# Further Examples

**E.g. 4** Consider the logit statistical model. The discussion of the example at the previous general note, as well as the definition of the model implies that given the parameter scenario $\theta \in \Theta$, and conditionally on $\sigma(X_n)$, $Y_{(i)} \sim Ber(q(\theta, X_i))$ with $q(\theta, X_i) = \dfrac{e^{-X_i\theta}}{1 - e^{-X_i\theta}}$, and if it is moreover assumed that $Y_n / \sigma(X_n)$ is comprised of independent variables

Thus the likelihood function is

$$l_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ln\left(\frac{e^{-x_i\theta}}{1+e^{-x_i\theta}}\right) \mathbb{1}_{\{y_i=1\}} + \frac{1}{n}\sum_{i=1}^{n} \ln\left(\frac{1}{1+e^{-x_i\theta}}\right) \mathbb{1}_{\{y_i=0\}}$$

* Notice that even though $y_n$ is conditionally on $\sigma(x_n)$ assumed independent, they are not homogeneous, as its marginals would depend on $x_i$, for any $i=1,\dots,n$. Hence the above form of the likelihood in comparison to the previous example inside the note (here due to inhomogeneity $q$ depends on $i$). If it were assumed that $y_n$

is conditionally dependent, then the likelihood function would be even more complicated, as it would also represent this conditional dependence.

The MLE is not generally analytically derivable; numerical optimization has to be performed for its derivation at the available sample, according to the algorithm that appears in the general theory. □

E.g. 5 We form an analogous analysis for the probit case.

Linewise to the previous example, we now have that $y_{(i)}/\theta(x_n) \sim Beq(q(\theta, x_i))$

where now $q(\theta, x_i) = 1 - \Phi(x_i, \theta)$

And $\Phi(y) = \int_{-\infty}^{y} \frac{1}{\sqrt{2\pi}} \exp(-x^2/2)\, dx.$

Assuming conditional independence for the elements of $y_n$ the form of the likelihood is now:

$$\ell_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ln(1 - \Phi(x_i; \theta)) \mathbb{1}_{y_{(i)}=1} +$$

$$+ \frac{1}{n} \sum_{i=1}^{n} \ln\left(\Phi(x_i\theta)\right) L_{y_i=0}$$

The MLE is likewise derivable via numerical optimization that conforms to the general optimization algorithm we have already established. Two implementations of the algorithm, one in the programming language of MATLAB and one in PYTHON follow.

Exercise: derive the analogous implementations for LOGIT.

# Matlab code for the simulation and the ML estimation of the Probit Model

```
% Probit MLE Estimation
% Simulates and evaluates the MLE for a Probit model.

clear;
clc;

% 1. Generate Simulated Data
n = 100;                    % Number of observations
X = [ones(n, 1), randn(n, 1)]; % Predictor matrix (add intercept as first column)
beta_true = [1; 2];         % True coefficients
z = X * beta_true;          % Latent variable
y = double(normcdf(z) > rand(n, 1)); % Binary outcomes using Probit

% 2. Define Log-Likelihood Function
logLikelihood = @(beta) -sum(y .* log(normcdf(X * beta)) + ...
                (1 - y) .* log(1 - normcdf(X * beta)));

% 3. Initial Guess for Coefficients
beta0 = zeros(size(X, 2), 1); % Start with zeros

% 4. Optimization to Maximize Likelihood
options = optimset('Display', 'iter', 'TolFun', 1e-6, 'TolX', 1e-6); % Options for fminunc
[beta_mle, negLogLikelihood] = fminunc(logLikelihood, beta0, options);

% 5. Display Results
disp('True Coefficients:');
disp(beta_true);

disp('Estimated Coefficients (MLE):');
disp(beta_mle);

disp('Negative Log-Likelihood at MLE:');
disp(negLogLikelihood);
```

[Simulation part: k=2]

[Estimation Part - works with arbitrary k]

↳ fmincon could have been also used to incorporate restrictions if needed

# A Python version of the previous

```
import numpy as np
from scipy.stats import norm
from scipy.optimize import minimize

# 1. Generate Simulated Data
np.random.seed(42)
n = 100                          # Number of observations
X = np.hstack([np.ones((n, 1)), np.random.randn(n, 1)])  # Add intercept as the first column
beta_true = np.array([1, 2])     # True coefficients
z = X @ beta_true                # Latent variable
y = (norm.cdf(z) > np.random.rand(n)).astype(int)  # Binary outcomes using Probit

# 2. Define the Log-Likelihood Function
def log_likelihood(beta, X, y):
    z = X @ beta
    probs = norm.cdf(z)
    # Avoid log(0) by adding a small epsilon
    log_likelihood = y * np.log(probs + 1e-10) + (1 - y) * np.log(1 - probs + 1e-10)
    return -np.sum(log_likelihood)  # Negative for minimization

# 3. Initial Guess for Coefficients
beta0 = np.zeros(X.shape[1])  # Start with zeros

# 4. Optimization to Maximize Likelihood
result = minimize(log_likelihood, beta0, args=(X, y), method='BFGS')

# 5. Extract Results
beta_mle = result.x
neg_log_likelihood = result.fun

# Display Results
print("True Coefficients:")
print(beta_true)

print("\nEstimated Coefficients (MLE):")
print(beta_mle)

print("\nNegative Log-Likelihood at MLE:")
print(neg_log_likelihood)
```

[Simulation part]

[Estimation Part]

the implementation uses the BFGS algorithm; it uses iteration scheme based on first and second derivatives.

This is illustrative; other choices are possible.