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

```python
import numpy as np

def fuzzy_max_min_composition(A, B):
    result = np.zeros((A.shape[0], B.shape[1]))
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            min_values = np.minimum(A[i], B[:, j])
            max_value = np.max(min_values)
            result[i, j] = max_value
    return result

def input_matrix(n, m):
    mat = []
    for i in range(n):
        row = list(map(float, input(f"Enter row {i+1}: "
   ).split()))
        assert len(row) == m
        mat.append(row)
    return np.array(mat)

if __name__ == "__main__":
    n, m = map(int, input("Enter dimensions of matrix A
   (n, m): ").split())
    A = input_matrix(n, m)
    m, p = map(int, input("Enter dimensions of matrix B
   (m, p): ").split())
    B = input_matrix(m, p)

    result = fuzzy_max_min_composition(A, B)
    print("Matrix A:\n", A)
    print("Matrix B:\n", B)
    print("Fuzzy max-min composition:\n", result)
```

Listing 1: Fuzzy max-min composition

```
Enter dimensions of matrix A (n, m): 2 2
Enter row 1: 0.4 0.6
Enter row 2: 0.7 0.3
Enter dimensions of matrix B (m, p): 2 2
Enter row 1: 0.6 0.9
Enter row 2: 0.5 0.6
Matrix A:
 [[0.4 0.6]
 [0.7 0.3]]
Matrix B:
 [[0.6 0.9]
 [0.5 0.6]]
Fuzzy max-min composition:
 [[0.5 0.6]
 [0.6 0.7]]
```

Listing 2: Output of Fuzzy max-min composition

2 Fuzzy composition using max-product

```
import numpy as np

def fuzzy_max_product_composition(A, B):
    result = np.zeros((A.shape[0], B.shape[1]))
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            product_values = np.multiply(A[i], B[:, j])
            max_value = np.max(product_values)
            result[i, j] = max_value
    return result

def input_matrix(n, m):
    mat = []
    for i in range(n):
        row = list(map(float, input(f"Enter row {i+1}: ").split()))
        assert len(row) == m
        mat.append(row)
    return np.array(mat)

if __name__ == "__main__":
    n, m = map(int, input("Enter dimensions of matrix A (n, m): ").split())
    A = input_matrix(n, m)
    m, p = map(int, input("Enter dimensions of matrix B (m, p): ").split())
    B = input_matrix(m, p)

    result = fuzzy_max_product_composition(A, B)
    print("Matrix A:\n", A)
    print("Matrix B:\n", B)
    print("Fuzzy max-product composition:\n", result)
```

Listing 3: Fuzzy max-product composition

```
Enter dimensions of matrix A (n, m): 2 2
Enter row 1: 0.7 0.5
Enter row 2: 0.6 0.1
Enter dimensions of matrix B (m, p): 2 2
Enter row 1: 0.6 0.7
Enter row 2: 0.1 0.1
Matrix A:
 [[0.7 0.5]
 [0.6 0.1]]
Matrix B:
 [[0.6 0.7]
 [0.1 0.1]]
Fuzzy max-product composition:
 [[0.42 0.49]
 [0.36 0.42]]
```

Listing 4: Output of Fuzzy max-product composition

3 Crisp composition using max-min

```
import numpy as np

def crisp_max_min_composition(A, B):
    result = np.zeros((A.shape[0], B.shape[1]), dtype=int)
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            min_values = np.minimum(A[i], B[:, j])
            max_value = np.max(min_values)
            result[i, j] = max_value
    return result

def input_matrix(n, m):
    mat = []
    for i in range(n):
        row = list(map(int, input(f"Enter row {i+1}: ").split()))
        assert len(row) == m
        mat.append(row)
    return np.array(mat)

if __name__ == "__main__":
    n, m = map(int, input("Enter dimensions of matrix A (n, m): ").split())
    A = input_matrix(n, m)
    m, p = map(int, input("Enter dimensions of matrix B (m, p): ").split())
    B = input_matrix(m, p)

    result = crisp_max_min_composition(A, B)
    print("Matrix A:\n", A)
    print("Matrix B:\n", B)
    print("Crisp max-min composition:\n", result)
```

Listing 5: Crisp max-min composition

```
Enter dimensions of matrix A (n, m): 2 2
Enter row 1: 1 0
Enter row 2: 1 1
Enter dimensions of matrix B (m, p): 2 2
Enter row 1: 1 0
Enter row 2: 0 1
Matrix A:
 [[1 0]
 [1 1]]
Matrix B:
 [[1 0]
 [0 1]]
Crisp max-min composition:
 [[1 0]
 [1 1]]
```

Listing 6: Output of Crisp max-min composition

4 Crisp composition using max-product

```
import numpy as np

def crisp_max_product_composition(A, B):
    result = np.zeros((A.shape[0], B.shape[1]), dtype=int)
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            product_values = np.multiply(A[i], B[:, j])
            max_value = np.max(product_values)
            result[i, j] = max_value
    return result

def input_matrix(n, m):
    mat = []
    for i in range(n):
        row = list(map(int, input(f"Enter row {i+1}: ").split()))
        assert len(row) == m
        mat.append(row)
    return np.array(mat)

if __name__ == "__main__":
    n, m = map(int, input("Enter dimensions of matrix A (n, m): ").split())
    A = input_matrix(n, m)
    m, p = map(int, input("Enter dimensions of matrix B (m, p): ").split())
    B = input_matrix(m, p)

    result = crisp_max_product_composition(A, B)
    print("Matrix A:\n", A)
    print("Matrix B:\n", B)
    print("Crisp max-product composition:\n", result)
```

Listing 7: Crisp max-product composition

```
Enter dimensions of matrix A (n, m): 2 2
Enter row 1: 1 0
Enter row 2: 0 0
Enter dimensions of matrix B (m, p): 2 2
Enter row 1: 1 0
Enter row 2: 1 0
Matrix A:
 [[1 0]
 [0 0]]
Matrix B:
 [[1 0]
 [1 0]]
Crisp max-product composition:
 [[1 0]
 [0 0]]
```

Listing 8: Output of Crisp max-product composition

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

```python
import numpy as np

def fuzzy_max_min_composition(A, B):
    result = np.zeros((A.shape[0], B.shape[1]))
    for i in range(A.shape[0]):
        for j in range(B.shape[1]):
            min_values = np.minimum(A[i], B[:, j])
            max_value = np.max(min_values)
            result[i, j] = max_value
    return result

def check_equivalence_property(R):
    R_squared = fuzzy_max_min_composition(R, R)
    return np.allclose(R, R_squared, rtol=1e-05, atol=1e-08)

def input_matrix(n, m):
    mat = []
    for i in range(n):
        row = list(map(float, input(f"Enter row {i+1}: ").split()))
        assert len(row) == m
        mat.append(row)
    return np.array(mat)

if __name__ == "__main__":
    n, m = map(int, input("Enter dimensions of matrix R (n, m): ").split())
    assert n == m, "Matrix R must be a square matrix."
    R = input_matrix(n, m)

    equivalence = check_equivalence_property(R)
    print("Matrix R:\n", R)
    if equivalence:
        print("The Fuzzy relation satisfies the equivalence property.")
```

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

Listing 9: Fuzzy equivalence

```
Enter dimensions of matrix R (n, m): 2 2
Enter row 1: 0.5 0.5
Enter row 2: 0.5 0.5
Matrix R:
 [[0.5 0.5]
 [0.5 0.5]]
The Fuzzy relation satisfies the equivalence property.


Enter dimensions of matrix R (n, m): 2 2
Enter row 1: 0.6 0.5
Enter row 2: 0.8 0.1
Matrix R:
 [[0.6 0.5]
 [0.8 0.1]]
The Fuzzy relation does not satisfy the equivalence property.
```

Listing 10: Output of Check Fuzzy relation equivalence