Neutrosophic Sets and Systems, Vol. 28, 2019
University of New Mexico

A Python Tool for Implementations on Bipolar Neutrosophic Matrices

Selçuk Topal[1], Said Broumi[2*], Assia Bakali[3], Mohamed Talea[2], Florentin Smarandache[4]

[1] Bitlis Eren University, Faculty of Science and Arts, Department of Mathematics, Bitlis, Turkey, s.topal@beu.edu.tr.

[2] Laboratory of Information Processing, Faculty of Science Ben M'Sik, University Hassan II, B.P 7955, Sidi Othman, Casablanca, Morocco, broumisaid78@gmail.com, s.broumi@flbenmsik.ma, taleamohamed@yahoo.fr

[3] Ecole Royale Navale, Boulevard Sour Jdid, B.P 16303 Casablanca, Morocco, assiabakali@yahoo.fr

[4] Department of Mathematics, University of New Mexico, 705 Gurley Avenue, Gallup, NM 87301, USA, smarand@unm.edu

*Correspondence: Said Broumi ; broumisaid78@gmail.com; Tel.: 2126611416232

Abstract: Bipolar neutrosophic matrices (BNM) are obtained by bipolar neutrosophic sets. Each bipolar neutrosophic number represents an element of the matrix. The matrices are representable multi-dimensional arrays (3D arrays). The arrays have nested list data type. Some operations, especially the composition is a challenging algorithm in terms of coding because there are so many nested lists to manipulate. This paper presents a Python tool for bipolar neutrosophic matrices. The advantage of this work, is that the proposed Python tool can be used also for fuzzy matrices, bipolar fuzzy matrices, intuitionistic fuzzy matrices, bipolar intuitionistic fuzzy matrices and single valued neutrosophic matrices.

Keywords: Python; Neutrosophic sets; bipolar neutrosophic sets; matrix; composition operation

1. Introduction

Smarandache [1] gave the concept of neutrosophic set (NS) by considering the triplets independent components whose values belong to real standard or nonstandard unit interval] - 0, 1^+[. Later on, Smarandache [1] gave single valued neutrosophic set (SVNS) to apply into the various engineering applications. The various properties of SVNS is being studied by Wang et al. [2]. Further, Zhang et al. [3] presented a concept of interval-valued NS (IVNS) where the different membership degrees are represented by interval. In [4] Deli et al. introduced the concept of bipolar neutrosophic sets and their applications based on multicriteria decision making problems. The same author [5] proposed the bipolar neutrosophic refined sets and their applications in medical diagnosis for more details about the applications and its sets, we refer to [6]. Since the existence of NS, various scholars have presented the approaches related to SVNS and bipolar neutrosophic sets into the different fields. For instance, Mumtaz et al. [7] developed the concept of bipolar neutrosophic soft sets that combines soft sets and bipolar neutrosophic sets. In [8, 9] Broumi et al. introduced the notion of bipolar single valued neutrosophic graph theory and its shortest path problem. Dey et al. [10] considered TOPSIS method for solving the decision making problem under bipolar neutrosophic environment. Akram et al. [11] described bipolar neutrosophic TOPSIS method and bipolar neutrosophic ELECTRE-I

method. Akram and Sarwar [12] studied the novel multiple criteria decision making methods based on bipolar neutrosophic sets and bipolar neutrosophic graphs. Akram and Sitara [13] introduced the concept of bipolar single-valued neutrosophic graph structures and discussed certain notions of bipolar single-valued neutrosophic graph structures with examples. Singh [14] introduced bipolar neutrosophic graph representation of concept lattice and it's processing using granular computing. Mullai and Broumi [15] presented shortest path problem by minimal spanning tree algorithm using bipolar neutrosophic numbers. Uluçay et al. [16] defined similarity measures of bipolar neutrosophic sets and their application to multiple criteria decision making. Based on literal neutrosophic numbers, Mamouni et al. [17] defined the addition and multiplication of two neutrosophic fuzzy matrices. in the light of Fuzzy Neutrosophic soft sets, Arockiarani [18] present a new technique for handling decision making problems and proposed some new notions on matrix representation. Karaaslan and Hayat [19] introduced some novel operations on neutrosophic matrices. Uma et al. [20] introduced two types of fuzzy neutrosophic soft Matrices. The same authors in [21] decomposed fuzzy neutrosophic soft matrix by means of its section of fuzzy neutrosophic soft matrix of Type-I. Hassan et al. [22] defined some special types of bipolar single valued neutrosophic graphs. Akram and Siddique [23] discussed certain types of edge irregular bipolar neutrosophic graphs. Pramanik [24] developed cross entropy measures of bipolar neutrosophic sets and interval bipolar neutrosophic sets. Wang *et al*. [25] defined Frank operations of bipolar neutrosophic numbers (BNNs) and proposed Frank bipolar neutrosophic Choquet Bonferroni mean operators by combining Choquet integral operators and Bonferroni mean operators based on Frank operations of BNNs. In the same study, Akram and Nasir [26] introduced the concept of p-competition bipolar neutrosophic graphs. then they defined generalization of bipolar neutrosophic competition graphs called m-step bipolar neutrosophic competition graphs. AKRAM and SHUM [27] defined Bipolar Neutrosophic Planar Graphs. Hashim et al. [28] provide an application of neutrosophic bipolar fuzzy sets in daily life's problem related with HOPE foundation that is planning to build a children hospital. Akram, and Luqman [29] generalized the concept of bipolar neutrosophic sets to hypergraphs. Das et al. [30] proposes an algorithmic approach for group decision making (GDM) problems using neutrosophic soft matrix (NSM) and relative weights of experts.

Broumi et al. [31-34] applied the concept of IVNS on graph theory and studied some interesting results. Broumi et al. [35] developed a Matlab toolbox for computing operational matrices under the SVNS environments. Pramanik et al [36] developed a hybrid structure termed "rough bipolar neutrosophic set". In [37] Pramanik et al. presented Bipolar neutrosophic projection based models for solving multi-attribute decision making problems. Broumi et al [38] developed the concept of bipolar complex neutrosophic sets and its application in decision making problem. Akram, et al.[39] applied the concept of bipolar neutrosophic sets to incidence graphs and studied some properties. For more details on the application of neutrosophic set theory, we refer the readers to [46-52].

Among all the above, matrices play a vital job in the expansion region of science and engineering. However, the classical matrix theory neglects the role of uncertainties during the analysis. Therefore, the decision process may contain a lot of uncertainties. Thus, the role of the fuzzy matrices and their extension including triangular fuzzy matrices, type-2 triangular fuzzy matrices, interval valued fuzzy matrices, intuitionistic fuzzy matrices, interval valued intuitionistic fuzzy matrices are studied deeply by several scholars. In [40] Zahariev, developed a Matlab software package to the fuzzy algebras. In

[41], authors solved intuitionistic fuzzy relational rational calculus problems using a fuzzy toolbox. Later on, in [42] Karunambigai and Kalaivani proposed some computing procedures in Matlab for intuitionistic fuzzy operational matrices with suitable examples. Uma et al. [43] studied determinant theory for fuzzy neutrosophic soft square matrices. Also, in [44] Uma et al. introduced the determinant and adjoint of a square Fuzzy Neutrosophic Soft Matrices (FNSMs) a defined the circular FNSM and study some relations on square FNSM such as reflexivity, transitivity and circularity.

Recently few researchers [45] developed a Python programs for computing operations on neutrosophic numbers, but all these programs cannot deal with neutrosophic matrices, to do best of our knowledge, there is no work conducted on developing python codes to compute the operations on single valued neutrosophic matrices and bipolar neutrosophic matrices. Thus, there is a need to develop the work in that direction. For it, the presented paper discusses various operations of bipolar neutrosophic sets and their corresponding Python code for different metrics. To achieve it, rest of the manuscript is summarized as. In section 2, some concepts related to SVNS, BNS are presented. Section 3 deals with the generations of Python programs for bipolar neutrosophic matrices with a numerical example and lastly, conclusion is summarized in section 4.

2.BACKGROUND AND BIPOLAR NEUTROSOPHIC SETS

In this section, some basic concepts on SVNS, BNS are briefly presented over the universal set ξ [1, 2, 4].

Definition 2.1 [1] A set A is said to be A neutrosophic set 'A' consists of three components namely truth, indeterminate and falsity denoted by $T_A, I_A(x)$ and $F_A(x)$ such that

$T_A(x), I_A(x), F_A(x) \in]^{-}0, 1^{+}[$ and $-0 \leq \sup T_A(x) + \sup I_A(x) + \sup F_A(x) \leq 3+$ (1)

Definition 2.2 [2] A SVNS 'A' on X is given as

$A = \{< x: T_A(x), I_A(x), F_A(x) > x \in \xi\}$ (2)

where the functions $T_A(x)$, $I_A(x)$, $F_A(x) \in [0.\ 1]$ are named "degree of truth, indeterminacy and falsity membership of x in A", such that

$0 \leq T_A(x) + I_A(x) + F_A(x) \leq 3$ (3)

Definition 2.3[4]. A bipolar neutrosophic set A in ξ is defined as an object of the form

$A=\{<x, (T_A^P(x), I_A^P(x), F_A^P(x), T_A^N(x), I_A^N(x), F_A^N(x))>: x \in \xi\}$, where $T_A^P(x), I_A^P(x), F_A^P(x): \xi \rightarrow [1, 0]$ and $T_A^N(x), I_A^N(x), F_A^N(x): \xi \rightarrow [-1, 0]$. The positive membership degree $T_A^P(x), I_A^P(x), F_A^P(x)$enotes the truth membership, indeterminate membership and false membership of an element $\in \xi$ corresponding to a bipolar neutrosophic set whereas the negative membership degree $T_A^N(x), I_A^N(x), F_A^N(x)$denotes the truth membership, indeterminate membership and false membership of an element $x \in \xi$ to some implicit counter-property corresponding to a bipolar neutrosophic set A. For convenience a bipolar neutrosophic number is represented by

$A= <(T_A^P, I_A^P, F_A^P, T_A^N, I_A^-, F_A^->$ (4)

Definition 2.4 [4]. In order to make a comparison between two BNN. The score function is applied to compare the grades of BNS. This function shows that greater is the value, the greater is the bipolar neutrosophic sets and by using this concept paths can be ranked. Suppose

$\tilde{A} =< T^P, \mathrm{I}^P, \mathrm{F}^P, T^N, \mathrm{I}^N, \mathrm{F}^N >$ be a bipolar neutrosophic number. Then, the score function $s(\tilde{A})$, accuracy function $a(\tilde{A})$ and certainty function $c(\tilde{A})$ of a BNN are defined as follows:

(i) $s(\tilde{A}) = \left(\frac{1}{6}\right) \times \left[T^P + 1 - I^P + 1 - F^P + 1 + T^N - I^N - F^N \right]$ (5)

(ii) $a(\tilde{A}) = T^P - F^P + T^N - F^N$ (6)

(iii) $c(\tilde{A}) = T^P - F^N$ (7)

Comparison of bipolar neutrosophic numbers

Let $\tilde{A}_1 =< T_1^p, \mathrm{I}_1^p, \mathrm{F}_1^p, T_1^n, \mathrm{I}_1^n, \mathrm{F}_1^n >$ and $\tilde{A}_2 =< T_2^p, \mathrm{I}_2^p, \mathrm{F}_2^p, T_2^n, \mathrm{I}_2^n, \mathrm{F}_2^n >$ be two bipolar neutrosophic numbers then

i. If $s(\tilde{A}_1) \succ s(\tilde{A}_2)$, then $\tilde{A}_1$ is greater than $\tilde{A}_2$, that is, $\tilde{A}_1$ is superior to $\tilde{A}_2$, denoted by $\tilde{A}_1 \succ \tilde{A}_2$.

ii. If $s(\tilde{A}_1) = s(\tilde{A}_2)$, and $a(\tilde{A}_1) \succ a(\tilde{A}_2)$ then $\tilde{A}_1$ is greater than $\tilde{A}_2$, that is, $\tilde{A}_1$ is superior to $\tilde{A}_2$, denoted by $\tilde{A}_1 \succ \tilde{A}_2$.

iii. If $s(\tilde{A}_1) = s(\tilde{A}_2)$, $a(\tilde{A}_1) = a(\tilde{A}_2)$, and $c(\tilde{A}_1) \succ c(\tilde{A}_2)$ then $\tilde{A}_1$ is greater than $\tilde{A}_2$, that is, $\tilde{A}_1$ is superior to $\tilde{A}_2$, denoted by $\tilde{A}_1 \succ \tilde{A}_2$.

iv. If $s(\tilde{A}_1) = s(\tilde{A}_2)$, $a(\tilde{A}_1) = a(\tilde{A}_2)$, and $c(\tilde{A}_1) = c(\tilde{A}_2)$ then $\tilde{A}_1$ is equal to $\tilde{A}_2$, that is, $\tilde{A}_1$ is indifferent to $\tilde{A}_2$, denoted by $\tilde{A}_1 = \tilde{A}_2$.

Definition 2.5 [4]: A bipolar neutrosophic matrix (BNM) of order m× n is defined as
$A_{\mathrm{BNM}} = \left[< a_{ij}, a_{ij_T}^P, a_{ij_I}^P, a_{ij_F}^P, a_{ij_T}^N, a_{ij_I}^N, a_{ij_F}^N > \right]_{\mathrm{m} \times \mathrm{n}}$ where
$a_{ij_T}^P$ is the positive membership value of element a_{ij} in A.
$a_{ij_T}^N$ is the negative membership value of element a_{ij} in A.
$a_{ij_T}^P$ is the positive indeterminate-membership value of element a_{ij} in A.
$a_{ij_T}^N$ is the negative indeterminate-membership value of element a_{ij} in A.
$a_{ij_T}^P$ is the positive non- membership value of element a_{ij} in A.
$a_{ij_T}^N$ is the negative non-membership value of element a_{ij} in A.

For simplicity, we write A as $A_{\mathrm{BNM}} = \left[< a_{ij_T}^P, a_{ij_I}^P, a_{ij_F}^P, a_{ij_T}^N, a_{ij_I}^N, a_{ij_F}^N > \right]_{\mathrm{m} \times \mathrm{n}}$.

3.COMPUTING THE BIPOLAR NEUTROSOPHIC MATRIX OPERATIONS USING PYTHON LANGUAGE

To generate the Python program for inputting the single valued neutrosophic matrices. The procedure is described as follows:

3.1 Checking the matrix is BNM or not

To generate the Python program for deciding for a given the matrix is bipolar neutrosophic matrix or, simple call of the function **BNMChecking ( )** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for BNM Checking
#A1.shape and A2.shape returns (3, 3, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 3 columns
# A.shape[2] = Each bipolar neutrosophic number has 6 tuple as usual
#One can use any matrices having arbitrary dimension
import numpy as np
#A1 is a BNM
A1= np.array([  [[0.000, 0.001, 0.002, -0.003, -0.004, -0.005],  [0.010, 0.011, 0.012, -0.013, -0.014, -0.015] , [0.020, 0.021, 0.022, -0.023, -0.024, -0.025]  ],
[[0.100,0.101,0.102,-0.103,-0.104, -0.105], [0.110,0.111,0.112,-0.113,-0.114,-0.115], [0.120,0.121,0.122,-0.123,-0.124,-0.125]  ],
 [[0.200,0.201,0.202,-0.203,-0.204,-0.205], [0.210, 0.211,0.212,-0.213,-0.214,-0.215], [0.220,0.221,0.222,-0.223,-0.224,-0.225]  ] ])
#A2 is not BNM
A2= np.array([  [[0.000, 0.001, 0.002, -0.003, -0.004, -0.005],  [0.010, 0.011, 0.012, -0.013, -0.014, -0.015] , [0.020, 0.021, 0.022, -0.023, -0.024, -0.025]  ],
[[0.100,0.101,0.102,-0.103,-0.104, -0.105],  [0.110,0.111,0.112,-0.113,-0.114,-0.115],
[0.120,0.121,0.122,-0.123,-0.124,-0.125]  ],
[[0.200,0.201,0.202,-0.203, 0.204,-0.205],  [0.210, 0.211,0.212,-0.213,-0.214,-0.215],
[0.220,0.221,0.222,-0.223,-0.224,-0.225]  ] ])
def BNMChecking (A):
    dimA=A.shape
    control=0
    counter = 0
    for i in range (0,dimA[0]):
        if counter == 1:
            break
        for j in range (0,dimA[0]):
            if counter == 1:
                break
            for  d in range (0, dimA[2]):
                if  counter ==0:
```

```
                    if (d==0 or d==1 or d==2) :
                        if   not (0 <=   A[i][j][d] <= 1):
                            counter=1
                            print (A[i][j], ' is not a bipolar neutrosophic number, so the matrix
is not a BNM')
                            control=1
                            break
                    if   (d==3 or d==4 or d==5) :
                        if not (-1 <=   A[i][j][d] <= 0) :
                            counter=1
                            print (A[i][j], ' is not a bipolar neutrosophic number, so the matrix
is not a BNM')
                            control=1
                            break
                else:
                    print (A[i][j], ' is not a bipolar neutrosophic number, so the matrix is not a
BNM')
                    break
    if control==0:
        print ('The matrix is a BNM')
```

Example 1. In this example we evaluate the checking the matrix C is BNM or not of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python environment like this:

```
Shell  AST
>>> %Run 'bnm checking.py'
>>> C= np.array([ [ [0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2
    ,-0.8]],
                    [[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.
    5,-0.1,-0.9]],
                     [[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0
    .4,-0.2,-0.2]],
                     [[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0
    .5,-0.5,-0.6]] ])

>>> BNMChecking (C)
  The matrix is a BNM
```

3.2. Determining complement of bipolar neutrosophic matrix

For a given BNM A= $\left[<T_{ij}^{P},I_{ij}^{P},F_{ij}^{P},T_{ij}^{N},I_{ij}^{N},F_{ij}^{N}>\right]_{\mathrm{m\times n}}$, the complement of A is defined as follow:

$$A^{c}=\left[<\{1\}-T_{ij}^{P},\{1\}-I_{ij}^{P},\{-1\}-F_{ij}^{P},\{1\}-T_{ij}^{N},\{-1\}-I_{ij}^{N},\{-1\}-F_{ij}^{N}>\right]_{\mathrm{m\times n}} \quad (8)$$

$$A^{c}=\left[<F_{ij}^{P},\{1\}-I_{ij}^{P},T_{ij}^{P},F_{ij}^{N},\{-1\}-I_{ij}^{N},T_{ij}^{N}>\right]_{\mathrm{m\times n}} \quad (9)$$

To generate the Python program for finding complement of bipolar neutrosophic matrix, simple call of the function **BNMCompelementOf()** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for (8)
import numpy as np
A= np.array([    [ [0.3,0.6,1,-0.2,-0.54,-0.4],    [0.1,0.2,0.8,-0.5,-0.34,-0.7]],
                 [ [0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22]],
                 [ [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32] ]
                 ])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = each bipolar neutrosophic number with 6 tuple as usual
def BNMCompelementOf( A ):
    global Ac
    dimA=A.shape                                    # Dimension of the matrix
    Ac= []      # Empty matrix with dimension of A to create complement of A
        for i in range (0,dimA[0]):          # for rows, here 3
        H=[]
        for j in range (0,dimA[1]):    # for columns, here 2
            H.extend([ [ 1-A[i][j][0], 1-A[i][j][1], 1-A[i][j][2], -1-(-A[i][j][3]), -1-(-A[i][j][4]), -1-(-A[i][j][5]) ] ])
            Ac.append(H)
    print    ('A= ', A)
    print ('**********************************************************************')
    print('Ac= ', np.array(Ac))
```

The function BNMCompelementOf (A) the below returns the complement matrix of a given bipolar neutrosophic matrix A for (9).

```
# BNM is representable by 3D Numpy Array ====> row, column and bipolar neutrosophic numbers having 6 tuples for (9)
import numpy as np
A= np.array([    [ [0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7]    ],
                 [ [0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22] ],
                 [ [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32]     ]])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = Each bipolar neutrosophic number with 6 tuple as usual
def BNMCompelementOf( A ):
    global Ac
    dimA=A.shape                                    # Dimension of the matrix
    Ac= []
```

```
    for i in range (0,dimA[0]):          # for rows, here 3
        H=[]
        for j in range (0,dimA[1]):   # for columns, here 2
            H.extend([[ A[i][j][2], 1-A[i][j][1], A[i][j][0], A[i][j][5], -1-(-A[i][j][4]), A[i][j][3] ] ])
        Ac.append(H)
  print   ('A= ', A)
  print ('**********************************************************************')
  print ('**********************************************************************')
  print('Ac= ', np.array(Ac))
```

The bipolar neutrosophic matrix A is a simple example, one can create his/her BNM and try it into the function **BNMCompelementOf** ( ):

3.3. Determining the score, accuracy and certainty matrices of bipolar neutrosophic matrix

To generate the python program for obtaining the score matrix, accuracy of bipolar neutrosophic matrix, simple call of the functions **ScoreMatrix( )**, **AccuracyMatrix( )** and **CertaintyMatrix( )** are defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for (5, 6 and 7)
  import numpy as np
  A= np.array([      [          [0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7]                ],
                     [          [0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22]          ],
                     [          [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32]    ]])
  def ScoreMatrix( A ):
      score=[]
      dimA=A.shape                                  # Dimension of the matrix
      for i in range (0,dimA[0]):                # for rows, here 3
          H=[]
          for j in range (0,dimA[1]):          # for columns, here 2
              H.extend([ [ ( A[i][j][0] + 1 - A[i][j][1] + 1 - A[i][j][2] + 1 + A[i][j][3] - A[i][j][4] - A[i][j][5] )/6 ] ])
          score.append(H)
      print('Score Matrix= ', np.array(score))
  def AccuracyMatrix ( A ):
      accuracy=[]
      dimA=A.shape                                  # Dimension of the matrix
      for i in range (0,dimA[0]):                # for rows, here 3
          H=[]
          for j in range (0,dimA[1]):          # for columns, here 2
              H.extend([ [    A[i][j][0] - A[i][j][2] + A[i][j][3] - A[i][j][5]    ] ])
          accuracy.append(H)
      print('Accuracy Matrix= ', np.array(accuracy))
```

```
def CertaintyMatrix ( A ):
    certainty = []
    dimA=A.shape                                # Dimension of the matrix
    for i in range (0,dimA[0]):                 # for rows, here 3
        H=[]
        for j in range (0,dimA[1]):             # for columns, here 2
            H.extend([ [    A[i][j][0] - A[i][j][5]    ] ])
        certainty.append(H)
    print('Certainty Matrix= ', np.array(certainty))
```

3.4. Computing union of two bipolar neutrosophic matrices

The union of two bipolar neutrosophic matrices A and B is defined as follow:

$$A \cup B = C = \left[< c_{ij_T}^P, c_{ij_I}^P, c_{ij_F}^P, c_{ij_T}^N, c_{ij_I}^N, c_{ij_F}^N >\right]_{\mathrm{m\times n}} \quad (10)$$

where

$c_{ij_T}^P = a_{ij_T}^P \vee b_{ij_T}^P, \quad c_{ij_T}^N = a_{ij_T}^N \wedge b_{ij_T}^N$

$c_{ij_I}^P = a_{ij_I}^P \wedge b_{ij_I}^P, \quad c_{ij_I}^N = a_{ij_I}^N \vee b_{ij_I}^N$

$c_{ij_F}^P = a_{ij_F}^P \wedge b_{ij_F}^P, \quad c_{ij_F}^N = a_{ij_F}^N \vee b_{ij_F}^N$

To generate the python program for finding the union of two bipolar neutrosophic matrices, simple call of the following function **Union(A, B)** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for (10)
import numpy as np
A= np.array([   [       [0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7]    ],
                [          [0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22] ],
                [   [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32]] ])
B= np.array([   [          [0.32,0.4,0.1,-0.25,-0.54,-0.4], [0.13,0.2,0.11,-0.55,-0.35,-0.72]      ],
                [[0.17,0.19,0.66,-0.87,-0.64,-0.92], [0.25,0.36,0.88,-0.33,-0.54,-0.22] ],
                [ [0.15,0.28,0.67,-0.39,-0.27,-0.55],[0.24,0.73,0.28,-0.26,-0.53,-0.52] ]
                ])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = each bipolar neutrosophic number with 6 tuple as usual
union=[]
def Union( A, B ):
    if A.shape == B.shape:
        dimA=A.shape
        for i in range (0,dimA[0]):          # for rows, here 3
            H=[]
            for j in range (0,dimA[1]):      # for columns, here 2
```

```
                    H.extend([[ max(A[i][j][0],B[i][j][0]) , min(A[i][j][1], B[i][j][1]), min(A[i][j][2], B[i][j][2]), max(A[i][j][3],B[i][j][3]), min(A[i][j][4], B[i][j][4]), min(A[i][j][5], B[i][j][5]) ] ])
                union.append(H)
    print('union= ', np.array(union)
```

Example 2. In this example we Evaluate the union of the two bipolar neutrosophic matrices C and D of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python code like this:

C= np.array([[[0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2,-0.8]],[[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.5,-0.1,-0.9]], [[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0.4,-0.2,-0.2]], [[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0.5,-0.5,-0.6]]])

D=

$$\begin{pmatrix} <.3,.4,.3,-.5,-.4,-.2> & <.1,.2,.7,-.5,-.2,-.3> & <.3,.2,.6,-.4,-.8,-.7> & <.2,.1,.3,-.2,-.4,-.4> \\ <.2,.2,.7,-.3,-.3,-.5> & <.3,.5,.6,-.6,-.7,-.4> & <.6,.5,.4,-.3,-.6,-.8> & <.3,.4,.4,-.3,-.5,-.3> \\ <.5,.3,.1,-.4,-.2,-.4> & <.5,.4,.3,-.3,-.8,-.2> & <.5,.8,.6,-.2,-.2,-.4> & <.4,.6,.5,-.1,-.6,-.5> \\ <.6,.1,.7,-.7,-.4,-.8> & <.4,.6,.4,-.4,-.2,-.5> & <.4,.9,.3,-.5,-.5,-.3> & <.4,.5,.4,-.3,-.7,-.4> \end{pmatrix}$$

The bipolar neutrosophic matrix D can be inputted in Python code like this:

D= np.array([[[0.3,0.4, 0.3,-0.5,-0.4,-0.2], [0.1,0.2,0.7,-0.5,-0.2,-0.3], [0.3,0.2,0.6,-0.4,-0.8,-0.7], [0.2,0.1,0.3,-0.2,-0.4,-0.4]], [[0.2,0.2,0.7,-0.3,-0.3,-0.5], [0.3,0.5,0.6,-0.6,-0.7,-0.4], [0.6,0.5,0.4,-0.3,-0.6,-0.8], [0.3,0.4,0.4,-0.3,-0.5,-0.3]], [[0.5,0.3,0.1,-0.4,-0.2,-0.4], [0.5,0.4,0.3,-0.3,-0.8,-0.2], [0.5,0.8,0.6,-0.2,-0.2,-0.4], [0.4,0.6,0.5,-0.1,-0.6,-0.5]], [[0.6,0.1,0.7,-0.7,-0.4,-0.8], [0.4,0.6,0.4,-0.4,-0.2,-0.5], [0.4,0.9,0.3,-0.5,-0.5,-0.3], [0.4,0.5,0.4,-0.3,-0.7,-0.4]]])

So, the union matrix of two bipolar neutrosophic matrices is portrayed as follow

$C_{BNS} \cup D_{BNS}$

$$= \begin{pmatrix} <.5,.4,.2,-.7,-.3,-.2> & <.4,.2,.5,-.7,-.2,-.3> & <.7,.2,.5,-.8,-.7,-.6> & <.2,.1,.3,-.5,-.2,-.4> \\ <.9,.2,.5,-.7,-.3,-.1> & <.7,.5,.6,-.7,-.5,-.1> & <.9,.4,.4,-.3,-.6,-.5> & <.5,.2,.4,-.5,-.1,-.3> \\ <.9,.3,.1,-.6,-.2,-.4> & <.5,.2,.2,-.4,-.7,-.2> & <.9,.5,.5,-.6,-.2,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.1,.2,-.8,-.4,-.1> & <.4,.5,.2,-.5,-.2,-.2> & <.5,.4,.3,-.5,-.5,-.2> & <.4,.4,.4,-.5,-.5,-.4> \end{pmatrix}$$

The result of union matrix of two bipolar neutrosophic matrices C and D can be obtained by the call of the command Union (C, D):

```
>>> Union(C, D)
Union =
 [[[ 0.5   0.4   0.2 -0.7 -0.3 -0.2]  [ 0.4   0.2   0.5 -0.7 -0.2 -0.3] [ 0.7   0.2   0.5 -0.8 -0.7 -0.6]   [ 0.2   0.1   0.3 -0.5 -0.2 -0.4]]
 [[ 0.9   0.2   0.5 -0.7 -0.3 -0.1]  [ 0.7   0.5   0.6 -0.7 -0.5 -0.1]   [ 0.9   0.4   0.4 -0.3 -0.6 -0.5]   [ 0.5   0.2   0.4 -0.5 -0.1 -0.3]]
 [[ 0.9   0.3   0.1 -0.6 -0.2 -0.4]  [ 0.5   0.2   0.2 -0.4 -0.7 -0.2]   [ 0.9   0.5   0.5 -0.6 -0.2 -0.2]   [ 0.7   0.5   0.3 -0.4 -0.2 -0.2]]
 [[ 0.9   0.1   0.2 -0.8 -0.4 -0.1]  [ 0.4   0.5   0.2 -0.5 -0.2 -0.2]   [ 0.5   0.4   0.3 -0.5 -0.5 -0.2]   [ 0.4   0.4   0.4 -0.5 -0.5 -0.4]]]
```

3.5. Computing intersection of two bipolar neutrosophic matrices

The union of two bipolar neutrosophic matrices A and B is defined as follow:

$$A \cap B = D = \left[< d^P_{ij_T}, d^P_{ij_I}, d^P_{ij_F}, d^N_{ij_T}, d^N_{ij_I}, d^N_{ij_F} >\right]_{\mathrm{m\times n}} \quad (11)$$

Where

$d^P_{ij_T} = a^P_{ij_T} \wedge b^P_{ij_T}, \quad d^N_{ij_T} = a^N_{ij_T} \vee b^N_{ij_T}$

$d^P_{ij_I} = a^P_{ij_I} \vee b^P_{ij_I}, \quad d^N_{ij_I} = a^N_{ij_I} \wedge b^N_{ij_I}$

$d^P_{ij_F} = a^P_{ij_F} \vee b^P_{ij_F}, \quad d^N_{ij_F} = a^N_{ij_F} \wedge b^N_{ij_F}$

To generate the python program for finding the intersection of two bipolar neutrosophic matrices, simple call of the function Intersection (A, B) is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for (11)
import numpy as np
A= np.array([   [         [0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7]                ],
                [         [0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22]          ],
                [          [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32]     ]
                ])
B= np.array([   [         [0.32,0.4,0.1,-0.25,-0.54,-0.4], [0.13,0.2,0.11,-0.55,-0.35,-0.72]         ],
                [         [0.17,0.19,0.66,-0.87,-0.64,-0.92], [0.25,0.36,0.88,-0.33,-0.54,-0.22]     ],
                [          [0.15,0.28,0.67,-0.39,-0.27,-0.55],[0.24,0.73,0.28,-0.26,-0.53,-0.52]     ]])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = each bipolar neutrosophic number with 6 tuple as usual
intersection=[]
def Intersection( A, B ):
    if A.shape == B.shape:
        dimA=A.shape
        for i in range (0,dimA[0]):          # for rows, here 3
            H=[]
            for j in range (0,dimA[1]):   # for columns, here 2
                H.extend([[ min(A[i][j][0],B[i][j][0]) , max(A[i][j][1], B[i][j][1]), max(A[i][j][2], B[i][j][2]), min(A[i][j][3],B[i][j][3]), max(A[i][j][4], B[i][j][4]), max(A[i][j][5], B[i][j][5]) ] ])
                intersection.append(H)
    print('Intersection= ', np.array(intersection))
```

Example 3. In this example we evaluate the intersection of the two bipolar neutrosophic matrices C and D of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python code like this:

C= np.array([[[0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2,-0.8]], [[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.5,-0.1,-0.9]], [[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0.4,-0.2,-0.2]], [[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0.5,-0.5,-0.6]])

D=

$$\begin{pmatrix} <.3,.4,.3,-.5,-.4,-.2> & <.1,.2,.7,-.5,-.2,-.3> & <.3,.2,.6,-.4,-.8,-.7> & <.2,.1,.3,-.2,-.4,-.4> \\ <.2,.2,.7,-.3,-.3,-.5> & <.3,.5,.6,-.6,-.7,-.4> & <.6,.5,.4,-.3,-.6,-.8> & <.3,.4,.4,-.3,-.5,-.3> \\ <.5,.3,.1,-.4,-.2,-.4> & <.5,.4,.3,-.3,-.8,-.2> & <.5,.8,.6,-.2,-.2,-.4> & <.4,.6,.5,-.1,-.6,-.5> \\ <.6,.1,.7,-.7,-.4,-.8> & <.4,.6,.4,-.4,-.2,-.5> & <.4,.9,.3,-.5,-.5,-.3> & <.4,.5,.4,-.3,-.7,-.4> \end{pmatrix}$$

The bipolar neutrosophic matrix D can be inputted in Python code like this:

D= np.array([[[0.3, 0.4, 0.3,-0.5,-0.4,-0.2], [0.1,0.2,0.7,-0.5,-0.2,-0.3], [0.3,0.2,0.6,-0.4,-0.8,-0.7], [0.2,0.1,0.3,-0.2,-0.4,-0.4]], [[0.2,0.2,0.7,-0.3,-0.3,-0.5], [0.3,0.5,0.6,-0.6,-0.7,-0.4], [0.6,0.5,0.4,-0.3,-0.6,-0.8], [0.3,0.4,0.4,-0.3,-0.5,-0.3]],

[[0.5,0.3,0.1,-0.4,-0.2,-0.4], [0.5,0.4,0.3,-0.3,-0.8,-0.2], [0.5,0.8,0.6,-0.2,-0.2,-0.4], [0.4,0.6,0.5,-0.1,-0.6,-0.5]],

[[0.6,0.1,0.7,-0.7,-0.4,-0.8], [0.4,0.6,0.4,-0.4,-0.2,-0.5], [0.4,0.9,0.3,-0.5,-0.5,-0.3], [0.4,0.5,0.4,-0.3,-0.7,-0.4]]])

So, the intersection matrix of two bipolar neutrosophic matrices is portrayed as follow

$C_{BNS} \cap D_{BNS}$

$$= \begin{pmatrix} <.3,.7,.3,-.5,-.4,-.6> & <.1,.4,.7,-.5,-.8,-.4> & <.3,.7,.6,-.4,-.8,-.7> & <.1,.5,.7,-.2,-.4,-.8> \\ <.2,.7,.7,-.3,-.7,-.5> & <.3,.6,.8,-.6,-.7,-.4> & <.6,.5,.6,-.1,-.7,-.8> & <.3,.4,.7,-.3,-.5,-.9> \\ <.5,.4,.2,-.4,-.3,-.7> & <.2,.4,.3,-.3,-.7,-.4> & <.5,.8,.6,-.2,-.5,-.4> & <.4,.6,.5,-.1,-.6,-.5> \\ <.6,.7,.7,-.7,-.6,-.8> & <.3,.6,.4,-.4,-.5,-.5> & <.4,.9,.5,-.1,-.7,-.3> & <.2,.5,.8,-.3,-.7,-.6> \end{pmatrix}$$

The result of intersection matrix of two bipolar neutrosophic matrices C and D can be obtained by the call of the command Intersection (C, D):

>>> Intersection (C, D)

Intersection =

[[[0.3 0.7 0.3 -0.5 -0.4 -0.6] [0.1 0.4 0.7 -0.5 -0.8 -0.4] [0.3 0.7 0.6 -0.4 -0.8 -0.7] [0.1 0.5 0.7 -0.2 -0.4 -0.8]]

[[0.2 0.7 0.7 -0.3 -0.7 -0.5] [0.3 0.6 0.8 -0.6 -0.7 -0.4] [0.6 0.5 0.6 -0.1 -0.7 -0.8] [0.3 0.4 0.7 -0.3 -0.5 -0.9]]

[[0.5 0.4 0.2 -0.4 -0.3 -0.7] [0.2 0.4 0.3 -0.3 -0.8 -0.4] [0.5 0.8 0.6 -0.2 -0.5 -0.4] [0.4 0.6 0.5 -0.1 -0.6 -0.5]]

[[0.6 0.7 0.7 -0.7 -0.6 -0.8] [0.3 0.6 0.4 -0.4 -0.5 -0.5] [0.4 0.9 0.5 -0.1 -0.7 -0.3] [0.2 0.5 0.8 -0.3 -0.7 -0.6]]]

3.6. Computing addition operation of two bipolar neutrosophic matrices.

The addition of two bipolar neutrosophic matrices A and B is defined as follow:

$$A \oplus B = S = \left[< s^P_{ij_T}, s^P_{ij_I}, s^P_{ij_F}, s^N_{ij_T}, s^N_{ij_I}, s^N_{ij_F} >\right]_{\mathrm{m\times n}} \quad (12)$$

Where

$$s^P_{ij_T} = a^P_{ij_T} + b^P_{ij_T} - a^P_{ij_T}.b^P_{ij_T}, \qquad s^N_{ij_T} = -\left(a^N_{ij_T}.b^N_{ij_T}\right)$$

$$s^P_{ij_I} = a^P_{ij_I}.b^P_{ij_I}, \qquad s^N_{ij_I} = -\left(-a^N_{ij_I} - b^N_{ij_I} - a^N_{ij_I}.b^N_{ij_I}\right)$$

$$s^P_{ij_F} = a^P_{ij_F}.b^P_{ij_F}, \qquad s^N_{ij_F} = -\left(-a^N_{ij_F} - b^N_{ij_F} - a^N_{ij_F}.b^N_{ij_F}\right)$$

To generate the python program for obtaining the addition of two bipolar neutrosophic matrices, simple call of the function **Addition (A, B)** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for (12)
```

```
import numpy as np
A= np.array([    [[0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7] ],
                 [[0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22]],
                 [ [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32] ]])
B= np.array([    [[0.32,0.4,0.1,-0.25,-0.54,-0.4], [0.13,0.2,0.11,-0.55,-0.35,-0.72] ],
                 [[0.17,0.19,0.66,-0.87,-0.64,-0.92], [0.25,0.36,0.88,-0.33,-0.54,-0.22] ],
                 [[0.15,0.28,0.67,-0.39,-0.27,-0.55],[0.24,0.73,0.28,-0.26,-0.53,-0.52]      ]])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = each bipolar neutrosophic number with 6 tuples as usual
addition=[]
def Addition( A, B ):
    if A.shape == B.shape:
        dimA=A.shape
        for i in range (0,dimA[0]):          # for rows, here 3
            H=[]
            for j in range (0,dimA[1]):    # for columns, here 2
                H.extend([[A[i][j][0]+B[i][j][0]-A[i][j][0]*B[i][j][0],      A[i][j][1]* B[i][j][1],
A[i][j][2]* B[i][j][2] -(-A[i][j][3]*B[i][j][3]), -(-A[i][j][4]-B[i][j][4] -A[i][j][4]*B[i][j][4] ), -(-A[i][j][5]-B[i][j][5]-A[i][j][5]*B[i][j][5]) ]])
                addition.append(H)
    print('Addition= ', np.array(addition))
```

Example 4. In this example we evaluate the addition of the two bipolar neutrosophic matrices C and D of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python code like this:

```
C= np.array([ [ [0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2,-0.8]],[[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.5,-0.1,-0.9]],

[[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0.4,-0.2,-0.2]],

[[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0.5,-0.5,-0.6]]])
```

D=

$$\begin{pmatrix} <.3,.4,.3,-.5,-.4,-.2> & <.1,.2,.7,-.5,-.2,-.3> & <.3,.2,.6,-.4,-.8,-.7> & <.2,.1,.3,-.2,-.4,-.4> \\ <.2,.2,.7,-.3,-.3,-.5> & <.3,.5,.6,-.6,-.7,-.4> & <.6,.5,.4,-.3,-.6,-.8> & <.3,.4,.4,-.3,-.5,-.3> \\ <.5,.3,.1,-.4,-.2,-.4> & <.5,.4,.3,-.3,-.8,-.2> & <.5,.8,.6,-.2,-.2,-.4> & <.4,.6,.5,-.1,-.6,-.5> \\ <.6,.1,.7,-.7,-.4,-.8> & <.4,.6,.4,-.4,-.2,-.5> & <.4,.9,.3,-.5,-.5,-.3> & <.4,.5,.4,-.3,-.7,-.4> \end{pmatrix}$$

The bipolar neutrosophic matrix D can be inputted in Python code like this:

D= np.array([[[0.3,0.4,0.3,-0.5,-0.4,-0.2], [0.1,0.2,0.7,-0.5,-0.2,-0.3], [0.3,0.2,0.6,-0.4,-0.8,-0.7], [0.2,0.1,0.3,-0.2,-0.4,-0.4]], [[0.2,0.2,0.7,-0.3,-0.3,-0.5], [0.3,0.5,0.6,-0.6,-0.7,-0.4], [0.6,0.5,0.4,-0.3,-0.6,-0.8], [0.3,0.4,0.4,-0.3,-0.5,-0.3]],

[[0.5,0.3,0.1,-0.4,-0.2,-0.4], [0.5,0.4,0.3,-0.3,-0.8,-0.2], [0.5,0.8,0.6,-0.2,-0.2,-0.4], [0.4,0.6,0.5,-0.1,-0.6,-0.5]], [[0.6,0.1,0.7,-0.7,-0.4,-0.8], [0.4,0.6,0.4,-0.4,-0.2,-0.5], [0.4,0.9,0.3,-0.5,-0.5,-0.3], [0.4,0.5,0.4,-0.3,-0.7,-0.4]]])

So, the addition matrix of two bipolar neutrosophic matrices is portrayed as follow

$C_{BNS} \oplus D_{BNS} =$

$$\begin{pmatrix} <.65,.28,.06,-.35,-.58,-.68> & <.46,.08,.35,-.35,-.84,-.58> & <.79,.14,.30,-.32,-.94,-.88> & <.28,.05,.21,-.10,-.52,-.88> \\ <.92,.14,.35,-.21,-.79,-.55> & <.79,.30,.48,-.42,-.85,-.46> & <.96,.20,.24,-.03,-.88,-.90> & <.65,.08,.28,-.15,-.55,-.93> \\ <.65,.12,.02,-.24,-.44,-.82> & <.60,.08,.06,-.12,-.94,-.52> & <.95,.40,.30,-.12,-.60,-.52> & <.82,.30,.15,-.04,-.68,-.60> \\ <.96,.07,.14,-.56,-.76,-.82> & <.58,.30,.08,-.20,-.60,-.60> & <.70,.36,.15,-.05,-.85,-.44> & <.52,.20,.32,-.15,-.85,-.76> \end{pmatrix}$$

The result of addition matrix of two bipolar neutrosophic matrices C and D can be obtained by the call of the command addition (C, D):

>>> Addition(C, D)

Addition=

[[[0.65 0.28 0.06 0.35 -0.58 -0.68] [0.46 0.08 0.35 0.35 -0.84 -0.58] [0.79 0.14 0.3 0.32 -0.94 -0.88] [0.28 0.05 0.21 0.1 -0.52 -0.88]]

[[0.92 0.14 0.35 0.21 -0.79 -0.55] [0.79 0.3 0.48 0.42 -0.85 -0.46] [0.96 0.2 0.24 0.03 -0.88 -0.9] [0.65 0.08 0.28 0.15 -0.55 -0.93]]

[[0.95 0.12 0.02 0.24 -0.44 -0.82] [0.6 0.08 0.06 0.12 -0.94 -0.52] [0.95 0.4 0.3 0.12 -0.6 -0.52] [0.82 0.3 0.15 0.04 -0.68 -0.6]]

[[0.96 0.07 0.14 0.56 -0.76 -0.82] [0.58 0.3 0.08 0.2 -0.6 -0.6] [0.7 0.36 0.15 0.05 -0.85 -0.44] [0.52 0.2 0.32 0.15 -0.85 -0.76]]]

3.7. Computing product of two bipolar neutrosophic matrices

The product of two bipolar neutrosophic matrices A and B is defined as follow:

$$A \odot B = R = \left[< r^P_{ij_T}, r^P_{ij_I}, r^P_{ij_F}, r^N_{ij_T}, r^N_{ij_I}, r^N_{ij_F} >\right]_{\mathrm{m}\times \mathrm{n}} \quad (13)$$

Where

$r^P_{ij_T} = a^P_{ij_T}.b^P_{ij_T}, \quad r^N_{ij_T} = -\left(-a^N_{ij_T} - b^N_{ij_T} - a^N_{ij_T}.b^N_{ij_T}\right)$

$r^P_{ij_I} = a^P_{ij_I} + b^P_{ij_I} - a^P_{ij_I}.b^P_{ij_I}, \quad r^N_{ij_I} = -\left(a^N_{ij_I}.b^N_{ij_I}\right)$

$r^P_{ij_F} = a^P_{ij_F} + b^P_{ij_F} - a^P_{ij_F}.b^P_{ij_F}, \quad r^N_{ij_F} = -\left(a^N_{ij_F}.b^N_{ij_F}\right)$

To generate the python program for finding the product operation of two bipolar neutrosophic matrices, simple call of the function **Product (A, B)** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for (13)
import numpy as np
A= np.array([  [      [0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7]          ],
               [      [0.1,0.12,0,-0.27,-0.44,-0.92], [0.5,0.33,0.58,-0.33,-0.24,-0.22]    ],
               [      [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32] ]])
B= np.array([  [      [0.32,0.4,0.1,-0.25,-0.54,-0.4], [0.13,0.2,0.11,-0.55,-0.35,-0.72]          ],
```

```
                [    [0.17,0.19,0.66,-0.87,-0.64,-0.92], [0.25,0.36,0.88,-0.33,-0.54,-0.22]    ],
                [    [0.15,0.28,0.67,-0.39,-0.27,-0.55],[0.24,0.73,0.28,-0.26,-0.53,-0.52]    ]])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = each bipolar neutrosophic number with 6 tuple as usual
product=[]
def Product( A, B ):
    if A.shape == B.shape:
        dimA=A.shape
        for i in range (0,dimA[0]):          # for rows, here 3
            H=[]
            for j in range (0,dimA[1]):    # for columns, here 2
                H.extend([[ A[i][j][0]*B[i][j][0]) , A[i][j][1]+ B[i][j][1]- (A[i][j][1]*B[i][j][1]),
A[i][j][2]+ B[i][j][2]- (A[i][j][2]*B[i][j][2]), -(-A[i][j][3]-B[i][j][3]-A[i][j][3]*B[i][j][3]), -(A[i][j][4]*
B[i][j][4]), -(A[i][j][5]* B[i][j][5]) ] ])
                product.append(H)
    print(' Product = ', np.array(product))
```

Example 5. In this example we evaluate the product of the two bipolar neutrosophic matrices C and D of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python code like this:

C= np.array([[[0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2,-0.8]], [[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.5,-0.1,-0.9]], [[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0.4,-0.2,-0.2]], [[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0.5,-0.5,-0.6]]])

D=$\begin{pmatrix} <.3,.4,.3,-.5,-.4,-.2> & <.1,.2,.7,-.5,-.2,-.3> & <.3,.2,.6,-.4,-.8,-.7> & <.2,.1,.3,-.2,-.4,-.4> \\ <.2,.2,.7,-.3,-.3,-.5> & <.3,.5,.6,-.6,-.7,-.4> & <.6,.5,.4,-.3,-.6,-.8> & <.3,.4,.4,-.3,-.5,-.3> \\ <.5,.3,.1,-.4,-.2,-.4> & <.5,.4,.3,-.3,-.8,-.2> & <.5,.8,.6,-.2,-.2,-.4> & <.4,.6,.5,-.1,-.6,-.5> \\ <.6,.1,.7,-.7,-.4,-.8> & <.4,.6,.4,-.4,-.2,-.5> & <.4,.9,.3,-.5,-.5,-.3> & <.4,.5,.4,-.3,-.7,-.4> \end{pmatrix}$

The bipolar neutrosophic matrix D can be inputted in Python code like this:

D= np.array([[[0.3,0.4,0.3,-0.5,-0.4,-0.2], [0.1,0.2,0.7,-0.5,-0.2,-0.3], [0.3,0.2,0.6,-0.4,-0.8,-0.7], [0.2,0.1,0.3,-0.2,-0.4,-0.4]], [[0.2,0.2,0.7,-0.3,-0.3,-0.5], [0.3,0.5,0.6,-0.6,-0.7,-0.4], [0.6,0.5,0.4,-0.3,-0.6,-0.8], [0.3,0.4,0.4,-0.3,-0.5,-0.3]],

[[0.5,0.3,0.1,-0.4,-0.2,-0.4], [0.5,0.4,0.3,-0.3,-0.8,-0.2], [0.5,0.8,0.6,-0.2,-0.2,-0.4], [0.4,0.6,0.5,-0.1,-0.6,-0.5]],

[[0.6,0.1,0.7,-0.7,-0.4,-0.8], [0.4,0.6,0.4,-0.4,-0.2,-0.5], [0.4,0.9,0.3,-0.5,-0.5,-0.3], [0.4,0.5,0.4,-0.3,-0.7,-0.4]]])

So, the product matrix of two bipolar neutrosophic matrices is portrayed as follow

$C_{BNS} \odot D_{BNS}$=

$$\begin{pmatrix} <.15,.82,.44,-.85,-.12,-.12> & <.04,.52,.85,-.85,-1.16,-.12> & <.21,.76,.80,-.88,-.56,-.42> & <.02,.55,.79,-.60,-.008,-.32> \\ <.18,.76,.85,-.79,-.21,-.05> & <.21,.80,.92,-.88,-.35,-.04> & <.54,.70,.76,-.37,-.42,-0.40> & <.15,.52,.82,-.65,-.05,-.27> \\ <.45,.58,.28,-.76,-.06,-.28> & <.10,.52,.44,-.58,-.56,-.08> & <.45,.90,.80,-.68,-.10,-.08> & <.28,.80,.65,-.46,-.12,-.10> \\ <.54,.73,.76,-.94,-.24,-.08> & <.12,.80,.52,-.70,-.10,-.10> & <.20,.94,.65,-.55,-.35,-.06> & <.08,.70,.88,-.65,-.35,-.24> \end{pmatrix}$$

The result of product matrix of two bipolar neutrosophic matrices C and D can be obtained by the call of the command Product (C, D):

>>> Product(C, D)

Product=

[[[0.15 0.82 0.44 -0.85 -0.12 -0.12] [0.04 0.52 0.85 -0.85 -0.16 -0.12] [0.21 0.76 0.8 -0.88 -0.56 -0.42] [0.02 0.55 0.79 -0.6 -0.08 -0.32]]

[[0.18 0.76 0.85 -0.79 -0.21 -0.05] [0.21 0.8 0.92 -0.88 -0.35 -0.04] [0.54 0.7 0.76 -0.37 -0.42 -0.4] [0.15 0.52 0.82 -0.65 -0.05 -0.27]]

[[0.45 0.58 0.28 -0.76 -0.06 -0.28] [0.1 0.52 0.44 -0.58 -0.56 -0.08] [0.45 0.9 0.8 -0.68 -0.1 -0.08] [0.28 0.8 0.65 -0.46 -0.12 -0.1]]

[[0.54 0.73 0.76 -0.94 -0.24 -0.08] [0.12 0.8 0.52 -0.7 -0.1 -0.1] [0.2 0.94 0.65 -0.55 -0.35 -0.06] [0.08 0.7 0.88 -0.65 -0.35 -0.24]]]

3.8. Computing transpose of bipolar neutrosophic matrix

To generate the python program for finding the transpose of bipolar neutrosophic matrix, simple call of the function **Transpose (A)** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for transpose
import numpy as np
A=np.array([[ [0.3,0.6,1,-0.2,-0.54,-0.4], [0.1,0.2,0.8,-0.5,-0.34,-0.7]    ],
                  [ [0.1,0.12,0,-0.27,-0.44,-0.92],[0.5,0.33,0.58,-0.33,-0.24,-0.22]],
                 [ [0.11,0.22,0.6,-0.29,-0.24,-0.52],[0.22,0.63,0.88,-0.28,-0.54,-0.32]] ])
#A.shape gives (3, 2, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 2 columns
# A.shape[2] = each bipolar neutrosophic number with 6 tuple as usual
def Transpose( A ):
    DimA= A. shape
    print (' the matrix ', DimA[0],' x ', DimA[1], ' dimension')
    trA = A.transpose()
    DimtrA= trA. shape
    print ('\n')
    print (' its transpose ', DimtrA[1],' x ', DimtrA[2], ' dimension')
    print ('\n' )
    print(' Transpose = ', trA)
```

Example 6. In this example we evaluate the transpose of the bipolar neutrosophic matrix C of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python code like this:

C= np.array([[[0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2,-0.8]], [[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.5,-0.1,-0.9]],

[[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0.4,-0.2,-0.2]],

[[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0.5,-0.5,-0.6]]])

So, the transpose matrix of bipolar neutrosophic matrices is portrayed as follow

<0.50, 0.70, 0.20,-0.70, -0.30, -0.60>	**<0.90, 0.70, 0.50,-0.70, -0.70, -0.10>**	**<0.30, 0.40, 0.20,-0.60, -0.30, -0.70>**	**<0.90, 0.70, 0.20,-0.80, -0.60, -0.10>**
<0.40, 0.40, 0.50,-0.70, -0.80, -0.40>	**<0.70, 0.60, 0.80,-0.70, -0.50, -0.10>**	**<0.20, 0.20, 0.20,-0.40, -0.70, -0.40>**	**<0.30, 0.50, 0.20,-0.50, -0.50, -0.20>**
<0.70, 0.70, 0.50,-0.80, -0.70, -0.60>	**<0.90, 0.40, 0.60,-0.10, -0.70, -0.50>**	**<0.90, 0.50, 0.50,-0.60, -0.50, -0.20>**	**<0.50, 0.40, 0.50,-0.10, -0.70, -0.20>**
<0.10, 0.50, 0.70,-0.50, -0.20, -0.80>	**<0.50, 0.20, 0.70,-0.50, -0.10, -0.90>**	**<0.70, 0.50, 0.30,-0.40, -0.20, -0.20>**	**<0.20, 0.40, 0.80,-0.50, -0.50, -0.60>**

>>> Transpose(C)

The matrix 4 x4 dimension

Its transpose 4 x 4 dimension

Transpose =

[[[0.5 0.9 0.9 0.9] [0.4 0.7 0.2 0.3] [0.7 0.9 0.9 0.5] [0.1 0.5 0.7 0.2]]

[[0.7 0.7 0.4 0.7] [0.4 0.6 0.2 0.5] [0.7 0.4 0.5 0.4] [0.5 0.2 0.5 0.4]]

[[0.2 0.5 0.2 0.2] [0.5 0.8 0.2 0.2] [0.5 0.6 0.5 0.5] [0.7 0.7 0.3 0.8]]

[[-0.7 -0.7 -0.6 -0.8] [-0.7 -0.7 -0.4 -0.5] [-0.8 -0.1 -0.6 -0.1] [-0.5 -0.5 -0.4 -0.5]]

[[-0.3 -0.7 -0.3 -0.6] [-0.8 -0.5 -0.7 -0.5] [-0.7 -0.7 -0.5 -0.7] [-0.2 -0.1 -0.2 -0.5]]

[[-0.6 -0.1 -0.7 -0.1] [-0.4 -0.1 -0.4 -0.2] [-0.6 -0.5 -0.2 -0.2] [-0.8 -0.9 -0.2 -0.6]]]

3.9 Computing composition of two bipolar neutrosophic matrices

To generate the python program for finding the composition of two bipolar neutrosophic matrices, simple call of the function **Composition ()** is defined as follow:

```
# BNM is represented by 3D Numpy Array => row, column and bipolar number with 6 tuples for Composition
#A.shape and B.shape returns (3, 3, 6) the dimension of A. (row, column, numbers of element (Bipolar Neutrosophic Number, 6 elements) )
# A.shape[0] = 3 rows
# A.shape[1] = 3 columns
# A.shape[2] = Each bipolar neutrosophic number has 6 tuple as usual
#One can use matrices with any dimensions but dimensions of two matrices must be the same and nxn
```

```
import math
import numpy as np
A= np.array( [ [ [0.3, 0.6, 1, -0.2, -0.54, -0.4], [0.1, 0.2, 0.8, -0.5, -0.34, -0.7], [0.020,0.021,0.022,-0.023,-0.024,-0.025]  ],
[ [0.17,0.19,0.66,-0.87,-0.64,-0.92], [0.25,0.36,0.88,-0.33,-0.54,-0.22], [0.120,0.121,0.122,-0.123,-0.124,-0.125]  ],
[ [0.15,0.28,0.67,-0.39,-0.27,-0.55],[0.24,0.73,0.28,-0.26,-0.53,-0.52], [0.220,0.221,0.222,-0.223,-0.224,-0.225]  ]  ] )
B=np.array([ [0.11,0.22,0.6,-0.29,-0.24,-0.52], [0.32,0.4,0.1,-0.25,-0.54,-0.4], [0.13,0.2,0.11,-0.55,-0.35,-0.72] ],
[ [0.100,0.101,0.102,-0.103,-0.104,-0.105], [1,0.111,0.112,-0.113,-0.114,-0.115], [0.720,0.821,0.152,-0.143,-0.194,-0.1]  ],
[ [0,0.73,0.202,-0.203,-0.204,-0.205],  [0.22,0.63,0.88,-0.28,-0.54,-0.32], [0.3,0,0.47,-0.223,-0.254,-0.295] ] ] )
def Composition( A, B ):
    global composition
    composition=[]
    dimA = A.shape
    H=[ ]
    if A.shape == B.shape and dimA[0] == dimA[1]:
        for i in range (0,dimA[0]):
            for j in range (0,dimA[0]):
                counter0=0
                for d in range (0, dimA[0]):
                    if counter0 ==0:
                        maxtt  =     [ A[i][d][0],B[d][j][0] ]
                        maxT = min(maxtt)
                        minii =  [A[i][d][1],B[d][j][1] ]
                        minI =  max(minii)
                        minff = [ A[i][d][2],B[d][j][2]]
                        minF = max( minff)
                        minntt= [    A[i][d][3],B[d][j][3] ]
                        minNT = max (minntt)
                        maxnii = [ A[i][d][4],B[d][j][4]  ]
                        maxNI =  min( maxnii  )
                        maxnff= [    A[i][d][5],B[d][j][5]  ]
                        maxNF = min (maxnff)
                        counter0  = 1
                    else:
                        maxT1          = [  A[i][d][0],B[d][j][0]  ]
                        maxT11        = min(maxT1)
                        maxT112      = [  maxT11 ,  maxT  ]
```

```
                        maxT              =   max(maxT112)
                        minI1         =   [ A[i][d][1],B[d][j][1]   ]
                        minI11      =   max(minI1)
                        minI112   = [ minI11, minI      ]
                        minI            = min(   minI112)
                        minF1           =      [ A[i][d][2],B[d][j][2]   ]
                        minF11        =   max(minF1)
                        minF112      = [ minF11, minF]
                        minF              = min(minF112 )
                        minNT1        =    [   A[i][d][3],B[d][j][3]          ]
                        minNT11      =   max( minNT1     )
                        minNT112   =    [ minNT11, minNT ]
                        minNT           =   min( minNT112   )
                        maxNI1          = [ A[i][d][4],B[d][j][4] ]
                        maxNI11       = min(   maxNI1     )
                        maxNI112    = [   maxNI11, maxNI   ]
                        maxNI             = max(maxNI112)
                        maxNF1       =   [ A[i][d][5],B[d][j][5]   ]
                        maxNF11    =   min ( maxNF1 )
                        maxNF112 =   [ maxNF11, maxNF ]
                        maxNF          =   max (   maxNF112   )
                H.append( [maxT,   minI, minF, minNT, maxNI, maxNF] )
        composition.extend(H)
    global nested
    nested = [   ]
    for k in range( int(math.sqrt(len(composition))) ):
                        nested.append(composition[k:k+int(math.sqrt(len(composition))) ] )
    print('Composition= ', np.array(nested))
```

Example 7. In this example we evaluate the composition of the two bipolar neutrosophic matrices C and D of order 4X4:

C=

$$\begin{pmatrix} <.5,.7,.2,-.7,-.3,-.6> & <.4,.4,.5,-.7,-.8,-.4> & <.7,.7,.5,-.8,-.7,-.6> & <.1,.5,.7,-.5,-.2,-.8> \\ <.9,.7,.5,-.7,-.7,-.1> & <.7,.6,.8,-.7,-.5,-.1> & <.9,.4,.6,-.1,-.7,-.5> & <.5,.2,.7,-.5,-.1,-.9> \\ <.9,.4,.2,-.6,-.3,-.7> & <.2,.2,.2,-.4,-.7,-.4> & <.9,.5,.5,-.6,-.5,-.2> & <.7,.5,.3,-.4,-.2,-.2> \\ <.9,.7,.2,-.8,-.6,-.1> & <.3,.5,.2,-.5,-.5,-.2> & <.5,.4,.5,-.1,-.7,-.2> & <.2,.4,.8,-.5,-.5,-.6> \end{pmatrix}$$

The bipolar neutrosophic matrix C can be inputted in Python code like this:

```
C= np.array([ [ [0.5,0.7,0.2,-0.7,-0.3,-0.6], [0.4,0.4,0.5,-0.7,-0.8,-0.4], [0.7,0.7,0.5,-0.8,-0.7,-0.6], [0.1,0.5,0.7,-0.5,-0.2,-
0.8]], [[0.9,0.7,0.5,-0.7,-0.7,-0.1], [0.7,0.6,0.8,-0.7,-0.5,-0.1], [0.9,0.4,0.6,-0.1,-0.7,-0.5], [0.5,0.2,0.7,-0.5,-0.1,-0.9]],
[[0.9,0.4,0.2,-0.6,-0.3,-0.7], [0.2,0.2,0.2,-0.4,-0.7,-0.4], [0.9,0.5,0.5,-0.6,-0.5,-0.2], [0.7,0.5,0.3,-0.4,-0.2,-0.2]],
[[0.9,0.7,0.2,-0.8,-0.6,-0.1], [0.3,0.5,0.2,-0.5,-0.5,-0.2], [0.5,0.4,0.5,-0.1,-0.7,-0.2], [0.2,0.4,0.8,-0.5,-0.5,-0.6]] ])
```

D=

$$\begin{pmatrix} <.3,.4,.3,-.5,-.4,-.2> & <.1,.2,.7,-.5,-.2,-.3> & <.3,.2,.6,-.4,-.8,-.7> & <.2,.1,.3,-.2,-.4,-.4> \\ <.2,.2,.7,-.3,-.3,-.5> & <.3,.5,.6,-.6,-.7,-.4> & <.6,.5,.4,-.3,-.6,-.8> & <.3,.4,.4,-.3,-.5,-.3> \\ <.5,.3,.1,-.4,-.2,-.4> & <.5,.4,.3,-.3,-.8,-.2> & <.5,.8,.6,-.2,-.2,-.4> & <.4,.6,.5,-.1,-.6,-.5> \\ <.6,.1,.7,-.7,-.4,-.8> & <.4,.6,.4,-.4,-.2,-.5> & <.4,.9,.3,-.5,-.5,-.3> & <.4,.5,.4,-.3,-.7,-.4> \end{pmatrix}$$

The bipolar neutrosophic matrix D can be inputted in Python code like this:

D= np.array([[[0.3,0.4,0.3,-0.5,-0.4,-0.2], [0.1,0.2,0.7,-0.5,-0.2,-0.3], [0.3,0.2,0.6,-0.4,-0.8,-0.7], [0.2,0.1,0.3,-0.2,-0.4,-0.4]], [[0.2,0.2,0.7,-0.3,-0.3,-0.5], [0.3,0.5,0.6,-0.6,-0.7,-0.4], [0.6,0.5,0.4,-0.3,-0.6,-0.8], [0.3,0.4,0.4,-0.3,-0.5,-0.3]], [[0.5,0.3,0.1,-0.4,-0.2,-0.4], [0.5,0.4,0.3,-0.3,-0.8,-0.2], [0.5,0.8,0.6,-0.2,-0.2,-0.4], [0.4,0.6,0.5,-0.1,-0.6,-0.5]], [[0.6,0.1,0.7,-0.7,-0.4,-0.8], [0.4,0.6,0.4,-0.4,-0.2,-0.5], [0.4,0.9,0.3,-0.5,-0.5,-0.3], [0.4,0.5,0.4,-0.3,-0.7,-0.4]]])

So, the composition matrix of two bipolar neutrosophic matrices is portrayed as follow

$C_{BNS} \odot D_{BNS}$=

$$\begin{pmatrix} <.5,.4,.3,-.5,-.4,-.5> & <.5,.5,.5,-.6,-.2,-.4> & <.5,.5,.5,-.5,-.5,-.6> & <.4,.4,.3,-.3,-.4,-.4> \\ <.5,.5,.5,-.6,-.2,-.4> & <.5,.5\ .5,-.5,-.5,-.6> & <.4,.4,.3,-.3,-.4,-.4> & <.5,.2,.5,-.5,-.4,-.2> \\ <.5\ .5,.5,-.5,-.5,-.6> & <.4,.4,.4,-.3,-.4,-.4> & <.5,.2,.5,-.5,-.4,-.2> & <.5,.4,.6,-.6,-.2,-.3> \\ <.4,.4,.3,-.3,-.4,-.1> & <.5,.2,.5,-.5,-.4\ -.2> & <.5,.4,.6,-.6,-.2,-.3> & <.6,.6,.6,-.5,-.5,-.5> \end{pmatrix}$$

The result of composition t matrix of two bipolar neutrosophic matrices C and D can be obtained by the call of the command Composition (C, D):

>>> Composition(C, D)

Composition=

[[[0.5 0.4 0.3 -0.5 -0.4 -0.5] [0.5 0.5 0.5 -0.6 -0.2 -0.4] [0.5 0.5 0.5 -0.5 -0.5 -0.6] [0.4 0.4 0.3 -0.3 -0.4 -0.4]]

[[0.5 0.5 0.5 -0.6 -0.2 -0.4] [0.5 0.5 0.5 -0.5 -0.5 -0.6] [0.4 0.4 0.3 -0.3 -0.4 -0.4] [0.5 0.2 0.5 -0.5 -0.4 -0.2]]

[[0.5 0.5 0.5 -0.5 -0.5 -0.6] [0.4 0.4 0.3 -0.3 -0.4 -0.4] [0.5 0.2 0.5 -0.5 -0.4 -0.2] [0.5 0.4 0.6 -0.6 -0.2 -0.3]]

[[0.4 0.4 0.3 -0.3 -0.4 -0.4] [0.5 0.2 0.5 -0.5 -0.4 -0.2] [0.5 0.4 0.6 -0.6 -0.2 -0.3] [0.6 0.6 0.6 -0.5 -0.5 -0.5]]]

4. Conclusion

In this paper, we have presented a useful Python tool for the calculations of matrices obtained by bipolar neutrosophic sets. The matrices have nested list data type, in other words, multi-dimensional arrays in the Python Programming Language. The importance of this work, is that the proposed Python tool can be used also for fuzzy matrices, bipolar fuzzy matrices, intuitionistic fuzzy matrices, bipolar intuitionistic fuzzy matrices and single valued neutrosophic matrices. This work will be extending with the implementation of Bipolar Complex Neutrosophic Matrices in the future. We have used Python Numpy module in order to provide convenience for possible users. We hope that the tool might be useful in data science, physics, scientific computing, decision making, engineering studies and other fields.

Author Contributions

S.T. implemented codes of the bipolar neutrosophic matrices and their operations and created the scripts on Python 3.7 by using Numpy module. S.B. offered the project paper and reviewed the implementations. Conceptualization, S.B. and S.T.; Methodology, S.T.; Validation, S.B., S.T., A.B., M.T and F.S.; Investigation, S.B. and S.T.; Resources, S.B., S.T., A.B., M.T and F.S; Writing-Original Draft

Preparation, S.B..; Writing—Review and Editing, S.B., S.T., A.B., M.T and F.S.; Supervision, S.B. and F.S.

Acknowledgment

The authors are very grateful to the chief editor and reviewers for their comments and suggestions, which is helpful in improving the paper.

REFERENCES

1. Smarandache, F., Neutrosophy. Neutrosophic Probability, Set, and Logic. ProQuest Information & Learning, Ann Arbor, Michigan, USA, 1998,105 p.
2. Wang, H.; Smarandache, F.; Zhang, Y. and Sunderraman, R. Single Valued Neutrosophic Sets. Multispace and Multistructure 4,2010, pp. 410-413.
3. Zhang H.Y.; Wang J.Q., Chen X.H., Interval neutrosophic sets and their application in multicriteria decision making problems. The Scientific World Journal 2014 doi:10.1155/2014/645953.
4. Deli, I.; Ali, M.; Smarandache, F. Bipolar neutrosophic sets and their applications based on multicriteria decision making problems. Advanced Mechatronic Systems, (ICAMechs), International Conference, 2015,249-254. doi: 10.1109/ICAMechS.2015.7287068
5. Deli, İ.; Şubaş, Y. Bipolar neutrosophic refined sets and their applications in medical diagnosis. International Conference on Natural Science and Engineering (ICNASE'16), 2016, 1121-1132. http://fs.gallup.unm.edu/NSS/.
6. Mumtaz, A.; Le Hoang Son, Deli, I.; and Nguyen Dang Tien. Bipolar neutrosophic soft sets and applications in decision making. Journal of Intelligent &Fuzzy Systems 33,2017, 4077–4087, DOI:10.3233/JIFS-17999
7. Broumi, S.; Smarandache F.; Talea M.; and Bakali A. An introduction to bipolar single valued neutrosophic graph theory. Applied Mechanics and Materials 841,2016, 184–191.
8. Broumi, S.; Bakali, A.; Talea, M.; Smarandache, F.; and Ali, M. Shortest path problem under bipolar neutrosophic setting. Applied Mechanics and Materials 859 (2016), 59–66.
9. Dey, P.P.; Pramanik, S.; Giri, BC. TOPSIS for solving multi-attribute decision making problems under bipolar neutrosophic environment. In: New trends in neutrosophic theory and applications; Smarandache F, Pramanik; Publishing House, Pons asbl, Brussels, 2016, pp 55–63
10. Akram, M.; Shumaiza; Smarandache, F. Decision-Making with Bipolar Neutrosophic TOPSIS and Bipolar Neutrosophic ELECTRE-I. Axioms 2018, 7, 33.
11. Akram, M.; Sarwar M. Novel multiple criteria decision making methods based on bipolar neutrosophic sets and bipolar neutrosophic graphs, Italian Journal of Pure and Applied Mathematics, 38(2017), 368-389.
12. Akram, M. and Sitara, M. Bipolar neutrosophic graph structures. J. Indones. Math. Soc. Vol. 23, No. 1 (2017), pp. 55–80
13. Singh P.K, Three-way bipolar neutrosophic concepts and its graphical visualization,
14. Mullai, M.; Broumi S.; Stephen; A. Shortest path problem by minimal spanning tree algorithm using bipolar neutrosophic numbers. International Journal of Mathematic Trends and Technology, Vol 46, N2, 2017. pp.80-87
15. Uluçay, V.; Deli, I.; Şahin, M. Similarity measures of bipolar neutrosophic sets and their application to

multiple criteria decision making. Neural Computing and Applications,2016, https://doi.org/10.1007/s00521-016-2479-1

16. Dhar, M.; Broumi, S.; Smarandache, F. A Note on Square Neutrosophic Fuzzy Matrices. Neutrosophic Sets and Systems, Vol. 3, 2014, pp.37-41
17. Arockiarani, I.; Sumathi, I. R. A Fuzzy NEUTROSOPHIC SOFT MATRIX APPROACH IN DECISION MAKING. Journal of Global Research in Mathematical Archives, Volume 2, No. 2, 2014, pp.14-2
18. Karaaslan, F.; Hayat, K. Some new operations on single-valued neutrosophic matrices and their applications in multi-criteria group decision making. Applied Intelligence, 2018. https://doi.org/10.1007/s10489-018-1226-y
19. R. Uma, Murugadas P.; Sriram S. Fuzzy Neutrosophic Soft Matrices of Type-I and Type-II,
20. Uma, R.; Murugadas, P.; and Sriram, S. Section of Fuzzy Neutrosophic Soft Matrix. International Journal of Pure and Applied Mathematics, Volume 118 No. 23, 2018, 79-87
21. Hassan, A.; Malik, M. A.; Broumi, S.; Bakali, A.; Talea, M.; Smarandache, F. Special types of bipolar single valued neutrosophic graphs. Annals of Fuzzy Mathematics and Informatics Volume 14, No. 1, 2017, pp. 55-73.
22. Akram, M; and Siddique, S. Certain Properties of Bipolar Neutrosophic Graphs. Southeast Asian Bulletin of Mathematics,2018, 42: 463–490
23. Pramanik, S.; Dey, P. P.; Smarandache, F. and Jun Ye, Cross entropy measures of bipolar and interval bipolar neutrosophic sets and their application for multi-attribute decision making,2018
24. Wang, L.; Zhang, H.; Wang, J. Frank Choquet Bonferroni operators of bipolar neutrosophic sets and their applications to multi-criteria decision-making problems. Int. J. of Fuzzy Systems. 2017, DOI: 10.1007/s40815-017-0373-3.
25. Akram, M.; Nasir, M. Certain Bipolar Neutrosophic Competition Graphs. Indones. Math. Soc. Vol. 24, No. 1 (2018), pp. 1-25.
26. Akram, M.; SHUM, K. P. Bipolar Neutrosophic Planar Graphs. Journal of Mathematical Research with Applications,2017, Vol. 37, No. 6, pp. 631–648
27. Hashim, R.M.; Gulistan, M.; Smarandache, F. Applications of neutrosophic bipolar fuzzy Sets in HOPE foundation for planning to Build a Children Hospital with Different Types of Similarity Measures, Symmetry 2018, 10, 331; doi:10.3390/sym10080331
28. Akram, M.; Luqman A. Bipolar neutrosophic hypergraphs with applications. Journal of Intelligent & Fuzzy Systems, vol. 33, no. 3, pp. 1699-1713, 2017
29. Das S.; Kumar S.; Kar S.; Pal T. Group decision making using neutrosophic soft matrix: An algorithmic approach. Journal of King Saud University – Computer and Information Sciences xxx (2017) xxx–xxx,
30. Broumi, S.; Talea, M.; Bakali, A.; Smarandache, F. On Bipolar Single Valued Neutrosophic Graphs. Journal of New Theory, N11, 2016, pp.84-102.
31. Broumi, S.; Bakali, A.; Talea, M.; Smarandache, F.; and Ali, M. Shortest Path Problem under Bipolar Neutrosophic Setting. Applied Mechanics and Materials, Vol. 859, 2016, pp 59-66.
32. Broumi, S.; Smarandache; F.; Talea, M.; and Bakali, A. An Introduction to Bipolar Single Valued Neutrosophic Graph Theory. Applied Mechanics and Materials, vol.841,2016, 184-191.

33. Broumi, S.; Bakali, A.; Talea, M.; Smarandache, F.; Verma; R. Computing Minimum Spanning Tree In Interval Valued Bipolar Neutrosophic Environment, International Journal of Modeling and Optimization, Vol. 7, No. 5, 2017, pp300-304.
34. Broumi, S.; Son, L.H.; Bakali, A.; Talea, M.; Smarandache, F. and Selvachandran, G. Computing Operational Matrices in Neutrosophic Environments: A Matlab Toolbox, Neutrosophic Sets and Systems, vol. 18, 2017, pp. 58-66. http://doi.org/10.5281/zenodo.1175160
35. Pramanik, S. and Mondal, K. Rough Bipolar Neutrosophic Set. Global Journal of Engineering Science and Research Management, 3(6): June, 2016, pp.71-81
36. Pramanik, S.; Dey, P. P.; Giri, B. C.; Smarandache, F. Bipolar Neutrosophic Projection based Models for Solving Multi-Attribute Decision Making Problems. Neutrosophic Sets & Systems, 15, 2017, 70-79.
37. Broumi, S.; Talea, M.; Bakali, A.; Smarandache, F.; Khan, M. A Bipolar Single Valued Neutrosophic Isolated Graphs: Revisited. International Journal of New Computer Architectures and their Applications (IJNCAA) 7(3): ,2017,89-94
38. Akram, M.; Ishfaq, N.; Smarandache, F.; Broumi, S. Application of Bipolar Neutrosophic sets to Incidence Graphs, Neutrosophic Sets and Systems, vol. 27, 2019, pp. 180-200. DOI: 10.5281/zenodo.3275595
39. Zahariev, Z. Software package and API in MATLAB for working with fuzzy algebras. In G. Venkov, R. Kovacheva, & V. Pasheva (Eds.), AIP Conference Proceedings, Vol. 1184, No. 1, 2009, pp. 341-348
40. Peeva, K.; Kyosev, Y. Solving problems in intuitionistic fuzzy relational calculus with fuzzy relational calculus toolbox. In Eight International Conference on IFSs, Varna (pp. 37-43).
41. Karunambigai, M. G.; O. K. Kalaivani, Software development in intuitionistic Fuzzy Relational Calculus. International Journal of Scientific and research Publication, 6(7), 2016, pp.311-331.
42. Uma, R.; Murugadas, P.; Sriram, S. Determinant Theory for Fuzzy Neutrosophic Soft Matrices. Progress in Nonlinear Dynamics and Chaos, Vol. 4, No. 2, 2016, 85-102
43. Uma, R.; Murugadas; P.; Sriram, S. The Determinant and Adjoint of Fuzzy Neutrosophic Soft Matrices. International Journal of Mathematics and its Applications, Volume 5, Issue 4{F ,2017, 821-833.
44. El-Ghareeb, H. A. Novel Open Source Python Neutrosophic Package. Neutrosophic Sets and Systems, vol. 25, 2019, pp. 136-160. DOI: 10.5281/zenodo.2631514
45. Abdel-Basset, M.; Mohamed, R.; Zaied, A. E. N. H.; Smarandache, F. A Hybrid Plithogenic Decision-Making Approach with Quality Function Deployment for Selecting Supply Chain Sustainability Metrics. Symmetry, 2019, 11(7), 903.
46. Abdel-Basset, M.; Nabeeh, N. A.; El-Ghareeb, H. A.; Aboelfetouh, A. Utilising neutrosophic theory to solve transition difficulties of IoT-based enterprises. Enterprise Information Systems, 2019, 1-21.
47. Nabeeh, N. A.; Abdel-Basset, M.; El-Ghareeb, H. A.; Aboelfetouh, A. Neutrosophic multi-criteria decision making approach for iot-based enterprises. IEEE Access, 2019, 7, 59559-59574.
48. Abdel-Baset, M.; Chang, V.; Gamal, A. Evaluation of the green supply chain management practices: A novel neutrosophic approach. Computers in Industry, 2019, 108, 210-220.
49. Abdel-Basset, M.; Saleh, M.; Gamal, A.; Smarandache, F. An approach of TOPSIS technique for developing supplier selection with group decision making under type-2 neutrosophic number. Applied Soft Computing, 2019, 77, 438-452.
50. Abdel-Baset, M.; Chang, V.; Gamal, A.; Smarandache, F. (2019). An integrated neutrosophic ANP and

VIKOR method for achieving sustainable supplier selection: A case study in importing field. Computers in Industry, 106, 94-110.

51. Abdel-Basset, M.; Manogaran, G.; Gamal, A.; Smarandache, F. A group decision making framework based on neutrosophic TOPSIS approach for smart medical device selection. Journal of medical systems, 2019, 43(2), 38.

52. Broumi S.; Bakali A; Talea M.; Smarandache F.; Singh P.K.; Uluçay V.; Khan M. Bipolar Complex Neutrosophic Sets and Its Application in Decision Making Problem. In: Kahraman C., Otay İ. (eds) Fuzzy Multi-Criteria Decision-Making Using Neutrosophic Sets. Studies in Fuzziness and Soft Computing, 2019, vol 369. Springer, Cham.

Received: 10 April, 2019; Accepted: 24 August, 2019