Вычисление интегралов методом Монте-Карло

Николай Карасов

2/17/2019

Многомерный интеграл

- Убедиться, что интеграл сходится.
- Реализовать процедуру многомерного Монте-Карло интегрирования для произвольной линейно-ограниченной области с равномерной интегрирующей плотностью.
- Вычислить интеграл методом Монте-Карло двумя способами: «в лоб» и через замену переменных области интегрирования к параллелепипеду («коробке»), или каким-либо иным «разумным» методом (например, за счет выбора зависимых случайных величин).

Сходимость

Имеется интеграл:

$$J = \frac{4}{\pi} \iiint\limits_{\substack{x_1 > 0 \\ 0 < x_2 < bx_1 \\ x_3 > cx_1}} \exp\left(b^2 x_1^2 - x_2^2 - x_3^2\right) \mathrm{d}x_1 \mathrm{d}x_2 \mathrm{d}x_3$$

- Интеграл сходится при $b < c$.
- Интеграл расходится при $b \geq c$.

Пусть $c = 2$, а $b = 1$.

```
b <- 1
c <- 2
```

Моделирование

Перейдем к "коробке" и зададим соответсвтующую функцию f_box.

```
f_box <- function(x1, x2, x3)
  return (4*b* c / pi * (tan(x1))^2 /
           (cos(x1) * x3)^2 *
           exp((tan(x1))^2 * (b^2 - b^2*(x2)^2 - (c)^2 * ((x3))^(-2))))
```

Зададим плотность p.

```
p <- function(x1)
  return(dunif(x1, 0, pi/2))

generate_unif_new <- function(n) {
  x <- runif(n, 0, pi/2)
  y <- runif(n)
  z <- runif(n)
```

```
  return (f_box(x,y,z) / p(x))
}

J <- mean(generate_unif_new(1000000))

J
```

```
## [1] 0.08095627
```

Метод зависимых случайных велечин

Зададим исходную функцию f.

```
f <- function(x1, x2, x3)
  return(4/pi * exp(b^2 * x1^2 - x2^2 - x3^2))
```

Запишем функцию плотности.

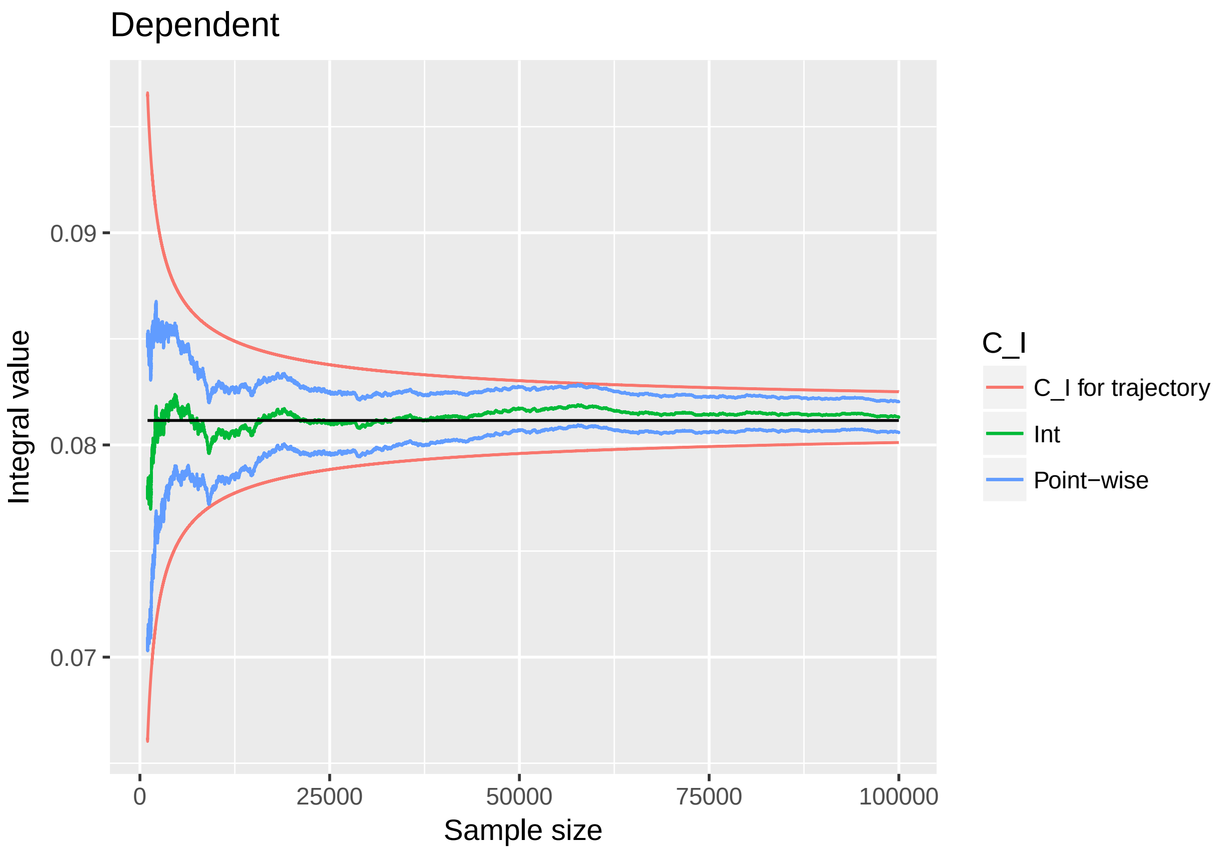
```
dep_new <- function(x1, x2, x3) {
  xi <- dexp(x1, 1)
  eta <- dunif(x2, min = 0, max = b * x1)
  zeta <- dexp(x3, 1) * exp(c * x1)
  return(xi * eta * zeta)
}

simulate_new <- function(n) {
  x1 <- rexp(n, 1)
  xk <- runif(n, min = 0, max = 1)
  x2 <- xk * b * x1
  xs <- rexp(n, rate = 1)
  x3 <- c * x1 + xs

  return(f(x1,x2,x3) / dep_new(x1,x2,x3))
}

J <- mean(simulate_new(1000000))
J
```

```
## [1] 0.08099651
```

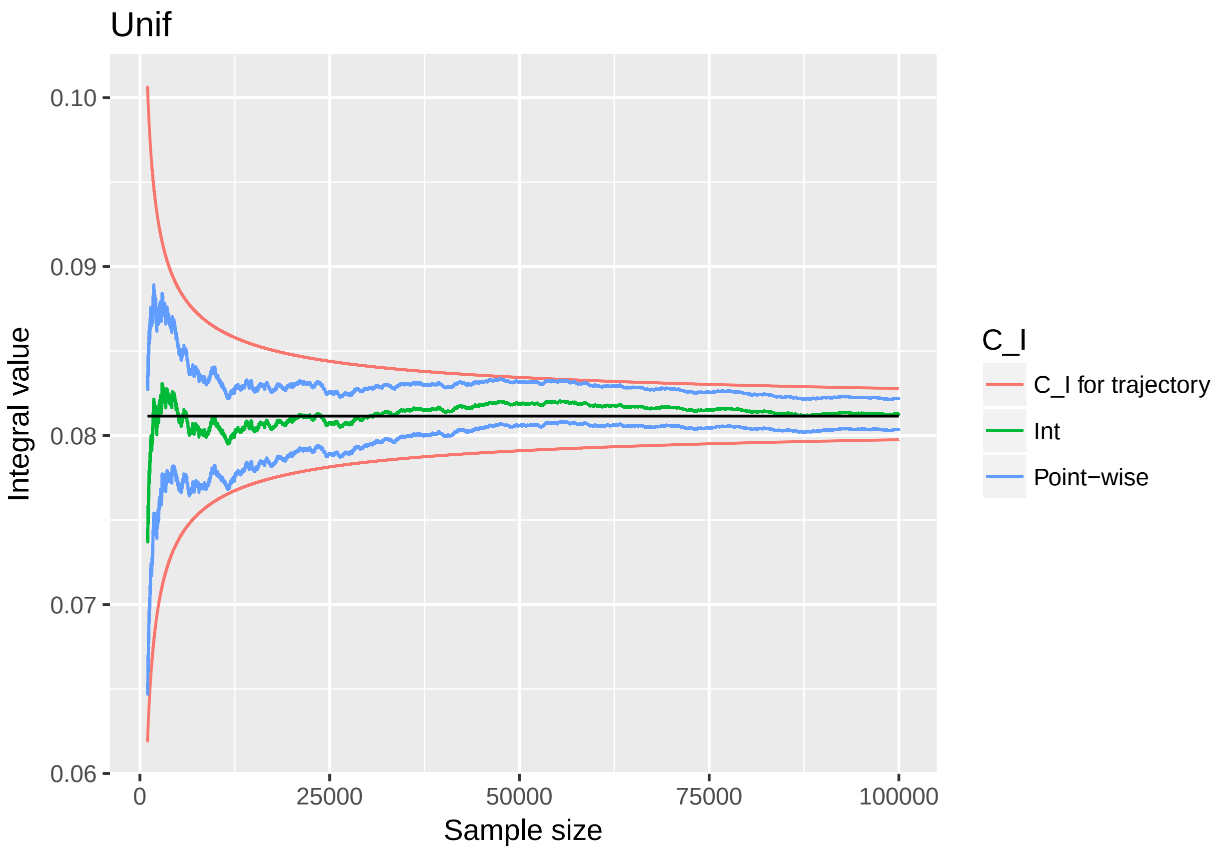
Доверительные интервалы

```
u <- function(t){
  0.1+3.15*sqrt(t)
}

conf_int <- function(f, n){
  sample <- f(n)
  sequence <- seq(n)
  #
  t <- sequence/n
```

```r
  x <- cumsum(sample)/sequence
  sd.mc <- sd(sample)
  s <- sd.mc*sqrt(n)*u(t)/floor(n*t)
  c_i_u <- x[n] + s
  c_i_l <- x[n] - s
  #Поточечный
  alpha <- 0.05
  c_y <- qnorm(1-alpha/2)
  s <- sd.mc * c_y / sqrt(sequence)
  p_c_i_u <- x + s
  p_c_i_l <- x - s
  c_i <- data.frame(N = sequence, C_i_l = c_i_l,
                              C_i_u =  c_i_u,
                              Path = x,
                              P_c_i_l = p_c_i_l,
                              P_c_i_u = p_c_i_u)

  return(c_i)
}

conf.int <- conf_int(simulate_new, 100000)
library("ggplot2")

conf.int <- conf.int[-(1:1000),]

ggplot(conf.int, aes(N, C_i_l, group = 1)) +
  geom_line(aes(color="C_I for trajectory")) +
  geom_line(data = conf.int, aes(N, C_i_u, color="C_I for trajectory"))+
  geom_line(data = conf.int, aes(N, Path, color="Int"))+
  geom_line(data = conf.int, aes(N, P_c_i_l, color="Point-wise"))+
  geom_line(data = conf.int, aes(N, P_c_i_u, color="Point-wise"))+
  geom_line(aes(N, 0.0811536))+
  labs(color="C_I") +
  xlab("Sample size") +
  ylab("Integral value") +
  ggtitle("Dependent")
```




```
conf.int.unif <- conf_int(generate_unif_new, 100000)
library("ggplot2")

conf.int.unif <- conf.int.unif[-(1:1000),]

ggplot(conf.int.unif, aes(N, C_i_l, group = 1)) +
  geom_line(aes(color="C_I for trajectory")) +
  geom_line(data = conf.int.unif, aes(N, C_i_u, color="C_I for trajectory"))+
  geom_line(data = conf.int.unif, aes(N, Path, color="Int"))+
  geom_line(data = conf.int.unif, aes(N, P_c_i_l, color="Point-wise"))+
  geom_line(data = conf.int.unif, aes(N, P_c_i_u, color="Point-wise"))+
    geom_line(aes(N, 0.0811536))+
  labs(color="C_I") +
  xlab("Sample size") +
  ylab("Integral value") +
  ggtitle("Unif")
```



Unif
Integral value
0.10
0.09
0.08
0.07
0.06
0
25000
50000
75000
100000
Sample size
C_I
C_I for trajectory
Int
Point−wise