EVPPI_Bristol

Wei Fang

12/03/2017

This is an R Markdown document which contains all the codes and results of all the EVPI and EVPPI for the EVPPI project in University of Bristol. Due to some update issues in the MLMC package, we currently use the functions in the package but with some modifications on codes.

The following four package are needed for MLMC functions:

```
require(parallel)
require(ggplot2)
require(grid)
require(Rcpp)
```

The following R code are for MLMC test can calculation. The first three come from the MLMC pacakge in R (see: https://cran.r-project.org/web/packages/mlmc/index.html) with some new updates by Wei.

```
source("mlmc.R") #MLMC code to calculate the value to some accuracy
source("mlmc.test.R") #test function of the convergence rates and output the MLMC results
source("plot.mlmc.test.R") #function to plot the result
source("multiplot.R") #this is the function from ggplot
```

The following five R code are developed by Wei to calculate different EVPI and EVPPI with N out samples and M^l inner samples. These can be called by **mlmc.R** to do MLMC calculations.

```
source("EVPI_l.R")
source("EVPPI_P_l.R")
source("EVPPI_cq_l.R")
source("EVPPI_lor1_l.R")
source("EVPPI_lor2_l.R")
```

EVPI

Suppose that the perfect information is available. That means by density ρ_Z we can generate samples of Z and for each sample $Z^{(n)}$ we can find the optimal decision d for that specific decision. Therefore the expected value of perfect information (EVPI) is

$$\mathrm{EVPI} = \mathbb{E}_Z\left[\max_{d\in D} f_d(Z)\right] - \max_{d\in D} \mathbb{E}_Z\left[f_d(Z)\right].$$

and the Monte Carlo estimator on each level l with M^l inner sample is

$$\overline{\mathrm{EVPI}}_l = \frac{1}{M^l}\sum_{m=1}^{M^l} \max_{d\in D} f_d(Z^{(m)}) - \max_{d\in D} \frac{1}{M^l}\sum_{m=1}^{M^l} f_d(Z^{(m)})$$

and MLMC estimator on each level l is

$$\overline{\mathrm{EVPI}}_l - \frac{1}{2}\left(\overline{\mathrm{EVPI}}^1_{l-1} + \overline{\mathrm{EVPI}}^2_{l-1}\right)$$

where the $\overline{\mathrm{EVPI}}^1_{l-1}$ uses the first M^{l-1} samples and $\overline{\mathrm{EVPI}}^2_{l-1}$ uses the second M^{l-1} samples.

Here is the code for **EVPI_l.R** which requre `Matrix`, `MASS` and `boot` packages. This function calculate N samples of the MLMC estimator on level l shown above.

```
EVPI_l <- function(l, N) {
    require(Matrix)
    require(MASS) # mvrnorm
    require(boot) # logit, inv.logit

    # define the cost and lambda
    lamda <- 20000
    C_t1 <- 300
    C_t2 <- 30

    # calculate the number of inner samples
    M <- 2^(l+1)

    # define the mean and covariance matrix
    mu_rec <- c(0.99,1.33)
    sigma_rec <- matrix(c(0.22,0.15,0.15,0.20),2,2,byrow = TRUE)
    mu_rel <- c(-1.48,-0.4)
    sigma_rel <- matrix(c(0.14,0.05,0.05,0.11),2,2,byrow = TRUE)

    # sum1 record the relavent statistics
    sum1 <- rep(0, 7)

    for(N1 in seq(1, N, by=10000)) {
      N2 <- min(10000, N-N1+1)

      # Sample all the random variables and tranform to M*N2 matrix
      P_nt_rec <- matrix(rbeta(M*N2,6,200),M,N2,byrow = TRUE)
      P_nt_rel <- matrix(rbeta(M*N2,2,100),M,N2,byrow = TRUE)

      lor_rec <- mvrnorm(M*N2, mu_rec, sigma_rec)
      lor_rel <- mvrnorm(M*N2, mu_rel, sigma_rel)

      lor_t1_rec <- matrix(lor_rec[,1],M,N2,byrow = TRUE)
      lor_t1_rel <- matrix(lor_rel[,1],M,N2,byrow = TRUE)
      P_t1_rec <- inv.logit(logit(P_nt_rec)+lor_t1_rec)
      P_t1_rel <- inv.logit(logit(P_nt_rel)+lor_t1_rel)

      lor_t2_rec <- matrix(lor_rec[,2],M,N2,byrow = TRUE)
      lor_t2_rel <- matrix(lor_rel[,2],M,N2,byrow = TRUE)
      P_t2_rec <- inv.logit(logit(P_nt_rec)+lor_t2_rec)
      P_t2_rel <- inv.logit(logit(P_nt_rel)+lor_t2_rel)

      C_rec = matrix(rnorm(M*N2,1000,50),M,N2,byrow = TRUE)
      C_rel = matrix(rnorm(M*N2,2000,100),M,N2,byrow = TRUE)
      C_no_rec = matrix(rnorm(M*N2,2500,125),M,N2,byrow = TRUE)

      Q_rec = matrix(rnorm(M*N2,26,2),M,N2,byrow = TRUE)
      Q_rel = matrix(rnorm(M*N2,23,3),M,N2,byrow = TRUE)
      Q_no_rec = matrix(rnorm(M*N2,20,4),M,N2,byrow = TRUE)

      # calculate the net benefits
```

```
    NB_nt = (lamda*(P_nt_rec*(1-P_nt_rel)*Q_rec + P_nt_rec*P_nt_rel*Q_rel
                   +(1-P_nt_rec)*Q_no_rec)
    - (P_nt_rec*(1-P_nt_rel)*C_rec + P_nt_rec*P_nt_rel*C_rel
       +(1-P_nt_rec)*C_no_rec ))
    NB_t1 = (lamda*(P_t1_rec*(1-P_t1_rel)*Q_rec + P_t1_rec*P_t1_rel*Q_rel
                   +(1-P_t1_rec)*Q_no_rec)
    - (C_t1 + P_t1_rec*(1-P_t1_rel)*C_rec + P_t1_rec*P_t1_rel*C_rel
       +(1-P_t1_rec)*C_no_rec ))
    NB_t2 = (lamda*(P_t2_rec*(1-P_t2_rel)*Q_rec + P_t2_rec*P_t2_rel*Q_rel
                   +(1-P_t2_rec)*Q_no_rec)
    - (C_t2 + P_t2_rec*(1-P_t2_rel)*C_rec + P_t2_rec*P_t2_rel*C_rel
       +(1-P_t2_rec)*C_no_rec ))

    # calculate the fine level and coarse level estimators
    Pb = colMeans(pmax(pmax(NB_nt,NB_t1),NB_t2))
    Pf = Pb-pmax(pmax(colMeans(NB_nt), colMeans(NB_t1)), colMeans(NB_t2))
    if(M==2)
    {
      Pc = Pb-0.5*(pmax(pmax(NB_nt[1,], NB_t1[1,]),NB_t2[1,])
                   +pmax(pmax(NB_nt[2,], NB_t1[2,]), NB_t2[2,]))

    } else {
    Pc = Pb-0.5*(pmax(pmax(colMeans(NB_nt[1:(M/2),]),
                 colMeans(NB_t1[1:(M/2),])),colMeans(NB_t2[1:(M/2),]))
                 +pmax(pmax(colMeans(NB_nt[((M/2)+1):M,]),
          colMeans(NB_t1[((M/2)+1):M,])), colMeans(NB_t2[((M/2)+1):M,])))
    }

    # update the statistics
    sum1[1] = sum1[1] + sum(Pf-Pc);
    sum1[2] = sum1[2] + sum((Pf-Pc)^2);
    sum1[3] = sum1[3] + sum((Pf-Pc)^3);
    sum1[4] = sum1[4] + sum((Pf-Pc)^4);
    sum1[5] = sum1[5] + sum(Pf);
    sum1[6] = sum1[6] + sum(Pf^2);
    sum1[7] = sum1[7] + M*N2;
  }
  return(sum1)
}
```

Note that N1 and N2 in this function are the trick of splitting the outer samples into a suitable size N2=10000 to achieve the best computational performance. If we only call the level function (e.g. EVPI_l(l, N)), then the number of outer samples is N. Both N1 and N2 are introduced to get the best efficiency of the matrix computation. That means, instead of computing the N out samples one by one, we compute N2 samples each time by using matrix computation. N2=10000 is the optimal size for matrix computation due to the tradeoff between the memory and speed. Then we need to do this calculation several times and N1 denote 1+the number of samples you have calculated. Therefore, if we want to compute EVPI(l,33000), we will do the for loop four times (N1=1, N2=10000), (N1=10001, N2=10000), (N1=20001, N2=10000) and (N1=30001, N2 = 3000). In the last iteration, N2=3000 because we only need to calculate that number of the samples, which is ensured by the first line in the for loop. We also do some experiment on it by only changing the value of N2 range from 10 to 10^6 to do the mlmc.test function with the accuracy 1 and recording the running time. Here is the result:

N2	Time (s)
1e+01	142.812
1e+02	33.334
1e+03	23.683
1e+04	23.560
1e+05	24.198
1e+06	30.394

From these results, we can see that N2=10^4 is the best.

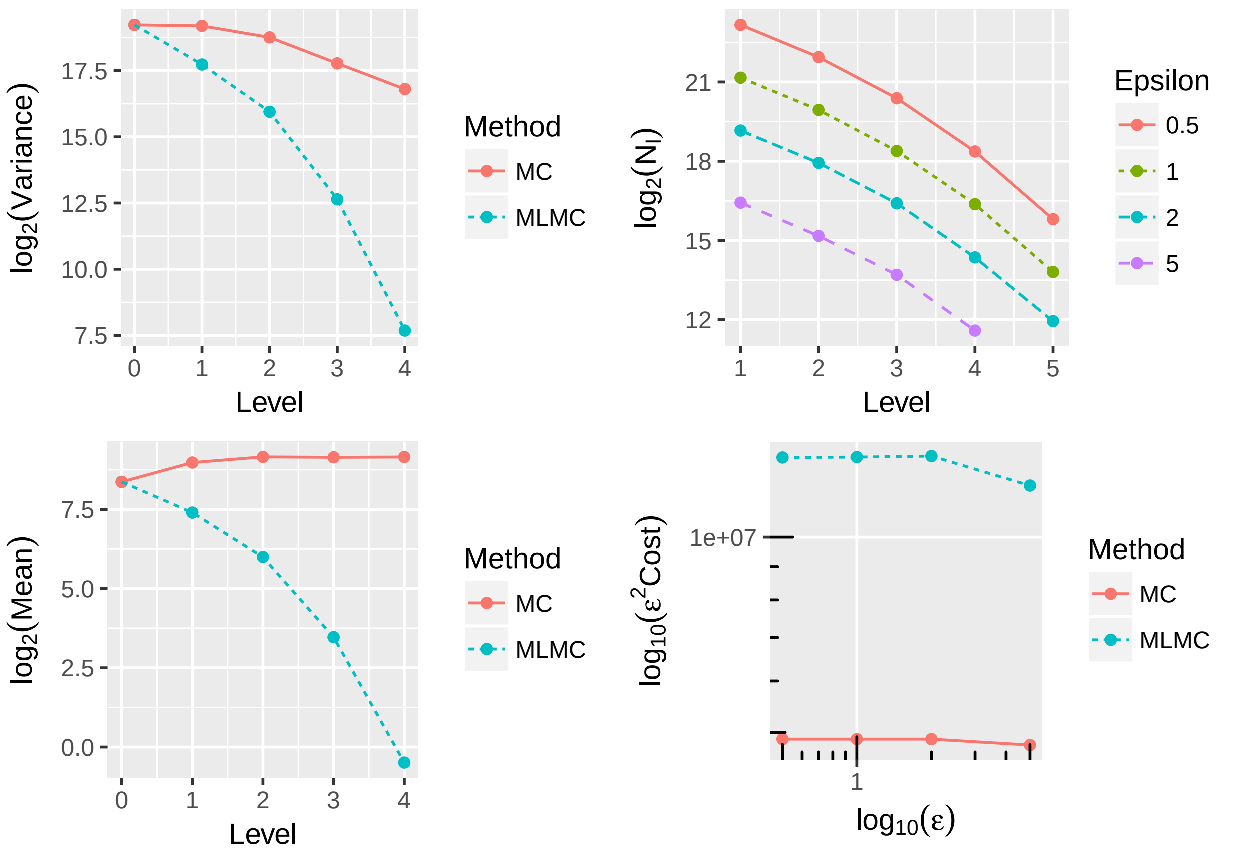
The following **mlmc.test** function test the EVPI value with M^l inner samples for each level. **N** is the number of the out samples for convergence test. **L** is the max level for convergence test. **N_0** is the initial number of out samples for MLMC caculation which should be larger than the kurtosis to ensure the correct estimation of the variance. **eps.v** is the accuracy you want to achieve. You can test different accuracy using vector. **Lmin** and **Lmax** is the min and max number of levels used in MLMC. **parallel** is for parallel computing which is a integer.

```r
set.seed(666) # Set random seed to ensure the same output
# without this MLMC will give different output each time
tst <- mlmc.test(EVPI_l, M=2, N=10000,
                 L=4, N0=10000,
                 eps.v=c(0.5,1,2,5),
                 Lmin=2, Lmax=10, parallel = 32)
```

```
##
## **********************************************************
## *** Convergence tests, kurtosis, telescoping sum check ***
## **********************************************************
##
##  l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)    kurtosis     check
## --------------------------------------------------------------------------
##  0   3.3043e+02  3.3043e+02  6.1407e+05  6.1407e+05  0.0000e+00  0.0000e+00
##  1   1.6881e+02  5.0399e+02  2.1757e+05  5.9802e+05  3.2059e+01  7.8256e-02
##  2   6.3791e+01  5.7022e+02  6.3150e+04  4.4255e+05  6.8721e+01  4.8033e-02
##  3   1.1051e+01  5.6533e+02  6.3752e+03  2.2405e+05  1.3617e+02  4.3616e-01
##  4   7.1243e-01  5.6903e+02  2.0539e+02  1.1440e+05  7.4995e+02  1.2070e-01
##
##  WARNING: kurtosis on finest level = 749.945201
##  indicates MLMC correction dominated by a few rare paths;
##  for (information on the connection to variance of sample variances,
##  see http://mathworld.wolfram.com/SampleVarianceDistribution.html
##
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 3.242238  (exponent for (MLMC weak convergence)
##  beta  in 4.132133  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## *****************************
## *** MLMC complexity tests ***
## *****************************
```

```
##
##    eps       value    mlmc_cost    std_cost  savings      N_l
## ------------------------------------------------------------
## 0.5000  5.7501e+02  5.307e+07  1.952e+07     0.37   9384097    4022085    1369461     338880      57225
## 1.0000  5.7468e+02  1.328e+07  4.881e+06     0.37   2347260    1007127     342918      84928      14342
## 2.0000  5.7461e+02  3.333e+06  1.220e+06     0.37    586361     250699      86906      21015       3943
## 5.0000  5.6856e+02  4.803e+05  1.912e+05     0.40     88409      36904      13320       3082
```

```r
EVPI = tst$P[1]
plot(tst,which=c("var", "mean", "Nl", "cost"),cols=2) # plot
```



The estimation of EVPI is 575.0147717 with root Mean Square Error 0.5.

EVPPI

Assume that the unknown parameters can be decomposed into two random variables as $Z = (X, Y)$ with $\Omega = \Omega_X \times \Omega_Y$ and only information of X is available. That means we can generate samples of X first and for each sample $X^{(n)}$, we can calculate the maximum of the conditional expectation of Y based on $X^{(n)}$. Therefore the expected value of partial perfect information (the value of X) is

$$\text{EVPPI} = \mathbb{E}_X\left[\max_{d\in D}\mathbb{E}_{Y|X}\left[f_d(X,Y)\right]\right] - \max_{d\in D}\mathbb{E}_Z\left[f_d(Z)\right]$$

So the conditional distribution of Y based on X is important and available. In MLMC, instead of directly estimating the EVPPI, we estimate the EVPI-EVPPI:

$$\text{DIFF} = \mathbb{E}_Z\left[\max_{d\in D} f_d(Z)\right] - \mathbb{E}_X\left[\max_{d\in D}\mathbb{E}_{Y|X}\left[f_d(X,Y)\right]\right]$$

and the Monte Carlo estimator on each level l with M^l inner sample based on $X^{(n)}$ is

$$\overline{\mathrm{DIFF}}_l = \frac{1}{M^l}\sum_{m=1}^{M^l} \max_{d\in D} f_d(X^{(n)}, Y^{(n,m)}) - \max_{d\in D} \frac{1}{M^l}\sum_{m=1}^{M^l} f_d(X^{(n)}, Y^{(n,m)})$$

and MLMC estimator on each level l is

$$\overline{\mathrm{DIFF}}_l - \frac{1}{2}\left(\overline{\mathrm{DIFF}}^1_{l-1} + \overline{\mathrm{DIFF}}^2_{l-1}\right)$$

where the $\overline{\mathrm{DIFF}}^1_{l-1}$ uses the first M^{l-1} samples and $\overline{\mathrm{DIFF}}^2_{l-1}$ uses the second M^{l-1} samples.

EVPPI for P

Here is the code for **EVPPI_P_l.R** which requre `Matrix`, `MASS` and `boot` packages:

```
EVPPI_P_l <- function(l, N) {
  require(Matrix)
  require(MASS) # mvrnorm
  require(boot) # logit

  lamda <- 20000
  C_t1 <- 300
  C_t2 <- 30
  M <- 2^(l+1)

  mu_rec <- c(0.99,1.33)
  sigma_rec <- matrix(c(0.22,0.15,0.15,0.20),2,2,byrow = TRUE)
  mu_rel <- c(-1.48,-0.4)
  sigma_rel <- matrix(c(0.14,0.05,0.05,0.11),2,2,byrow = TRUE)

  sum1 <- rep(0, 7)

  for(N1 in seq(1, N, by=10000)) {
    N2 <- min(10000, N-N1+1)

    P_nt_rec <- matrix(rep(rbeta(N2,6,200),M),M,N2,byrow = TRUE)
    P_nt_rel <- matrix(rep(rbeta(N2,2,100),M),M,N2,byrow = TRUE)

    lor_rec <- mvrnorm(N2, mu_rec, sigma_rec)
    lor_rel <- mvrnorm(N2, mu_rel, sigma_rel)

    # Generate N2 samples of P and repeat it to M*N2 matrix
    lor_t1_rec <- matrix(rep(lor_rec[,1],M),M,N2,byrow = TRUE)
    lor_t1_rel <- matrix(rep(lor_rel[,1],M),M,N2,byrow = TRUE)
    P_t1_rec <- inv.logit(logit(P_nt_rec)+lor_t1_rec)
    P_t1_rel <- inv.logit(logit(P_nt_rel)+lor_t1_rel)

    lor_t2_rec <- matrix(rep(lor_rec[,2],M),M,N2,byrow = TRUE)
    lor_t2_rel <- matrix(rep(lor_rel[,2],M),M,N2,byrow = TRUE)
    P_t2_rec <- inv.logit(logit(P_nt_rec)+lor_t2_rec)
    P_t2_rel <- inv.logit(logit(P_nt_rel)+lor_t2_rel)

    C_rec = matrix(rnorm(M*N2,1000,50),M,N2,byrow = TRUE)
```

```
    C_rel = matrix(rnorm(M*N2,2000,100),M,N2,byrow = TRUE)
    C_no_rec = matrix(rnorm(M*N2,2500,125),M,N2,byrow = TRUE)

    Q_rec = matrix(rnorm(M*N2,26,2),M,N2,byrow = TRUE)
    Q_rel = matrix(rnorm(M*N2,23,3),M,N2,byrow = TRUE)
    Q_no_rec = matrix(rnorm(M*N2,20,4),M,N2,byrow = TRUE)

    NB_nt = (lamda*(P_nt_rec*(1-P_nt_rel)*Q_rec + P_nt_rec*P_nt_rel*Q_rel
                   +(1-P_nt_rec)*Q_no_rec)
    - (P_nt_rec*(1-P_nt_rel)*C_rec + P_nt_rec*P_nt_rel*C_rel
       +(1-P_nt_rec)*C_no_rec ))
    NB_t1 = (lamda*(P_t1_rec*(1-P_t1_rel)*Q_rec + P_t1_rec*P_t1_rel*Q_rel
                   +(1-P_t1_rec)*Q_no_rec)
    - (C_t1 + P_t1_rec*(1-P_t1_rel)*C_rec + P_t1_rec*P_t1_rel*C_rel
       +(1-P_t1_rec)*C_no_rec ))
    NB_t2 = (lamda*(P_t2_rec*(1-P_t2_rel)*Q_rec + P_t2_rec*P_t2_rel*Q_rel
                   +(1-P_t2_rec)*Q_no_rec)
    - (C_t2 + P_t2_rec*(1-P_t2_rel)*C_rec + P_t2_rec*P_t2_rel*C_rel
       +(1-P_t2_rec)*C_no_rec ))

    Pb = colMeans(pmax(pmax(NB_nt,NB_t1),NB_t2))
    Pf = Pb-pmax(pmax(colMeans(NB_nt), colMeans(NB_t1)), colMeans(NB_t2))
    if(M==2)
    {
      Pc = Pb-0.5*(pmax(pmax(NB_nt[1,], NB_t1[1,]),NB_t2[1,])
                   +pmax(pmax(NB_nt[2,], NB_t1[2,]), NB_t2[2,]))

    } else {
      Pc = Pb-0.5*(pmax(pmax(colMeans(NB_nt[1:(M/2),]),
                  colMeans(NB_t1[1:(M/2),])),colMeans(NB_t2[1:(M/2),]))
                 +pmax(pmax(colMeans(NB_nt[((M/2)+1):M,]),
          colMeans(NB_t1[((M/2)+1):M,])), colMeans(NB_t2[((M/2)+1):M,])))
    }
    sum1[1] = sum1[1] + sum(Pf-Pc);
    sum1[2] = sum1[2] + sum((Pf-Pc)^2);
    sum1[3] = sum1[3] + sum((Pf-Pc)^3);
    sum1[4] = sum1[4] + sum((Pf-Pc)^4);
    sum1[5] = sum1[5] + sum(Pf);
    sum1[6] = sum1[6] + sum(Pf^2);
    sum1[7] = sum1[7] + M*N2;
  }
  return(sum1)
}
```

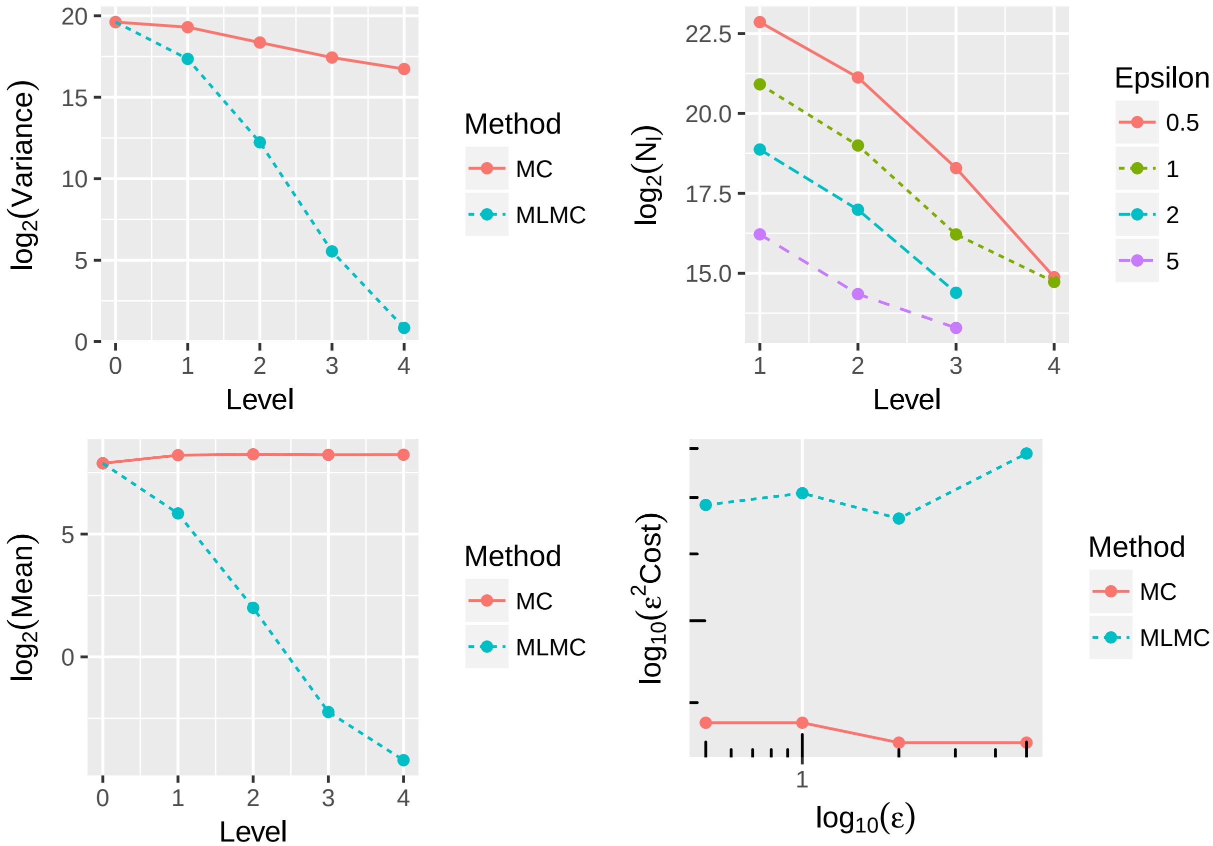
We can use the MLMC to calculate the value:

```
set.seed(666) # Set random seed to ensure the same output
# without this MLMC will give different output each time
tst <- mlmc.test(EVPPI_P_l, M=2, N=10000,
                 L=4, N0=10000,
                 eps.v=c(0.5,1,2,5),
                 Lmin=2, Lmax=10, parallel = 32)
```

```
##
```

```
## **********************************************************
## *** Convergence tests, kurtosis, telescoping sum check ***
## **********************************************************
##
##  l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)    kurtosis     check
## -------------------------------------------------------------------------
##  0   2.3555e+02  2.3555e+02  8.0231e+05  8.0231e+05  0.0000e+00  0.0000e+00
##  1   5.7241e+01  2.9567e+02  1.6738e+05  6.4520e+05  2.1013e+02  4.5583e-02
##  2   3.9898e+00  3.0335e+02  4.8219e+03  3.3620e+05  1.3548e+03  8.4638e-02
##  3   2.1168e-01  2.9881e+02  4.6532e+01  1.7748e+05  5.7816e+03  1.5706e-01
##  4   5.4363e-02  2.9978e+02  1.7921e+00  1.0892e+05  1.8839e+03  4.0428e-02
##
##  WARNING: kurtosis on finest level = 1883.894134
##  indicates MLMC correction dominated by a few rare paths;
##  for (information on the connection to variance of sample variances,
##  see http://mathworld.wolfram.com/SampleVarianceDistribution.html
##
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 3.098783  (exponent for (MLMC weak convergence)
##  beta  in 5.696884  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## ******************************
## *** MLMC complexity tests ***
## ******************************
##
##   eps       value   mlmc_cost   std_cost  savings     N_l
## ---------------------------------------------------------
## 0.5000  3.0096e+02  2.743e+07  1.514e+07     0.55   7614290   2289902    320345     30067
## 1.0000  3.0049e+02  7.081e+06  3.786e+06     0.53   1972960    523424     76009     27103
## 2.0000  3.0173e+02  1.652e+06  8.965e+05     0.54    480082    130047     21459
## 5.0000  3.0487e+02  3.157e+05  1.434e+05     0.45     76142     20845     10000
```

```r
plot(tst,which=c("var", "mean", "Nl", "cost"),cols=2) # plot
```



The estimation of EVPPI for P is 274.0576662 with root Mean Square Error 1.

EVPPI for cq

Here is the code for **EVPPI_cq_l.R** which requre `Matrix`, `MASS` and `boot` packages:

```
EVPPI_cq_l <- function(l, N) {
  require(Matrix)
  require(MASS) # mvrnorm
  require(boot) # logit

  lamda <- 20000
  C_t1 <- 300
  C_t2 <- 30
  M <- 2^(l+1)

  mu_rec <- c(0.99,1.33)
  sigma_rec <- matrix(c(0.22,0.15,0.15,0.20),2,2,byrow = TRUE)
  mu_rel <- c(-1.48,-0.4)
  sigma_rel <- matrix(c(0.14,0.05,0.05,0.11),2,2,byrow = TRUE)

  sum1 <- rep(0, 7)

  for(N1 in seq(1, N, by=10000)) {
    N2 <- min(10000, N-N1+1)
```

```
  P_nt_rec <- matrix(rbeta(M*N2,6,200),M,N2,byrow = TRUE)
  P_nt_rel <- matrix(rbeta(M*N2,2,100),M,N2,byrow = TRUE)

  lor_rec <- mvrnorm(M*N2, mu_rec, sigma_rec)
  lor_rel <- mvrnorm(M*N2, mu_rel, sigma_rel)

  lor_t1_rec <- matrix(lor_rec[,1],M,N2,byrow = TRUE)
  lor_t1_rel <- matrix(lor_rel[,1],M,N2,byrow = TRUE)
  P_t1_rec <- inv.logit(logit(P_nt_rec)+lor_t1_rec)
  P_t1_rel <- inv.logit(logit(P_nt_rel)+lor_t1_rel)

  lor_t2_rec <- matrix(lor_rec[,2],M,N2,byrow = TRUE)
  lor_t2_rel <- matrix(lor_rel[,2],M,N2,byrow = TRUE)
  P_t2_rec <- inv.logit(logit(P_nt_rec)+lor_t2_rec)
  P_t2_rel <- inv.logit(logit(P_nt_rel)+lor_t2_rel)

    # Generate N2 samples of C Q and repeat it to M*N2 matrix
  C_rec = matrix(rep(rnorm(N2,1000,50),M),M,N2,byrow = TRUE)
  C_rel = matrix(rep(rnorm(N2,2000,100),M),M,N2,byrow = TRUE)
  C_no_rec = matrix(rep(rnorm(N2,2500,125),M),M,N2,byrow = TRUE)

  Q_rec = matrix(rep(rnorm(N2,26,2),M),M,N2,byrow = TRUE)
  Q_rel = matrix(rep(rnorm(N2,23,3),M),M,N2,byrow = TRUE)
  Q_no_rec = matrix(rep(rnorm(N2,20,4),M),M,N2,byrow = TRUE)

  NB_nt = (lamda*(P_nt_rec*(1-P_nt_rel)*Q_rec + P_nt_rec*P_nt_rel*Q_rel
                 +(1-P_nt_rec)*Q_no_rec)
  - (P_nt_rec*(1-P_nt_rel)*C_rec + P_nt_rec*P_nt_rel*C_rel
     +(1-P_nt_rec)*C_no_rec ))
  NB_t1 = (lamda*(P_t1_rec*(1-P_t1_rel)*Q_rec + P_t1_rec*P_t1_rel*Q_rel
                 +(1-P_t1_rec)*Q_no_rec)
  - (C_t1 + P_t1_rec*(1-P_t1_rel)*C_rec + P_t1_rec*P_t1_rel*C_rel
     +(1-P_t1_rec)*C_no_rec ))
  NB_t2 = (lamda*(P_t2_rec*(1-P_t2_rel)*Q_rec + P_t2_rec*P_t2_rel*Q_rel
                 +(1-P_t2_rec)*Q_no_rec)
  - (C_t2 + P_t2_rec*(1-P_t2_rel)*C_rec + P_t2_rec*P_t2_rel*C_rel
     +(1-P_t2_rec)*C_no_rec ))

  Pb = colMeans(pmax(pmax(NB_nt,NB_t1),NB_t2))
  Pf = Pb-pmax(pmax(colMeans(NB_nt), colMeans(NB_t1)), colMeans(NB_t2))
  if(M==2)
  {
    Pc = Pb-0.5*(pmax(pmax(NB_nt[1,], NB_t1[1,]),NB_t2[1,])
                 +pmax(pmax(NB_nt[2,], NB_t1[2,]), NB_t2[2,]))

  } else {
    Pc = Pb-0.5*(pmax(pmax(colMeans(NB_nt[1:(M/2),]),
                  colMeans(NB_t1[1:(M/2),])),colMeans(NB_t2[1:(M/2),]))
                +pmax(pmax(colMeans(NB_nt[((M/2)+1):M,]),
         colMeans(NB_t1[((M/2)+1):M,])), colMeans(NB_t2[((M/2)+1):M,])))
  }
  sum1[1] = sum1[1] + sum(Pf-Pc);
  sum1[2] = sum1[2] + sum((Pf-Pc)^2);
```

```
    sum1[3] = sum1[3] + sum((Pf-Pc)^3);
    sum1[4] = sum1[4] + sum((Pf-Pc)^4);
    sum1[5] = sum1[5] + sum(Pf);
    sum1[6] = sum1[6] + sum(Pf^2);
    sum1[7] = sum1[7] + M*N2;
  }
  return(sum1)
}
```

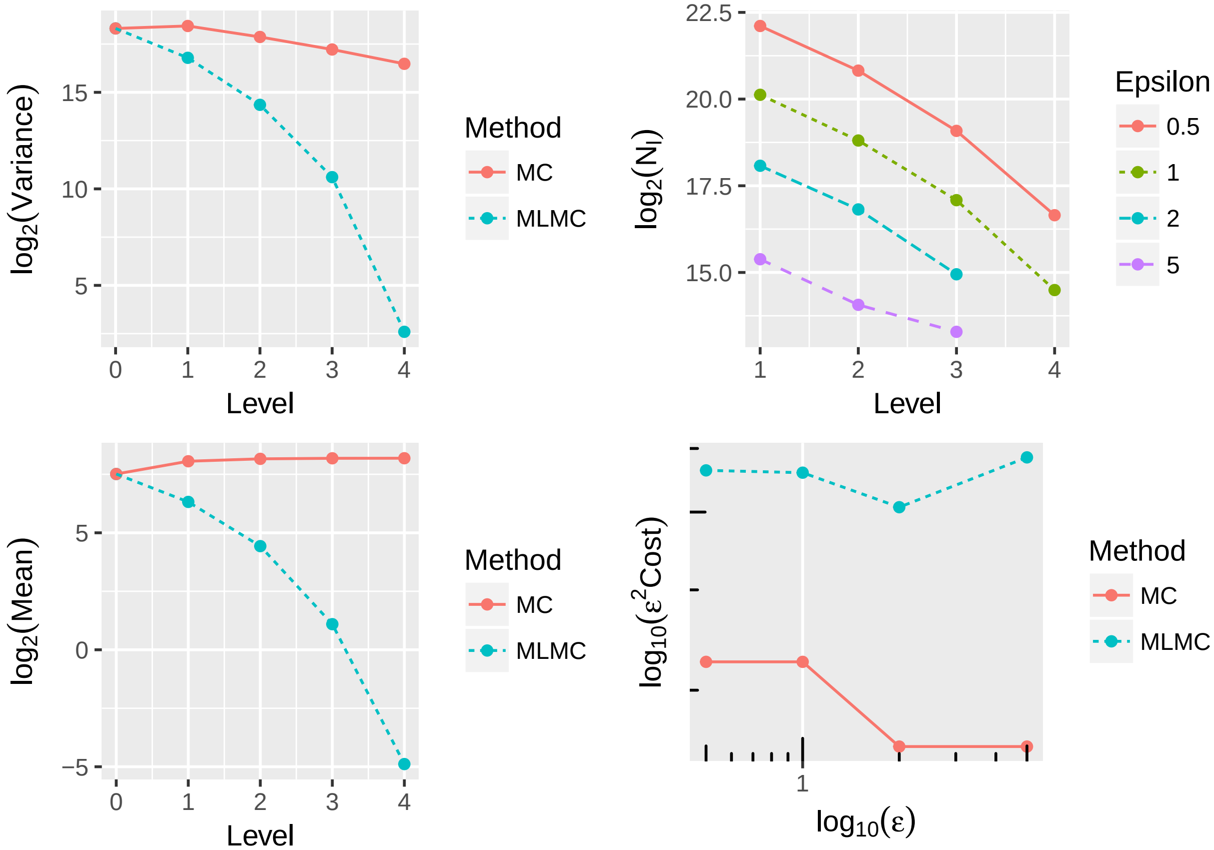
We can use the MLMC to calculate the value:

```
set.seed(666) # Set random seed to ensure the same output
# without this MLMC will give different output each time
tst <- mlmc.test(EVPPI_cq_l, M=2, N=10000,
                 L=4, N0=10000,
                 eps.v=c(0.5,1,2,5),
                 Lmin=2, Lmax=10, parallel = 32)
```

```
##
## **********************************************************
## *** Convergence tests, kurtosis, telescoping sum check ***
## **********************************************************
##
##  l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)    kurtosis     check
## -------------------------------------------------------------------------
##  0   1.8280e+02  1.8280e+02  3.2471e+05  3.2471e+05  0.0000e+00  0.0000e+00
##  1   7.9924e+01  2.6673e+02  1.1305e+05  3.5478e+05  5.4340e+01  8.8787e-02
##  2   2.1577e+01  2.8635e+02  2.0907e+04  2.3928e+05  2.4673e+02  5.3007e-02
##  3   2.1356e+00  2.9111e+02  1.5599e+03  1.5254e+05  6.9163e+02  9.5141e-02
##  4   3.3837e-02  2.9167e+02  6.0411e+00  9.1159e+04  8.9516e+03  2.5360e-02
##
##  WARNING: kurtosis on finest level = 8951.577164
##  indicates MLMC correction dominated by a few rare paths;
##  for (information on the connection to variance of sample variances,
##  see http://mathworld.wolfram.com/SampleVarianceDistribution.html
##
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 4.658330  (exponent for (MLMC weak convergence)
##  beta  in 5.878438  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## *****************************
## *** MLMC complexity tests ***
## *****************************
##
##   eps       value   mlmc_cost   std_cost  savings     N_l
## ---------------------------------------------------------
## 0.5000  2.8720e+02  2.254e+07  1.302e+07     0.58   4521747    1851928    555407    102866
## 1.0000  2.8785e+02  5.598e+06  3.254e+06     0.58   1142298     458085    139031     23050
## 2.0000  2.8728e+02  1.268e+06  6.381e+05     0.50    276434     115518     31584
```

```
## 5.0000   2.8820e+02   2.340e+05   1.021e+05      0.44      42678      17149      10000
```

```
plot(tst,which=c("var", "mean", "Nl", "cost"),cols=2) # plot
```



The estimation of EVPPI for C and Q is 287.8128835 with root Mean Square Error 1.

EVPPI for lor of CBT

Here is the code for **EVPPI_lor1_l.R** which requre `Matrix`, `MASS` and `boot` packages:

```
EVPPI_lor1_l <- function(l, N) {
  require(Matrix) # matrix computation
  require(MASS) # multivariate normal generator
  require(boot) # logit function

  # some constants and parameters in the model
  lamda <- 20000
  C_t1 <- 300
  C_t2 <- 30
  mu_rec <- c(0.99,1.33) # lor2
  sigma_rec <- matrix(c(0.22,0.15,0.15,0.20),2,2,byrow = TRUE)
  mu_rel <- c(-1.48,-0.4) #lor3
  sigma_rel <- matrix(c(0.14,0.05,0.05,0.11),2,2,byrow = TRUE)

  M <- 2^(l+1) # number of inner samples
  sum1 <- rep(0, 7)
```

```r
for(N1 in seq(1, N, by=10000)) {
  # generate N2 samples together for better computational efficiency
  N2 <- min(10000, N-N1+1)

  # Generate M*N2 samples of Probability of no treatment
  P_nt_rec <- matrix(rbeta(M*N2,6,200),M,N2,byrow = TRUE)
  P_nt_rel <- matrix(rbeta(M*N2,2,100),M,N2,byrow = TRUE)

  # Generate N2 samples of lor_2 and repeat it to M*N2 matrix
  lor_rec <- mvrnorm(N2, mu_rec, sigma_rec)
  lor_rel <- mvrnorm(N2, mu_rel, sigma_rel)
  lor_t1_rec <- matrix(rep(lor_rec[,1],M),M,N2,byrow = TRUE)
  lor_t1_rel <- matrix(rep(lor_rel[,1],M),M,N2,byrow = TRUE)
  P_t1_rec <- inv.logit(logit(P_nt_rec)+lor_t1_rec)
  P_t1_rel <- inv.logit(logit(P_nt_rel)+lor_t1_rel)

  # calculate the conditional normal distribution of the lor_t2 based on N2 samples of lor_t1
  mu_t2_rec = (lor_t1_rec-mu_rec[1])*sigma_rec[1,2]/sigma_rec[1,1]+mu_rec[2];
  mu_t2_rel = (lor_t1_rel-mu_rel[1])*sigma_rel[1,2]/sigma_rel[1,1]+mu_rel[2];
  sigma_t2_rec = sqrt(sigma_rec[2,2]-sigma_rec[1,2]^2/sigma_rec[1,1]);
  sigma_t2_rel = sqrt(sigma_rel[2,2]-sigma_rel[1,2]^2/sigma_rel[1,1]);
# Generate M*N2 samples of lor_t2 conditioned on lor_t1 and repeat it to M*N2 matrix
  lor_t2_rec = mu_t2_rec + sigma_t2_rec*matrix(rnorm(M*N2,0,1),M,N2,byrow=TRUE);
  lor_t2_rel = mu_t2_rel + sigma_t2_rel*matrix(rnorm(M*N2,0,1),M,N2,byrow=TRUE);
  P_t2_rec = inv.logit(logit(P_nt_rec)+lor_t2_rec);
  P_t2_rel = inv.logit(logit(P_nt_rel)+lor_t2_rel);

  # generate M*N2 independent inner samples
  C_rec = matrix(rnorm(M*N2,1000,50),M,N2,byrow = TRUE)
  C_rel = matrix(rnorm(M*N2,2000,100),M,N2,byrow = TRUE)
  C_no_rec = matrix(rnorm(M*N2,2500,125),M,N2,byrow = TRUE)

  Q_rec = matrix(rnorm(M*N2,26,2),M,N2,byrow = TRUE)
  Q_rel = matrix(rnorm(M*N2,23,3),M,N2,byrow = TRUE)
  Q_no_rec = matrix(rnorm(M*N2,20,4),M,N2,byrow = TRUE)

  # calculate the NB value for all 3 treatments of all samples using formula (5.1)
  NB_nt = (lamda*(P_nt_rec*(1-P_nt_rel)*Q_rec + P_nt_rec*P_nt_rel*Q_rel
                 +(1-P_nt_rec)*Q_no_rec)
  - (P_nt_rec*(1-P_nt_rel)*C_rec + P_nt_rec*P_nt_rel*C_rel
    +(1-P_nt_rec)*C_no_rec ))
  NB_t1 = (lamda*(P_t1_rec*(1-P_t1_rel)*Q_rec + P_t1_rec*P_t1_rel*Q_rel
                 +(1-P_t1_rec)*Q_no_rec)
  - (C_t1 + P_t1_rec*(1-P_t1_rel)*C_rec + P_t1_rec*P_t1_rel*C_rel
    +(1-P_t1_rec)*C_no_rec ))
  NB_t2 = (lamda*(P_t2_rec*(1-P_t2_rel)*Q_rec + P_t2_rec*P_t2_rel*Q_rel
                 +(1-P_t2_rec)*Q_no_rec)
  - (C_t2 + P_t2_rec*(1-P_t2_rel)*C_rec + P_t2_rec*P_t2_rel*C_rel
    +(1-P_t2_rec)*C_no_rec ))

  # the first term in estimator (9.1)
  Pb = colMeans(pmax(pmax(NB_nt,NB_t1),NB_t2))
```

```r
    # DIFF estimator (9.1) with M inner samples for level l
    Pf = Pb-pmax(pmax(colMeans(NB_nt), colMeans(NB_t1)), colMeans(NB_t2))

    # Average of two DIFF estimator with M/2 inner samples for level l-1
    if(M==2)
    {
      Pc = Pb-0.5*(pmax(pmax(NB_nt[1,], NB_t1[1,]),NB_t2[1,])
                   +pmax(pmax(NB_nt[2,], NB_t1[2,]), NB_t2[2,]))

    } else {
      Pc = Pb-0.5*(pmax(pmax(colMeans(NB_nt[1:(M/2),]),
                     colMeans(NB_t1[1:(M/2),])),colMeans(NB_t2[1:(M/2),]))
                   +pmax(pmax(colMeans(NB_nt[((M/2)+1):M,]),
                 colMeans(NB_t1[((M/2)+1):M,])), colMeans(NB_t2[((M/2)+1):M,])))
    }

    # update all the statistics needed by MLMC
    sum1[1] = sum1[1] + sum(Pf-Pc);
    sum1[2] = sum1[2] + sum((Pf-Pc)^2);
    sum1[3] = sum1[3] + sum((Pf-Pc)^3);
    sum1[4] = sum1[4] + sum((Pf-Pc)^4);
    sum1[5] = sum1[5] + sum(Pf);
    sum1[6] = sum1[6] + sum(Pf^2);
    sum1[7] = sum1[7] + M*N2;
  }
  return(sum1) # return statistics
}
```

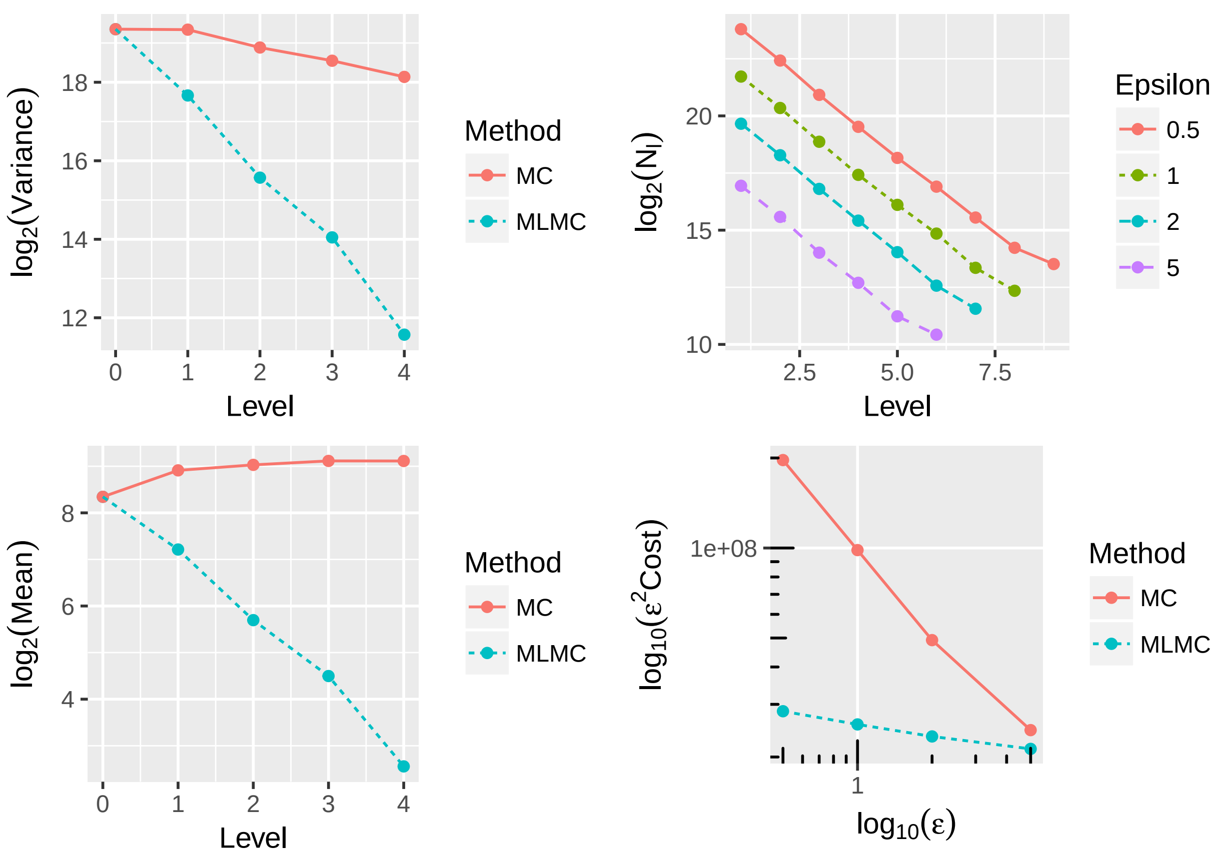
We can use the MLMC to calculate the value:

```r
set.seed(666) # Set random seed to ensure the same output
# without this MLMC will give different output each time
tst <- mlmc.test(EVPPI_lor1_l, M=2, N=10000,
                 L=4, N0=10000,
                 eps.v=c(0.5,1,2,5),
                 Lmin=2, Lmax=10, parallel = 32)
```

```
##
## **********************************************************
## *** Convergence tests, kurtosis, telescoping sum check ***
## **********************************************************
##
## l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)    kurtosis     check
## --------------------------------------------------------------------------
## 0   3.2492e+02  3.2492e+02  6.6945e+05  6.6945e+05  0.0000e+00  0.0000e+00
## 1   1.4834e+02  4.8171e+02  2.0766e+05  6.6378e+05  4.8424e+01  1.3479e-01
## 2   5.1858e+01  5.2265e+02  4.8650e+04  4.8418e+05  6.6135e+01  2.1006e-01
## 3   2.2573e+01  5.5440e+02  1.6945e+04  3.8341e+05  9.3511e+01  2.1159e-01
## 4   5.8828e+00  5.5381e+02  3.0403e+03  2.8826e+05  2.0209e+02  1.7826e-01
##
##  WARNING: kurtosis on finest level = 202.085385
##  indicates MLMC correction dominated by a few rare paths;
##  for (information on the connection to variance of sample variances,
##  see http://mathworld.wolfram.com/SampleVarianceDistribution.html
```

```
##
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 1.570001  (exponent for (MLMC weak convergence)
##  beta  in 2.000076  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## *****************************
## *** MLMC complexity tests ***
## *****************************
##
##   eps       value    mlmc_cost   std_cost  savings     N_l
## -----------------------------------------------------------
## 0.5000  5.6865e+02  1.138e+08  7.872e+08     6.92  14556679    5620269    1983560     752775     293384
## 1.0000  5.6864e+02  2.570e+07  9.839e+07     3.83   3457486    1332890     478818     175349      70573
## 2.0000  5.6762e+02  5.858e+06  1.230e+07     2.10    827542     318042     114507      43726      16814
## 5.0000  5.6584e+02  8.516e+05  9.839e+05     1.16    125874      49029      16553       6638       2404
```

```
plot(tst,which=c("var", "mean", "Nl", "cost"),cols=2) # plot
```



The estimation of EVPPI for lor of CBT is 6.3664424 with root Mean Square Error 1.

EVPPI for lor of antidepression

Here is the code for **EVPPI_lor2_l.R** which requre `Matrix`, `MASS` and `boot` packages:

```
EVPPI_lor1_l <- function(l, N) {
  require(Matrix)
  require(MASS) # mvrnorm
  require(boot) # logit

  lamda <- 20000
  C_t1 <- 300
  C_t2 <- 30
  M <- 2^(l+1)

  mu_rec <- c(0.99,1.33)
  sigma_rec <- matrix(c(0.22,0.15,0.15,0.20),2,2,byrow = TRUE)
  mu_rel <- c(-1.48,-0.4)
  sigma_rel <- matrix(c(0.14,0.05,0.05,0.11),2,2,byrow = TRUE)

  sum1 <- rep(0, 7)

  for(N1 in seq(1, N, by=10000)) {
    N2 <- min(10000, N-N1+1)

    P_nt_rec <- matrix(rbeta(M*N2,6,200),M,N2,byrow = TRUE)
    P_nt_rel <- matrix(rbeta(M*N2,2,100),M,N2,byrow = TRUE)

    lor_rec <- mvrnorm(N2, mu_rec, sigma_rec)
    lor_rel <- mvrnorm(N2, mu_rel, sigma_rel)

    # Generate N2 samples of lor_t2 and repeat it to M*N2 matrix
    lor_t2_rec <- matrix(rep(lor_rec[,2],M),M,N2,byrow = TRUE)
    lor_t2_rel <- matrix(rep(lor_rel[,2],M),M,N2,byrow = TRUE)
    P_t2_rec <- inv.logit(logit(P_nt_rec)+lor_t2_rec)
    P_t2_rel <- inv.logit(logit(P_nt_rel)+lor_t2_rel)

    # calculate the conditional normal distribution of the lor_t1 based on N2 samples of lor_t2
    mu_t1_rec = (lor_t2_rec-mu_rec[2])*sigma_rec[1,2]/sigma_rec[2,2]+mu_rec[1];
    mu_t1_rel = (lor_t2_rel-mu_rel[2])*sigma_rel[1,2]/sigma_rec[2,2]+mu_rel[1];
    sigma_t1_rec = sqrt(sigma_rec[1,1]-sigma_rec[1,2]^2/sigma_rec[2,2]);
    sigma_t1_rel = sqrt(sigma_rel[1,1]-sigma_rel[1,2]^2/sigma_rel[2,2]);

    # Generate M*N2 samples of lor_t2 conditioned on lor_t1 and repeat it to M*N2 matrix
    lor_t1_rec = mu_t1_rec + sigma_t1_rec*matrix(rnorm(M*N2,0,1),M,N2,byrow=TRUE);
    lor_t1_rel = mu_t1_rel + sigma_t1_rel*matrix(rnorm(M*N2,0,1),M,N2,byrow=TRUE);
    P_t1_rec = inv.logit(logit(P_nt_rec)+lor_t1_rec);
    P_t1_rel = inv.logit(logit(P_nt_rel)+lor_t1_rel);

    C_rec = matrix(rnorm(M*N2,1000,50),M,N2,byrow = TRUE)
    C_rel = matrix(rnorm(M*N2,2000,100),M,N2,byrow = TRUE)
    C_no_rec = matrix(rnorm(M*N2,2500,125),M,N2,byrow = TRUE)

    Q_rec = matrix(rnorm(M*N2,26,2),M,N2,byrow = TRUE)
    Q_rel = matrix(rnorm(M*N2,23,3),M,N2,byrow = TRUE)
```

```r
    Q_no_rec = matrix(rnorm(M*N2,20,4),M,N2,byrow = TRUE)

    NB_nt = (lamda*(P_nt_rec*(1-P_nt_rel)*Q_rec + P_nt_rec*P_nt_rel*Q_rel
                  +(1-P_nt_rec)*Q_no_rec)
    - (P_nt_rec*(1-P_nt_rel)*C_rec + P_nt_rec*P_nt_rel*C_rel
       +(1-P_nt_rec)*C_no_rec ))
    NB_t1 = (lamda*(P_t1_rec*(1-P_t1_rel)*Q_rec + P_t1_rec*P_t1_rel*Q_rel
                  +(1-P_t1_rec)*Q_no_rec)
    - (C_t1 + P_t1_rec*(1-P_t1_rel)*C_rec + P_t1_rec*P_t1_rel*C_rel
       +(1-P_t1_rec)*C_no_rec ))
    NB_t2 = (lamda*(P_t2_rec*(1-P_t2_rel)*Q_rec + P_t2_rec*P_t2_rel*Q_rel
                  +(1-P_t2_rec)*Q_no_rec)
    - (C_t2 + P_t2_rec*(1-P_t2_rel)*C_rec + P_t2_rec*P_t2_rel*C_rel
       +(1-P_t2_rec)*C_no_rec ))

    Pb = colMeans(pmax(pmax(NB_nt,NB_t1),NB_t2))
    Pf = Pb-pmax(pmax(colMeans(NB_nt), colMeans(NB_t1)), colMeans(NB_t2))
    if(M==2)
    {
      Pc = Pb-0.5*(pmax(pmax(NB_nt[1,], NB_t1[1,]),NB_t2[1,])
                  +pmax(pmax(NB_nt[2,], NB_t1[2,]), NB_t2[2,]))

    } else {
      Pc = Pb-0.5*(pmax(pmax(colMeans(NB_nt[1:(M/2),]),
                         colMeans(NB_t1[1:(M/2),])),colMeans(NB_t2[1:(M/2),]))
                  +pmax(pmax(colMeans(NB_nt[((M/2)+1):M,]),
               colMeans(NB_t1[((M/2)+1):M,])), colMeans(NB_t2[((M/2)+1):M,])))
    }
    sum1[1] = sum1[1] + sum(Pf-Pc);
    sum1[2] = sum1[2] + sum((Pf-Pc)^2);
    sum1[3] = sum1[3] + sum((Pf-Pc)^3);
    sum1[4] = sum1[4] + sum((Pf-Pc)^4);
    sum1[5] = sum1[5] + sum(Pf);
    sum1[6] = sum1[6] + sum(Pf^2);
    sum1[7] = sum1[7] + M*N2;
  }
  return(sum1)
}
```

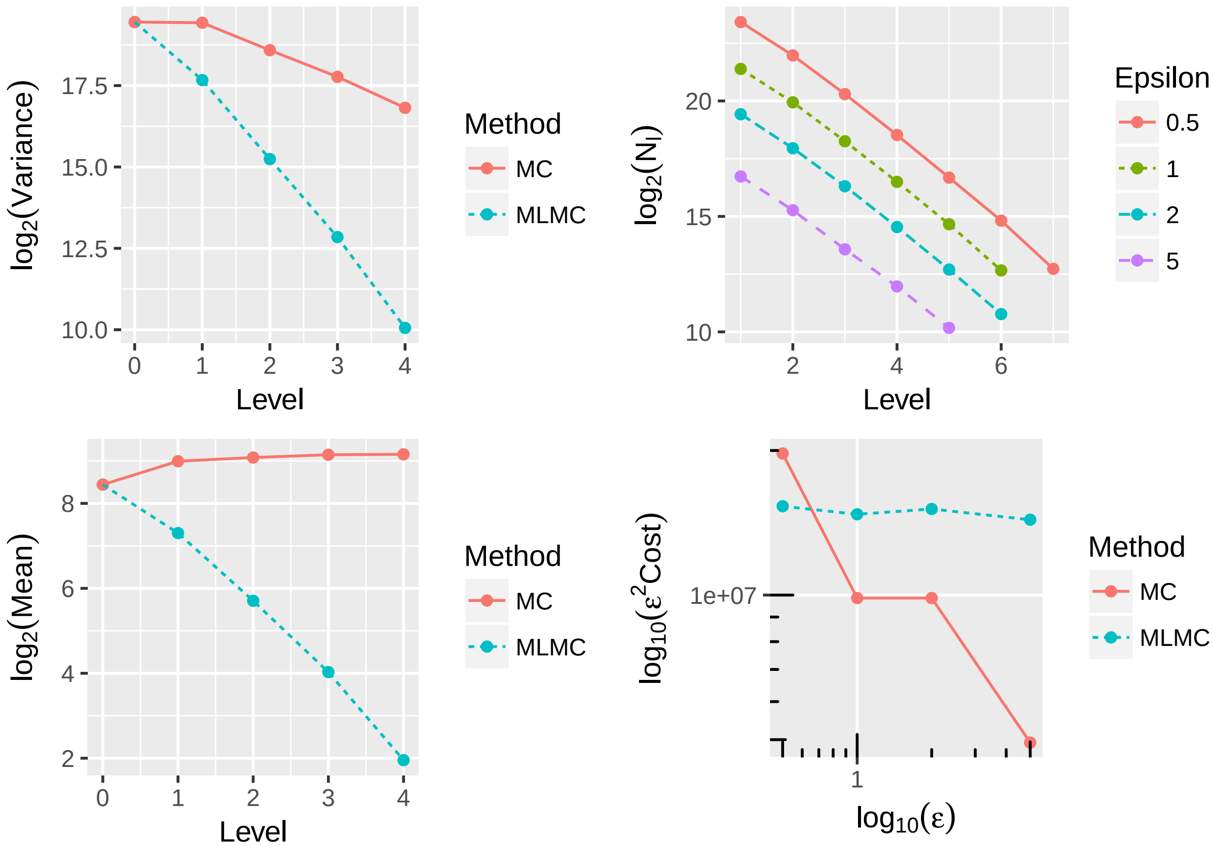
We can use the MLMC to calculate the value:

```r
set.seed(666) # Set random seed to ensure the same output
# without this MLMC will give different output each time
tst <- mlmc.test(EVPPI_lor2_l, M=2, N=10000,
                 L=4, N0=10000,
                 eps.v=c(0.5,1,2,5),
                 Lmin=2, Lmax=10, parallel = 32)
```

```
##
## **********************************************************
## *** Convergence tests, kurtosis, telescoping sum check ***
## **********************************************************
##
## l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)    kurtosis     check
```

```
## ---------------------------------------------------------------------------
## 0   3.4775e+02  3.4775e+02  7.1715e+05  7.1715e+05  0.0000e+00  0.0000e+00
## 1   1.5773e+02  5.0980e+02  2.0861e+05  7.0674e+05  4.2171e+01  6.7084e-02
## 2   5.2193e+01  5.4122e+02  3.8737e+04  3.9428e+05  7.2387e+01  4.1580e-01
## 3   1.6311e+01  5.6624e+02  7.3573e+03  2.2332e+05  8.1351e+01  2.4485e-01
## 4   3.8734e+00  5.7030e+02  1.0637e+03  1.1552e+05  2.5227e+02  7.4581e-03
##
##  WARNING: kurtosis on finest level = 252.273782
##  indicates MLMC correction dominated by a few rare paths;
##  for (information on the connection to variance of sample variances,
##  see http://mathworld.wolfram.com/SampleVarianceDistribution.html
##
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 1.876090  (exponent for (MLMC weak convergence)
##  beta  in 2.593240  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## *****************************
## *** MLMC complexity tests ***
## *****************************
##
##   eps       value   mlmc_cost   std_cost  savings     N_l
## --------------------------------------------------------
## 0.5000  5.7475e+02  6.124e+07  7.886e+07     1.29  11191739   4111534   1287314    377388    104929
## 1.0000  5.7251e+02  1.473e+07  9.858e+06     0.67   2740902   1006357    312613     92423     25917
## 2.0000  5.7708e+02  3.777e+06  2.464e+06     0.65    704854    253380     81059     23819      6646
## 5.0000  5.6872e+02  5.740e+05  1.972e+05     0.34    108657     39503     12210      4006      1153
```

```r
plot(tst,which=c("var", "mean", "Nl", "cost"),cols=2) # plot
```



The estimation of EVPPI for lor of antidepression is 0.2655106 with root Mean Square Error 1.

Summary

MLMC works well for all the calculations and starts to show the computaional savings when calculating the EVPPI for lor, i.e. involving the conditional sampling.

Comparision with QMC

For comparision, we set the **mean square error (MSE)** to be 0.25 ($\varepsilon = 0.5$) and divide it into two parts: **weak error**: 0.25, **variance**: 0.1875.

We first run the **MLMC** function to achieve the required MSE, which gives the computational costs of the MLMC and standard MC. Then, by using the same number of inner samples, we do QMC on the outer samples and achieve the required variance. This kind of comparision is relatively unfair for MLMC, since other algorithm do not need to find the required number of inner samples.

Table 2: Comparision of Computational Cost (10^6)

	Standard MC	MLMC	QMC
EVPPI_P	15.14	26.93	5.63
EVPPI_cq	13.02	22.85	5.12
EVPPI_lor1	787.20	108.50	8.19