

Soft Computing: Fuzzy and Crisp Composition, and Fuzzy Equivalence

Dhruba Saha

Roll no: B.Sc(Sem-VI)Comp-05

B.Sc(Honors) 3rd year 6th sem

Department of Computer & System Sciences

Siksha Bhavana

Visva Bharati

Dependencies

The following dependencies are required for the implementation of the assignment:

- Python 3.x
- NumPy library

Github Repository

<https://github.com/dhrubasaha08/DSE-4-Soft-Computing-Lab>

1 Fuzzy composition using max-min

```
1 import numpy as np
2
3 def fuzzy_max_min_composition(A, B):
4     result = np.zeros((A.shape[0], B.shape[1]))
5     for i in range(A.shape[0]):
6         for j in range(B.shape[1]):
7             min_values = np.minimum(A[i], B[:, j])
8             max_value = np.max(min_values)
9             result[i, j] = max_value
10    return result
11
12 def input_matrix(n, m):
13     mat = []
14     for i in range(n):
15         row = list(map(float, input(f"Enter row {i+1}: ").
16                               .split()))
17         assert len(row) == m
18         mat.append(row)
19     return np.array(mat)
20
21 if __name__ == "__main__":
22     n, m = map(int, input("Enter dimensions of matrix A
23     (n, m): ").split())
24     A = input_matrix(n, m)
25     m, p = map(int, input("Enter dimensions of matrix B
26     (m, p): ").split())
27     B = input_matrix(m, p)
28
29     result = fuzzy_max_min_composition(A, B)
30     print("Matrix A:\n", A)
31     print("Matrix B:\n", B)
32     print("Fuzzy max-min composition:\n", result)
```

Listing 1: Fuzzy max-min composition

```
1 Enter dimensions of matrix A (n, m): 2 2
2 Enter row 1: 0.4 0.6
3 Enter row 2: 0.7 0.3
4 Enter dimensions of matrix B (m, p): 2 2
5 Enter row 1: 0.6 0.9
6 Enter row 2: 0.5 0.6
7 Matrix A:
8 [[0.4 0.6]
9  [0.7 0.3]]
10 Matrix B:
11 [[0.6 0.9]
12  [0.5 0.6]]
13 Fuzzy max-min composition:
14 [[0.5 0.6]
15  [0.6 0.7]]
```

Listing 2: Output of Fuzzy max-min composition

2 Fuzzy composition using max-product

```
1 import numpy as np
2
3 def fuzzy_max_product_composition(A, B):
4     result = np.zeros((A.shape[0], B.shape[1]))
5     for i in range(A.shape[0]):
6         for j in range(B.shape[1]):
7             product_values = np.multiply(A[i], B[:, j])
8             max_value = np.max(product_values)
9             result[i, j] = max_value
10    return result
11
12 def input_matrix(n, m):
13     mat = []
14     for i in range(n):
15         row = list(map(float, input(f"Enter row {i+1}: ").
16                               .split()))
17         assert len(row) == m
18         mat.append(row)
19     return np.array(mat)
20
21 if __name__ == "__main__":
22     n, m = map(int, input("Enter dimensions of matrix A
23     (n, m): ").split())
24     A = input_matrix(n, m)
25     m, p = map(int, input("Enter dimensions of matrix B
26     (m, p): ").split())
27     B = input_matrix(m, p)
28
29     result = fuzzy_max_product_composition(A, B)
30     print("Matrix A:\n", A)
31     print("Matrix B:\n", B)
32     print("Fuzzy max-product composition:\n", result)
```

Listing 3: Fuzzy max-product composition

```
1 Enter dimensions of matrix A (n, m): 2 2
2 Enter row 1: 0.7 0.5
3 Enter row 2: 0.6 0.1
4 Enter dimensions of matrix B (m, p): 2 2
5 Enter row 1: 0.6 0.7
6 Enter row 2: 0.1 0.1
7 Matrix A:
8 [[0.7 0.5]
9  [0.6 0.1]]
10 Matrix B:
11 [[0.6 0.7]
12  [0.1 0.1]]
13 Fuzzy max-product composition:
14 [[0.42 0.49]
15  [0.36 0.42]]
```

Listing 4: Output of Fuzzy max-product composition

3 Crisp composition using max-min

```
1 import numpy as np
2
3 def crisp_max_min_composition(A, B):
4     result = np.zeros((A.shape[0], B.shape[1]), dtype=
5         int)
6     for i in range(A.shape[0]):
7         for j in range(B.shape[1]):
8             min_values = np.minimum(A[i], B[:, j])
9             max_value = np.max(min_values)
10            result[i, j] = max_value
11
12    return result
13
14 def input_matrix(n, m):
15     mat = []
16     for i in range(n):
17         row = list(map(int, input(f"Enter row {i+1}: ").
18             split()))
19         assert len(row) == m
20         mat.append(row)
21     return np.array(mat)
22
23 if __name__ == "__main__":
24     n, m = map(int, input("Enter dimensions of matrix A
25     (n, m): ").split())
26     A = input_matrix(n, m)
27     m, p = map(int, input("Enter dimensions of matrix B
28     (m, p): ").split())
29     B = input_matrix(m, p)
30
31     result = crisp_max_min_composition(A, B)
32     print("Matrix A:\n", A)
33     print("Matrix B:\n", B)
34     print("Crisp max-min composition:\n", result)
```

Listing 5: Crisp max-min composition

```
1 Enter dimensions of matrix A (n, m): 2 2
2 Enter row 1: 1 0
3 Enter row 2: 1 1
4 Enter dimensions of matrix B (m, p): 2 2
5 Enter row 1: 1 0
6 Enter row 2: 0 1
7 Matrix A:
8 [[1 0]
9  [1 1]]
10 Matrix B:
11 [[1 0]
12  [0 1]]
13 Crisp max-min composition:
14 [[1 0]
15  [1 1]]
```

Listing 6: Output of Crisp max-min composition

4 Crisp composition using max-product

```
1 import numpy as np
2
3 def crisp_max_product_composition(A, B):
4     result = np.zeros((A.shape[0], B.shape[1]), dtype=
5         int)
6     for i in range(A.shape[0]):
7         for j in range(B.shape[1]):
8             product_values = np.multiply(A[i], B[:, j])
9             max_value = np.max(product_values)
10            result[i, j] = max_value
11    return result
12
13 def input_matrix(n, m):
14     mat = []
15     for i in range(n):
16         row = list(map(int, input(f"Enter row {i+1}: ").
17             split()))
18         assert len(row) == m
19         mat.append(row)
20     return np.array(mat)
21
22 if __name__ == "__main__":
23     n, m = map(int, input("Enter dimensions of matrix A
24     (n, m): ").split())
25     A = input_matrix(n, m)
26     m, p = map(int, input("Enter dimensions of matrix B
27     (m, p): ").split())
28     B = input_matrix(m, p)
29
30     result = crisp_max_product_composition(A, B)
31     print("Matrix A:\n", A)
32     print("Matrix B:\n", B)
33     print("Crisp max-product composition:\n", result)
```

Listing 7: Crisp max-product composition

```
1 Enter dimensions of matrix A (n, m): 2 2
2 Enter row 1: 1 0
3 Enter row 2: 0 0
4 Enter dimensions of matrix B (m, p): 2 2
5 Enter row 1: 1 0
6 Enter row 2: 1 0
7 Matrix A:
8 [[1 0]
9  [0 0]]
10 Matrix B:
11 [[1 0]
12  [1 0]]
13 Crisp max-product composition:
14 [[1 0]
15  [0 0]]
```

Listing 8: Output of Crisp max-product composition

5 Check whether a Fuzzy relation satisfies the equivalence property or not

```
1 import numpy as np
2
3 def fuzzy_max_min_composition(A, B):
4     result = np.zeros((A.shape[0], B.shape[1]))
5     for i in range(A.shape[0]):
6         for j in range(B.shape[1]):
7             min_values = np.minimum(A[i], B[:, j])
8             max_value = np.max(min_values)
9             result[i, j] = max_value
10    return result
11
12 def check_equivalence_property(R):
13     R_squared = fuzzy_max_min_composition(R, R)
14     return np.allclose(R, R_squared, rtol=1e-05, atol=1e-08)
15
16 def input_matrix(n, m):
17     mat = []
18     for i in range(n):
19         row = list(map(float, input(f"Enter row {i+1}: ").split()))
20         assert len(row) == m
21         mat.append(row)
22     return np.array(mat)
23
24 if __name__ == "__main__":
25     n, m = map(int, input("Enter dimensions of matrix R (n, m): ").split())
26     assert n == m, "Matrix R must be a square matrix."
27     R = input_matrix(n, m)
28
29     equivalence = check_equivalence_property(R)
30     print("Matrix R:\n", R)
31     if equivalence:
32         print("The Fuzzy relation satisfies the equivalence property.")
```

```

33     else:
34         print("The Fuzzy relation does not satisfy the
equivalence property.")

```

Listing 9: Fuzzy equivalence

```

1  Enter dimensions of matrix R (n, m): 2 2
2  Enter row 1: 0.5 0.5
3  Enter row 2: 0.5 0.5
4  Matrix R:
5  [[0.5 0.5]
6  [0.5 0.5]]
7  The Fuzzy relation satisfies the equivalence property.
8
9
10 Enter dimensions of matrix R (n, m): 2 2
11 Enter row 1: 0.6 0.5
12 Enter row 2: 0.8 0.1
13 Matrix R:
14 [[0.6 0.5]
15 [0.8 0.1]]
16 The Fuzzy relation does not satisfy the equivalence
property.

```

Listing 10: Output of Check Fuzzy relation equivalence