Rosie Manoli
Course: Decision Support Systems
Simulation Assignment on Decision-Making Systems in Octave

Introduction:
This assignment focuses on the simulation of Decision-Making Systems using Octave, based on a real-life scenario. The objective was to analyze and model a practical decision-making process, utilizing computational methods to evaluate possible outcomes. The project was conducted as a team assignment, where I collaborated with two colleagues to develop and implement the simulation. The code will be provided in the repository.

Part A:
(a). Our company is engaged in the manufacturing and repair of computers and other products. At this time, it aims to improve the software of the HR module of its ERP system to ensure better employee management and enhance the efficiency and coordination of the subsystem.
(b). To achieve this, the company is looking for new, modern software that fits its system. For this reason, it sets specific criteria and factors and the software options will be evaluated based on them. The criteria are as follows:

1. The cost of the software and its maintenance
2. The user training time for the new software, specifically the number of hours for training and usage
3. The number of departments compatible with the new software

The method chosen for making the decision is MAUT (Multi-Attribute Utility Theory)
(c). There are three suppliers for the selection of the HR software, and the software options meet the above criteria. The alternatives are HR Software 1, HR Software 2, and HR Software 3.

Criteria	**Alternatives**		
	HR Software 1	HR Software 2	HR Software 3
Cost	5.000€	2.500€	7.000€
User Training Time	20 hours	10 hours	15 hours
Number of Departments	5	3	8

For the correct selection of the software, participation from various departments of the company will be involved. Initially, the Director of Human Resources and some members of the subsystem team will participate, as they are the ones who know what is needed and how the subsystem works. Additionally, for decision-making, the IT Director and their IT team are essential due to the technical knowledge they can provide, aimed at evaluating and ensuring the smooth integration of the new software. Furthermore, the Financial Director will examine the cost of the new software and whether it is a sound and beneficial investment for the company. Finally, the end users will assess the functionality and performance of the new software, and the CEO will make the final decision for the company.

Part B:

1. Initially, for the creation of the Pairwise Comparison:

As seen from the code in the repository, we created a function called pairwise_MAUT, which takes a matrix S for the combinations of criteria and n, which is set to three as defined in the first part. By randomly selecting from matrix S, the upper triangular part of matrix P is filled, while the lower triangular part of matrix P is computed by inverting the scores from the upper triangular part. If P(j,i) is the score from the upper triangular part, then P(j,i) = 1/P(i,j).

```
octave:6> P=pairwise_MAUT(S,n)
P =

   1.0000   0.2500   6.0000
   4.0000   1.0000   5.0000
   0.1667   0.2000   1.0000
```

Figure 1. Indicative example of the pairwise_MAUT function

Next, we create the eigenmethod_MAUT function to calculate the weights of the criteria and evaluate the Consistency Ratio of the comparison matrix generated earlier.

As seen in the code, we define a function called eigenmethod_MAUT, which takes the matrix P and computes its eigenvalues (lambda) and eigenvectors (V) using the eig command. Then, we construct a matrix L and check for any imaginary parts in its elements. Only the real eigenvalues are stored in L. After that, we find the maximum eigenvalue and its corresponding index. Using the eigenvector associated with this maximum eigenvalue, we compute the weights and normalize them so that their sum equals 1. To calculate the Consistency Index, we use the formula (Lmax - n) / (n - 1), as n the dimension of matrix P. Additionally, the possible values of the Random Index (RI) for different matrix sizes are stored in a separate matrix. The Consistency Ratio (CR) is then computed as CR = CI / RI and if n is equal to 1 or 2, the matrix is automatically considered consistent. If the matrix is not consistent, the pairwise_MAUT function is called again to generate a new consistent matrix.

```
octave:7> [W,CR]=eigenmethod_MAUT(P)
W =

   0.273792
   0.649233
   0.076975

CR = 0.2689
```

Figure 2. Indicative example of the eigenmethod_MAUT function

To model the MAUT method as a whole, first, we initialize the following variables:
M the number of experts, which is 15, N the number of criteria, which is 3, Nalter the number of alternatives, which is also 3, S the matrix containing the values of the comparison scale, and Pc the pairwise comparison matrix for the criteria, initialized with ones. Then, we call the pairwise_MAUT function to generate the comparison matrices for each expert and store them in the previously created Pc matrix. After that, for each criteria comparison matrix generated, we calculate the weights of the criteria using the eigenmethod_MAUT function and store them in the matrix w. Then, we compute the mean of these weights using the mean function and store them in the matrix W. Next, the step-by-step implementation of the MAUT method follows. First, we store the 3 alternatives and the values of the criteria, as defined in the first part of the assignment. Then, we determine the maximum and minimum values for each column and define which is the worst and which is the best value for each criterion. The next step is to calculate the utility for each alternative

and for each criterion. This was implemented by creating the utility function based on the theory from the course slides.
The function takes as arguments x, which is the value of the criterion, xi_minus, which is the worst value of the selected criterion, and xi_plus, which is the best value of the criterion. Based on the formula, we see that U is equal to (x - xi_minus) / (xi_plus - xi_minus). So, for each alternative and for each criterion, we call the utility function and use the values defined above. Then, we store the results in three matrices, one for each alternative. The final step for the final values of the alternatives, based on utility, is to multiply the criterion weights by each normalized criterion and sum the three values to get the final score. Therefore, we define a matrix criteria_weights, which is equal to the matrix W where we stored the mean weights, and then we perform matrix multiplication for each alternative with the weights. Finally, using a for loop, we sum all the values and find the final score for each criterion.

```
>> MAUTscript

ans = 3
Usoftware1 = 0.2722
Usoftware2 = 0.6390
Usoftware3 = 0.5368
```

Figure 3. Indicative example of the whole MAUT method

Finally, after implementing the entire method and calculating the weights and utilities of the alternatives, we can also represent the results graphically. For both the weights and the utilities, we create bar charts—each with a different color. In figure 1, we set the x-axis to represent the criteria and the y-axis to display the weights. Similarly, in figure 2, we set the x-axis to represent the alternatives and the y-axis to display the utilities. This way, we can visually interpret the results.

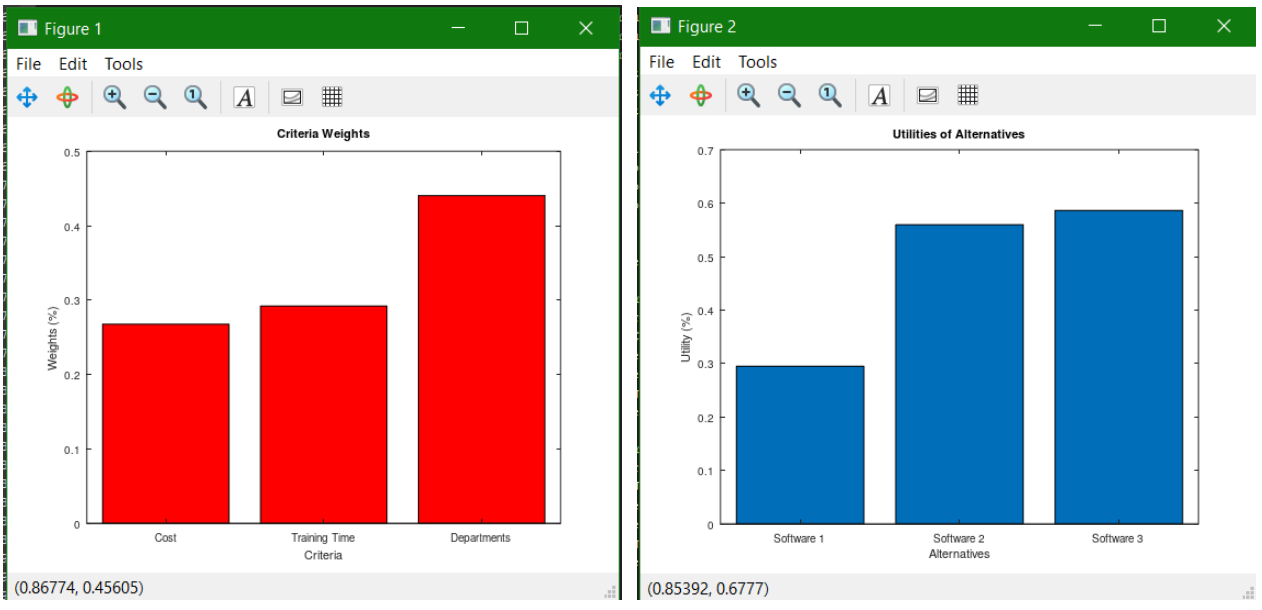

Figure 4. Indicative example of the weights diagram and the utilities diagram

2. In case a specialist did not provide a value for an element in one or more pairwise comparison matrices, we created the function new_value_MAUT. In this function, we accept the pairwise comparison matrix Pc and create a new matrix Pc_null, which is a copy of Pc. Then, we check each value in the Pc matrix. If a value is zero and it is not on the main diagonal (i.e., it is not in the same

row and column), the function updates this value with the reciprocal value of the symmetric position in the matrix. Finally, we return the updated Pc_null matrix.

Part C:

In this part of the project, based on the previous section, we perform the Monte Carlo simulation. Initially, we calculate the initial scores and rankings of the alternatives using the initial weights, and then we initialize the PRR matrix to store the results for each value of s. Next, for each value of s, we perform N iterations, and in each iteration, we add disturbances to the weights, normalize the new weights, and calculate the new scores and priorities. Then, we check whether the new rankings differ from the initial ones, and if they do, we increase the rank reversal counter. Additionally, we calculate the Probability of Rank Reversal (PRR) as the ratio of reversals to the total number of iterations. For the presentation of the results, as shown in the code above and in the results below, we display a graph of PRR as a function of s and present the initial and final priorities of the alternative solutions.

```
Original Scores and Rankings:
   0.2778
   0.7500
   0.4250
   2
   3
   1
Final PRR values:
   0.3479
   0.5156
   0.5938
   0.6469
   0.6743
```

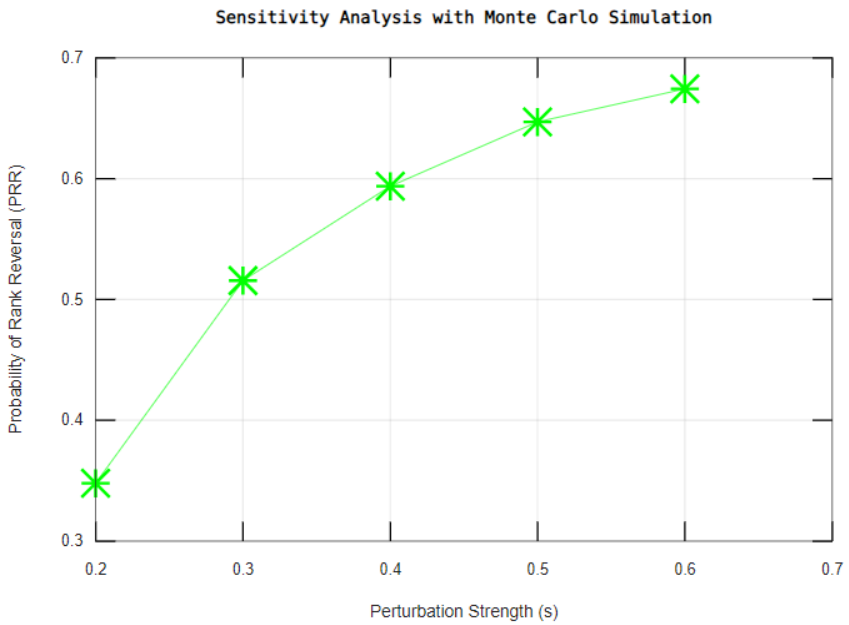

Figure 5. Indicative example of the Sensitivity Analysis with Monte Carlo Simulation