Notes on *Discrete Choice Methods with Simulation* (Train, 2009)

Chapter 5. Probit

- The standard logit model is limited in three important ways:
  - It cannot represent random taste variation.
  - It exhibits restrictive substitution patterns due to the IIA property.
  - It cannot be used with panel data when unobserved factors are correlated over time for each decision maker.
- GEV models relax the second of these restrictions, but not the other two.
- Probit models can deal with all three, however, they require normal distributions for all unobserved components of utility, which, in some situations, are inappropriate and can lead to perverse forecasts (e.g., the coefficients of price is less likely to be positive).

PYTHON

```
import numpy as np
import scipy
import matplotlib.pyplot as plt
```

5.1. Choice probabilities

- Suppose that utility can be decomposed into observed and unobserved parts: $U_{nj} = V_{nj} + \varepsilon_{nj}$ for any j. Let $\varepsilon_n = [\varepsilon_{n1}, \cdots, \varepsilon_{nJ}]'$ be the vector of unobserved terms. We assume that ε_n is distributed normal with a mean vector of zero and covariance matrix Ω. The density of ε_n is

$$\boldsymbol{\Phi}\left(\varepsilon_n\right) = \frac{1}{(2\pi)^{J/2}|\Omega|^{1/2}} \mathrm{e}^{-\frac{1}{2}\varepsilon_n'\Omega^{-1}\varepsilon_n}.$$

- The choice probability is

$$P_{ni} = \int \mathbb{I}\left(V_{ni} + \varepsilon_{ni} > V_{nj} + \varepsilon_{nj},\ \forall j \neq i\right) \boldsymbol{\Phi}\left(\varepsilon_n\right) \mathrm{d}\varepsilon_n.$$

  This integral does not have a closed form, and it must be evaluated numerically through simulation.
- Since only differences in utility matter, the choice probabilities can be equivalently expressed as $(J-1)$ -dimensional integrals. Define $\tilde{U}_{nji} = U_{nj} - U_{ni}$, $\tilde{V}_{nji} = V_{nj} - V_{ni}$, and $\tilde{\varepsilon}_{nji} = \varepsilon_{nj} - \varepsilon_{ni}$. Let $\tilde{\varepsilon}_{ni}$ be the $(J-1)$-dimensional vector made up of these $\tilde{\varepsilon}_{nji}$, and its density is

$$\boldsymbol{\Phi}\left(\tilde{\varepsilon}_{ni}\right) = \frac{1}{(2\pi)^{\frac{1}{2}(J-1)}\left|\tilde{\Omega}_i\right|^{1/2}} \mathrm{e}^{-\frac{1}{2}\tilde{\varepsilon}_{ni}'\tilde{\Omega}_i^{-1}\tilde{\varepsilon}_{ni}},$$

  where $\tilde{\Omega}_i = M_i \Omega M_i'$ is the covariance matrix of $\tilde{\varepsilon}_{ni}$. M_i is the $((J-1) \times J)$-dimensional matrix that transforms vector ε_n into $\tilde{\varepsilon}_{ni}$. Then the choice probability expressed in utility differences is

$$P_{ni} = \int \mathbb{I}\left(\tilde{V}_{nji} + \tilde{\varepsilon}_{nji} < 0,\ \forall j \neq i\right) \boldsymbol{\Phi}\left(\tilde{\varepsilon}_{ni}\right) \mathrm{d}\tilde{\varepsilon}_{ni}.$$

5.2 Identification

- A parameter is *identified* if it can be estimated. Parameters that do not affect the behavior of decision makers cannot be estimated.
- As described earlier, any discrete choice model must be normalized to take account of the fact that the level and scale of utility are irrelevant. In logit and nested logit models, the normalization for scale and level occurs automatically with the distributional assumptions that are placed on the error terms. With probit models, however, normalization for scale and level does not occur automatically. The researcher must normalize the model directly.
- A procedure that can always be used to normalize a probit model and assure that all parameters are identified:
  1. To set the level of utility, take the differences with respect to the first alternatives and define error differences as $\tilde{\varepsilon}_{nj1} = \varepsilon_{nj} - \varepsilon_{n1}$ and the vector of error differences as $\tilde{\varepsilon}_{n1} = [\tilde{\varepsilon}_{n21}, \cdots, \tilde{\varepsilon}_{nJ1}]'$. The covariance matrix for this vector is then

$$\tilde{\Omega}_1 = \begin{bmatrix} \theta_{22} & \theta_{23} & \cdots & \theta_{2J} \\ \cdot & \theta_{33} & \cdots & \theta_{3J} \\ \vdots & \vdots & & \vdots \\ \cdot & \cdot & \cdots & \theta_{JJ} \end{bmatrix}.$$

  2. To set the scale of utility, set the top-left element of $\tilde{\Omega}_1$ to 1. This normalization for scale gives the following covariance matrix:

$$\tilde{\Omega}_1^* = \begin{bmatrix} 1 & \theta_{23}^* & \cdots & \theta_{2J}^* \\ \cdot & \theta_{33}^* & \cdots & \theta_{3J}^* \\ \vdots & \vdots & & \vdots \\ \cdot & \cdot & \cdots & \theta_{JJ}^* \end{bmatrix},$$

     where there are $(J+1)J/2 - 1$ parameters to be estimated. These are the only identified parameters in the model.
- Researchers often impose restrictions on the elements of Ω to reflect their assumptions about error correlations. These restrictions may be sufficient to normalize the model. To test this, we can apply the procedure stated above: First, derive $\tilde{\Omega}_1^*$ based on the restricted Ω. Then, check whether each constrained element of Ω can be recovered from $\tilde{\Omega}_1^*$. If so, the restrictions are sufficient for normalization.

5.3. Taste variation

- Probit is particularly well suited for incorporating random coefficients, provided that the coefficients are normally distributed.
- Assume that representative utility is linear in parameters and that the coefficients vary randomly over decision makers. The utility is $U_{nj} = \beta_n' x_{nj} + \varepsilon_{nj}$, where β_n is the vector of coefficients for decision maker n representing that person's tastes. Suppose β_n is normally distributed in the population with mean b and covariance W. The goal of the research is to estimate the parameters b and W.
- Therefore, we have $U_{nj} = b' x_{nj} + \tilde{\beta}_n' x_{nj} + \varepsilon_{nj}$, where $\tilde{\beta}_n = \beta_n - b$. The last two terms are random, and denote their sum as η_{nj} to obtain $U_{nj} = b' x_{nj} + \eta_{nj}$. Assuming that $\varepsilon_{nj} \sim_{\text{i.i.d.}} N(0, \sigma_\varepsilon^2)$, then η_{nj} is normally distributed with

$$\begin{aligned} \mathbb{E}[\eta_{nj}] &= \mathbb{E}[\beta_n - b]' x_{nj} + \mathbb{E}[\varepsilon_{nj}] = 0, \\ \operatorname{Cov}(\eta_{ni}, \eta_{nj}) &= \operatorname{Cov}\left(\tilde{\beta}_n' x_{ni}, \tilde{\beta}_n' x_{nj}\right) = x_{ni}' W x_{nj}, \ \forall i \neq j, \\ \operatorname{Var}[\eta_{nj}] &= \operatorname{Var}\left[\tilde{\beta}_n' x_{nj}\right] + \operatorname{Var}[\varepsilon_{nj}] = x_{nj}' W x_{nj} + \sigma_\varepsilon^2. \end{aligned}$$

Therefore, $\eta_n \equiv [\eta_{n1}, \cdots, \eta_{nJ}]' \sim N\left(\mathbb{0}, XWX' + \sigma_\varepsilon^2 I_J\right)$, where $X \equiv [x_{n1}, \cdots, x_{nJ}]'$. The last step is set the scale of utility. A convenient normalization is $\sigma_\varepsilon^2 = 1$. Finally, we have $\eta_n \equiv [\eta_{n1}, \cdots, \eta_{nJ}]' \sim N\left(\mathbb{0}, XWX' + I_J\right)$.

- Let $\Omega \equiv XWX' + I_J$, the derivation above suggests how a probit model can be derived from a random coefficient model.

5.4. Substitution patterns and failure of IIA

- Probit can represent any substitution pattern through appropriate specification of its covariance matrices Ω. A full covariance matrix can be estimated, or the researcher can impose structure on the covariance matrix to represent particular sources of nonindependence.
- Consider a full covariance matrix Ω without any restriction. When normalized for scale and level, it becomes Ω_1^* and the elements of Ω_1^* are estimated. The estimated values can represent any substitution pattern, however, the estimated values of the θ^*'s provide essentially no interpretable information in themselves (e.g., $\theta_{33}^* > \theta_{44}^*$ does not necessarily imply $\sigma_{33} > \sigma_{44}$). As we cannot recover Ω from Ω_1^*, estimating a full covariance matrix renders the estimated parameters essentially uninterpretable.
- Consider a covariance matrix with some restrictions on its structure. These restrictions reduce the ability of the model to represent various substitution patterns, however, if the structure is correct, then the true substitution pattern would be able to be interpreted by the estimates (see the textbook for an example).

5.5. Panel data

- Consider a decision maker who faces a choice among J alternatives in each of T time periods or choices situations. The vector of errors for all alternatives in all time periods is denoted as $\varepsilon_n = [\varepsilon_{n11}, \cdots, \varepsilon_{nJ1}, \varepsilon_{n12}, \cdots, \varepsilon_{nJ2}, \cdots, \varepsilon_{n1T}, \cdots, \varepsilon_{nJT}]'$. The covariance matrix of this vector is denoted as Ω, which has dimension $JT \times JT$. Consider a sequence of alternatives, one for each time period, $\mathbf{i} = \{i_1, \cdots, i_T\}$. The probability that the decision maker makes this sequence of choices is

$$P_{n\mathbf{i}} = \Pr\left(V_{ni_t t} + \varepsilon_{ni_t t} > V_{njt} + \varepsilon_{njt}, \forall j \neq i_t, \forall t\right),$$

  or in utility differences:

$$P_{n\mathbf{i}} = \Pr\left(\tilde{V}_{njt} + \tilde{\varepsilon}_{njt} < 0, \forall j \neq i_t, \forall t\right),$$

5.6. Simulation of the choice probabilities

- The probit probabilities do not have a closed-form expression and must be approximated numerically. Simulation has proven to be very general and useful for approximating probit probabilities.

5.6.1. Accept–reject simulator

- Given the choice probabilities are

$$P_{ni} = \int \mathbb{I}\left(V_{ni} + \varepsilon_{ni} > V_{nj} + \varepsilon_{nj},\ \forall j \neq i\right) \Phi\left(\varepsilon_n\right) \mathrm{d}\varepsilon_n,$$

  the accept–reject (AR) simulator of this integral is calculated as follows:

  1. Draw a value of the J-dimensional vector of errors, ε_n , from a normal density with zero mean and covariance Ω. Label the draw ε_n^r with $r = 1$, and the elements of the draw as $\varepsilon_{n1}^r, \cdots, \varepsilon_{nJ}^r$.

2. Using these values of the errors, calculate the utility that each alternative obtains with these errors. That is, calculate $U_{nj}^r = V_{nj} + \varepsilon_{nj}^r$ for any j.
3. Determine whether the utility of alternative i is greater than that for all other alternatives. That is, calculate $I^r = 1$ if $U_{ni}^r > U_{nj}^r$ for any $j \neq i$, indicating an accept, and $I^r = 0$ otherwise, indicating a reject.
4. Repeat steps 1–3 many times. Label the number of repetitions (including the first) as R, so that r takes values of 1 through R.
5. The simulated probability is the proportion of draws that are accepts: $\check{P}_{ni} = \frac{1}{R}\sum_{r=1}^{R} I^r$.

- We can use Choleski transformation to draw from a multivariate normal distribution (see 9.1.5 in Chapter 9).
- This procedure can also be applied analogously to utility differences, which reduces the dimension of the integral to $J-1$. The choice probabilities can be expressed in terms of utility differences:

$$P_{ni} = \int \mathbb{I}\left(\tilde{V}_{nji} + \tilde{\varepsilon}_{nji} < 0,\ \forall j \neq i\right) \Phi\left(\tilde{\varepsilon}_{ni}\right) \mathrm{d}\tilde{\varepsilon}_{ni},$$

  where $\Phi(\tilde{\varepsilon}_{ni})$ is the joint normal density with zero mean and covariance $\tilde{\Omega}_i = M_i \Omega M_i'$.
- The AR simulator is very general and can be applied to any model for which draws can be obtained for the random terms and the behavior that the decision maker would exhibit with these draws can be determined.
- However, the AR simulator has several disadvantages, particularly when used in the context of MLE. Recall that the log-likelihood function is $\mathrm{LL} = \sum_n \sum_j d_{nj} \ln P_{nj}$, where $d_{nj} = 1$ if n chose j and 0 otherwise. When the probabilities cannot be calculated exactly, as in the case of probit, the simulated log-likelihood function is used instead, with the true probabilities replaced with the simulated probabilities: $\mathrm{SLL} = \sum_n \sum_j d_{nj} \ln \check{P}_{nj}$. The value of the parameters that maximizes SLL is called the maximum simulated likelihood estimator (MSLE). There are two issues of using the AR simulator in SLL :
  - $\check{P}_{ni}$ can be zero for any finite number of draws of R, however, the log of zero is undefined. Therefore, SLL cannot be calculated if the simulated probability is zero for any decision maker in the sample.
  - The simulated probabilities are step functions and not smooth in the parameters. Hence, they are not twice differentiable. This facts hinders the numerical procedures that are used to locate the maximum of SLL.

> Exercise: Suppose that V_{nj} is the same for each $j \in \{1, \cdots, J\}$ and $\varepsilon_{nj} \sim_{\text{i.i.d.}} N(0,1)$. Due to symmetry, we know that the choice probabilities $P_{nj} = 1/J$ for any j. Calculate P_{ni} numerically with the AR simulator.

PYTHON

```python
# Accept-reject simulator (1)

np.random.seed(123456789)
J = 10
b = np.zeros([J])
omega = np.eye(J)
R = 99999

eps = np.random.multivariate_normal(b, omega, R)
maximizer_index = np.argmax(eps, axis=1)
```

```
p_n1_hat = (maximizer_index==0).sum() / R
print(p_n1_hat)
```

```
0.1004410044100441
```

> Exercise: Suppose that $V_n = [1, 1.2, 1.4, 1.6, 2]'$ and $\varepsilon_n \sim N(\mathbb{0}, \Omega)$, where
>
> $$\Omega = \begin{bmatrix} 1 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.1 & 1 & 0.1 & 0.2 & 0.3 \\ 0.2 & 0.1 & 1 & 0.1 & 0.2 \\ 0.3 & 0.2 & 0.1 & 1 & 0.1 \\ 0.4 & 0.3 & 0.2 & 0.1 & 1 \end{bmatrix}.$$
>
> Calculate P_{ni} numerically with the AR simulator.

PYTHON

```
# Accept-reject simulator (2)

np.random.seed(123456789)
J = 5
V = np.array([1, 1.2, 1.4, 1.6, 1.8]).reshape([-1, 1])
b = np.zeros([J])
omega = np.array([[1, 0.1, 0.2, 0.3, 0.4],
                  [0.1, 1, 0.1, 0.2, 0.3],
                  [0.2, 0.1, 1, 0.1, 0.2],
                  [0.3, 0.2, 0.1, 1, 0.1],
                  [0.4, 0.3, 0.2, 0.1, 1]])
R = 99999

eps = np.random.multivariate_normal(b, omega, R)
U = eps + np.tile(V, R).T
maximizer_index = np.argmax(U, axis=1)
p_n1_hat = (maximizer_index==0).sum() / R
print(p_n1_hat)
```

```
0.06945069450694506
```

5.6.2. Smoothed AR simulators

- One way to mitigate the difficulties with the AR simulator is to replace the 0–1 AR indicator with a smooth, strictly positive function (e.g., logit function). The *logit-smoothed AR simulator* is implemented in the following steps:
  1. Draw a value of the J-dimensional vector of errors, ε_n , from a normal density with zero mean and covariance Ω. Label the draw ε_n^r with $r = 1$, and the elements of the draw as $\varepsilon_{n1}^r, \cdots, \varepsilon_{nJ}^r$.
  2. Using these values of the errors, calculate the utility that each alternative obtains with these errors. That is, calculate $U_{nj}^r = V_{nj} + \varepsilon_{nj}^r$ for any j.

3. Put these utilities into the logit formula. That is, calculate

$$S^r = \frac{\mathrm{e}^{U_{ni}^r/\lambda}}{\sum_j \mathrm{e}^{U_{nj}^r/\lambda}},$$

where $\lambda > 0$ is a scale factor specified by the researcher.

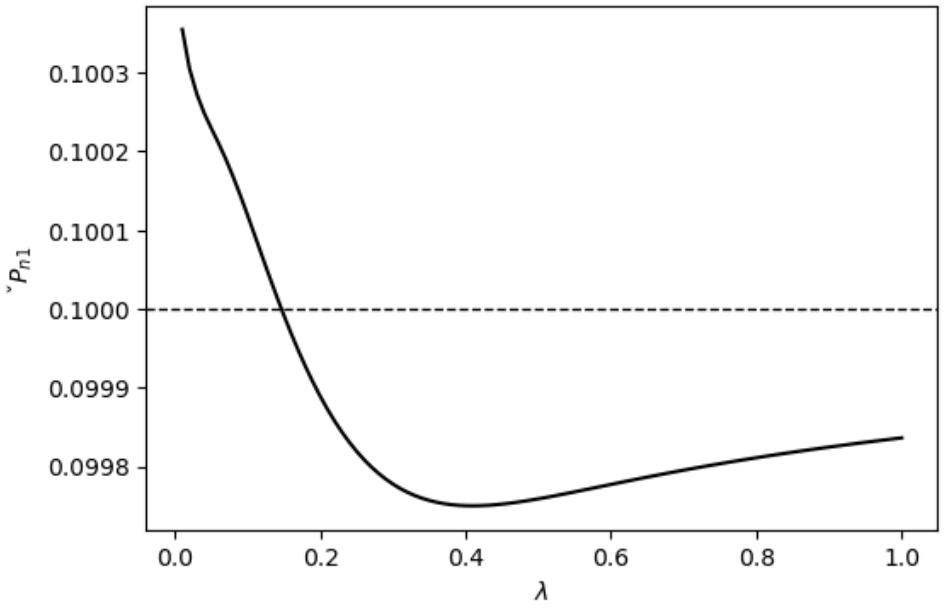
4. Repeat steps 1–3 many times. Label the number of repetitions (including the first) as R, so that r takes values of 1 through R.
5. The simulated probability is the proportion of draws that are accepts: $\check{P}_{ni} = \frac{1}{R}\sum_{r=1}^{R} S^r$.

- Note that the SAR simulator is biased. The scale factor λ determines the degree of smoothing. As $\lambda \to 0$, S^r approaches the indicator function I^r, and then the SAR simulator approaches an unbiased estimator. The researcher wants to set λ low enough to obtain a good approximation but not so low as to reintroduce numerical difficulties. There is little guidance on the appropriate level of λ. Perhaps the nest approach is for the researcher to experiment with different λ's.

> Exercise: Suppose that V_{nj} is the same for each $j \in \{1, \cdots, J\}$ and $\varepsilon_{nj} \sim_{\text{i.i.d.}} N(0,1)$. Due to symmetry, we know that the choice probabilities $P_{nj} = 1/J$ for any j. Calculate P_{ni} numerically with the smoothed AR simulator.

PYTHON

```python
# Smoothed AR simulator (1)

np.random.seed(123456789)
J = 10
b = np.zeros([J])
omega = np.eye(J)
R = 99999
lbd = 1

eps = np.random.multivariate_normal(b, omega, R)

def smoothed_AR_lbd(eps, lbd):
    S_numerator = np.exp(eps[:, 0] / lbd)
    S_denominator = np.exp(eps / lbd).sum(axis=1)
    S = S_numerator / S_denominator
    p_n1_hat = S.mean()

    return p_n1_hat

lbd_range = np.linspace(0.01, 1, 100)
p_n1_hat = np.array([smoothed_AR_lbd(eps, i) for i in lbd_range])

fig, ax = plt.subplots(figsize=(6, 4), dpi=100)
ax.plot(lbd_range, p_n1_hat, 'k-')
ax.set_xlabel('$\lambda$')
ax.set_ylabel('$\check{P}_{n1}$')
ax.axhline(0.1, color='k', ls='--', lw=1)
plt.show()
```



Exercise: Suppose that $V_n = [1, 1.2, 1.4, 1.6, 2]'$ and $\varepsilon_n \sim N(\mathbb{0}, \Omega)$, where

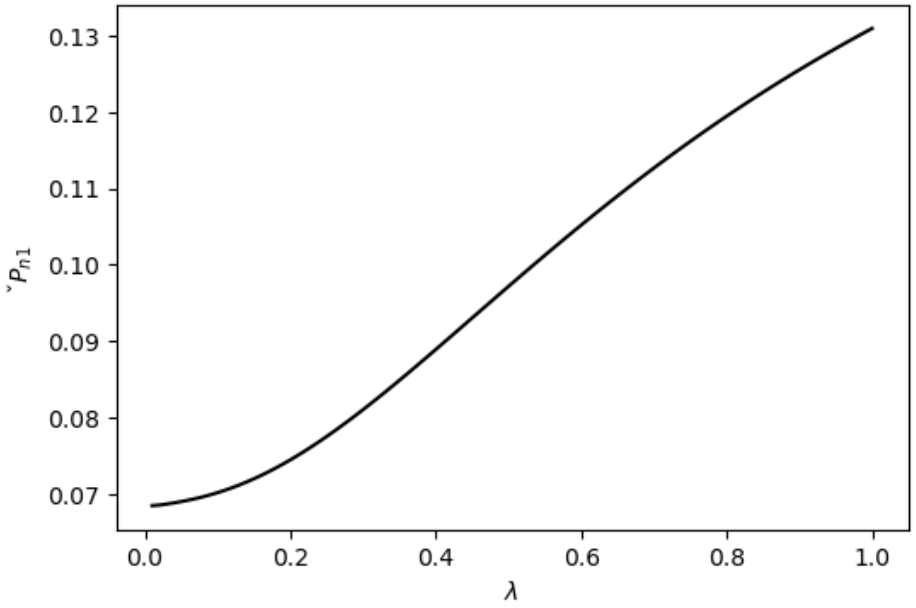
$$\Omega = \begin{bmatrix} 1 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.1 & 1 & 0.1 & 0.2 & 0.3 \\ 0.2 & 0.1 & 1 & 0.1 & 0.2 \\ 0.3 & 0.2 & 0.1 & 1 & 0.1 \\ 0.4 & 0.3 & 0.2 & 0.1 & 1 \end{bmatrix}.$$

Calculate P_{ni} numerically with the smoothed AR simulator.

PYTHON

```python
# Smoothed AR simulator (2)

np.random.seed(123456789)
J = 5
V = np.array([1, 1.2, 1.4, 1.6, 1.8]).reshape([-1, 1])
b = np.zeros([J])
omega = np.array([[1, 0.1, 0.2, 0.3, 0.4],
                  [0.1, 1, 0.1, 0.2, 0.3],
                  [0.2, 0.1, 1, 0.1, 0.2],
                  [0.3, 0.2, 0.1, 1, 0.1],
                  [0.4, 0.3, 0.2, 0.1, 1]])
R = 99999
lbd = 1

eps = np.random.multivariate_normal(b, omega, R)
U = eps + np.tile(V, R).T

def smoothed_AR_lbd(U, lbd):
    S_numerator = np.exp(U[:, 0] / lbd)
    S_denominator = np.exp(U / lbd).sum(axis=1)
    S = S_numerator / S_denominator
    p_n1_hat = S.mean()

    return p_n1_hat

lbd_range = np.linspace(0.01, 1, 100)
p_n1_hat = np.array([smoothed_AR_lbd(U, i) for i in lbd_range])
```

```
fig, ax = plt.subplots(figsize=(6, 4), dpi=100)
ax.plot(lbd_range, p_n1_hat, 'k-')
ax.set_xlabel('$\lambda$')
ax.set_ylabel('$\check{P}_{n1}$')
plt.show()
```



5.6.3. GHK simulator

- The most widely use probit simulator is called GHK, which operates on utility differences.
- Suppose that utility can be expressed as $U_{nj} = V_{nj} + \varepsilon_{nj}$ with $\varepsilon_n \equiv [\varepsilon_{n1}, \cdots, \varepsilon_{nJ}]' \sim N(0, \Omega)$. Transform to utility differences against alternative i: $\tilde{U}_{nji} = \tilde{V}_{nji} + \tilde{\varepsilon}_{nji}$ with $\tilde{\varepsilon}_{ni} \equiv [\tilde{\varepsilon}_{n1i}, \cdots, \tilde{\varepsilon}_{nJi}]' \sim N(0, \tilde{\Omega}_i)$, where $\tilde{\Omega}_i$ is derived from Ω. Re-express the errors as a Choleski transformation of i.i.d. standard normal deviates:

$$
\begin{aligned}
&L_i \quad \text{s.t.} \quad L_i L_i' = \tilde{\Omega}_i, \\
&L_i = \begin{bmatrix}
c_{11} & 0 & \cdots & \cdots & \cdots & 0 \\
c_{21} & c_{22} & 0 & \cdots & \cdots & 0 \\
c_{31} & c_{32} & c_{33} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{bmatrix}.
\end{aligned}
$$

  Then we have $\tilde{U}_{ni} = \tilde{V}_{ni} + L_i \eta_n$, where $\eta_n = [\eta_{1,n}, \cdots, \eta_{J-1,n}]$ is a vector of i.i.d. standard normal deviates. Written explicitly, the model is

$$
\begin{aligned}
\tilde{U}_{n1i} &= \tilde{V}_{n1i} + c_{11}\eta_1, \\
\tilde{U}_{n2i} &= \tilde{V}_{n2i} + c_{21}\eta_1 + c_{22}\eta_2, \\
\tilde{U}_{n3i} &= \tilde{V}_{n3i} + c_{31}\eta_1 + c_{32}\eta_2 + c_{33}\eta_3,
\end{aligned}
$$

and so on. The choice probabilities are

$$\begin{aligned}
P_{ni} =& \Pr\left(\tilde{U}_{nji} < 0, \forall j \neq i\right) \\
=& \Pr\left(\eta_1 < \frac{-\tilde{V}_{n1i}}{c_{11}}\right) \\
&\times \Pr\left(\eta_2 < \frac{-\left(\tilde{V}_{n2i} + c_{21}\eta_1\right)}{c_{22}} \middle| \eta_1 < \frac{-\tilde{V}_{n1i}}{c_{11}}\right) \\
&\times \Pr\left(\eta_3 < \frac{-\left(\tilde{V}_{n3i} + c_{31}\eta_1 + c_{32}\eta_2\right)}{c_{33}} \middle| \eta_1 < \frac{-\tilde{V}_{n1i}}{c_{11}}, \eta_2 < \frac{-\left(\tilde{V}_{n2i} + c_{21}\eta_1\right)}{c_{22}}\right) \\
&\times \cdots.
\end{aligned}$$

- The GHK simulator is calculated as follows:
  1. Calculate
  $$\Pr\left(\eta_1 < \frac{-\tilde{V}_{n1i}}{c_{11}}\right) = \Phi\left(\frac{-\tilde{V}_{n1i}}{c_{11}}\right).$$
  2. Draw a value of η_1, labeled η_1^r, from a truncated standard normal truncated at $-\tilde{V}_{1in}/c_{11}$. This draw is obtained as follows: (a) Draw a standard uniform μ_1^r. (b) Calculate $\eta_1^r = \Phi^{-1}(\mu_1^r \Phi(-\tilde{V}_{n1i}/c_{11}))$.
  3. Calculate
  $$\Pr\left(\eta_2 < \frac{-\left(\tilde{V}_{n2i} + c_{21}\eta_1\right)}{c_{22}} \middle| \eta_1 = \eta_1^r\right) = \Phi\left(\frac{-\left(\tilde{V}_{n2i} + c_{21}\eta_1^r\right)}{c_{22}}\right).$$
  4. Draw a value of η_2, label as η_2^r, from a truncated standard normal truncated at $-(\tilde{V}_{n2i} + c_{21}\eta_1^r)/c_{22}$. This draw is obtained as follows: (a) Draw a standard uniform μ_2^r. (b) Calculate $\eta_2^r = \Phi^{-1}(\mu_2^r \Phi(-(\tilde{V}_{n2i} + c_{21}\eta_1^r)/c_{22}))$.
  5. Calculate
  $$\Pr\left(\eta_3 < \frac{-\left(\tilde{V}_{n3i} + c_{31}\eta_1 + c_{32}\eta_2\right)}{c_{33}} \middle| \eta_1 = \eta_1^r, \eta_2 = \eta_2^r\right) = \Phi\left(\frac{-\left(\tilde{V}_{n3i} + c_{31}\eta_1^r + c_{32}\eta_2^r\right)}{c_{33}}\right).$$
  6. And so on for all alternatives but i.
  7. The simulated probability for this r-th draw of $\{\eta_1, \eta_2, \cdots\}$ is calculated as
  $$\begin{aligned}
  \check{P}_{ni}^r =& \Phi\left(\frac{-\tilde{V}_{n1i}}{c_{11}}\right) \\
  &\times \Phi\left(\frac{-\left(\tilde{V}_{n2i} + c_{21}\eta_1\right)}{c_{22}}\right) \\
  &\times \Phi\left(\frac{-\left(\tilde{V}_{n3i} + c_{31}\eta_1^r + c_{32}\eta_2^r\right)}{c_{33}}\right) \\
  &\times \cdots.
  \end{aligned}$$
  8. Repeat steps 1–7 many times, for $r = 1, \cdots, R$.

9. The simulated probability is

$$\check{P}_{ni} = \frac{1}{R} \sum_r \check{P}_{ni}^r.$$

Exercise: Suppose that $V_n = [1, 1.2, 1.4, 1.6, 2]'$ and $\varepsilon_n \sim N(\mathbb{0}, \Omega)$, where

$$\Omega = \begin{bmatrix} 1 & 0.1 & 0.2 & 0.3 & 0.4 \\ 0.1 & 1 & 0.1 & 0.2 & 0.3 \\ 0.2 & 0.1 & 1 & 0.1 & 0.2 \\ 0.3 & 0.2 & 0.1 & 1 & 0.1 \\ 0.4 & 0.3 & 0.2 & 0.1 & 1 \end{bmatrix}.$$

Calculate P_{ni} numerically with the GHK simulator.

PYTHON

```python
# GHK simulator

np.random.seed(123456789)
J = 5
V = np.array([1, 1.2, 1.4, 1.6, 1.8]).reshape([-1, 1])
b = np.zeros([J])
omega = np.array([[1, 0.1, 0.2, 0.3, 0.4],
                  [0.1, 1, 0.1, 0.2, 0.3],
                  [0.2, 0.1, 1, 0.1, 0.2],
                  [0.3, 0.2, 0.1, 1, 0.1],
                  [0.4, 0.3, 0.2, 0.1, 1]])
R = 99999

## Calculate V_tilde
M1 = np.array([[-1, 1, 0, 0, 0],
               [-1, 0, 1, 0, 0],
               [-1, 0, 0, 1, 0],
               [-1, 0, 0, 0, 1]])
V_tilde = M1 @ V

## Calculate omega_tilde and apply Choleski transformation to it
omega_tilde = M1 @ omega @ M1.T
L = np.linalg.cholesky(omega_tilde)

## Define a function for a single draw
def ghk_draw(V_tilde, L):
    p = np.zeros([J-1, 1])
    eta = np.zeros([J-1, 1])

    for j in range(J-1):
        p[j] = scipy.stats.norm.cdf(-(V_tilde[j] + L[j, :].reshape([1, -1]) @ eta) / L[j, j])
        eta[j] = scipy.stats.norm.ppf(np.random.uniform() * scipy.stats.norm.cdf(-(V_tilde[j] +
L[j, :].reshape([1, -1]) @ eta) / L[j, j]))

    return p.prod()

## Get the simulated probability
p_n1_hat = np.array([ghk_draw(V_tilde, L) for r in range(R)]).mean()
print(p_n1_hat)
```

```
0.06833065659353162
```

- Further, the author provides a procedure to derive SLL with the GHK simulator that ensures that (1) different covariance matrices $\tilde{\Omega}_i$ are consistent, (2) all covariance matrices $\tilde{\Omega}_j$ are positive definite, and (3) the model is normalized for scale and level of utility:
  1. Parameterize the Choleski factor of $\tilde{\Omega}_1$:

$$L_1 = \begin{bmatrix} 1 & 0 & \cdots & \cdots & \cdots & 0 \\ c_{21} & c_{22} & 0 & \cdots & \cdots & 0 \\ c_{31} & c_{32} & c_{33} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \end{bmatrix}_{(J-1)\times(J-1)},$$

     whose top-left element is set to 1. The elements c_{kl} of this Choleski factor are the parameters that are estimated in the model. Since $\tilde{\Omega}_1 = L_1 L_1'$, it ensures that $\tilde{\Omega}_1$ is positive definite.
  2. Create a $J \times J$ Choleski factor for Ω by adding a row of zeros at the top of L_1 and a column of zeros at the left:

$$L = \begin{bmatrix} 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 1 & 0 & \cdots & \cdots & \cdots & 0 \\ 0 & c_{21} & c_{22} & 0 & \cdots & \cdots & 0 \\ 0 & c_{31} & c_{32} & c_{33} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots \end{bmatrix}_{J\times J}.$$

     Then, calculate $\Omega = LL'$. With this Ω, derive $\tilde{\Omega}_i = M_i \Omega M_i'$ for any j. Further, obtain L_i from $\tilde{\Omega}_i$. This ensures that all $\tilde{\Omega}_j$ ($\forall j$) are consistent.
  3. Utility of the person who has chosen alternative i is expressed as $\tilde{U}_{ni} = \tilde{V}_{ni} + L_i \eta_n$, where η_n is a $(J-1)$-dimensional vector of i.i.d. standard normal deviates. The GHK simulator is applied to this expression.