

Assignment Sheet: Transformation Methods, Rejection Sampling, and Importance Sampling — Model Answers

The Python code in these model answers requires the following libraries:

```

1 from scipy import *
2 from scipy import optimize
3 from scipy import random
4 from scipy import stats
5 import pylab
    
```

1. (a) Steps 1 to 3 are a rejection sampling algorithm sampling from a uniform distribution the the “inside” of the unit circle, i.e. $\{(u_1, u_2) \in \mathbb{R}^2 : u_1^2 + u_2^2 < 1\}$. Thus after step 3, the density of (U_1, U_2) is

$$f_{U_1, U_2}(u_1, u_2) = \frac{1}{\pi} \cdot \mathbf{1}_{u_1^2 + u_2^2 < 1}.$$

Let (ϑ, R) be the polar coordinates of (U_1, U_2) , then

$$U_1 = \cos(\vartheta) \cdot \sqrt{R^2}, \quad U_2 = \sin(\vartheta) \cdot \sqrt{R^2}.$$

We will now show that $\vartheta \sim U(0, 2\pi)$ and $R^2 \sim U(0, 1)$: For $\vartheta \in (0, 2\pi)$ and $r^2 \in (0, 1)$ we have that

$$\begin{aligned}
 f_{\vartheta, R^2}(\vartheta, r^2) &= \underbrace{f_{U_1, U_2}(\cos(\vartheta) \cdot \sqrt{r^2}, \sin(\vartheta) \cdot \sqrt{r^2})}_{=\frac{1}{\pi}} \cdot \begin{vmatrix} \frac{\partial u_1}{\partial \vartheta} & \frac{\partial u_1}{\partial r^2} \\ \frac{\partial u_2}{\partial \vartheta} & \frac{\partial u_2}{\partial r^2} \end{vmatrix} \\
 &= \begin{vmatrix} -r \sin(\vartheta) & \frac{\cos(\vartheta)}{2r} \\ r \cos(\vartheta) & \frac{\sin(\vartheta)}{2r} \end{vmatrix} = \frac{1}{2} \\
 &= \frac{1}{2\pi},
 \end{aligned}$$

thus $\vartheta \sim U(0, 2\pi)$ and $R^2 \sim U(0, 1)$, and ϑ and R^2 are independent.

As we can rewrite

$$\begin{aligned}
 X_1 &= U_1 \sqrt{\frac{-2 \log(R^2)}{R^2}} = \underbrace{\frac{U_1}{R}}_{=\frac{R \cos(\vartheta)}{R} = \cos(\vartheta)} \cdot \sqrt{-2 \log(R^2)} = \cos(\vartheta) \cdot \sqrt{-2 \log(R^2)} \\
 X_2 &= U_2 \sqrt{\frac{-2 \log(R^2)}{R^2}} = \underbrace{\frac{U_2}{R}}_{=\frac{R \sin(\vartheta)}{R} = \sin(\vartheta)} \cdot \sqrt{-2 \log(R^2)} = \sin(\vartheta) \cdot \sqrt{-2 \log(R^2)},
 \end{aligned}$$

the Polar Marsaglia method is equivalent to the Box-Muller method and thus also generates two independent samples from the $N(0, 1)$ distribution.

```

(b) # Question 1b
7
8 # Function draws a pair of N(0,1) random numbers using the Polar Marsaglia method
9 def polarMarsaglia():
10     while True:                                     # Keep sampling until inside unit circle
11         u = -1 + 2*random.random_sample(2)         # Draw U_1, U_2 ~ U[-1,1]
12         r2 = sum(u**2)
13         if r2 <= 1:                                  # If radius is less than one, we are done
14             break
15         factor = sqrt(-2*log(r2)/r2)
16         return u * factor
17
18 x = array([polarMarsaglia() for i in range(50)]).ravel()
    
```

In R...

```

1 # Question 1b
2
3 # Function draws a pair of N(0,1) random numbers using the Polar Marsaglia method
4 polarMarsaglia <- function() {
5     repeat {                                     # Keep sampling until inside unit circle
6         u <- -1 + 2*runif(2)                       # Draw U_1, U_2 ~ U[-1,1]
7         r2 <- sum(u**2)
8         if (r2 <= 1)                               # If radius is less than one, we are done
9             break
10    }
11    factor <- sqrt(-2*log(r2)/r2)
12    u * factor
13 }
14
15 x <- numeric(100)                                # Create vector to store result
16 for (i in 1:50)
17     x[2*i+c(-1,0)] <- polarMarsaglia()           # Fill with pairs of values

```

2. (a) Using the fact that U_1 and U_2 have the same distribution and are independent we obtain

$$\begin{aligned}
2 \cdot \text{Cov}(F^-(U_1), F^-(1-U_1)) &= 2 \cdot \mathbb{E}(F^-(U_1)F^-(1-U_1)) - 2 \cdot \mathbb{E}(F^-(U_1)) \mathbb{E}(F^-(1-U_1)) \\
&\stackrel{U_1 \stackrel{iid}{\sim} U_2}{=} 2 \cdot \mathbb{E}(F^-(U_1)F^-(1-U_1)) - 2 \cdot \mathbb{E}(F^-(U_1)) \mathbb{E}(F^-(1-U_2)) \\
&\stackrel{U_1 \perp U_2}{=} 2 \cdot \mathbb{E}(F^-(U_1)F^-(1-U_1)) - 2 \cdot \mathbb{E}(F^-(U_1)) \mathbb{E}(F^-(1-U_2)) \\
&\stackrel{U_1 \stackrel{iid}{\sim} U_2}{=} \mathbb{E}(F^-(U_1)F^-(1-U_1)) - \mathbb{E}(F^-(U_1)) \mathbb{E}(F^-(1-U_2)) \\
&\quad - \mathbb{E}(F^-(U_2)F^-(1-U_1)) + \mathbb{E}(F^-(U_2)F^-(1-U_2)) \\
&= \mathbb{E}((F^-(U_1) - F^-(U_2))(F^-(1-U_1) - F^-(1-U_2)))
\end{aligned}$$

(b) Let without loss of generality $u_1 \leq u_2$ (otherwise relabel u_1 and u_2), thus $1-u_2 \leq 1-u_1$. As $F^-(\cdot)$ is non-decreasing we have that $F^-(u_1) \leq F^-(u_2)$ and $F^-(1-u_2) \leq F^-(1-u_1)$. Thus

$$\underbrace{(F^-(u_1) - F^-(u_2))}_{\leq 0} \underbrace{(F^-(1-u_1) - F^-(1-u_2))}_{\geq 0} \leq 0.$$

(c) $2 \cdot \text{Cov}(F^-(u_1), F^-(1-u_1)) \stackrel{(a)}{=} \mathbb{E}((F^-(U_1) - F^-(U_2))(F^-(1-U_1) - F^-(1-U_2))) \stackrel{(b)}{\leq} 0$

(d) We have that

$$\begin{aligned}
\text{Var}\left(\frac{F^-(U_1) + F^-(1-U_1)}{2}\right) &= \frac{\text{Var}(F^-(U_1)) + 2 \cdot \text{Cov}(F^-(U_1), F^-(1-U_1)) + \text{Var}(F^-(1-U_1))}{4} \\
&= \frac{\text{Var}(F^-(U_1))}{2} + \underbrace{\frac{\text{Cov}(F^-(U_1), F^-(1-U_1))}{2}}_{\leq 0} \\
&\leq \frac{\text{Var}(F^-(U_1))}{2} = \text{Var}\left(\frac{F^-(U_1) + F^-(U_2)}{2}\right) = \frac{\text{Var}(F^-(U_1))}{2}
\end{aligned}$$

The variance of a Monte Carlo estimate can be reduced by using negatively correlated samples. The method described above (often referred to as “antithetic variables”) is one way of obtaining a negatively correlated sample.

3. (a) We have that In order to be able to carry out rejection sampling we need that there is a finite $M > 0$ such that $\phi_{(0,1)}(x) < M\phi_{(1,\sigma^2)}(x)$. This is the case iff $\phi_{(0,1)}(x)/\phi_{(1,\sigma^2)}(x)$ is bounded.

$$\frac{\phi_{(0,1)}(x)}{\phi_{(1,\sigma^2)}(x)} = \frac{\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-1)^2}{2\sigma^2}\right)} = \sqrt{\sigma^2} \exp\left(-\frac{x^2}{2} + \frac{(x-1)^2}{2\sigma^2}\right) = \sqrt{\sigma^2} \exp\left(-\frac{1}{2\sigma^2}((\sigma^2-1)x^2 + 2x - 1)\right)$$

It is easy to see that the right-hand side is bounded above iff $\sigma^2 > 1$ (the “overall sign” of the x^2 has to be negative, otherwise the exponential is unbounded).

Note that if $\sigma^2 < 1$, then the instrumental distribution $\phi_{(1,\sigma^2)}(\cdot)$ has thinner tails than the target distribution $\phi_{(0,1)}(\cdot)$.

(b) To find M we need to find the maximum (over x) of the ratio computed in part (a). This is equivalent to finding the minimum of

$$(\sigma^2 - 1)x^2 + 2x - 1 = \left(x + \frac{1}{\sigma^2 - 1}\right)^2 - \frac{1}{\sigma^2 - 1} - \frac{1}{(\sigma^2 - 1)^2},$$

which is attained at $x = -1/(\sigma^2 - 1)$. Plugging this into the above ratio yields $\sqrt{\sigma^2} \exp\left(\frac{1}{2(\sigma^2-1)}\right)$. Thus

$$M \geq \sqrt{\sigma^2} \exp\left(\frac{1}{2(\sigma^2-1)}\right) = M_{opt}$$

(c) The probability of accepting a value in rejection sampling is $1/M$, thus we need to minimise M . Differentiating the logarithm of M_{opt} yields

$$\frac{\partial \log M_{opt}}{\partial \sigma^2} = \frac{1}{2\sigma^2} - \frac{1}{2(\sigma^2-1)^2} = \frac{\sigma^4 - 3\sigma^2 + 1}{2\sigma^2(\sigma^2-1)^2} = 0,$$

which has as solution $\sigma^2 = \frac{\sqrt{5}+3}{2}$ (and a negative solution, which is not relevant).

4. (a) We have that

$$\begin{aligned} \mathbb{E}_g(W(X)^2) &= \int \left(\frac{\phi_{(0,1)}(x)}{\phi_{(1,\sigma^2)}(x)} \right)^2 \phi_{(1,\sigma^2)}(x) dx = \int \frac{\phi_{(0,1)}(x)^2}{\phi_{(1,\sigma^2)}(x)} dx \\ &= \int \frac{\frac{1}{2\pi} \exp(-x^2)}{\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-1)^2}{2\sigma^2}\right)} dx \\ &= \sqrt{\frac{\sigma^2}{2\pi}} \int \exp\left(-x^2 + \frac{1}{2\sigma^2}(x-1)^2\right) dx \\ &= \sqrt{\frac{\sigma^2}{2\pi}} \int \exp\left(-\frac{1}{2\sigma^2}((2\sigma^2-1)x^2 + 2x - 1)\right) dx \tag{1} \\ &= \sqrt{\frac{\sigma^2}{2\pi}} \int \exp\left(-\frac{1}{2\sigma^2}((2\sigma^2-1)(x + 1/(2\sigma^2-1))^2 - 1 - 1/(2\sigma^2-1))\right) dx \\ &= \sqrt{\frac{\sigma^2}{2\pi}} \sqrt{\frac{2\pi\sigma^2}{2\sigma^2-1}} \underbrace{\int \sqrt{\frac{2\sigma^2-1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2/(2\sigma^2-1)}(x + 1/(2\sigma^2-1))^2\right) dx}_{=1} \\ &\quad \cdot \exp\left(\frac{1}{2\sigma^2} + \frac{1}{2\sigma^2(2\sigma^2-1)}\right) \\ &= \frac{\sigma^2}{\sqrt{2\sigma^2-1}} \exp\left(\frac{1}{2\sigma^2} + \frac{1}{2\sigma^2(2\sigma^2-1)}\right) \\ &= \frac{\sigma^2}{\sqrt{2\sigma^2-1}} \exp\left(\frac{1}{2\sigma^2-1}\right) \end{aligned}$$

Using $\text{Var}_g(W(X)) = \mathbb{E}_g(W(X)^2) - (\mathbb{E}_g(W(X)))^2$ and $\mathbb{E}_g(W(X)) = 1$ we obtain that

$$\text{Var}_g(W(X)) = \frac{\sigma^2}{\sqrt{2\sigma^2-1}} \exp\left(\frac{1}{2\sigma^2-1}\right) - 1,$$

(b) In (1) the right-hand side is finite iff $\sigma^2 > 1/2$ (the ‘‘overall sign’’ of the x^2 has to be negative, otherwise the integral is $+\infty$).

(c) I don’t know how to find the minimum of the variance in closed form, so I had to resort to minimising the variance numerically:

```

19 # Question 4b
20
21 def impsamp_var(sigma2):
22     return sigma2 / sqrt(2*sigma2-1) * exp(1/(2*sigma2-1)) - 1
23
24 optimize.fmin(impsamp_var, 2)

```

In R ...

```

18 impsamp.var <- function(sigma2) {
19     sigma2 / sqrt(2*sigma2-1) * exp(1/(2*sigma2-1)) - 1
20 }
21
22 optimize(impsamp.var, c(0.5,5))

```

With both programs I obtain $\sigma^2 \approx 2.2808$ as optimal solution.

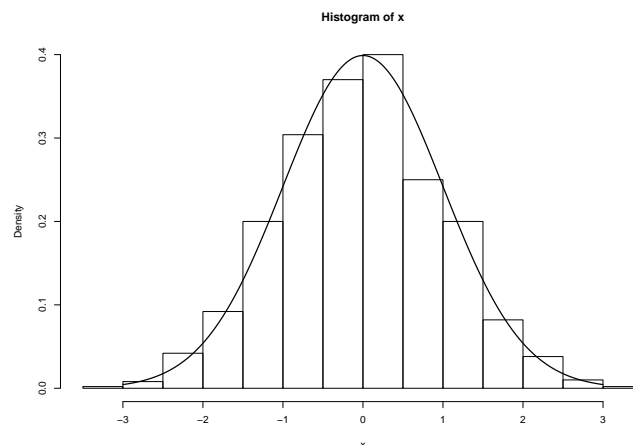
5. (a) # Question 5a

```
27
28 def rejection_example(n, sigma2):
29     x = zeros(n) # Create vector to store result
30     M = sqrt(sigma2) * exp(1/(2*(sigma2-1))) # Optimal M
31
32     for i in xrange(n):
33         while True: # Keep proposing ...
34             x_new = stats.norm(1,sqrt(sigma2)).rvs(1)
35                 # Generate x_new ~ N(1,sigma2)
36             accept = stats.norm(0,1).pdf(x_new) / (M*stats.norm(1,sqrt(sigma2)).pdf(x_new))
37                 # Compute probability of acceptance
38             if (random.random_sample()<accept): # ... until a proposed value is accepted
39                 break
40             x[i] = x_new
41     return i
42
43 n = 1000
44 x = rejection_example(n, (sqrt(5)+3)/2) # Generate sample
45 pylab.hist(x, normed=True) # Draw histogram
46 t = linspace(-3, 3, 1000)
47 pylab.plot(t, stats.norm(0,1).pdf(t)) # Add density of target dist'n
```

In R...

```
23 # Question 5a
24
25 rejection.example <- function(n, sigma2) {
26     x <- numeric(n) # Create vector to store result
27     sigma2 <- (sqrt(5)+3)/2 # Optimal variance
28     M <- sqrt(sigma2) * exp(1/(2*(sigma2-1))) # Optimal M
29     for (i in 1:n) {
30         repeat { # Keep proposing ...
31             x.new <- rnorm(1,1,sqrt(sigma2)) # Generate x.new ~ N(1,sigma2)
32             accept <- dnorm(x.new,0,1) / (M*dnorm(x.new,1,sqrt(sigma2)))
33                 # Compute probability of acceptance
34             if (runif(1)<accept) # ... until a proposed value is accepted
35                 break
36         }
37         x[i] <- x.new
38     }
39 }
40
41 n <- 1000
42 x <- rejection.example(n, (sqrt(5)+3)/2) # Generate sample
43 hist(x, freq=FALSE) # Draw histogram
44 t <- seq(-3, 3, length.out=1000)
45 lines(t, dnorm(t,0,1), lwd=2) # Add density of target dist'n
```

I obtained the histogram shown below:



(b) # Question 5b

```
49 def importance_sample(n, sigma2):
```

```

50 x = stats.norm(1,sqrt(sigma2)).rvs(n) # Draw from instrumental dist'n
51 w = stats.norm(0,1).pdf(x) / stats.norm(1,sqrt(sigma2)).pdf(x)
52 # Compute weights
53 return x, w
54
55 n = 1000
56 [x, w] = importance_sample(n, 2.2808) # Generate weighted sample
57 print(sum(x*w)/sum(w)) # Self-normalised estimate of E(X)
58 print(sum(x**2*w)/sum(w)) # Self-normalised estimate of E(X^2)

```

In R ...

```

46 # Question 5b
47 importance.sample <- function(n, sigma2) {
48   x <- rnorm(n,1,sqrt(sigma2)) # Draw from instrumental dist'n
49   w <- dnorm(x,0,1) / dnorm(x,1,sqrt(sigma2)) # Compute weights
50   list(x=x, w=w)
51 }
52
53 n <- 1000
54 s = importance.sample(n, 2.2808) # Generate weighted sample
55 sum(s$x*s$w)/sum(s$w) # Self-normalised estimate of E(X)
56 sum(s$x**2*s$w)/sum(s$w) # Self-normalised estimate of E(X^2)

```

6. (a) From the lectures we have that $\mathbb{E}_g(w(X)) = \int \frac{f(x)}{g(x)} g(x) dx = 1$. Further

$$\mathbb{E}_g(w(X)^2) = \int \frac{f(x)^2}{g(x)^2} g(x) dx = \int f(x) \underbrace{\frac{f(x)}{g(x)}}_{<M} dx < M \underbrace{\int f(x) dx}_{=1} = M.$$

Thus

$$\text{Var}_g(w(X)) = \mathbb{E}_g(w(X)^2) - (\mathbb{E}_g(w(X)))^2 < M - 1.$$

(b) The importance sampling estimate has finite variance iff $\mathbb{E}_g(h(X)^2 w(X)^2) < +\infty$. We have that

$$\mathbb{E}_g(w(X)^2 h(X)^2) = \int \frac{f(x)^2}{g(x)^2} h(x)^2 g(x) dx = \int \underbrace{\frac{f(x)}{g(x)}}_{<M} h(x)^2 f(x) dx < M \cdot \mathbb{E}_f(h(X)^2) < +\infty,$$

as $\text{Var}_f(h(X)) < +\infty$.

7. (a) We have that $M = 1/(1 - \Phi_{(\mu, \sigma^2)}(x))$ and thus the probability of acceptance is 0 if $X \leq \tau$, and 1 otherwise ($X \sim N(\mu, \sigma^2)$). Thus rejection sampling simplifies to:

1. Draw $X \sim N(\mu, \sigma^2)$.
2. If $X > \tau$ accept X as a sample from the left-truncated normal distribution, otherwise reject it and go back to 1.

```

59 # Question 7a
60
61 def sample_truncated_normal_a(n, mu, sigma2, tau):
62   x = zeros(n) # Create vector to store result
63   num_proposed = 0 # Set counter of proposed values
64   for i in xrange(n):
65     while True: # Keep proposing ...
66       x_new = stats.norm(mu,sqrt(sigma2)).rvs(1)
67       # Generate x.new ~ N(mu,sigma2)
68       num_proposed = num_proposed + 1 # Increment counter
69       if x_new > tau: # ... until x_new > tau (accepted)
70         break
71     x[i] = x_new
72   print "Proportion of accepted values" # Print proportion of accepted values
73   print float(n)/float(num_proposed)
74   return x

```

In R ...

```

57 # Question 7a
58
59 sample.truncated.normal.a <- function(n, mu, sigma2, tau) {
60   x <- numeric(n) # Create vector to store result
61   num.proposed <- 0 # Set counter of proposed values
62   for (i in 1:n) {
63     repeat { # Keep proposing ...
64       x.new <- rnorm(1,mu,sqrt(sigma2)) # Generate x.new ~ N(mu,sigma2)
65       num.proposed <- num.proposed + 1 # Increment counter
66       if (x.new>tau) # ... until x_new > tau (accepted)
67         break
68     }
69     x[i] <- x.new
70   }
71   cat("Proportion_of_accepted_values\n") # Print proportion of accepted values
72   print(n/num.proposed)
73   x
74 }

```

(b) # Question 7b

```

76
77 x = sample_truncated_normal_a(10, 0, 1, 4)

```

In R...

```

75 # Question 7b
76
77 x <- sample.truncated.normal.a(10, 0, 1, 4)

```

We should require something of the order of 300,000 attempts in order to get 10 accepted values. The expected number of attempts necessary is $10 \cdot M = 10/(1 - \Phi(4)) \approx 315744$. This leads to a proportion of rejected values of around $1 - 1/300000 \approx 0.99997$.

When I ran the above code, only 0.00227% of the values proposed were accepted.

(c) One approach is to generate $Z \sim N(0, 1)$ and to set $X := 4 + |Z|$. The corresponding proposal density is

$$g(x) = \begin{cases} 0 & \text{for } x \leq 4 \\ \sqrt{\frac{2}{\pi}} \exp\left(-\frac{(x-4)^2}{2}\right) & \text{for } x > 4 \end{cases}$$

The ratio of the densities is for $x > 4$

$$\frac{f(x)}{g(x)} = \frac{1}{2(1 - \Phi(4))} \exp(-x^2/2 + (x-4)^2/2) = \frac{1}{2(1 - \Phi(4))} \exp(-4x + 8),$$

which is decreasing in x , so we have that $f(x) < Mg(x)$ with

$$M = \frac{f(4)}{g(4)} = \frac{1}{2(1 - \Phi(4))} \exp(-4^2/2) = \frac{1}{2(1 - \Phi(4))} \exp(-8) \approx 5.30.$$

```

78 # Question 7c
79
80 def sample_truncated_normal_c(n, tau):
81   M = 1 / ( 2 * (1-stats.norm(0,1).cdf(tau)) ) * exp(-tau**2/2)
82   # Compute M
83   x = zeros(n) # Create vector to store result
84   num_proposed = 0 # Set counter of proposed values
85   for i in xrange(n):
86     while True: # Keep proposing ...
87       x_new = tau + abs(stats.norm(0,1).rvs(1))
88       # Draw new value from instrumental dist'n
89       num_proposed = num_proposed + 1 # Increment counter
90       f = stats.norm(0,1).pdf(x_new) / (1-stats.norm(0,1).cdf(tau))
91       # Evaluate target density
92       g = 2 * stats.norm(tau,1).pdf(x_new) # Evaluate instrumental density
93       accept = f / ( M * g ) # Probability of accepting
94       if random.random_sample(1)<accept: # ... until one value is accepted
95         break
96       x[i] = x_new

```

```

97     print("Proportion of accepted values\n")           # Print proportion of accepted values
98     print(float(n)/float(num_proposed))
99     return x
100
101 x = sample_truncated_normal_c(10, 4)

```

In R...

```

78 # Question 7c
79
80 sample.truncated.normal.c <- function(n, tau) {
81     M <- 1 / ( 2 * (1-pnorm(tau) )) * exp(-tau^2/2)
82                                     # Compute M
83     x <- numeric(n)                  # Create vector to store result
84     num.proposed <- 0                # Set counter of proposed values
85     for (i in 1:n) {
86         repeat {                    # Keep proposing ...
87             x.new <- tau + abs(rnorm(1)) # Draw new value from instrumental dist'n
88             num.proposed <- num.proposed + 1 # Increment counter
89             f <- dnorm(x.new) / (1-pnorm(tau)) # Evaluate target density}
90             g <- 2 * dnorm(x.new,mean=tau) # Evaluate instrumental density
91             accept <- f / ( M * g ) # Probability of accepting
92             if (runif(1)<accept) # ... until one value is accepted
93                 break
94         }
95         x[i] <- x.new
96     }
97     cat("Proportion of accepted values\n")           # Print proportion of accepted values
98     print(n/num.proposed)
99     x
100 }
101
102 x <- sample.truncated.normal.c(10, 4)

```

When I ran the above code, I required 46 attempts, giving a proportion of rejected values of about 79%. The expected number of attempts required is $10 \cdot M \approx 53$.

```

82 # Question 8a and b
103
104 def is_beta(n, alpha, beta):
105     x = random.random_sample(n) # Draw from instrumental dist'n
106     w = stats.beta(alpha,beta).pdf(x) # Compute weights
107     return x, w
108
109 n = 10
110 N = 100000
111 result_selfnorm = ones(N)
112 result_simple = ones(N)
113
114 for i in xrange(N):
115     [x, w] = is_beta(n, 2, 3) # Draw weighted sample
116     result_selfnorm[i] = sum(x*(1-x)*w)/sum(w) # Compute self-normalised estimate
117     result_simple[i] = sum(x*(1-x)*w)/n # Compute simple estimate
118
119 bias_selfnorm = mean(result_selfnorm) - 0.2 # Compute the bias
120 bias_simple = mean(result_simple) - 0.2
121
122 var_selfnorm = var(result_selfnorm) # Compute the variance
123 var_simple = var(result_simple)
124
125 mse_selfnorm = bias_selfnorm**2 + var_selfnorm # Compute m.s.e.
126 mse_simple = bias_simple**2 + var_simple
127
128 pylab.figure()
129 pylab.boxplot([result_simple, result_selfnorm])
130 pylab.show

```

In R...

```

103 # Question 8a and b

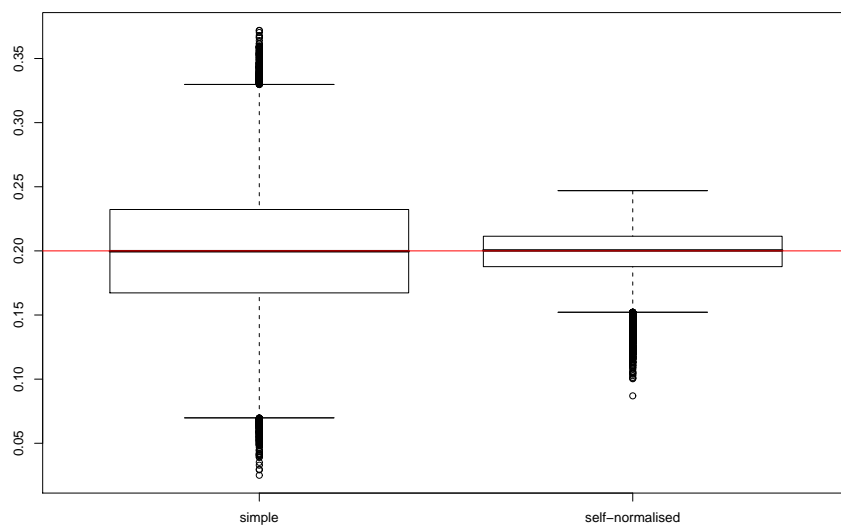
```

```

104
105 is.beta <- function (n, alpha, beta) {
106   x <- runif(n) # Draw from instrumental dist'n
107   w <- dbeta(x,alpha,beta) # Compute weights
108   list(x=x, w=w)
109 }
110
111 n <- 10
112 N <- 100000
113 result.selfnorm = numeric(N)
114 result.simple = numeric(N)
115
116 for (i in 1:N) {
117   s <- is.beta(n, 2, 3) # Draw weighted sample
118   result.selfnorm[i] <- sum(s$x*(1-s$x)*s$w)/sum(s$w) # Compute self-normalised estimate
119   result.simple[i] <- sum(s$x*(1-s$x)*s$w)/n # Compute simple estimate
120 }
121
122 bias.selfnorm <- mean(result.selfnorm) - 0.2 # Compute the bias
123 bias.simple <- mean(result.simple) - 0.2
124
125 var.selfnorm <- var(result.selfnorm) # Compute the variance
126 var.simple <- var(result.simple)
127
128 mse.selfnorm <- bias.selfnorm^2 + var.selfnorm # Compute m.s.e.
129 mse.simple <- bias.simple^2 + var.simple
130
131 boxplot(as.data.frame(cbind(result.simple, result.selfnorm)),
132         names=c("simple", "self-normalised"))

```

The results I obtained were:



	Estimated bias	Estimated variance	Estimated MSE
'Simple' estimate $\tilde{\mu}$	-4.67×10^{-5}	2.26×10^{-3}	2.26×10^{-3}
Self-normalised estimate $\hat{\mu}$	-1.33×10^{-3}	3.22×10^{-4}	3.24×10^{-4}

The self-normalised estimator has a significantly smaller mean-square error, so we would prefer it in this case.