A Vignette for the R package: ezsim

TszKin Julian Chan <ctszkin@gmail.com>

July 23, 2011

Contents

1 Introduction **1**

2 Pre-simulation **6**
2.1 Parameters 7
2.2 Data Generating Process 8
2.3 Estimators 9
2.4 True Value 9
2.5 Display Name 10

3 Post-simulation **10**
3.1 Summary Table 10
3.1.1 Subset of the Summary Table 10
3.1.2 More Summary Statistics 10
3.1.3 Customize the Summary Statistics 11
3.2 Plotting the simulation 11
3.2.1 Plotting an ezsim object 11
3.2.2 Subset of the Plot 11
3.2.3 Parameters Priority of the Plot 13
3.2.4 Density Plot 15
3.2.5 Plot the summary ezsim 17
3.2.6 Plot the Power Function 20

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.

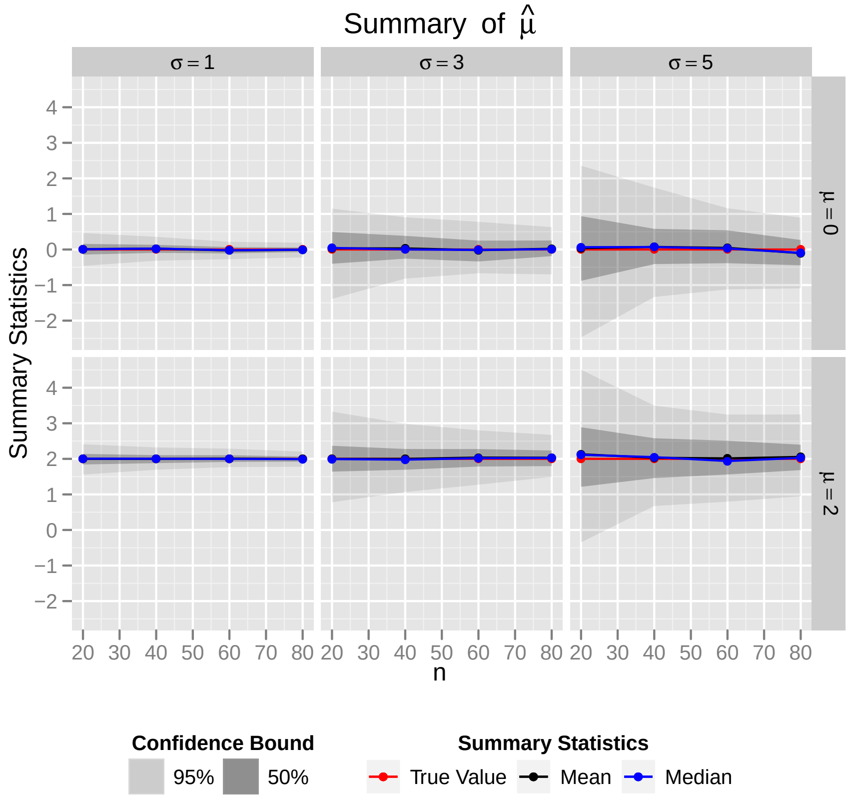
We would like to replicate the simulation for 200 times. n takes value from 20,40,60,80 . μ takes value from 0,2. σ takes value from 1,3,5.

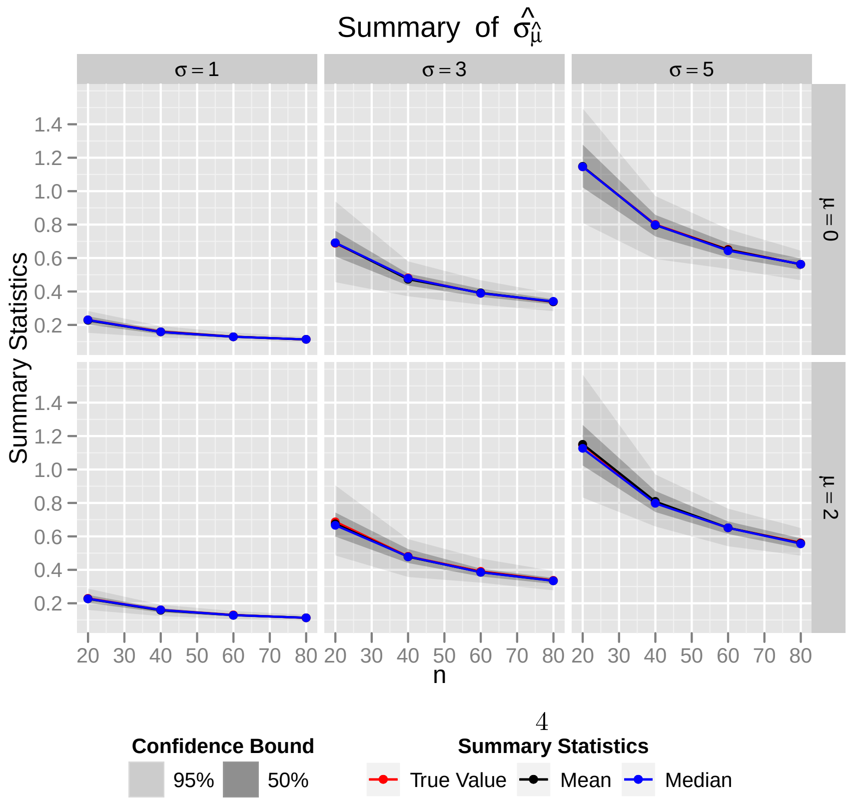
```
> library(ezsim)
> ezsim_basic<-ezsim(
+     m             = 200,
+     run           = TRUE,
+     core          = 1,
+     display_name  = c(mean_hat="hat(mu)",sd_mean_hat="hat(sigma[hat(mu)])"),
+     parameter_def = createParDef(list(n=seq(20,80,20),mu=c(0,2),sigma=c(1,3,5))),
+     dgp           = 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)
```

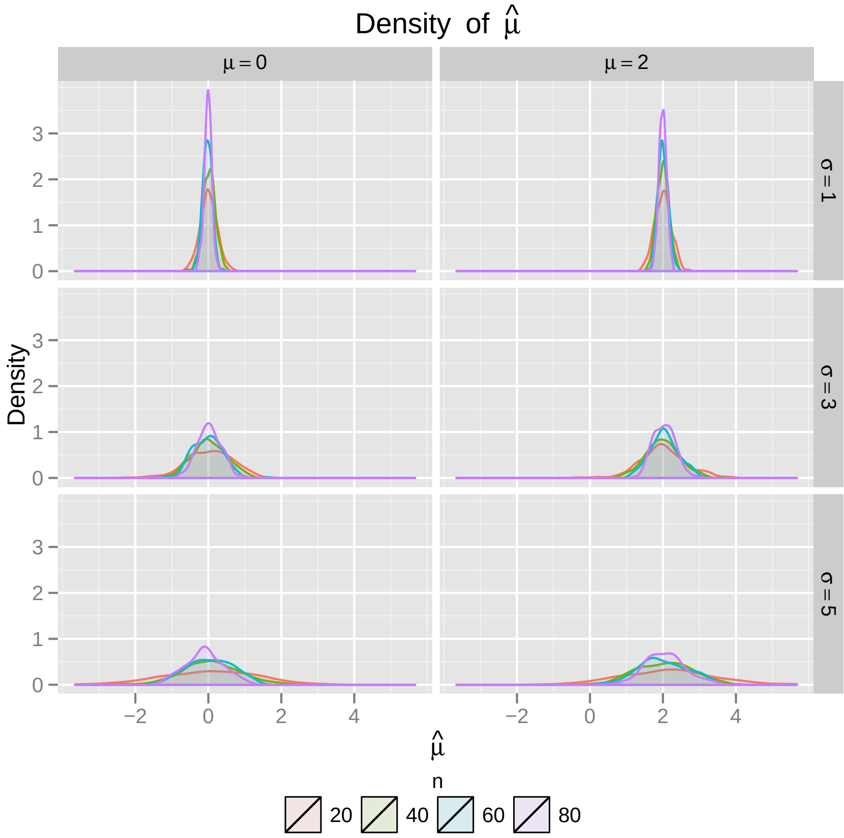
```
             estimator  n sigma mu    Mean     TV    Bias     SD   rmse JB_test
1              hat(mu) 20     1  0  0.0106 0.0000  0.0106 0.2298 0.2301  0.9198
2              hat(mu) 20     1  2  1.9993 2.0000 -0.0007 0.2255 0.2255  0.8646
3              hat(mu) 20     3  0  0.0330 0.0000  0.0330 0.6237 0.6246  0.0577
4              hat(mu) 20     3  2  1.9998 2.0000 -0.0002 0.6335 0.6335  0.0347
5              hat(mu) 20     5  0  0.0380 0.0000  0.0380 1.2679 1.2685  0.5275
6              hat(mu) 20     5  2  2.1287 2.0000  0.1287 1.2446 1.2512  0.8584
7              hat(mu) 40     1  0  0.0230 0.0000  0.0230 0.1723 0.1738  0.7900
8              hat(mu) 40     1  2  1.9974 2.0000 -0.0026 0.1622 0.1622  0.7210
9              hat(mu) 40     3  0  0.0310 0.0000  0.0310 0.4663 0.4673  0.7038
10             hat(mu) 40     3  2  1.9993 2.0000 -0.0007 0.4765 0.4765  0.4037
11             hat(mu) 40     5  0  0.0726 0.0000  0.0726 0.7845 0.7878  0.6496
12             hat(mu) 40     5  2  2.0298 2.0000  0.0298 0.7496 0.7502  0.3630
13             hat(mu) 60     1  0 -0.0261 0.0000 -0.0261 0.1281 0.1308  0.9117
14             hat(mu) 60     1  2  2.0066 2.0000  0.0066 0.1366 0.1368  0.4754
15             hat(mu) 60     3  0 -0.0189 0.0000 -0.0189 0.4090 0.4094  0.1381
16             hat(mu) 60     3  2  2.0366 2.0000  0.0366 0.3906 0.3923  0.5705
17             hat(mu) 60     5  0  0.0461 0.0000  0.0461 0.6229 0.6246  0.1668
18             hat(mu) 60     5  2  2.0131 2.0000  0.0131 0.6784 0.6785  0.5914
19             hat(mu) 80     1  0 -0.0083 0.0000 -0.0083 0.1023 0.1026  0.4071
20             hat(mu) 80     1  2  1.9945 2.0000 -0.0055 0.1058 0.1059  0.8558
21             hat(mu) 80     3  0  0.0214 0.0000  0.0214 0.3469 0.3476  0.0031
22             hat(mu) 80     3  2  2.0323 2.0000  0.0323 0.3059 0.3076  0.4124
23             hat(mu) 80     5  0 -0.1038 0.0000 -0.1038 0.5176 0.5279  0.5476
24             hat(mu) 80     5  2  2.0546 2.0000  0.0546 0.5564 0.5591  0.6043
25 hat(sigma[hat(mu)]) 20     1  0  0.2271 0.2294 -0.0023 0.0345 0.0346  0.4108
26 hat(sigma[hat(mu)]) 20     1  2  0.2263 0.2294 -0.0031 0.0358 0.0360  0.8843
27 hat(sigma[hat(mu)]) 20     3  0  0.6906 0.6882  0.0024 0.1196 0.1197  0.3257
28 hat(sigma[hat(mu)]) 20     3  2  0.6754 0.6882 -0.0129 0.1131 0.1138  0.0177
29 hat(sigma[hat(mu)]) 20     5  0  1.1476 1.1471  0.0005 0.1774 0.1774  0.9816
30 hat(sigma[hat(mu)]) 20     5  2  1.1520 1.1471  0.0049 0.1894 0.1894  0.0000
31 hat(sigma[hat(mu)]) 40     1  0  0.1581 0.1601 -0.0021 0.0194 0.0195  0.4465
```

```
32 hat(sigma[hat(mu)]) 40      1  2  0.1580 0.1601 -0.0022 0.0173 0.0175  0.3280
33 hat(sigma[hat(mu)]) 40      3  0  0.4733 0.4804 -0.0071 0.0516 0.0521  0.5464
34 hat(sigma[hat(mu)]) 40      3  2  0.4780 0.4804 -0.0023 0.0586 0.0586  0.3437
35 hat(sigma[hat(mu)]) 40      5  0  0.7970 0.8006 -0.0036 0.0990 0.0990  0.9872
36 hat(sigma[hat(mu)]) 40      5  2  0.8085 0.8006  0.0078 0.0826 0.0829  0.3247
37 hat(sigma[hat(mu)]) 60      1  0  0.1298 0.1302 -0.0004 0.0117 0.0117  0.2192
38 hat(sigma[hat(mu)]) 60      1  2  0.1292 0.1302 -0.0010 0.0123 0.0124  0.8760
39 hat(sigma[hat(mu)]) 60      3  0  0.3917 0.3906  0.0011 0.0359 0.0359  0.8815
40 hat(sigma[hat(mu)]) 60      3  2  0.3863 0.3906 -0.0042 0.0379 0.0382  0.2737
41 hat(sigma[hat(mu)]) 60      5  0  0.6492 0.6509 -0.0018 0.0618 0.0619  0.1629
42 hat(sigma[hat(mu)]) 60      5  2  0.6515 0.6509  0.0006 0.0568 0.0568  0.7048
43 hat(sigma[hat(mu)]) 80      1  0  0.1129 0.1125  0.0004 0.0089 0.0089  0.2456
44 hat(sigma[hat(mu)]) 80      1  2  0.1131 0.1125  0.0006 0.0091 0.0091  0.3944
45 hat(sigma[hat(mu)]) 80      3  0  0.3384 0.3375  0.0009 0.0267 0.0267  0.1811
46 hat(sigma[hat(mu)]) 80      3  2  0.3347 0.3375 -0.0028 0.0285 0.0287  0.9095
47 hat(sigma[hat(mu)]) 80      5  0  0.5622 0.5625 -0.0003 0.0472 0.0472  0.1978
48 hat(sigma[hat(mu)]) 80      5  2  0.5591 0.5625 -0.0034 0.0434 0.0435  0.1654
```

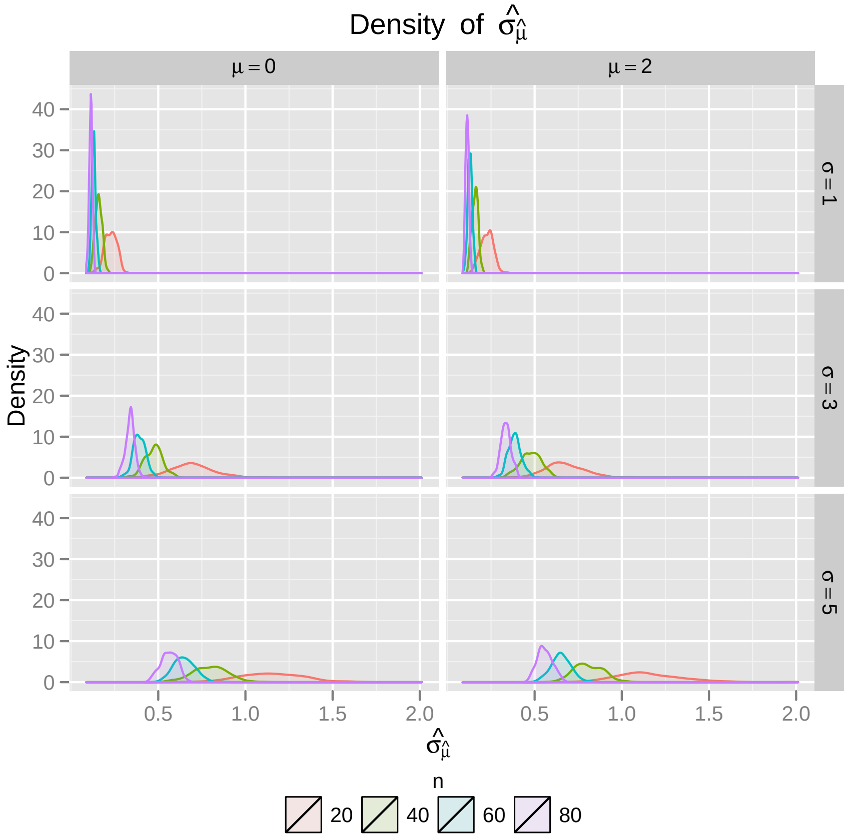
```
> plot(ezsim_basic)
> plot(ezsim_basic, "density")
```


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


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



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



2 Pre-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. Estimators : It computes the estimates from the data generated by dgp.

Also there are four optional components, they are:

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. run : If it is true, then the simulation will be ran right after the ezsim

4. object is created. Otherwise, you can run it manually by `run(ezsim_basic)`. Default is `TRUE`.

5. core : The number of CPU core to be used in the simulation. It is implemented by `foreach` and `doSNOW`. Default is 1.

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, `scalars`(the first argument) and `others`. `scalars` are all scalar parameters in the parameters set. Any vectors or matrix are regarded as a sequence of the same parameter. If some of your parameters are vector, matrix or other data types, you need to define it as `others`. `others` can take any data type but are fixed in the simulation.
In our example, all parameters are scalars. We can create a parameterDef object by:

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

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

$mu
[1] 0 2

$sigma
[1] 1 3 5


Others 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
```

`setScalars` and `setOthers` 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(scalars = list(mu1 = 5, mu2 = 3, n = c(10,
+     20)), others = list(Sigma = matrix(c(1, 0.4, 0.4, 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)
+ }
```

```
> test(par_def, dgp, index = 1)
```

```
 [1]  0.3460318 -0.7644364 -1.0389696 -1.2526108 -0.7066684  2.0273549
 [7] -1.8098753  1.4685719  0.5087217  1.2583626 -0.5306350  0.6596260
[13]  1.7332196 -0.2290221 -0.7474162 -0.6828299 -0.1314443  0.7391476
[19]  0.8119996 -1.2614423
```

```
> test(par_def, dgp, index = 2)
```

```
 [1] -0.00111411 -1.02256125  0.52392272 -0.81678037  0.39237660  1.41866312
 [7] -1.49545384  1.51450064 -0.25748931  1.38715466  1.57991660  0.01591299
[13]  0.56026950 -0.59601117  1.02834695 -1.73852254  0.28653992  0.28470918
[19]  0.93371063  1.34756338 -0.11160695  0.12269541 -0.07385384  1.27671521
[25]  0.94298343  1.07400810 -2.15790704  0.51536450  1.09691406 -1.61404404
[31]  1.05863627 -1.72136710  0.09288358  1.63228177  1.23516780 -0.40861012
[37] -0.47453615 -0.06616759  0.31985732 -0.53873052
```

```
> 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))
+ }
> test(par_def2, dgp_2)
```

```
            x1       x2
 [1,] 4.927191 4.217992
 [2,] 4.693803 3.086956
 [3,] 5.577923 3.374341
 [4,] 4.800721 3.269211
 [5,] 6.476667 2.371373
 [6,] 6.017221 3.518932
 [7,] 3.673366 1.198036
 [8,] 5.673229 3.094817
 [9,] 3.900227 3.755388
[10,] 4.573381 3.176679
```

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 `test` 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(test(par_def, dgp, index = 1))
```

```
  mean_hat sd_mean_hat
 0.6192730   0.2931263
```

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))
+ }
> test(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)])")
```

3 Post-simulation

3.1 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.

3.1.1 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 JB_test
1   hat(mu) 20     1  0 0.0106  0  0.0106 0.2298 0.2301  0.9198
2   hat(mu) 20     1  2 1.9993  2 -0.0007 0.2255 0.2255  0.8646
3   hat(mu) 20     3  0 0.0330  0  0.0330 0.6237 0.6246  0.0577
4   hat(mu) 20     3  2 1.9998  2 -0.0002 0.6335 0.6335  0.0347
5   hat(mu) 40     1  0 0.0230  0  0.0230 0.1723 0.1738  0.7900
6   hat(mu) 40     1  2 1.9974  2 -0.0026 0.1622 0.1622  0.7210
7   hat(mu) 40     3  0 0.0310  0  0.0310 0.4663 0.4673  0.7038
8   hat(mu) 40     3  2 1.9993  2 -0.0007 0.4765 0.4765  0.4037
```

3.1.2 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 0.0106  0  0.0106            Inf 0.2298 0.2301 -0.6071
2   hat(mu) 20     1  2 1.9993  2 -0.0007        -0.0003 0.2255 0.2255  1.4436
3   hat(mu) 20     3  0 0.0330  0  0.0330            Inf 0.6237 0.6246 -2.2180
4   hat(mu) 20     3  2 1.9998  2 -0.0002        -0.0001 0.6335 0.6335 -0.3102
5   hat(mu) 40     1  0 0.0230  0  0.0230            Inf 0.1723 0.1738 -0.5584
6   hat(mu) 40     1  2 1.9974  2 -0.0026        -0.0013 0.1622 0.1622  1.5727
7   hat(mu) 40     3  0 0.0310  0  0.0310            Inf 0.4663 0.4673 -1.1915
8   hat(mu) 40     3  2 1.9993  2 -0.0007        -0.0003 0.4765 0.4765  0.8760
      Q25 Median    Q75    Max JB_test
```

```
1 -0.1433 0.0061 0.1577 0.6897  0.9198
2  1.8410 2.0050 2.1390 2.6984  0.8646
3 -0.4009 0.0452 0.4926 1.4634  0.0577
4  1.6404 1.9933 2.3681 3.8057  0.0347
5 -0.0945 0.0260 0.1315 0.4844  0.7900
6  1.8796 2.0029 2.1037 2.3917  0.7210
7 -0.2566 0.0009 0.3834 1.0993  0.7038
8  1.6945 1.9770 2.2776 3.6411  0.4037
```

3.1.3 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 -0.1433 0.0061 0.1577 -0.4612 0.4640
2   hat(mu) 20     1  2  1.8410 2.0050 2.1390  1.5582 2.4094
3   hat(mu) 20     3  0 -0.4009 0.0452 0.4926 -1.3888 1.1485
4   hat(mu) 20     3  2  1.6404 1.9933 2.3681  0.7800 3.3277
5   hat(mu) 40     1  0 -0.0945 0.0260 0.1315 -0.3149 0.3572
6   hat(mu) 40     1  2  1.8796 2.0029 2.1037  1.6942 2.3215
7   hat(mu) 40     3  0 -0.2566 0.0009 0.3834 -0.8174 0.9061
8   hat(mu) 40     3  2  1.6945 1.9770 2.2776  1.0673 2.9870
```

3.2 Plotting the simulation

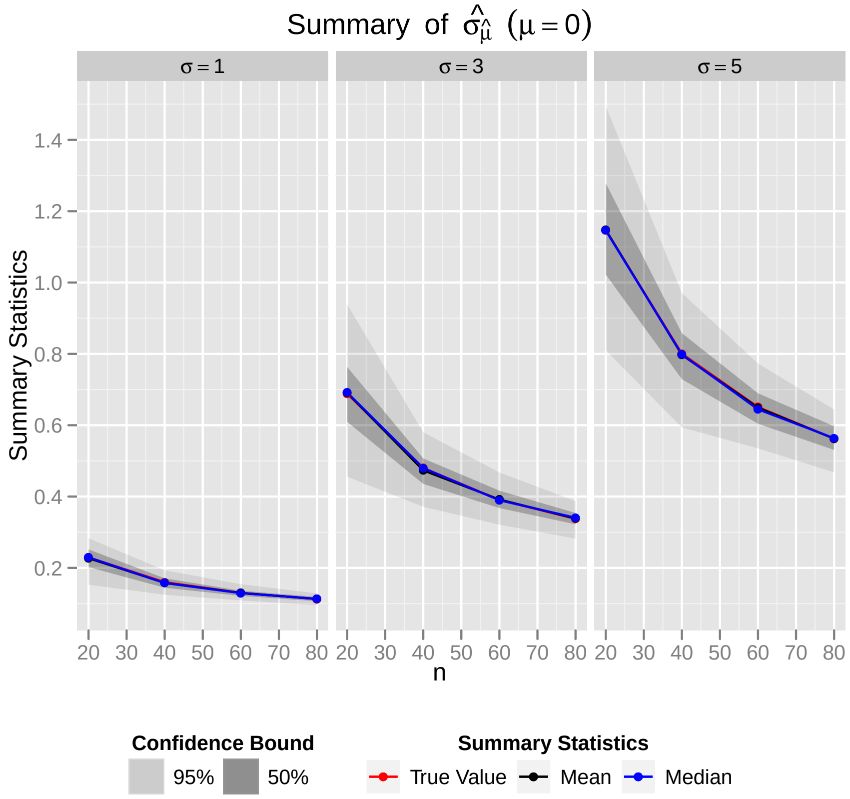
3.2.1 Plotting an ezsim object

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 with the most number of value. Rest of them will be facets of the plot. Each estimator will occupy one plot. See section 1 for examples.

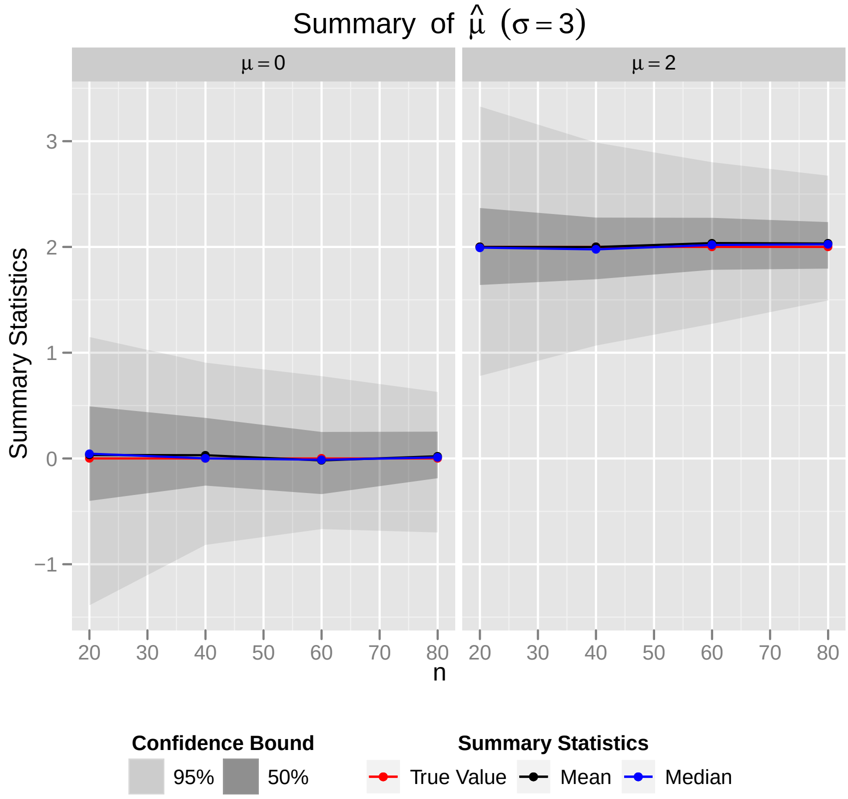
3.2.2 Subset of the Plot

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))
```



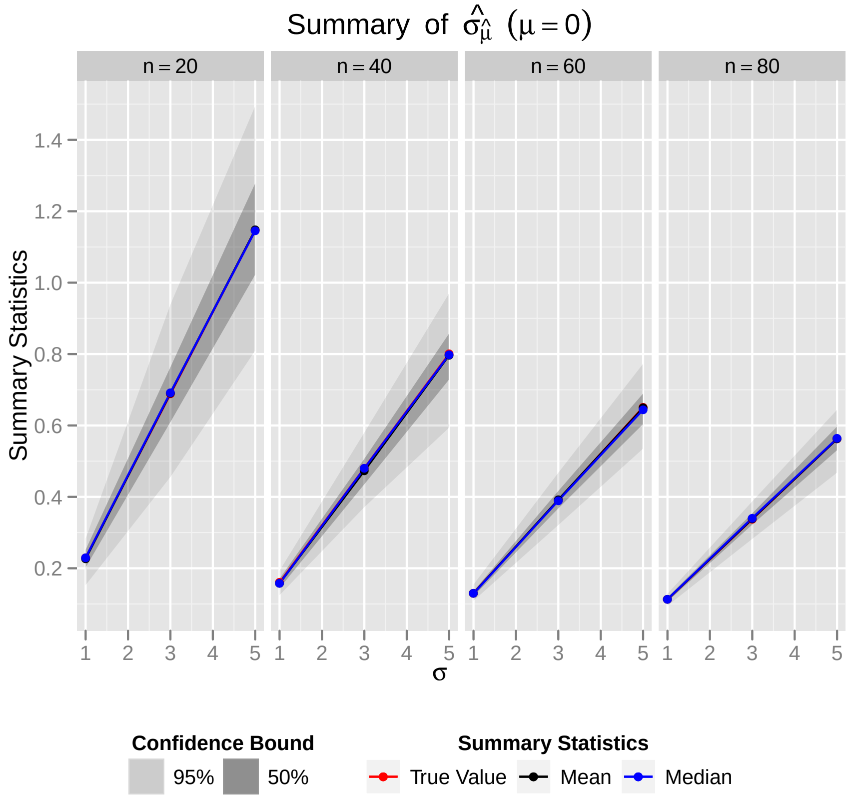

```
> plot(ezsim_basic, subset = list(estimator = "mean_hat", sigma = 3))
```



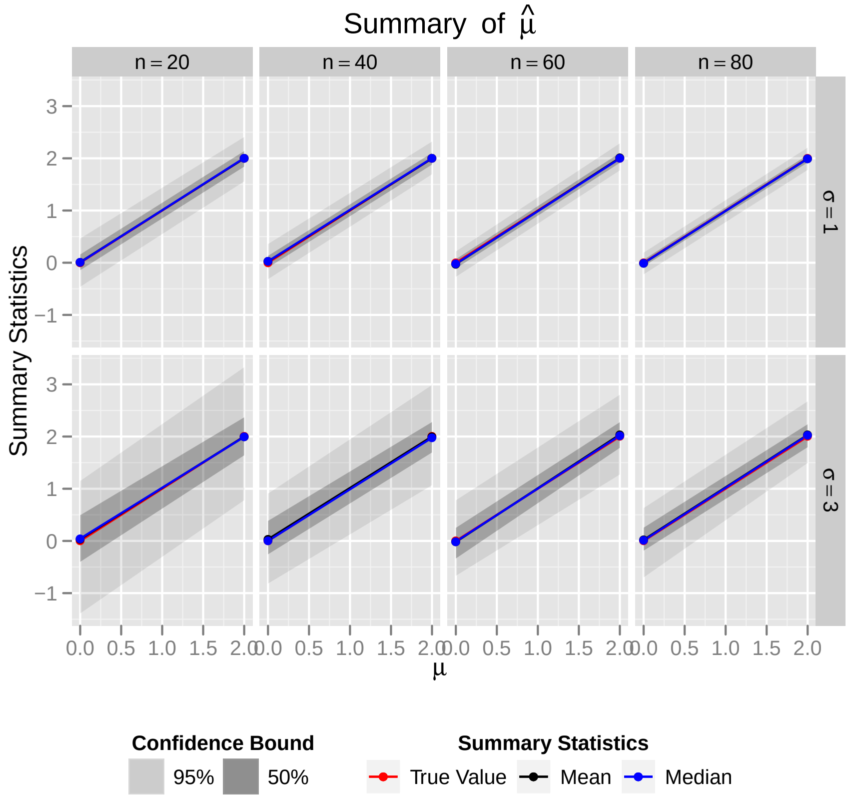
3.2.3 Parameters Priority of the Plot

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"))
```



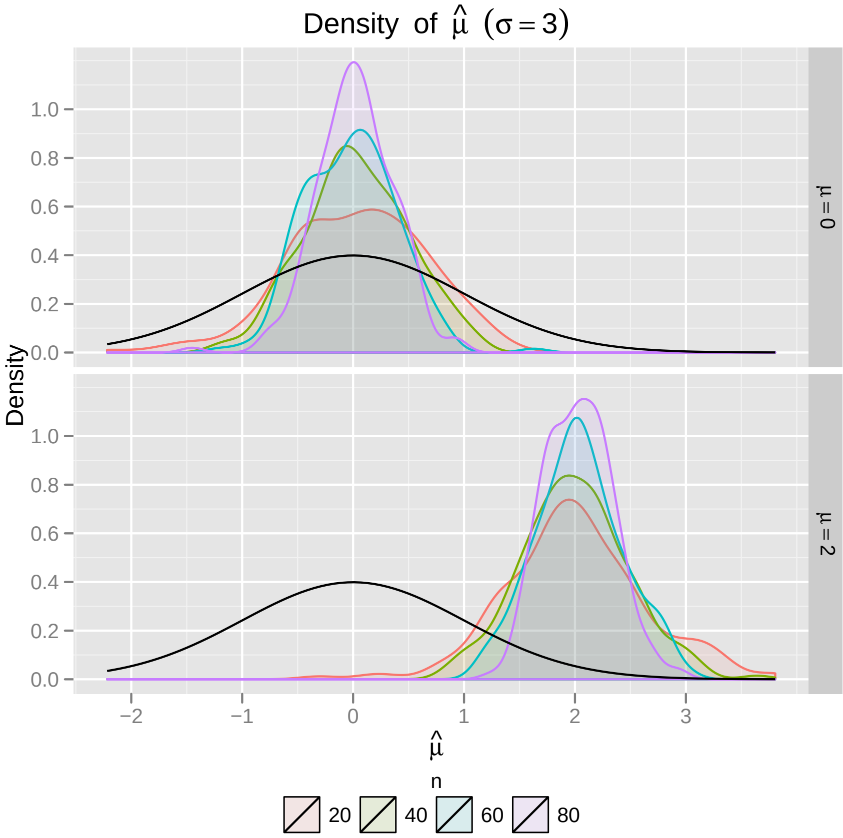

```
> plot(ezsim_basic, subset = list(estimator = "mean_hat", sigma = c(1,
+     3)), parameters_priority = "mu")
```



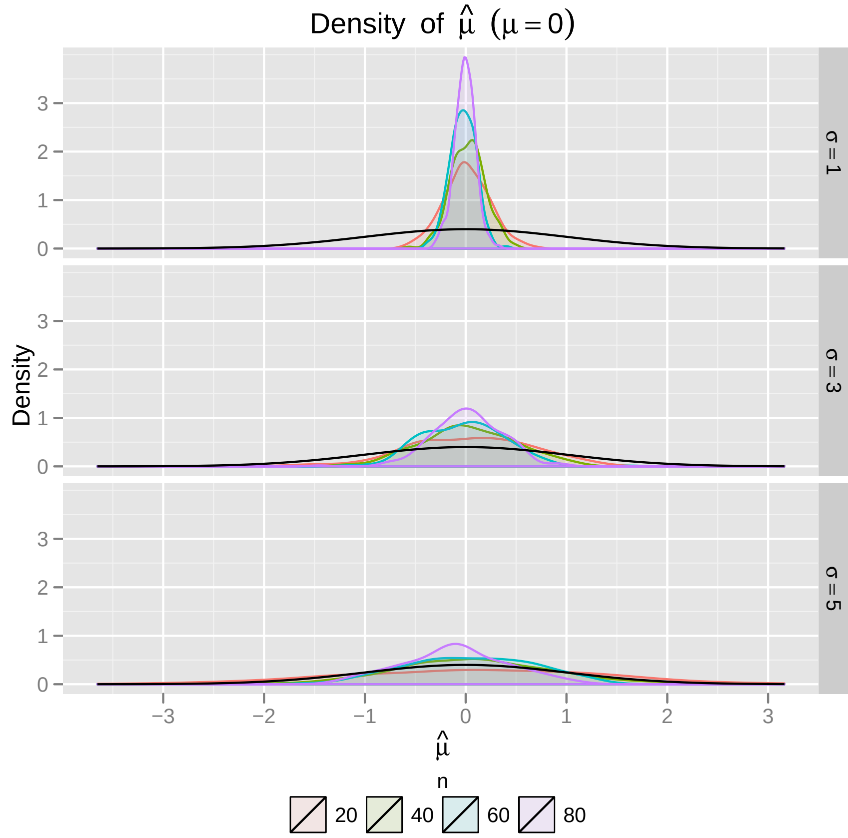
3.2.4 Density Plot

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)
```

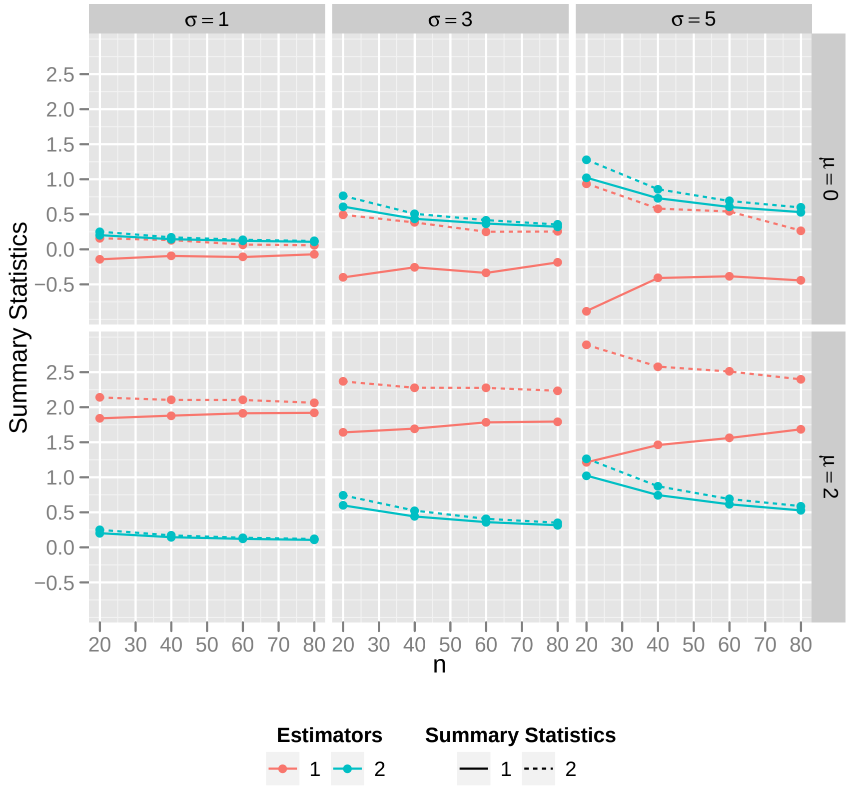



```
> plot(ezsim_basic, "density", subset = list(estimator = "mean_hat",
+     mu = 0), parameters_priority = "n", benchmark = dnorm)
```

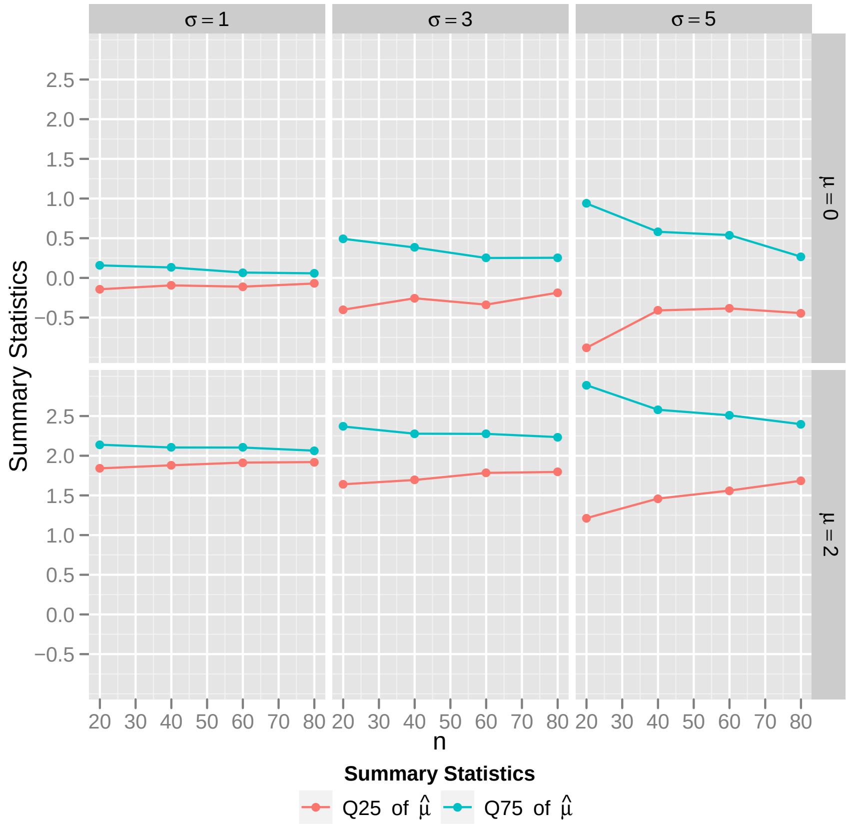


3.2.5 Plot the summary ezsim

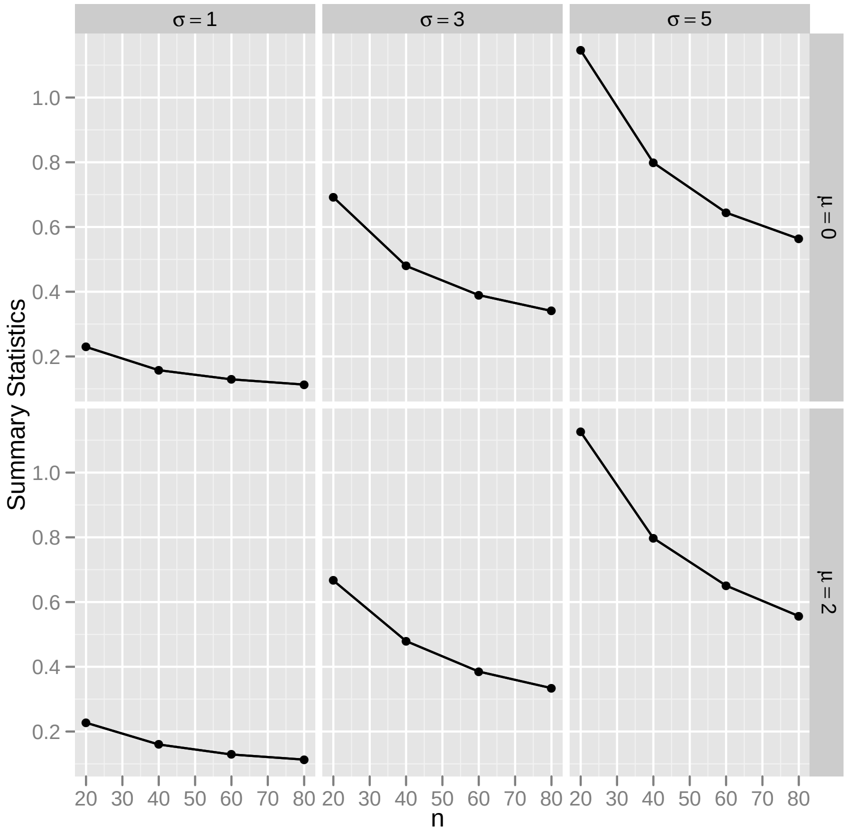
```
> plot(summary(ezsim_basic, c("q25", "q75")))
```




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




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



3.2.6 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,
+     core          = 1,
+     display_name  = c(b="beta",es="sigma[e]^2",xs="sigma[x]^2"),
+     parameter_def = createParDef(scalars=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)
```