Written Exam Economics summer 2021

Introduction to Programming and Numerical Analysis

From May 28th 10.00 AM to May 30th 10.00 AM

This exam question consists of 1 pages in total

Answers only in English.

A take-home exam paper cannot exceed 10 pages – and one page is defined as 2400 keystrokes

In addition to the Jupyter Notebook containing your exam answers, you must hand in your portfolio of projects completed during the semester. Therefore, place your exam notebook, together with all accompanying files, in a folder in your local git repository. Zip the whole repository into 1 file. Name the file according to your group name (eg. 'myGroup.zip') and upload this file to Digital Exam.
Furthermore: Write which groups you have given peer feed back to (and for which projects) in the main README.md file of your repository. Also write the names of all group members. (The main README is located together with the 3 projects folders, the .gitignore and the LICENSE file.)

In [1]:

```python
import numpy as np
from types import SimpleNamespace
%load_ext autoreload
%autoreload 2

%matplotlib inline
import matplotlib.pyplot as plt
plt.style.use('seaborn-whitegrid')

# Import additional libraries:
```

The logit model

In the following, we will consider the logit model for a binary discrete choice. That is, an agent is either taking a specific action, or not taking it (Think buying a car, exit the labor market etc).

We imagine that the benefit of taking the action in question is described by a linear utility index y_i^*. This depends on two exogenous variables x_1 and x_2 and a random shock ϵ:

$$
\begin{aligned}
y_i^* =& \beta_0 + \beta_1 x_i^1 + \beta_2 x_i^2 + \epsilon_i \\
=& x_i \beta + \epsilon_i \\
& \epsilon \sim logistic(0,1)
\end{aligned}
$$

The econometrician does not observe the utility index; only the *actual choice* based on the index is observed. We therefore associate the indicator variable y_i with the choice taken by individual i

$$
\begin{aligned}
y_i = 1 \Leftrightarrow y^* > 0 \Leftrightarrow \text{Choice is taken} \\
y_i = 0 \Leftrightarrow y^* \leq 0 \Leftrightarrow \text{Choice is not taken}
\end{aligned}
$$

Because we assume that the utility shocks follow a logistic distribution, we can formulate the **probability** that an individual chooses to take the action by

$$
\begin{aligned}
P(y_i = 1 | x_i; \beta) &= \frac{\exp(x_i \beta)}{1 + \exp(x_i \beta)} \\
P(y_i = 0 | x_i; \beta) &= 1 - P(y_i = 1 | x_i; \beta)
\end{aligned}
$$

We can now use the formulation of choice probabilities to estimate the parameters β by maximum likelihood. That is, we write up the log-likelihood function

$$
LL(\beta) = \sum_{i=1}^{N} y_i \log(P(y_i = 1 | x_i; \beta)) + (1 - y_i) \log(1 - P(y_i = 1 | x_i; \beta)) \tag{1}
$$

Maximizing $LL(\beta)$ with respect to β yields the estimated parameters $\hat{\beta}$

$$
\hat{\beta} = \arg\max_{\beta} LL(\beta)
$$

The function `DGP()` will create the N observations of (y_i, x_i):

In [2]:

```python
def DGP(mp):
    ''' The data generating process behind binary choice model

    Args:
        mp (SimpleNamespace): object containing parameters for data generation

    Returns:
        y_obs (ndarray): indicator for binary choices made by individuals
        x_obs (ndarray): independent variables

    '''

    # a. Exogenous variables
    x0 = np.tile(1.0, mp.N)
    x1 = np.random.normal(**mp.x1_distr)
    x2 = np.random.normal(**mp.x2_distr)
    x_obs = np.vstack((x0, x1, x2)).T

    # b. Probabilities of action choice
    y_prb = np.exp(x_obs @ mp.beta) / (1 + np.exp(x_obs @ mp.beta))

    # c. Draw binary choices from the binomial distribution
    y_obs = np.random.binomial(1, y_prb)
    return y_obs, x_obs
```

Create your data using the following parameterization:

In [3]:

```python
# Parameters
mp = SimpleNamespace()
mp.beta = np.array([0.15, 0.1, 0.2])
mp.N = 100_000
mp.x1_distr = {'loc': 4, 'scale': 3, 'size': mp.N}
mp.x2_distr = {'loc': 1, 'scale': 0.5, 'size': mp.N}

# Create data
np.random.seed(2021)
y_obs, x_obs = DGP(mp)
```

Question 1: Create a function that calculates the log-likelihood of your data based on a β. That is, the function must take as arguments an array `beta`, `y_obs` and `x_obs`

In [ ]:

```python
# Example
def log_likelihood(beta, y_obs, x_obs):
    pass
```

Question 2: Make a 3d-plot of the likelihood function where β_1 and β_2 are on the horizontal axes, and the log-likelihood is on the vertical axis. Visually confirm that it peaks at the data generating β_1 and β_2.

Note: You can let β_0= `mp.beta[0]`. Make sure that `mp.beta[1]` and `mp.beta[2]` are in the grids over β_1 and β_2.

Question 3: Estimate β by maximum likelihood. You may use a gradient-free approach or gradients if you will.

Question 4: Based on your estimated parameters, simulate a choice `y_sim` pr individual in `x_obs`.
Create an output table that shows the following 4 statistics:
The number of times where:

- `y_obs` = 1 and `y_sim` = 1
- `y_obs` = 1 and `y_sim` = 0
- `y_obs` = 0 and `y_sim` = 1
- `y_obs` = 0 and `y_sim` = 0

Comment on the distribution of occurances across cells in the table.

Question 5: Test if your initial guess of β will have an impact on the final estimate. Why do you think there is/is not an impact?

Consumption saving with borrowing

We are now considering the consumption-savings model with an extension: households may **borrow** money in the first period. Additionally, there are **2 kinds** of households: the first type will likely see a **low level** of period 2 income, whereas the second type will likely see a **high** second period income.

A household lives for 2 periods and makes decisions on consumption and saving in each period.

Second period:
Solving the consumer problem in the second period is similar to the baseline case we have seen before.

The household gets utility from **consuming** and **leaving a bequest** (warm glow),

$$
\begin{aligned}
v_2(m_2) &= \max_{c_2} \frac{c_2^{1-\rho}}{1-\rho} + \nu \frac{(a_2 + \kappa)^{1-\rho}}{1-\rho} \\
& \text{s.t.} \\
& a_2 = m_2 - c_2 \\
& a_2 \geq 0
\end{aligned}
$$

where

- m_t is cash-on-hand
- c_t is consumption
- a_t is end-of-period assets
- $\rho > 1$ is the risk aversion coefficient
- $\nu > 0$ is the strength of the bequest motive
- $\kappa > 0$ is the degree of luxuriousness in the bequest motive
- $a_2 \geq 0$ ensures the household *cannot* die in debt

First period:
The household gets utility from immediate consumption. Household takes into account that next period income is stochastic.

$$
\begin{aligned}
v_1(m_1) &= \max_{c_1} \frac{c_1^{1-\rho}}{1-\rho} + \beta \mathbb{E}_1 \left[v_2(m_2) \right] \\
& \text{s.t.} \\
a_1 &= m_1 - c_1 \\
m_2 &= (1+r) a_1 + y_2 \\
y_2 &= \begin{cases} 1 - \Delta & \text{with prob. } P_{low} \\ 1 + \Delta & \text{with prob. } P_{high} \end{cases} \\
a_1 &> -\frac{1-\Delta}{1+r}
\end{aligned}
$$

where

- $\beta > 0$ is the discount factor
- $\mathbb{E}_1$ is the expectation operator conditional on information in period 1
- y_2 is income in period 2
- $\Delta \in (0,1)$ is the level of income risk
- r is the interest rate
- $\frac{1-\Delta}{1+r} > c_1 - m_1$ ensures the household cannot borrow *more* than it will be able to repay in next period when y_2 is received.

The **2 types** of households are defined by their different (P_{low}, P_{high}):

- Type 1:
  - $P_{low} = 0.9$
  - $P_{high} = 0.1$
- Type 2:
  - $P_{low} = 0.1$
  - $P_{high} = 0.9$

In [82]:

```python
# Parameters
rho = 3
kappa = 0.5
nu = 0.1
r = 0.04
beta = 0.95
Delta = 0.5
# Add income prb parameters

# Tip: for each household type, create a SimpleNamespace
# or dictionary for storing all the parameters
```

Question 1 Solve the model for each type of household. Plot the value functions $v_1(m_1)$ and $v_2(m_2)$ in one graph for each household type. Comment on the differences.

Question 2 From the model solution, obtain the optimal consumption functions $c_1^*(m_1)$ and $c_2^*(m_2)$. Plot these in one graph for each type of household. Comment on the observed differences between household types.

Question 3 Simulate `simN` households of each type based on the distribution of m_1 below. You can use the same distribution for both household types. What is the fraction of households who *borrow* in period 1, $c_1 > m_1$, in each group?

In [ ]:

```python
np.random.seed(2021)
simN = 1000
# No one gets negative m in first period
sim_m1 = np.fmax(np.random.normal(1, 1, size = simN), 0)
```

Division by Newton's method

One can obtain the numerical ratio of 2 real numbers **using only multiplication** and harnessing Newton's method! This may be helpful when the numbers are very large because division methods of large numbers is costly.

Our objective is to find the numerical x

$$
x = \frac{n}{d}
$$

given the two numbers n, d.

First note that if we can find the numeric value $\tilde{d}$

$$
\tilde{d} = \frac{1}{d}
$$

then we can readily obtain x by

$$
x = n \times \tilde{d}
$$

Therefore, our objective comes down to finding the value of $\tilde{d}$ and the rest is trivial.

Second, note that Newton's method can be used to find the root x^* of a function $f(x)$ by the iteration steps

$$
x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \equiv \mathcal{N}(x_k)
$$

This means that if we can define some function $f(x)$ such that

$$
f(x) = 0 \Leftrightarrow x = \frac{1}{d}
$$

then the root x^* provides us with the numerical value that we want.

Third, note that the function $g(x)$

$$
g(x) = \frac{1}{x} - d
$$

has the property $g(\tilde{d}) = 0$, which means that $g(x)$ is a good candidate for $f(x)$.

Question 1: By applying the function $g(x)$ in Newton's method, we can avoid any use of division during the run of the algorithm.
Derive the expression $\frac{g(x)}{g'(x)}$. Do you see why there is no division involved?

Question 2: Implement the algorithm below in code and test it.

Division algorithm

1. Choose a tolerance level $\epsilon > 0$. Provide an initial guess $\tilde{d}_0$. Set $k = 0$.
2. Calculate $g(\tilde{d}_k)$.
3. If $|g(\tilde{d}_k)| < \epsilon$ then stop and return $x = n \times \tilde{d}_k$.
4. Calculate a new candidate root $\tilde{d}_{k+1} = \mathcal{N}(\tilde{d}_k)$.
5. Set $k = k + 1$ and return to step 2.

Important: if the starting point $\tilde{d}_0$ is too far off target, then you might not get convergence.
You can test your implementation with the example:
n = 37.581
d = 5.9
$\tilde{d}_0$ = 0.2

In [ ]:

```python
def newton_division(n, d, d0, max_iter=500, tol=1e-8):
    pass
```