prep_problem_set_solved

June 7, 2024

[1]:
```
import numpy as np
import pandas as pd
from functools import partial
from scipy.optimize import minimize
```

1 Advanced Applied Econometrics

1.1 Static discrete choice labour supply

Stochastic discrete choice labour supply models are very popular in empirical research and have frequently been used as a tool to perform tax policy analysis. In this problem set we will examine a basic example of this class of model, and for simplicity we will abstract from any demographic heterogeneity (although it is straightforward to incorporate this).

Suppose that individuals have the choice to work $h \in [0, 10, 20, 30, 40]$ hours per week. Preferences over these discrete alternatives may be described by a parametric utility function:

$$U(c, h) = \gamma \left[\frac{c^\theta}{\theta} - \alpha\ h\right] + \varepsilon_h$$

where the state specific errors ε_h are assumed to follow a Type-I extreme value distribution.

Consumption is given by $c = y + wh$, where y is non-labour income, and w is the gross hourly wage rate generated by the following log-linear relationship: $\log w = \mu_w + \varepsilon_w$, and where the unobserved component of wages ε_w is Normally distributed with mean 0 and standard deviation σ_w.

Suppose that the parameter values are:

$\Omega = \mu_w = 1, \sigma_w = 0.55, \theta = 0.3, \alpha = 0.1, \gamma = 2$ and that $y \sim$Uniform$[10, 100]$.

With this structure simulate a dataset of 1000 observations.

1.1.1 Define functions

[2]:
```
def simulate_non_random_utility_for_hours(
    hours_options, hourly_wage_individuals, non_labor_income_individuals,␣
 ↪utility_params
):
    """Calculate utility for each hour choice option.

    Parameters
```

```
    ----------
    hours_options : np.ndarray
        Array of possible hours options. Shape is (num_hours_options,).
    hourly_wage_individuals : np.ndarray
        Array of hourly wage rates. Shape is (num_obs,).
    non_labor_income_individuals : np.ndarray
        Array of non-labor income. Shape is (num_obs,).
    utility_params : np.ndarray
        Array of utility parameters. Shape is (3,).

    Returns
    -------
    utility : np.ndarray
        Array of utility for each hour choice option. Of shape (num_obs,␣
↪num_hours_options).

    """
    # Read utility parameters
    alpha = utility_params[0]
    gamma = utility_params[1]
    theta = utility_params[2]

    # Calculate labor income for each hour choice
    labor_income = np.outer(hourly_wage_individuals, hours_options)

    # Calculate consumption for each hour choice. To add non-labor income to␣
↪each column
    # representing each hour option, we need to add a new axis to␣
↪non_labor_income,
    # which creates a column vector.
    consumption = labor_income + non_labor_income_individuals[:, np.newaxis]

    # Calculate utility for each hour choice
    utility = gamma * (consumption**theta / theta - alpha * hours_options)
    return utility
```

1.1.2 Define parameters

[3]:
```
alpha_assumed = 0.1
gamma_assumed = 2
theta_assumed = 0.3
utility_params_assumed = np.array([alpha_assumed, gamma_assumed, theta_assumed])

mean_wagerate = 1  # mu_w
sd_wagerate = 0.55  # sigma_w
```

```
num_obs = 1_000
possible_hours = np.array([0, 10, 20, 30, 40])
num_possible_hours = len(possible_hours)
```

1.1.3 Generate Utilities for each alternative and determine optimal choice

[4]:
```
# Set seed for reproducibility
seed = 123
np.random.seed(seed)

# Draw random taste shocks
taste_shocks = np.random.gumbel(loc=0, scale=1, size=(num_obs,␣
 ↪num_possible_hours))
# Draw non-labor income
non_labor_income = np.random.uniform(low=10, high=100, size=num_obs)
# Draw wage shocks
wage_shocks = np.random.normal(loc=0, scale=sd_wagerate, size=num_obs)
# Construct wage rates
wage_rate = np.exp(mean_wagerate + wage_shocks)

# Generate utility
utilities_hour_options = (
    simulate_non_random_utility_for_hours(
        hours_options=possible_hours,
        hourly_wage_individuals=wage_rate,
        utility_params=utility_params_assumed,
        non_labor_income_individuals=non_labor_income,
    )
    + taste_shocks
)

# Determine optimal hourly choice
optimal_choice = np.argmax(utilities_hour_options, axis=1)
```

1.1.4 Save data

[5]:
```
# Define DataFrame
data = pd.DataFrame(index=range(num_obs))

# Save non-labor income, choice and hours
data["non_labor_income"] = non_labor_income
data["choice"] = optimal_choice
data["hours"] = possible_hours[optimal_choice]

# Save labor income. Note for non-workers, we have zero hours and therefore␣
 ↪zero income
```

```
data["labor_income"] = data["hours"] * wage_rate
# Save hourly wage. For non-workers we do not observe wages in reality, so we␣
↪set them to NaN
data["hourly_wage"] = np.nan
data.loc[data["hours"] != 0, "hourly_wage"] = wage_rate[optimal_choice != 0]
data
```

[5]:

```
     non_labor_income  choice  hours  labor_income  hourly_wage
0           58.717484       1     10     12.993310     1.299331
1           30.266018       4     40    102.777031     2.569426
2           85.389812       3     30     93.794352     3.126478
3           62.266746       2     20     59.823773     2.991189
4           54.386252       3     30    118.754269     3.958476
..                ...     ...    ...           ...          ...
995         99.742698       1     10      8.080295     0.808030
996         77.685279       0      0      0.000000          NaN
997         53.715886       3     30     52.490830     1.749694
998         70.355164       3     30     54.564253     1.818808
999         22.221077       4     40    186.850524     4.671263

[1000 rows x 5 columns]
```

1.2 Questions

With the simulated data answer the following questions:

1. What is the distribution of work hours in your simulated dataset?

[6]:

```
# Hours in percentage
data["hours"].value_counts(normalize=True)
```

[6]:

```
30    0.241
40    0.222
20    0.190
10    0.183
0     0.164
Name: hours, dtype: float64
```

2. How does the distribution of non-labour income and wages vary with work hours?

[7]:

```
data["non_labor_income"].corr(data["hours"])
```

[7]:

```
-0.2741251064266052
```

[8]:

```
data["hourly_wage"].corr(data["hours"])
```

[8]:

```
0.41763866284569234
```

3. Write down the log-likelihood function for this model to estimate the preference parameters (α, γ, θ). Is there any problem with the likelihood function for non-workers? How would you circumvent the problem?

The likelihood function is defined as the joint probability of observing an individual's decision (in this case, hours worked), given their characteristics (such as non-labor income and wages) and the structural parameters. Assuming a Type-I extreme value distribution for taste shocks in the utility function, the probability of each decision can be calculated using a closed-form solution. Let h_i, y_i, and w_i represent individual i's choice of hours, non-labor income, and wage rate, respectively. Then, the probability of choosing h_i given a set of structural parameters θ, α, γ can be calculated as follows:

$$P(h_i|y_i, w_i, \theta, \alpha, \gamma) = \frac{exp[u(h_i, y_i, w_i, \theta, \alpha, \gamma)]}{\sum_{h \in \{0,10,20,30,40\}} exp[u(h, y_i, w_i, \theta, \alpha, \gamma)]}$$

with

$$u(h_i, y_i, w_i, \theta, \alpha, \gamma) = \gamma \left[\frac{(y_i + w_i\, h_i)^\theta}{\theta} - \alpha\ h \right]$$

The likelihood function, then is given by:

$$L(h_1, ..., h_{1000}|y_1, ..., y_{1000}, w_1, ..., w_{1000}, \theta, \alpha, \gamma) = \prod_{i=1}^{1000} P(h_i|y_i, w_i, \theta, \alpha, \gamma)$$

and the log-likelihood function, therefore by:

$$\log\ L(h_1, ..., h_{1000}|y_1, ..., y_{1000}, w_1, ..., w_{1000}, \theta, \alpha, \gamma) = \sum_{i=1}^{1000} \log\ P(h_i|y_i, w_i, \theta, \alpha, \gamma)$$

Note, that the wage rate is only observed for workers. We circumvent this problem by assuming the mean hourly wage of workers as the hypothetical wage rate for non-workers. This allows us to calculate the likelihood contributions of non-workers.

4. Write your own code to estimate the preference parameters using the dataset you generated above (which comprises work hours, non-labour income, and wages only for workers). Having estimated the model, check that the parameter estimates are close to those you used to generate the dataset.

[9]:
```
# First assign mean wage to non-workers
data.loc[data["hours"] == 0, "hourly_wage"] = data["hourly_wage"].mean()
```

[10]:
```
def log_likelihood(
    utility_params,
    hours_options,
    wage_rate_data,
```

```
    non_labor_income_data,
    optimal_choice_data,
):
    """This is the log-likelihood function calculating the negative sum of log␣
 ↪probabilities.

    Args:
        utility_params (np.array): Array of utility parameters. Shape is (3,)  ␣
 ↪.
        hours_options (np.array): Array of possible hours options. Shape is␣
 ↪(num_hours_options,).
        wage_rate_data (np.array): Array of wage rates. Shape is (num_obs,).
        non_labor_income_data (np.array): Array of non-labor income. Shape is␣
 ↪(num_obs,).
        optimal_choice_data (np.array): Array of optimal choices. Shape is␣
 ↪(num_obs,).


    Returns:
        float: Negative sum of log probabilities.
    """
    # Calculate utilities for each option
    utilities = simulate_non_random_utility_for_hours(
        hours_options=hours_options,
        hourly_wage_individuals=wage_rate_data,
        utility_params=utility_params,
        non_labor_income_individuals=non_labor_income_data,
    )
    # Create scale to avoid overflow. The rescaling cancels for the
    # choice probabilities.
    scale = np.max(utilities, axis=1)[:, np.newaxis]
    # Exponentiate utilities
    exp_utilities = np.exp(utilities - scale)
    # Sum over colmuns and create column vector
    sum_exp_utilities = exp_utilities.sum(axis=1)[:, np.newaxis]

    # Calculate choice probabilities
    choice_probs = exp_utilities / sum_exp_utilities
    # Select choice probabilities for optimal choices in data
    selected_choice_probs = np.array(
        [choice_probs[row, col] for row, col in enumerate(optimal_choice_data)]
    )

    # Sum over log probabilities
    sum_log_robs = np.sum(np.log(selected_choice_probs))

    # Return negative sum for minimization
```

```python
    return -sum_log_robs
```

[11]:
```python
# Fix all other arguments except utility_params
partial_loglike = partial(
    log_likelihood,
    hours_options=possible_hours,
    wage_rate_data=wage_rate,
    non_labor_income_data=data["non_labor_income"].values,
    optimal_choice_data=data["choice"].values,
)
```

[12]:
```python
minimize(
    partial_loglike,
    x0=np.array([0.5, 2, 0.5]),
    bounds=[(1e-6, 1), (1, 5), (1e-6, 1)],
    method="L-BFGS-B",
)
```

[12]:
```
  message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  success: True
   status: 0
      fun: 1335.6522421736058
        x: [ 1.054e-01  1.774e+00  3.097e-01]
      nit: 21
      jac: [-1.656e-01  1.614e-03  1.070e-01]
     nfev: 104
     njev: 26
 hess_inv: <3x3 LbfgsInvHessProduct with dtype=float64>
```