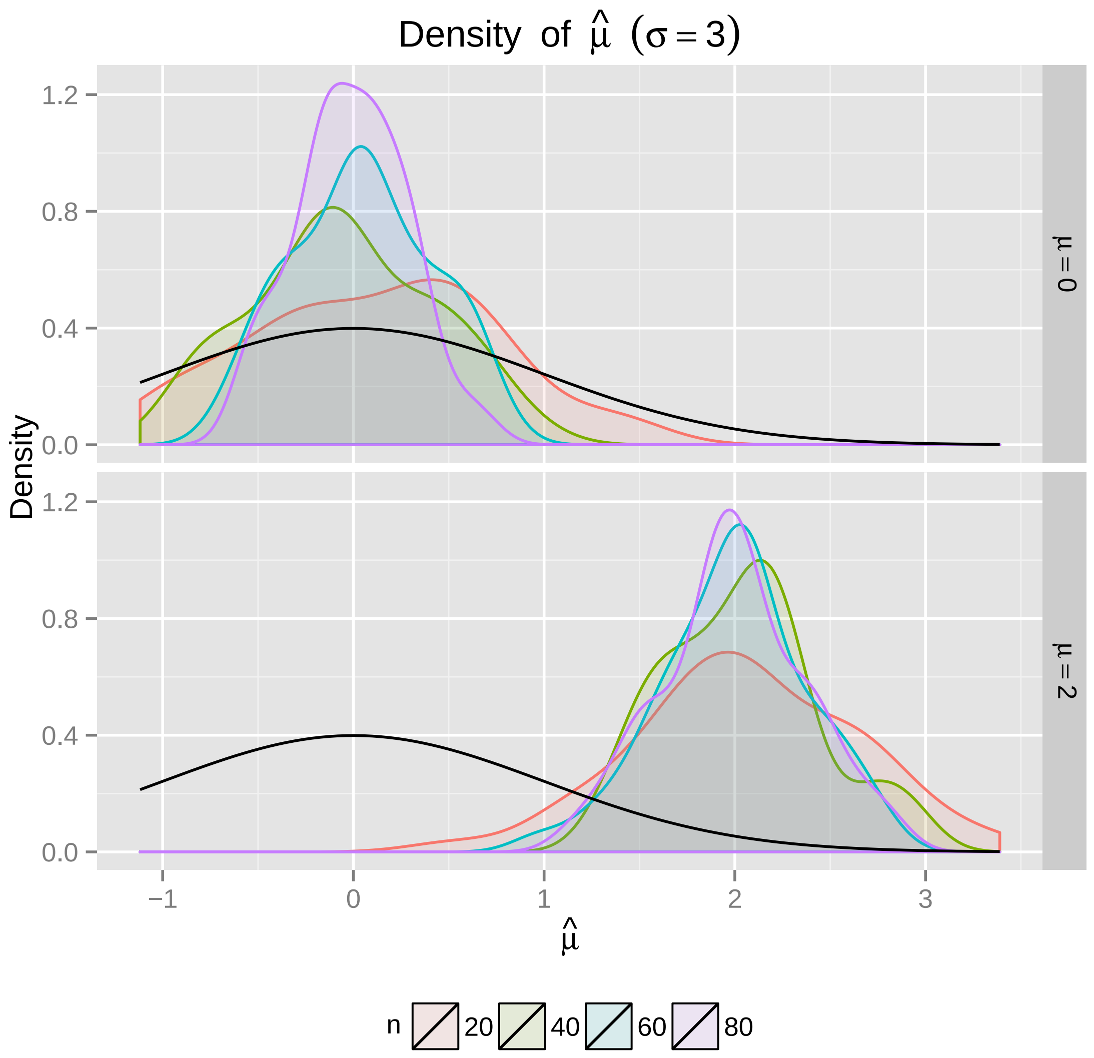
```
> plot(ezsim_basic,subset=list(estimator="mean_hat",sigma=c(1,3)),parameters_priority="mu")
```



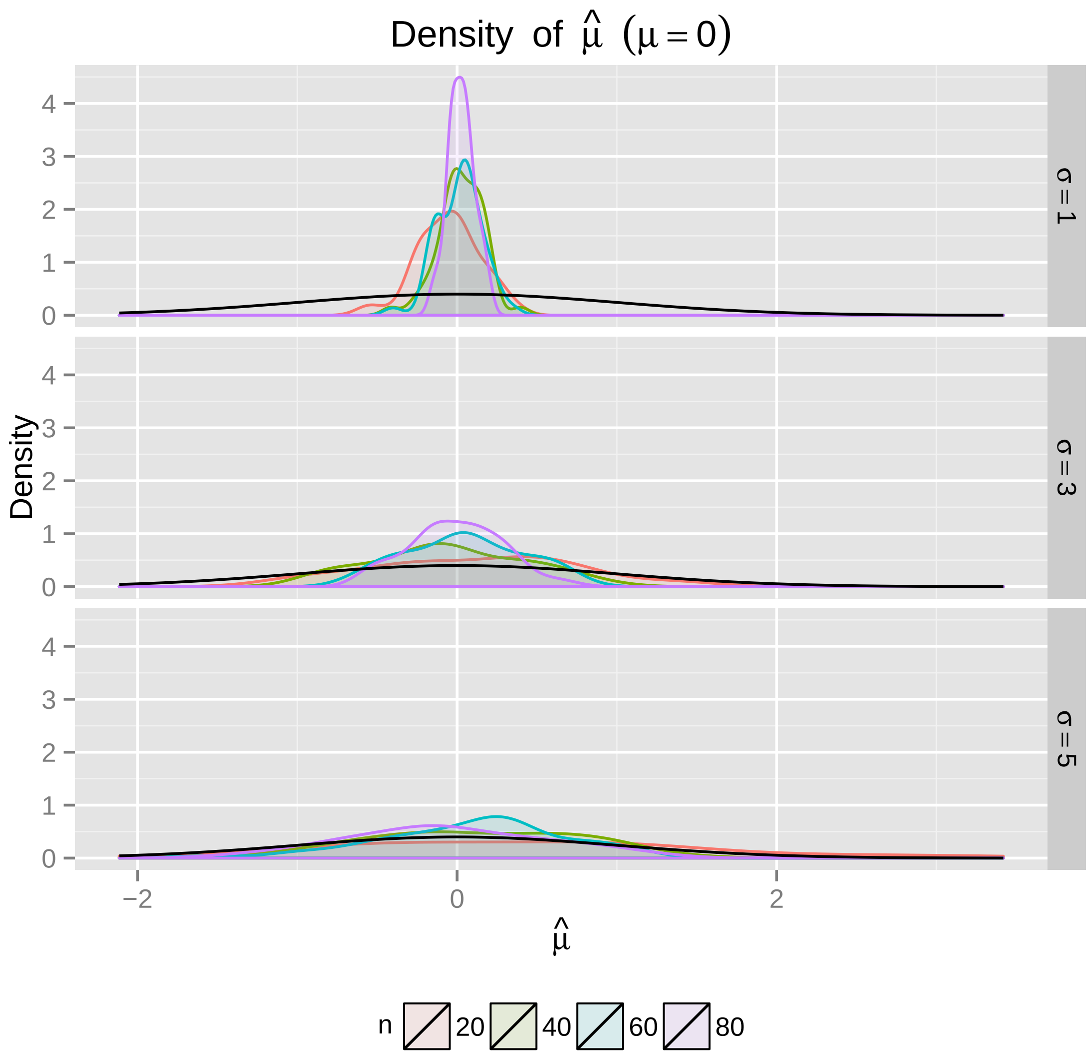
4.2.4 Density Plot

Plot the density funtcion of the estimator. `subset` and `parameter_priority` are valid for density plot. You can specify `benchmark=dnorm` by adding a density of the standard normal distribution. `dorm` can be replaced by other density function. See section 1 for examples.

```
> plot(ezsim_basic,"density",
+       subset=list(estimator="mean_hat",sigma=3),
+       parameters_priority="n",benchmark=dnorm)
```

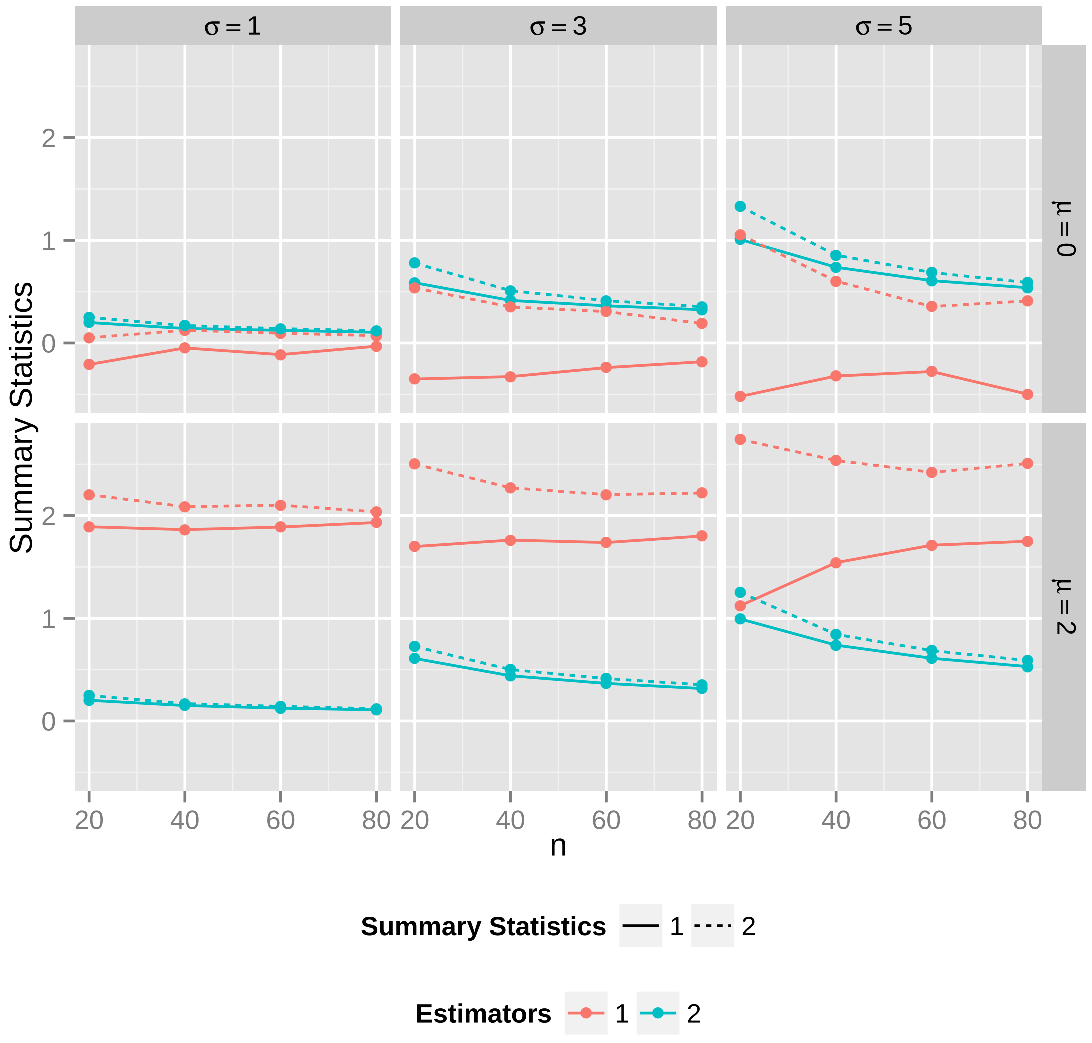


```
> plot(ezsim_basic,"density",
+       subset=list(estimator="mean_hat",mu=0),
+       parameters_priority="n" ,benchmark=dnorm)
```

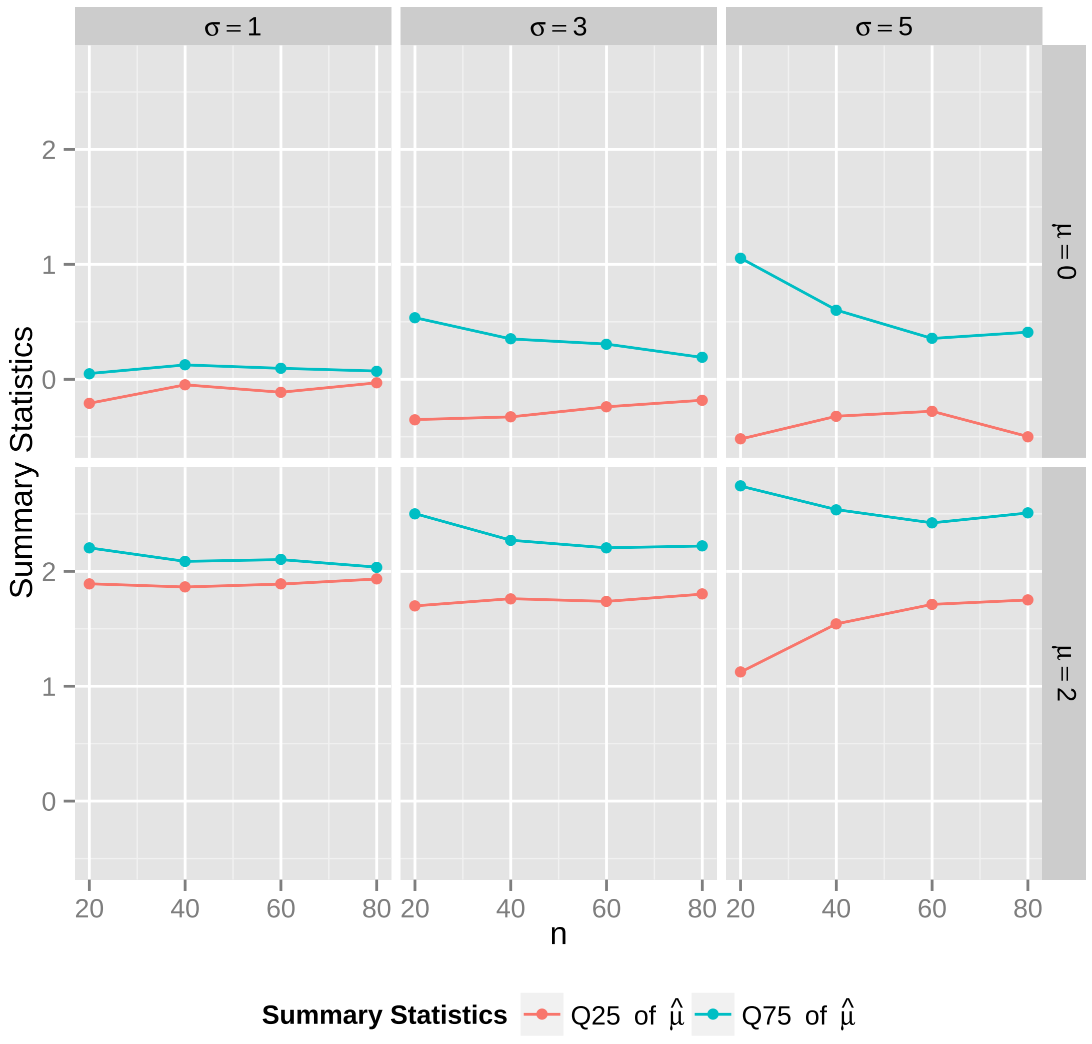


4.2.5 Plot the summary ezsim

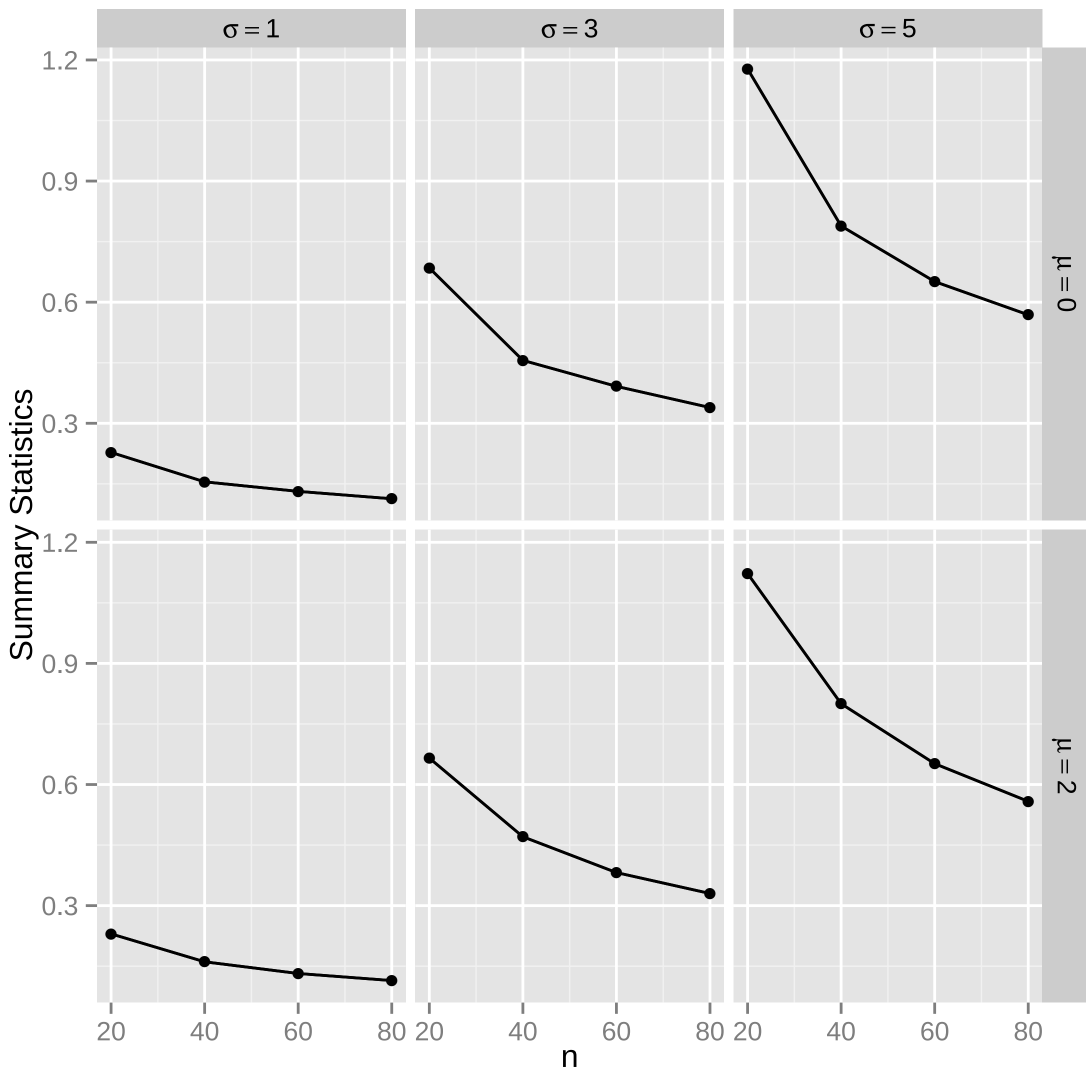
```
> plot(summary(ezsim_basic,c("q25","q75")))
```



```
> plot(summary(ezsim_basic,c("q25","q75"),subset=list(estimator="mean_hat")))
```



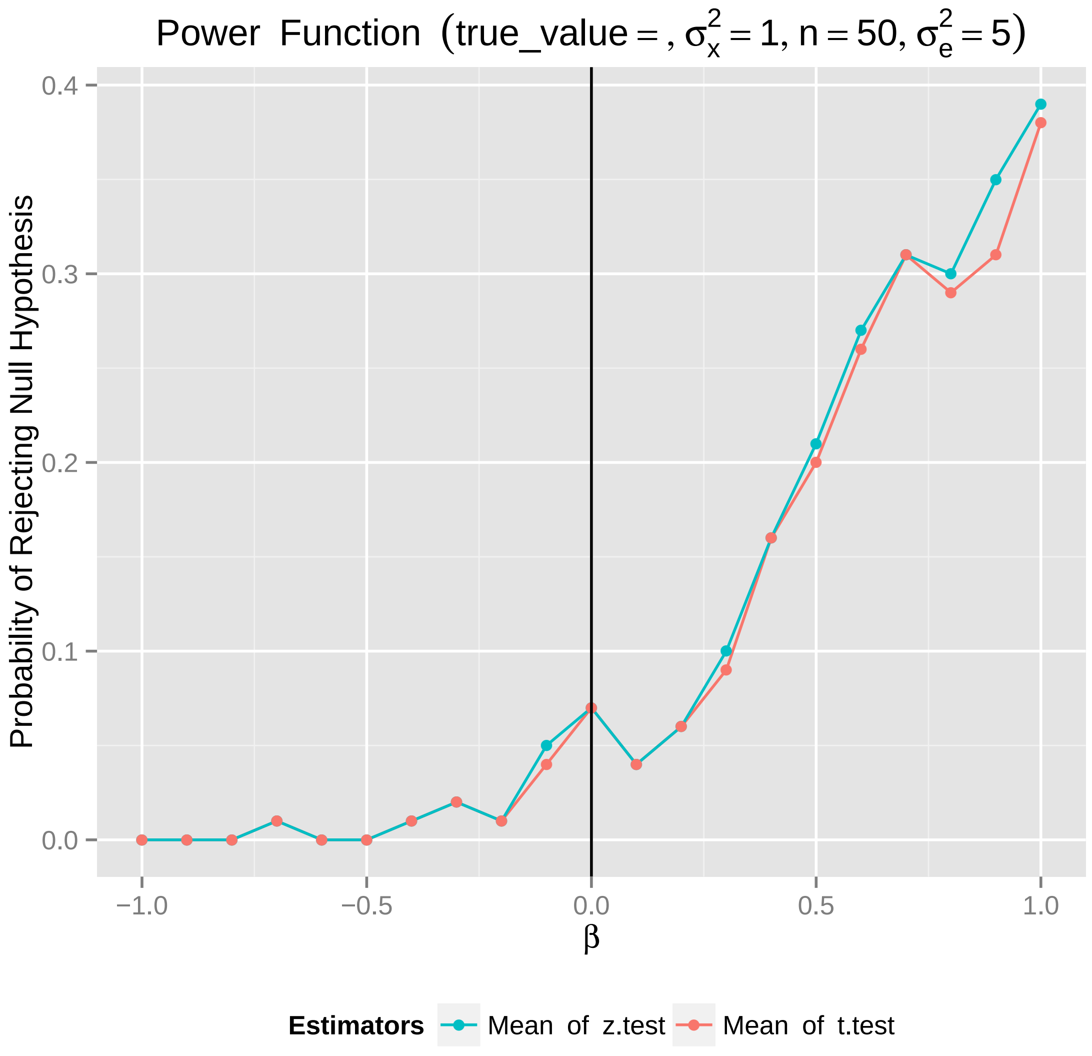
```
> plot(summary(ezsim_basic,c("median"),subset=list(estimator="sd_mean_hat")))
```



4.2.6 Plot the Power Function

If the estimator is an indicator of rejecting a null hypothesis(0: fail to reject null hypothesis; 1: reject null hypothesis), then we can plot the power function. A vertical line will be drawn if `null_hypothesis` is specified. The intersection of hte vertical line(value of null hypothesis) and the power function is the size of the test. The following example shows the power function of testing whether the coefficient of a linear model is larger than one with t-test and z-test.

```
> ez_powerfun<-ezsim(
+     m             = 100,
+     run           = TRUE,
+     display_name  = c(b="beta",es="sigma[e]^2",xs="sigma[x]^2"),
+     parameter_def = createParDef(selection=list(xs=1,n=50,es=5,b=seq(-1,1,0.1))),
+     dgp           = function(){
+                         x<-rnorm(n,0,xs)
+                         e<-rnorm(n,0,es)
+                         y<-b * x + e
+                         data.frame(y,x)
+                     },
+     estimator     = function(d){
+                         r<-summary(lm(y~x-1,data=d))
+                         stat<-r$coef[,1]/r$coef[,2]
+
+                         # test whether b > 0
+                         # level of significance : 5%
+                         out <- stat > c(qnorm(.95), qt(0.95,df=r$df[2]))
+                         names(out)<-c("z-test","t-test")
+                         out
+                     }
+ )

> plot(ez_powerfun,"powerfun",null_hypothesis=0)
```


Power Function (true_value = , $\sigma_x^2 = 1$, n = 50, $\sigma_e^2 = 5$)
Probability of Rejecting Null Hypothesis
β
Estimators
Mean of z.test
Mean of t.test