Python II assignment

Deadline: Wednesday, November 22, 2023, 23.59 (CET).

1 Pandora's box problem

a) **[3 points]** We calculated the expected values of discrete distributions by using the properties of arrays that they share with matrices and combining that with basic probability knowledge.

b) **[5 points]** We calculated the solutions to the required equation using the brent method. Using the map function we managed to return values to a system of exuations without any loops. Another option of solving that equation would've been using the minimise least squares method, but since it didn't contain explicit brackets and only implicitly used brent, we only used it to check our results. As for the choice of brackets, we picked optimal ones according to the following criteria: we used a bracket starting from 0 because any root "y" must be larger than 0 as c is always positive; we used a bracket ending at [c-max(Xi)] for any specific distribution since, if the E(Xi) is "to small to reach c" even when y=0, we will need a negative y, of the size that is missing between E(Xi) and c. Hence, worst case scenario, -(max(Xi)-c)=-max(Xi)+c.

c) **[6 points]**

First step: getting pdf of the inputted distribution distribution.

Second step: defining the full function that's has to be integrated.

Third step: integral of the function and subtracting c so it is =0 we can determine y which is the objective of the question. In this step as pdfs are not generally the same for whole R we need to take a look where pdf is a function and not 0 there is no point integrating multiplication with 0 ,so I define a the bounds of an integral from the support of the pdf. This bounds also are limited using the percent-point function and inverse survival function because we know that over these bounds there is a 0.1 percent chance that the pdf has an actual value if this step is excluded results are nonsense.Then I integrating with respect to x using examples from scipy.integrat forum.

Forth step: aplying brentq method to determine y as instructed in the exercise. For this step since we are working with integration of a pdf which is essentially cdf we can use the functions of precent-point function and inverse survival function to get general bounds to find y. These bounds seem to work good if the c is not more than 1500 times larger than the mean of the distribution, and the mean and std are in logical proportions, so i would say that it is a success with choosing them. And then when brackets are chosen the only thing left is solving the equations respect to y using brents method as instructed.

d) **[1 point]** I used combination of lamda and map so we can efficiently get the values for each mean , standard deviation and cost, this function should work for any normal distribution. At first I ran in to a problem with printing the result but if you convert it to a list the result prints out fine.

e) **[6 points]** For this exercise I just did the steps given in the exercise. First I combined the 3 inputs for each box so we can implement the calculations, and compere the boxes. Then I sorted boxes by reservation price this is step ii) of weitzmans , selecting all rows but only first column which is reservation price so we can sort by it , index 0 smallest reservation price so needed to reverse so we can 'open' box wit the highest reservation price first. This step is needed for later step where I extract the index of the chosen box. Then I sorted the actual box matrix with the indexes just obtained. Then I found all boxes where reward is higher equal the reservation price to see if any boxes meet the criterion iii) of weitzmans. Got the indexes of boxes which have reward higher equal to reservation price. Then for the next part I separated the function in 2 parts with an if statement. If there were no box that met the criterion iii) then we would open all boxes then we calculate the cost of it and pick the box with highest reward. If we have cost and reward then we can calculate utility. And knowing that in this case we open all boxes we know that the index of box we pick is just the index of a box with highest reward so we can simply use argmax function. The second case is where the criterion is met so then we do general steps at first. We get the cost of opining a box at each step so we know how much is to open first n boxes in the actual opening order. Get reward at a step so we can easily access the rewards of each box in the opening order. Then we select the box which first appears and meets the criterion iii) . Then of all opened boxes we select the one with highest reward. Then using previous step of opening costs we get the total cost of opening boxes until the criterion is met. Now we can easily calculate the utility. And now using the index list we obtained in on of the first steps i can easily access the index of selected box from the original matrix. Now when everything is obtained I returned the results that were asked for, the utility and the index of selected box.

f) **[1 point]** In question e we provided the vector of reservation prices, but now we call function of b to return the reservation prices. Then call function e using the output of b, cost vector c and value vector v.

g) **[3 points]** First we generate a random value between 0 and 1 that can be viewed as a probability. Using the input pdf vector P we sum its entries row wise such that the x[i]=P[i]+P[i-1]+...+P[0] where i represents the i-th element of a certain row of the matrix. This creates a cdf matrix. Then use the generated value pgen to find X for which P(pgen¡X) for each row. This value X is returned in a vector which consists of solutions for each row.

h) **[3 points]** Usign fucntion of b we once again attain the needed reservation prices. We create a loop which takes place T times, which can be specified. In this loop a value vector v is generated using the function of g. This vector then is used to calculate utility using e, then this is added to a variable which is later used to calculate the mean utility after T itterations.

2 Sparse vector approximation

i) **[2 points]**
In this question, we had to create a function, which for given input vector x, returns a column index sampled according to X. In the function, we first calculate cumulative distribution function using the $np.cumsum$ function. Then, we get a random number in range 0-1 using $np.random.rand$. Finally we return the argument (a column index) for which the CDF captures the random value. We get the column index from the $np.argmax$ with a condition $cdf >= random\_no$.

j) **[4 points]**
In part k, we have to write a function that takes as input an integer n and K given columns indices stored in a vector *ck* and return $x^{(i)}$ with *i*=0,1,...,K - so the function has 2 inputs. We first create the variable *k* which is equal the length of the vector *ck*. Then a 0-entry matrix in created with size *n* rows by *k* columns using $np.zeros$. A variable *rrange*, which is an integer sequence of k numbers (0,1,2,...,k-1), is created in order apply the matrix calculations in the later step. Later, we insert 1 in the j^{th} row of the matrix and the indexed column from the vector *ck*. Then, using $np.cumsum$ we calculate a cumulative sum per each row (axis=1). The results are stored in an array. We later apply the formula from the assignment instructions and return the x^k according to the formula.

k) **[3 points]**
In part k, we had to write a function that outputs an $L \times K$ matrix with the difference in $|yAx - y^l Ax^k|$ for all combinations (l,k). The function has 5 inputs, the vectors x and y, matrix A and parameters K and L. We first calculate variables m, n, which are the length of corresponding x, y vectors - we did it by applying $np.size$ function. We later create a vector of L (K) realisations of the random variable Y (X) (sample of row (column) indices), by using the function defined in part i ($return\_column\_index$). We later create the approximation vectors using the $return\_xk$ function (question 2j) with the size corresponding to matching the future matrix calculations. For the approximation of row vectors, we apply $np.tranpose$ in order to be able to use matrix calculations. We then separately apply the matrix multiplication on the 'real' vectors and its sparse approximation in order to later take the absolute difference between the two, which we return.

l) **[3 points]**
In question l, we are putting the newly defined functions into use. We have to calculate and plot the difference between the diagonal elements (k=l=q) of the randomly generated $n \times n$ matrix A. We plot the scatter graph using $matplotlib.pyplot$. One can spot that for almost all diagonal entries, the difference is almost 0 (with a exception of a few low entries).

3 Who did what?

Dans Lismanis - question 1 f,g,h
Nikodem Baehr - question 2
Rudolfs Jansons - question 1 c,d,e.
Andrei Agapie - question 1 a,b.

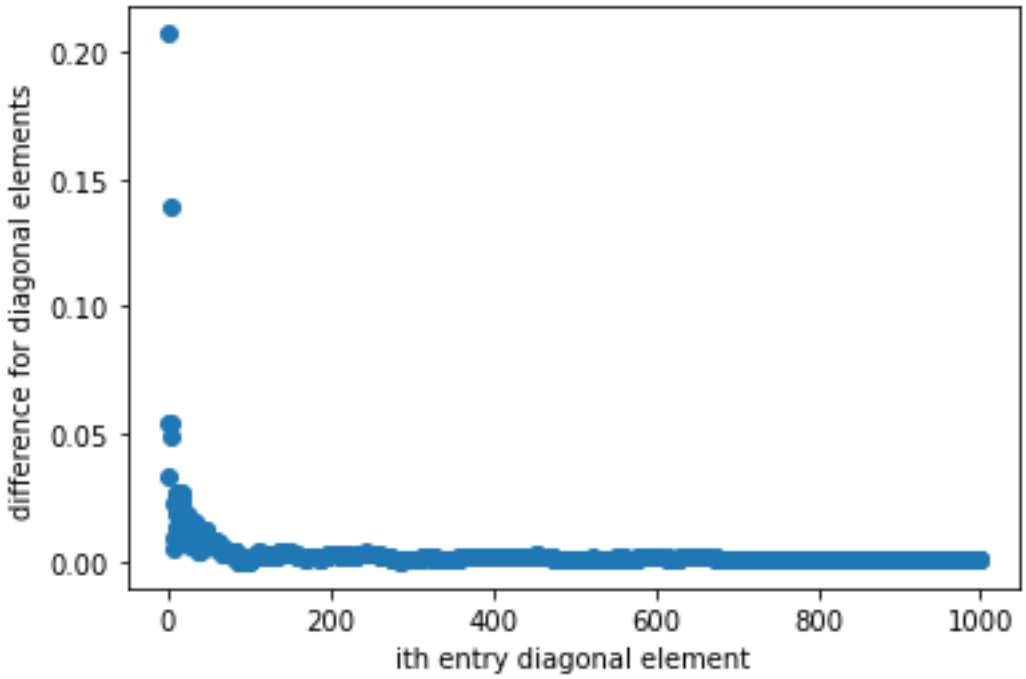

Figure 1: Scatter plot of the absolute difference of diagonal entries