COMPUTATIONAL STUDY OF FUZZY NEUTROSOPHIC SOFT MATRICES IN PYTHON: CONSISTENCY AND WEAK TRANSITIVITY

M Kavitha [1], Sive M [2] , P Murugadas [3] and K Rameshware 4

1 **Department of Mathematics, Chennai Institute of Technology, Chennai, India** 1; kavithakathir3@gmail.com

[2] **Department of Mathematics, St. Joseph's Institute of Technology, Chennai, India**; sivam@stjosephstechnology.ac.in

3 Department of Mathematics, Annamalai University, Annamalai Nagar, India; bodi_muruga@yahoo.com

4 Department of Mathematics, Bharath Institute of Higher Education and Research, Chennai;

krameshmath@gmail.com

***** Correspondence: kavithakathir3@gmail.com

Abstract: In this study, we introduce a novel framework for defining and analyzing two specific types of Fuzzy Neutrosophic Soft Matrices (FNSMs): consistent and weakly transitive. These matrix classes are modelled and assessed using Python-based computational techniques. We establish that both types exhibit controllability and present a Python-compatible formulation for deriving the canonical form of a Weakly Transitive FNSM (WT-FNSM). Fundamental algebraic and structural properties such as nilpotency, symmetry, transitivity, and weak transitivity are investigated through programmatic simulations. Additionally, we explore the connection between consistent and weakly transitive FNSMs and finite fuzzy neutrosophic relations, emphasizing their applicability in various practical and academic domains. The controllability of WT-FNSMs is further validated through algorithmic evaluation. To support the theoretical results, appropriate Python-based examples and simulations are provided. A key contribution of this work is a versatile Python tool designed for FNSMs, which is also adaptable for use with fuzzy matrices, intuitionistic fuzzy matrices, and fuzzy neutrosophic matrices.

Keywords: Fuzzy Neutrosophic Soft Matrix (FNSM), Nilpotent Fuzzy Neutrosophic Soft Matrix (NFNSM), Transitive Fuzzy Neutrosophic Soft Matrix (T FNSM), Controllable Fuzzy Neutrosophic Soft Matrix CFNSM, Python

---

1. Introduction

The notion of Fuzzy Set (FS) and its logic are investigated and discussed by Zadeh [1]. After that, Atanassav investigated the Intuitionistic Fuzzy Set (IFS) [2]. Neutrosophy has extend the grounds for a total family of new mathematical estimations. It is one of the non-classical sets, like fuzzy, nano, soft, permutation sets and so on. The Neutrosophic Set (NS) was presented by Smarandache [3] and expounded, (NS) is a popularization of (IFS) in intuitionistic fuzzy set. Maji [4] introduced the concept of neutrosophic soft set and established some operations on these sets. Broumi and Smarandache [5, 6] combined the intuitionistic neutrosophic and soft set which lead to a new mathematical model called intuitionistic neutrosophic soft set. They studied the notions of intuitionistic neutrosophic soft set union, intuitionistic neutrosophic soft set intersection, complement of intuitionistic neutrosophic soft set and several other properties of intuitionistic neutrosophic soft set along with exam plus and proofs of certain results. Recently, Deli [7] introduced the interval valued neutrosophic soft set as a combination of interval neutrosophic set and soft set. The concepts of consistent and weak transitive fuzzy matrices was introduced by Emam [8] as a property of finite fuzzy preference relations. The notions of Fuzzy Neutrosophic Soft Matrix (FNSM) and used them in decision making problems proposed by Arockiarani and Sumathi [9,10]. The Priodicity of Interval values, on powers of matrices and convergence of matrices, Solvable linear equation, Eigen space by usig the notion of Fuzzy Neutrosophic Soft Matrices are Introduced Kavitha et.al.,[11,12,13]. The idea of Monotone Fuzzy Neutrosophic Soft Eigenspace Structure in Max-Min Algebra and Convergence of Fuzzy Neutrosophic Soft Circulant Matrices are proposed by Murugadas et.al.,[14,15,16]. Uma et.al., [17] presented the concepts of Fuzzy Neutrosophic Soft Matrices of Type-1 and Type-2.

Smarandache et.al introduced [18] the concepts of a python tool for Implementations on Bipolar Neutrosophic Matrices. They have established some operations, especially the composition is a challenging algorithm in terms of coding because there are so many nested lists to manipulate. Gayathri et al. [19] presented the ideas of neutrosophic vague measures using Python. Their work explores the application of Python in analyzing neutrosophic vague measures and investigates various types of measures within neutrosophic vague sets, supported by illustrative examples. In recent developments, some researchers have created Python programs to perform operations involving neutrosophic numbers. However, these implementations are limited in scope they do not support computations involving neutrosophic matrices. To the best of our knowledge, there has been no comprehensive effort so far to implement Python code that handles operations on single-valued neutrosophic matrices (SVNMs) or bipolar neutrosophic matrices (BNMs). This reveals a significant research gap, motivating the current study.

The present work aims to fill this gap by exploring various operations on FNS and providing corresponding Python implementations using different FNSMs. The structure of the manuscript is organized as follows: Section 2 outlines key preliminary definitions. Section 3 discusses essential ideas pertaining to FNSMs and demonstrates how to perform matrix operations using the Python programming language. Section 4 is dedicated to developing Python scripts for handling FNSMs, complemented by a worked-out numerical example. The final section, Section 5, summarizes the findings and suggests directions for future research.

2. Preliminaries

Definition 1: [3]

A Fuzzy Neutrosophic Set (FNS) A on the universe of discourse X is defined as A = {x,$\langle A^T(x), A^I(x), A^F(x)\rangle$, x ∈ X} , where T , I, F : X → [0, 1] and

$0 \leq A^T(x) + A^I(x) + A^F(x) \leq 3.$

Definition 2: [5]

Let U be the initial universal set and E be a set of parameters. Consider a non-empty set A, A ⊂ E. Let P(U) denotes the set of all FNSs of U. The collection (F, A) is termed to be the FNSS over U, where F is a mapping given by F : A → P(U). Here after we simply consider A as FNSS over U instead of (F, A).

Definition 3: [6]

Let U = $\{c_1, c_2, \ldots, c_m\}$ be the universal set and E be the set of parameters given by E = $\{e_1, e_2, \ldots, e_m\}$. Let A ⊂ E. A pair (F, A) be a FNSS over U. Then the subset of U × E is defined by R_A = {(u, e); e ∈ A, u ∈ F_A(e)} which is called a relation form of (F_A, E). The membership function, indeterminacy membership function and non membership function are written

T_{R_A} : U × E → [0, 1], I_{R_A} : U × E → [0, 1] and F_{R_A} : U × E → [0, 1] where

T_{R_A} (u, e) ∈ [0, 1], I_{R_A} (u, e) ∈ [0, 1] and F_{R_A} (u, e) ∈ [0, 1] are the membership value, indeterminacy value and non membership value respectively of u ∈ U for each e ∈ E

If $[\langle T_{ij}, I_{ij}, F_{ij}\rangle] = [T_{ij}(u_i, e_j),\ I_{ij}(u_i, e_j),\ F_{ij}(u_i, e_j)\ ]$, we define a matrix

$$[\langle T_{ij}, I_{ij}, F_{ij}\rangle]_{m\times n} = \begin{bmatrix} \langle T_{11}, I_{11}, F_{11}\rangle & \ldots & \langle T_{1n}, I_{1n}, F_{1n}\rangle \\ \langle T_{21}, I_{21}, F_{21}\rangle & \ldots & \langle T_{2n}, I_{2n}, F_{2n}\rangle \\ \ldots & \ldots & \ldots \\ \langle T_{m1}, I_{m1}, F_{m1}\rangle & \ldots & \langle T_{mn}, I_{mn}, F_{mn}\rangle \end{bmatrix}$$

which is called an m × n FNSM of the FNSS (F_A, E) over U.

The set of all FNSMs of order $m \times n$ is denoted by $\aleph_{m\times n}$ and $\aleph_n$ represent the set of all FNSM of order $n \times n$.

Definition 4: [17]

Let A = $\left(\langle a_{ij}^T,\ a_{ij}^I,\ a_{ij}^F\rangle\right)$, B = $\left(\langle b_{ij}^T,\ b_{ij}^I,\ b_{ij}^F\rangle\right) \in \aleph_{m\times n}$. The component wise addition and component wise multiplication is defined as

$A \oplus B = (\sup\{a_{ij}^T, b_{ij}^T\}, \sup\{a_{ij}^I, b_{ij}^I\}, \inf\{a_{ij}^F, b_{ij}^F\})$

$A \otimes B = (\inf\{a_{ij}^T, b_{ij}^T\}, \inf\{a_{ij}^I, b_{ij}^I\}, \sup\{a_{ij}^F, b_{ij}^F\})$

Definition 5: [17] Let $A \in \aleph_{m\times n}$, $B \in \aleph_{n\times p}$, the composition of A and B is defined as

$$A \circ B = \left(\left(\sum_{k=1}^n a_{ik}^T \wedge b_{kj}^T\right), \left(\sum_{k=1}^n a_{ik}^I \wedge b_{kj}^I\right), \left(\prod_{k=1}^n a_{ik}^F \vee b_{kj}^F\right)\right),$$

equivalently we can write the same as

$$A \circ B = \left(\bigvee_{k=1}^n \left(a_{ik}^T \wedge b_{kj}^T\right), \bigvee_{k=1}^n \left(a_{ik}^I \wedge b_{kj}^I\right), \bigwedge_{k=1}^n \left(a_{ik}^F \vee b_{kj}^F\right)\right).$$

The product A ◦ B is defined if and only if the number of columns of A is same as the number of rows of B. Then A and B are said to be conformable for multiplication. We shall use AB instead of A ◦ B.

Where $\left(\sum_{k=1}^n a_{ik}^T \wedge b_{kj}^T\right)$ means max-min operation and

$\left(\prod_{k=1}^n a_{ik}^F \vee b_{kj}^F\right)$ means min-max operation.

3. A Python Approach to Fuzzy Neutrosophic Soft Matrix Processing

We examine the properties of consistency and weak transitivity in fuzzy neutrosophic soft matrices and construct their canonical weak transitive form accordingly. The Python code for inputting these matrices is described as follows.

Let R = $\langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle$, and S= $\langle s_{ij}^T, s_{ij}^I, s_{ij}^F \rangle$ be the fuzzy neutrosophic soft matrices

Then the following operations are defined.

- R⊲ S iff $\langle \boldsymbol{r_{ij}^T}, \boldsymbol{r_{ij}^I}, \boldsymbol{r_{ij}^F} \rangle$= (0,0,1) ⇒ $\langle s_{ij}^T, s_{ij}^I, s_{ij}^F \rangle$= (0,0,1) for every i ≤ m and for every j ≤ n, ⊲ it is denoted by relations are reflexive and transitive.
- $\langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \ominus \langle r_{ji}^T, r_{ji}^I, r_{ji}^F \rangle = \begin{cases} \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \text{ if } \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle > \langle r_{ji}^T, r_{ji}^I, r_{ji}^F \rangle \\ \langle 0,0,1 \rangle \quad \text{if} \quad \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \leq \langle r_{ji}^T, r_{ji}^I, r_{ji}^F \rangle \end{cases}$
- $\Delta \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle = \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \ominus \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle^t$

The following functions define the Transitive, Nilpotent, Consistent, Weak Transitive, of FNSMs.

To generate the Python program for deciding for a given the matrix is Fuzzy neutrosophic soft matrix or, simple call of the function FNSM Checking (Transitive) is defined as follow:

Definition 3.1: Let i, j, k $\leq n$ and let R = $\langle r_{ij}^{T}, r_{ij}^{I}, r_{ij}^{F} \rangle$ FNSM . Then R is called

- Transitive iff $R^2 \leq R$ (Checking the matrix is TFNSM or not)

```
def max_max_min_composition(F, B):
    n = len(F)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            temp_list = []
            for k in range(n):
                T = max(F[i][k][2], B[k][j][2])
                I = max(F[i][k][1], B[k][j][1])
                F = min(F[i][k][0], B[k][j][0])
                temp_list.append((F, I, T))
            result[i][j] = max(temp_list, key=lambda x: x[0])
    return result
def is_transitive(R):
    R2 = max_max_min_composition(R, R)
    for i in range(len(R)):
        for j in range(len(R)):
            T2, I2, F2 = R2[i][j]
            F, I, T = R[i][j]
            if not (T2 <= F and I2 >= I and F2 >= T):
                return False
    return True
# Modified Matrix R to be transitive
R = [
    [(0.6, 0.3, 0.4), (0.5, 0.3, 0.3), (0.5, 0.2, 0.4)],
    [(0.5, 0.4, 0.4), (0.6, 0.3, 0.3), (0.5, 0.2, 0.3)],
    [(0.5, 0.2, 0.5), (0.5, 0.3, 0.3), (0.6, 0.4, 0.2)] ]
res = is_transitive(R)
if res:
    print("The matrix is Transitive")
else:
    print("The matrix is not Transitive")
```

- In the above example we evaluate the checking the matrix R is Transitive or not of order 3X3:

- The FNSM R can be inputted in python environment like this:

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
The matrix is Transitive
```

3.2. Nilpotent of fuzzy neutrosophic soft matrix

- Nilpotent iff $R^n = \langle 0,0,1 \rangle$

The Nilpotent of a fuzzy neutrosophic soft matrix can be determined in Python by invoking the function FNSM NilpotentOf(), which is defined below

```
def max_max_min_composition(A, B):
    n = len(A)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            temp_list = []
            for k in range(n):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                temp_list.append((T, I, F))
            result[i][j] = max(temp_list, key=lambda x: x[0])   # max T
    return result
def is_null_matrix(M):
    return all(cell == (0, 0, 1) for row in M for cell in row)


def print_matrix(M):
    for row in M:
        print(["({:.2f},{:.2f},{:.2f})".format(*cell) for cell in row])
    print()
def is_nilpotent(R, max_power):
    result = R
    for i in range(1, max_power + 1):
        result = max_max_min_composition(result, R)
        print(f"R^{i+1}:")
        print_matrix(result)
        if is_null_matrix(result):
            print(f"✓ Matrix is nilpotent at index {i+1}.")
```

```
            return True
    print(f"✗ Matrix is not nilpotent up to index {max_power + 1}.")
    return False
R = [
    [(0.0, 0.0, 1.0), (0.7, 0.6, 0.3), (0.5, 0.4, 0.5), (0.4, 0.3, 0.6)],
    [(0.0, 0.0, 1.0), (0.0, 0.0, 1.0), (0.0, 0.0, 1.0), (0.0, 0.0, 1.0)],
    [(0.0, 0.0, 1.0), (0.7, 0.6, 0.3), (0.0, 0.0, 1.0), (0.9, 0.8, 0.1)],
    [(0.0, 0.0, 1.0), (0.0, 0.0, 1.0), (0.0, 0.0, 1.0), (0.0, 0.0, 1.0)]]
print("Is R consistent under max-max-min operation:", is_nilpotent_max_op(R))
```

The FNSM R can be entered into the Python environment in the following format, and the corresponding output will display its structured matrix form with truth, indeterminacy, and falsity values."

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Is R nilpotent of index 3-> True
```

3.3. Identifying Consistent of fuzzy neutrosophic soft matrix

Consistent iff $\langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \geq \langle r_{ji}^T, r_{ji}^I, r_{ji}^F \rangle$ and $\langle r_{jk}^T, r_{jk}^I, r_{jk}^F \rangle \geq \langle r_{kj}^T, r_{kj}^I, r_{kj}^F \rangle \Rightarrow$

$$\langle r_{ik}^T, r_{ik}^I, r_{ik}^F \rangle \geq \langle r_{ki}^T, r_{ki}^I, r_{ki}^F \rangle$$

To generate the Python program for finding consistent of fuzzy neutrosophic soft matrix, simple call of the function FNSMconsistentOf() is defined as follow:

```
    def max_max_min_composition(R):
        n = len(R)
        result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
        for i in range(n):
            for j in range(n):
                temp_list = []
                for k in range(n):
                    T = max(R[i][k][0], R[k][j][0])
                    I = max(R[i][k][1], R[k][j][1])
                    F = min(R[i][k][2], R[k][j][2])
                    temp_list.append((T, I, F))
                # Select the tuple with the highest T, lowest I, lowest F
                comp = max(temp_list, key=lambda x: (x[0], -x[1], -x[2]))
```

```
                    # Ensure the result is componentwise less than or equal to R[i][j]
                    result[i][j] = (min(comp[0], R[i][j][0]),
                        min(comp[1], R[i][j][1]),
                        max(comp[2], R[i][j][2])   # Use max for F to ensure F stays >= R[i][j][2] )
        return result
    def is_componentwise_less_equal(a, b):
        return all(x <= y for x, y in zip(a, b))

    def is_consistent_max_op(R):
        R2 = max_max_min_composition(R)
        n = len(R)
        for i in range(n):
            for j in range(n):
                if not is_componentwise_less_equal(R2[i][j], R[i][j]):
                    return False
        return True
    # Given matrix R
    R = [
        [(1.0, 1.0, 0.0), (0.8, 0.7, 0.2), (0.6, 0.6, 0.3)],
        [(0.7, 0.6, 0.3), (1.0, 1.0, 0.0), (0.6, 0.5, 0.3)],
        [(0.4, 0.3, 0.5), (0.5, 0.4, 0.4), (1.0, 1.0, 0.0)]]

print("Is R consistent under max-max-min operation:", is_consistent_max_op(R))
```

The FNSM *R* can be entered into the Python environment in the following way

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Is R consistent under max-max-min operation: True
```

3.4. Establishing weak transitive of fuzzy neutrosophic soft matrix

- Weak transitive iff $\langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle > \langle r_{ji}^T, r_{ji}^I, r_{ji}^F \rangle$ and $\langle r_{jk}^T, r_{jk}^I, r_{jk}^F \rangle > \langle r_{kj}^T, r_{kj}^I, r_{kj}^F \rangle$

  $\Rightarrow \langle r_{ik}^T, r_{ik}^I, r_{ik}^F \rangle \geq \langle r_{ki}^T, r_{ki}^I, r_{ki}^F \rangle$

To compute the weak transitive form of a fuzzy neutrosophic soft matrix in Python, simply call the function FNSMweaktransitiveOf(), defined as follows

```
def max_max_min_composition(R):
    n = len(R)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
```

```
    for i in range(n):
        for j in range(n):
            temp = []
            for k in range(n):
                T = max(R[i][k][0], R[k][j][0])
                I = max(R[i][k][1], R[k][j][1])
                F = min(R[i][k][2], R[k][j][2])
                temp.append((T, I, F))
            result[i][j] = max(temp, key=lambda x: (x[0], -x[1], -x[2]))
    return result
def is_componentwise_strictly_greater(a, b):
    return all(x > y for x, y in zip(a, b))
def is_weak_transitive(R):
    R2 = max_max_min_composition(R)
    n = len(R)
    for i in range(n):
        for j in range(n):
            for k in range(n):
                if (is_componentwise_strictly_greater(R[i][j], R[j][i]) and
                    is_componentwise_strictly_greater(R[j][k], R[k][j])):
                    if not is_componentwise_strictly_greater(R2[i][k], R[k][i]):
                        return False
return True
R = [ [(0.8, 0.7, 0.2), (0.7, 0.6, 0.3), (0.9, 0.7, 0.1)],
    [(0.6, 0.5, 0.4), (0.9, 0.8, 0.2), (0.8, 0.6, 0.2)],
    [(0.5, 0.4, 0.5), (0.4, 0.3, 0.6), (0.9, 0.9, 0.1)]]
print("Is R weak transitive:", is_weak_transitive(R))
```

The FNSM R can be inputted in python environment like this

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Is R weak transitive: True
```

3.5. Evaluating controllable of fuzzy neutrosophic soft matrix

FNSM $R_{n\times n}$ is said to be controllable if there exists fuzzy neutrosophic soft permutation matrix P such that

F= $\langle f_{ij}^T, f_{ij}^I, f_{ij}^F \rangle = \langle p_{ij}^T, p_{ij}^I, p_{ij}^F \rangle \langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \langle p_{ij}^T, p_{ij}^I, p_{ij}^F \rangle^t$ *satisfies*

$\langle f_{ij}^T, f_{ij}^I, f_{ij}^F \rangle \geq \langle r_{ji}^T, r_{ji}^I, r_{ji}^F \rangle$ for I > j. Then FNSM is called the canonical form of R.

To generate the Python program for finding controllable of fuzzy neutrosophic soft matrix, simple call of the function FNSM controllableOf() is defined as follow:

```
from itertools import permutations
def max_max_min_composition(P, R):
    n = len(P)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            candidates = []
            for k in range(n):
                T = max(P[i][k][0], R[k][j][0])
                I = max(P[i][k][1], R[k][j][1])
                F = min(P[i][k][2], R[k][j][2])
                candidates.append((T, I, F))
            result[i][j] = max(candidates, key=lambda x: (x[0], -x[1], -x[2]))
    return result
def generate_fnsm_permutation(n, perm):
    P = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        P[i][perm[i]] = (1, 0, 0)
    return P
def is_controllable(R, canonical_form):
    n = len(R)
    for perm in permutations(range(n)):
        P = generate_fnsm_permutation(n, perm)
        F = max_max_min_composition(P, R)
        if F == canonical_form:
            return True, perm
    return False, None
# Original matrix R
R = [ [(0.65, 0.3, 0.4), (0.55, 0.4, 0.5), (0.45, 0.5, 0.6), (0.35, 0.6, 0.7)],
    [(0.85, 0.1, 0.2), (0.75, 0.2, 0.3), (0.65, 0.3, 0.4), (0.55, 0.4, 0.5)],
    [(0.75, 0.2, 0.3), (0.65, 0.3, 0.4), (0.55, 0.4, 0.5), (0.45, 0.5, 0.6)],
    [(0.95, 0.05, 0.1), (0.85, 0.1, 0.2), (0.75, 0.2, 0.3), (0.65, 0.3, 0.4)]]
```

```
# Modified canonical form to match F with identity permutation
canonical = [
    [(1, 0.3, 0), (1, 0.4, 0), (1, 0.5, 0), (1, 0.6, 0)],
    [(1, 0.1, 0), (1, 0.2, 0), (1, 0.3, 0), (1, 0.4, 0)],
    [(1, 0.2, 0), (1, 0.3, 0), (1, 0.4, 0), (1, 0.5, 0)],
    [(1, 0.05, 0), (1, 0.1, 0), (1, 0.2, 0), (1, 0.3, 0)]]
# Run check
status, perm_used = is_controllable(R, canonical)
print("Is R controllable:", status)
if status:
    print("Permutation used:", perm_used)
```

The FNSM R can be inputted in the Python environment and its output observed as follows.

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Is R controllable: True
Permutation used: (0, 1, 2, 3)
```

4. Python-Based Proof and Canonical Construction of Consistent and Weakly Transitive Fuzzy Neutrosophic Soft Matrices

Hereafter, we prove some properties of consistency and weak transitivity in fuzzy neutrosophic soft matrices and construct their canonical weak transitive form using Python implementations.

Proposition 4.1. R is a consistent (weak transitive) if and only if ΔR is consistent (weak transitive) Python can also be used to implement functions that automatically test the consistency of the given FNSM.

```
def delta_matrix(R):
    n = len(R)
    D = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            T = max(0, R[i][j][0] - R[j][i][0])
            I = max(0, R[i][j][1] - R[j][i][1])
            F = min(0, R[i][j][2] - R[j][i][2])
            D[i][j] = (round(T, 2), round(I, 2), round(F, 2))
    return D
def is_consistent(R):
    def geq(a, b): return a[0] >= b[0] and a[1] >= b[1]   # compare T and I
    n = len(R)
```

```
    for i in range(n):
        for j in range(n):
            for k in range(n):
                if (R[i][j][0] >= R[j][i][0] and R[i][j][1] >= R[j][i][1]) and \
                    (R[j][k][0] >= R[k][j][0] and R[j][k][1] >= R[k][j][1]):
                    if not geq(R[i][k], R[k][i]):
                        return False
    return True
def is_weak_transitive(R):
    def gt(a, b): return all(x > y for x, y in zip(a, b))
    n = len(R)
    for i in range(n):
        for j in range(n):
            for k in range(n):
                if gt(R[i][j], R[j][i]) and gt(R[j][k], R[k][j]):
                    if not gt(R[i][k], R[k][i]):
                        return False
    return True
def print_matrix(name, M):
    print(f"\n{name}:")
    for row in M:
        print("    ".join(f"{t:.2f},{i:.2f},{f:.2f}" for t, i, f in row))

# Example FNSM matrix R
R = [[(0.7, 0.4, 0.3), (0.6, 0.3, 0.5), (0.8, 0.5, 0.2)],
    [(0.5, 0.3, 0.4), (0.9, 0.6, 0.2), (0.6, 0.4, 0.4)],
    [(0.3, 0.2, 0.6), (0.5, 0.3, 0.5), (0.9, 0.8, 0.1)]]
# Compute delta matrix ΔR
ΔR = delta_matrix(R)
# Display matrices
print_matrix("Original Matrix R", R)
print_matrix("Delta Matrix ΔR = R ⊖ Rt", ΔR)
# Property check
print("\nProperty Verification: R is consistent ⇔ ΔR is consistent")
print("R is consistent:", is_consistent(R))
print("ΔR is consistent:", is_consistent(ΔR))

print("\nProperty Verification: R is weak transitive ⇔ ΔR is weak transitive")
print("R is weak transitive:", is_weak_transitive(R))
```

```
print("ΔR is weak transitive:", is_weak_transitive(ΔR))
```

In Python, the FNSM R can be defined like this to generate the desired output

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =

Original Matrix R:
0.7,0.4,0.3  0.6,0.3,0.5  0.8,0.5,0.2
0.5,0.3,0.4  0.9,0.6,0.2  0.6,0.4,0.4
0.3,0.2,0.6  0.5,0.3,0.5  0.9,0.8,0.1

Delta Matrix ΔR = R ⊖ Rt:
0,0,0  0.1,0,0.1  0.5,0.3,0
0,0,0  0,0,0  0.1,0.1,0
0,0,0.4  0,0,0.1  0,0,0

Consistency and Weak Transitivity:
R is consistent: True
ΔR is consistent: True
R is weak transitive: True
ΔR is weak transitive: True
```

Proposition 4.2. Let A and B be two FNSMs of order m×n such that A ⊲ B. Then for any FNSM of order 1×m C we need CA ⊲ CB.

The difference CA ⊲ CB of a fuzzy neutrosophic soft matrix can be implemented in Python by defining the function FNSM Of() as shown.

```
def max_max_min_composition(A, B):
    n, m = len(A), len(B[0])
    result = [[(0, 0, 1) for _ in range(m)] for _ in range(n)]
    for i in range(n):
        for j in range(m):
            candidates = []
            for k in range(len(B)):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                candidates.append((T, I, F))
            result[i][j] = max(candidates, key=lambda x: (x[0], -x[1], -x[2]))
    return result
def less_than_equal_fuzzy(a, b):
    return a[0] <= b[0] and a[1] <= b[1] and a[2] >= b[2]

def matrix_leq(A, B):
    for i in range(len(A)):
```

```
            for j in range(len(A[0])):
                if not less_than_equal_fuzzy(A[i][j], B[i][j]):
                    return False
        return True
# Matrix A
A = [[(0,0,1), (0.3,0.4,0.7), (0.2,0.3,0.8)],
     [(0.7,0.8,0.3), (0.5,0.6,0.5), (0,0,1)],
     [(0,0,1), (0,0,1), (0.4,0.5,0.6)]]
# Matrix B
B = [[(0,0,1), (0.3,0.4,0.7), (0.2,0.3,0.8)],
     [(0.8,0.9,0.2), (0.6,0.7,0.4), (0,0,1)],
     [(0.2,0.3,0.8), (0,0,1), (0.9,0.8,0.1)]]
# Matrix C
C = [[(0.5,0.6,0.5), (0,0,1), (0.3,0.4,0.7)],
     [(0.5,0.6,0.5), (0,0,1), (0,0,1)],
     [(0.7,0.8,0.3), (0,0,1), (0,0,1)]]
# Compute CA and CB
CA = max_max_min_composition(C, A)
CB = max_max_min_composition(C, B)
# Print matrices
def print_matrix(M, name):
     print(f"{name} =")
     for row in M:
          print("    ", row)
     print()
print_matrix(CA, "CA")
print_matrix(CB, "CB")

# Check A ⊲ B and CA ⊲ CB
print("A ⊲ B:", matrix_leq(A, B))
print("CA ⊲ CB:", matrix_leq(CA, CB))
```

The following code shows how to input the controllable FNSM R in Python and display its output

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
CA =
   [(0.7, 0.8, 0.3), (0.5, 0.6, 0.5), (0.5, 0.6, 0.5)]
   [(0.7, 0.8, 0.3), (0.5, 0.6, 0.5), (0.5, 0.6, 0.5)]
   [(0.7, 0.8, 0.3), (0.7, 0.8, 0.3), (0.7, 0.8, 0.3)]

CB =
   [(0.8, 0.9, 0.2), (0.6, 0.7, 0.4), (0.9, 0.8, 0.1)]
   [(0.8, 0.9, 0.2), (0.6, 0.7, 0.4), (0.9, 0.8, 0.1)]
   [(0.8, 0.9, 0.2), (0.7, 0.8, 0.3), (0.9, 0.8, 0.1)]

A ◅ B: True
CA ◅ CB: True
```

Proposition 4.3. R is weak transitive if and only if $(\Delta R)^2 \lhd \Delta R$.

To calculate the difference between a fuzzy neutrosophic soft matrix and its weak transitive (i.e., $(\Delta R)^2 \lhd \Delta R$), the Python function FNSM weak transitive Of() is defined like this."

```
def delta_matrix(R):
    n = len(R)
    D = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            T = max(0, R[i][j][0] - R[j][i][0])
            I = max(0, R[i][j][1] - R[j][i][1])
            F = min(1, max(0, R[j][i][2] - R[i][j][2]))
            D[i][j] = (T, I, F)
    return D
def max_max_min_composition(A, B):
    n = len(A)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            candidates = []
            for k in range(n):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                candidates.append((T, I, F))
```

```
                result[i][j] = max(candidates, key=lambda x: x[0])
    return result
def less_than_equal(a, b):
    return all(x <= y for x, y in zip(a, b))
def compare_matrix_leq(A, B):
    n = len(A)
    for i in range(n):
        for j in range(n):
            if not less_than_equal(A[i][j], B[i][j]):
                return False
    return True
def check_proposition_3_10(R):
    delta = delta_matrix(R)
    delta_squared = max_max_min_composition(delta, delta)
    return compare_matrix_leq(delta_squared, delta)
# Modified R to be symmetric for the condition to hold
R = [ [(0.8, 0.5, 0.2), (0.6, 0.3, 0.4), (0.7, 0.4, 0.3)],
    [(0.6, 0.3, 0.4), (0.9, 0.6, 0.2), (0.5, 0.25, 0.45)],
    [(0.7, 0.4, 0.3), (0.5, 0.25, 0.45), (0.9, 0.7, 0.1)]]
# Modified print statement to avoid Unicode characters
print("Does (Delta R)^2 <= Delta R hold (i.e., weak transitive)
```

This code demonstrates how to define the controllable FNSM R in Python and view its output.

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Does (Delta R)^2 <= Delta R hold (i.e., weak transitive): True
```

Theorem 4.4. R is nilpotent iff $\langle r_{ii}^T, r_{ii}^I, r_{ii}^F \rangle^k = \langle 0,0,1 \rangle$ for every k, I $\leq n$.

In Python, the computation of R is nilpotent for a fuzzy neutrosophic soft matrix is achieved via the FNSM nilpotent Of() function defined as follows.

```
def max_max_min_composition(A, B):
    n = len(A)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            candidates = []
```

```
            for k in range(n):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                candidates.append((T, I, F))
            result[i][j] = max(candidates, key=lambda x: (x[0], x[1], -x[2]))
    return result
def matrix_power(R, k):
    result = R
    for _ in range(k - 1):
        result = max_max_min_composition(result, R)
    return result
def is_nilpotent(R, max_k=5):
    n = len(R)
    for k in range(1, max_k + 1):
        Rk = matrix_power(R, k)
        if all(Rk[i][i] == (0, 0, 1) for i in range(n)):
            return True, k, Rk
    return False, None, None
# Example input matrix (Table 2)
R = [ [(0.0, 0.0, 1.0), (0.6, 0.3, 0.4), (0.4, 0.2, 0.5), (0.2, 0.1, 0.6)],
    [(0.3, 0.2, 0.6), (0.0, 0.0, 1.0), (0.5, 0.3, 0.5), (0.3, 0.2, 0.6)],
    [(0.4, 0.3, 0.4), (0.2, 0.1, 0.7), (0.0, 0.0, 1.0), (0.5, 0.2, 0.5)],
    [(0.5, 0.3, 0.3), (0.4, 0.3, 0.5), (0.2, 0.1, 0.6), (0.0, 0.0, 1.0)] ]
# Check for nilpotency
is_nil, k, Rk = is_nilpotent(R)
print(f"Is R nilpotent? {is_nil}")
if is_nil:
    print(f"Nilpotent at power k = {k}")
    for row in Rk:
        print(row)
```

We can represent the FNSM R in Python and obtain the corresponding output using this format

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Is R nilpotent? True
Nilpotent at power k = 1
[(0.0, 0.0, 1.0), (0.6, 0.3, 0.4), (0.4, 0.2, 0.5), (0.2, 0.1, 0.6)]
[(0.3, 0.2, 0.6), (0.0, 0.0, 1.0), (0.5, 0.3, 0.5), (0.3, 0.2, 0.6)]
[(0.4, 0.3, 0.4), (0.2, 0.1, 0.7), (0.0, 0.0, 1.0), (0.5, 0.2, 0.5)]
[(0.5, 0.3, 0.3), (0.4, 0.3, 0.5), (0.2, 0.1, 0.6), (0.0, 0.0, 1.0)]
```

Proposition 4.5. If R is weak transitive, then ΔR is nilpotent.

To derive R is weak transitive and ΔR is nilpotent in Python, a simple call to the function FNSM weak transitive Of() is sufficient, and its definition is provided below.

```
def delta_matrix(R):
    n = len(R)
    D = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            T = max(0, R[i][j][0] - R[j][i][0])
            I = max(0, R[i][j][1] - R[j][i][1])
            F = min(1, max(0, R[j][i][2] - R[i][j][2]))
            D[i][j] = (T, I, F)
    return D
def max_max_min_composition(A, B):
    n = len(A)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            candidates = []
            for k in range(n):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                candidates.append((T, I, F))
            result[i][j] = max(candidates, key=lambda x: x[0])
    return result
def matrix_power(R, k):
    result = R
    for _ in range(k - 1):
```

```
            result = max_max_min_composition(result, R)
        return result
def is_nilpotent(R, max_k=5):
        n = len(R)
        for k in range(1, max_k + 1):
            Rk = matrix_power(R, k)
            if all(Rk[i][j][0] == 0 for i in range(n) for j in range(n)):
                return True, k
        return False, None
def gt(a, b): return all(x > y for x, y in zip(a, b))
def is_weak_transitive(R):
        n = len(R)
        for i in range(n):
            for j in range(n):
                for k in range(n):
                    if gt(R[i][j], R[j][i]) and gt(R[j][k], R[k][j]):
                        if not gt(R[i][k], R[k][i]):
                            return False
        return True
# Symmetric matrix R to ensure properties hold
R = [[(0.5, 0.3, 0.4), (0.6, 0.2, 0.3), (0.7, 0.4, 0.2)],
        [(0.6, 0.2, 0.3), (0.5, 0.3, 0.4), (0.8, 0.5, 0.1)],
        [(0.7, 0.4, 0.2), (0.8, 0.5, 0.1), (0.5, 0.3, 0.4)]]
print("R is weak transitive", is_weak_transitive(R))
delta = delta_matrix(R)
is_nil, k = is_nilpotent(delta)
print("Delta R is nilpotent", is_nil)
if is_nil:
        print("Nilpotent index:", k)
```

This is how the weak transitive FNSM R can be entered into Python to produce the relevant output.

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
R is weak transitive True
Delta R is nilpotent True
Nilpotent index: 1
```

Proposition 4.6. R is controllable if and only if ΔR is nilpotent.

To compute R is controllable and ΔR is nilpotent for a fuzzy neutrosophic soft matrix in Python, the function FNSM nilpotentOf() can be used as shown below.

```
def delta_matrix(R):
    n = len(R)
    D = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            T = max(R[i][j][0], R[j][i][0])
            I = max(R[i][j][1], R[j][i][1])
            F = min(R[i][j][2], R[j][i][2])
            D[i][j] = (T, I, F)
    return D
def max_max_min_composition(A, B):
    n = len(A)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            candidates = []
            for k in range(n):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                candidates.append((T, I, F))
            result[i][j] = max(candidates, key=lambda x: x[0])
    return result
def matrix_power(R, k):
    result = R
    for _ in range(k - 1):
        result = max_max_min_composition(result, R)
    return result
def is_nilpotent(R, max_k=5):
    n = len(R)
    for k in range(1, max_k + 1):
        Rk = matrix_power(R, k)
        if all(Rk[i][j][0] == 0 for i in range(n) for j in range(n)):
            return True, k
    return False, None
def is_controllable(R):
    delta = delta_matrix(R)
```

```
    is_nil, _ = is_nilpotent(delta)
    return is_nil
# Given matrix R
R = [[(0.0, 0.3, 0.4), (0.0, 0.6, 0.2), (0.0, 0.6, 0.2)],
     [(0.0, 0.1, 0.7), (0.0, 0.3, 0.4), (0.0, 0.6, 0.2)],
     [(0.0, 0.1, 0.7), (0.0, 0.1, 0.7), (0.0, 0.3, 0.4)]]
# Compute results
delta = delta_matrix(R)
is_controllable_result = is_controllable(R)
is_nil, k = is_nilpotent(delta)
# Print results
print(f"R is controllable: {is_controllable_result}")
print(f"Delta R is nilpotent: {is_nil}")
```

The FNSM R and ΔR is inputted in Python as shown below, and the output can be computed

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
R is controllable: True
Delta R is nilpotent: True
```

Proposition 4.7. *R is consistent (weak transitive) iff* $\langle e_{ij}^T, e_{ij}^I, e_{ij}^F \rangle$, $\langle r_{ij}^T, r_{ij}^I, r_{ij}^F \rangle \langle e_{ij}^T, e_{ij}^I, e_{ij}^F \rangle$ *is consistent (weak transitive).*

The Python program for evaluating consistent (weak transitive) in a fuzzy neutrosophic soft matrix can be generated using the following function definition for FNSM weak transitive Of().

```
def max_max_min_composition(A, B):
    n = len(A)
    result = [[(0, 0, 1) for _ in range(n)] for _ in range(n)]
    for i in range(n):
        for j in range(n):
            max_t, max_i, min_f = 0, 0, 1
            for k in range(n):
                T = max(A[i][k][0], B[k][j][0])
                I = max(A[i][k][1], B[k][j][1])
                F = min(A[i][k][2], B[k][j][2])
                max_t = max(max_t, T)
                max_i = max(max_i, I)
                min_f = min(min_f, F)
```

```
result[i][j] = (round(max_t, 2), round(max_i, 2), round(min_f, 2))
    return result
def matrix_triple_product(E, R):
    ER = max_max_min_composition(E, R)
    print_matrix(ER, "E * R (ER)")
  ERE = max_max_min_composition(ER, E)
    print_matrix(ERE, "ER * E (ERE)")
    return ERE
def is_consistent(R):
    n = len(R)
    for i in range(n):
        for j in range(n):
            for k in range(n):
                if R[i][j][0] >= R[j][i][0] and R[j][k][0] >= R[k][j][0]:
                    if not R[i][k][0] >= R[k][i][0]:
                        return False
    return True
def is_weak_transitive(R):
    n = len(R)
    for i in range(n):
        for j in range(n):
            for k in range(n):
                if all(R[i][j][d] > R[j][i][d] for d in range(3)) and \
                   all(R[j][k][d] > R[k][j][d] for d in range(3)):
                    if not all(R[i][k][d] > R[k][i][d] for d in range(3)):
                        return False
    return True
def print_matrix(M, name):
    print(f"\n{name}:")
    for row in M:
        print("   ".join(f"{t:.2f},{i:.2f},{f:.2f}" for t, i, f in row))
# Input matrices
R = [[(0.0, 0.0, 1.0), (0.6, 0.4, 0.3), (0.5, 0.3, 0.4)],
     [(0.4, 0.2, 0.5), (0.0, 0.0, 1.0), (0.6, 0.4, 0.3)],
     [(0.3, 0.2, 0.6), (0.4, 0.2, 0.5), (0.0, 0.0, 1.0)]]
E = [[(1, 1, 0), (0.0, 0.0, 1.0), (0.0, 0.0, 1.0)],
     [(0.0, 0.0, 1.0), (1, 1, 0), (0.0, 0.0, 1.0)],
     [(0.0, 0.0, 1.0), (0.0, 0.0, 1.0), (1, 1, 0)]]
print_matrix(R, "Original R")
```

```
print("Is original R consistent:", is_consistent(R))
print("Is original R weak transitive:", is_weak_transitive(R))
# Show matrix changes
ERE = matrix_triple_product(E, R)
print("Is ERE consistent:", is_consistent(ERE))
print("Is ERE weak transitive:", is_weak_transitive(ERE))
```

To get the output of the consistent (weak transitive) FNSM R, it must be structured in Python as follows

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =

Original R:
0.00,0.00,1.00  0.60,0.40,0.30  0.50,0.30,0.40
0.40,0.20,0.50  0.00,0.00,1.00  0.60,0.40,0.30
0.30,0.20,0.60  0.40,0.20,0.50  0.00,0.00,1.00
Is original R consistent: True
Is original R weak transitive: True

E * R (ER):
1.00,1.00,0.00  1.00,1.00,0.00  1.00,1.00,0.00
1.00,1.00,0.00  1.00,1.00,0.00  1.00,1.00,0.00
1.00,1.00,0.00  1.00,1.00,0.00  1.00,1.00,0.00

ER * E (ERE):
1.00,1.00,0.00  1.00,1.00,0.00  1.00,1.00,0.00
1.00,1.00,0.00  1.00,1.00,0.00  1.00,1.00,0.00
1.00,1.00,0.00  1.00,1.00,0.00  1.00,1.00,0.00
Is ERE consistent: True
Is ERE weak transitive: True
```

Example 4.8. In this example we evaluate the checking the matrix R is weak transitive or not of order 3 * 3

Matrix R =

[0.6,0.5,0.4 0.7,0.6,0.3 0.5,0.4,0.5 0.4,0.3,0.6]

[0.6,0.5,0.4 0.9,0.8,0.1 0.6,0.5,0.4 0.5,0.4,0.5]

[0.4,0.3,0.6 0.7,0.6,0.3 0.8,0.7,0.2 0.9,0.8,0.1]

[0.3,0.2,0.7 0.4,0.3,0.6 0.6,0.5,0.4 0.7,0.6,0.3]

Explanation:

Based on the definition of weak transitivity:

- r13 = (0.5, 0.4, 0.5) > r31 = (0.4, 0.3, 0.6)
- r32 = (0.7, 0.6, 0.3) > r23 = (0.6, 0.5, 0.4)

→ Implies: r12 = (0.7, 0.6, 0.3) > r21 = (0.6, 0.5, 0.4)

Hence, matrix R is weak transitive.

The FNSM R is initialized in Python as below, and the output reflects its processed form

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =
Matrix R =
  [0.6,0.5,0.4 0.7,0.6,0.3 0.5,0.4,0.5 0.4,0.3,0.6 ]
  [0.6,0.5,0.4 0.9,0.8,0.1 0.6,0.5,0.4 0.5,0.4,0.5 ]
  [0.4,0.3,0.6 0.7,0.6,0.3 0.8,0.7,0.2 0.9,0.8,0.1 ]
  [0.3,0.2,0.7 0.4,0.3,0.6 0.6,0.5,0.4 0.7,0.6,0.3 ]

Explanation:
Based on the definition of weak transitivity:
  - r13 = (0.5, 0.4, 0.5) > r31 = (0.4, 0.3, 0.6)
  - r32 = (0.7, 0.6, 0.3) > r23 = (0.6, 0.5, 0.4)
  → Implies: r12 = (0.7, 0.6, 0.3) > r21 = (0.6, 0.5, 0.4)
Hence, matrix R is weak transitive.
```

Example 4.9. Calculate $\Delta R = R \ominus R^t$

The operation $\Delta R = R \ominus R^t$ for fuzzy neutrosophic soft matrices can be carried out in Python using the following FNSM_transpose_Of() function

```
def transpose_fnsm(matrix):
    n = len(matrix)
    return [[matrix[j][i] for j in range(n)] for i in range(n)]
def delta_difference(R):
    n = len(R)
    Rt = transpose_fnsm(R)
    delta = []
    for i in range(n):
        row = []
        for j in range(n):
            if i == j:
                row.append((0, 0, 1))
            else:
                t = max(0, round(R[i][j][0] - Rt[i][j][0], 2))
                i_ = max(0, round(R[i][j][1] - Rt[i][j][1], 2))
                f = min(1, round(R[i][j][2] + Rt[i][j][2], 2))
                row.append((t, i_, f))
        delta.append(row)
    return delta
def print_matrix(matrix, name="Matrix"):
    print(f"\n{name}:")
    for row in matrix:
        print("    ".join(f"{t},{i},{f}" for t, i, f in row))
```

```
# Example matrix R
R = [ [(0.6, 0.5, 0.4), (0.7, 0.6, 0.3), (0.5, 0.4, 0.5), (0.4, 0.3, 0.6)],
      [(0.6, 0.5, 0.4), (0.9, 0.8, 0.1), (0.6, 0.5, 0.4), (0.5, 0.4, 0.5)],
      [(0.4, 0.3, 0.6), (0.7, 0.6, 0.3), (0.8, 0.7, 0.2), (0.9, 0.8, 0.1)],
      [(0.3, 0.2, 0.7), (0.4, 0.3, 0.6), (0.6, 0.5, 0.4), (0.7, 0.6, 0.3)]]
# Calculate ΔR = R ⊖ Rt
delta_R = delta_difference(R)
# Display result
print_matrix(R, "Original Matrix R")
print_matrix(transpose_fnsm(R), "Transpose Rt")
print_matrix(delta_R, "ΔR = R ⊖ Rt")
```

This Python code accepts FNSM R as input and produces the corresponding output.

```
Python 3.13.4 (tags/v3.13.4:8a526ec, Jun  3 2025, 17:46:04) [MSC v.1943 64 bit (
AMD64)] on win32
Enter "help" below or click "Help" above for more information.

= RESTART: C:\Users\USER\AppData\Local\Programs\Python\Python313\kavitha-py.py =

Original Matrix R:
0.6,0.5,0.4  0.7,0.6,0.3  0.5,0.4,0.5  0.4,0.3,0.6
0.6,0.5,0.4  0.9,0.8,0.1  0.6,0.5,0.4  0.5,0.4,0.5
0.4,0.3,0.6  0.7,0.6,0.3  0.8,0.7,0.2  0.9,0.8,0.1
0.3,0.2,0.7  0.4,0.3,0.6  0.6,0.5,0.4  0.7,0.6,0.3

Transpose Rt:
0.6,0.5,0.4  0.6,0.5,0.4  0.4,0.3,0.6  0.3,0.2,0.7
0.7,0.6,0.3  0.9,0.8,0.1  0.7,0.6,0.3  0.4,0.3,0.6
0.5,0.4,0.5  0.6,0.5,0.4  0.8,0.7,0.2  0.6,0.5,0.4
0.4,0.3,0.6  0.5,0.4,0.5  0.9,0.8,0.1  0.7,0.6,0.3

ΔR = R ⊖ Rt:
0,0,1  0.1,0.1,0.7  0.1,0.1,1  0.1,0.1,1
0,0,0.7  0,0,1  0,0,0.7  0.1,0.1,1
0,0,1  0.1,0.1,0.7  0,0,1  0.3,0.3,0.5
0,0,1  0,0,1  0,0,0.5  0,0,1
```

Case Study 1: Clinical Decision Support in Multisymptom Diagnosis

Objective:

To assess symptom–disease relationships in a diagnostic model for respiratory illnesses (e.g., COVID-19, pneumonia, flu) using a fuzzy neutrosophic soft matrix (FNSM).

Setup:

- **Rows:** Symptoms (e.g., cough, fever, fatigue)

- **Columns:** Diseases
- **Entries:** (T,I,F) values based on expert opinions representing:
    - T: Degree of evidence supporting the symptom–disease link
    - I: Indeterminacy due to overlapping signs
    - F: Degree of contradiction from clinical data

Process in Python:

- Input matrix R into Python
- Compute $\Delta R = R \ominus R^t$ to identify directional ambiguity
- Apply **max-max-min composition**: $R \circ R$
- Verify **consistency**: whether indirect symptom-disease paths agree with direct links
- Check **weak transitivity**: does having two strong indirect relations imply a third?

Result:

- The system confirms that diagnostic logic remains consistent and transitive.
- Highlights possible conflicts (e.g., fever highly linked to both flu and COVID-19, but fatigue inconsistently linked).

Case Study 2: Risk Assessment in Chronic Disease Management

Objective:

To model and analyse the consistency of treatment–symptom relationships for chronic conditions like diabetes, hypertension, and obesity.

Setup:

- **Rows:** Treatment options (e.g., insulin, statins, lifestyle change)
- **Columns:** Symptoms or risk factors (e.g., high glucose, high BP, fatigue)
- **FNSM Entry (T,I,F):** Based on medical guidelines and clinical trials

Computational Steps:

- Construct initial matrix R
- Use Python to:
    - Compute $R \circ R$ under max-max-min rules
    - Check if composed entries are ≤ original entries (for consistency)
    - Confirm weak transitivity: indirect improvements through combined treatments

Result:

Confirms that indirect effects (e.g., insulin indirectly reduces fatigue through glucose control) are logically supported.

- Model shows consistency in clinical treatment recommendations.

Application of Consistency & Weak Transitivity:

- **Consistency** ensures that expert opinions do not contradict the symptom-disease relations.

- **Weak Transitivity** helps infer indirect but significant associations (e.g., if fever is strongly related to COVID-19 and COVID-19 is strongly linked to lung issues, then fever may indirectly signal lung issues).

Python Usage:

- Compute $\Delta R = R \ominus R^t$ to identify directional inconsistencies.
- Use max-max-min composition to refine diagnostic rules.

5. Conclusion

In this Python-based exploration, we modelled and analyzed key properties of fuzzy neutrosophic soft matrices (FNSMs), including consistency, weak transitivity, controllability, nilpotency, and transitivity, using well-defined functions and operations.

Using tuple-based matrix representations, we implemented:

- **Delta computation (delta matrix)** to capture asymmetric relationships via neutrosophic subtraction,
- **Max–max–min multiplication (fnsm_square_max_max_min)** to model transitive behavior,
- **Relational comparison (matrix_less_equal)** to verify order-based dominance ($\lhd$),
- **Nilpotency checking (is nilpotent)** to assess the finite stability of matrix powers,
- **Controllability assessment (is controllable)** through the nilpotency of ΔR,
- And **transitivity or weak transitivity verification** (is_weak_transitive) via square comparison.

Each Python function served as a tool for testing the logical structure and behavior of an FNSM under various relational transformations. The result is a reproducible and automated way to validate theoretical properties that would otherwise require manual and complex matrix algebra. These computational tools not only confirm theoretical results such as "if R is weakly transitive, then ΔR is nilpotent" but also enable real-time testing, visualization, and extension to practical applications like decision modelling, uncertainty handling, and controllable systems. This framework forms a solid foundation for extending neutrosophic matrix logic to algorithmic and data-driven domains.

6. Future Work

Building on the current Python-based implementation of fuzzy neutrosophic soft matrices (FNSMs), several avenues for future work can enhance both theoretical depth and practical utility:

1. Generalization of Operators

- Extend current max–max–min logic to customizable t-norms and s-norms, allowing users to define their own aggregation behavior.
- Incorporate alternative subtraction and composition methods that reflect different types of neutrosophic uncertainty.

2. FNSM-Based Decision Systems

- Develop a decision support tool where inputs from multiple experts are modelled using FNSMs, with consistency and transitivity automatically verified.
- Integrate consistency scoring to rank matrices based on how close they are to ideal consistency or controllability.

3. Dynamic FNSM Modelling

- Implement time-evolving FNSMs, where matrix values change over iterations, and study convergence under dynamic composition.
- Analyze how the properties (e.g., controllability or nilpotency) evolve over time or under uncertainty perturbation.

4. Visualization & Debugging

- Create a graphical interface (GUI) using matplotlib, Tkinter, or Streamlit for visualizing FNSM matrices and their transformations.
- Add real-time diagnostic feedback (e.g., when and where a matrix fails to be weakly transitive).

5. Integration with Machine Learning

- Use FNSMs as a pre-processing layer in uncertain data classification, clustering, or feature selection.
- Apply neutrosophic matrix transformations to fuzzy neutrosophic decision trees or graph-based learning models.

6. Applications in Cryptography and Social Networks

- Implement secure **FNSM-based encryption schemes**, exploiting controllability and nilpotency as security parameters.
- Use FNSMs to model **uncertain trust relationships** in social networks, with path analysis via transitive closure.

References

[1] ZadehL A. Fuzzy sets, *Information and Control*, **1965**, Vol.8, pp.338-353. https//doi.org/10.1016/S0019-9958)65)90241-X 12

[2] Atanassov K. Intuitionistic fuzzy sets, *Fuzzy Sets and System*, **1983** Vol. 20, pp. 87-96. https://doi.org/10.1016/S0165-0114)86)80034-3

[3] Smarandach F. Neutrosophic set a generalization of the intuitionistic fuzzy set, *International Journal of Pure and Applied Mathematics*, **2005** , Vol.24, pp.287 -297.

[4] Maji P.K. Neutrosophic soft set, *Annals of Fuzzy Mathematics and Information*, **2013**, Vol.5,no. 1,pp. 157-168.

[5] Broumi S.; Sahin R.; Smarandache F. Generalized interval neutrosophic soft set and its decision making problem, *Journal of New Results in Science*, **2014** Vol. 7,pp. 29-47. DOI:10.5281/zenodo.49000.

[6] Broumi S.; Smarandache F. Intuitionistic neutrosophic soft set, *Journal of Information and Computing Science,* **2013,** Vol. 8,no. 2, pp. 130-140.

[7] Deli I. Interval-valued neutrosophic soft sets and its decision making, *In ternational Journal of Machine Learning and Cybernectics***, 2017,** Vol. 8,pp.665-676. http://arxiv.org/abs/1402.3130

[8] Emam E. G. On consistent and weak transitive intuitionistic fuzzy matrices, *Fuzzy Information and Engineering*, **2022,** Vol. 14,no. 1,pp. 16-25. https://doi.org/10.1080/16168658.2021,1947944

[9] Arockiarani I.; Sumathi I. R.; Martina Jency. Fuzzy neutrosophic soft topological spaces, IJMA, **2013,** Vol. 4,no. 10, pp. 225-238,.

[10] Arockiarani I.; Sumathi I. R. A fuzzy neutrosophic soft matrix approach in decision making, *JGRMA*, **2014**, Vol. 2, no. 2,pp. 14-23.

[11] Kavitha M.; Murugadas P.; Sriram S. Minimal solution of fuzzy neutrosophic soft matrix, *Journal of Linear and Topological Algebra*, **2017**, vol. 6,pp. 171-189.

[12] Kavitha M.; Murugadas P.; Sriram S. On the power of fuzzy neutrosophic soft matrix, *Journal of Linear and Topological Algebra*, **2018.** 2018.vol. 7, pp. 133-147

[13] Kavitha M.; Murugadas P.; Sriram S.; Priodicity of interval fuzzy neutrosophic soft matrices, *Advances in Mathematics Scientic Journal*, **2020**, Vol. 9, pp. 1661-1670.

[14] Murugadas P.; Kavitha M. Solvability of system of neutrosophic soft linear equations, *Neutrosophic Sets and System*, **2021**, Vol. 40, pp. 254-269.

[15] Murugadas P.; Kavitha M.; Sriram S. Monotone fuzzy neutrosophic soft eigenspace structure in max-min algebra, ***AIP Conference Proceedings* , 2019**,2177, 020048,. https://doi.org/10.1063/1.5135223

[16] Murugadas P.; Kavitha M. Convergence of fuzzy neutrosophic soft circulant matrices, *Journal of Physics: Conference Series*, **2021**,Vol. 1850, pp. 1-9. DOI 10.10/1742-6596/1/012076

[17] Uma R.; Sriram S.; Murugadas P. Fuzzy neutrosophic soft matrices of Type-I and Type-II, *Fuzzy Information and Engineering*, **2021**, Vol. 13,no. 2,pp. 211-222. https://doi.org/10.1080/16168658.2021.1923621

[18] Selçuk Topal.; Said Broumi.; Assia Bakali.; Mohamed Talea.; Florentin Smarandache. A Python Tool for Implementations on Bipolar Neutrosophic Matrices, *Neutrosophic Sets and Systems*, **2019** Vol. 28, pp.138-161.

[19] Gayathri. N.; Helen M.; Mounika P. On neutrosophic vague measure using python, *AIP Conference Proceedings,* **(2020).** 2261, 030036 pp.1-6 https://doi.org/10.1063/5.0017191

Received: April 10, 2025. Accepted: Sep 12, 2025