Palestine Journal of Mathematics

Vol 15(Special Issue II)(2026) , 223–232

Computational Analysis of Interior and Closure Operators in Neutrosophic Crisp Sets Using Python Program

K. Tharani and V. Kokilavani

MSC 2010 Classifications: Primary 54A40; Secondary 03E72, 03F55, 06B10.

Keywords and phrases: Neutrosophic Crisp Set (NCS), Neutrosophic Crisp Topological Space (NCTS), Neutrosophic Crisp Interior NCint(A), Neutrosophic Crisp Closure NCcl(A).

Corresponding Author: K. THARANI

Abstract Python serves as a powerful computational tool, enabling efficient analysis of mathematical structures by automating complex calculations. In classical topology, crisp sets are characterized by rigid true/false boundaries, limiting their applicability to real-world uncertainties. Neutrosophic theory provides a generalized framework that accommodates indeterminacy, offering a more comprehensive representation of topological spaces. This study extends the work of V. Kokilavani and K. Tharani by employing Python to analyze Neutrosophic Crisp Sets (NCS) and Neutrosophic Crisp Topological Spaces (NCTS). Specifically, it investigates fundamental operations, including the Neutrosophic Crisp Interior of A (NCint(A)) and Neutrosophic Crisp Closure of A (NCcl(A)). These computations, implemented using Python, facilitate an automated approach to exploring the structural relationships within the Neutrosophic Crisp topology.

1 Introduction

Mathematical methods have long been developed to explain and solve problems encountered in daily life. Among these, the fuzzy set theory introduced by Zadeh (1965) has attracted significant attention, as it approximates human linguistic concepts and provides tools for handling imprecision, ambiguity, and uncertainty. Over time, several extensions of fuzzy theory have been proposed to address the limitations of traditional fuzzy and intuitionistic fuzzy sets. As a result, many theories in diverse forms have been put out to deal with the imprecision problem. Since intuitionistic fuzzy logic and fuzzy sets were unable to handle information indeterminacy and could not convey inaccurate membership information, neutrosophic theory has been proposed as a better approach. This is due to the fact that information is frequently vague and imprecise, and it usually contains both neutral and opposition's information.

Neutrosophic sets analyses and applications are increasingly becoming enhanced by different computational tools that support the process. For instance, Topal et al. [8] implemented some Python programs that are able to compute outcome, precision and certainty matrices for bipolar neutrosophic matrices. In the same manner, El-Ghareeb [2] developed an open-source Python library aimed at enhancing computation of neutrosophic numbers and sets with emphasis on single-valued and interval-valued types. Furthermore, Saranya S. et al. [5] also developed a C# application for basic arithmetic operations such as complement, union, intersection, inclusion on neutrosophic sets. Also, Sleem et al. [7] developed software dealing with interval valued neutrosophic set (IVNS) which provides facilities for normalizing IVNS matrices.

J. J. Mershia Rabuni and N. Balamani [1] made it even easier and less demanding by describing a Python script for dealing with the set operations on neutrosophic soft sets that are union, intersection and complement. M. Vivek Prabu and M. Rahini [9] also used Python to find the closure and interior subsets for given sets in topologies. Giorgio Nordo et al. [4] provided an open-source Python framework suitable for carrying out research on neutrosophic sets existing in different universes by creating specific classes for symbols and mappings that are useful in research in this field. According to these advances, recently, V. Kokilavani and K. Tharani[3] have used Python to study NCSs and NCTSs. This article describes the NCTS calculator application that addresses important mathematical tools, such as the NCint(A), NCcl(A), which streamline

these calculations and improve their accuracy.

2 Preliminaries

This section recalls the fundamental definitions and concepts necessary for the subsequent discussion.

Definition 2.1. [6] A Neutrosophic Crisp Set (NCS) is a set of items with different levels of truth, indeterminacy and falsity. If A_1, A_2, and A_3 are the subsets of a universal set X, then the formal expression for NCS A is $\langle A_1, A_2, A_3\rangle$.
NCS's Types:

(i) Type 1 NCS: In this type, A_1 is disjoint from both A_2 and A_3 and additionally, A_2 and A_3 are disjoint from each other.

(ii) Type 2 NCS: This type is an extension of Type 1 with an additionally requires that the union of A_1, A_2, and A_3 equals the universal set X.

(iii) Type 3 NCS: In type 3, the subsets A_1, A_2, and A_3 are all disjoint and their union equals the universal set X.

Definition 2.2. [6] Consider a non-empty set X, and let A and B be NCS's defined as A=$\langle A_1, A_2, A_3\rangle$, B=$\langle B_1, B_2, B_3\rangle$, respectively. The set A can be regarded as a subset B if any of the conditions listed below are met:

(i) The subset A_1 is contained within B_1, A_2 is contained within B_2 and B_3 is contained within A_3.

(ii) A_1 is included in B_1, B_2 is included in A_2 & B_3 is included in A_3.

Definition 2.3. [6] Consider a non-empty set X and A=$\langle A_1, A_2, A_3\rangle$ and B=$\langle B_1, B_2, B_3\rangle$ represent NCS's A and B, respectively. The intersection of A and B can be determined using one of the following forms,

(i) The intersection of A_1 and B_1, the intersection of A_2 and B_2 & the union of A_3 and B_3.

(ii) The intersection A_1 and B_1, the union of A_2 and B_2 & the union of A_3 and B_3.

Definition 2.4. [6] Consider a non-empty set X and A=$\langle A_1, A_2, A_3\rangle$ and B=$\langle B_1, B_2, B_3\rangle$ represent NCS's A and B, respectively. The union of A and B can be obtained by any one of the following forms,

(i) The union of A_1 and B_1, the union A_2 and B_2 & the intersection of A_3 and B_3.

(ii) The union of A_1 and B_1, the intersection of A_2 and B_2 & the intersection of A_3 and B_3.

Definition 2.5. [6] Consider A=$\langle A_1, A_2, A_3\rangle$ as a NCS on X, A^c is representing the complement of neutrosophic crisp of A, can be defined in three different ways:
(i) A^c=$\langle A_1^c, A_2^c, A_3^c\rangle$, where each subset is individually complemented.
(ii) A^c=$\langle A_3, A_2, A_1\rangle$, with the subsets rearranged in reverse order.
(ii) A^c=$\langle A_3, A_2^c, A_1\rangle$, where A_2 is complemented, and the position of A_1 and A_3 is swapped.

Definition 2.6. [6] A structure(X,Γ) on a set X contains at least one element, is a $NCTS$, Γ which is a group of neutrosophic crisp subsets ∈ X, fulfills the following requirements:
(i) Both ϕ_N, X_N belong to Γ, where ϕ_N can take one of the forms $\langle\phi,\phi,X\rangle$ or $\langle\phi,X,X\rangle$ or $\langle\phi,X,\phi\rangle$ or $\langle\phi,\phi,\phi\rangle$ and X_N can take one of the forms $\langle X,\phi,\phi\rangle$ or $\langle X,X,\phi\rangle$ or $\langle X,\phi,X\rangle$ or $\langle X,X,X\rangle$.
(ii) Closed under finite intersections: For any A_1, $A_2 \in \Gamma$, the intersection A_1, $A_2 \in \Gamma$.
(iii) For every family of sets $\{A_j : j \in J\} \subseteq$ (X, Γ), the union of all A_j is also an element of Γ. In this framework, the elements of Γ are called neutrosophic crisp open set $(NCOS)$, and complement of each $NCOS$ is referred to as a neutrosophic crisp closed set $(NCCS)$.

Definition 2.7. [6] Let (X,Γ) represent $NCTS$ on X and A be a NCS on X. The neutrosophic crisp closure of A (abbreviated as NCcl(A) or NC Closure of A) and neutrosophic crisp interior of A (abbreviated as NCint(A) or NC Interior of A) on X are defined as follows:
NCcl(A) = ∩ {A: $A \subseteq C$ & C is a neutrosophic crisp closed set in X}.
NCint(A)=∪ {F: $F \subseteq A$ & F is a neutrosophic crisp open set in X}

3 METHODOLOGY

This study focuses on analyzing NCS operators to better understand the structures and relationships within neutrosophic crisp sets and topological spaces. Manually calculating these operators is complex and time-consuming. For example, a set X = $\{a, b\}$ generates 42 possible Neutrosophic Crisp Sets. Computing NCint(A), NCcl(A), and other operations for each set is labor-intensive and impractical for larger sets. To overcome these challenges, we developed a Python program to automate the calculations. This approach significantly reduces the time required and increases accuracy by efficiently handling the complex computations involved in analyzing Neutrosophic Crisp Set operators.

4 COMPUTATION OF NEUTROSOPHIC CRISP INTERIOR AND CLOSURE BY USING PYTHON

Pseudocode:

BEGIN
Import required libraries (ast, PrettyTable)

INPUT: Read the Neutrosophic Crisp Set
WHILE True **DO**
Prompt user to enter the neutrosophic crisp set
TRY parse input using ast.literal_eval
Validate if input is a tuple with exactly three list elements
BREAK the loop **IF** valid
CATCH SyntaxError **OR** ValueError
Display error message and prompt again
END WHILE

INPUT: Read Neutrosophic Crisp Topological Space(NCTS)
WHILE True **DO**
Prompt user to enter NCTS
TRY parse input using ast.literal_eval
Validate if input is a list of tuples with three list elements each
BREAK the loop **IF** valid
CATCH SyntaxError **OR** ValueError
Display error message and prompt again
END WHILE

INPUT: Read NCTS_Complement
WHILE True **DO**
Prompt user to enter NCTS_complement
TRY parse input using ast.literal_eval
Validate if input is a list of tuples with three list elements each
BREAK the loop **IF** valid
CATCH SyntaxError **OR** ValueError
Display error message and prompt again
END WHILE

INPUT: Choose NCsubset type, NCunion type, NCintersection type

COMPUTE: Find NCInterior(A)
Initialize NCinterior_list = [ ], NCinteriors = [ ]
FOR EACH subset B **IN** NCS **DO**
Compute NCinterior_subset(B, NCS, NCsubset_type_choice)
IF result is not False **THEN** Add to NCinteriors

NCinteriors **THEN** NCunion them using NCunion_type **ELSE** take single
Add result to NCinterior_list

COMPUTE: Find NCClosure(A)
Initialize NCclosure_list = [ ], NCclosures = [ ]
FOR EACH subset B **IN** NCTS_complement **DO**
Compute NCclosure_subset(NCS, B, NCsubset_type_choice)
IF result is not False **THEN** Add to NCclosures
END FOR
IF multiple NCclosures **THEN** NCintersect them using NCintersection_type **ELSE** take single
Add result to NCclosure_list

OUTPUT: Display results in a table
CREATE PrettyTable with columns "NCS", "NCint(A)", "NCcl(A)"
Add row containing NCS, NCinterior_list[0], NCclosure_list[0]
Print table
END

5 RESULT AND DISCUSSION

CODING:

```
import ast
#NCS denoted as Neutrosophic Crisp Set
while True:
    NCS_input = input("Enter the neutrosophic crisp set (e.g., ([], [], ['b'])): ")
    try:
        NCS = ast.literal_eval(NCS_input)
        if not isinstance(NCS, tuple):
            raise ValueError("Invalid format for NCS. Input must be a tuple.")
        if len(NCS) != 3:
            raise ValueError("Invalid format for NCS. Tuple must have exactly three elements.")
        for part in NCS:
            if not isinstance(part, list):
                raise ValueError("Invalid format for NCS. All elements of the tuple must be lists.")
        break
    except (SyntaxError, ValueError) as e:
        print(f"{e}. Please try again.")
```

Figure 1. Coding for Enter the Neutrosophic Crisp Set

To begin with, there is an infinite loop that repeatedly prompts the user to enter a valid neutrosophic crisp set after which they have provided a correct format. The acceptable user input is something like ([ ], [ ], ['b']), and the code saves this as a string. This string is then transformed into a Python data structure such as a tuple by using ast.literal_eval. Thus enables the code to check if the argument is in the correct format. The first thing the code does is check whether the input is a tuple or it raises an error stating "Invalid format for NCS . Input must be a tuple". Subsequently, the code makes sure the provided tuple has exactly three elements inside it. If it is not the case, another error is raised: "Invalid format for NCS. Tuple must have exactly three elements." The code also validates that each element in the provided tuple is a list. It will raise an error "Invalid format for NCS. All elements of the tuple must be lists." if an element is not a list. Lastly, if something goes wrong such as invalid input or a syntax error, the code will instead display the problem and provide the user a chance to correct it. As soon as the input has been validated through all the steps, the loop is broken and the program proceeds with the accepted neutrosophic crisp set.

OUTPUT:

```
Enter the neutrosophic crisp set (e.g., ([], [], ['b'])): [[],['a'],[]]
Invalid format for NCS. Input must be a tuple. Please try again.
Enter the neutrosophic crisp set (e.g., ([], [], ['b'])): ([],['a'])
Invalid format for NCS. Tuple must have exactly three elements. Please try again.
Enter the neutrosophic crisp set (e.g., ([], [], ['b'])): ((),('a'),())
Invalid format for NCS. All elements of the tuple must be lists. Please try again.
Enter the neutrosophic crisp set (e.g., ([], [], ['b'])): ([],['a'],[])
Enter NCTS (e.g., [([], [], ['a', 'b']), etc.,]):
```

Figure 2. Output of Enter the Neutrosophic Crisp Set

CODING:

```
#NCTS denoted as Neutrosophic Crisp Topological Space
while True:
    NCTS_input = input("Enter NCTS (e.g., [([], [], ['a', 'b']), etc.,]): ")
    try:
        NCTS = ast.literal_eval(NCTS_input)
        if not isinstance(NCTS, list):
            raise ValueError("Invalid format for NCTS. Please enter a list.")
        for subset in NCTS:
            if not isinstance(subset, tuple):
                raise ValueError("Invalid format for NCTS. Input must be a tuple.")
            for part in subset:
                if not isinstance(part, list):
                    raise ValueError("Invalid format for NCTS. Expected a list.")
        break
    except (SyntaxError, ValueError) as e:
        print(f"{e}. Please try again.")
```

Figure 3. Coding for Enter the Neutrosophic Crisp Topological Space

By using this code, the user can input a Neutrosophic Crisp Topological Space in the correct format and ensure that it is valid. Initially, the user is prompted to enter an NCTS, which is a list of tuples with lists as elements within each tuple. To illustrate, a valid input would look like this: [([ ], [ ],[‘a’]), ([‘a’], [‘b’, ‘c’],[ ])]. Ast.literal_eval converts the input text into a Python object without running any unsafe code. Once the input is converted, the code checks if it follows the expected structure. First, it checks that the input is a list. Then, it checks if each item in the list is a tuple and if the elements inside these tuples are lists. The program shows a message about an error and asks that the user try again if anything doesn’t match the required format. This process continues till the input is correct. Once the input passes all of the checks, the program proceeds to the next steps. This ensures that the NCTS is always properly structured and available for analysis.

OUTPUT:

Figure 4. Output of Enter the Neutrosophic Crisp Topological Space

CODING:

```
while True:
    NCTS_complement_input = input("Enter NCTS_complement (e.g., [(['a', 'b'], ['a', 'b'], []),etc.,]): ")
    try:
        NCTS_complement = ast.literal_eval(NCTS_complement_input)
        if not isinstance(NCTS_complement, list):
            raise ValueError("Invalid format for NCTS_complement. Please enter a list.")
        for subset in NCTS_complement:
            if not isinstance(subset, tuple):
                raise ValueError("Invalid format for NCTS_complement. Input must be a tuple.")
            for part in subset:
                if not isinstance(part, list):
                    raise ValueError("Invalid format for NCTS_complement. Expected a list.")
        break
    except (SyntaxError, ValueError) as e:
        print(f"{e}. Please try again.")
```

Figure 5. Coding for Enter the Complement of Neutrosophic Crisp Topological Space

The user's input of a Neutrosophic Crisp Topological Space complement (NCTS_complement) in the correct format is verified by this function. Input such as [([‘a’, ‘b’], [‘a’, ‘b’], [ ])], which should be a list of tuples, is requested from the user. Each tuples elements should be listed. After that, the program safely converts the user's input a string into a Python object using ast.literal_eval. After conversion, the code validates the structure. It first checks if the overall input is a list, then ensures each item in the list is a tuple, and finally checks that each element within those tuples is a list. If any part of the input does not fit the required structure, an error message appears, explaining the issue, and the user is prompted to try again. This process continues until the input is valid, at which time the loop is terminated and the validated NCTS_complement is ready for further usage. This method ensures that the input is properly formatted and prevents problems in later processing.

OUTPUT:

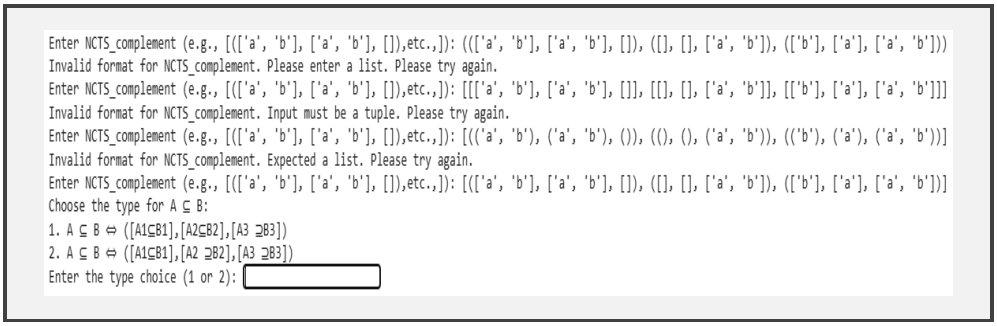


Figure 6. Output of Enter the Complement of Neutrosophic Crisp Topological Space

CODING:

```
# User selects subset type
print("Choose the type for A ⊆ B:")
print("1. A ⊆ B ⇔ ([A1⊆B1],[A2⊆B2],[A3 ⊇B3])")
print("2. A ⊆ B ⇔ ([A1⊆B1],[A2 ⊇B2],[A3 ⊇B3])")
NCsubset_type_choice = int(input("Enter the type choice (1 or 2): "))

print("Choose the type of union:")
print("1. Union Type 1: [A1 ∪ B1, A2 ∪ B2, A3 ∩ B3]")
print("2. Union Type 2: [A1 ∪ B1, A2 ∩ B2, A3 ∩ B3]")
NCunion_type = int(input("Enter the index of the union type: "))

print("\nChoose the type of intersection:")
print("1. Intersection Type 1: [A1 ∩ B1, A2 ∩ B2, A3 ∪ B3]")
print("2. Intersection Type 2: [A1 ∩ B1, A2 ∪ B2, A3 ∪ B3]")
NCintersection_type = int(input("Enter the index of the intersection type: "))
```

Figure 7. Coding for Choose the Neutrosophic Crisp Subset, Union and Intersection

This code allows the user to specify rules for working with Neutrosophic Crisp Sets (NCS), with a focus on neutrosophic crisp subsets, unions, and intersections. First, it prompts the user to select how to construct a neutrosophic crisp subset relationship between two neutrosophic crisp sets, A and B. The user enters 1 or 2 to choose one of the two options that are displayed. Next, the code prompts the user to select how to compute the neutrosophic crisp union of two neutrosophic crisp sets. Again, there are two options with particular guidelines for combining the neutrosophic crisp sets, which the user selects by entering 1 or 2. Lastly, the user inputs 1 or 2 to select one of two choices for calculating the neutrosophic crisp intersection of two neutrosophic crisp sets. Variables are used to store the user's preferences for the neutrosophic crisp subset type, intersection type, and union type. This makes it simple to apply the chosen rules in calculations.

CODING:

```
# Find Neutrosophic Crisp interior of A
NCinterior_list = []
NCinteriors = []
for B in NCTS:
    NCinterior = NCinterior_subset(B, NCS, NCsubset_type_choice)
    if NCinterior:
        NCinteriors.append(NCinterior)
if len(NCinteriors) > 1:
    NCunion_result = NCinteriors[0]
    for NCinterior in NCinteriors[1:]:
        NCunion_result = NCunion_of_subsets(NCunion_result, NCinterior, NCunion_type)
    NCinterior_list.append(NCunion_result)
elif len(NCinteriors) == 1:
    NCinterior_list.append(NCinteriors[0])
```

Figure 8. Coding for Finding the Neutrosophic Crisp Interior of A

This code computes the neutrosophic crisp interior of a Neutrosophic Crisp Set (NCS) using a specific neutrosophic crisp subset type and then combines the results with a defined neutrosophic crisp union type. Examines whether each B in the provided Neutrosophic Crisp Topological Space (NCTS) satisfies the neutrosophic crisp subset condition with the NCS. If it works, the outcome is added to a list of neutrosophic crisp interiors (NCinteriors). If there are numerous neutrosophic crisp interiors in NCinteriors, they are combined with the neutrosophic crisp union type chosen. The first neutrosophic crisp interior is used as a starting point, and the following neutrosophic crisp interiors are merged with it one by one. The NCinterior_list is updated with

the final combined result. If there is just one valid neutrosophic crisp interior, it is simply added to the NCinterior_list. This procedure guarantees that all valid neutrosophic crisp interiors are correctly calculated and combined.

CODING:

```
# Find Neutrosophic Crisp closure of A
NCclosure_list = []
NCclosures = []
for B in NCTS_complement:
    NCclosure = NCclosure_subset(NCS, B, NCsubset_type_choice)
    if NCclosure:
        NCclosures.append(NCclosure)
if len(NCclosures) > 1:
    NCintersection_result = NCclosures[0]
    for NCclosure in NCclosures[1:]:
        NCintersection_result = NCintersection_of_subsets(NCintersection_result,
                                                          NCclosure, NCsubset_type_choice)
    NCclosure_list.append(NCintersection_result)
elif len(NCclosures) == 1:
    NCclosure_list.append(NCclosures[0])
```

Figure 9. Coding for Finding the Neutrosophic Crisp Closure of A

This code computes the neutrosophic crisp closure of a Neutrosophic Crisp Set (NCS) using a specific neutrosophic crisp subset type and then combines the results with a defined neutrosophic crisp intersection type. It examines if each B in the complement of Neutrosophic Crisp Topological Space (NCTS_complement) satisfies the neutrosophic crisp subset condition with the NCS. If it works, the outcome is added to a list of neutrosophic crisp closure (NCclosure). If there are numerous neutrosophic crisp closure in NCclosure, they are combined with the neutrosophic crisp intersection type chosen. The first neutrosophic crisp interior is used as a starting point, and the following neutrosophic crisp closure are merged with it one by one. The NCclosure_list is updated with the final combined result. If there is just one valid neutrosophic crisp closure, it is directly added to the NCclosure_list. The final list, NCclosure_list, contains the calculated neutrosophic crisp closures for the NCS.

CODING:

```
from prettytable import PrettyTable
table = PrettyTable(["NCS", "NCinterior(A)", "NCclosure(A)"])
table.add_row([NCS, NCinterior_list, NCclosure_list])
print(table)
```

Figure 10. Coding for Creating a PrettyTable

The code begins by importing the PrettyTable module, which generates structured tables for displaying data in an easy to read. The table is then constructed with three column headers: “NCS” for the Neutrosophic Crisp Set, “NCinterior(A)” for the computed neutrosophic crisp interior of set A, and “NCclosure(A)” for the computed neutrosophic crisp closure of set A. The specified neutrosophic crisp set, its neutrosophic crisp interior, and its neutrosophic crisp closure are then represented by the values of NCS, NCinterior_list, and NCclosure_list, which are added to the table using the add_row method. The formatted table is finally printed by the print(table) command, which facilitates a visual representation of the computed outcomes for each NCS.

OUTPUT:

```
Enter the neutrosophic crisp set (e.g., ([], [], ['b'])): ([],['a'],[])
Enter NCTS (e.g., [([], [], ['a', 'b']), etc.,]): [([], [], ['a', 'b']), (['a', 'b'], ['a', 'b'], []), (['a'], ['b'], [])]
Enter NCTS_complement (e.g., [(['a', 'b'], ['a', 'b'], []),etc.,]): [(['a', 'b'], ['a', 'b'], []), ([], [], ['a', 'b']), (['b'], ['a'], ['a', 'b'])]
Choose the type for A ⊆ B:
1. A ⊆ B ⇔ ([A1⊆B1],[A2⊆B2],[A3 ⊇B3])
2. A ⊆ B ⇔ ([A1⊆B1],[A2 ⊇B2],[A3 ⊇B3])
Enter the type choice (1 or 2): 1
Choose the type of union:
1. Union Type 1: [A1 ∪ B1, A2 ∪ B2, A3 ∩ B3]
2. Union Type 2: [A1 ∪ B1, A2 ∩ B2, A3 ∩ B3]
Enter the index of the union type: 1

Choose the type of intersection:
1. Intersection Type 1: [A1 ∩ B1, A2 ∩ B2, A3 ∪ B3]
2. Intersection Type 2: [A1 ∩ B1, A2 ∪ B2, A3 ∪ B3]
Enter the index of the intersection type: 1
+----------------+-----------------------+--------------------------------+
|      NCS       |    NCinterior(A)      |         NCclosure(A)           |
+----------------+-----------------------+--------------------------------+
| ([], ['a'], []) | [([], [], ['a', 'b'])] | [(['a', 'b'], ['a', 'b'], [])] |
+----------------+-----------------------+--------------------------------+
```

Figure 11. Output of NCint(A) and NCcl(A)

Based on the given output, the program processes the input Neutrosophic Crisp Set A, the NCTS, and the NCTS_complement to compute the neutrosophic crisp interior and neutrosophic crisp closure of A as follows:

(i) **Neutrosophic Crisp Set (A):**
The input A = ([ ],[‘a’],[ ]) represents a Neutrosophic Crisp Set where the three components indicate elements in the truth ([ ]), indeterminacy ([‘a’]), and falsehood ([ ]).

(ii) **Neutrosophic Crisp Interior (NCint(A)):**
The program analyzes each neutrosophic crisp set in NCTS to check if it satisfies the chosen neutrosophic crisp subset condition A ⊆ B (type 1: $A_1 \subseteq B_1, A_2 \subseteq B_2, A_3 \supseteq B_3$).

- For B = ([ ], [ ], [‘a’, ‘b’]), the neutrosophic crisp subset requirement is satisfied.
- The neutrosophic crisp subset requirement is not valid for the other neutrosophic crisp sets in NCTS. The result is simply stored as NCinterior(A) = [([ ], [ ], [‘a’, ‘b’])] since only one valid neutrosophic crisp interior set can be determined.

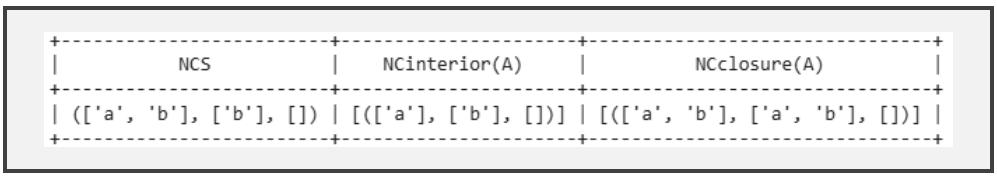
(iii) **Neutrosophic Crisp Closure (NCcl(A)):**
Similarly, the program analyzes each neutrosophic crisp set B in NCTS_complement to see if they satisfy the same condition A⊆ B (type 1).

- For B = ([‘a’, ‘b’], [‘a’, ‘b’], [ ]), the neutrosophic crisp subset requirement is satisfied.
- For the other neutrosophic crisp sets in NCTS_complement, the requirement is not satisfied. Thus, the result is stored as: NCcl(A) = ([‘a’, ‘b’], [‘a’, ‘b’], [ ]) since only one valid neutrosophic crisp closure set is found.

(iv) **Tabular Output:**
A table format is used to display the results:

- A: The Neutrosophic Crisp Set ([ ],[‘a’],[ ]).
- NCinterior(A): Computed the Neutrosophic Crisp Interior ([([ ],[ ], [‘a’,’b’])]).
- NCclosure(A): Computed the Neutrosophic Crisp Closure ([([‘a’,’b’],[‘a’,’b’], [])]).

OUTPUT:



NCS	NCinterior(A)	NCclosure(A)
(['a', 'b'], ['b'], [])	[(['a'], ['b'], [])]	[(['a', 'b'], ['a', 'b'], [])]

Figure 12. Output for NCinterior and NCclosure of A

Using the above conditions neutrosophic crisp subset type, neutrosophic crisp union type, and neutrosophic crisp intersection type, along with the concepts of neutrosophic crisp topological space and its complement, now consider the neutrosophic crisp set as ([‘a’, ‘b’], [‘b’], [ ]). For this neutrosophic crisp set, the computed neutrosophic crisp interior is [([ ], [ ], [‘a’, ‘b’]), ([‘a’], [‘b’], [ ])], which is stored in NCinteriors. NCinteriors are considered neutrosophic crisp unions since their length exceeds one. The resulting set, ([‘a’], [‘b’], [ ]), is stored in NCinterior_list. Based on the given subset condition, this result illustrates how the program generates valid neutrosophic crisp interiors and neutrosophic crisp closures.

6 Conclusion

The neutrosophic crisp interior and closure functions are two basic operations of NCTS that have been efficiently computed in this article using Python programming: NC Closure of A, NC Interior of A. By automating these procedures, we provided an effective way for dealing with the difficulties inherent in neutrosophic crisp sets, allowing for accurate and promptly calculations of these operators. This method makes NCTS analysis easier.

References

[1] N. Balamani and J. J. Mershia Rabuni, *Computation of Neutrosophic Soft Topology Using Python*, Neutrosophic Sets and Systems, **60**, (2023).

[2] H. A. El-Ghareeb, *Novel Open Source Python Neutrosophic Package*, Neutrosophic Sets and Systems, **25**, 136–160, (2019).

[3] V. Kokilavani and K. Tharani, *Utilizing Python for Neutrosophic Theory: A Study of Neutrosophic Crisp Sets and Topological Spaces*, Journal of Computational Analysis and Applications, **33**(7), 1042–1053, (2024).

[4] G. Nordo, S. Jafari, A. Mehmood, and B. Basumatary, *A Python Framework for Neutrosophic Sets and Mappings*, Neutrosophic Sets and Systems, **65**, (2024).

[5] S. Saranya, M. Vigneshwaran, and S. Jafari, *C# Application to Deal with Neutrosophic g^{α}-Closed Sets in Neutrosophic Topology*, Applications and Applied Mathematics, **15**(1), 226–239, (2020).

[6] A. A. Salama, F. Smarandache, and V. Kroumov, *Neutrosophic Crisp Sets & Neutrosophic Crisp Topological Space*, Neutrosophic Sets and Systems, **2**, 25–30, (2014).

[7] A. Sleem, M. Abdel-Baset, and I. El-Henawy, *PyIVNS: A Python-Based Tool for Interval-Valued Neutrosophic Operations and Normalization*, SoftwareX, **12**, (2020).

[8] S. Topal, S. Broumi, A. Bakali, M. Talea, and F. Smarandache, *A Python Tool for Implementations on Bipolar Neutrosophic Matrices*, Neutrosophic Sets and Systems, **28**, 138–161, (2019).

[9] M. Vivek Prabu and M. Rahini, *Application of Kuratowski's Closure Operator in Python Program*, AIP Conference Proceedings, **2718**(1), 030003, (2023).

Author information

K. Tharani, Research Scholar, Department of Mathematics, Kongunadu Arts and Science College(Autonomous), Coimbatore, India.
E-mail: tharanitopo23@gmail.com

V. Kokilavani, Associate Professor and Head, Department of Mathematics, Kongunadu Arts and Science College(Autonomous), Coimbatore, India.
E-mail: vanikasc@yahoo.co.in