TILBURG UNIVERSITY

35V3A1: Computational Aspects in Econometrics
2023-2024
Pieter Kleer
P.S.KLEER@TILBURGUNIVERSITY.EDU

Python II Assignment

Deadline: Wednesday, November 22, 23:59 (CET), 2023

General instructions

- This is group assignment and all group members are expected to contribute to the same degree. Group members are the ONLY ones that contribute to the assignment.[1] Every goup member should be able to explain what was handed in upon request.
- You are allowed to use any module from Python's Standard Library as well as the *NumPy*, *SciPy*, and *Matplotlib* packages and their dependencies. No other publicly available packages or codes are allowed (unless mentioned explicitly or used in the course materials).
- Apart from correctly solving the problems, your submission is also assessed on other criteria, such as efficiency, hard coding, DRY, single responsibility, coding style & documentation, KISS. See the introductory slides of the Python II module. As a rule of thumb, the more you exploit the functionality as explained in the theory of the Python II module, the higher your grade. In particular, avoid for loops whenever possible.[2]
- Always add titles, labels, and/or legends to your figures to identify the figure and its contents. Add the names of the team members at the top of the files.
- If you have questions about the assignment, ask them in class or send me an e-mail.

What to hand in?

- A short report **(PDF)** in which you explain your solutions and a Python file with your code. For both there is a **mandatory template** on Canvas (replace `[group_number]` by your group number in both).
  - If you are asked to produce figures, these have to be included in the report as well (apart from being outputted by your Python code).
  - Give a description of every solution in the report to illustrate you understood what you implemented. **No explanation = no points.** You should also add some comments to the Python file, so that its structure is clear, but use the report for a more elaborate explanation.
  - Give a short summary in the report of who did what for this assignment.
- All requested documents have to be handed in before the deadline mentioned above. Late submissions automically get grade 1.0 (no discussion possible).

[1] Aids like ChatGPT are forbidden. Fraud suspicions in this or other regards are sent to the Examination Board.
[2] Solutions without for-loops are not *always* faster, but for now the goal is to have you think about such solutions.

1 Pandora's box problem

In Pandora's box problem we are given n boxes $\{1,\ldots,n\}$, each equipped with a probability distribution $\mathcal{F}_i$ for $i = 1,\ldots,n$ that are known to us. Opening box i costs c_i for $i = 1,\ldots,n$, after which its reward v_i is revealed. The reward v_i is the realization of a random variable $X_i \sim \mathcal{F}_i$. You can open as many boxes as you want. After having opened a box, you can decide to continue with opening another box, or you keep the best reward that you have seen so far among all the opened boxes (after this the process stops and we are finished).

We need to choose an algorithm (ALG) that specifies which boxes to open, and when to stop opening boxes and select the highest reward seen so far. Given that a subset $S \subset \{1,\ldots,n\}$ of boxes has been opened, the decision on whether or not to open another box can be based on the distributions $\mathcal{F}_1,\ldots,\mathcal{F}_n$, costs $c_1,\ldots,c_n$ and the realized rewards form the boxes in S that we have seen.

The goal is to maximize the expected utility of the chosen box (containing the highest reward seen) minus the total costs of all the opened boxes. More precisely, if $O(ALG)$ is the set of opened boxes by ALG, for realized rewards $(v_1,\ldots,v_n)$, then the utility of ALG is

$$u_{\text{ALG}}(v_1,\ldots,v_n) = \max_{i\in \text{O(ALG)}} v_i - \sum_{i\in \text{O(ALG)}} c_i. \tag{1}$$

Note that only the rewards from the opened boxes are used to calculate the utility. Given that v_i is a realization of random variable X_i for all i, the *expected utility* is given as

$$\mathbb{E}_{X_i\sim\mathcal{F}_i}[u_{\text{ALG}}(X_1,\ldots,X_n)]. \tag{2}$$

We will only be approximating the expected utility for a given ALG, which is done as follows for a given integer T:

1) Generate for $j \in \{1,\ldots,T\}$ a vector $v^{(j)} = (v_{1j},\ldots,v_{nj})$ where v_{ij} is a realization of a random variable $X_i \sim \mathcal{F}_i$ for $i = 1,\ldots,n$.
2) Compute for every vector $v^{(j)}$ the utility under ALG using (1).
3) Compute the average utility of all T input vectors $v^{(j)}$ by

$$\frac{1}{T}\sum_{j=1}^{T} u_{\text{ALG}}(v^{(j)}). \tag{3}$$

If we let $T \to \infty$, then this approximation should converge to the expression in (2). In our case, we will choose T to be a large finite number.

1.1 Weitzman's algorithm

The optimal algorithm for solving Pandora's box problem is given by steps i)-iv) below for a vector of realized rewards $(v_1,\ldots,v_n)$.

i) Compute for every box the *reservation price* τ_i, which is the value of y that solves the equation

$$\mathbb{E}_{X_i\sim\mathcal{F}_i}[\max(X_i - y, 0)] = c_i. \tag{4}$$

ii) Open the boxes in decreasing order of reservation price: First open the box with the highest τ_i, then the second highest, etc...

iii) Open boxes according to the order of ii), until you arrive at a box for which $v_i \geq \tau_i$.

iv) At that point, stop opening boxes and select the highest reward seen so far from all the opened boxes.

We will implement this algorithm in the following questions. We first focus on discrete distributions, for a given nonnegative support $\{a_1,\ldots,a_q\}$, with $0 \leq a_1 \leq a_2 \leq \cdots \leq a_q$, with correponding probabilities $\{p_1,\ldots,p_q\}$. That is, the discrete distribution is given by

$$\mathbb{P}(X = a_k) = p_k \text{ for } k = 1,\ldots,q.$$

In the questions a)-b) below, we will use so-called *discrete distribution matrices* $A,P \in \mathbb{R}^{n\times q}_{\geq 0}$ to represent n different discrete distributions, all supported on q (possibly different) points. That is, every combination (a^i,p^i) of rows i from A and P represent one discrete distribution, where a^i contains the supports points, and p^i the corresponding probabilities.[3]

a) **[3 points]** Write a function that takes as input a value y, and discrete distribution matrices $A,P \in \mathbb{R}^{n\times q}_{\geq 0}$, and outputs for $i \in \{1,\ldots,n\}$ the quantity $\mathbb{E}[\max(X_i - y,0)]$, i.e., for every row combination of A and P.

b) **[5 points]** Based on your function in part a), write a function that takes as input discrete distribution matrices A,P and (column) vector $c = [c_1,\ldots,c_n]'$, and outputs for $i \in \{1,\ldots,n\}$ the solution to (4) That is, it outputs the solution to (4) for every combination (a^i,p^i,c_i).
You must use Brent's method in your function (think of an appropriate bracket choice!).

In part c), we will write a similar function as in part b), but now for a continuous random variable, which in Python are modelled as objects in SCIPY.STATS.RV_CONTINUOUS (see documentation and lecture materials).

c) **[6 points]** Write a function that takes as input a (real-valued) continuous distribution $\mathcal{F}$ and cost $c \in \mathbb{R}_{\geq 0}$ and outputs the solution y to

$$\int_{-\infty}^{\infty} \max(x-y,0) f(x) \mathrm{dx} = c \tag{5}$$

where f is the probability density function of $\mathcal{F}$. This equation is the interpretation of (4) for continuous distributions.
Read the SCIPY.INTREGRATE *documentation and apply it correctly to solve this question. Again, use Brent's method with an appropriate bracket choice.*

d) **[1 point]** Execute your function in part c) for five normal distributions with mean and standard deviation pairs (μ_i,σ_i) for $i = 1,\ldots,5$, where $\mu = [4,6,2,7,10]$ and $\sigma = [1,2,5,7,11]$. The cost vector is $c = [1,1,1,1,1]$.

[3] Tip: Start by writing your code for one distribution, and then generalize it to discrete distribution matrices.

We will next implement steps ii)-iv) of Weitzman's algorithm.

e) **[6 points]** Write a function that takes as input a vector (representing the reservation prices) $\tau = [\tau_1, \ldots, \tau_n]$, cost vector $c = [c_1, \ldots, c_n]$, and a vector of values $[v_1, \ldots, v_n]$. It should execute steps ii)-iv) of Weitzman's algorithm, and outputs the utility in (1) and the index of a box containing the chosen reward.

f) **[1 point]** Run Weitzman's algorithm as in i)-iv) on the discrete distribution matrices A, P, cost vector c and reward vector v as in the template. Do this by combining your functions in part b) and e).

Next, we will be running Weitzman's algorithm on many randomly generated reward vectors, in order to empirically approximate the expected performance in (2). We will focus on discrete distributions that have (common) support $\{1, 2, 3, \ldots, K\}$.

g) **[3 points]** Write a function that takes as input a matrix $P \in [0,1]^{n \times K}$ of which every row corresponds to a probability distribution $p = [p_1, \ldots, p_K]$ with support $\{1, \ldots, K\}$. The function should output a randomly generated number from every one of the n distributions corresponding to the rows of P.
As a source of randomness, only the function NUMPY.RANDOM.RAND() *is allowed here.*

h) **[3 points]** Using the input from the template (distribution matrix P and given cost vector c), approximate the empircal average in (2) for $T = 100.000$ (hundred thousand), by implementing steps 1)-3) from the introduction, using your functions in parts b), e) and g).

2 Sparse vector approximation

In this exercise, we will look the problem of sparse vector approximation.[4] We consider the following setting: We are given an $m \times n$ matrix $A \in [0,1]^{m \times n}$, a non-negative column vector $x = (x_1, \ldots, x_n) \in [0,1]^n$ with $\sum_i x_i = 1$, and a non-negative row vector $y = (y_1, \ldots, y_m) \in [0,1]^m$ with $\sum_j y_j = 1$.

We can interpret x and y as discrete probability distributions over the column indices $\{1, \ldots, n\}$ and row indices $\{1, \ldots, m\}$ of the matrix A, respectively. That is, we have a random variable X that samples column index i with prob. x_i for $i = 1, \ldots, n$, and a random variable Y that samples row index j with prob. y_j for $j = 1, \ldots, m$.

i) **[2 points]** Write a function that, for given input vector x, returns a column index sampled according to X.
As a source of randomness, only the function NUMPY.RANDOM.RAND() *is allowed here. This question has some similarities with g).*

[4] This problem has many applications for example in learning theory and in the computation of Nash equilibria in game theory.

The idea of sparse vector appoximation is to sample a number of column indices $c_1, \ldots, c_K \in \{0, 1, \ldots, n\}$ (i.e., K realizations of the random variable X), and row indices $r_1, \ldots, r_L \in \{0, 1 \ldots, m\}$ (i.e., L realizations of the random variable Y). Note that the c_i do not have to be distinct! The same holds for the r_j. We then consider, for $\ell = 1, \ldots, L$ and $k = 1, \ldots, K$ the absolute difference

$$|yAx - y^{(\ell)} A x^{(k)}| \tag{6}$$

where

$$x^{(k)} = \frac{1}{k} \sum_{i=1}^{k} e^{c_i} \quad \text{and} \quad y^{(\ell)} = \frac{1}{\ell} \sum_{i=1}^{\ell} (f^{r_i})^T \tag{7}$$

with $e^j \in \{0,1\}^n$ the unit column vector given by

$$e_i^j = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

for $j = 1, \ldots, n$, and similarly $(f^{r_i})^T \in \{0,1\}^m$ the row vector with a 1 in position j and zeros elsewhere. Sparse vector approximation now means that the expression in (6) for $k = K$ and $\ell = L$ converges to zero as K and L grow large, and this happens already for relatively small values (compared to m and n). Informally speaking, $y^{(L)} A x^{(K)}$ serves as a good approximation to yAx. We will verify this numerically.

j) **[4 points]** Write a function that takes as input an integer n and K given columns indices $c_1, \ldots, c_K \in \{0, \ldots, n\}$, and returns $x^{(1)}, \ldots, x^{(K)}$ as in (7).

k) **[3 points]** Write a function that takes as input a column vector x, row vector y and matrix A, as well as parameters K and L, and that outputs an $L \times K$ matrix with the difference in (6) for all combinations (ℓ, k) with $\ell = 1, \ldots, L$ and $k = 1, \ldots, K$.

ℓ) **[3 points]** Take $n = 100$ and $x = y = \frac{1}{n}(1, 1, \ldots, 1)$. Generate resp. $m = K = L = 1000$ samples of x and y (corresponding to $c_1, \ldots, c_{1000}$ and $r_1, \ldots, r_{1000}$). Take A a randomly generated $n \times n$ matrix with entries in $[0, 1]$. Plot for $q = 1, \ldots, m$, the difference in (6) for $\ell = k = q$.