Question 2: Discrete Choice Model Estimation

Introduction

This report presents an analysis of consumer choice behavior between two chocolate snacks, Kinoko and Takenoko, employing a discrete choice model to estimate utilities and price sensitivity at a fixed consumer income level of 300 JPY.

Model Setup and Data

The dataset used records choices under various price scenarios for Kinoko and Takenoko. Consumer preferences are modeled using a multinomial logit framework, where utilities are dependent on snack pricing and a fixed income simplifies to a component of baseline utility in each choice scenario (Task 2).

Estimation Methodology

Utilities for Kinoko and Takenoko are expressed as:

- $U_{Kinoko} = \alpha_{Kinoko} + \beta \times (Income - Price_{Kinoko})$
- $U_{Takenoko} = \alpha_{Takenoko} + \beta \times (Income - Price_{Takenoko})$

Maximization of the log-likelihood function for parameter estimation utilizes the Newton-Raphson method, focusing on updating parameters through gradient and Hessian calculations.

Results and Interpretation

The final parameters after model convergence are:

- $\alpha_{Kinoko} = -4.292$
- $\alpha_{Takenoko} = -3.707$
- $\beta = 0.0388$

These parameters suggest that without an outside option, it's only possible to estimate relative differences between Kinoko and Takenoko rather than their absolute utilities, indicating preferences relative to each other (Task 1). The positive β reflects normal price sensitivity among consumers.

Conclusions

While it is technically feasible to estimate model parameters from a single choice occasion, reliability might be compromised without sufficient data variability (Task 3). Future enhancements could include modeling consumer heterogeneity through random coefficients and validating the model against market data.

Appendix

```
import pandas as pd
import numpy as np
from numpy.linalg import inv
from pathlib import Path
# from IO_ProblemSets.config import PS1

# Task 4
# Load your data
PS1 = Path(__file__).parent.resolve()
data_path = PS1.joinpath("data").resolve()
data = pd.read_csv(data_path/'data_KinokoTakenoko.csv')

# Define price settings for occasions
price_mapping = {
    'X1': (200, 200),
    'X2': (170, 200),
    'X3': (240, 200),
    'X4': (200, 250),
    'X5': (200, 180)
}

# Map the occasions to their corresponding price settings
data['kinoko_price'], data['takenoko_price'] =
    zip(*data['occasion'].map(price_mapping))

# Assume income
income = 300

# Log-likelihood function
def log_likelihood(params):
    alpha_kinoko, alpha_takenoko, beta = params
    ll = 0
    for _, row in data.iterrows():
        u_kinoko = alpha_kinoko + beta * (income - row['kinoko_price'])
        u_takenoko = alpha_takenoko + beta * (income - row['takenoko_price'])
        u_other = 0  # Baseline utility for outside option

        max_util = max(u_kinoko, u_takenoko, u_other)
        sum_exp = np.exp(u_kinoko - max_util) + np.exp(u_takenoko - max_util)
        + np.exp(u_other - max_util)

        if row['choice'] == 1:
            ll += u_kinoko - np.log(sum_exp)
        elif row['choice'] == 2:
```

```
            ll += u_takenoko - np.log(sum_exp)
        elif row['choice'] == 0:
            ll += u_other - np.log(sum_exp)
    return -ll

# Gradient of the log-likelihood function
def gradient(params):
    alpha_kinoko, alpha_takenoko, beta = params
    grad = np.zeros(3)
    for _, row in data.iterrows():
        p_kinoko = income - row['kinoko_price']
        p_takenoko = income - row['takenoko_price']

        u_kinoko = alpha_kinoko + beta * p_kinoko
        u_takenoko = alpha_takenoko + beta * p_takenoko
        u_other = 0

        denom = np.exp(u_kinoko) + np.exp(u_takenoko) + np.exp(u_other)

        p_kinoko_prob = np.exp(u_kinoko) / denom
        p_takenoko_prob = np.exp(u_takenoko) / denom
        p_other_prob = np.exp(u_other) / denom

        grad[0] += (row['choice'] == 1) - p_kinoko_prob
        grad[1] += (row['choice'] == 2) - p_takenoko_prob
        grad[2] +=
            ((row['choice'] == 1) * p_kinoko
            + (row['choice'] == 2) * p_takenoko
            - (p_kinoko_prob * p_kinoko + p_takenoko_prob * p_takenoko))
    return -grad

# Hessian of the log-likelihood function
def hessian(params):
    alpha_kinoko, alpha_takenoko, beta = params
    hess = np.zeros((3, 3))
    for _, row in data.iterrows():
        p_kinoko = income - row['kinoko_price']
        p_takenoko = income - row['takenoko_price']

        u_kinoko = alpha_kinoko + beta * p_kinoko
        u_takenoko = alpha_takenoko + beta * p_takenoko
        u_other = 0

        denom = np.exp(u_kinoko) + np.exp(u_takenoko) + np.exp(u_other)

        p_kinoko_prob = np.exp(u_kinoko) / denom
```

```
        p_takenoko_prob = np.exp(u_takenoko) / denom

        hess[0, 0] +=
            -p_kinoko_prob * (1 - p_kinoko_prob)
        hess[1, 1] +=
            -p_takenoko_prob * (1 - p_takenoko_prob)
        hess[2, 2] +=
            -(p_kinoko_prob * p_kinoko ** 2
            + p_takenoko_prob * p_takenoko ** 2)
    return -hess

# Initial guesses for parameters
params = np.array([0.0, 0.0, -0.01])

# Newton-Raphson iteration
for _ in range(1000):
    grad = gradient(params)
    hess = hessian(params)
    step = np.dot(inv(hess), grad)
    params -= step  # Update parameters
    print("Iteration: Parameters updated to", params)

print("Final estimated parameters:", params)
```