```
%%% Q.9: Is it certain that there are no cyclic paths (except FPs' paths) across
all paths in the STG?

% Verify STG is acyclic (using (unweighted) graph Laplacian matrix)
function LaplacianSummaryTable = isAcyclic(source, target, startNodes) % Define
function to create information table about graph nature

% Create adjacency matrix (A)
A = zeros(length(startNodes)); % A with default elemets

for inSource = 1:length(source) % Iterate through all source nodes in graph

    A(source(inSource), target(inSource)) = 1; % Update element for presence of
A_ij

end

% Compute out-degree matrix (D)
D = diag(sum(A, 2)); % Create diagonal matrix with sum along each row of A
(out-degree of each node)

% Compute Laplacian matrix
L = D - A;

% Columns for L-derived information table
propertyL = {}; % Initialize cell to store probed properties of L

conditionCheckedL = {}; % Initialize cell to store conditions for cyclicity/
acyclicity

conditionFoundL = {}; % Initialize cell to store what's found about satisfying
corresponding conditions

inferenceL = {}; % Initialize cell to store inference if corresponding condition
is satisfied

% Matrix algebra to verify if STG is acyclic
for inConditions = 1:4 % Iterate through 4 check conditions

    if inConditions == 1 % Property 1 is nullity: if nullity(L) > 0, graph may
have cycles

        % nullity(L) = n – rank(L)
        numColumnsL = size(L, 2); % Store number of columns in L

        rankL = rank(L); % Compute rank of L

        nullityL = numColumnsL - rankL; % Compute nullity of L

        % Display result
        disp(['Rank of L: ', num2str(rankL)]);
        disp(['Nullity of L: ', num2str(nullityL)]);
```

```
        propertyL{inConditions} = 'Nullity'; % Add property to its column

        conditionCheckedL{inConditions} = 'nullity(L) > 0'; % Add sought
condition to its column

        conditionFoundL{inConditions} = ['nullity(L) = ', num2str(nullityL)]; %
Add computed condition to its column

        if nullityL > 0 % Condition indicating cycles may be present

            inferenceL{inConditions} = 'Cycles may be present'; % Add inference
to its column

        else % Condition for no cycles

            inferenceL{inConditions} = 'Graph may be acyclic'; % Add inference
to its column

        end

    elseif inConditions == 2 % Property 2 is rank: if r(L) < n - 1, graph may
have cycles

        % Display result
        disp(['Rank of L: ', num2str(rankL)]);
        disp(['Number of graph nodes: ', num2str(length(startNodes))]);

        propertyL{inConditions} = 'Rank'; % Add property to its column

        conditionCheckedL{inConditions} = 'r(L) < n - 1'; % Add sought condition
to its column

        conditionFoundL{inConditions} = ['r(L) = ', num2str(rankL)]; % Add
computed condition to its column

        if rankL < numColumnsL - 1 % Condition indicating potental presence of
cycles

            inferenceL{inConditions} = ['Cycles may be present; n = ',
num2str(length(startNodes))]; % Add inference to its column

        elseif rankL == numColumnsL - 1 % Condition indicating no cycles

            inferenceL{inConditions} = 'Graph may be acyclic; n = 64'; % Add
inference to its column

        end

    elseif inConditions == 3 % Property 3 is eigenvalues: if any eigenvalue is
0, graph may have cycles

        eigenvaluesL = eig(L); % Compute eigenvalues of L

        tolerance = 1e-10; % Tolerance for "zero"
```

```
        hasZeroEigenvalue = any(abs(eigenvaluesL) < tolerance); % Find logical
value for presence/absence of zero eigenvalue

        allPositiveReal = all(real(eigenvaluesL) > 0); % Check if all
eigenvalues have positive real parts

        % Display result
        disp('Eigenvalues of L:');
        disp(eigenvaluesL);
        disp(['count:', num2str(length(eigenvaluesL))]);

        propertyL{inConditions} = 'Eigenvalues'; % Add property to its column

        conditionCheckedL{inConditions} = 'any(eigenvalues) = 0'; % Add sought
condition to its column

        if hasZeroEigenvalue % Condition indicating cycles may be present

            conditionFoundL{inConditions} = '0 eigenvalue found'; % Add computed
condition to its column

            inferenceL{inConditions} = 'Cycles may be present'; % Add inference
to its column

        elseif allPositiveReal % Condition for no cycles

            inferenceL{inConditions} = 'Cycles are not present'; % Add inference
to its column

        end

    elseif inConditions == 4 % Property 4 is A^k: if trace(A^k) > 0, graph has
at least one cycle of k length

        % k ≤ n
        allTracesZero = true; % Initialize flag to check if all trace for all k
values is zero

        numStartNodes = length(startNodes); % Store number of graph nodes

        traceValues = zeros(1, numStartNodes); % Initialize row vector to track
trace for all k values

        for inStartNodes = 1:numStartNodes % Iterate through all graph nodes

            k = inStartNodes; % Assign k value

            Ak = A^k; % Adjacency matrix raised to power k

            tracek = trace(Ak); % Trace of A^k

            traceValues(k) = tracek; % Store current k tr(A^k) value for display
```

```
            if abs(tracek) > 0 % Check if current k tr(A^k) value is non-zero

                allTracesZero = false; % Update flag value

            end

        end

        nonZeroTracek = sum(find(traceValues > 0)); % Count number of k for
which trace values are not zero

        % Display results
        disp(['Traces of A^k for k = 1 to ', num2str(numStartNodes), ':']);
        disp(traceValues);

        propertyL{inConditions} = 'Power of adjacency matrix (k; k <= n)'; % Add
property to its column

        conditionCheckedL{inConditions} = 't(A^k) > 0'; % Add sought condition
to its column

        conditionFoundL{inConditions} = ['Are all traces zero? - ',
mat2str(allTracesZero)]; % Add computed condition to its column

        if nonZeroTracek > 0 % Condition indicating cycle(s) is present

            inferenceL{inConditions} = 'At least one cycle is present'; % Add
inference to its column

        elseif allTracesZero % Condition for no cycles

            inferenceL{inConditions} = 'Graph is acyclic'; % Add inference to
its column

        end

    end

end

% Transpose of all columns (row vectors) to make column vectors for table
propertyInTable = propertyL';
conditionCheckedInTable = conditionCheckedL';
conditionFoundInTable = conditionFoundL';
inferenceInTable = inferenceL';

% Create summary table
LaplacianSummaryTable = table(propertyInTable, conditionCheckedInTable,
conditionFoundInTable, inferenceInTable); % Create L information table

end

% Execute function to create summary table for Laplacian matrix-derived STG
cyclicity verification
```

```
LaplacianSummaryTable = isAcyclic(source, target, startNodes)
```

Output:

```
Rank of L: 61

Nullity of L: 3

Rank of L: 61

Number of graph nodes: 64

Eigenvalues of L:

10

3

14

15

15

12

7

2

3

1

8

4

19

15

19

16

9

5

5

13

13

8

3

1
```

2

14

6

10

11

8

10

8

11

15

7

3

2

9

3

11

10

4

1

5

3

1

2

1

3

1

0

3

1

5

5

3

```
3

3

1

1

3

1

0

0

count:64


Traces of A^k for k = 1 to 64:

3     3     3     3     3     3     3     3     3     3     3     3     3     3     3
3     3     3     3     3     3     3     3     3     3     3     3     3     3     3
3     3     3     3     3     3     3     3     3     3     3     3     3     3     3
3     3     3     3     3     3     3     3     3     3     3     3     3     3     3
3     3     3     3


LaplacianSummaryTable = 4×4 table
```

	propertyInTable	conditionCheckedInTable	conditionFoundInTable	inferenceInTable
1	'Nullity'	'nullity(L) > 0'	'nullity(L) = 3'	'Cycles may be present'
2	'Rank'	'r(L) < n - 1'	'r(L) = 61'	'Cycles may be present; n = 64'
3	'Eigenvalues'	'any(eigenvalues) = 0'	'0 eigenvalue found'	'Cycles may be present'
4	'Power of adjacency matrix (k; k <= n)'	't(A^k) > 0'	'Are all traces zero? - false'	'At least one cycle is present'