

# Tensor Node Notation

## Standard Representations

For every plain tree network, there exists a standard representation, as analogon to the matrix SVD. We might therefor also call it tree SVD.

The following function takes a plain network and standardizes it.

```
type STAND.m
```

```
function [N,fN,sigma,sigma_map] = STAND(N,r,G)
% STAND standardizes a plain network
%
% [N,fN,sigma] = STAND(N,r) returns the standard representation (fN,sigma)
% of the tensor represented by N. The output N is the orthogonal with
% respect to r.
%
% The input N must be a plain network.
%
% sigma will contain the singular values of the different matricizations
% corresponding to the network.
%
% It holds boxtimes(N{:}) = boxtimes(fN{:},sigma{:}).
%
% See also: TENSOR_NODE_NOTATION8_STANDARD_REPRESENTATIONS.mlx, ORTHO

if nargin <= 1
    r = 1;
end

if nargin <= 2
    G = net_derive_G(N);
end

k = numnodes(G);
fN = cell(1,k);
sigma = cell(1,k-1);
N = ORTHO(N,r,G);
STANDREC(r,[]);

function Q = STANDREC(b,P)
    Nbtilde = N{b};
    for h = int_setdiff(neighbors(G,b)',P)
        gamma = str_intersect(N{h}.mode_names,N{b}.mode_names);
        [~,R] = node_qr(N{b},gamma);
        N{h} = boxtimes(R,N{h});
        Q = STANDREC(h,b);
        Nbtilde = boxtimes(Nbtilde,matrix_node_pinv(R));
        Nbtilde = boxtimes(boxtimes_part(sigma{findedge(G,h,b)},Q,{},gamma),Nbtilde);
    end
    N{b} = Nbtilde;

    if ~isempty(P)
        gamma = str_intersect(N{P}.mode_names,N{b}.mode_names);
        [U,s,Q] = node_svd(N{b},listdiff(N{b}.mode_names,gamma),gamma,0);
        N{b} = U;
        fN{b} = U;
```

```

        ind = findedge(G,P,b);
        sigma{ind} = s;
        sigma_map.(gamma{1}) = ind;
    else
        fN{b} = N{b};
    end

    for h = int_setdiff(neighbors(G,b)',P)
        s = sigma{findedge(G,h,b)};
        s.data = (s.data).^(-1);
        fN{b} = boxtimes_part(s,fN{b},{},s.mode_names);
    end
end
end
end

```

## Standard representation for the TT-format

For the TT-format, the situation is a bit simpler and the standard representation or tree SVD is also known as canonical MPS format. We write it as extended representation

$$\mathcal{G}^\sigma = (\mathcal{G}, \sigma) = (\mathcal{G}_1, \sigma^{(1)}, \mathcal{G}_2, \sigma^{(2)}, \dots, \mathcal{G}_d)$$

For  $d = 2$  this is an ordinary matrix SVD.

```

load('TT-format');
alpha = mna('alpha',1:4);
beta = mna('beta',1:3);
G{:}

```

```

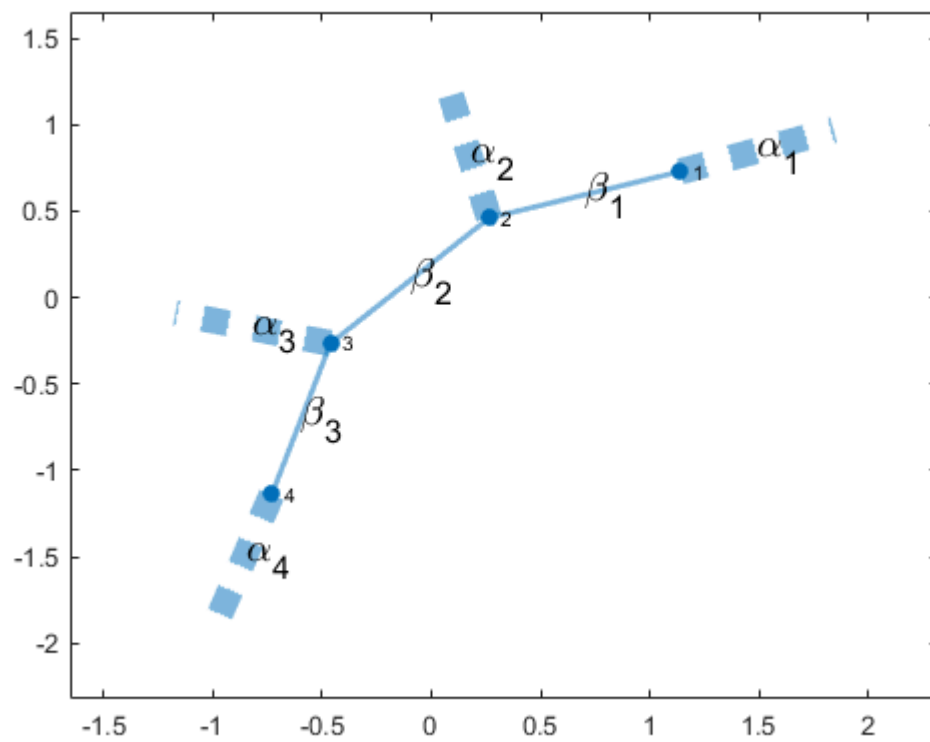
ans = struct with fields:
    mode_names: {'alpha_1' 'beta_1'}
    pos: [1x1 struct]
    data: [10x2 double]
ans = struct with fields:
    mode_names: {'beta_1' 'alpha_2' 'beta_2'}
    pos: [1x1 struct]
    data: [2x10x2 double]
ans = struct with fields:
    mode_names: {'beta_2' 'alpha_3' 'beta_3'}
    pos: [1x1 struct]
    data: [2x10x2 double]
ans = struct with fields:
    mode_names: {'beta_3' 'alpha_4'}
    pos: [1x1 struct]
    data: [2x10 double]

```

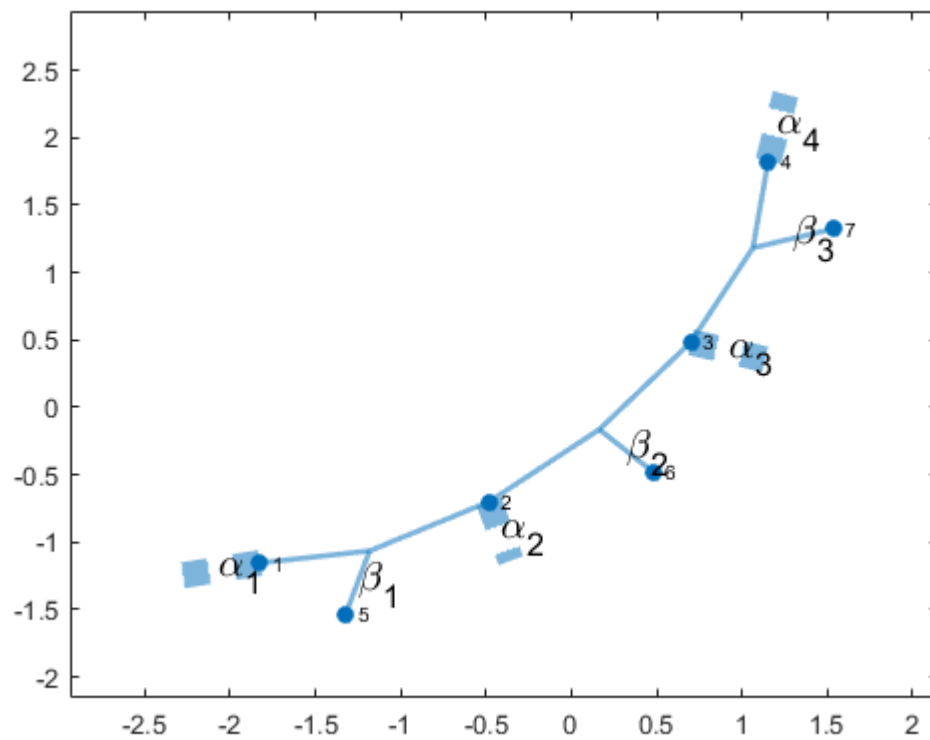
```
[G,fG,sigma,sigma_map] = STAND(G);
```

The output  $G$  represents the same tensor as before, but gauge conditions have changed. The tree SVD is contained in  $fG$  and  $\sigma$ .

```
net_view(G)
```



```
net_view(fG{:},sigma{:})
```

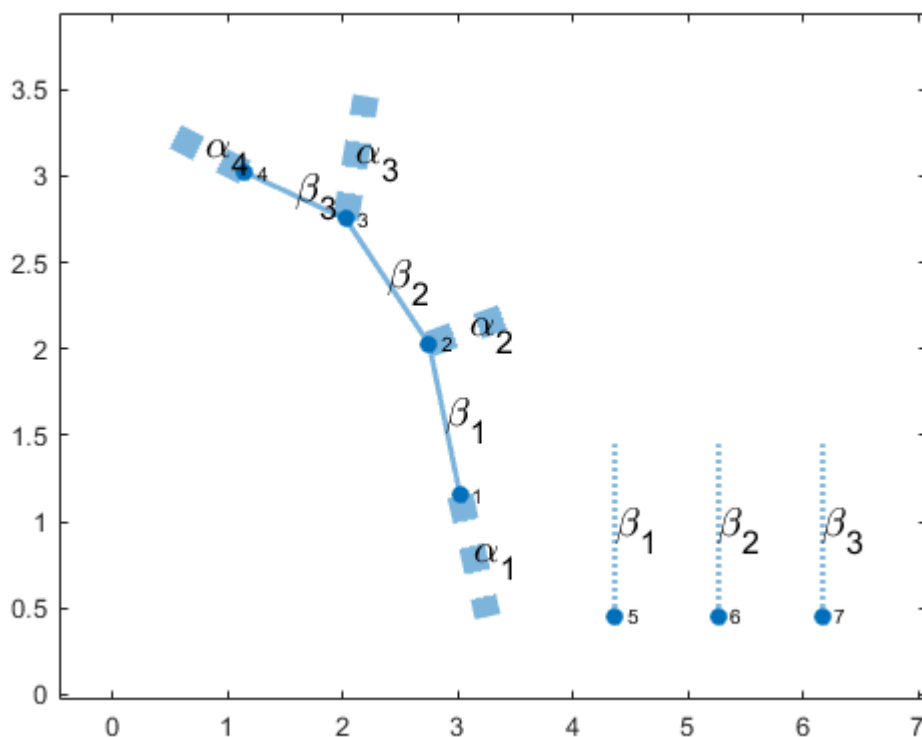


We have to be careful here. Due to the rules of nested `boxtimes` calls, the following call produces the wrong result:

```
warning('wrong contraction')
```

Warning: wrong contraction

```
net_view(fG,sigma)
```



We have added an additional node to each previous edge. There are several gauge properties fulfilled by the  $\mathcal{G}$  and  $\sigma$ . The bookkeeping to the TT-format is easy, since the edges in the graph of  $G$  are already sorted. Hence the output `sigma_map` is already sorted and we do not need it. For other networks, this output is needed to relate the entries of `sigma` to the network.

```
sigma_map
```

```
sigma_map = struct with fields:
    beta_3: 3
    beta_2: 2
    beta_1: 1
```

```
graphG = net_derive_G(G);
graphG.Edges
```

```
ans = 3x1 table
```

	EndNodes	
1	1	2
2	2	3
3	3	4

Together,  $\mathcal{G}$  and  $\sigma$  form all node SVDs of  $T = [\times] G$  which are contained in the TT-format.

```
T = boxtimes(G);
[U_,sigma_,V_] = deal(cell(1,4));
for i = 1:3
    [U_{i},sigma_{i},V_{i}] = node_svd(T,alpha(1:i),beta(i));
end

for i = 1:3
    Ui = boxtimes(fG{1:i},sigma{1:i-1});
    get_data(boxtimes(U_{i},Ui,'_',alpha)) % this should be diagonal with entries in {
    Vi = boxtimes(fG{i+1:end},sigma{i+1:end});
    get_data(boxtimes(V_{i},Vi,'_',alpha)) % this should be diagonal with entries in {
    net_dist(sigma{i},sigma_{i})
end
```

```
ans = 2x2
    1.0000    -0.0000
    0.0000     1.0000
ans = 2x2
    1.0000    -0.0000
    0.0000     1.0000
ans = 0
ans = 2x2
    1.0000     0.0000
    0.0000    -1.0000
ans = 2x2
    1.0000     0.0000
    0.0000    -1.0000
ans = 0
ans = 2x2
    1.0000    -0.0000
   -0.0000    -1.0000
ans = 2x2
    1.0000    -0.0000
   -0.0000    -1.0000
ans = 0
```

Additionally, the single  $\mathcal{G}_i$  together with  $\sigma_i$  fulfill orthogonality properties as well.

```
fGs2 = boxtimes(fG{2},sigma{2},'^',beta(2)); % fGs2 is beta_1-orthogonal
get_data(boxtimes(fGs2,fGs2,'_\',beta(1))) % contract all but beta(1)
```

```
ans = 2x2
```

```

1.0000 -0.0000
-0.0000 1.0000

```

```

fGs3 = boxtimes(fG{3},sigma{3},'^',beta(3)); % beta_2-orthogonal
get_data(boxtimes(fGs3,fGs3,'_\',beta(2)))

```

```

ans = 2x2
1.0000 0.0000
0.0000 1.0000

```

```

get_data(boxtimes(fG{4},fG{4}','_\',beta(3))) % fG{3} is beta_3-orthogonal

```

```

ans = 2x2
1.0000 0.0000
0.0000 1.0000

```

```

get_data(boxtimes(fG{1},fG{1}','_\',beta(1))) % fG{1} is beta_1-orthogonal

```

```

ans = 2x2
1.0000 -0.0000
-0.0000 1.0000

```

```

sfG2 = boxtimes(sigma{1},fG{2},'^',beta(1)); % beta_2-orthogonal
get_data(boxtimes(sfG2,sfG2,'_\',beta(2)))

```

```

ans = 2x2
1.0000 0.0000
0.0000 1.0000

```

```

sfG3 = boxtimes(sigma{2},fG{3},'^',beta(2)); % beta_3-orthogonal
get_data(boxtimes(sfG3,sfG3,'_\',beta(3)))

```

```

ans = 2x2
1.0000 -0.0000
-0.0000 1.0000

```

## Standard representation (tree SVD) for plain networks

Analogous orthogonality conditions hold for arbitrary plain networks:

```

load('random-format');
n = node_size(NR)

```

```

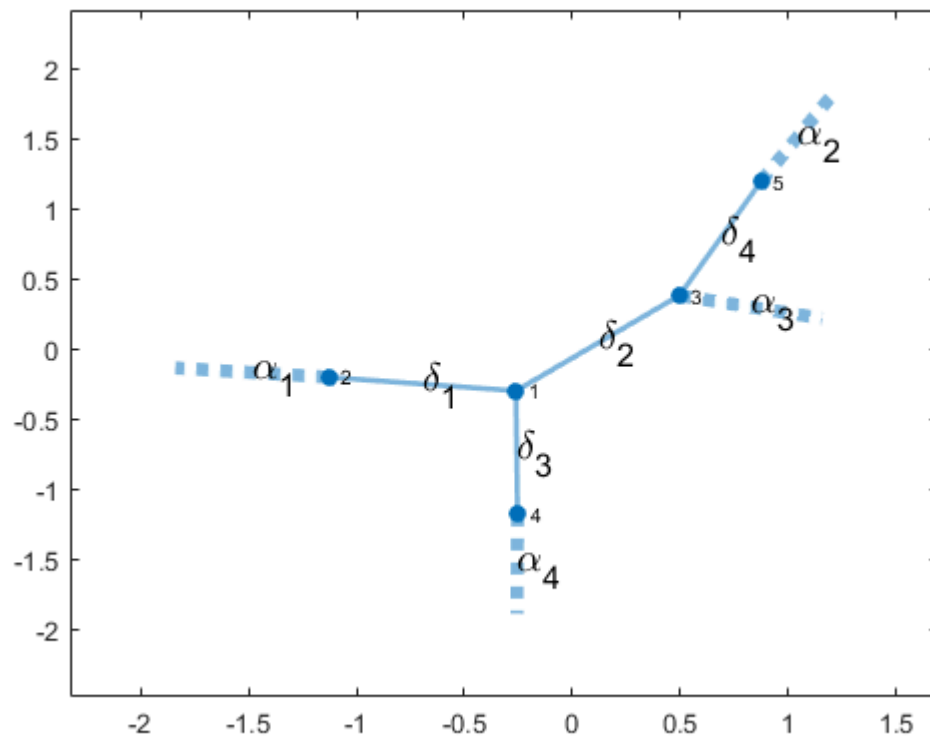
n = struct with fields:
alpha_1: 5
alpha_2: 5
alpha_3: 5
alpha_4: 5
delta_1: 2
delta_2: 2
delta_3: 2
delta_4: 2

```

```

net_view(NR)

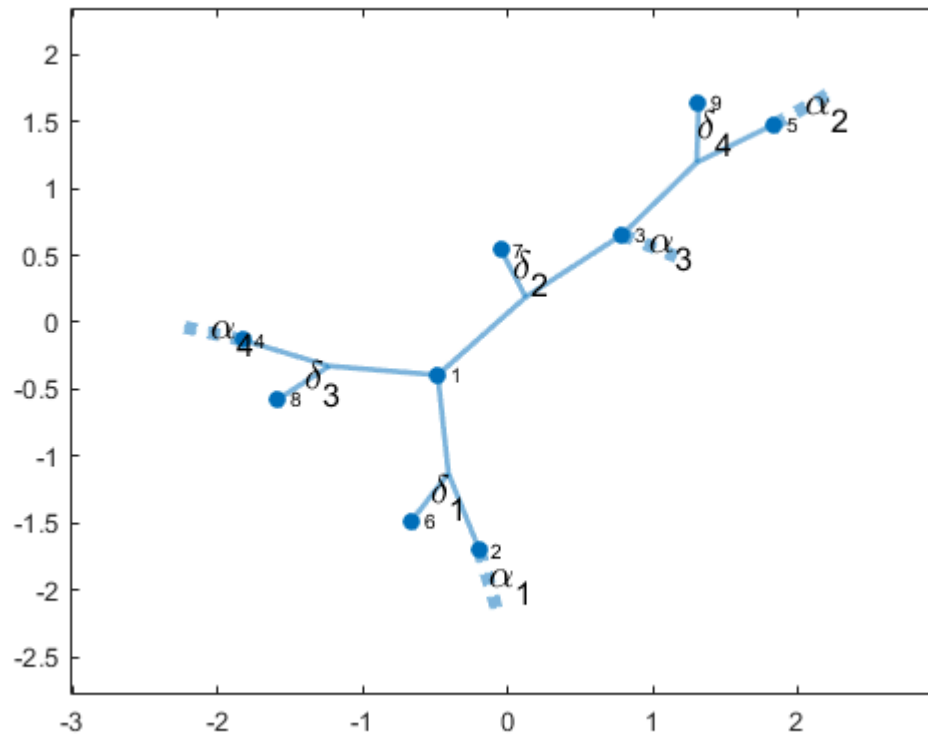
```



```
[NR,fNR,sigmaR,sigmaR_map] = STAND(NR);
sigmaR_map
```

```
sigmaR_map = struct with fields:
    delta_1: 1
    delta_4: 4
    delta_2: 2
    delta_3: 3
```

```
net_view([fNR,sigmaR])
```



To check our gauge conditions, we need to find out which SVDs are contained in this specific format.

```
[G,m] = net_derive_G(NR);
r = 1;
[M,m_prec] = derive_M(G,m,r)
```

```
M = 5×1 cell array
    {1×4 cell}
    {1×1 cell}
    {1×2 cell}
    {1×1 cell}
    {1×1 cell}
m_prec = 5×1 cell array
    {0×0 double}
    {'delta_1' }
    {'delta_2' }
    {'delta_3' }
    {'delta_4' }
```

Now,  $M\{k\}$  contains all outer mode names (the dashed ones) of the nodes within  $\text{branch}_r(k)$  and  $m\_prec\{k\}$  the mode name of the  $k$ -th node that points in direction of the root node  $r$ .

```
NR{3}
```

```
ans = struct with fields:
    mode_names: {'delta_4' 'alpha_3' 'delta_2'}
    pos: [1×1 struct]
```



```
data: [2×5×2 double]
```

```
M{3}
```

```
ans = 1×2 cell array  
    {'alpha_2'}    {'alpha_3'}
```

```
m_prec{3}
```

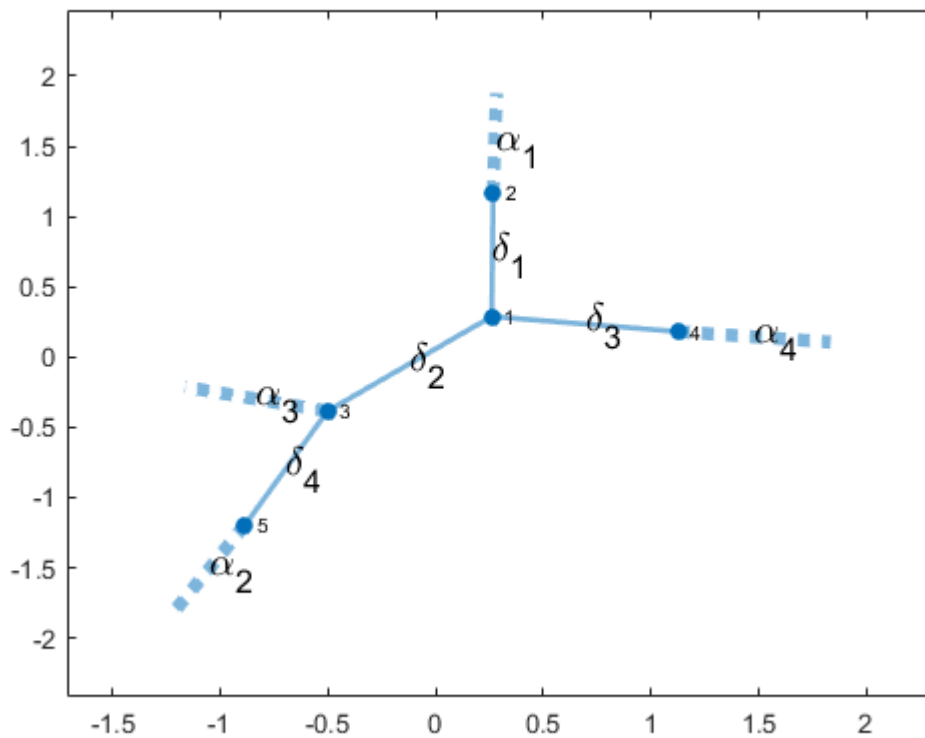
```
ans =  
'delta_2'
```

To determine these branches, we can call the equally named `branches`. The output  $\text{Br}\{k\}\{2\}$  then equals  $\text{branch}_r(k)$ , the branch starting from (and including) node  $k$  with respect to the root  $r$  in the tree  $G$ . The entry  $\text{Br}\{k\}\{1\}$  is its complement.

```
Br = branches(G,r)
```

```
Br = 1×5 cell array  
    {1×2 cell}    {1×2 cell}    {1×2 cell}    {1×2 cell}    {1×2 cell}
```

```
net_view(NR)
```



```
Br{3}{1}
```

```
ans = 1×3
```

1      2      4

```
Br{3}{2}
```

```
ans = 1x2
      3    5
```

```
m_prec(int_setdiff(Br{3}{1},1))
```

```
ans = 2x1 cell array
      {'delta_1'}
      {'delta_3'}
```

```
m_prec(int_setdiff(Br{3}{2},3))
```

```
ans = 1x1 cell array
      {'delta_4'}
```

Now we can check the according SVDs. The bookkeeping in an arbitrary network is a bit nasty, so that we have to rely on the graph information:

```
TR = boxtimes(NR)
```

```
TR = struct with fields:
    mode_names: {'alpha_1' 'alpha_3' 'alpha_4' 'alpha_2'}
        pos: [1x1 struct]
        data: [5x5x5x5 double]
```

```
nm = length(M);
[U_,sigma_,V_] = deal(cell(1,nm));
for i = 2:nm
    [U_{i},sigma_{i},V_{i}] = node_svd(TR,M{i},m_prec{i});
end

for i = 2:nm
    % U
    m_i_inner = m_prec(int_setdiff(Br{i}{2},i))
    sig_ind = getfields(sigmaR_map,m_i_inner)
    if ~isempty(sig_ind)
        sig_ind = [sig_ind{:}];
    end
    Ui = boxtimes(fNR{Br{i}{2}},sigmaR{sig_ind});
    get_data(boxtimes(U_{i},Ui,'_\ ',m_prec{i}))

    % sigma
    net_dist(sigmaR{sigmaR_map.(m_prec{i})},sigma_{i})

    % equality (up to signs) of V follows by the ones of U and sigma
end
```

```
m_i_inner =
    0x0 empty cell array
```

```

sig_ind =

    []
ans = 2x2
    1.0000    -0.0000
    0.0000     1.0000
ans = 3.7253e-09
m_i_inner = 1x1 cell array
    {'delta_4'}
sig_ind = 1x1 cell array
    {[4]}
ans = 2x2
   -1.0000    -0.0000
    0.0000   -1.0000
ans = 0
m_i_inner =

    0x0 empty cell array
sig_ind =

    []
ans = 2x2
    1.0000     0.0000
    0.0000    -1.0000
ans = 3.7253e-09
m_i_inner =

    0x0 empty cell array
sig_ind =

    []
ans = 2x2
    1.0000     0.0000
   -0.0000     1.0000
ans = 0

```

We can also check orthogonality conditions for single nodes:

For example, the following node `fNRs` is orthogonal with respect to a certain mode size.

```
inner_mode_names = m_prec(2:end)'
```

```
inner_mode_names = 1x4 cell array
    {'delta_1'}    {'delta_2'}    {'delta_3'}    {'delta_4'}
```

```
i = 1;
k = 3;
fNR{k}
```

```
ans = struct with fields:
    mode_names: {'delta_4' 'alpha_3' 'delta_2'}
        pos: [1x1 struct]
        data: [2x5x2 double]
```

```
m_k_inner = str_intersect(inner_mode_names,m{k})
```

```
m_k_inner = 1x2 cell array
    {'delta_4'}    {'delta_2'}
```

We select all but one tuple of singular values next to node  $G\{k\}$ .

```
sig_ind = getfields(sigmaR_map,m_k_inner([1:i-1,i+1:end]));
sig_ind = [sig_ind{:}]
```

```
sig_ind = 2
```

```
fNRs = boxtimes(fNR{k},sigmaR{sig_ind},'^',m_k_inner)
```

```
fNRs = struct with fields:
    mode_names: {'delta_4' 'alpha_3' 'delta_2'}
    pos: [1x1 struct]
    data: [2x5x2 double]
```

```
m_k_inner{i}
```

```
ans =
'delta_4'
```

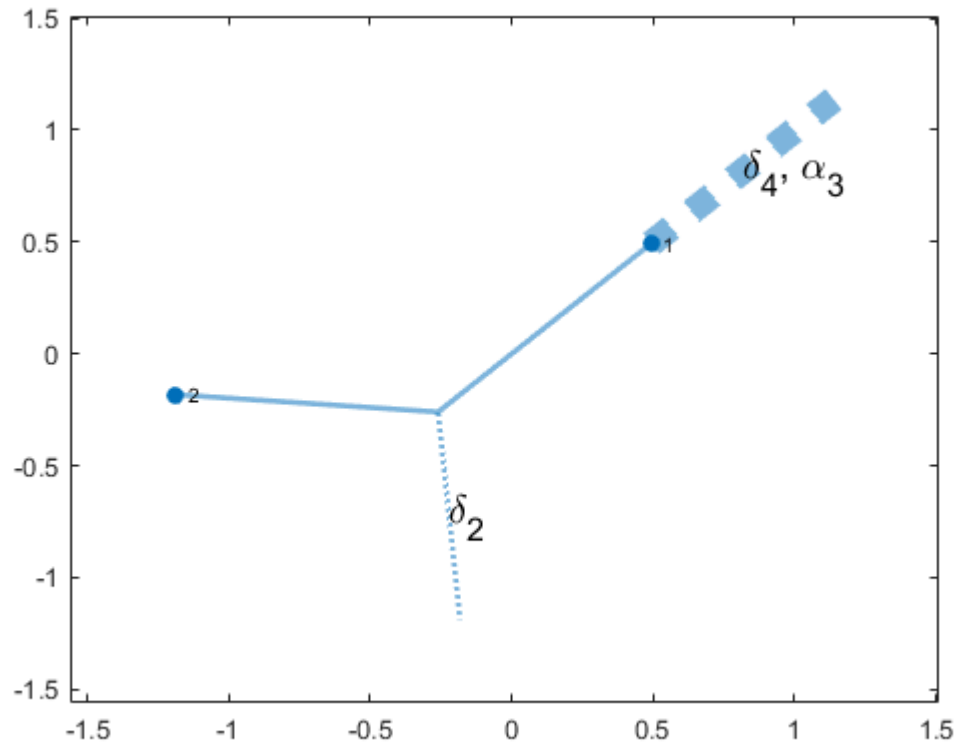
```
get_data(boxtimes(fNRs,fNRs,'_\ ',m_k_inner{i}))
```

```
ans = 2x2
    1.0000    0.0000
    0.0000    1.0000
```

```
m_k_inner{i}
```

```
ans =
'delta_4'
```

```
net_view(fNR{k},sigmaR{sig_ind},'^',m_k_inner);
```



All orthogonality constraints are checked by the following loop. Everytime we contract all but one neighboring tuples of singular values into a node, this combination is orthogonal with respect to the mode names appearing in the left out tuple.

```
for k = 1:length(fNR)

    m_k_inner = str_intersect(inner_mode_names,m{k});
    for i = 1:length(m_k_inner)
        sig_ind = getfields(sigmaR_map,m_k_inner([1:i-1,i+1:end]));
        if ~isempty(sig_ind)
            sig_ind = [sig_ind{:}];
            fNRs = boxtimes(fNR{k},sigmaR{sig_ind},'^',m_k_inner);
        else
            fNRs = fNR{k};
        end

        get_data(boxtimes(fNRs,fNRs,'_\ ',m_k_inner{i}))
    end
end
```

```
ans = 2x2
    1.0000    0.0000
    0.0000    1.0000

ans = 2x2
    1.0000   -0.0000
   -0.0000    1.0000

ans = 2x2
    1.0000   -0.0000
```

```

-0.0000    1.0000
ans = 2x2
    1.0000   -0.0000
   -0.0000    1.0000
ans = 2x2
    1.0000    0.0000
    0.0000    1.0000
ans = 2x2
    1.0000    0.0000
    0.0000    1.0000
ans = 2x2
    1.0000    0.0000
    0.0000    1.0000
ans = 2x2
    1.0000    0.0000
    0.0000    1.0000

```

The function `check_if_standard` will test with which tolerance the representation is a standard representation:

```
error = check_if_standard(fNR,sigmaR)
```

```
error = 1.5169e-15
```