Monte-Carlo examples in R

Serial vs parallel code

Meelis Utt

Setup

```
# load necessary package
library(magrittr)
library(data.table)
library(ggplot2)
# library(dplyr)
# library(knitr)
library(parallel)
library(foreach)
library(doParallel)
```

```
## Loading required package: iterators
```

```
knitr::opts_chunk$set(fig.width = 5, fig.heighta = 3)
options(scipen = 1000)
```

Let's source the setup (function, analytical solution, number of iterations).

```
source("Setup.R",echo = T)
```

```
##
## > options(scipen = 1000)
##
## > n <- 10000000
##
## > header <- c("n", "computational result", "analytical result",
## +     "error", "time")
##
## > f <- function(x) {
## +     return(x^2 + x^4 + sin(x) + cos(x) + x^25)
## + }
##
## > analytical <- integrate(f, 0, 1)$value
```

Serial implementations

Let's start with a simple implementation of Monte-Carlo method.

```
MCser1 <- function(n){
  start <- Sys.time()
  i <- runif(n,0,1)
  EX <- mean(f(i))
```

```
  end <- Sys.time()
  time <- difftime(end,start)
  error <- EX - analytical %>% abs
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCser1(n)
```

```
## [1] 10000000.00000000000        1.87294669155        1.87296355073
## [4]       -0.00001685919        2.43539118767
```

Let's try a bit more vectorized solution, using the apply function family.

```
MCser2 <- function(n,ncols=1000){
  start <- Sys.time()
  dt <- matrix(runif(n,0,1),ncol = ncols)
  EX <- sapply(1:ncols,function(i,dt){
      EX <- dt[,i] %>% f %>% mean
    },dt) %>% mean
  end <- Sys.time()
  time <- difftime(end,start)
  error <- EX - analytical
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCser2(n)
```

```
## [1] 10000000.0000000000        1.8727003921        1.8729635507
## [4]       -0.0002631586        1.8938279152
```

Let's try an apprach using data.table.

```
MCser3 <- function(n){
  start <- Sys.time()
  dt <- data.table(unif = runif(n,0,1))
  EX <- dt[,.(EX = mean(f(unif)))] %>% unlist %>% unname
  end <- Sys.time()
  time <- difftime(end,start)
  error <- EX - analytical
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCser3(n)
```

```
## [1] 10000000.0000000000        1.8733368159        1.8729635507
## [4]        0.0003732651        1.7508175373
```

Let's try divide-and-conquer approach with data.table.

```
MCser4 <- function(n,ncols=1000){
  start <- Sys.time()
  dt <- matrix(runif(n,0,1),ncol=ncols) %>% data.table
  EX <- dt[,lapply(.SD,f)][,lapply(.SD,mean)][,.(EX = sum(.SD)/ncols)] %>% unlist %>% unname
  end <- Sys.time()
  time <- difftime(end,start)
  error <- EX - analytical
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCser4(n)
```

```
## [1] 10000000.00000000000        1.87302751896        1.87296355073
## [4]        0.00006396823        1.89381122589
```

Parallel implementations

Let's try different parallel implementations. First let's start with package *parallel*.

```
MCpar1 <- function(n){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
  # Initiate cluster
  cl <- makeCluster(no_cores)
  intermean <- parSapply(cl, rep(n/no_cores,no_cores),function(ni,f){
      EX <- mean(f(runif(ni,0,1)))
    },f
  )
  stopCluster(cl)
  EX <- mean(intermean)
  end <- Sys.time()
  time <- difftime(end,start)
  error <- EX - analytical
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCpar1(n)
```

```
## [1] 10000000.00000000000        1.87303533481        1.87296355073
## [4]        0.00007178408        1.35661602020
```

Now let's try approach analogical to MCser2.

```
MCpar2 <- function(n){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
  cl <- makeCluster(no_cores)
```

```
  dt <- matrix(runif(n,0,1),ncol = no_cores)
  intermean <- parSapply(cl, 1:no_cores,function(i,f,dt){
      EX <- mean(f(dt[,i]))
    },f,dt
  )
  stopCluster(cl)
  EX <- mean(intermean)
  end <- Sys.time()
  error <- EX - analytical
  time <- difftime(end,start)
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCpar2(n)
```

```
## [1] 10000000.0000000000        1.8728338755        1.8729635507
## [4]       -0.0001296752        3.8680872917
```

This approach was not very good. But let's have one more try at analogical solution to MCser2.

```
MCpar2_2 <- function(n,ncols=1000){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
  cl <- makeCluster(no_cores)
  dt <- matrix(runif(n,0,1),ncol = ncols)
  intermean <- parSapply(cl, 1:ncols,function(i,f,dt){
      EX <- mean(f(dt[,i]))
    },f,dt
  )
  stopCluster(cl)
  EX <- mean(intermean)
  end <- Sys.time()
  error <- EX - analytical
  time <- difftime(end,start)
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCpar2_2(n)
```

```
## [1] 10000000.000000000000        1.872959927135        1.872963550735
## [4]       -0.000003623599        3.687784671783
```

This solution was bit better, but still worse than the previous examples.
Let's try the package *foreach* now.

```
MCpar3 <- function(n){
  start <- Sys.time()
  # Calculate the number of cores
  no_cores <- detectCores()
```

```
  # Initiate cluster
  cl<-makeCluster(no_cores)
  registerDoParallel(cl)
  EX <- foreach(ni = rep(n/no_cores,no_cores),.combine=mean,.export="f") %dopar% mean(f(runif(ni,0,1)))
  stopImplicitCluster()
  end <- Sys.time()
  time <- difftime(end,start)
  error <- EX - analytical
  return(c(n,EX,analytical,error,time))
}
header
```

```
## [1] "n"                    "computational result" "analytical result"
## [4] "error"                "time"
```

```
MCpar3(n)
```

```
## [1] 10000000.000000000        1.873372468        1.872963551        0.000408917
## [5]        1.439738750
```