The project focuses on the ecological-economic model of Leontief-Ford.

The subject of the project involves fuzzy optimization problems based on the Leontief-Ford model, implemented using the Python programming language.

The goal of the project is to develop and analyze the Leontief-Ford model using fuzzy logic methods for optimization in ecological-economic systems. This goal is achieved by studying the theoretical foundations of fuzziness, developing an optimization algorithm based on this model, and introducing fuzziness into the model's parameters.

The outcome of the project includes developed optimization algorithms that use fuzzy input parameters based on the Leontief-Ford model. These algorithms were implemented in Python and can be used to analyze various ecological-economic scenarios, furthering understanding of the role and significance of fuzziness in economic models.

The static model of Leontief-Ford

The project focuses on the static Leontief-Ford model, one of the first interdisciplinary models that incorporates the interconnectedness of the economy and the environment. The model is a balance-based extension of the classic "input-output" model and covers two groups of sectors: primary production (material production) and ancillary production (related to the handling of hazardous waste). The main conditions of the model are expressed through a system of equations:

$$\begin{aligned} x_1 &= A_{11}x_1 + A_{12}x_2 + y_1, \\ x_2 &= A_{21}x_1 + A_{22}x_2 - y_2. \end{aligned} \tag{1.1}$$

In this system $x_1 = (x_1^1, x_2^1, \dots, x_n^1)^T$ is the column vector of gross output, $y_1 = (y_1^1, y_2^1, \dots, y_n^1)^T$ is the column vector of final product (consumption), $x_2 = (x_1^2, x_2^2, \dots, x_m^2)^T$ is the column vector of the volumes of destroyed pollutants, $y_2 = (y_1^2, y_2^2, \dots, y_m^2)^T$ is the column vector of the volumes of undestroyed pollutants. $A_{11} = (a_{ij}^{11})_1^n$ this is a square matrix representing the production costs of product *i* per unit of output *j*, $A_{12} = (a_{ig}^{12})_{i,g=1}^{n,m}$ this is a rectangular matrix depicting the production costs of product *i* per unit of pollutant *g destruction,* $A_{21} = (a_{kj}^{21})_{k,j=1}^{m,n}$ this is a rectangular matrix representing the output of pollutant *k* per unit of production *j*, $A_{22} = (a_{kg}^{22})_1^m$ – This is a square matrix indicating the output of pollutant *k* per unit of pollutant *g* destruction.

The model (1.1) is obtained when recording the balances for each sector of primary production (material production):

$$x_i^1 = \sum_{j=1}^{n} x_{ij}^{11} + \sum_{g=1}^{m} x_{ig}^{12} + y_i^1, \qquad i = 1, 2, \dots, n, \tag{1.2}$$

and for each sector of ancillary production (destruction of pollutants):

$$x_k^2 = \sum_{j=1}^{n} x_{kj}^{21} + \sum_{g=1}^{m} x_{kg}^{22} - y_k^2, \qquad k = 1, 2, \dots, m, \tag{1.3}$$

where x_{ij}^{11} are the costs of production *i* in sector *j* of the main (primary) production, x_{ig}^{12} are the costs of production *i* in sector *g* of the ancillary production (related to pollutant destruction), x_{kj}^{21} is the emission of pollutant *k* by sector *j* of the main production, x_{kg}^{22} is the emission of pollutant *k* by sector *g* of the ancillary production.

These relationships (1.2) (1.3) characterize the dependencies among the $2(n+m)+(n+m)^2$ quantities $x_i^1,\ y_i^1,\ x_k^2,\ y_k^2,\ x_{ij}^{11},\ x_{ig}^{12},\ x_{kj}^{21},\ x_{kg}^{22}$. The relationships have a very general character. To build the mathematical model, the primary assumption is that $x_{ij}^{11},\ x_{ig}^{12},\ x_{kj}^{21}$ and x_{kg}^{22} are functions of the production volumes of the respective sector:

$$\begin{gathered} x_{ij}^{11} = \varphi_{ij}^{11}\left(x_j^1\right), \\ x_{ig}^{12} = \varphi_{ig}^{12}\left(x_g^2\right), \\ x_{kj}^{21} = \varphi_{kj}^{21}\left(x_j^1\right), \\ x_{kg}^{22} = \varphi_{kg}^{22}\left(x_g^2\right), \\ i,j = 1,2,\dots,n, \\ k,g = 1,2,\dots,m. \end{gathered} \tag{1.4.}$$

The simplest version of the Leontief-Ford model assumes a proportional relationship between production costs and production volumes, employing linear homogeneous functions for production expenses:

$$x_{ij}^{11} = a_{ij}^{11}x_j^1,\ x_{ig}^{12} = a_{ig}^{12}x_g^2,\ x_{kj}^{21} = a_{kj}^{21}x_j^1,\ x_{kg}^{22} = a_{kg}^{22}x_g^2. \tag{1.5}$$

The proportionality coefficients $a_{ij}^{11} \geq 0,\ a_{ig}^{12} \geq 0,\ a_{kj}^{21} \geq 0$ and $a_{kg}^{22} \geq 0$ are respectively the coefficients for direct costs of production *i* for manufacturing unit *j*, direct costs of production *i* for destroying pollutant *g*, direct output of pollutant *k* by manufacturing unit *j*, and direct output of pollutant *k* by destroying unit *g*. These coefficients together form the matrices:

$$A_{11} = (a_{ij}^{11})_1^n,$$

$$A_{12} = (a_{ig}^{12})_{i,g=1}^{n,m},$$

$$A_{21} = (a_{kj}^{21})_{k,j=1}^{m,n},$$

$$A_{22} = (a_{kg}^{22})_{1}^{m}.$$

Substituting the values x_{ij}^{11}, x_{ig}^{12}, x_{kj}^{21} and x_{kg}^{22} from equation (1.5) into equations (1.2) and (1.3), we obtain a system of $n + m$ linear algebraic equations with $2(n + m)$ variables x_i^1, y_i^1, x_k^2, y_k^2:

$$x_i^1 = \sum_{j=1}^{n} a_{ij}^{11} x_j^1 + \sum_{g=1}^{m} a_{ig}^{12} x_g^2 + y_i^1, \qquad i = 1, 2, \dots, n, \tag{1.6}$$

$$x_k^2 = \sum_{j=1}^{n} a_{kj}^{21} x_j^1 + \sum_{g=1}^{m} a_{kg}^{22} x_g^2 - y_k^2, \qquad k = 1, 2, \dots, m. \tag{1.7}$$

In vector-matrix form, the system of equations (1.6), (1.7) takes the form of equation (1.1).

Assuming that the coefficients $a_{ij}^{11} \geq 0$, $a_{ig}^{12} \geq 0$, $a_{kj}^{21} \geq 0$, $a_{kg}^{22} \geq 0$, we implicitly extend the hypotheses of the main inter-industry balance model to all types of production activities (material production and pollutant destruction). Furthermore, we will consider the matrices A_{11}, A_{12}, A_{21} and A_{22} as non-negative:

$$A_{11} \geq 0,$$

$$A_{12} \geq 0,$$

$$A_{21} \geq 0,$$

$$A_{22} \geq 0.$$

In applied models, the nomenclature of destroyed pollutants is smaller than the nomenclature of existing pollutants. The expansion of the former occurs when the concentration of new pollutants becomes significant in terms of living conditions or production, and when technical and economic possibilities for combating existing pollutants are created.

It's also worth noting that the economic significance of the Leontief-Ford model requires all its variables to be non-negative, that is:

$$
\begin{aligned}
x_i^1 &\geq 0, \\
x_k^2 &\geq 0, \\
y_i^1 &\geq 0, \\
y_k^2 &\geq 0.
\end{aligned}
\tag{1.8}
$$

Remark: The Leontief-Ford inter-industry model (1.1), (1.8) can also be considered as a generalization of the classic inter-industry balance scheme for an open economic system, where the net import *i* of certain types of products exceeds non-productive consumption *y* (i.e., this production is also imported for productive consumption). In this case, the final product *y* for these industries is negative: $y = y - i \leq 0$. Then, by grouping industries with positive final products into a block (x_1, y_1), and others (with non-positive final production) into a block $(x_2, -y_2)$, we obtain matrices of direct material costs $A_{11} \geq 0,\ A_{12} \geq 0,\ A_{21} \geq 0,\ A_{22} \geq 0$ and the model of inter-industry balance in the form of (1.1), where $y_1 > 0,\ y_2 \geq 0$. Therefore, the results obtained from the study of the Leontief-Ford model can also be used for studying inter-industry balances of open economies.

Conditions for the existence of non-negative solutions

The interaction between human society and the natural environment should be considered as part of a single ecological-economic system at any level. It is considered that the most successful incorporation of the ecological factor in economic-mathematical models in the linear economy was achieved specifically through the "input-output" model of Leontief-Ford (1.1).

All matrices in the system of linear equations (1.1) are considered non-negative, and the economic significance of the Leontief-Ford model requires that all its variables be non-negative

Let's formally solve the system of linear algebraic equations (1.1) in two ways. The first method involves initially finding x_2 from the second equation, substituting the found x_2 into the first equation, and solving it with respect to x_1.

The second method involves initially finding x_1 from the first equation, substituting the found x_1 into the second equation, and solving it with respect to x_2. As a result, we obtain:

$$x_1 = (E_1 - A_1)^{-1}[y_1 - A_{12}(E_2 - A_{22})^{-1}y_2],$$
$$x_2 = (E_2 - A_2)^{-1}[A_{21}(E_1 - A_{11})^{-1}y_1 - y_2],$$

where E_1 and E_2 are the identity matrices of orders *n* and *m* respectively,, A_1 and A_2 are square matrices of orders *n* and *m* respectively:

$$A_1 = A_{11} + A_{12}(E_2 - A_{22})^{-1}A_{21}, \quad (1.9)$$
$$A_2 = A_{22} + A_{21}(E_1 - A_{11})^{-1}A_{12}. \quad (1.10)$$

Let's introduce the concept of productivity of the model. A productive economic system corresponds to such a matrix *A* that ensures the possibility of obtaining a final product $(y > 0)$ at appropriate proportions of production development $(x \geq 0)$. Matrix *A* is called productive if there exists a non-negative vector $x \geq 0$, that allows obtaining a positive vector of final product, $(E - A)x = y > 0$. For the productivity of matrix *A* a necessary and sufficient condition is the non-negativity of the matrix $B = (E - A)^{-1} \geq 0$. If matrix *A* is indecomposable, then matrix $B = (E - A)^{-1} > 0$.

We generalize the concept of productivity of a matrix in the case of a block matrix *A*, corresponding to the system (1.1):

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \geq 0. \quad (1.11)$$

We will consider the non-negative block matrix (1.11) productive if matrices A_{11}, A_{22}, as well as matrices A_1, A_2, defined by relations (1.9) and (1.10), are productive.

The productivity of matrices A_1 and A_2 implies the profitability of the primary and ancillary production over the entire production cycle of products and the entire cycle of pollutant destruction..

If matrices A_{11}, A_{22}, A_1 and A_2 are productive, then matrices

$$(E_1 - A_{11})^{-1} \geq 0, \quad (E_2 - A_{22})^{-1} \geq 0,$$
$$(E_1 - A_1)^{-1} \geq 0, \quad (E_2 - A_2)^{-1} \geq 0,$$

exist and have non-negative elements.

Basic Concepts of Fuzzy Set Theory

Fuzzy sets, a key tool in fuzzy logic developed by Lotfi Zadeh in 1965, were created to model the uncertainty and imprecision often encountered in the real world. Unlike traditional or "crisp" sets where an element either belongs to the set or not, a fuzzy set allows elements to have varying degrees of membership. This membership degree, measured from 0 to 1, indicates how "certain" an element belongs to the set. The closer the value is to 1, the more an element belongs to the set; values close to 0 indicate lesser belonging.

Fuzzy sets are defined through membership functions. The membership function $\mu A(x)$ for a fuzzy set A maps each element x in the universal set X to a value between 0 and 1. This can be expressed as:

$$A = \{(x, \mu A(x)) | x \in X\},$$

where x is an element of the universal set X, and $\mu A(x)$ is the degree of membership of x to A.

Fuzzy sets thus provide greater flexibility in modeling situations that are imprecise or undefined. They are widely used in various fields including artificial intelligence, decision-making systems, image processing, medical systems, and many more.

Fuzzy numbers are a special case of fuzzy sets used to represent and handle the uncertainty of numerical values. They represent a "soft" or imprecise range of numbers instead of a single precise value.

Like general fuzzy sets, fuzzy numbers are defined using membership functions. The membership function $\mu A(x)$ for a fuzzy number A maps each number x on the real line to a value between 0 and 1. This can be expressed as:

$$A = \{(x, \mu A(x)) | x \in R\},$$

where x is a number on the real numerical line R , and $\mu A(x)$ is the degree of membership of x to A.

One type of fuzzy numbers frequently used is the trapezoidal fuzzy numbers. They are defined by four numbers (a,b,c,d), where a and d are the left and right boundaries of the range, and b and c are the points where the membership degree reaches its maximum. The membership function for a trapezoidal fuzzy number typically has a shape that rises linearly from a to b, remains constant at 1 from b to c, and then declines linearly from c to d:

$$\begin{cases} \mu A(x) = 0, \ \ if \ x < a \ \ or \ \ x > d, \\ \mu A(x) = (x - a)/(b - a), \ \ if \ a \le x < b, \\ \mu A(x) = 1, \ \ if \ b \le x \le c, \\ \mu A(x) = (d - x)/(d - c), \ \ if \ c < x \le d. \end{cases} \quad (1.12)$$

This means that the degree of membership gradually increases from a to b, remains constant from b to c, and then decreases again from c to d.

Other membership functions include:

- **Triangular Membership Function:** This function has the shape of a triangle and is defined by three points: the left boundary a, the right boundary c, and the peak b, where the function reaches 1. The function is defined as:.

$$\begin{cases} \mu A(x) = 0, \ \ if \ \ x < a \ \ or \ \ x > c, \\ \mu A(x) = (x - a)/(b - a), \ \ if \ \ a \le x < b, \\ \mu A(x) = (c - x)/(c - b), \ \ if \ \ b \le x < c. \end{cases} \quad (1.13)$$

- **Gaussian Membership Function:** This function has the form of the classical Gaussian (normal) distribution. It is characterized by two parameters: the mean c and the standard deviation σ. The function is defined as:

$$\mu A(x) = \exp(-0.5((x - c)/\sigma)^2). \quad (1.14)$$

- Sigmoidal Membership Function: The sigmoid function has an "S" shape and is used to represent a process that transitions from one state to another

through a transitional period. It is defined by two parameters: the transition point c and the transition steepness a. The function is defined as.

$$\mu A(x) = 1/\big(1 + \exp\big(-a(x - c)\big)\big). \quad (1.15)$$

Despite the variability in fuzzy task specifications depending on the context, the general algorithm for processing fuzzy data can be described as follows

1. **Defining Fuzzy Sets:** The first step involves defining fuzzy sets that represent the uncertainty in the problem. This includes choosing membership functions that represent the degree of membership of elements to these sets.
2. **Constructing a Fuzzy Model:** The next step is to build a model that describes the problem using fuzzy sets. This may involve defining fuzzy rules if a fuzzy inference system is used, or using fuzzy numbers to represent imprecise values.
3. **Fuzzy Infernce:** After building the model, fuzzy inferences can be made using processes such as fuzzy activation, fuzzy aggregation, and defuzzification. This may include using methods such as max-min or mean of maximum (MOM).
4. **Interpreting Results:** The final step involves interpreting the results obtained. This may include converting the fuzzy conclusion into a specific numerical value through a process known as defuzzification, and analyzing this value to address the initial problem.

It's important to note that the specific details of these steps can vary significantly depending on the particular type of task and the method of fuzzy modeling used.

Optimization Problem Based on the Leontief-Ford Model

Optimal Functioning of the Ecological Economy

The Leontief-Ford model (1.1) is used as a component in more complex ecological-economic models. For instance, let's demonstrate the use of this model to construct the production function of the ecological economy. It is evident that the ecological economy is based on market relations, which encompass both economic and ecological components.

Let c_1 be the row vector of the economic valuation of final products, c_2 he row vector of economic valuation of damages from pollutant emissions into the environment, b the column vector of available general (primary) resources, and $B_1 \geq 0$ and $B_2 \geq 0$ the technological matrices of these resources spent on respective activities. Then, the optimal functioning of the ecological economy can be described by the following linear programming problem:

$$\begin{gathered} c_1 y_1 - c_2 y_2 \to \max, \\ x_1 = A_{11} x_1 + A_{12} x_2 + y_1, \\ x_2 = A_{21} x_1 + A_{22} x_2 - y_2, \\ B_1 x_1 + B_2 x_2 \leq b, \\ x_1 \geq 0, x_2 \geq 0, \\ y_1 \geq 0, y_2 \geq 0. \end{gathered} \tag{2.1}$$

Considering the components of vector b as parameters in the problem (2.1), and solving this parametric linear programming problem, we find the optimal solutions::

$$x_1^*(b), \qquad x_2^*(b), \qquad y_1^*(b), \qquad y_2^*(b),$$

based on which the ecological-economic production function is formed:

$$F(b) = c_1 y_1^*(b) - c_2 y_2^*(b).$$

If the constraints $y_1 \geq 0$ and $y_2 \geq 0$ are removed in problem (2.1), this problem, after excluding variables y_1 and y_2 can be simplified to:

$$\begin{gathered} (c_1(E_1 - A_{11}) - c_2 A_{21}) x_1 + \big(c_1 A_{12} - c_2(E_2 - A_{22})\big) x_2 \to \max, \\ B_1 x_1 + B_2 x_2 \leq b, \\ x_1 \geq 0, x_2 \geq 0. \end{gathered} \tag{2.2}$$

Problem (2.2) describes the optimal functioning of a hypothetical ecological economy that can both produce products and implement clean environment policies. The production function of such an ecological economy is expressed as:

$$F_0(b) = (c_1(E_1 - A_{11}) - c_2A_{21})\bar{x}_1(b) + \big(c_1A_{12} - c_2(E_2 - A_{22})\big)\bar{x}_2(b),$$

where $\bar{x}_1(b)$ and $\bar{x}_2(b)$ are the optimal solutions of problem (2.2).

Given that the feasible region in problem (2.2) is parametrically defined, if the goal is only to find the parameters of the production function (the objective function of this problem), a rational approach to achieving such a goal is to switch to the dual problem in (2.2):

$$pb \to \min,$$
$$pB_1 \geq c_1(E_1 - A_{11}) - c_2A_{21},$$
$$pB_2 \geq c_1A_{12} - c_2(E_2 - A_{22}),$$
$$p \geq 0,$$

where p is a vector of resource prices of the corresponding dimension. Analyzing this problem, we conclude that the sufficient condition for the existence of its solution is the conditions:

$$c_1(E_1 - A_{11}) \geq c_2A_{21} \quad \text{та} \quad c_2A_{12} \geq c_2(E_2 - A_{22}),$$

which link the coefficients c_1 and c_2 together.

Since the feasible region in the dual problem is uniquely defined, the orientation of the hyperplane level of the objective function will depend on the components of the parametric vector b thus obtaining the optimal value of the objective function as a function of the parameters of vector *b.*

And since, as is well-known, the values of the objective functions for the primal and dual problems coincide at the optimal solutions, this is how we obtain the optimal production function $F(b)$ as the objective function of problem (2.2).

It's worth noting that this approach can, under certain circumstances, also be applied when solving problems with fuzzy constraints. In such cases, fuzzy

constraints are incorporated into the linear programming model, allowing for flexibility and tolerance in the constraint satisfaction, which reflects more realistic or practical scenarios where parameters may not always be rigid or precisely known. This adaptability makes the model more robust and applicable to a wider range of real-world problems where uncertainty and vagueness are inherent.

The Two-Dimensional Case

Consider a variant of problem (2.1) where $n = 1,\ m = 1$. This implies that there is one product being produced and one type of pollution being generated. Essentially, this means that problem (2.1) has two dimensions.

In the file **linear_program_setup.py**, there is a program that uses the Leontief-Ford model to solve the linear programming problem of form (2.1) using the **scipy.optimize** library. Here's a more detailed explanation of the code setup::

Initially, the model parameters are defined: A_{11}, A_{12}, A_{21}, A_{22}, C_1, C_2, B_1, B_2 and b.

- $A_{11},\ A_{12},\ A_{21},\ A_{22}$ are technological matrices for direct production costs and direct emissions of pollutants, respectively.
- c_1 and c_2 are linear vectors assessing the economic value of final products and the damage from environmental pollutant emissions, respectively.
- $B_1,\ B_2$ are technological matrices of resource costs.
- b is the vector representing available general resources.

Next, the coefficients for the objective function and constraints are set:

```
c = [-C1*(E1-A11)+C2*A21, (C2*(E2-A22)-C1*A12)]
```

These are the coefficients for the objective function, which according to model (2.1), we aim to maximize: $(c_1(E_1 - A_{11}) - c_2A_{21})x_1 + \big(c_1A_{12} - c_2(E_2 - A_{22})\big)x_2$. This expression is adapted for use in the **linprog** function, which minimizes the expression, so we take the negative of these coefficients.

```
A_ub = [[B1, B2],
        [-E1+A11, -A12],
        [-A21,-E2+A22]]
b_ub = [b,0,0]
```

This is the matrix of coefficients and the vector of constant terms corresponding to the inequality constraints in problem (2.1):

$$(E_1 - A_{11})x_1 - A_{12}x_2 \geq 0,$$

$$A_{21}x_1 - (E_2 - A_{22})x_2 \geq 0,$$

$$B_1x_1 + B_2x_2 \leq b.$$

```
x0_bounds = (0, None)
x1_bounds = (0, None)
```

In this model, the variables x_1 and x_2 must be non-negative, so the lower bounds are set to zero and upper bounds are set to **None** (indicating infinity).

```
res = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=[x0_bounds, x1_bounds], method='highs')
```

This is a call to the **linprog** function, which solves the linear programming problem. We pass the coefficients of the objective function, the inequality constraints, and the variable sign bounds. The optimization method is set to 'highs', which is one of the newest and most efficient solvers for linear programming. The results are displayed on the screen and shown on a graph.

With model parameter values of $A_{11} = 0.3$, $A_{12} = 0.4$, $A_{21} = 0.5$, $A_{22} = 0.6$, $c_1 = 0.9$, $c_2 = 0.8$, $B_1 = B_2 = 4$, $b = 80$, $E_1 = E_2 = 1$, with model parameter values of:

```
Optimal value: 4.6
X1, X2: [20.  0.]
```

In the results, we see the optimal value of the objective function and the vector of variables that provide this value.

The graph (Figure 2.1) illustrates the areas corresponding to the constraints of the problem. The feasible solution region is located at the intersection of these areas. A red dot indicates the optimal solution of the problem.

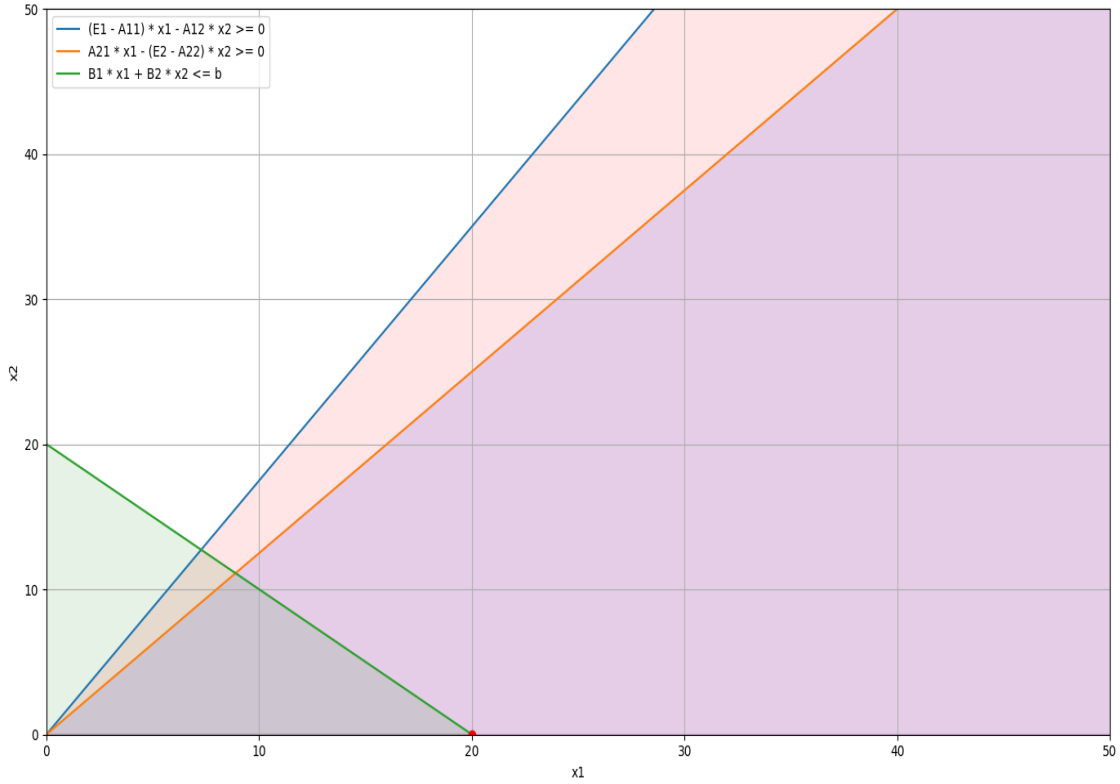

Figure 2.1. Feasible region of the problem (2.1)

Three-Dimensional Case

To provide greater flexibility and accuracy in our model and to better explain the variety of factors influencing ecology and economy, the ecological-economic production model based on the Leontief-Ford model will be extended from one-dimensional matrices to matrices of dimension $n = 2,\ m = 1$. In practical terms, this means that the problem (2.1) will now have three dimensions.

Using matrices of larger size has several advantages over single matrices. Firstly, they allow us to model the situation more intricately by considering more interconnections between various input and output variables. This can include different types of resources, various economic sectors, and various impacts on the environment, among others.

Secondly, by using larger matrix dimensions, we can circumvent some limitations. For instance, in smaller-dimensional matrices, scaling issues can arise where the larger values of one factor might overshadow the influence of smaller factors. Increasing the dimensionality of the problem allows for more precise modeling of these interrelationships.

Thirdly, larger matrices can help more accurately model the real world. In real-world scenarios, one often deals with a large number of variables, and using models of appropriate dimensions helps better accommodate this complexity.

Thus, expanding the model to use matrices of dimensions $n = 2,\ m = 1$ is an important step towards a more realistic and accurate model. This will help us better understand the relationships between the economy and the environment and assist in developing more effective strategies for balanced development.

Now that the parameters for problem (2.1) are as follows:

$$A_{11} = \begin{pmatrix} a_{11}^{11} & a_{12}^{11} \\ a_{21}^{11} & a_{22}^{11} \end{pmatrix}$$ – a matrix;

$$A_{12} = \begin{pmatrix} a_{1}^{12} \\ a_{2}^{12} \end{pmatrix}$$ – a vector;

$$A_{21} = (a_{1}^{21} \quad a_{2}^{21})$$ – a vector;

$$A_{22} = a_{22} \text{ – a scalar;}$$

$$B_1 = (b_1^1 \quad b_2^1) \text{ – a vector;}$$

$$B_2 = b_2 \text{ – a scalar;}$$

$$E_1 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ – a matrix;}$$

$$E_2 = 1;$$

$$b \text{ – a scalar;}$$

$$x_1 = \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix},\ y_1 = \begin{pmatrix} y_{11} \\ y_{12} \end{pmatrix} \text{ – a vector;}$$

$$c_1 = (c_{11} \quad c_{12}) \text{ – a vector;}$$

$$x_2,\ y_2,\ c_2 \text{ – a scalar.}$$

Substituting these variables into problem (2.1), we obtain:

$$(c_{11} \quad c_{12}) \begin{pmatrix} y_{11} \\ y_{12} \end{pmatrix} - c_2 y_2 \to \max,$$

$$\begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} = \begin{pmatrix} a_{11}^{11} & a_{12}^{11} \\ a_{21}^{11} & a_{22}^{11} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} + \begin{pmatrix} a_1^{12} \\ a_2^{12} \end{pmatrix} x_2 + \begin{pmatrix} y_{11} \\ y_{12} \end{pmatrix},$$

$$x_2 = (a_1^{21} \quad a_2^{21}) \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} + a_{22} x_2 - y_2,$$

$$(b_1^1 \quad b_2^1) \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} + b_2 x_2 \le b,$$

$$\begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} \ge 0, \qquad x_2 \ge 0,$$

$$\begin{pmatrix} y_{11} \\ y_{12} \end{pmatrix} \ge 0, \qquad y_2 \ge 0.$$

Next, we expand the model component by component. Multiplying vectors c_1 and y_1 to write down the objective function of the problem:

$$c_{11} y_{11} + c_{12} y_{12} - c_2 y_2 \to \max.$$

Next, we express $y_{11},\ y_{12}$ and y_2 from the corresponding constraints. First, we express y_{11} and y_{12}:

$$\begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} = \begin{pmatrix} a_{11}^{11} & a_{12}^{11} \\ a_{21}^{11} & a_{22}^{11} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} + \begin{pmatrix} a_1^{12} \\ a_2^{12} \end{pmatrix} x_2 + \begin{pmatrix} y_{11} \\ y_{12} \end{pmatrix};$$

$$\begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} - \begin{pmatrix} a_{11}^{11} & a_{12}^{11} \\ a_{21}^{11} & a_{22}^{11} \end{pmatrix} \begin{pmatrix} x_{11} \\ x_{12} \end{pmatrix} - \begin{pmatrix} a_1^{12} \\ a_2^{12} \end{pmatrix} x_2 = \begin{pmatrix} y_{11} \\ y_{12} \end{pmatrix};$$

$$\left(\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}-\begin{pmatrix}a_{11}^{11} & a_{12}^{11}\\ a_{21}^{11} & a_{22}^{11}\end{pmatrix}\right)\begin{pmatrix}x_{11}\\ x_{12}\end{pmatrix}-\begin{pmatrix}a_1^{12}\\ a_2^{12}\end{pmatrix}x_2=\begin{pmatrix}y_{11}\\ y_{12}\end{pmatrix};$$

$$\begin{pmatrix}1-a_{11}^{11} & -a_{12}^{11}\\ -a_{21}^{11} & 1-a_{22}^{11}\end{pmatrix}\begin{pmatrix}x_{11}\\ x_{12}\end{pmatrix}-\begin{pmatrix}a_1^{12}\\ a_2^{12}\end{pmatrix}x_2=\begin{pmatrix}y_{11}\\ y_{12}\end{pmatrix};$$

$$(1-a_{11}^{11})x_{11}-a_{12}^{11}x_{12}-a_1^{12}x_2=y_{11};$$

$$-a_{21}^{11}x_{11}+(1-a_{22}^{11})x_{12}-a_2^{12}x_2=y_{12}.$$

Now, let's express y_2:

$$x_2=(a_1^{21}\quad a_2^{21})\begin{pmatrix}x_{11}\\ x_{12}\end{pmatrix}+a_{22}x_2-y_2;$$

$$(a_1^{21}\quad a_2^{21})\begin{pmatrix}x_{11}\\ x_{12}\end{pmatrix}-x_2-a_{22}x_2=y_2;$$

$$a_1^{21}x_{11}+a_2^{21}x_{12}+(a_{22}-1)x_2=y_2.$$

Next, multiply vectors B_1 and x_1 to get the last constraint of the problem in the required form:

$$b_1^1x_{11}+b_2^1x_{12}+b_2x_2\le b.$$

Finally, substitute $y_{11},\ y_{12}$ and y_2 into the objective function and highlight the coefficients for the variables $x_{11},\ x_{12}$ and x_2:

$$c_{11}\big((1-a_{11}^{11})x_{11}-a_{12}^{11}x_{12}-a_1^{12}x_2\big)+c_{12}(-a_{21}^{11}x_{11}+(1-a_{22}^{11})x_{12}-a_2^{12}x_2)$$
$$-$$
$$-c_2(a_1^{21}x_{11}+a_2^{21}x_{12}+(a_{22}-1)x_2)\to\max;$$

$$x_{11}(c_{11}(1-a_{11}^{11})-c_{12}a_{21}^{11}-c_2a_1^{21})+x_{12}(-c_{11}a_{12}^{11}+c_{12}(1-a_{22}^{11})-c_2a_2^{21})$$
$$+$$
$$+x_2\big(-c_{11}a_1^{12}-c_{12}a_2^{12}-c_2(a_{22}-1)\big)\to\max.$$

After completing all the previous steps, we obtain the following model:

$$(c_{11}(1-a_{11}^{11})-c_{12}a_{21}^{11}-c_2a_1^{21})x_{11}+(-c_{11}a_{12}^{11}+c_{12}(1-a_{22}^{11})-c_2a_2^{21})x_{12}$$
$$+$$
$$+\big(-c_{11}a_1^{12}-c_{12}a_2^{12}-c_2(a_{22}-1)\big)x_2\to\max.$$

$$
\begin{gathered}
(1-a_{11}^{11})x_{11}-a_{12}^{11}x_{12}-a_{1}^{12}x_{2}\geq 0,\\
-a_{21}^{11}x_{11}+(1-a_{22}^{11})x_{12}-a_{2}^{12}x_{2}\geq 0,\\
a_{1}^{21}x_{11}+a_{2}^{21}x_{12}+(a_{22}-1)x_{2}\geq 0,\\
b_{1}^{1}x_{11}+b_{2}^{1}x_{12}+b_{2}x_{2}\leq b,\\
x_{11}\geq 0,\qquad x_{12}\geq 0,\qquad x_{2}\geq 0.
\end{gathered}
\tag{2.3}
$$

Having detailed the model in form (2.3), we proceed to implement it in the code. The first step in this process is to convert the maximization function into a minimization function. This is a necessary step as **linprog**—an optimization solver available in the SciPy Python library—only performs minimization tasks. Note that maximizing a function is equivalent to minimizing its negative value. Thus, we can easily adapt the task for use with **linprog** by simply changing the sign of the objective function:

$$
\begin{gathered}
\big(c_{12}a_{21}^{11}+c_{2}a_{1}^{21}-c_{11}(1-a_{11}^{11})\big)x_{11}+(c_{11}a_{12}^{11}-c_{12}(1-a_{22}^{11})+c_{2}a_{2}^{21})x_{12}+\\
+\big(c_{11}a_{1}^{12}+c_{12}a_{2}^{12}+c_{2}(a_{22}-1)\big)x_{2}\rightarrow \min.
\end{gathered}
\tag{2.4}
$$

In the code, this is implemented as follows:

```
c_coeff = [-(C11 * (1 - A11[0][0]) - C12 * A11[1][0] - C2*A21[0][0]),
           -(-C11*A11[0][1]+C12*(1-A11[1][1])-C2*A21[0][1]),
           -(-C11*A12[0]-C12*A12[1]-C2*(A22-1))]
```

Next, we need to insert the constraints into the code. To successfully implement the constraints in the code, they need to be adapted to the format acceptable for **linprog**. This format requires all constraints to be formatted in the form $Ax + b \leq 0$. We have:

$$
\begin{gathered}
-(1-a_{11}^{11})x_{11}+a_{12}^{11}x_{12}+a_{1}^{12}x_{2}\leq 0,\\
a_{21}^{11}x_{11}-(1-a_{22}^{11})x_{12}+a_{2}^{12}x_{2}\leq 0,\\
-a_{1}^{21}x_{11}-a_{2}^{21}x_{12}-(a_{22}-1)x_{2}\leq 0,\\
b_{1}^{1}x_{11}+b_{2}^{1}x_{12}+b_{2}x_{2}\leq b,\\
x_{11}\geq 0,\qquad x_{12}\geq 0,\qquad x_{2}\geq 0.
\end{gathered}
$$

In the code, constraints are implemented as:

```
A_ub = [[B11[0], B12[0], B2[0]],
        [-1 + A11[0][0], A11[0][1],  A12[0]],
        [A11[1][0], -1 + A11[1][1], A12[1]],
        [-A21[0][0], -A21[0][1], -A22+1]]
b_vector = [b2[0], 0, 0, 0]
x_bounds = [(0, None), (0, None), (0, None)]
```

Using the provided variable values:

$$A_{11} = \begin{pmatrix} 0.5 & 0.3 \\ 0.4 & 0.3 \end{pmatrix},$$

$$A_{12} = \begin{pmatrix} 0.3 \\ 0.2 \end{pmatrix},$$

$$A_{21} = (0.3 \quad 0.2),$$

$$A_{22} = a_{22} = 0.2,$$

$$B_1 = (0.6 \quad 0.5),$$

$$B_2 = b_2 = 0.4,$$

$$b = 150,$$

$$c_1 = (0.5 \quad 0.6),$$

$$c_2 = 0.55,$$

the code issues the following results:

```
Optimal Solution= 12.919708029197084 x_values= [109.48905109 113.13868613  69.34306569]
```

When these values are used in the optimization code, the results provide the "Optimal Solution" and "x_values". The "Optimal Solution" represents the optimal value of the objective function obtained as a result of solving the optimization problem. The "x_values" are the vector of optimal values for the variables x_{11}, x_{12} and x_2 obtained from solving the optimization problem.

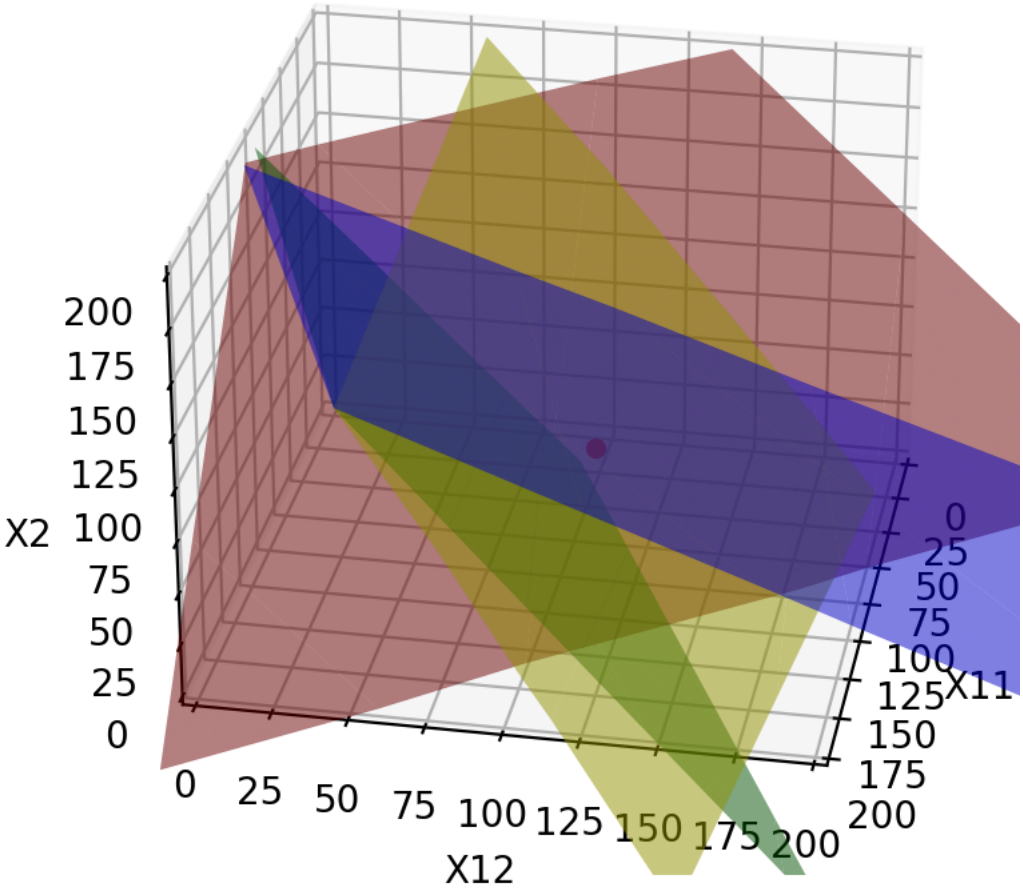

Figure 2.2. Optimal Solution for Problem (2.3)

In Figure 2.2, a graph is presented where the optimal values of the variables x_{11}, x_{12} and $x_2$re shown as a red dot. The following constraints are visually represented in the graph:

$$(1 - a_{11}^{11})x_{11} - a_{12}^{11}x_{12} - a_1^{12}x_2 \geq 0 \quad \text{(displayed in blue)},$$

$$-a_{21}^{11}x_{11} + (1 - a_{22}^{11})x_{12} - a_2^{12}x_2 \geq 0 \quad \text{(displayed in yellow)},$$

$$a_1^{21}x_{11} + a_2^{21}x_{12} + (a_{22} - 1)x_2 \geq 0 \quad \text{(displayed in green)},$$

$$b_1^1x_{11} + b_2^1x_{12} + b_2x_2 \leq b \quad \text{(displayed in red)}.$$

The full version of the code can be found in the file **fuzzy_membership_functions.py**.

Fuzzy Formulations of Problems Based on the Leontief-Ford Model

Optimization Problem with Fuzzily Defined Objective Function Parameters

After exploring the theoretical foundations of fuzzy logic, including its fundamental concepts such as fuzzy sets, membership functions, and various methods for representing and processing fuzziness, let's apply this knowledge in practice. We will introduce an element of uncertainty by adding fuzziness to some parameters of our problem. This will allow us to investigate how fuzziness can affect the decision-making process and choose the best possible solution considering this uncertainty.

For this purpose, we will use the Python library **skfuzzy**, specifically developed for working with fuzzy systems. It offers a wide range of functions for dealing with fuzzy sets, including functions for defining and visualizing membership functions.

We will consider fuzziness in the parameters c_1 and c_2, which influence the formation of the objective function of the optimization model discussed in the previous chapter. The goal is to understand the impact of fuzziness on the optimal solutions to the problem. This will not only help us better understand the relationship between fuzziness and decision-making processes but also evaluate the practical application of fuzzy logic methods in optimization and linear programming.

Solving an optimization problem with fuzzy parameters differs from traditional solving. Introducing fuzziness allows for more flexible responses to changing conditions of the problem. Considering the fuzziness of parameters enables the consideration of a range of possible scenarios, which facilitates the search for a more robust and adaptive solution, rather than a specific optimal

point. This can be particularly useful in conditions of uncertainty or rapidly changing environments.

The algorithm for solving the optimization problem with fuzziness in the parameters differs in that it requires processing all combinations of values c_1 and c_2. This "probability" reflects the degree of suitability of the parameter combination and is determined as the minimum of the two membership values to the respective fuzzy sets for c_1 and c_2. he formula for calculating the degree of membership looks like this:

$$\varphi = \min(\mu c_1;\ \mu c_2), \tag{3.1}$$

where μc_1 and μc_2 are the membership values to the respective fuzzy sets.

Fuzzy Parameters of the Objective Function with Trapezoidal Membership Functions

Previously, we discussed the functional code for solving a three-dimensional optimization problem based on the Leontief-Ford model (1.1). However, considering that the parameters of the objective function c_1 and c_2 often prove to be imprecise in the real world, incorporating fuzziness into these parameters can significantly enhance the accuracy of related forecasts. As noted in section 3.1, the algorithm for solving the optimization problem undergoes some changes after introducing fuzziness into the objective function parameters.

Firstly, we will introduce trapezoidal fuzziness (1.12) into the parameters c_1 and c_2. Since c_1 in the model is a two-dimensional vector, fuzziness will be introduced into each of its components, namely c_{11} and c_{12}.

Fuzz.trapmf() is a function from the scikit-fuzzy library (submodule **fuzz**), used to create a fuzzy trapezoidal membership function. Each set is defined by four points that determine the shape of the trapezoid. We will choose the following values: for c_{11}: $[0.3, 0.4, 0.6, 0.8]$, for c_{12}: $[0.35, 0.45, 0.7, 0.95]$ and

for c_2: $[0.4, 0.5, 0.6, 0.8]$. The values of these three parameters vary from 0.1 to 0.8 in steps of 0.05. This is implemented in the code as follows:

```
C11_domain = np.arange(0.1, 0.8, 0.05)
C12_domain = np.arange(0.1, 0.8, 0.05)
C2_domain = np.arange(0.1, 0.8, 0.05)

C11_values = fuzz.trapmf(C11_domain, [0.3, 0.4, 0.6, 0.85])
C12_values = fuzz.trapmf(C12_domain, [0.35, 0.45, 0.7, 0.95])
C2_values = fuzz.trapmf(C2_domain, [0.4, 0.5, 0.6, 0.8])
```

Figure 3.1 shows how fuzziness affects the parameters. You can see the distribution of values for each parameter considering the introduced fuzziness.

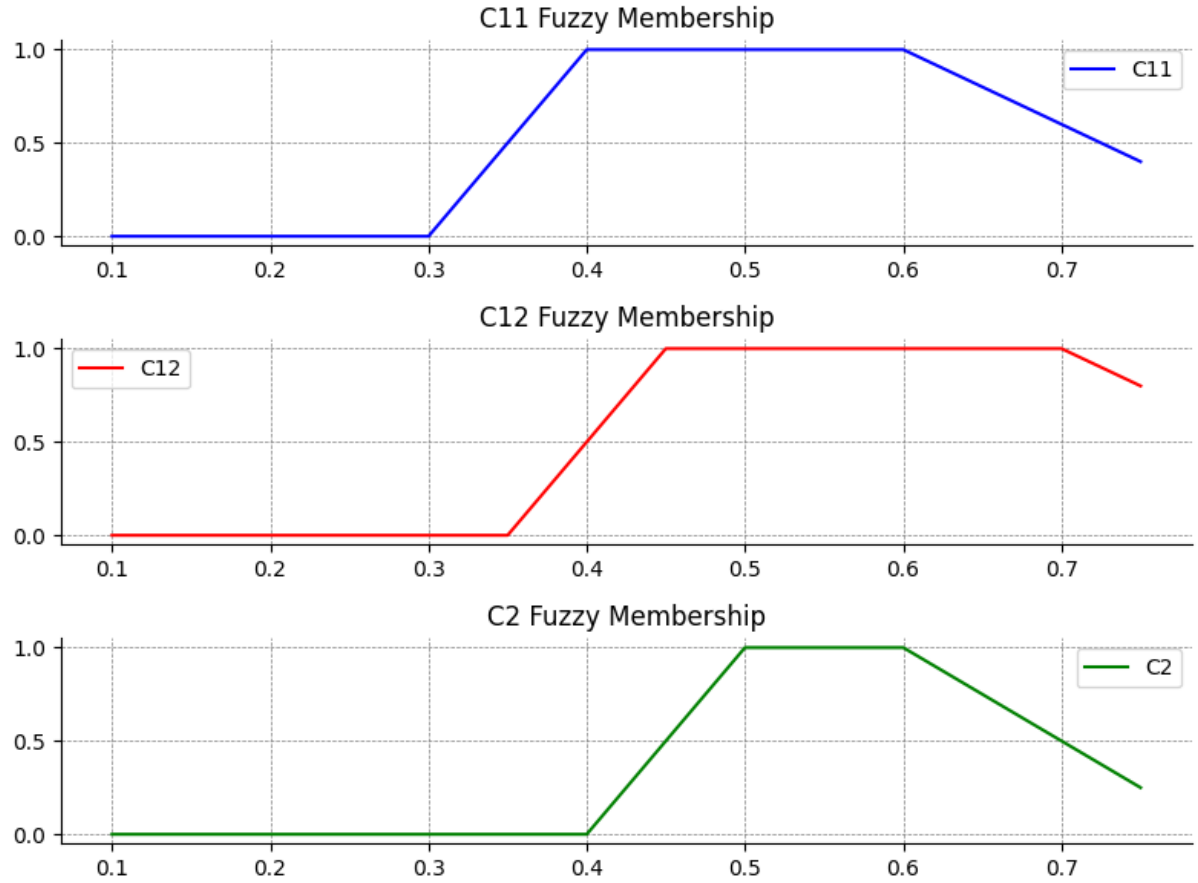

Figure 3.1. Trapezoidal Membership Functions of the Coefficients of the Objective Function

After introducing fuzzy settings for the parameters of the objective function of problem (2.3), the next important step is to generate all possible combinations

of values for these parameters. This process is crucial as it allows modeling a broad spectrum of possible scenarios in the production process. This approach helps in understanding how different values within the fuzzy ranges can impact the outcome, providing a more robust decision-making framework in the presence of uncertainty. This process is implemented in the following code snippet:

```
for i in range(len(C11_domain)):
    for j in range(len(C12_domain)):
        for k in range(len(C2_domain)):
```

During the iterative process, the code also calculates the degree of membership for the combination of parameters, determined according to formula (3.1).

The formula for calculating the degree of membership in the code is presented as follows:

```
min(C11_values[i], C12_values[j], C2_values[k])
```

Also, as we have to iterate through all possible combinations of the parameters c_{11}, c_{12} and c_2, the objective function (2.4) has also changed. Now it will look like this:

```
c_coeff = [-(C11_domain[i] * (1 - A11[0][0]) - C12_domain[j] * A11[1][0] - C2_domain[k]*A21[0][0]),
           -(-C11_domain[i]*A11[0][1]+C12_domain[j]*(1-A11[1][1])-C2_domain[k]*A21[0][1]),
           -(-C11_domain[i]*A12[0]-C12_domain[j]*A12[1]-C2_domain[k]*(A22-1))]
```

After iterating through all possible combinations of parameters c_{11}, c_{12} and c_2, and solving optimization problems, the code generates a table that presents the results of these calculations.

- c_{11}, c_{12}, c_2 – the values of parameters c_{11}, c_{12} and c_2 for each separate optimization task. for each separate optimization task.
- «*Optimal Solution*» – this is the optimal value of the objective function obtained as a result of solving the optimization problem.

- «*Chance*» – this is the minimum value of the degree of membership from the three parameters c_{11}, c_{12} and c_2 (according to (3.1)). This indicator can be used to assess the risk associated with each separate optimization task.
- «*x_values*» – this is the vector of optimal values of the variables x_{11}, x_{12} and x_2 that were obtained as a result of solving the optimization problem.

Adding values to the table is performed in this code snippet:

```
y11=(1-A11[0][0])*res.x[0]-A11[0][1]*res.x[1]-A12[0]*res.x[2]
y12=-A11[1][0]*res.x[0]+(1-A11[1][1])*res.x[1]-A12[1]*res.x[2]
y2=A21[0][0]*res.x[0]+A21[0][1]*res.x[1]+(A22-1)*res.x[2]
b=B11[0]*res.x[0]+B12[0]*res.x[1]+B2[0]*res.x[2]

if res.success and y11>=0 and y12>=0 and y2>=0 and b>=b2[0]:
    y22=A21[0][0]*res.x[0]+A21[0][1]*res.x[1]+(A22-1)*res.x[2]
    data_list.append({"C11": C11,
                      "C12": C12,
                      "C2": C2,
                      "Optimal Solution": -res.fun,
                      "Chance": "{:.2f}".format(min(C11_values[i], C12_values[j], C2_values[k])),
                      "x_values": res.x})
```

It is important to note that Python is a high-level, interpreted programming language, and as such, it is focused on ease of use and readability of code, not on low-level control over computation accuracy and memory management. Because of this, certain nuances may arise when dealing with constraints in optimization tasks.

Most optimizers, including those used in Python libraries, are iterative and use specific stopping criteria to determine when they should stop searching for the optimal solution. These stopping criteria can cause errors if they are too strict. That is why, before adding data to the table, the code rechecks whether the optimizer has indeed found a solution that meets all constraints.

The results of the calculations for the optimization task with trapezoidal membership functions for the coefficients of the objective function are shown in Table 3.1.

In Table 3.1, only 30 combinations of c_{11}, c_{12} and c_2, are displayed, sorted by the maximum value of the "Chance" column. In total, the code produced optimal solutions for 2669 combinations of c_{11}, c_{12} and c_2.

Table 3.1

	C11	C12	C2	Optimal Solut	Chance	x_values
1	0.6	0.7	0.55	19.71	1.00	[104.65116279 174.41860465 0.]
2	0.5	0.6	0.55	12.92	1.00	[109.48905109 113.13868613 69.34306569]
3	0.4	0.6	0.5	15.00	1.00	[104.65116279 174.41860465 0.]
4	0.4	0.45	0.6	9.69	1.00	[109.48905109 113.13868613 69.34306569]
5	0.4	0.45	0.55	9.69	1.00	[109.48905109 113.13868613 69.34306569]
6	0.4	0.45	0.5	9.69	1.00	[109.48905109 113.13868613 69.34306569]
7	0.4	0.6	0.55	12.92	1.00	[109.48905109 113.13868613 69.34306569]
8	0.55	0.45	0.6	9.69	1.00	[109.48905109 113.13868613 69.34306569]
9	0.6	0.65	0.5	19.01	1.00	[104.65116279 174.41860465 0.]
10	0.6	0.65	0.55	15.70	1.00	[104.65116279 174.41860465 0.]
11	0.6	0.65	0.6	14.00	1.00	[109.48905109 113.13868613 69.34306569]
12	0.4	0.6	0.6	12.92	1.00	[109.48905109 113.13868613 69.34306569]
13	0.45	0.7	0.5	23.02	1.00	[104.65116279 174.41860465 0.]
14	0.55	0.5	0.5	10.77	1.00	[109.48905109 113.13868613 69.34306569]
15	0.55	0.5	0.55	10.77	1.00	[109.48905109 113.13868613 69.34306569]
16	0.45	0.7	0.55	19.71	1.00	[104.65116279 174.41860465 0.]
17	0.6	0.7	0.5	23.02	1.00	[104.65116279 174.41860465 0.]
18	0.6	0.45	0.55	9.69	1.00	[109.48905109 113.13868613 69.34306569]
19	0.55	0.5	0.6	10.77	1.00	[109.48905109 113.13868613 69.34306569]
20	0.6	0.5	0.55	10.77	1.00	[109.48905109 113.13868613 69.34306569]
21	0.6	0.5	0.5	10.77	1.00	[109.48905109 113.13868613 69.34306569]
22	0.5	0.7	0.55	19.71	1.00	[104.65116279 174.41860465 0.]
23	0.5	0.65	0.6	14.00	1.00	[109.48905109 113.13868613 69.34306569]
24	0.5	0.65	0.5	19.01	1.00	[104.65116279 174.41860465 0.]
25	0.4	0.7	0.55	19.71	1.00	[104.65116279 174.41860465 0.]
26	0.6	0.6	0.6	12.92	1.00	[109.48905109 113.13868613 69.34306569]
27	0.45	0.45	0.5	9.69	1.00	[109.48905109 113.13868613 69.34306569]
28	0.45	0.45	0.55	9.69	1.00	[109.48905109 113.13868613 69.34306569]
29	0.6	0.55	0.6	11.84	1.00	[109.48905109 113.13868613 69.34306569]
30	0.45	0.6	0.6	12.92	1.00	[109.48905109 113.13868613 69.34306569]

Following the table, the code outputs a graph (Figure 3.2), which displays the optimal values of the variables x_{11}, x_{12} and x_2 as red points, as well as the constraints:

$$(1 - a_{11}^{11})x_{11} - a_{12}^{11}x_{12} - a_1^{12}x_2 \geq 0 \quad \text{(displayed in blue)},$$

$$-a_{21}^{11}x_{11} + (1 - a_{22}^{11})x_{12} - a_2^{12}x_2 \geq 0 \quad \text{(displayed in yellow)},$$

$a_1^{21}x_{11} + a_2^{21}x_{12} + (a_{22} - 1)x_2 \geq 0$ (displayed in green),

$b_1^1 x_{11} + b_2^1 x_{12} + b_2 x_2 \leq b$ (displayed in red).

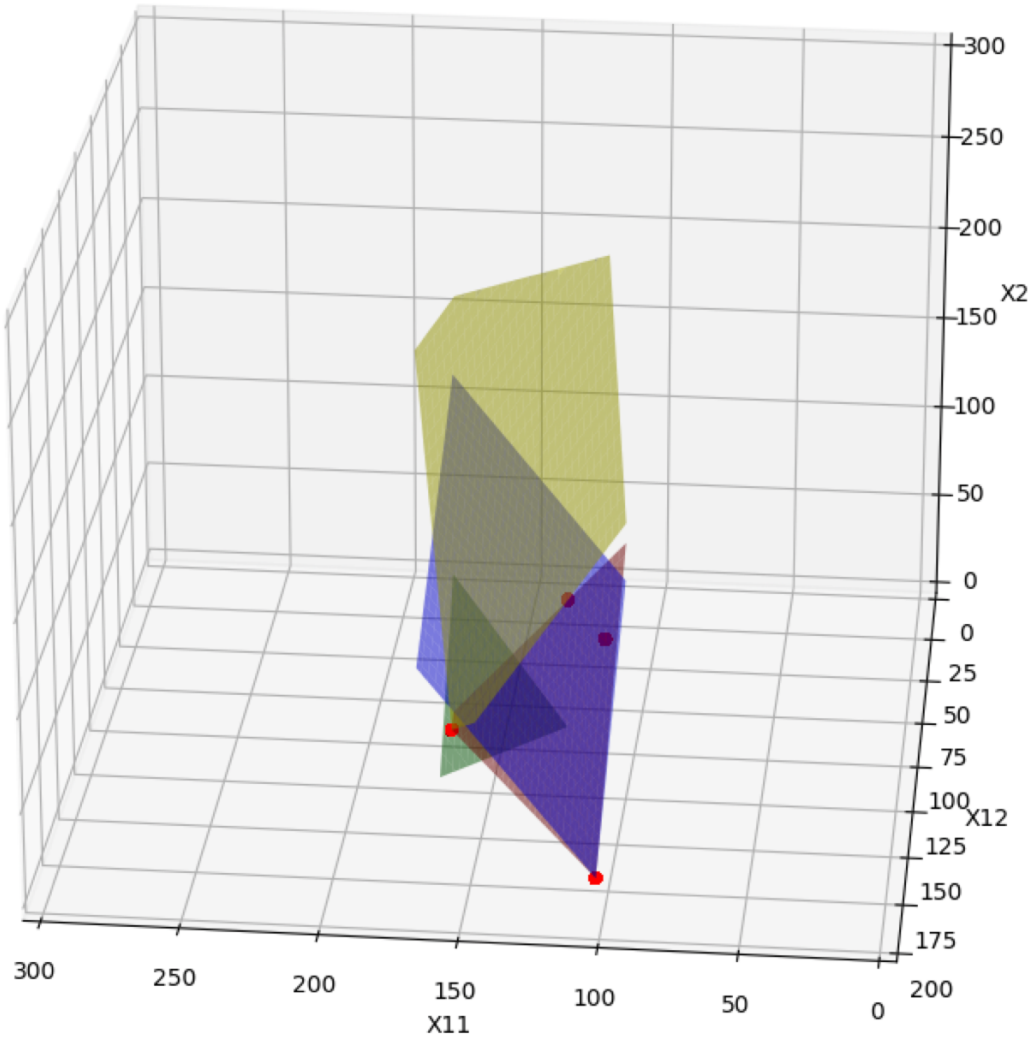

Figure 3.2. Optimal Solution of Problem (2.3) with Trapezoidal Membership Functions for the Coefficients of the Objective Function

For the full version of the code, you can refer to the file **optimal_solution_risk_analysis.py**.

Fuzzy Parameters of the Objective Function with Triangular Membership Functions

Triangular fuzziness (1.13) can be particularly useful for optimization tasks, especially when dealing with uncertainty and variability of parameters. Its

simplicity and intuitive linear rise or decline make it convenient for interpretation and practical applications. Since triangular fuzzy numbers can represent a varying degree of input data variability, they reflect greater randomness compared to trapezoidal numbers, where the membership function remains constant over a certain interval. Moreover, using triangular fuzzy numbers in mathematics can simplify calculations compared to other types of fuzzy numbers. Thus, in modeling various situations in optimization tasks that require consideration of ambiguity and variability of parameters, triangular fuzziness can offer valuable advantages.

We will modify the code from the file **optimal_solution_risk_analysis.py**, introducing a triangular membership function into the parameters c_{11}, c_{12} and c_2. *Fuzz.trimf()* , a function from the scikit-fuzzy library, is used to create a fuzzy triangular membership function. Each set is defined by three points that determine the shape of the triangle. We choose the following values: for c_{11}: $[0.3, 0.5, 0.7]$, for c_{12}: $[0.35, 0.55, 0.75]$ and for c_2: $[0.4, 0.6, 0.8]$. The values of these three parameters vary from 0.1 to 0.8 in steps of 0.05. This is implemented in the code as follows

```
C11_domain = np.arange(0.1, 0.8, 0.05)
C12_domain = np.arange(0.1, 0.8, 0.05)
C2_domain = np.arange(0.1, 0.8, 0.05)

C11_values = fuzz.trimf(C11_domain, [0.3, 0.5, 0.7])
C12_values = fuzz.trimf(C12_domain, [0.35, 0.55, 0.75])
C2_values = fuzz.trimf(C2_domain, [0.4, 0.6, 0.8])
```

Figure 3.3 demonstrates how fuzziness affects the parameters of the objective function coefficients. It displays the distribution of values for each parameter considering the introduced fuzziness.

After iterating through all possible combinations of parameters c_{11}, c_{12} and c_2, and after solving the optimization problem, the code generates a table (Table 3.2).

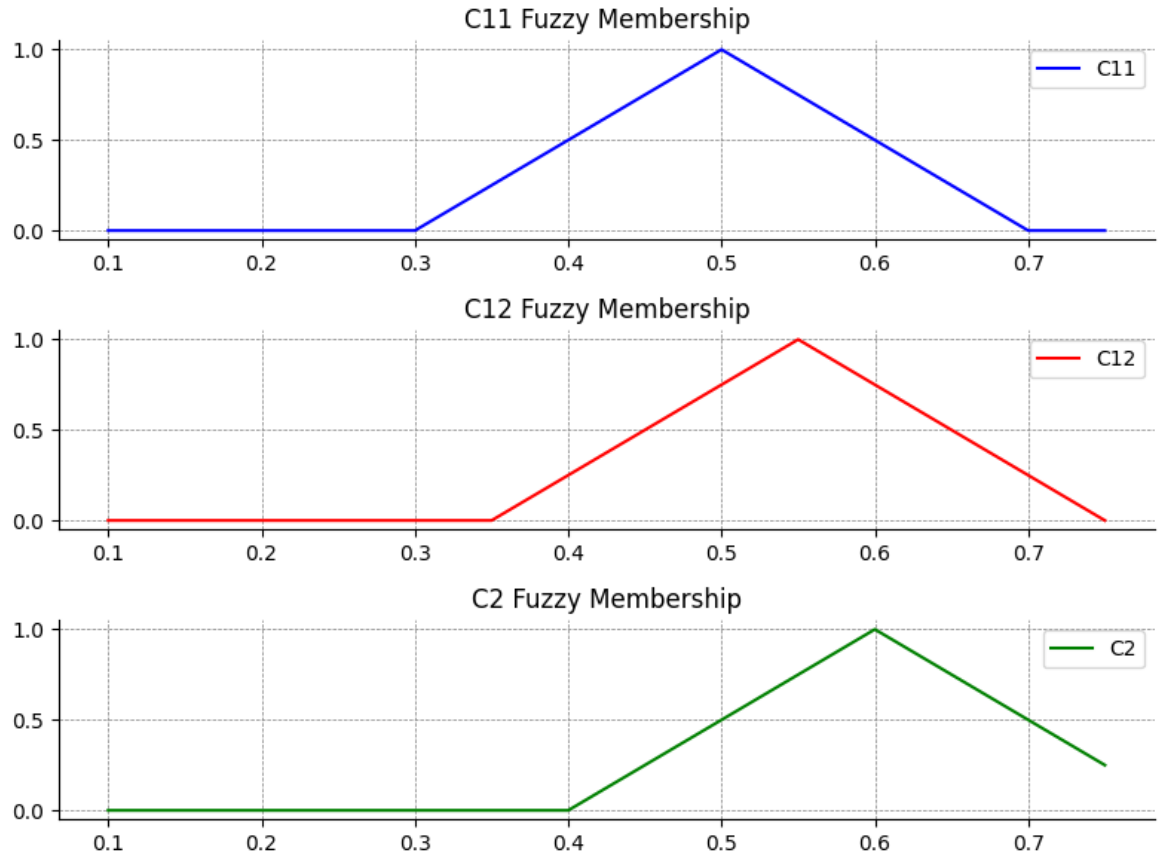

Figure 3.3. Triangular Membership Functions for the Coefficients of the Objective Function

As noted, compared to trapezoidal fuzziness, triangular fuzziness is more random, which is why there is only one of the 2669 combinations with a risk assessment (degree of membership) equal to 1.

Table 3.2

	C11	C12	C2	Optimal Solut	Chance	x_values
1	0.5	0.55	0.6	11.84	1.00	[109.48905109 113.13868613 69.34306569]
2	0.55	0.55	0.55	11.84	0.75	[109.48905109 113.13868613 69.34306569]
3	0.45	0.6	0.6	12.92	0.75	[109.48905109 113.13868613 69.34306569]
4	0.45	0.6	0.55	12.92	0.75	[109.48905109 113.13868613 69.34306569]
5	0.55	0.55	0.6	11.84	0.75	[109.48905109 113.13868613 69.34306569]
6	0.55	0.55	0.65	11.84	0.75	[109.48905109 113.13868613 69.34306569]
7	0.5	0.5	0.55	10.77	0.75	[109.48905109 113.13868613 69.34306569]
8	0.55	0.6	0.65	12.92	0.75	[109.48905109 113.13868613 69.34306569]
9	0.55	0.6	0.6	12.92	0.75	[109.48905109 113.13868613 69.34306569]
10	0.5	0.5	0.65	10.77	0.75	[109.48905109 113.13868613 69.34306569]
11	0.45	0.55	0.65	11.84	0.75	[109.48905109 113.13868613 69.34306569]
12	0.45	0.55	0.6	11.84	0.75	[109.48905109 113.13868613 69.34306569]
13	0.45	0.55	0.55	11.84	0.75	[109.48905109 113.13868613 69.34306569]
14	0.45	0.5	0.55	10.77	0.75	[109.48905109 113.13868613 69.34306569]
15	0.45	0.5	0.6	10.77	0.75	[109.48905109 113.13868613 69.34306569]
16	0.45	0.5	0.65	10.77	0.75	[109.48905109 113.13868613 69.34306569]
17	0.5	0.6	0.65	12.92	0.75	[109.48905109 113.13868613 69.34306569]
18	0.5	0.6	0.6	12.92	0.75	[109.48905109 113.13868613 69.34306569]
19	0.5	0.6	0.55	12.92	0.75	[109.48905109 113.13868613 69.34306569]
20	0.55	0.6	0.55	12.92	0.75	[109.48905109 113.13868613 69.34306569]
21	0.5	0.55	0.65	11.84	0.75	[109.48905109 113.13868613 69.34306569]
22	0.45	0.6	0.65	12.92	0.75	[109.48905109 113.13868613 69.34306569]
23	0.5	0.5	0.6	10.77	0.75	[109.48905109 113.13868613 69.34306569]
24	0.5	0.55	0.55	11.84	0.75	[109.48905109 113.13868613 69.34306569]
25	0.55	0.5	0.55	10.77	0.75	[109.48905109 113.13868613 69.34306569]
26	0.55	0.5	0.65	10.77	0.75	[109.48905109 113.13868613 69.34306569]
27	0.55	0.5	0.6	10.77	0.75	[109.48905109 113.13868613 69.34306569]
28	0.6	0.45	0.55	9.69	0.50	[109.48905109 113.13868613 69.34306569]
29	0.4	0.6	0.5	15.00	0.50	[104.65116279 174.41860465 0.]
30	0.4	0.6	0.55	12.92	0.50	[109.48905109 113.13868613 69.34306569]

Fuzzy Parameters of the Objective Function with Bell-Shaped Membership Functions

Bell-shaped membership functions, such as Gaussian (1.14) and sigmoidal (1.15), provide additional flexibility in modeling fuzziness in optimization tasks. These functions can be particularly useful when dealing with complex or high-dimensional data, where fuzziness and variability can be heterogeneous and vary across a wide range.

The Gaussian membership function can be advantageous for modeling situations where central values are most probable, and values deviating from the mean are less likely. The Gaussian function has a smooth, symmetrical profile, making it ideal for representing such scenarios.

The sigmoidal membership function can be useful for modeling situations where it is necessary to differentiate between two categories or states. This can be particularly valuable in solving binary optimization problems.

Unlike trapezoidal and triangular functions, which are often used to represent small and relatively homogeneous uncertainties, Gaussian and sigmoidal functions provide greater flexibility and precision in depicting more complex or distributed fuzzy scenarios.

First, we will introduce the Gaussian function into the parameters of the objective function of problem (2.3) c_{11}, c_{12} and c_2. For this purpose, we will use the **fuzz.gaussmf()** function from the scikit-fuzzy library. Here, it is necessary to specify two values that will correspond to the center of the Gaussian curve and its width. For example, we might choose the following values: for c_{11}: 0.4 and 0.6; for c_{12}: 0.5 for 0.7; and, finally, for c_2: 0.3 and 0.8. The values of these three parameters vary from 0.1 to 0.8 in steps of 0.05. This is implemented in the code as follows:

```
C11_domain = np.arange(0.1, 0.8, 0.05)
C12_domain = np.arange(0.1, 0.8, 0.05)
C2_domain = np.arange(0.1, 0.8, 0.05)

C11_values = fuzz.gaussmf(C11_domain, 0.4, 0.6)
C12_values = fuzz.gaussmf(C12_domain, 0.5, 0.7)
C2_values = fuzz.gaussmf(C2_domain, 0.3, 0.8)
```

Figure 3.4 illustrates how uncertainty affects the relevant parameters.

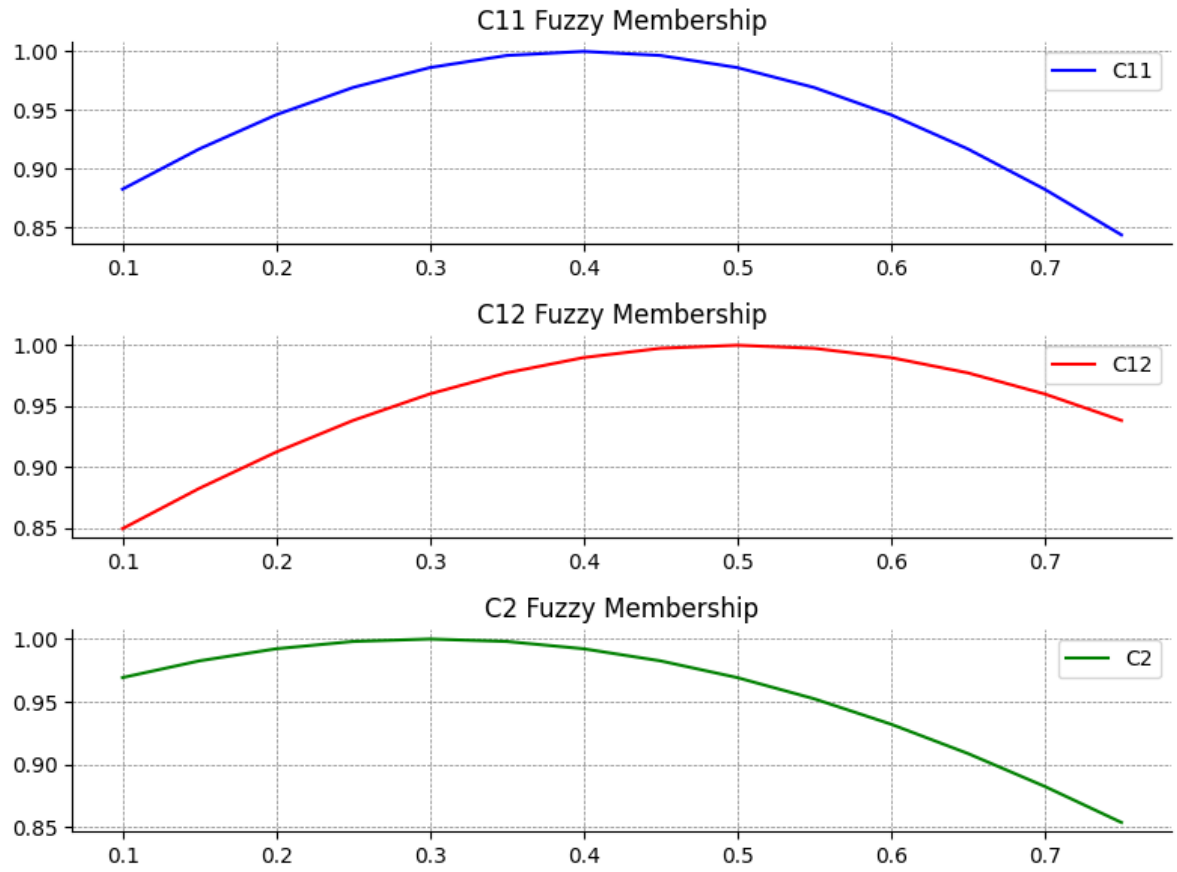

Figure 3.4. Gaussian Membership Functions of the Coefficients of the Objective Function for Problem (2.3)

After iterating through all possible combinations of the parameters c_{11}, c_{12} and c_2, and solving the optimization problem, the code generates the corresponding table (Table 3.3).

Analysis of Table 3.3 reveals that the risk assessment is "smooth." The risk assessment for 2669 combinations ranges from 0.84 to 1, with no step greater than 0.01.

Table 3.3

	C11	C12	C2	Optimal Solut	Chance	x_values
1	0.35	0.55	0.25	27.56	1.00	[104.65116279 174.41860465 0.]
2	0.4	0.5	0.25	23.55	1.00	[104.65116279 174.41860465 0.]
3	0.45	0.5	0.3	20.23	1.00	[104.65116279 174.41860465 0.]
4	0.4	0.55	0.35	20.93	1.00	[104.65116279 174.41860465 0.]
5	0.4	0.5	0.3	20.23	1.00	[104.65116279 174.41860465 0.]
6	0.45	0.45	0.3	16.22	1.00	[104.65116279 174.41860465 0.]
7	0.35	0.5	0.35	16.92	1.00	[104.65116279 174.41860465 0.]
8	0.35	0.55	0.3	24.24	1.00	[104.65116279 174.41860465 0.]
9	0.4	0.45	0.3	16.22	1.00	[104.65116279 174.41860465 0.]
10	0.45	0.5	0.25	23.55	1.00	[104.65116279 174.41860465 0.]
11	0.4	0.45	0.35	12.91	1.00	[104.65116279 174.41860465 0.]
12	0.45	0.45	0.25	19.53	1.00	[104.65116279 174.41860465 0.]
13	0.45	0.5	0.35	16.92	1.00	[104.65116279 174.41860465 0.]
14	0.35	0.45	0.35	12.91	1.00	[104.65116279 174.41860465 0.]
15	0.45	0.45	0.35	12.91	1.00	[104.65116279 174.41860465 0.]
16	0.4	0.55	0.25	27.56	1.00	[104.65116279 174.41860465 0.]
17	0.35	0.45	0.3	16.22	1.00	[104.65116279 174.41860465 0.]
18	0.35	0.55	0.35	20.93	1.00	[104.65116279 174.41860465 0.]
19	0.4	0.55	0.3	24.24	1.00	[104.65116279 174.41860465 0.]
20	0.35	0.5	0.25	23.55	1.00	[104.65116279 174.41860465 0.]
21	0.35	0.45	0.25	19.53	1.00	[104.65116279 174.41860465 0.]
22	0.4	0.5	0.35	16.92	1.00	[104.65116279 174.41860465 0.]
23	0.4	0.45	0.25	19.53	1.00	[104.65116279 174.41860465 0.]
24	0.35	0.5	0.3	20.23	1.00	[104.65116279 174.41860465 0.]
25	0.45	0.55	0.35	20.93	1.00	[104.65116279 174.41860465 0.]
26	0.45	0.55	0.3	24.24	1.00	[104.65116279 174.41860465 0.]
27	0.45	0.55	0.25	27.56	1.00	[104.65116279 174.41860465 0.]
28	0.3	0.6	0.3	28.26	0.99	[104.65116279 174.41860465 0.]
29	0.35	0.55	0.4	17.62	0.99	[104.65116279 174.41860465 0.]
30	0.35	0.6	0.25	31.57	0.99	[104.65116279 174.41860465 0.]

The next step involves introducing the sigmoid membership function into the parameters c_{11}, c_{12} and c_2. For this, we will use the **fuzz.sigmf()** function from the scikit-fuzzy library. The **fuzz.sigmf()** function requires values that define how quickly the function transitions from minimum to maximum, and a value that determines the "center" of the curve, i.e., the values of the parameters c_{11}, c_{12} and c_2 at which the respective function reaches its median value. We define these values as follows: c_{11}: 30 and 0.3; c_{12}: 30 and 0.45 and c_2: 30 and 0.5. The values of these three parameters vary from 0.1 to 0.8 in steps of 0.05. This is implemented in the code as follows:

```
C11_domain = np.arange(0.1, 0.8, 0.05)
C12_domain = np.arange(0.1, 0.8, 0.05)
C2_domain = np.arange(0.1, 0.8, 0.05)

C11_values = fuzz.sigmf(C11_domain, 0.3, 30)
C12_values = fuzz.sigmf(C12_domain, 0.45, 30)
C2_values = fuzz.sigmf(C2_domain, 0.5, 30)
```

Figure 3.5 shows how fuzziness impacts the parameters of the objective function coefficients.

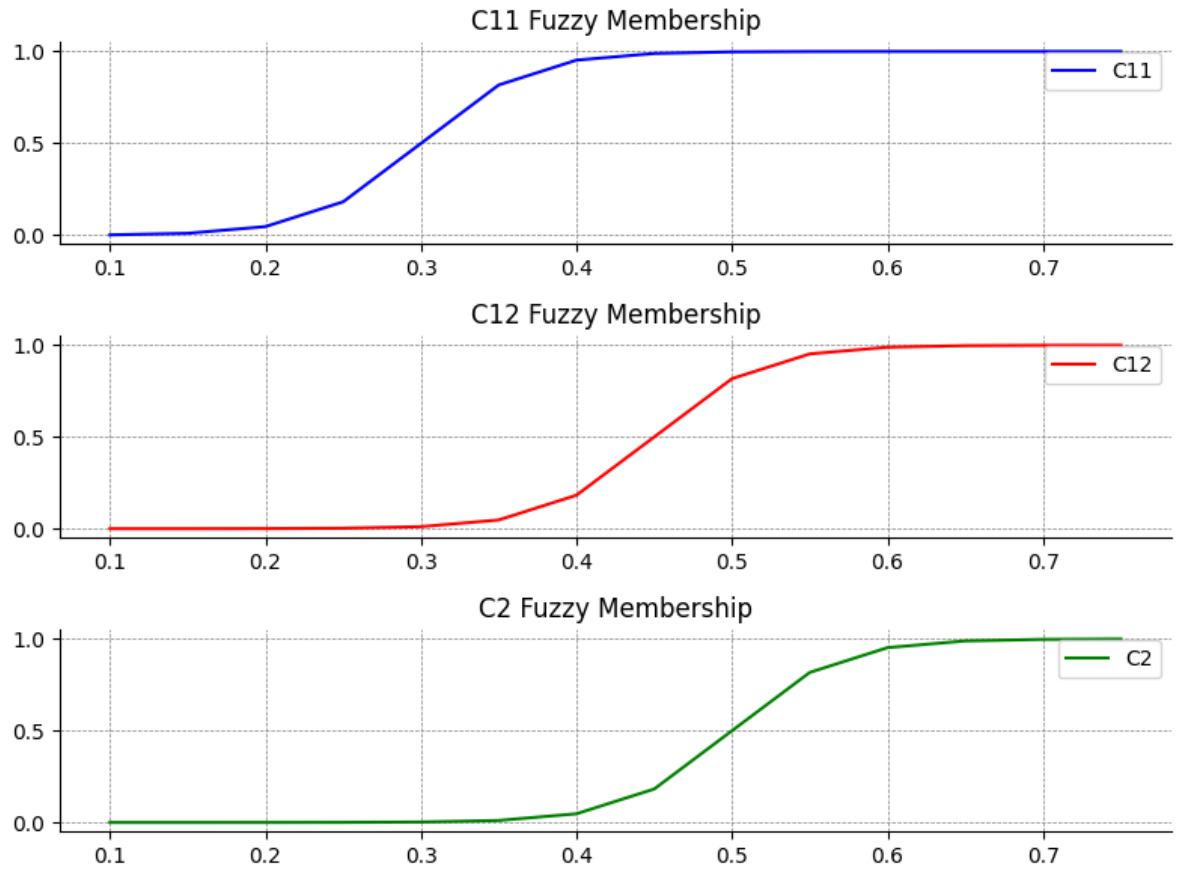

Figure 3.5. Sigmoid Membership Functions for Coefficients c_{11}, c_{12} and c_2

After iterating through all possible combinations of parameters c_{11}, c_{12} and c_2 and solving the optimization problem (2.3), the code generates information into the appropriate table (Table 3.4).

Analysis of Table 3.4 indicates that the risk assessment values "smoothly" transition from 0 to 1 in all 2669 combinations, allowing for a more precise risk assessment.

Table 3.4

	C11	C12	C2	Optimal Solut	Chance	x_values
1	0.75	0.75	0.75	16.15	1.00	[109.48905109 113.13868613 69.34306569]
2	0.75	0.65	0.75	14.00	1.00	[109.48905109 113.13868613 69.34306569]
3	0.55	0.65	0.75	14.00	1.00	[109.48905109 113.13868613 69.34306569]
4	0.6	0.75	0.75	16.15	1.00	[109.48905109 113.13868613 69.34306569]
5	0.65	0.7	0.7	15.07	1.00	[109.48905109 113.13868613 69.34306569]
6	0.65	0.7	0.75	15.07	1.00	[109.48905109 113.13868613 69.34306569]
7	0.5	0.75	0.75	16.15	1.00	[109.48905109 113.13868613 69.34306569]
8	0.7	0.65	0.7	14.00	1.00	[109.48905109 113.13868613 69.34306569]
9	0.7	0.65	0.75	14.00	1.00	[109.48905109 113.13868613 69.34306569]
10	0.55	0.7	0.7	15.07	1.00	[109.48905109 113.13868613 69.34306569]
11	0.55	0.7	0.75	15.07	1.00	[109.48905109 113.13868613 69.34306569]
12	0.7	0.7	0.7	15.07	1.00	[109.48905109 113.13868613 69.34306569]
13	0.7	0.7	0.75	15.07	1.00	[109.48905109 113.13868613 69.34306569]
14	0.7	0.75	0.7	16.15	1.00	[109.48905109 113.13868613 69.34306569]
15	0.7	0.75	0.75	16.15	1.00	[109.48905109 113.13868613 69.34306569]
16	0.6	0.7	0.75	15.07	1.00	[109.48905109 113.13868613 69.34306569]
17	0.6	0.7	0.7	15.07	1.00	[109.48905109 113.13868613 69.34306569]
18	0.6	0.65	0.7	14.00	1.00	[109.48905109 113.13868613 69.34306569]
19	0.6	0.65	0.75	14.00	1.00	[109.48905109 113.13868613 69.34306569]
20	0.5	0.65	0.7	14.00	1.00	[109.48905109 113.13868613 69.34306569]
21	0.5	0.65	0.75	14.00	1.00	[109.48905109 113.13868613 69.34306569]
22	0.55	0.75	0.7	16.15	1.00	[109.48905109 113.13868613 69.34306569]
23	0.55	0.75	0.75	16.15	1.00	[109.48905109 113.13868613 69.34306569]
24	0.65	0.75	0.7	16.15	1.00	[109.48905109 113.13868613 69.34306569]
25	0.5	0.7	0.7	15.07	1.00	[109.48905109 113.13868613 69.34306569]
26	0.5	0.7	0.75	15.07	1.00	[109.48905109 113.13868613 69.34306569]
27	0.65	0.75	0.75	16.15	1.00	[109.48905109 113.13868613 69.34306569]
28	0.75	0.65	0.7	14.00	1.00	[109.48905109 113.13868613 69.34306569]
29	0.5	0.75	0.7	16.15	1.00	[109.48905109 113.13868613 69.34306569]
30	0.75	0.7	0.75	15.07	1.00	[109.48905109 113.13868613 69.34306569]