PSTAT 194CS HW 2
Random Number Generation and Sampling

Ostapenko, Vasiliy (vostapenko, 774 970 8)
Collaborated with: N/A

Contents

Exercise 1

Sampling discrete distributions.

```
geom_pdf = function(x, p) {
  if((x %% 1 == 0) & (x >= 0)) {
    return(p*(1-p)^x)
  }
  return(0)
}
```

```
gen_geom_rv = function(u, p) {
  done = FALSE
  x = 0
  cml_p = 0
  while(!done) {
    cml_p = cml_p + geom_pdf(x=x, p=p)
    if(u <= cml_p) {
      done = TRUE
    } else {
      x = x + 1
    }
  }
  return(x)
}
```

```
N = 1000
p = 0.4
u_vec = runif(n=N, min=0, max=1)
geom_vec = sapply(u_vec, gen_geom_rv, p=p)
```

```
mean(geom_vec)
```

```
## [1] 1.585
```

```
(1-p) / p
```

```
## [1] 1.5
```

Exercise 2

Monte Carlo Integration.

```
N = 10000
a = 0
b = 2
u_vec = runif(n=N, min=a, max=b)
h_vec = cos(u_vec * exp(u_vec))
(b-a) * (1/N) * sum(h_vec)
```

```
## [1] 0.3406
```

```
N = 10000
a = 0
b = 1
u1_vec = runif(n=N, min=a, max=b)
u2_vec = runif(n=N, min=a, max=b)

h_vec = (exp(-(u1_vec+u2_vec)^2))*(u1_vec^2 + u2_vec)
(b-a) * (b-a) * (1/N) * sum(h_vec)
```

```
## [1] 0.2398
```

```
N = 10000
a = 0
b1 = 3
b2 = 4
u1_vec = runif(n=N, min=a, max=b1)
u2_vec = runif(n=N, min=a, max=b2)

h_vec = (exp(-(u1_vec+u2_vec)^2))*(u1_vec^2 + u2_vec)
(b2-a) * (b1-a) * (1/N) * sum(h_vec)
```

```
## [1] 0.3795
```

Exercise 3

Variance reduction in Monte Carlo Integration: Antithetic sampling.

```
log1p_mc = function(k, a, b, antithetic=FALSE) {
  if(antithetic == TRUE) {
    u_vec = 1 - runif(n=k, min=a, max=b)
  }
  else {
    u_vec = runif(n=k, min=a, max=b)
  }
  h_vec = log(u_vec + 1)
  return((b-a) * (1/k) * sum(h_vec))
}
```

```
k = 1000
a = 0
b = 1
i_naught = log1p_mc(k=k, a=a, b=b, antithetic=FALSE)
i_naught
```

```
## [1] 0.3902
```

```
sim1 = function(n, k, a, b) {
  x = vector(mode="numeric", length=n)
  for(i in 1:n) {
    x[i] = log1p_mc(k=k, a=a, b=b, antithetic=FALSE)
  }
  return(x)
}
n = 1000
x = sim1(n=n, k=k, a=a, b=b)
e_naught = sqrt(var(x))
e_naught
```

```
## [1] 0.006283
```

```
k1 = 500
k2 = 500
i_1_1 = log1p_mc(k=k1, a=a, b=b, antithetic=FALSE)
i_1_2 = log1p_mc(k=k2, a=a, b=b, antithetic=TRUE)
i_1 = (1/2)*(i_1_1 + i_1_2)
i_1
```

```
## [1] 0.3878
```

i_1 represents the average of two Monte Carlo integrations of the same exact integral, namely log(x+1) dx from x=0 to x=1. i_1 is very close to i_naught, within about .015 for this run.

```
sim2 = function(n, k, a, b) {
  x = vector(mode="numeric", length=n)
  for(i in 1:n) {
    first = log1p_mc(k=k, a=a, b=b, antithetic=FALSE)
    second = log1p_mc(k=k, a=a, b=b, antithetic=TRUE)
    x[i] = (1/2) * (first+second)
  }
  return(x)
}
x = sim2(n=n, k=k1, a=a, b=b)
e_1 = sqrt(var(x))
e_1
```

```
## [1] 0.006335
```

e_1 is very close to e_naught, within about .00005 for this run.

Exercise 4

Variance reduction in Monte Carlo Integration: Importance Sampling. We integrate (5/2)x^(3/2) to get the CDF of g(x), which is x^(5/2). Now we invert it and get the inverse CDF, which is x^(2/5). Finally, we generate a random uniform(0, 1) variable, plug it into the invere CDF, and arrive at a sample from g(x) between 0 and 1 inclusive.

```
n = 1000
u = runif(n=n, min=0, max=1)
g = u^(2/5)
```

```
f_mc = function(k, a, b) {
  u_vec = runif(n=k, min=a, max=b)
  g_vec = u_vec^(2/5)
  h_vec = (0.4 * log(g_vec+1))/(g_vec^(3/2))
  return((b-a) * (1/k) * sum(h_vec))
}
i_2 = f_mc(k=n, a=0, b=1)
i_2
```

```
## [1] 0.3838
```

i__one represents one attempt of Monte Carlo integration of the integral log(x+1) from x=0 to x=1. In this instance, we chose to sample from g(x) indirectly instead of sampling from the uniform directly and our h(x) changes - it becomes, namely, f(x)/g(x).

```
sim3 = function(n, k, a, b) {
  x = vector(mode="numeric", length=n)
  for(i in 1:n) {
    x[i] = f_mc(k=k, a=a, b=b)
  }
  return(x)
}
n = 1000
x = sim3(n=n, k=k, a=a, b=b)
e_2 = sqrt(var(x))
e_2
```

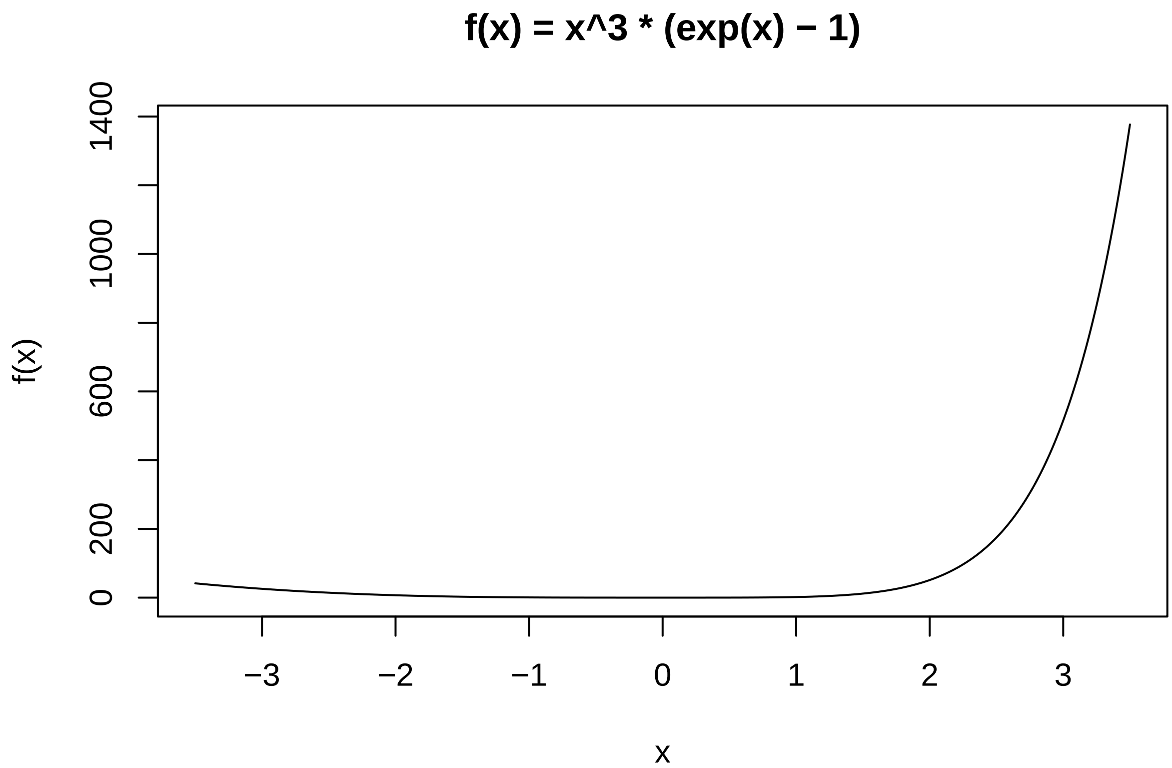
```
## [1] 0.004307
```

The standard error of estimator i__2, namely e__2, is slightly lower than that of either i__naught or i__1. At 0.0045, it is about 0.0015 lower than the other two standard errors.

Exercise 5

```
func = function(x) {
  return(x^3 * (exp(x) - 1))
}
```

```
curve(func, from=-3.5, to=3.5, n=1000, xlab="x", ylab="f(x)",
      main="f(x) = x^3 * (exp(x) - 1)")
```



The function is fairly constant between about -2.5 and 1.5. After that, in the positive direction, it takes on exponential character.

```
func_mc = function(k, a, b) {
  u_vec = runif(n=k, min=a, max=b)
  h_vec = func(x=u_vec)
  return((b-a) * (1/k) * sum(h_vec))
}
k = 10000
s_naught = func_mc(k=k, a=-3, b=3)
s_naught
```

```
## [1] 245
```

```
intervals = list(c(-3, -2), c(-2, -1), c(-1, 0),
                 c(0, 1), c(1, 2), c(2, 3))

s_1 = 0
for(i in 1:length(intervals)) {
  a = intervals[[i]][1]
  b = intervals[[i]][2]
  partial = func_mc(k=k/6, a=a, b=b)
  s_1 = s_1 + partial
}
s_1
```

```
## [1] 245.9
```

```
sim4 = function(n, k, a, b) {
  x = vector(mode="numeric", length=n)
  for(i in 1:n) {
    x[i] = func_mc(k=k, a=a, b=b)
  }
  return(x)
}
n = 1000
x = sim4(n=n, k=10000, a=-3, b=3)
e_snaught = sqrt(var(x))
e_snaught
```

```
## [1] 5.669
```

```
sim5 = function(n, k, a, b) {
  x = vector(mode="numeric", length=n)
  for(i in 1:n) {
    sigma = 0
    for(i in 1:length(intervals)) {
        a = intervals[[i]][1]
        b = intervals[[i]][2]
        partial = func_mc(k=k/6, a=a, b=b)
        sigma = sigma + partial
    }
    x[i] = sigma
  }
  return(x)
}
n = 1000
x = sim5(n=n, k=10000, a=-3, b=3)
e_s1 = sqrt(var(x))
e_s1
```

```
## [1] 7.676
```

The first estimator is more efficient. The most important area to sample from to estimate the integral is approximately between 2.5 and 3.

Exercise 6 (5.2)

```
x = seq(0.1, 2.5, length=10)
stdnorm_mc = function(k) {
  h_vec = vector(mode="numeric", length=length(x))
  for(i in 1:length(x)) {
    u_vec = runif(k, 0, x[i])
    g_vec = x[i] * exp(-(u_vec^2)/2)
    h_vec[i] = mean(g_vec)/sqrt(2*pi)+(1/2)
  }
  return(h_vec)
}
k = 10000
stdnorm_cdf = stdnorm_mc(k=k)
rbind(x, stdnorm_cdf, pnorm(x))
```

```
##               [,1]   [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
```

```
## x           0.1000 0.3667 0.6333 0.9000 1.1667 1.4333 1.7000 1.9667 2.2333
## stdnorm_cdf 0.5398 0.6431 0.7367 0.8160 0.8788 0.9252 0.9557 0.9745 0.9923
##             0.5398 0.6431 0.7367 0.8159 0.8783 0.9241 0.9554 0.9754 0.9872
##              [,10]
## x           2.5000
## stdnorm_cdf 0.9955
##             0.9938
```

```r
x = seq(2, 2, length=1)
sim6 = function(n, k) {
  y = vector(mode="numeric", length=n)
  for(i in 1:n) {
    y[i] = stdnorm_mc(k=k)
  }
  return(y)
}
y = sim6(n=1000, k=k)
phi_2 = mean(y)
phi_2
```

```
## [1] 0.9773
```

```r
phi_2_ci = phi_2 + qnorm(c(0.025, 0.975)) * sqrt(var(y))
phi_2_ci
```

```
## [1] 0.9728 0.9818
```

Exercise 7 (5.4)

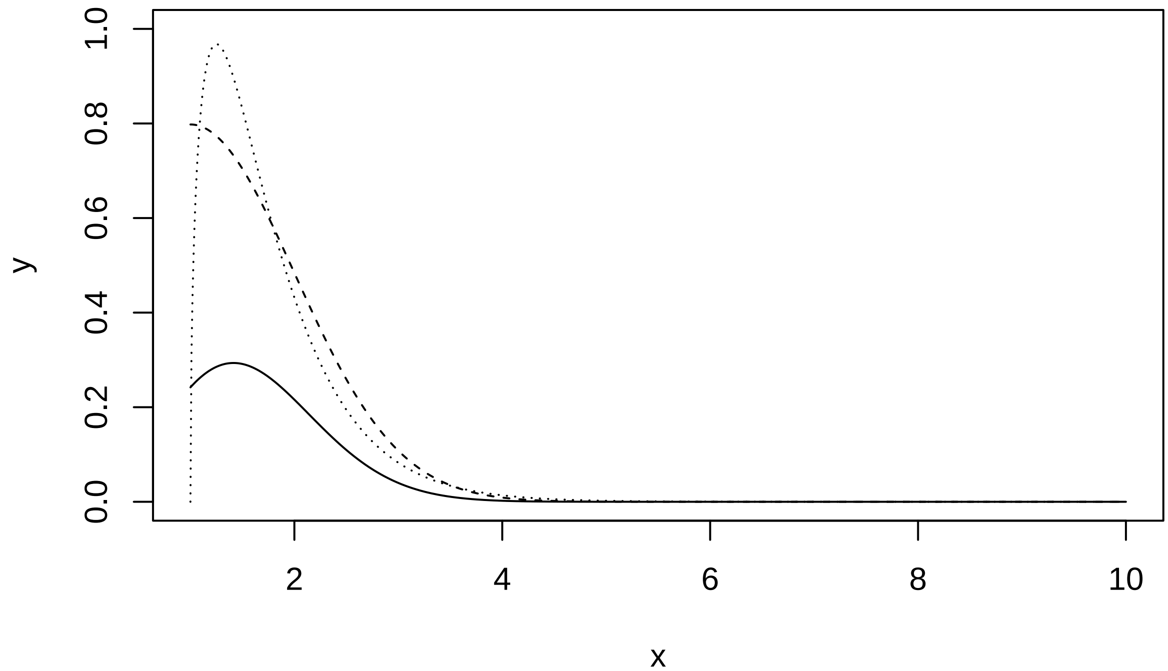
```r
x = seq(0.1, 0.9, 0.1)
beta33_mc = function(w, k, a, b) {
  y = w[1]
  alpha = a[1]
  beta = b[1]
  u = rgamma(k, alpha, 1)
  v = rgamma(k, beta, 1)
  z = u/(u+v)
  return(mean(z<=y))
}
m = length(x)
p = vector(mode="numeric", length=m)
for(i in 1:m) {
  p[i] = beta33_mc(w=x[i], k=10000, a=3, b=3)
}
rbind(x, pbeta(x, 3, 3), p)
```

```
##      [,1]    [,2]   [,3]   [,4]   [,5]   [,6]   [,7]   [,8]   [,9]
## x 0.10000 0.20000 0.3000 0.4000 0.5000 0.6000 0.7000 0.8000 0.9000
##   0.00856 0.05792 0.1631 0.3174 0.5000 0.6826 0.8369 0.9421 0.9914
## p 0.00780 0.06000 0.1661 0.3289 0.5075 0.6826 0.8404 0.9394 0.9925
```

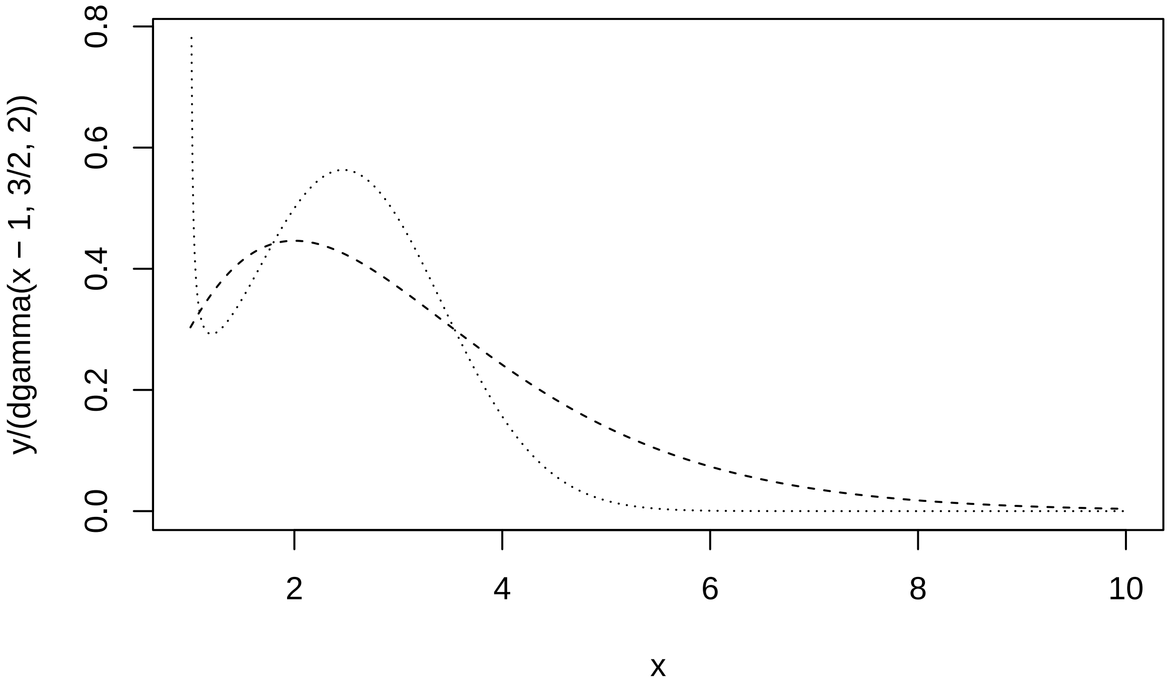
Exercise 8 (5.13)

Considering the shape of g(x), two importance functions which come to mind are the normal distribution pdf as well as the gamma distribution pdf.

```r
x = seq(1, 10, 0.01)
y = x^2 * exp(-x^2 / 2)/sqrt(2 * pi)
plot(x=x, y=y, type="l", ylim=c(0, 1))
lines(x, 2*dnorm(x, 1), lty=2)
lines(x, dgamma(x-1, 3/2, 2), lty=3)
```




```r
{
  plot(x, y/(dgamma(x-1, 3/2, 2)), type="l", lty=3)
  lines(x, y/(2*dnorm(x, 1)), lty=2)
}
```



The normal pdf importance function would probably produce a lower variance estimator as the plot demonstrates.

Exercise 9 (5.14)

```
xnorm_sim1 = function(n, k) {
  y = vector(mode="numeric", length=k)
  for(i in 1:length(y)) {
    x_vec = sqrt(rchisq(n, 1)) + 1
    f_vec = 2*dnorm(x_vec, 1)
    h_vec = x_vec^2 * exp(-x_vec^2 / 2)/sqrt(2*pi)
    y[i] = mean(h_vec/f_vec)
  }
  return(y)
}
xnorm_sim2 = function(n, k) {
  y = vector(mode="numeric", length=k)
  for(i in 1:length(y)) {
    x_vec = rgamma(n, 3/2, 2) + 1
    f_vec = dgamma(x_vec-1, 3/2, 2)
    h_vec = x_vec^2 * exp(-x_vec^2 / 2)/sqrt(2*pi)
    y[i] = mean(h_vec/f_vec)
  }
  return(y)
}
x = xnorm_sim1(n=10000, k=1000)
```

```r
y = xnorm_sim2(n=10000, k=1000)
```

```r
c(mean(x), mean(y))
```

```
## [1] 0.4006 0.4006
```

```r
c(var(x), var(y))
```

```
## [1] 1.918e-07 1.210e-06
```

The first choice for our importance function produces a more efficient estimator.