# MLMC Code Demo

Howard Thom

04/02/2021

## Introduction

Script to demonstrate EVPPI using MLMC for a simple net benefit function This can be applied to any economic model implemented in R. The primary modifications are to the generate_input_parameters and generate_net_benefit functions. The first creates a data frame of sampled parameters; the function must include the option to keep all or a subset of sampled parameters fixed for the whole sample. The latter function converts these sampled parameters into net benefits.

Following these primary modifications, the remaining steps are simple. Modify the EVPPI_x_std_p function, which uses the two primary functions to generate net benefits for a number of samples M*N, and EVPPI_x_l_p, which acts as a wrapper for the general level l estimating function EVPPI_l_p and passing it EVPPI_x_std_p.

Finally, run mlmc.test with EVPPI_x_l_p as an input and other specifications for the MLMC algorithm.

The code is explained below.

The example economic mdoel has a net benefit function for two decision options

$$NB_1(x,y) = 400x + 200y$$

$$NB_0(x,y) = 0$$

And these call two parameters

$$x \sim Normal(0,2), y \sim Normal(2,4)$$

## R Code

First load the necessary packages. Note that bcea is loaded only for comparison with the MLMC estimates.

```
require(grid)
require(Rcpp)
require(doRNG)
library(BCEA)
require(mlmc)
```

# MLMC source files

These are based on GPL-2 'Matlab' code by Mike Giles (http://people.maths.ox.ac.uk/~gilesm/mlmc/) Authors are Louis Aslett aslett@stats.ox.ac.uk, Mike Giles Mike.Giles@maths.ox.ac.uk, and Tigran Nagapetyan nagapetyan@stats.ox.ac.uk

```
# MLMC code to calculate the value to some accuracy
source("mlmc.R")
# Test function of the convergence rates and output the MLMC results
source("mlmc.test.R")
```

## Model definition

Here we define global parameters and the economic model functions. This is where the primary modifications are needed to run this script on other models.

```
# Global options
n_samples = 1000 # Only used for EVPI
n_treatment <- 2

# Net benefit functions are  NB1(x,y) = 400x+200y, NB0(x,y) = 0
generate_net_benefit <- function(n_samples, input_parameters) {
  NetB <- matrix(nrow = n_samples, ncol = n_treatment)
  NetB[, 1] <- 400 * input_parameters$x + 200 * input_parameters$y
  NetB[, 2] <- rep(0, n_samples)

  return(NetB)
}

# Two random parameters x~Normal(0, 2), y~Normal(2, 4)
generate_input_parameters <- function(n_samples, hold_constant = c()) {
  input_parameters <- as.data.frame(matrix(nrow = n_samples, ncol = 2))
  colnames(input_parameters) <- c("x", "y")

  # Only generate one random value for parameters in the hold_constant vector
  if(!is.element("x", hold_constant)) {
    input_parameters[, "x"] <- rnorm(n_samples, mean = 0, sd = sqrt(2))
  } else {
    input_parameters[, "x"] <- rnorm(1, mean = 0, sd = sqrt(2))
  }
  if(!is.element("y", hold_constant)) {
    input_parameters[, "y"] <- rnorm(n_samples, mean = 2, sd = sqrt(4))
  } else {
    input_parameters[, "y"] <- rnorm(1, mean = 2, sd = sqrt(4))
  }

  return(input_parameters)
}
```

## Wrappers for parameter and net benefit functions

The two wrappers below are needed for EVPPI of $x$ and EVPPI of $y$

```
# Wrapper function to generate the net benefit holding the parameters of interest constant
# Repeats parameters of interest M times and generates random for remainder
# Calculates net benefit based on these
EVPPI_x_std_p<-function(M,N)
{
  # N is the number of outer samples, M=2^l is the number of inner samples
  # Total number of samples is NN
  NN <- M*N

  input_parameters <- generate_input_parameters(n_samples = NN, hold_constant = c("x"))
  NetB <- generate_net_benefit(n_samples = NN, input_parameters = input_parameters)
```

```
  return(NetB)
}

# Correpsonding wrapper function for y
EVPPI_y_std_p<-function(M,N)
{
  NN <- M*N
  input_parameters <- generate_input_parameters(n_samples = NN, hold_constant = c("y"))
  NetB <- generate_net_benefit(n_samples = NN, input_parameters = input_parameters)
  return(NetB)
}
```

# MLMC level l estimator

Function to provide level $l$ difference estimate using estimator $d_\ell^{(n)}$ based on parameters and net benefit function.

See the main paper for details but the fine $l$ estimator is

$$e_\ell^{(n)} = \max_{d\in D} \frac{1}{2^\ell} \sum_{m=1}^{2^\ell} f_d(X^{(n)}, Y^{(n,m)}) - \max_{d\in D} \frac{1}{2^\ell N} \sum_{i=1}^{N} \sum_{m=1}^{2^\ell} f_d(X^{(i)}, Y^{(i,m)}).$$

While the coarse $l-1$ estimator is

$$e_{\ell-1}^{(n)} \; = \; \frac{1}{2}\left[\max_{d\in D} \frac{1}{2^{\ell-1}} \sum_{m=1}^{2^{\ell-1}} f_d(X^{(n)}, Y^{(n,m)}) + \max_{d\in D} \frac{1}{2^{\ell-1}} \sum_{m=2^{\ell-1}+1}^{2^\ell} f_d(X^{(n)}, Y^{(n,m)})\right] - \max_{d\in D} \frac{1}{2^\ell N} \sum_{i=1}^{N} \sum_{m=1}^{2^\ell} f_d(X^{(i)}, Y^{(i,m)})$$

The difference estimator $d_\ell^{(n)}$ is then

$$d_\ell^{(n)} \; = \; \max_{d\in D} \frac{1}{2^\ell} \sum_{m=1}^{2^\ell} f_d(X^{(n)}, Y^{(n,m)}) - \frac{1}{2}\left[\max_{d\in D} \frac{1}{2^{\ell-1}} \sum_{m=1}^{2^{\ell-1}} f_d(X^{(n)}, Y^{(n,m)}) + \max_{d\in D} \frac{1}{2^{\ell-1}} \sum_{m=2^{\ell-1}+1}^{2^\ell} f_d(X^{(n)}, Y^{(n,m)})\right]$$

The corresponding MLMC estimator of DIFF (=EVPI-EVPPI) is

$$\widehat{\mathrm{DIFF}}_\ell^*(N_1, N_2, ..., N_L) = -\sum_{\ell=1}^{L} d_\ell^{(n)}$$

```
EVPPI_l_p<-function(l,N, EVPPI_std_p = NULL)
{
  if(is.null(EVPPI_std_p)) return("EVPPI_std_p must be supplied")
  # This function generates all the random samples for the EVPPI calculation
  # Then calculates the net benefit function

  # N is the number of outer samples, M=2^l is the number of inner samples
  sum1 <- rep(0, 7)
  # Need to store results of different stats
  M = 2^(l + 1)
  Np = max(M, 128)

  inputs = 1:ceiling(M * N / Np)
```

```
  # Iterate over the inputs
  # Can add parallel computation here
  results <- foreach(i=inputs,.export=c("n_samples", "n_treatment",
                                       "generate_net_benefit", "generate_input_parameters",
                                       'EVPPI_x_std_p'),.packages='MASS') %dorng% {
                                         NN=min(Np, N*M-(i-1)*Np)
                                         ######################################################
                                         EVPPI_std_p(M,NN/M)

                                         ##########################################33########
                                       }
  #NetB <- EVPPI_x_std_p(M, N)

# Convert results of foreach to NetB matrix
  # If not interested in parallelisation could merge the foreach and for loop to simplify
  NetB = matrix(NA, M*N, n_treatment)
  for(i in inputs){
    NN <- min(Np, N*M-(i-1)*Np)
    nn = min(i*Np,N*M)
    NetB[(nn-NN+1):nn,] = matrix(unlist(results[i]),NN,n_treatment)
  }
  # Net benefit based on partial perfect information for every sample
  NetB_max = apply(NetB,1,max)
  # Expected value of max over each set of inner samples
  NetB_max_sample = apply(matrix(NetB_max,N,M,byrow=TRUE),1,mean)
  # Matrices needed for antithetic variable variance reduction
  NetB_low_c_1 = matrix(NA,N,n_treatment)
  NetB_low_c_2 = matrix(NA,N,n_treatment)
  NetB_low_f = matrix(NA,N,n_treatment)
  for(n in 1:n_treatment){
    # Net benefts for treatment n in matrix of size outer by inner samples
    temp = matrix(NetB[,n],N,M,byrow=TRUE)
    # Average net benefit over each set of inner samples
    NetB_low_f[,n] = apply(temp,1,mean)
    # Antithetic variable construction splits samples into first and second halves
    # Split formula into cases l=0 and l>0
    if(M==2){
      NetB_low_c_1[,n] = temp[,1]
      NetB_low_c_2[,n] = temp[,2]
    }else{
      if(N==1){
        NetB_low_c_1[,n] = sum(temp[,1:max(M/2,2)])/max(M/2,2)
        NetB_low_c_2[,n] = sum(temp[,min(M/2+1,M-1):M])/(M-min(M/2+1,M-1)+1)
      }else{
        NetB_low_c_1[,n] = rowSums(temp[,1:max(M/2,2)])/max(M/2,2)
        NetB_low_c_2[,n] = rowSums(temp[,min(M/2+1,M-1):M])/(M-min(M/2+1,M-1)+1)
      }

    }
  }
  NetB_low_f_sample = apply(NetB_low_f,1,max)
  # Put antithetic variable construction together
  NetB_low_c_sample = (apply(NetB_low_c_1,1,max)+apply(NetB_low_c_2,1,max))/2
```

```
  # Fine estimator (i.e. e_l^(n))
  Pf = NetB_max_sample - NetB_low_f_sample
  # Coarse estimator (i.e. e_(l-1)^(n))
  Pc = NetB_max_sample - NetB_low_c_sample

  # Sum the moments of the estimator
  # First is the mean of the difference estimator d_l^(n)
  # Summing these difference estimates gives an estimate of DIFF= EVPI-EVPPI
  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*N;

  return(sum1)
}
```

# Wrapper for EVPPI_l_p

These functions are needed for the EVPPI of $x$ and EVPPI of $y$ and pass the necessary EVPPI_std_p to the level l estimator EVPPI_l_p

```
# Wrapper function for EVPPI of x
EVPPI_x_l_p <- function(l = l,N = N) {
  return(EVPPI_l_p(l, N, EVPPI_std_p = EVPPI_x_std_p))
}
# Wrapper function for EVPPI of y
EVPPI_y_l_p <- function(l = l,N = N) {
  return(EVPPI_l_p(l, N, EVPPI_std_p = EVPPI_y_std_p))
}
```

# EVPI and EVPPI by regression

For comparison, we use GP model from BCEA to estimate the EVPPI of $x$ and $y$

```
# EVPI is about 121.08
bcea_object$evi
```

```
## [1] 124.7058
```

```
# EVPPI of x is about 63/74/91/75/65  Average 73.6
bcea_evppi_x$evppi
```

```
## [1] 84.82218
```

```
# EVPPI for y is about 23/34/46/37/20 Average 32
bcea_evppi_y$evppi
```

```
## [1] 35.57351
```

# EVPPI of x using MLMC

Now use the mlmc.test() function to estimate the EVPPI of $x$.

M refinement cost factor $2^{\gamma}$ in the general MLMC Throrem) N number of samples to use in the tests L number of levels to use in the tests N0 initial number of samples which are used for the first 3 levels and for any subsequent levels which are automatically added. Must be >0 eps.v a vector of all the target accuracies in the tests. Must all be >0 Lmin the minimum level of refinement. Must be $\geq 2$ Lmax the maximum level of refinement. Must be $\geq$ Lmin

```
set.seed(33)
tst_x <- mlmc.test(EVPPI_x_l_p, M=2, N=1024,
                   L=4, N0=1024,
                   eps.v=c(60,30,15,7,3,1),
                   Lmin=2, Lmax=10)
```

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

## Warning: executing %dopar% sequentially: no parallel backend registered

##  0   2.3338e+01  2.3338e+01  3.3343e+03  3.3343e+03  0.0000e+00  0.0000e+00
##  1   1.0059e+01  2.9487e+01  9.4862e+02  2.5981e+03  1.8955e+01  2.9895e-01
##  2   5.0736e+00  3.8240e+01  3.4724e+02  2.3560e+03  2.6048e+01  3.3219e-01
##  3   2.6050e+00  4.0921e+01  1.3008e+02  2.2648e+03  4.1764e+01  7.4623e-03
##  4   1.2841e+00  4.4510e+01  4.4248e+01  2.2174e+03  5.7800e+01  2.4267e-01
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 0.991101  (exponent for (MLMC weak convergence)
##  beta  in 1.486120  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## *****************************
## *** MLMC complexity tests ***
## *****************************
##
##   eps       value   mlmc_cost   std_cost  savings     N_l
## -----------------------------------------------------------
## 60.0000  3.4719e+01  1.434e+04  6.981e+00     0.00      1024      1024      1024
## 30.0000  3.8124e+01  1.434e+04  2.792e+01     0.00      1024      1024      1024
## 15.0000  3.5871e+01  1.434e+04  1.117e+02     0.01      1024      1024      1024
## 7.0000  3.0580e+01  1.461e+04  9.860e+02     0.07      1024      1024      1024        17
## 3.0000  4.2076e+01  1.919e+04  1.051e+04     0.55      1677      1024      1024       136        43
## 1.0000  4.0766e+01  1.424e+05  3.784e+05     2.66     15466      7219      2711      1424       548
```

```
# EVPPI is lowest eps estimate
# Remember MLMC estimates DIFF from EVPI so need to subtract this from total EVPPI
# The EVPI in this case using 10^7 samples is 121
121 - tst_x$P[length(tst_x$P)]
```

```
## [1] 80.23447
```

# EVPPI of y using MLMC

Now use the mlmc.test() function to estimate the EVPPI of $y$.

```r
# Calculate the EVPPI for y
set.seed(123)
tst_y <- mlmc.test(EVPPI_y_l_p, M=2, N=1024,
                   L=4, N0=1024,
                   eps.v=c(60,30,15,7),
                  Lmin=2, Lmax=10)
```

```
##
## **********************************************************
## *** Convergence tests, kurtosis, telescoping sum check ***
## **********************************************************
##
##  l   ave(Pf-Pc)    ave(Pf)   var(Pf-Pc)    var(Pf)    kurtosis     check
## -------------------------------------------------------------------------
##  0   4.7667e+01  4.7667e+01  8.7237e+03  8.7237e+03  0.0000e+00  0.0000e+00
##  1   2.5744e+01  7.2458e+01  3.3345e+03  7.1803e+03  1.2056e+01  4.3084e-02
##  2   1.0585e+01  7.5732e+01  9.8151e+02  5.3056e+03  2.1614e+01  4.1284e-01
##  3   6.4307e+00  8.5748e+01  4.6089e+02  4.9403e+03  2.4915e+01  2.3240e-01
##  4   3.2644e+00  8.8795e+01  1.7613e+02  4.3195e+03  3.9201e+01  1.5534e-02
##
## ******************************************************
## *** Linear regression estimates of MLMC parameters ***
## ******************************************************
##
##  alpha in 0.848566  (exponent for (MLMC weak convergence)
##  beta  in 1.239168  (exponent for (MLMC variance)
##  gamma in 1.000000  (exponent for (MLMC cost)
##
## *****************************
## *** MLMC complexity tests ***
## *****************************
##
##   eps       value   mlmc_cost   std_cost  savings     N_l
## ---------------------------------------------------------
## 60.0000  8.1438e+01  1.434e+04  1.572e+01     0.00      1024      1024      1024
## 30.0000  8.1093e+01  1.434e+04  6.288e+01     0.00      1024      1024      1024
## 15.0000  9.3721e+01  1.539e+04  1.638e+03     0.11      1024      1024      1024          24          11
## 7.0000  9.1478e+01  1.626e+04  3.761e+03     0.23      1024      1024      1024          66          27
```

```r
# EVPPI is lowest eps estimate
# Remember MLMC estimates DIFF from EVPI so need to subtract this from total EVPPI
# The EVPI in this case using 10^7 samples is 121
121 - tst_y$P[length(tst_y$P)]
```

```
## [1] 29.52207
```