STP Toolbox for Matlab/Octave*

Hongsheng Qi, Daizhan Cheng

†Key Laboratory of Systems and Control, Chinese Academy of Sciences
‡Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Beijing 100190, P.R.China
E-mail: qihongsh@amss.ac.cn, dcheng@iss.ac.cn

Last Updated on August 26, 2016

1 Introduction

The semi-tensor product (STP) of matrices is a novel matrix product, which is a generalization of conventional matrix product for the case when the two factor matrices even do not meet the dimension matching condition [1, 2, 3, 4, 5].

The STP toolbox for Matlab[1] and GNU Octave[2] is developed for calculating the semi-tensor product (STP) of (logical) matrices and its application to the analysis and control of Boolean networks.

The semi-tensor product of matrices is defined as follows

Definition 1.1 *1. Let X be a row vector of dimension np, and Y be a column vector with dimension p. Then we split X into p equal-size blocks as $X^1, \cdots, X^p$, which are $1 \times n$ rows. The (left) STP, denoted by $\ltimes$, is defined as*

$$\begin{cases} X \ltimes Y = \sum\limits_{i=1}^{p} X^i y_i \in \mathbb{R}^n, \\ Y^{\mathrm{T}} \ltimes X^{\mathrm{T}} = \sum\limits_{i=1}^{p} y_i (X^i)^{\mathrm{T}} \in \mathbb{R}^n. \end{cases} \tag{1}$$

2. Let $A \in \mathcal{M}_{m\times n}$ and $B \in \mathcal{M}_{p\times q}$. If either n is a factor of p, say $nt = p$ and denote it as $A \prec_t B$, or p is a factor of n, say $n = pt$ and denote it as $A \succ_t B$, then the (left) STP of A and B , denoted by $C = A \ltimes B$, is defined as the following: C consists of $m \times q$ blocks as $C = (C^{ij})$ and each block is

$$C^{ij} = A^i \ltimes B_j, \quad i = 1, \cdots, m, \quad j = 1, \cdots, q,$$

where A^i is i-th row of A and B_j is the j-th column of B.

The above definition is for the two matrices satisfying multiple dimension condition, the following definition is a general case for two arbitrary matrices.

Definition 1.2 *Let $A \in \mathcal{M}_{m\times n}$ and $B \in \mathcal{M}_{p\times q}$. The (left) semi-tensor product of A and B is defined as*

$$A \ltimes B = (A \otimes I_{t/n})(B \otimes I_{t/p}), \tag{2}$$

where t is the least common multiple of n and p, and $\otimes$ is the Kronecker product.

---

*Supported partly by National Natural Science Fundation of China under Grants 60674022, 60736022, 60221301, and 61104065.

[1] http://www.mathworks.com

[2] http://www.octave.org

We use some simple numerical examples to describe it.

Example 1.3 1. Let $X = \begin{bmatrix} 1 & 2 & 3 & -1 \end{bmatrix}$ and $Y = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. Then

$$X \ltimes Y = \begin{bmatrix} 1 & 2 \end{bmatrix} \cdot 1 + \begin{bmatrix} 3 & -1 \end{bmatrix} \cdot 2 = \begin{bmatrix} 7 & 0 \end{bmatrix}.$$

2. Let

$$A = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 2 & 3 & 1 & 2 \\ 3 & 2 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1 & -2 \\ 2 & -1 \end{bmatrix}.$$

Then

$$A \ltimes B = \begin{bmatrix} \begin{pmatrix} 1 & 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} & \begin{pmatrix} 1 & 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} -2 \\ -1 \end{pmatrix} \\ \begin{pmatrix} 2 & 3 & 1 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} & \begin{pmatrix} 2 & 3 & 1 & 2 \end{pmatrix} \begin{pmatrix} -2 \\ -1 \end{pmatrix} \\ \begin{pmatrix} 3 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} & \begin{pmatrix} 3 & 2 & 1 & 0 \end{pmatrix} \begin{pmatrix} -2 \\ -1 \end{pmatrix} \end{bmatrix} = \begin{bmatrix} 3 & 4 & -3 & -5 \\ 4 & 7 & -5 & -8 \\ 5 & 2 & -7 & -4 \end{bmatrix},$$

or

$$A \ltimes B = A(B \otimes I_2) = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 2 & 3 & 1 & 2 \\ 3 & 2 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 & -0 \\ 0 & 1 & -0 & -2 \\ 2 & 0 & -1 & -0 \\ 0 & 2 & -0 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 4 & -3 & -5 \\ 4 & 7 & -5 & -8 \\ 5 & 2 & -7 & -4 \end{bmatrix}.$$

$\square$

Definition 1.4 *1. An $n \times p$ matrix, A, is called a logical matrix if*

$$A = \left[\delta_n^{i_1} \ \delta_n^{i_2} \ \cdots \ \delta_n^{i_p}\right], \tag{3}$$

where δ_n^i is the i-th column of the identity matrix I_n.

2. The condense form of a logical matrix (as A in (3)) is denoted as

$$A = \delta_n[i_1, i_2, \cdots, i_p]. \tag{4}$$

Remark 1.5 *According to (4), an $n \times p$ logical matrix is described by a vector of dimension p and a parameter n. In the toolbox `lm` object is used to express a logical matrix as*

$$\begin{cases} \texttt{lm}.n = n, \\ \texttt{lm}.v = [i_1, i_2, \cdots, i_p]. \end{cases} \tag{5}$$

Example 1.6 *Consider*

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

It is a logical matrix and it can be expressed in condensed form as

$$A = \delta_4[1, 3, 2, 4].$$

Expressing A to a `lm` object, we have

$$\begin{cases} A.n = 4, \\ A.v = [1\ 3\ 2\ 4]. \end{cases}$$

In this expression we only need 5 bytes instead of 16 bytes in the memory. $\square$

Definition 1.7 *The swap matrix* $W_{[m,n]}$ *is an* $mn \times mn$ *matrix constructed in the following way: label its columns by* $(11, 12, \cdots, 1n, \cdots, m1, m2, \cdots, mn)$ *and its rows by* $(11, 21, \cdots, m1, \cdots, 1n, 2n, \cdots, mn)$. *Then its element in the position* $((I, J), (i, j))$ *is assigned as*

$$w_{(IJ),(ij)} = \delta_{i,j}^{I,J} = \begin{cases} 1, & I = i \text{ and } J = j, \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

Remark 1.8 *Let* $X \in \mathbb{R}^m$ *and* $Y \in \mathbb{R}^n$ *be two columns. Then*

$$W_{[m,n]} \ltimes X \ltimes Y = Y \ltimes X. \tag{7}$$

Example 1.9 *Let* $m = 2$ *and* $n = 3$*, the swap matrix* $W_{[2,3]}$ *is constructed as*

$$W_{[2,3]} = \begin{array}{c} \begin{array}{cccccc} (11) & (12) & (13) & (21) & (22) & (23) \end{array} \\ \left[\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \end{array} \begin{array}{c} \\ (11) \\ (21) \\ (12) \\ (22) \\ (13) \\ (23) \end{array}.$$

In condensed form we have

$$W_{[2,3]} = \delta_6[1, 3, 5, 2, 4, 6].$$

□

Definition 1.10 *Let* A *be an* $m \times n$ *matrix,* $m = pq$*,* $n = rs$*. Express* A *in blocks as*

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1s} \\ A_{21} & A_{22} & \cdots & A_{2s} \\ \vdots & & & \\ A_{q1} & A_{q2} & \cdots & A_{qs} \end{bmatrix}, \tag{8}$$

where A_{ij} *are* $p \times r$ *matrices. Then the block transpose* $A^{T(p,r)}$ *is defined as*

$$A = \begin{bmatrix} A_{11} & A_{21} & \cdots & A_{q1} \\ A_{12} & A_{22} & \cdots & A_{q2} \\ \vdots & & & \\ A_{1s} & A_{2s} & \cdots & A_{qs} \end{bmatrix}. \tag{9}$$

2 Functions and Objects

This section provides detailed description for the basic functions in this toolbox. Both Matlab and Octave support the object-oriented programming, thus the toolbox defines `stp` object for the calculations of STP and `lm` object for the logical matrices.

2.1 Basic Functions

1. $C = sp(A, B)$

   Description: The function performs the (left) semi-tensor product of two matrices A and B based on Definition 1.2.

   Argument(s): Two matrices A and B with arbitrary dimensions.

   Returned Value: $C = A \ltimes B$.

2. $C = sp1(A, B)$

   Description: The function performs the (left) semi-tensor product of two matrices A and B according to Definition 1.1.

   Argument(s): Two matrices A and B with arbitrary dimensions.

   Returned Value: $C = A \ltimes B$.

   Note that sp and $sp1$ are functionally same. Because they use different algorithms inside, $sp1$ should be faster than sp for multiple dimension condition.

3. $C = spn(A_1, A_2, \cdots, A_n)$

   Description: The function performs the (left) semi-tensor product of finite set of matrices $A_1, \cdots, A_n$.

   Argument(s): Finite matrices $A_1, \cdots, A_n$ which are of arbitrary dimension.

   Returned Value: $C = \ltimes_{i=1}^{n} A_i$.

4. $B = bt(A, p, r)$

   Description: The function performs the block transpose of A (refer to Definition 1.10).

   Argument(s): A is the matrix to be transposed, the size of fixed blocks is $p \times r$.

   Returned Value: $B = A^{\mathrm{T}(p,r)}$.

5. $W = wij(m, n)$

   Description: The function produces an $mn \times mn$ swap matrix (refer to Definition 1.7).

   Argument(s): Two positive integers m and n. n is optional, default n is m.

   Returned Value: Matrix W of dimension $mn \times mn$.

6. $v = vc(A)$

   Description: The function converts a matrix to its column stacking form.

   Argument(s): Matrix $A = (a_{ij})_{m\times n}$.

   Returned Value: $v = [a_{11} \; \cdots \; a_{m1} \; \cdots \; a_{1n} \; \cdots \; a_{mn}]^{\mathrm{T}}$.

7. $v = vr(A)$

   Description: The function converts a matrix to its row stacking form.

   Argument(s): Matrix $A = (a_{ij})_{m\times n}$.

   Returned Value: $v = [a_{11} \; \cdots \; a_{1n} \; \cdots \; a_{m1} \; \cdots \; a_{mn}]^{\mathrm{T}}$.

8. $A = invvc(x, m)$

   Description: Let $x = (x_1, x_2, \cdots, x_p)$. The function will reshape x into a matrix A with row number m as

$$A = \begin{bmatrix} x_1 & x_{m+1} & \cdots & x_{p-m+1} \\ x_2 & x_{m+2} & \cdots & x_{p-m+2} \\ \vdots & & & \\ x_m & x_{2m} & \cdots & x_p \end{bmatrix}.$$

   If p is not a multiple of m, the least number of zeros will be added at the end of x such that the length of x becomes a multiple of m.

   Argument(s): x is a vector; m is the row number of the resulting matrix, and it is optional. Default m is $ceil(sqrt(length(v)))$.

   Returned Value: Matrix A with row number m.

9. $A = invvr(x, n)$

   Description: Let $x = (x_1, x_2, \cdots, x_p)$. The function will reshape x into a matrix A with column number n as

   $$A = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \\ x_{n+1} & x_{n+2} & \cdots & x_{2n} \\ \vdots & & & \\ x_{p-n+1} & x_{p-n+2} & \cdots & x_p \end{bmatrix}.$$

   If p is not a multiple of n, the least number of zeros will be added at the end of x such that the length of x becomes a multiple of n.

   Argument(s): x is a vector; m is the column number of the resulting matrix, and it is optional. Default m is $ceil(sqrt(length(v)))$.

   Returned Value: Matrix A with column number m.

10. $v = dec2any(a, k, len)$

    Description: The function converts a decimal number a into a k-based number as

    $$a = a_s k^s + a_{s-1} k^{s-1} + \cdots + a_1 k + a_0, \quad a_s > 0.$$

    [Say, $k = 2$, the result is a binary number in vector form. In fact the function $dec2binv(a, len)$ is for binary case, which is a wrapper of this function. Note that in Matlab/Octave there is *dec2base* (or *dec2bin*) to do the same thing, but its returned value is a string.]

    Argument(s): a is a positive integer; k is optional, and $k \geq 2$. Default k is 2. Default len is 0, it means $a_s \neq 0$, but if $len > 0$ and $len > s + 1$, $len - s - 1$ zeros should be added at the beginning of returned value.

    Returned Value: $v = [a_s \; a_{s-1} \; \cdots \; a_1 \; a_0]$.

2.2 `stp` Object

Here we would like to introduce the functions for `stp` object.

1. $M = stp(A)$

   Description: `stp` class constructor.

   Argument(s): Matrix A.

   Returned Value: `stp` object M.

Since `stp` object is a simple wrapper, more usage could be found in the examples in Section 3.

2.3 `lm` Object

Now we will introduce the functions for logical matrices or `lm` object.

1. $M = lm(A)$ or $M = lm(v, n)$

   Description: `lm` class constructor.

   Argument(s): i) Logical matrix A; ii) vector $v = [v_1 \; v_2 \; \cdots \; v_p]$ and positive integer n satisfying $0 \leq v_i \leq n, \; 1 \leq i \leq p$. (Case i, refer to Definition 1.4 and Example 1.6; Case ii, $lm.n = n$, $lm.v = v$.)

   Returned Value: `lm` object M.

2. $C = lsp(A, B)$

   Description: The function performs the semi-tensor product of logcial matrices A and B.

   Argument(s): A, B are `lm` objects. (refer to Definition 1.4 and Remark 1.5 for the structure.)

   Returned Value: $C = A \ltimes B$ is an `lm` object.

3. $C = lspn(A_1, A_2, \cdots, A_n)$

   Description: The function performs the semi-tensor product of logical matrices $A_1, A_2, \cdots, A_n$.

   Argument(s): $A_1, \cdots, A_n$ are `lm` objects. (refer to Definition 1.4 and Remark 1.5 for the structure.)

   Returned Value: $C = \ltimes_{i=1}^{n} A_i$ is an `lm` object.

4. $M = leye(n)$

   Description: The function produces an $n \times n$ identity matrix.

   Argument(s): Positive integer n.

   Returned Value: `lm` object M.

5. $M = lmn(k)$

   Description: The function produces the structure matrix of negation for k-valued logic ($k \geq 2$).

   Argument(s): k is optional, default k is 2.

   Returned Value: `lm` object M.

6. $M = lmc(k)$

   Description: The function produces the structure matrix of conjunction for k-valued logic ($k \geq 2$).

   Argument(s): k is optional, default k is 2.

   Returned Value: `lm` object M.

7. $M = lmd(k)$

   Description: The function produces the structure matrix of disjunction for k-valued logic ($k \geq 2$).

   Argument(s): k is optional, default k is 2.

   Returned Value: `lm` object M.

8. $M = lmi(k)$

   Description: The function produces the structure matrix of implication for k-valued logic ($k \geq 2$).

   Argument(s): k is optional, default k is 2.

   Returned Value: `lm` object M.

9. $M = lme(k)$

   Description: The function produces the structure matrix of equivalance for k-valued logic ($k \geq 2$).

   Argument(s): k is optional, default k is 2.

   Returned Value: `lm` object M.

10. $M = lmr(k)$

    Description: The function produces the power-reducing matrix for k-valued logic ($k \geq 2$).

    Argument(s): k is optional, default k is 2.

    Returned Value: `lm` object M.

11. $M = lmu(k)$

    Description: The function produces the dummy matrix for k-valued logic ($k \geq 2$). The dummy matrix M satisfies the following property

    $$MXY = Y, \quad \forall\, X, Y \in D_k.$$

    Argument(s): k is optional, default k is 2.

    Returned Value: `lm` object M.

12. $M = lmrand(m, n)$

    Description: The function produces an $m \times n$ logical matrix randomly.

    Argument(s): Positive integers m and n. n is optional, default n is m.

    Returned Value: `lm` object M.

13. $M = lwij(m, n)$

    Description: The function produces an $mn \times nn$ swap matrix.

    Argument(s): Positive integers m and n. n is optional, default n is m.

    Returned Value: `lm` object M.

14. $M = randlm(m, n)$

    Description: Alias function of *lmrand*.

More usage on `lm` object please find in the examples in Section 3.

3 Examples

Some examples which illustrate the basic usages are listed below with codes, and more examples could be found in the toolbox.

```
% This example is to show how to perform semi-tensor product

x = [1 2 3 -1];
y = [2 1]';
r1 = sp(x,y)
% r1 = [5, 3]

x = [2 1];
y = [1 2 3 -1]';
r2 = sp(x,y)
% r2 = [5; 3]

x = [1 2 1 1;
     2 3 1 2;
     3 2 1 0];
y = [1 -2;
     2 -1];
r3 = sp(x,y)
r4 = sp1(x,y)
% r3 = r4 = [3,4,-3,-5;4,7,-5,-8;5,2,-7,-4]

r5 = sp(sp(x,y),y)
r6 = spn(x,y,y)
% r5 = r6 = [-3,-6,-3,-3;-6,-9,-3,-6;-9,-6,-3,0]
% This example is to show the usage of stp class.
% Many useful methods are overloaded for stp class, thus you can use stp object as double.

x = [1 2 1 1;
     2 3 1 2;
     3 2 1 0];
y = [1 -2;
     2 -1];

% Covert x and y to stp class
a = stp(x);
b = stp(y);

% mtimes method is overloaded by semi-tensor product for stp class
c0 = spn(x,y,y)
c = a*b*b, class(c)

% Convert an stp object to double
c1 = double(c), class(c1)

% size method for stp class
size(c)

% length method for stp class
length(c)

% subsref method for stp class
c(1,:)

% subsasgn method for stp class
c(1,1) = 3
```

```
% This example is to show the usage of lm class.
% Many methods are overloaded for lm class.

% Consider classical (2-valued) logic here
k = 2;

T = lm(1,k); % True
F = lm(k,k); % False

% Given a logical matrix, and convert it to lm class
A = [1 0 0 0;
     0 1 1 1]
M = lm(A)
% or we can use
% M = lm([1 2 2 2], 2)

% Use m-function to perform semi-tensor product for logical matrices
r1 = lspn(M,T,F)

% Use overloaded mtimes method for lm class to perform semi-tensor product
r2 = M*T*F

% Create an 4-by-4 logical matrix randomly
M1 = lmrand(4)
% M1 = randlm(4)

% Convert an lm object to double
double(M1)

% size method for lm class
size(M1)

% diag method for lm class
diag(M1)

% Identity matrix is a special type of logical matrix
I3 = leye(3)

% plus method is overloaded by Kronecher product for lm class
r3 = M1 + I3
% Alternative way to perform Kronecher product of two logical matrices
r4 = kron(M1,I3)

% Create an lm object by assignment
M2 = lm;
M2.n = 2;
M2.v = [1 1 2 2];
M2
```

```
% This example is to show how to use vector form of logic to solve the following question:
% A said B is a liar, B said C is a liar, and C said A and B are both liars. Who is the liar?

% Set A: A is honest, B: B is honest, C: C is honest

k = 2;        % Two-valued logic
MC = lmc(k); % structure matrix for conjunction
ME = lme(k); % structure matrix for equivalance
MN = lmn(k); % structure matrix for negation
MR = lmr(k); % power-reducing matrix

% The logical expression can be written as
logic_expr = '(A=!B)&(B=!C)&(C=(!A&!B))';
% where = is equivalance, & is conjunction, and ! is negation

% convert the logic expresson to its matrix form
matrix_expr = lmparser(logic_expr);

% then obtain its canonical matrix form
expr = stdform(matrix_expr);

% calculate the structure matrix
L = eval(expr)

% The uniqe solution for L*x=[1 0]^T is x=[0 0 0 0 0 1 0 0]^T:=8[6]
sol = v2s(lm(6,8))

% One can see sol=[0 1 0], which means only B is honest, A and C are liars.
```

```
% Examples for Boolean network

% Initializing
k = 2;
```

```
options = [];

% Please note that in this toolbox any variable intialized with capital M is defined as a logical matrix, otherwise it will be considered as logical vector.
% The followings are some commonly used logical matrices
ME = lme(k); % equivalence
MI = lmi(k); % implicaiton
MD = lmd(k); % disjunction
MN = lmn(k); % negation
MR = lmr(k); % power-reducing matrix
MC = lmc(k); % conjunction
MX = lm([2 1 1 2], 2); % xor

% choose a number from 1-5 to select a Boolean network
n = 3;

switch n
    case 1
        % Dynamics of Boolean network
            % A(t+1) = MC*B(t)*C(t)
            % B(t+1) = MN*A(t)
            % C(t+1) = MD*B(t)*C(t)
        % Set X(t)=A(t)B(t)C(t), then
        eqn = 'MC B C MN A MD B C';
    case 2
        % Dynamics of Boolean network
        % A(t+1) = MC*B(t)*C(t)
        % B(t+1) = MN*A(t)
        % C(t+1) = B(t)
        eqn = 'MC B C MN A B';
    case 3
        % Dynamics of Boolean network
            % E(t+1) = MX*E(t)*I(t)
            % H(t+1) = MX*F(t)*H(t)
            % F(t+1) = MX*F(t)*J(t)
            % I(t+1) = MX*G(t)*I(t)
            % G(t+1) = MX*G(t)*MX*F(t)*H(t)
            % J(t+1) = MX*MX*E(t)*I(t)*J(t)
        % Set X(t)=E(t)H(t)F(t)I(t)G(t)J(t), then
        if k ≠ 2
            error('This example is only for the case k=2.');
        end
        eqn = 'MX E I MX F H MX F J MX G I MX G MX F H MX MX E I J ';
        % set the variables' order, otherwise they will be sorted in the dictionary order
        options = lmset('vars',{'E','H','F','I','G','J'});
    case 4
        % Dynamics of Boolean network
            % A(t+1) = MN*MI*K(t)*H(t)
            % B(t+1) = MN*MI*A(t)*C(t)
            % C(t+1) = MI*D(t)*I(t)
            % D(t+1) = MC*J(t)*K(t)
            % E(t+1) = MI*C(t)*F(t)
            % F(t+1) = MN*MI*E(t)*G(t)
            % G(t+1) = MN*MC*B(t)*E(t)
            % H(t+1) = MN*MI*F(t)*G(t)
            % I(t+1) = MN*MI*H(t)*I(t)
            % J(t+1) = J(t)
            % K(t+1) = K(t)
        % Set X(t)=A(t)B(t)C(t)D(t)E(t)F(t)G(t)H(t)I(t)J(t)K(t), then
        eqn = 'MN MI K H MN MI A C MI D I MC J K MI C F MN MI E G MN MC B E MN MI F G MN MI H I J K';
    case 5
        % Dynamics of Boolean network
            % A(t+1) = MN*MD*C(t)*F(t)
            % B(t+1) = A(t)
            % C(t+1) = B(t)
            % D(t+1) = MC*MC*MN*I(t)*MN*C(t)*MN*F(t)
            % E(t+1) = D(t)
            % F(t+1) = E(t)
            % G(t+1) = MN*MD*F(t)*I(t)
            % H(t+1) = G(t)
            % I(t+1) = H(t)
        % Set X(t)=A(t)B(t)C(t)D(t)E(t)F(t)G(t)H(t)I(t), then
        eqn = 'MN MD C F A B MC MC MN I MN C MN F D E MN MD F I G H';
    otherwise
        return
end

% Convert the equation to a canonical form
[expr,vars] = stdform(eqn,options,k);

% Calculate the network transition matrix
L = eval(expr)

% Analyze the dynamics of the Boolean network
[n,l,c,r0,T] = bn(L,k);

fprintf('Number of attractors: %d\n\n',n);
fprintf('Lengths of attractors:\n');
disp(l);
fprintf('\nAll attractors are displayed as follows:\n\n');
for i=1:length(c)
    fprintf('No. %d (length %d)\n\n',i,l(i));
    disp(c{i});
end
fprintf('Transient time: [r0, T] = [%d %d]\n\n',r0,T);
```

References

[1] D. Cheng, *Matrix and Polynomial Approach to Dynamics Control Systems*, Beijing: Science Press, 2002.

[2] D. Cheng, H. Qi, *Semi-tensor Product of Matrices — Theory and Applications*, Beijing: Science Press, 2007. (Second Edition, 2011, Both in Chinese)

[3] D. Cheng, Sime-tensor product of matrices and its applications — A survey, *Proc. ICCM 2007*, Higher Education Press, Hangzhou, 641-668, 2007.

[4] D. Cheng, H. Qi, Z.Q. Li, *Analysis and Control of Boolean Networks: A Semi-tensor Product Approach*, London: Springer, 2011.

[5] D. Cheng, H. Qi, Y. Zhao, *An Introduction to Semi-tensor Product of Matrices and Its Applications*, Singapore: World Scientific, 2012.