Converting Team Dynamics Measure JS code to JAVA

Files:-

- ☑ ***DiscreteRecurrence***

Input:-

Js:-

```
let sequence = [[1,1,1,1,0,0,1]];
console.log(DiscreteRecurrence(sequence));
```

Java:-

```
int[][] sequence = {{1, 1, 1, 1, 0, 0, 1}};
double[] result = DiscreteRecurrence(sequence);
```

Output:-

Js:—

```
satyamshekhar@satyams-mbp windowed_recurrence_forward_window %
        node DiscreteRecurrence.js
[ 3.1429, 45.4545 ]
```

Java:-

```
satyamshekhar@satyams-mbp java % java DiscreteRecurrence
RR: 3.1429
DET: 45.4545
```

- ☑ ***Entropy***

Input:-

Js:-

```
var X = [[1,1],[1,2],[1,1],[1,2],[1,1],[1,2],[1,1],[1,2],[1,1],[1,1]]
console.log(Entropy(X));
```

Java:-

```
int[][] X = {{1, 1},{1, 2},{1, 1},{1, 2},{1, 1},{1, 2},{1, 1},{1, 2},{1, 1},{1, 1}};
double[] result = computeEntropy(X);
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp windowed_entropy_window % node
        Entropy.js
[ 0, 0.971 ]
```

Java:-

```
satyamshekhar@satyams-mbp java % java Entropy
Entropy: 0.0,0.971
```

- ☑ ***LayeredDynamicsRelaxationTimes***

Input:-

Js :-

```
console.log(LayeredDynamicsRelaxationTimes(0,5,[1,2,3,3,2,1,2,1,1]));
```

Java:-

```
int[] series = {1,2,3,3,2,1,2,1,1};
int[] result = LayeredDynamicsRelaxationTimes(0, 5, series);
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp layered-dynamics % node
        LayeredDynamicRelaxationTimes.js
[ 2, 0, 3 ]
```

Java:-

```
satyamshekhar@satyams-mbp java % java LayeredDynamicRelaxationTimes
[2, 0, 3]
```

- ☑ ***LayeredDynamicsRMSEFunction***

Input:-

Js:-
```
console.log(LayeredDynamicsRMSEFunction([1,2,3,4,5,6,3,4,5],2,2,2));
```

Java:-
```
int[] var1 = new int[]{1, 2, 3, 4, 5, 6, 3, 4, 5};
double[] var2 = LayeredDynamicsRMSEFunction(var1, 2, 2, 2);
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp LayeredDynamicsRMSE % node
        LayeredDynamicsRMSEFunction.js
[
  1.5,   1, 0.5, 0.4,
  1.4, 1.6, 0.6, 0.4
]
```

Java:-

```
satyamshekhar@satyams-mbp java % java LayeredDynamicsRMSEFunction
[1.5, 1.0, 0.5, 0.4, 1.4, 1.6, 0.6, 0.4]
```

- ☑ ***ami***

Input:-

Js:-
```
console.log(ami([[11,69],[54,74],[98,8],[19,18],[29,90]],[[5,2]],3));
```

Java:-
```
int[] var1 = new int[]{1, 2, 3, 4, 5, 6, 3, 4, 5};
double[] var2 = LayeredDynamicsRMSEFunction(var1, 2, 2, 2);
```

Output:-

Js:-

satyamshekhar@satyams-mbp ami % node ami.js
[
  [[0.42], [1], [0.9181], [1]],
  [[-0.4763], [-0.8128], [0.9191], [1]]
]

Java:-

satyamshekhar@satyams-mbp java % java ami
AMI:
[0.42]
[1.0]
[0.9181]
[1.0]

Corr:
[-0.4763]
[-0.8128]
[0.9191]
[1.0]

- ☑ ***createBitfieldVector***

My xlsx contains:-

Channel1	Channel2	Channel3	Channel4	Channel5	Channel6	Channel7	Channel8	Channel9	Channel10	Channel11	Channel12
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

<u>Input:-</u>

Js:-

```
createBitfieldVector("BinaryMatrix5911_v2_columns.xlsx");
```

Java:-

```
String csvFile = "BinaryMatrix5911_v2_columns(Columns).csv";
List<Object> result = createBitfieldVector(csvFile);
```

<u>Output:-</u>

Js:-

```
satyamshekhar@satyams-mbp speakerHistogramsTDMS_noInt % node
        createBitfieldVector.js
[
          0,          0,          0,
          0,          0, 1001001001,
  1001001001, 1001001001, 1001001001,
          0,          0,          0,
          0,          0,          0,
          0,          0,          0,
          0
]
[
  '',
  '',
  '',
  '',
  '',
  'Channel1 + Channel4 + Channel7 + Channel10',
```

```
  'Channel1 + Channel4 + Channel7 + Channel10',
  'Channel1 + Channel4 + Channel7 + Channel10',
  'Channel1 + Channel4 + Channel7 + Channel10',
  '',
  '',
  '',
  '',
  '',
  '',
  '',
  '',
  '',
  '',
  '',
  ''
]
```

Java:-

```
satyamshekhar@satyams-mbp java % java CreateBitfieldVector
Bitfields:
[0, 0, 0, 0, 0, 1001001001, 1001001001, 1001001001, 1001001001, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0]

Labels:
[, , , , , Channel1 + Channel4 + Channel7 + Channel10, Channel1 + Channel4
        + Channel7 + Channel10, Channel1 + Channel4 + Channel7 +
        Channel10, Channel1 + Channel4 + Channel7 + Channel10, , , , , , , , , ,
        ]
```

- ☑ ***prob***

Input:-

Js:-

```
console.log(prob([[1,0,0,1,1,0,0,1,1,1]],3));
```

Java:-

```
double[][] y = {{1, 0, 0, 1, 1, 0, 0, 1, 1, 1}};
Object[] result = prob(y, 3);
```

Output:-

Js:-

satyamshekhar@satyams-mbp ami % node prob.js
[[0.4, 0.6], 2]

Java:-
satyamshekhar@satyams-mbp java % java prob
nn: [0.4, 0.6]
nBins: 2

- ☑ ***probxy***

Input:-

Js:-
```
console.log(probxy([[1,2],[1,2],[1,2]],2,3));
console.log(probxy([[2,4],[3,5],[5,6]],[[1,2,3]],[[4,5,6]]));
```

Java:-
```
double[][] input1 = {{1, 2}, {1, 2}, {1, 2}};
double[][] input2 = {{2, 4}, {3, 5}, {5, 6}};
double[][] customBinsX = {{1, 2, 3}};
double[][] customBinsY = {{4, 5, 6}};

double[][] out1 = probxy(input1, 2, 3);
double[][] out2 = probxy(input2, customBinsX, customBinsY);
```

Output:-

Js:-

satyamshekhar@satyams-mbp ami % node probxy.js
[[0, 0, 0], [0, 0, 1]]
[[0, 0], [0.3333, 0]]

Java:-
satyamshekhar@satyams-mbp java % java probxy
Test 1:
[0.0, 0.0, 0.0]
[0.0, 0.0, 1.0]

Test 2:
[0.0, 0.0]

[0.3333, 0.0]

☑ ***rhist***

Input:-

Js:-

```
// Base case
testCase('Base case', [[1, 2, 3], [4, 5, 6]], 2);

// Single bin
testCase('Single bin', [[1, 2, 3, 4, 5, 6]], 1);

// More bins than unique values
testCase('More bins than values', [[1, 1, 2, 2, 3, 3]], 10);

// Duplicate values
testCase('All same values', [[5, 5], [5, 5]], 4);

// Negative values
testCase('Negative values', [[-3, -2], [0, 1, 2]], 3);

// Values on bin edges
testCase('Edge-aligned values', [[1, 2, 3, 4]], 3);

// Normalize by bin width
testCase('Density normalization', [[1, 2, 3], [4, 5, 6]], 2, 1);

// Empty input
testCase('Empty input', [[]]);

// Invalid format (not 2D)
testCase('Invalid input (1D)', [1, 2, 3]);
```

Java:-

```
// Base case
printResult("Base case", rhist(new double[][]{{1, 2, 3}, {4, 5, 6}}, 2));

// Single bin
```

```
printResult("Single bin", rhist(new double[][]{{1, 2, 3, 4, 5, 6}}, 1));

// More bins than values
printResult("More bins than values", rhist(new double[][]{{1, 1, 2, 2, 3, 3}},
        10));

// All same values
printResult("All same values", rhist(new double[][]{{5, 5}, {5, 5}}, 4));

// Negative values
printResult("Negative values", rhist(new double[][]{{-3, -2}, {0, 1, 2}}, 3));

// Edge-aligned values
printResult("Edge-aligned values", rhist(new double[][]{{1, 2, 3, 4}}, 3));

// Density normalization
printResult("Density normalization", rhist(new double[][]{{1, 2, 3}, {4, 5, 6}}, 2,
        1));

// Empty input
printResult("Empty input", rhist(new double[][]{{}}, 10));

// Invalid input (1D-like)
printResult("Invalid input (1D)", rhist(new double[][]{{1}, {2}, {3}}, 10));
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp ami % node rhist_test.js

=== Base case ===
nn: [ 0.5, 0.5 ]
centers: [ 2.25, 4.75 ]

=== Single bin ===
nn: [ 1 ]
centers: [ 3.5 ]

=== More bins than values ===
nn: [
```

```
  0.3333,      0, 0,
       0, 0.3333, 0,
       0,      0, 0,
  0.3333
]
centers: [
  1.1, 1.3, 1.5, 1.7,
  1.9, 2.1, 2.3, 2.5,
  2.7, 2.9
]

=== All same values ===
nn: [ 1, 0, 0, 0 ]
centers: [ 5, 5, 5, 5 ]

=== Negative values ===
nn: [ 0.5, 0, 0.5 ]
centers: [ -2.3333, -1, 0.3333 ]

=== Edge-aligned values ===
nn: [ 0.5, 0.25, 0.25 ]
centers: [ 1.5, 2.5, 3.5 ]

=== Density normalization ===
nn: [ 0.2, 0.2 ]
centers: [ 2.25, 4.75 ]

=== Empty input ===
nn: [
  NaN, NaN, NaN, NaN,
  NaN, NaN, NaN, NaN,
  NaN, NaN
]
centers: [
  NaN, NaN, NaN, NaN,
  NaN, NaN, NaN, NaN,
  NaN, NaN
]

=== Invalid input (1D) ===
nn: [
  NaN, NaN, NaN, NaN,
```

NaN, NaN, NaN, NaN,
NaN, NaN
]
centers: [
NaN, NaN, NaN, NaN,
NaN, NaN, NaN, NaN,
NaN, NaN
]

Java:-
satyams-mbp:java satyamshekhar$ java rhist

=== Base case ===
nn: [0.5, 0.5]
centers: [2.25, 4.75]

=== Single bin ===
nn: [1.0]
centers: [3.5]

=== More bins than values ===
nn: [0.3333, 0.0, 0.0, 0.0, 0.3333, 0.0, 0.0, 0.0, 0.0, 0.3333]
centers: [1.1, 1.3, 1.5, 1.7, 1.9, 2.1, 2.3, 2.5, 2.7, 2.9]

=== All same values ===
nn: [1.0, 0.0, 0.0, 0.0]
centers: [5.0, 5.0, 5.0, 5.0]

=== Negative values ===
nn: [0.4, 0.2, 0.4]
centers: [-2.1667, -0.5, 1.1667]

=== Edge-aligned values ===
nn: [0.5, 0.25, 0.25]
centers: [1.5, 2.5, 3.5]

=== Density normalization ===
nn: [0.2, 0.2]
centers: [2.25, 4.75]

=== Empty input ===
nn: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

centers: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

=== Invalid input (1D) ===
nn: [0.3333, 0.0, 0.0, 0.0, 0.3333, 0.0, 0.0, 0.0, 0.0, 0.3333]
centers: [1.1, 1.3, 1.5, 1.7, 1.9, 2.1, 2.3, 2.5, 2.7, 2.9]

Test cases that do not have same output

Since there was no test case provided in the original JS side code, I have tested on few of my own test cases. There are some differences in the output for a few input i-e:- for negative values and for invalid input(1D). This has nothing to do with logic but with how Both languages handle it.

Issue 1:Negative values output differs

JavaScript output:

nn: [0.5, 0, 0.5]
centers: [-2.3333, -1, 0.3333]

Java output:

nn: [0.4, 0.2, 0.4]
centers: [-2.1667, -0.5, 1.1667]

Root cause:

This is **not a bug** in logic — it's due to how Java handles floating-point **division and rounding** with:

double binWidth = (max - min) / x;
vs JS:
var binWidth = (max - min)/x;

But here's the catch:

- In JS, slice edges are float-rounded differently when using Math.min, Math.max and dynamic bin sizing.
- Java calculates bin edges and centers **more precisely**, and Math.round(... * 10000)/10000 introduces tiny numerical shifts (e.g., -2.1667 vs -2.3333)

So the values are correct, but not bitwise identical — just floating-point effects.

##Issue 2:Invalid input (1D)

JS output:

nn: [NaN, ..., NaN]
centers: [NaN, ..., NaN]

Java output:

nn: [0.3333, 0.0, ..., 0.3333]
centers: [1.1, 1.3, ..., 2.9]

Root cause:

Your JavaScript test used a 1D array like [1,2,3], which **triggers an error** during the flattening step.

In Java: new double{{1}, {2}, {3}} - is still treated as a valid 2D array.
So flattening works fine, and the histogram is built on values [1,2,3], producing valid bins:
[0.3333, 0.0, 0.0, 0.0, 0.3333, ..., 0.3333]

Therefore: this difference is expected and valid — it's not a logic error, just a platform behaviour difference.

- ☑ ***speakerHistogramsTDMS_noInt***

My xlsx contains:-

Channel1	Channel2	Channel3	Channel4	Channel5	Channel6	Channel7	Channel8	Channel9	Channel10	Channel11	Channel12
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

<u>Input:-</u>

Js:-

```
var T = readFile("BinaryMatrix5911_v2_columns.xlsx");
speakerHistogramsTDMS_noInt(T,1,T.length);
```

Java:-

```
String csvFile = "BinaryMatrix5911_v2_columns(Columns).csv";
List<Object> result = createBitfieldVector(csvFile);
```

<u>Output:-</u>

Js:-

```
satyamshekhar@satyams-mbp speakerHistogramsTDMS_noInt % node
        speakerHistogramsTDMS_noInt.js
[
  [ 'Channel1', 4 ],
  [ 'Channel4', 4 ],
  [ 'Channel7', 4 ],
  [ 'Channel10', 4 ],
```

```
  [ 'Channel2', 0 ],
  [ 'Channel3', 0 ],
  [ 'Channel5', 0 ],
  [ 'Channel6', 0 ],
  [ 'Channel8', 0 ],
  [ 'Channel9', 0 ],
  [ 'Channel11', 0 ],
  [ 'Channel12', 0 ]
]
```

Java:-

```
satyamshekhar@satyams-mbp java % java SpeakerHistogramsTDMSNoInt
[[Channel1, 4], [Channel4, 4], [Channel7, 4], [Channel10, 4], [Channel2, 0],
    [Channel3, 0], [Channel5, 0], [Channel6, 0], [Channel8, 0], [Channel9,
    0], [Channel11, 0], [Channel12, 0]]
```

☑ ***unique_speech_components_ami_noInt***

My xlsx contains:-

Channel1	Channel2	Channel3	Channel4	Channel5	Channel6	Channel7	Channel8	Channel9	Channel10	Channel11	Channel12
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
1	0	0	1	0	0	1	0	0	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

Input:-

Js:-

```
var T = readFile("BinaryMatrix5911_v2_columns.xlsx");
var [T1,T2] = speakerHistogramsTDMS_noInt(T,1,T.length);
unique_speech_components_ami_noInt(T1,10);
```

Java:-

```
List<Map<String, String>> T =
CreateBitfieldVector.readCsvFile(“BinaryMatrix5911_v2_columns(Columns).c
sv");

List<Object> hist =
SpeakerHistogramsTDMSNoInt.speakerHistogramsTDMS_noInt(T, 1,
T.size());

int[][] T1 = (int[][]) hist.get(0);

uniqueSpeechComponentsAmiNoInt(T1, 10);
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp speakerHistogramsTDMS_noInt % node
        unique_speech_components_ami_tdms.js
[
  [
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0
  ],
  [
    0, 0, 0, 0, 0,
    0, 0, 0, 0, 0,
    0, 0
  ],
```

```
[
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,
  0, 0
],
[
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,
  0, 0
],
[
  0, 0, 0, 0, 0,
  0, 0, 0, 0, 0,
  0, 0
],
[
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0
],
[
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0
],
[
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0
],
[
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0,
  2.8368, 0, 0
],
[
  2.8368, 0, 0,
  2.8368, 0, 0,
```

```
    2.8368, 0, 0,
    2.8368, 0, 0
  ],
  [
    2.8368, 0, 0,
    2.8368, 0, 0,
    2.8368, 0, 0,
    2.8368, 0, 0
  ],
  [
    2.5359, 0, 0,
    2.5359, 0, 0,
    2.5359, 0, 0,
    2.5359, 0, 0
  ],
  [
    2.052,     0,     0, 2.052,
        0,     0, 2.052,     0,
        0, 2.052,     0,     0
  ],
  [
    1.3185, 0, 0,
    1.3185, 0, 0,
    1.3185, 0, 0,
    1.3185, 0, 0
  ]
]
```

Java:-

```
satyamshekhar@satyams-mbp java % java
        UniqueSpeechComponentsAmiNoInt
[
  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000],
  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000],
  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000],
  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
        0.0000, 0.0000, 0.0000],
```

```
  [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000,
     0.0000, 0.0000, 0.0000],
  [2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000,
     2.8368, 0.0000, 0.0000],
  [2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000,
     2.8368, 0.0000, 0.0000],
  [2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000,
     2.8368, 0.0000, 0.0000],
  [2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000,
     2.8368, 0.0000, 0.0000],
  [2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000,
     2.8368, 0.0000, 0.0000],
  [2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000, 2.8368, 0.0000, 0.0000,
     2.8368, 0.0000, 0.0000],
  [2.5359, 0.0000, 0.0000, 2.5359, 0.0000, 0.0000, 2.5359, 0.0000, 0.0000,
     2.5359, 0.0000, 0.0000],
  [2.0520, 0.0000, 0.0000, 2.0520, 0.0000, 0.0000, 2.0520, 0.0000, 0.0000,
     2.0520, 0.0000, 0.0000],
  [1.3185, 0.0000, 0.0000, 1.3185, 0.0000, 0.0000, 1.3185, 0.0000, 0.0000,
     1.3185, 0.0000, 0.0000]
]
```

☑ ***windowed_entropy_window***

Input:-

Js:-

```
console.log(windowed_entropy_window([[1],[1],[0],[0],[0],[1],[1],[1],[1],[1],
     [1]],4));
```

Java:-

```
double[][] seq = {{1}, {1}, {0}, {0}, {0}, {1}, {1}, {1}, {1}, {1}, {1}};

double[][] result = windowed_entropy_window(seq, 4);
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp windowed_entropy_window % node
        windowed_entropy_window.js
[
  [ 0 ],      [ 0 ],
  [ 0 ],      [ 0 ],
  [ 1 ],      [ 0.8113 ],
  [ 0.8113 ], [ 1 ],
  [ 0.8113 ], [ 0 ],
  [ 0 ]
]
```

Java:-

```
satyamshekhar@satyams-mbp java % java windowed_entropy_window
Entropy Series:
[0.0]
[0.0]
[0.0]
[0.0]
[1.0]
[0.8113]
[0.8113]
[1.0]
[0.8113]
[0.0]
[0.0]
```

- ☑ ***windowed_recurrence_forward_window***

Input:-

Js:-

```
windowed_recurrence_forward_window([[1,1,2,3,4,5,6,7,8,9]],4);
```

Java:-

```
double[][] var1 = new double[][]{{1.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}};

byte var2 = 4;

double[][][] var3 = windowed_recurrence_forward_window(var1, var2);
```

Output:-

Js:-

```
satyamshekhar@satyams-mbp windowed_recurrence_forward_window %
        node windowed_recurrence_forward_window.js
[
  [ 100 ], [ 100 ],
  [ 100 ], [ 100 ],
  [ 0.5 ], [ 100 ],
  [ 100 ], [ 100 ],
  [ 100 ], [ 100 ]
]
[
  [ 100 ], [ 100 ],
  [ 100 ], [ 100 ],
  [ 100 ], [ NaN ],
  [ NaN ], [ NaN ],
  [ NaN ], [ NaN ]
]
```

Java:-

```
satyamshekhar@satyams-mbp java % java
        windowed_recurrence_forward_window
RR Series:
[100.0]
[100.0]
[100.0]
[100.0]
[0.5]
[100.0]
[100.0]
[100.0]
[100.0]
[100.0]

DET Series:
[100.0]
```

[100.0]
[100.0]
[100.0]
[100.0]
[NaN]
[NaN]
[NaN]
[NaN]
[NaN]

Overall:-

things to look at:-

- For the DiscreteRecurrence, and windowed_recurrence_forward_window, The NaN thing
- **LayeredDynamicRelaxationTimes — line no 20 (JS side code)**
  - // TODO: revisit logic – matches JS bug behavior — comment present in line number 28 in java side code. This comment is present on java side code, line no 28.

js side code :-
if(failureDuration[i] - Math.max(failureDuration == 0)){
Math.max(failureDuration == 0) will always be false

for now in **java side code**, i have replaced the code with something to mirror the js side code
if (failureDuration[i] - 0 != 0)

- In CreateBitfieldVector.java, in the original js code, It directly reads the data from xlsx, same is not possible to do in java. To achieve similar output, either apachePOI(using which we can directly read the data from xlsx) or converting the xlsx to csv and then read the data. I am using CSV (we can directly read data from csv in java).
- Also How Java Handles NaN and how JS does, bcoz Java sometimes instead of output NaN, it outputs 0 (so that us why you will see sometimes NaN on JS side output but 0 on Java side output).