Monte-Carlo examples in R

Serial vs parallel code

Meelis Utt

Setup

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", "analytical", "error",
## +     "walltime", "type")
##
## > 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,"MCser1"))
}
data.table(t(MCser1(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87312157698611 1.87296355073463 0.000158026251479537
##           walltime   type
## 1: 2.1086106300354 MCser1
```

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

```
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) %>% abs
  return(c(n,EX,analytical,error,time,"MCser2"))
}
data.table(t(MCser2(n))) %>% setNames(header)
```

```
##           n    computational       analytical                 error
## 1: 10000000 1.87300128353283 1.87296355073463 0.0000377327982024056
##           walltime   type
## 1: 1.7653968334198 MCser2
```

Let's try an approach 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) %>% abs
  return(c(n,EX,analytical,error,time,"MCser3"))
}
data.table(t(MCser3(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87298681647747 1.87296355073463 0.000023265742841172
##            walltime   type
## 1: 1.66933250427246 MCser3
```

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) %>% abs
  return(c(n,EX,analytical,error,time,"MCser4"))
}
data.table(t(MCser4(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87284836166252 1.87296355073463 0.000115189072112054
##            walltime   type
## 1: 1.81064772605896 MCser4
```

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
  )
  on.exit(stopCluster(cl))
  EX <- mean(intermean)
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCpar1"))
}
data.table(t(MCpar1(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87309074553834 1.87296355073463 0.000127194803714481
##            walltime   type
## 1: 1.22565364837646 MCpar1
```

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
  )
  on.exit(stopCluster(cl))
  EX <- mean(intermean)
  end <- Sys.time()
  error <- (EX - analytical) %>% abs
  time <- difftime(end,start)
  return(c(n,EX,analytical,error,time,"MCpar2"))
}
data.table(t(MCpar2(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87326255732499 1.87296355073463 0.000299006590361861
##            walltime   type
## 1: 3.25085091590881 MCpar2
```

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
  )
  on.exit(stopCluster(cl))
  EX <- mean(intermean)
  end <- Sys.time()
  error <- (EX - analytical) %>% abs
  time <- difftime(end,start)
  return(c(n,EX,analytical,error,time,"MCpar2_2"))
}
data.table(t(MCpar2_2(n))) %>% setNames(header)
```

```
##           n    computational       analytical                error
## 1: 10000000 1.87314601528394 1.87296355073463 0.000182464549312922
##           walltime     type
## 1: 3.1351101398468 MCpar2_2
```

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)))
  on.exit(stopCluster(cl))
  # stopImplicitCluster()
  end <- Sys.time()
  time <- difftime(end,start)
  error <- (EX - analytical) %>% abs
  return(c(n,EX,analytical,error,time,"MCpar3"))
}
data.table(t(MCpar3(n))) %>% setNames(header)
```

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

```
##           n    computational       analytical                error
## 1: 10000000 1.87333968656737 1.87296355073463 0.000376135832744318
##            walltime   type
## 1: 1.82036471366882 MCpar3
```

Benchmarking

Now, let's visualize the walltimes of implemented solutions.

```
iterations <- c(2.5,5,7.5,10)*10**(5:7)
```

```
## Warning in c(2.5, 5, 7.5, 10) * 10^(5:7): longer object length is not a multiple
## of shorter object length
```

```
funs <- c(
  MCser1,MCser2,MCser3,MCser4,
  MCpar1,MCpar3 #MCpar2,MCpar2_2,
)
data <- sapply(funs,function(f,iterations){
  sapply(iterations,function(n){
    f(n)
  })
},iterations) %>%
  matrix(ncol=6,byrow=T) %>%
  data.table %>%
  setNames(header) %>%
  mutate_at(header[-grep(x=header,pattern="type")],as.numeric)

data
```

```
##            n computational analytical         error    walltime   type
##  1:   250000       1.873130   1.872964 0.00016643353  0.03385305 MCser1
##  2:  5000000       1.873030   1.872964 0.00006595650  0.69568539 MCser1
##  3: 75000000       1.872907   1.872964 0.00005654916 11.68683290 MCser1
##  4:  1000000       1.872530   1.872964 0.00043352101  0.14016461 MCser1
##  5:   250000       1.871317   1.872964 0.00164682859  0.20487976 MCser2
##  6:  5000000       1.872773   1.872964 0.00019064751  0.77347493 MCser2
##  7: 75000000       1.872913   1.872964 0.00005082639 11.00078845 MCser2
##  8:  1000000       1.873018   1.872964 0.00005454452  0.25182199 MCser2
##  9:   250000       1.875004   1.872964 0.00204025997  0.03408027 MCser3
## 10:  5000000       1.873260   1.872964 0.00029644140  0.65575528 MCser3
## 11: 75000000       1.873049   1.872964 0.00008524746 11.90426970 MCser3
## 12:  1000000       1.872933   1.872964 0.00003028198  0.13201976 MCser3
## 13:   250000       1.874320   1.872964 0.00135683934  0.23717046 MCser4
## 14:  5000000       1.873407   1.872964 0.00044341028  0.82303739 MCser4
## 15: 75000000       1.873138   1.872964 0.00017413568 11.54206014 MCser4
## 16:  1000000       1.873115   1.872964 0.00015100499  0.34342837 MCser4
## 17:   250000       1.871166   1.872964 0.00179786906  0.44379544 MCpar1
## 18:  5000000       1.873437   1.872964 0.00047358039  0.81169033 MCpar1
## 19: 75000000       1.872815   1.872964 0.00014894196  6.08725572 MCpar1
## 20:  1000000       1.873463   1.872964 0.00049947203  0.59535599 MCpar1
## 21:   250000       1.870885   1.872964 0.00207891484  0.46692729 MCpar3
## 22:  5000000       1.873080   1.872964 0.00011612905  1.09262633 MCpar3
## 23: 75000000       1.873205   1.872964 0.00024189381 10.45459032 MCpar3
## 24:  1000000       1.872843   1.872964 0.00012044145  0.57980013 MCpar3
##            n computational analytical         error    walltime   type
```

```
ggplot(data=data,aes(x=n,y=walltime,group=type,color=type)) +
  geom_point() +
  geom_line() +
  labs(
       title="Serial vs parallel implementations",
       x="Nr of iterations",
       y="Walltime (sec)"
       ) +
  theme(
        plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5)
```

```
    ) +
guides(color=guide_legend(title="Function"))
```