Numerical Integration and Random Number Generation

Last class we looked at the tobit model with likelihood

\[ L(\theta|X,Y) = \prod_{i=1}^{n} \Phi \left( \frac{-X_i\beta}{\sigma} \right)^{1(Y_i=0)} \left[ \phi \left( \frac{Y_i - X_i\beta}{\sigma} \right) \frac{1}{\sigma} \right]^{1(Y_i>0)} \]

and loglikelihood

\[ \mathcal{L}(\theta|X,Y) = \sum_{i:Y_i=0} \ln \left[ \Phi \left( \frac{-X_i\beta}{\sigma} \right) \right] + \sum_{i:Y_i>0} \ln \left[ \phi \left( \frac{Y_i - X_i\beta}{\sigma} \right) \frac{1}{\sigma} \right]. \]

In order for us to be able to evaluate this likelihood, we need to evaluate the normal cdf. This is given by the integral

\[ \Phi \left( \frac{-X_i\beta}{\sigma} \right) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-\frac{X_i\beta}{\sigma}} e^{-\frac{1}{2}s^2} ds \]

which does not have an analytical solution. Although very fast and accurate approximation methods that do not require us to evaluate this integral exist, it helps us illustrate the point that we need to learn how to evaluate integrals numerically.

Numerical Integration: Newton-Coates Formulas

Let \( f \) be a continuous function whose domain includes the closed interval \([x_1, x_{N+1}]\). We now investigate ways of approximating the definite integral

\[ \int_{x_1}^{x_{N+1}} f(x) \, dx. \]

The most straightforward numerical integration technique would be to approximate the integral by the use of Riemann sums. Approximations obtained in this way are also called Newton-Coates formulas. If we divide the interval into \( N + 1 \) equally spaced points of size \( \Delta x \), a Riemann approximation to the integral would be

\[ L_N = \sum_{j=1}^{N} f(x_j) \Delta x \]

if we evaluate using the left hand endpoints;

\[ R_N = \sum_{j=1}^{N} f(x_{j+1}) \Delta x \]
if we evaluate using the right hand endpoints; and

\[ M_N = \sum_{j=1}^{N} f \left( \frac{x_{j+1} + x_j}{2} \right) \Delta x \]

if we use the midpoint rule.

This type of approximations are also called rectangular rules since all we are doing is approximating the integral by summing rectangular areas. For the case in which we use \( L_N \) our approximation would look like

\[
T_N = \sum_{j=1}^{N} \frac{f(x_{j+1}) + f(x_j)}{2} \Delta x = \frac{L_N + R_N}{2}
\]

and graphically

It is clear that not choosing a sufficiently small step-size \( \Delta a \) can lead to a systematic error. The choice of the interpolating function can be refined to obtain more accuracy with fewer evaluations of the function \( f(x) \).

The trapezoidal rule uses trapezoids (i.e., a rectangle with a triangle on top) instead of rectangles to approximate the area. The trapezoidal rule takes the form

\[
T_N = \sum_{j=1}^{N} \frac{f(x_{j+1}) + f(x_j)}{2} \Delta x = \frac{L_N + R_N}{2}
\]
If instead of approximating the interval with a straight line between two points we approximate with a polynomial on three points, we obtain Simpson’s \( \frac{1}{3} \) rule

\[ S_N = \frac{T_N + 2M_N}{3}. \]

That is, the integral between any two of our points \( \int_{x_j}^{x_{j+1}} f(x) \, dx \) would be approximated by dividing the \( [x_j, x_{j+1}] \) interval further in half. Call this middle point \( x_{j,j+1} \) then

\[ \int_{x_j}^{x_{j+1}} f(x) \, dx \approx (x_{j+1} - x_j) \frac{f(x_j) + 4f(x_{j,j+1}) + f(x_{j+1})}{6}. \]

All of the preceding formulas assume that we now in advance how many points we want to use when evaluating the integral. A natural refinement would be instead to start increasing the number of points until some convergence criterion (i.e., until the evaluation of a given interval changes by only \( \varepsilon \)) is reached. So, for example, for the trapezoidal rule the simple integration routine

\[ T_N = \sum_{j=1}^{N} \frac{f(x_{j+1}) + f(x_j)}{2} \Delta x \]

where \( N \) is known in advance can be substituted for
1. Define \( \text{old}T = 0 \)

2. FOR \( j = 1, 2, ..., JMAX \)

   (a) Calculate
   \[
   T = \sum_{i=1}^{j} \frac{f(x_{i+1}) + f(x_i)}{2} \Delta x_j
   \]

   (b) IF \( (|T - \text{old}T| \leq \varepsilon \times |\text{old}T|) \) then exit the loop and finish. Otherwise

   (c) Redefine \( \text{old}T = T \) and continue the loop.

Here \( JMAX \) is some maximum number of points we are willing to use and the \( j \) subscript on \( \Delta x \) means that we adjust it so that the interval of integration is divided into \( j \) equally spaced intervals of size \( \Delta x_j \).

The webpage includes Fortran code for these refined versions of both the trapezoidal and Simpson’s rule. However, the preferred Newton-Coates method of integration is called the Romberg method of integration which can yield accurate results for \textit{many} fewer function evaluations.

The concept of Richardson’s deferred approach to the limit lies behind the Romberg method of integration. It tells us to evaluate the integral at various stepsizes \( \Delta x \) and then extrapolate them to the limit case of \( \Delta x = 0 \). If we let \( K \) be a minimum number of subintervals used before starting to extrapolate the algorithm would be

1. Define \( KM = K - 1, \Delta x_1 = 1 \)

2. FOR \( j = 1, 2, ..., JMAX \)

   (a) Define \( T_j = \sum_{i=1}^{j} \frac{f(x_{i+1}) + f(x_i)}{2} \Delta x_j \)

   (b) IF \( (j \geq K) \) then

      i. Extrapolate \( T_{j-KM}, ..., T_j \) to the case where \( \Delta x = 0 \). Let the result be \( SS \) and the change in the value of \( SS \) be \( \Delta SS \).

      ii. IF \( (|\Delta SS| \leq \varepsilon \times |SS|) \) exit the loop and finish. Otherwise keep going.

   (c) Define \( T_{j+1} = T_j \) and \( \Delta x_{j+1} = \frac{\Delta x_j}{4} \) and continue the loop.
Up to now we have dealt with proper integrals. To deal with integrals that have infinite limits we are going to need a routine that does not need evaluation at the endpoints\textsuperscript{1}. The trapezoid rule we have been using until now is of no use to us for this case. Instead we are going to use a generalization of the midpoint rule. The idea behind evaluating improper integrals is to make a change in variables so as to map an infinite range of integration into a finite one.

A good example that works for functions that decrease towards infinity faster than $\frac{1}{x^2}$ is

$$\int_{a}^{b} f(x) \, dx = \int_{\frac{1}{b}}^{\frac{1}{a}} \frac{1}{t^2} f\left(\frac{1}{t}\right) \, dt$$

for $ab > 0$. That is, this change of variables can be used for integrals between $b = \infty$ and $a > 0$ or $a = -\infty$ and $b < 0$. Using the midpoint rule properly adjusted to make the change in variables, we can then use a slightly changed version of Romberg integration, the only change being that instead of dividing by 4 when defining $\Delta x_{j+1}$ we divide by 9 to account for the fact that we are now using the midpoint rule.

With this altered Romberg routine we can now do integrals when one of the limits is infinite and the other limit is of the same sign. If you need to integrate say between $a < 0$ and $b = \infty$ what you do is break the integral into two parts. One an integral between $a$ and some positive number $c$, and then add the integral between $c$ and $b$ done using our changed routine\textsuperscript{2}. In the same way an integral in the range $(-\infty, \infty)$ would be done by breaking it into 3 pieces.

**Numerical Integration: Gaussian Quadrature**

The numerical integration methods described so far are based on a rather simple choice of evaluation points for the function $f(x)$. However, if one has the freedom to choose not only the weights but also the points at which to evaluate $f(x)$, a careful choice can, in principle, lead to much more accuracy in evaluating the integral in question. That is, in theory, since we now have twice as many degrees of freedom, we can achieve formulas with almost twice the order as the Newton-Coates formulas for the same number of function evaluations. However, an important thing to understand is that higher order does not automatically translate into higher accuracy.

\textsuperscript{1}A similar procedure is used for integrals with singularities. See the Numerical Recipes book.

\textsuperscript{2}Of course, no computer can actually use $\infty$ or $-\infty$. What we do is simply substitute the limit by a very large number. In FORTRAN you define the largest number the machine can support with the huge comand as in $b = HUGE(b)$. 

5
Gaussian quadrature also approximates the integral by using a weighted sum
\[
\int_{a}^{b} W(x) f(x) \, dx \approx \sum_{j=1}^{N} \omega_j f(x_j),
\]
where the function \( W(x) \) can be chosen so as to help us make the problem simpler (like eliminating singularities for example). A more natural way of looking at this approximation comes from defining \( g(x) = W(x) f(x) \) and \( v_j = \frac{\omega_j}{W_j} \) so that
\[
\int_{a}^{b} g(x) \, dx \approx \sum_{j=1}^{N} v_j g(x_j).
\]

The simplest form of Gaussian integration is based on the use of an optimally chosen polynomial to approximate the integrand \( f(x) \) over the interval \([-1, 1]\). Although the details of the determination of this polynomial are beyond the scope of this presentation the case in which we set \( W(x) = 1 \) is simple enough to illustrate what is going on. In this particular case, the points at which to evaluate \( f(x) \) are the roots of so called Legendre polynomials, over the interval. Suppose we were to set \( N = 2 \) so we would be looking for a rule of the form
\[
\int_{-1}^{1} f(x) \, dx \approx \omega_1 f(x_1) + \omega_2 f(x_2) + E(f).
\]
We require that \( E(f) = 0 \) for \( f(x) = \{1, x, x^2, x^3\} \) so that we have four equations on 4 unknowns

\[
\begin{align*}
    f = 1 : & \omega_1 + \omega_2 = 2 \\
    f = x : & \omega_1 x_1 + \omega_2 x_2 = 0 \\
    f = x^2 : & \omega_1 x_1^2 + \omega_2 x_2^2 = \frac{2}{3} \\
    f = x^3 : & \omega_1 x_1^3 + \omega_2 x_2^3 = 0.
\end{align*}
\]
and solving we get
\[
\begin{align*}
    \omega_1 &= \omega_2 = 1 \\
    x_1 &= -x_2 = \frac{1}{\sqrt{3}}.
\end{align*}
\]
In particular, in the Gauss-Legendre case the resulting value of the integral approximation is of the same accuracy as a simple polynomial method (such as Simpson’s Rule) of about twice the degree.
(i.e., of degree $2N$). Thus, the carefully designed choice of function evaluation points in the Gauss-Legendre form results in the same accuracy for about half the number of function evaluations, and thus at about half the computing effort.

To extend our result to the $[a, b]$ interval from the $[-1, 1]$ interval we simply make a change of variables

$$t = c + mx$$
$$c = \frac{(b + a)}{2}$$
$$m = \frac{(b - a)}{2}$$

so that

$$\int_{a}^{b} f(t) dt = m \int_{-1}^{1} f(c + mx) dx$$
$$\approx m \sum_{j=1}^{N} \omega_j f(c + mx_j).$$

For several common $W(x)$ the quadrature rules (i.e., the polynomials we need to solve) are well known. The most common rules are:

1. Gauss-Legendre: $W(x) = 1$ on the interval $[-1, 1]$.

2. Gauss-Chebyshev: $W(x) = (1 - x^2)^{-\frac{1}{2}}$ on the interval $[-1, 1]$.

3. Gauss-Hermite: $W(x) = e^{-x^2}$ on the interval $(-\infty, \infty)$.

4. Gauss-Laguerre: $W(x) = x^\alpha e^{-x}$ on the interval $[0, \infty)$ for a given $\alpha$.

5. Gauss-Jacobi: $W(x) = (1 - x)^\alpha (1 + x)^\beta$ on the interval $[-1, 1]$ for a given $\alpha$ and a given $\beta$.

Routines to find the weights and points for the different rules are available on the webpage.
Numerical Integration: Multidimensional Integrals

Multidimensional integrals are, simply put, a very hard task using the methods we have shown so far for at least two reasons. The first is that the number of function evaluations required to evaluate an integral in \( n \) dimensions grows to the \( n^{th} \) power of the number needed for a one dimensional integral. That is, if we require 20 points to evaluate an integral in one dimension, we will require \( 20 \times 20 \times 20 = 20^3 = 8,000 \) evaluations for an integral in 3 dimensions. The second difficulty lies in the fact that whereas the limits of integration for a one-dimensional integral are simply a pair of numbers, they can be very complicated functions when moving to a multidimensional case.

Take a simple two dimensional integral of the form

\[
\int_{a}^{b} \int_{c(x_2)}^{d(x_2)} g(x_2, x_1) \, dx_1 \, dx_2.
\]

How do we create an algorithm to solve this integral? First define:

\[
G(x_2) = \int_{c(x_2)}^{d(x_2)} g(x_2, x_1) \, dx_1
\]

and

\[
H = \int_{a}^{b} G(x_2) \, dx_2.
\]

Then it is simply a matter of applying any of our routines recursively to perform the multidimensional integration.

Numerical Integration: Monte Carlo Integration

Monte Carlo integration can sometimes be the only feasible answer to a hard multidimensional integration problem. By hard here we mean problems with very complicated boundaries where the integrand is not strongly peaked in very small regions and relatively low accuracy is tolerable. The idea behind Monte Carlo is simple: find a region with simple boundaries that includes the region of integration. Find a simple method to sample random numbers inside the simple region and then evaluate whether the resulting point is inside or outside the region of integration.

Suppose you want to evaluate the integral

\[
\int_{a}^{b} g(x) \, dx
\]
we know from before that this can be approximated by

\[
\frac{(b - a)}{N} \sum_{i=1}^{N} g(x_i).
\]

The difference now is that, instead of picking numbers at regular intervals or as the result of solving some polynomial, we pick them randomly in a region that contains our region of integration. Notice that whereas other integration methods have an error proportional to \(\frac{1}{N^d}\) whereas the error in Monte Carlo integration is purely statistical and so it is only proportional to \(\frac{1}{\sqrt{N}}\).³

The simplest version of this method (sometimes called Crude Monte Carlo) would have us pick a box like \(R_1\) on the figure and sample uniformly on that box. It is immediately clear, that sampling from a region like \(R_2\) (say an exponential distribution) that surrounds our region of integration

³A simple example is the estimation of \(\pi\) by a two dimensional Monte Carlo integration method

\[
\pi = \int_{-1}^{1} \int_{-1}^{1} p(x, y) \, dx \, dy
\]

\[
\approx \frac{4}{N} \sum_{i=1}^{N} p(x_i, y_i)
\]

\[
p(x, y) = \begin{cases} 
1 & \text{if } x^2 + y^2 \leq 1 \\
0 & \text{otherwise}
\end{cases}
\]
tightly, would get us an estimate of the integral with less computing effort. Sampling from a distribution other than the uniform distribution is sometimes called importance sampling.

Formally, Monte Carlo integration can be defined as

**Definition 1** The **Monte Carlo Integration method** consists of an independent sample of values $x_1, x_2, \ldots, x_N$ from a distribution $F(x)$ and an approximation to the integral

$$M = \int_a^b g(x) \, dx$$

$$= \int_a^b \frac{g(x)}{f(x)} f(x) \, dx$$

given by

$$\hat{M} = \frac{1}{N} \sum_{i=1}^{N} g(x_i) f(x_i).$$

The density $f(x)$ is called the importance function.

Notice that, since $M$ can be interpreted as the expected value of $\frac{g(x)}{f(x)}$, it follows from the law of large numbers that $\hat{M}$ is a consistent estimator for it. Furthermore, it is straightforward to prove (check it!) that it is also unbiased and that its variance is given by

$$\text{var} \left( \hat{M} \right) = \frac{1}{N} \left( \int \frac{g^2(x)}{f(x)} \, dx - M^2 \right).$$

Notice that the variance of the estimator is an inverse function of the importance function $f(x)$ so an appropriate choice of $p$ should lead to the smallest possible variance. This means that we need to study how to generate (pseudo) random samples from not only the uniform distribution but other distributions as well.

**Pseudo Random Number Generation**

The building block for any work on stochastic modeling or Monte Carlo technique is the pseudo uniform random number generator. A uniform random deviate is just a random number that lies within a specified range (typically $(0, 1)$) where any number is equally likely as any other. There is, of course, a logical impossibility in generating random numbers from a computer that can only
produce output from a program and is thus entirely predictable. This is where the pseudo in pseudo random numbers comes to play\(^4\).

Uniform random generators typically create their deviates by solving some deterministic difference equation \(u_i = D (u_{i-1})\) given some initial value \(u_0\) (the seed of the generator). Good generators produce sequences that are virtually indistinguishable from true i.i.d., sequences. Since any programming language that you use for numerical computing will contain a uniform random number generator there will be no reason to write your own. In theory, this is all you require since random numbers from any other distribution can be generated from the uniform distribution by means of the inverse cdf method.

**Lemma 1** If \(u \sim U (0, 1)\), then \(F^- (u) \sim F\) where \(F^-\) is a “generalized” inverse of \(F\). That is \(F^- (u) = \inf \{ x : F (x) \geq u \}\).

**Proof** We must show that \(\Pr (F^- (u) \leq x) = F (x)\).

Suppose first that \(F\) is continuous and strictly increasing. Then

\[
F^- (u) \leq x \iff u \leq F (x).
\]

If \(F\) is either non-decreasing or has discontinuities then

\[
\{ x : F (x) \geq u \} = \left[ F^- (u), \infty \right),
\]

so

\[
F^- (u) \leq x \iff u \leq F (x).
\]

In any case it follows that

\[
\Pr (F^- (u) \leq x) = \Pr (u \leq F (x)) = F (x).
\]

So, in theory we can generate random numbers from any distribution by generating uniform random numbers first and then evaluating the inverse cdf at those values. This, however, is not always either feasible or the best way of generating random numbers. Here is an example in which it works pretty well.

\(^4\)From now on I will refer to pseudo random numbers simply as random numbers or random deviates, you however have been warned as to the “real” nature of these numbers.
Example 3 Suppose we wish to sample from an exponential distribution with parameter \( \lambda \),
\[
x \sim EXP(\lambda), \quad \lambda > 0.
\]
The cdf is
\[
F(x) = 1 - e^{-\lambda x}
\]
so
\[
u = F(x) \Rightarrow x = F^{-1}(u) = -\frac{1}{\lambda} \ln (1 - u).
\]
The algorithm is then:

1. **Sample** \( u \) **from a uniform distribution on** \((0, 1)\).
2. **Define** \( x = -\frac{1}{\lambda} \ln (1 - u) \).

An example of a distribution for which this method is not particularly recommended is the standard normal. In particular, we require to estimate the inverse cdf of a normal distribution to generate numbers of the form
\[
x = \Phi^{-1}(u).
\]
Not only can this be costly in terms of computation (although not that much to be honest) but it is rather inaccurate for numbers close to 0 or 1.

An alternative to the inverse cdf method is the transformation method. To see how this works remember that if \( X_1, X_2, \ldots, X_N \) has a joint pdf given by \( f_{X_1, X_2, \ldots, X_N} \) and \( Y_1, Y_2, \ldots, Y_N \) are each a continuous function of all the \( X \)'s then
\[
f_{Y_1, Y_2, \ldots, Y_N} = f_{X_1, X_2, \ldots, X_N} \left| \frac{\partial (X_1, \ldots, X_N)}{\partial (Y_1, \ldots, Y_N)} \right|.
\]
A nice exercise for you to check is to prove that the inverse cdf method for the exponential can actually be derived by using the Jacobian of the transformation formula above for the case of \( N = 1 \).

An important example of the use of the multidimensional case is the Box-Muller method for generating random numbers from a standard normal distribution. Take the transformation of two uniform numbers \( x_1, x_2 \) and two quantities \( y_1, y_2 \) such that
\[
y_1 = \sqrt{-2 \ln x_1} \cos (2\pi x_2)
y_2 = \sqrt{-2 \ln x_1} \sin (2\pi x_2).
\]
or, solving for $x_1$ and $x_2$

$$x_1 = e^{\left(-\frac{1}{2}(y_1^2 + y_2^2)\right)}$$
$$x_2 = \frac{1}{2\pi} \tan^{-1}\left(\frac{y_2}{y_1}\right).$$

The determinant of the Jacobian of this transformation can be found to be

$$-\frac{1}{\sqrt{2\pi}} e^{-\frac{y_1^2}{2}} \frac{1}{\sqrt{2\pi}} e^{-\frac{y_2^2}{2}}$$

which is the product of two standard normals so $y_1$ and $y_2$ are actually two independent standard normals!

This might not seem to be a very attractive routine since apparently we would need to evaluate a sine and a cosine. However, if you remember or look for trigonometric properties you can see that, if we define $v_1$ and $v_2$ as the ordinate and abscissa of a random point in the inner circle around the origin then we can define $x_1 = v_1^2 + v_2^2$ and the angle between $(v_1, v_2)$ with respect to the $v_1$ axis can serve as $2\pi x_2$. Why is this useful? Because then we can rewrite

$$\cos(2\pi x_2) = \frac{v_1}{\sqrt{v_1^2 + v_2^2}}$$
$$\sin(2\pi x_2) = \frac{v_2}{\sqrt{v_1^2 + v_2^2}}$$

and we avoid having to evaluate the trigonometric functions!

So, to sample from a standard normal using the Box-Muller transformation we would

1. IF logical is false

   (a) Sample two uniform numbers $u_1, u_2$ on $(0,1)$.

   (b) Define $v_1 = 2u_1 - 1$ and $v_2 = 2u_2 - 1$ so now we have numbers on $(-1,1)$.

   (c) Define $v = v_1^2 + v_2^2$.

   (d) IF $v > 1$ OR $v = 0$ return to step a since they are not a random point in the unit circle.

   (e) Define $aux = \sqrt{-\frac{2\ln v}{v}}$ (start the transformation)

   (f) Define $y_1 = aux * v_1$ and $y_2 = aux * v_2$. 

13
(g) Return $y_1$ as my normal random number, save $y_2$ and set logical to true.

2. IF logical is true

(a) Return the saved $y_2$ as my normal random number.

The final algorithm to sample random numbers from a given distribution that I am going to present is called the accept-reject algorithm. This method is based on sampling from a proposal distribution from which it is easy to sample and then accept or reject the proposed number based on some probability so that the accepted numbers have the correct distribution. One advantage of this method is that you do not need to know the normalizing constant of the distribution you are trying to sample from.

Let $f(x)$ be the pdf of the distribution we are trying to sample from and let $g(x)$ be the pdf of my proposal distribution. Then, if we impose the requirement that:

**Requirement** For the accept-reject algorithm to work a constant $M$ such that

$$f(x) \leq Mg(x),$$

for all $x$ in the support of $f$ must exist.

the algorithm would be given by

1. Generate $s \sim g$ and $u \sim U(0,1)$

2. Accept $x = s$ if $u \leq \frac{f(x)}{Mg(x)}$

3. Otherwise return to 1.

The main problem of accept-reject is finding a good proposal distribution so that we accept as the proposed number as often as possible. Notice that the probability of accepting a draw is

$$\Pr \left( u \leq \frac{f(x)}{Mg(x)} \right) = \int_{-\infty}^{\infty} \int_{0}^{\frac{f(x)}{Mg(x)}} dug(x) dx$$

$$= \int_{-\infty}^{\infty} \frac{f(x)}{Mg(x)} g(x) dx$$

$$= \frac{1}{M}.$$
so we want $M$ to be as close to 1 as possible (i.e., we want the proposal and target distributions to be as similar as possible).

Suppose we wanted to sample standard normal random numbers using accept-reject and we use a double exponential distribution

$$g(x) = \frac{\alpha}{2} e^{-\alpha |x|}, \alpha > 0.$$ 

We need to derive the most efficient bound $M$.

$$\frac{f(x)}{g(x)} = \frac{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}}{\frac{\alpha}{2} e^{-\alpha |x|}}$$

$$= \sqrt{\frac{2}{\pi}} \alpha^{-1} e^{-\frac{1}{2}\alpha^2}$$

$$: \quad = M(\alpha)$$

The $\alpha$ that minimizes this ratio is $\alpha = 1$.

A final note on how to sample random numbers. All of the examples we have presented here are for standard normal random variables. To sample from a $N(\mu, \sigma^2)$ we would

1. Generate $u \sim N(0, 1)$.

2. Compute $x = \mu + \sigma u$.

To get random vectors from $N(\mu, \Sigma)$, a $K$ dimensional normal distribution, we would

1. Compute the Choleski decomposition of $\Sigma, \Sigma = CC'$, where $C$ is a lower triangular matrix.

2. Generate $u_j \sim N(0, 1), j = 1, ..., K$ and let $u = (u_1, ..., u_K)'$.

3. Compute $x = \mu + Cu$. 