

Tensor Node Notation

Plain Networks

In a plain network, each node must not have duplicate mode names and every mode name must appear in at most two nodes. Furthermore, the network must correspond to a tree graph.

```
d = 4
```

```
d = 4
```

```
alpha = mna('alpha',1:d);  
n_alpha = assign_mode_size(alpha,5);
```

Root-to-leaf graph creation - TT-format

If we do not want to specify the mode names of a specific format manually, we can use the algorithm RTLGRAPH.

```
type RTLGRAPH.m
```

```
function [G,m,M,r,inner_names] = RTLGRAPH(alpha,S,naming)  
% RTLGRAPH Generates a graph corresponding to the sequence S  
%  
% [G,m,M,r] = RTLGRAPH(alpha,S) successively adds edges that split the  
% mode names in alpha according to the sequence S.  
%  
% alpha must be a cell array containing mode names  
% 'alpha_1',...,'alpha_N' for some N.  
%  
% The entries of S must be pairwise distinct and meet a hierarchy  
% condition.  
%  
% G is the graph (Matlab formatting) corresponding to the sequence S with  
% nodes 1,...,k with k = length(S)+1.  
%  
% m is a cell, where m{i} contains the mode names associated to node  
% number i.  
%  
% M is a cell, where M{i} contains all outer mode names alpha_i  
% accumulated in branch_r(i). Thereby M{i+1} = S{i} and M{1} = alpha.  
%  
% r = 1 is the root node.  
%  
% Examples:  
% alpha = mna('alpha',[1,2,3,4]);  
% S = binary_tree(alpha);  
% [G,m,M,r] = RTLGRAPH(alpha,S)  
%  
% graph_view(G,m);  
% M{2} % returns {'alpha_1','alpha_2'}  
% m{2} % returns {'beta_1_2','beta_1','beta_2'}  
%  
% See also: TENSOR_NODE_NOTATION7_PLAIN_NETWORKS, GRAPH, VIEW,  
% BINARY_TREE, LINEAR_TREE, RANDOM_TREE, DERIVE_M
```

```

k = length(S)+1;

if nargin <= 2
    naming = [];
else
    inner_names = mna(naming,1:k-1);
end

G = graph; G = addnode(G,1);
m = cell(1,k);
m{1} = alpha;
r = 1;

M = cell(1,k-1);
M{1} = alpha;

for i = 1:k-1
    b = find_sub_leaf(G,M,S{i},1,[]);
    if isempty(naming)
        beta_name = alpha_to_beta(S{i}); % form name for edge
    else
        beta_name = inner_names{i};
    end

    G = addedge(G,b,i+1);
    m{b} = listdiff([m{b},beta_name],S{i}); % why listdiff?
    m{i+1} = [beta_name,S{i}];
    M{i+1} = S{i};

    % view(G,m);
end

end

function u = find_sub_leaf(G,M,S,h1,P)

branch = int_setdiff(neighbors(G,h1),P);
if isempty(branch)
    u = h1;
    return;
end

for h2 = branch
    if all(ismember(S,M{h2}))
        u = find_sub_leaf(G,M,S,h2,h1);
        return
    end
end

u = h1;

end

function name = alpha_to_beta(S)
name = 'beta';
nr = zeros(1,length(S));
for i = 1:length(S)
    nr(i) = str2num(strrep(S{i},'alpha',''));
end
nr = sort(nr,'ascend');
name = [name,num2str(nr,'%d')];
end

```

As input, the algorithm expects a sequence S of subsets of a set of outer mode names, for example of α . These subsets generate a graph G , and a mode name map m , which corresponds to a plain network N . Each edge in G splits the outer mode names α in two parts, one of which always is one element in the sequence S .

For the TT-format, S is quite simple:

```
S = linear_tree(alpha);
S{:}

ans = 1×3 cell array
    {'alpha_2'}    {'alpha_3'}    {'alpha_4'}
ans = 1×2 cell array
    {'alpha_3'}    {'alpha_4'}
ans = 1×1 cell array
    {'alpha_4'}
```

We only have to specify the name scheme for the, to be created, inner mode names:

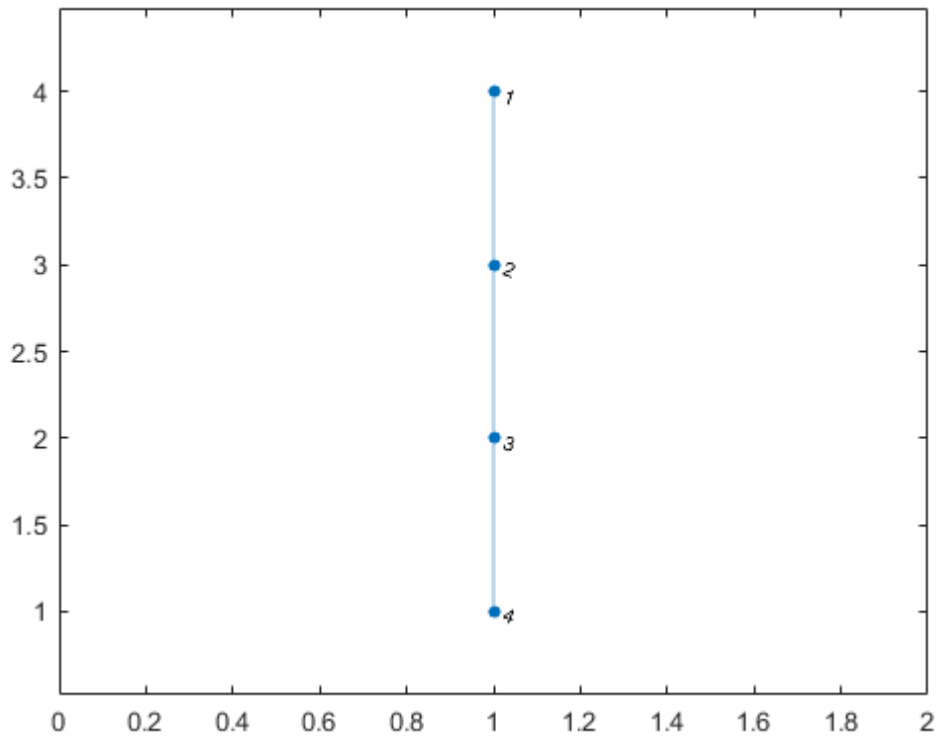
```
[G,m,~,~,beta] = RTLGRAPH(alpha,S,'beta')

G =
graph with properties:
    Edges: [3×1 table]
    Nodes: [4×0 table]
m = 1×4 cell array
    {1×2 cell}    {1×3 cell}    {1×3 cell}    {1×2 cell}
beta = 1×3 cell array
    {'beta_1'}    {'beta_2'}    {'beta_3'}
```

```
m{:}

ans = 1×2 cell array
    {'alpha_1'}    {'beta_1'}
ans = 1×3 cell array
    {'beta_1'}    {'alpha_2'}    {'beta_2'}
ans = 1×3 cell array
    {'beta_2'}    {'alpha_3'}    {'beta_3'}
ans = 1×2 cell array
    {'beta_3'}    {'alpha_4'}
```

```
plot(G)
```



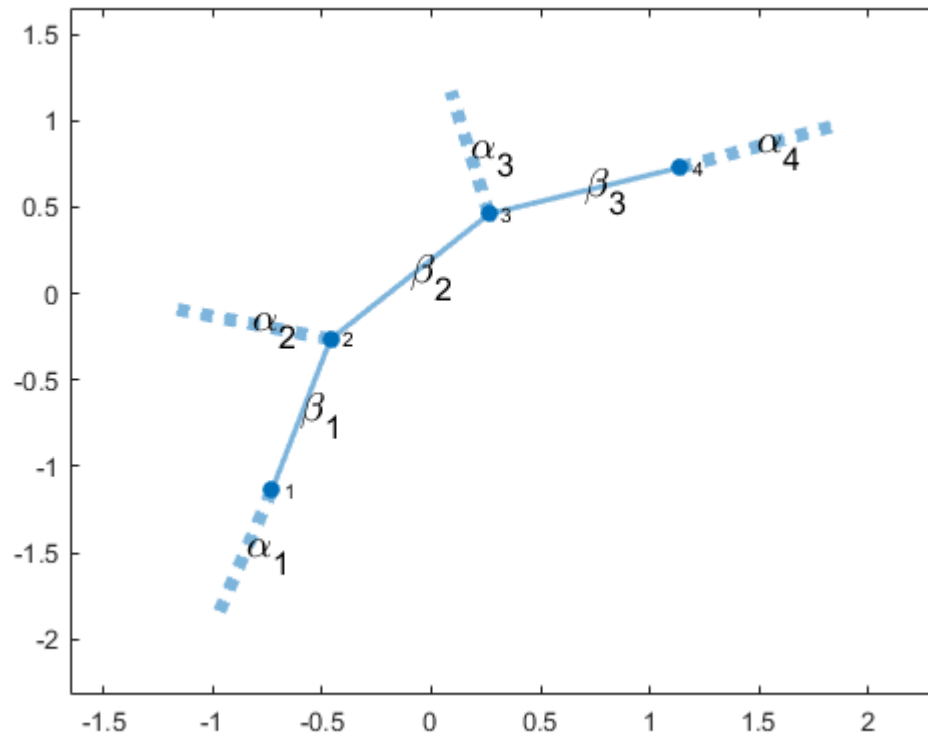
```
n_beta = assign_mode_size(beta,2);
n = merge_fields(n_alpha,n_beta);
```

We can use `m` to initialize the network:

```
NTT = init_net(m,n)
```

```
NTT = 1×4 cell array
      {1×1 struct}   {1×1 struct}   {1×1 struct}   {1×1 struct}
```

```
net_view(NTT)
```



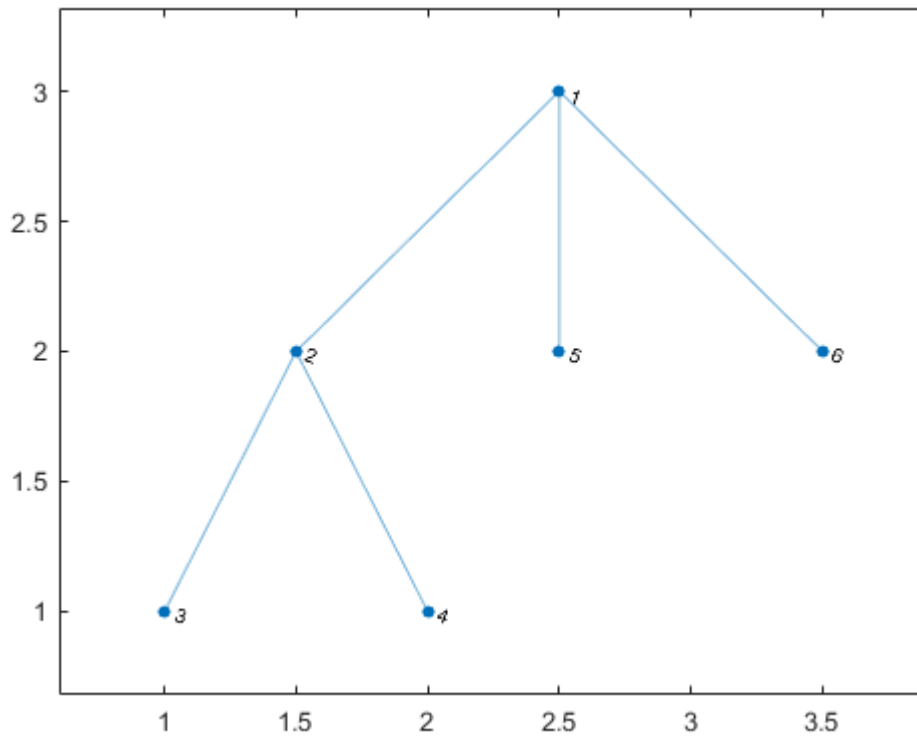
Creating a binary HT-format

The same process can be repeated for a binary HT-formats. However, in order to fit into the scheme of plain networks, there is not root transfer tensor.

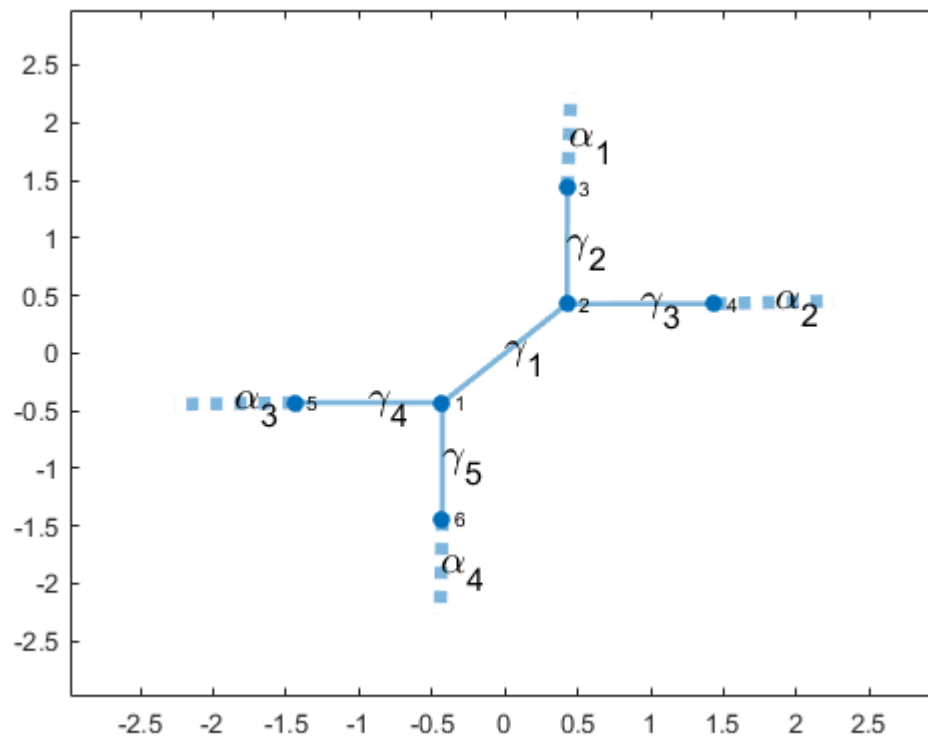
```
alpha = mna('alpha',1:d);
S = binary_tree(alpha);
S{:}
```

```
ans = 1×2 cell array
    {'alpha_1'}    {'alpha_2'}
ans = 1×1 cell array
    {'alpha_1'}
ans = 1×1 cell array
    {'alpha_2'}
ans = 1×1 cell array
    {'alpha_3'}
ans = 1×1 cell array
    {'alpha_4'}
```

```
[G,m,~,~,gamma] = RTLGRAPH(alpha,S,'gamma');
plot(G)
```



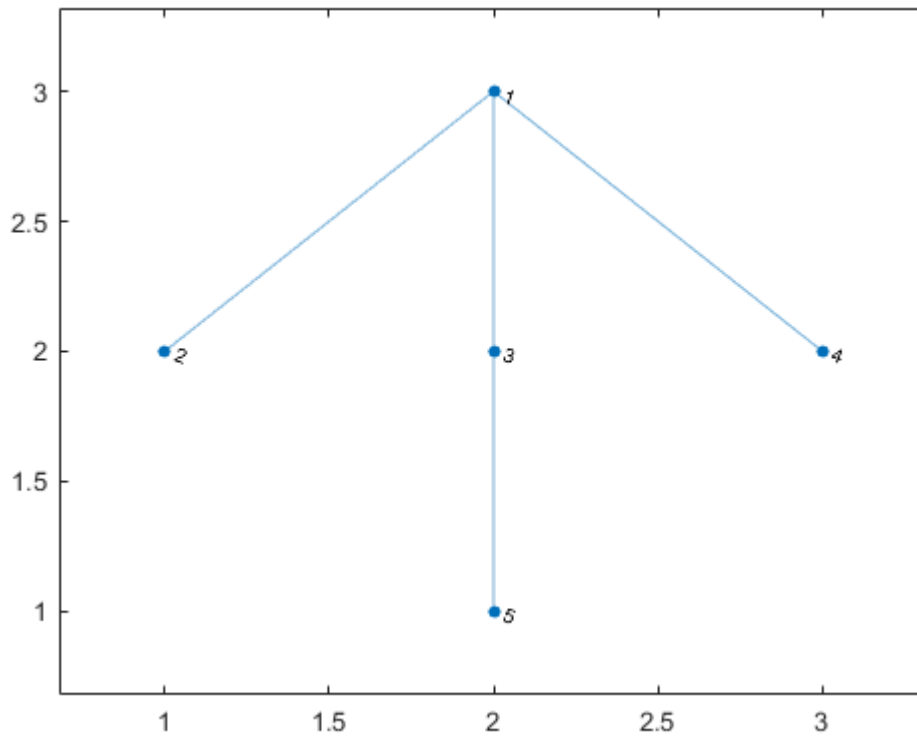
```
n_gamma = assign_mode_size(gamma,2);  
n = merge_fields(n_alpha,n_gamma);  
NHT = init_net(m,n);  
net_view(NHT)
```



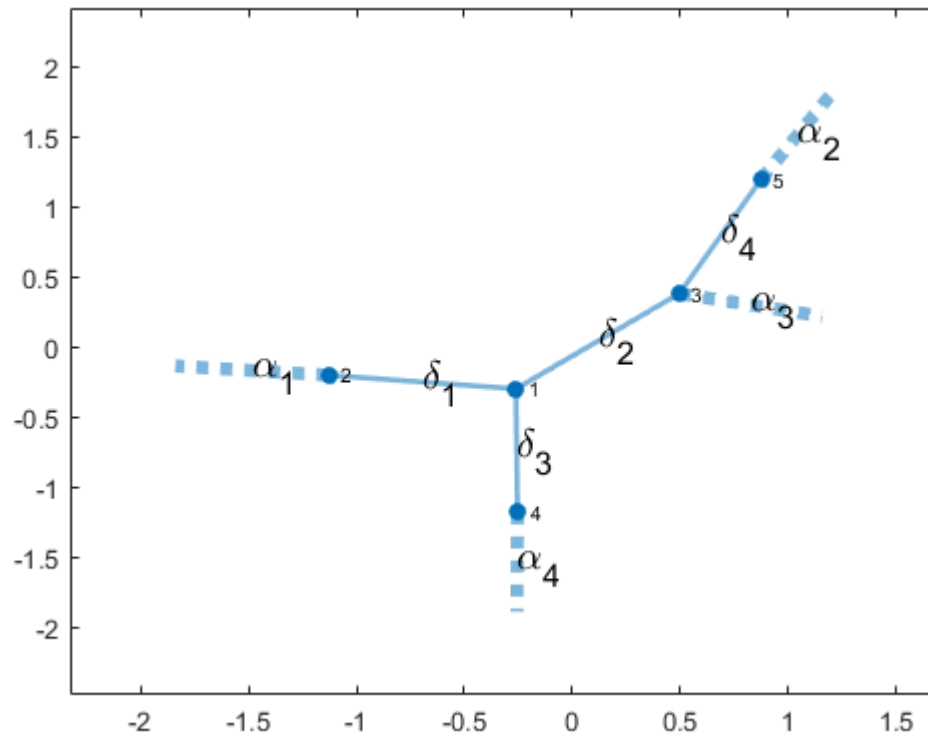
Random plain networks

The function `random_tree(alpha,k)` provides an artificially random tree that restricts single nodes to at most k connections:

```
alpha = mna('alpha',1:d);
rng(2)
S = random_tree(alpha,4);
[G,m,~,~,delta] = RTLGRAPH(alpha,S,'delta');
plot(G)
```



```
n_delta = assign_mode_size(delta,2);  
n = merge_fields(n_alpha,n_delta);  
NR = init_net(m,n);  
net_view(NR)
```

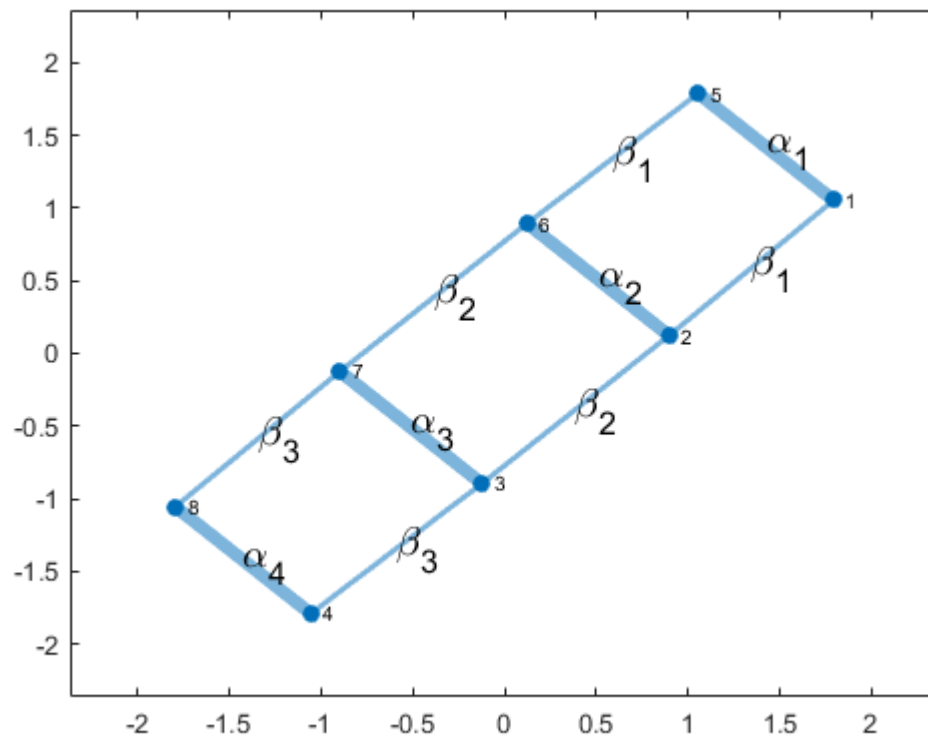



Computations between arbitrary plain networks

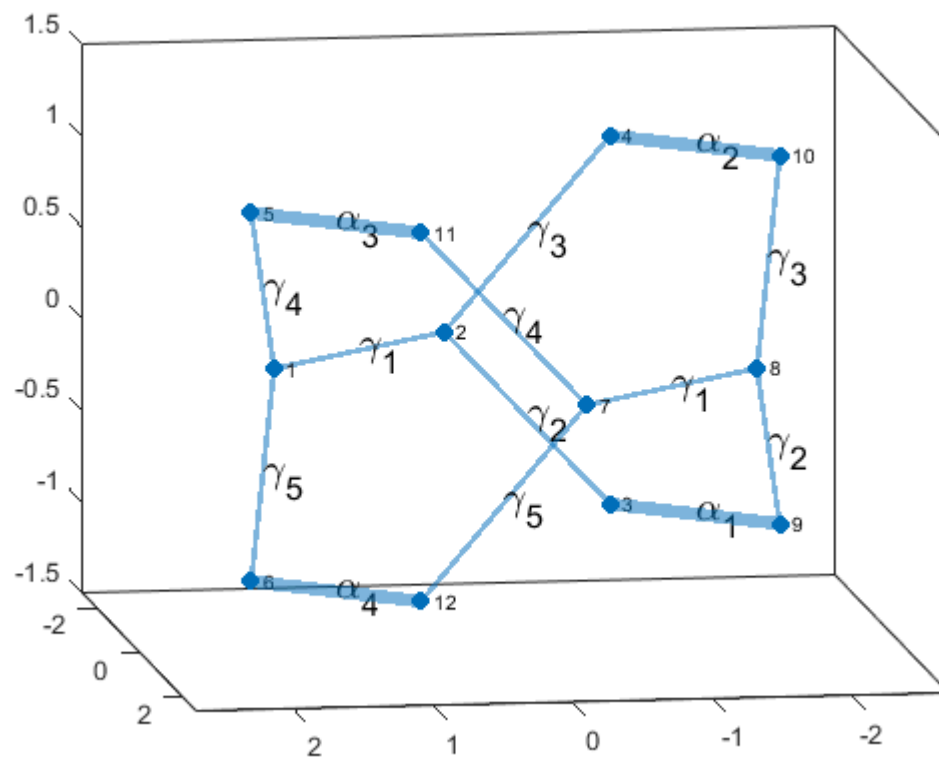
The function `boxtimes` allows us to do computations between these networks, even if we would have used the same inner mode for all three.

Their outer mode names tell `boxtimes` which edges are to be connected. The resulting networks can however become complicated.

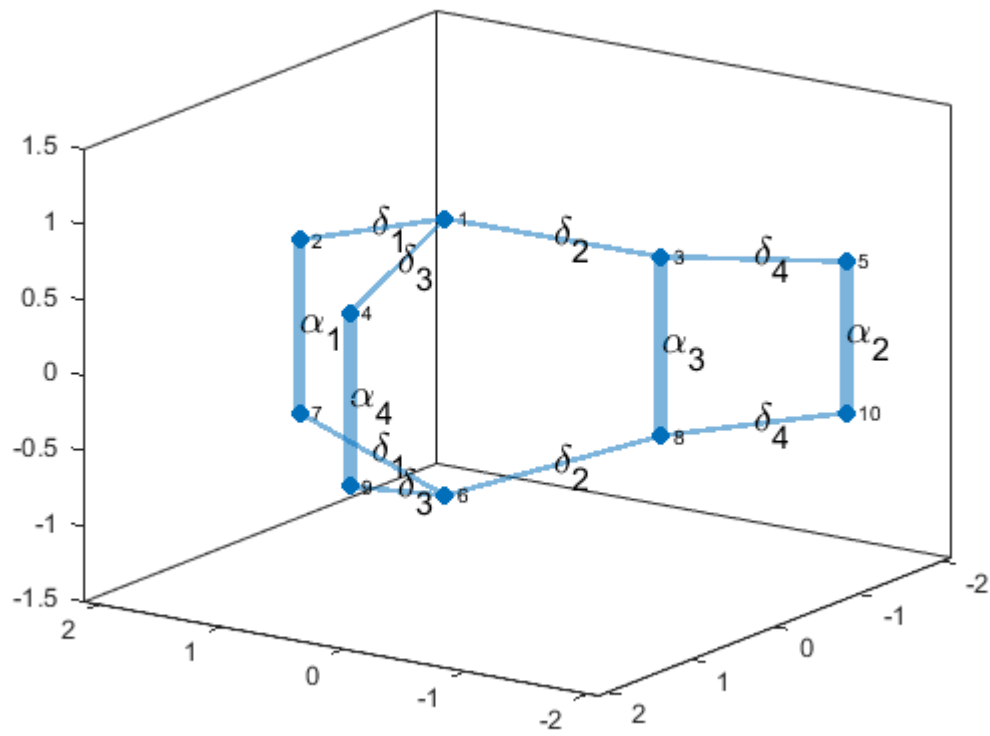
```
NTT = randomize_net(NTT);
NHT = randomize_net(NHT);
NR = randomize_net(NR);
save('random-format', 'NR');
net_view(NTT, NTT)
```



```
net_view(NHT,NHT,'Layout','force3')
view([171.30 12.40])
```

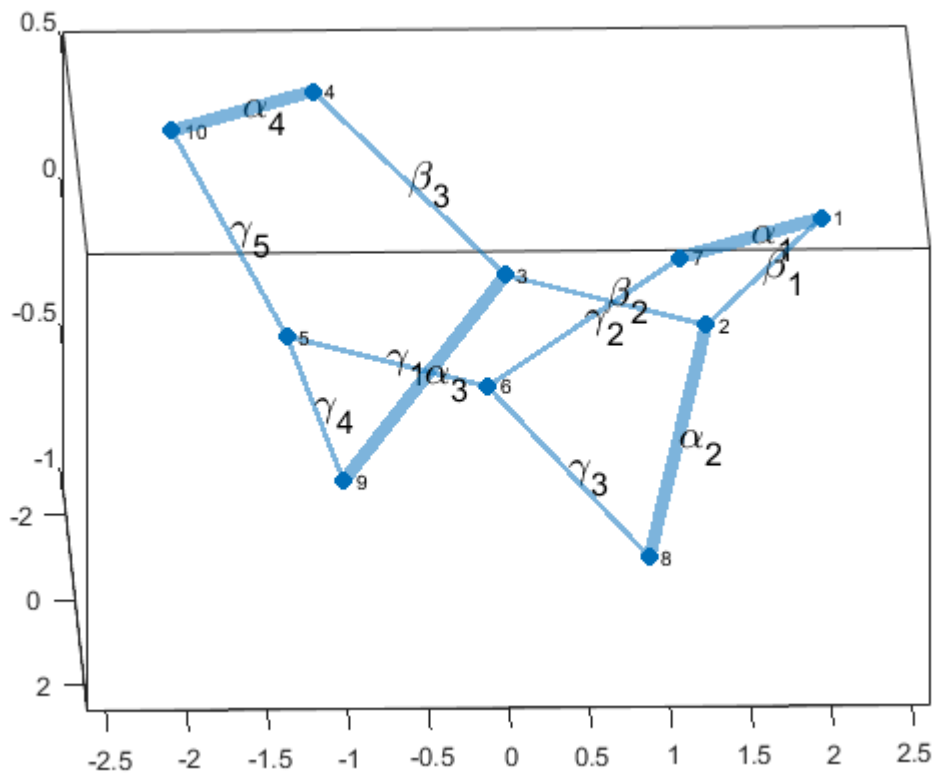


```
net_view(NR,NR,'Layout','force3')
view([-145.50 20.40])
```



The two networks do not necessarily need to be of the same type:

```
net_view(NTT,NHT,'Layout','force3')
view([1.70 -26.00])
```



Scalar products

We can either take scalar products between the tensor being represented, or measure their distance.

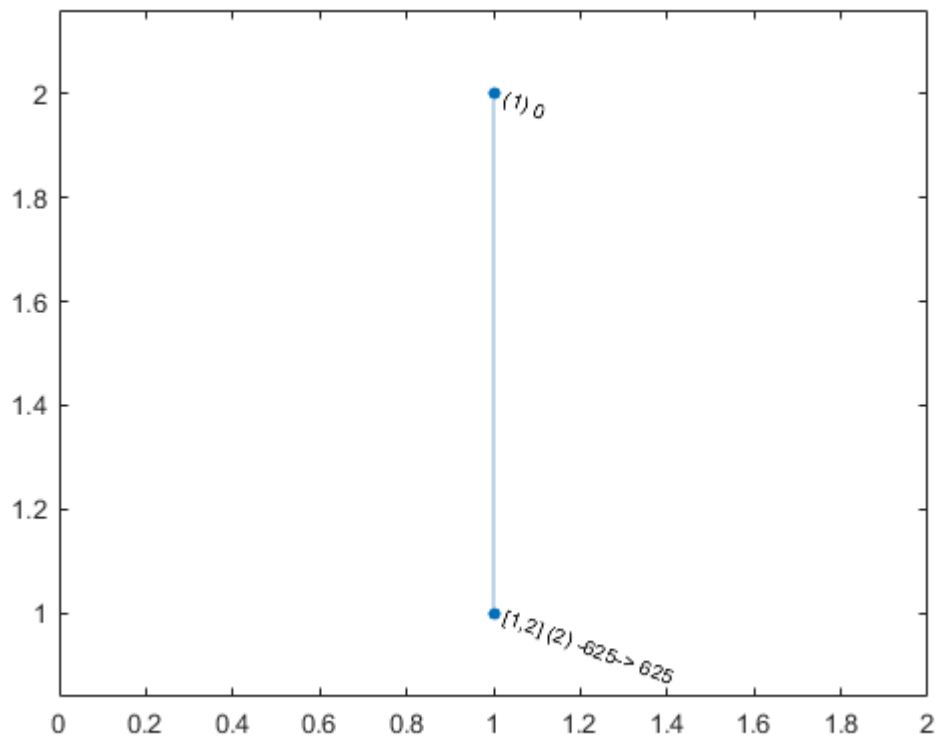
```
TTT = boxtimes(NTT)
```

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

```
THT = boxtimes(NHT)
```

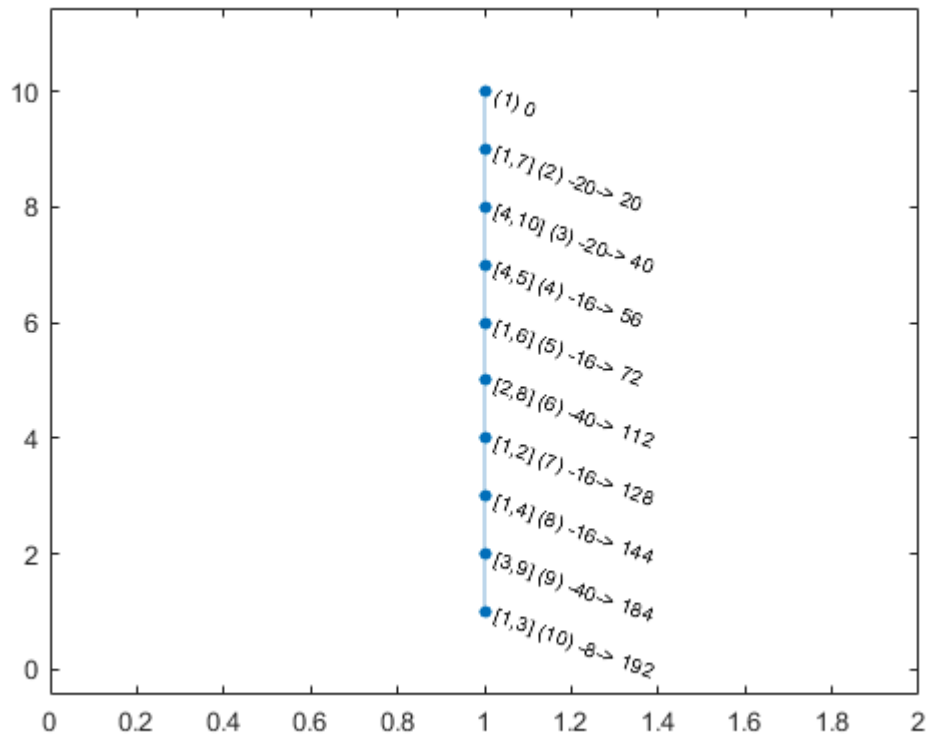
```
THT = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4'}
    pos: [1x1 struct]
    data: [5x5x5x5 double]
```

```
get_data(boxtimes(TTT,THT,'mode','show'))
```



```
Accum cost: 625, Iterations: 2
Contractions:
[ 1 <- 2] -      625 ->      625
Indices of nodes remaining: 1
ans = 0.0013
```

```
get_data(boxtimes(NTT,NHT,'mode','show'))
```



Accum cost: 192, Iterations: 10

Contractions:

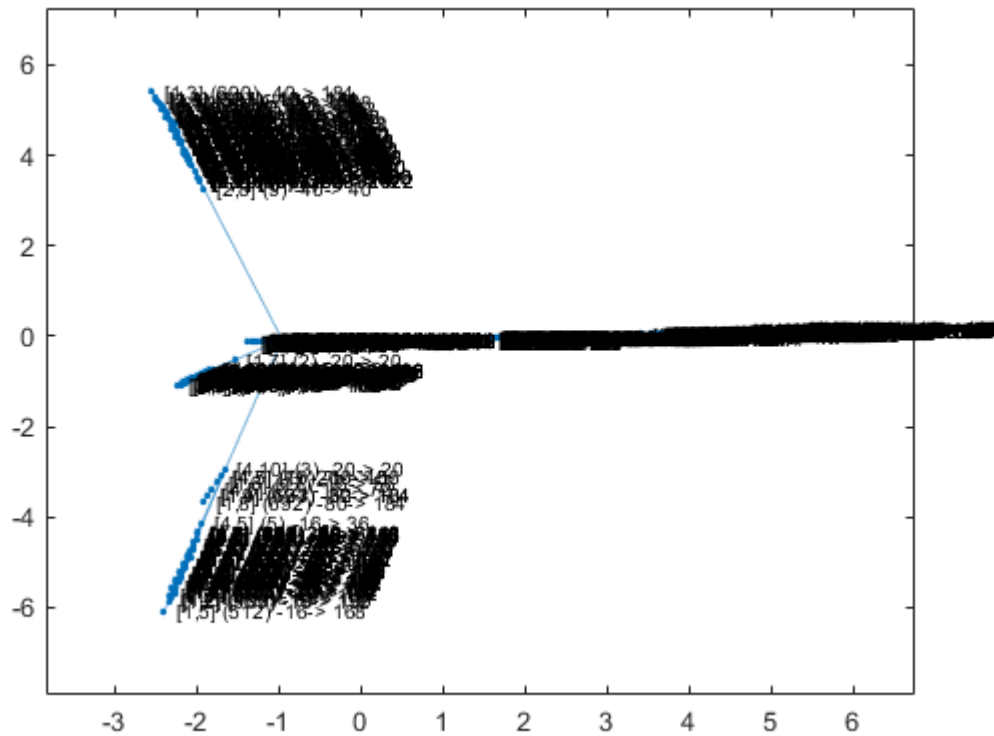
[1 <- 7] -	20 ->	20
[4 <- 10] -	20 ->	40
[4 <- 5] -	16 ->	56
[1 <- 6] -	16 ->	72
[2 <- 8] -	40 ->	112
[1 <- 2] -	16 ->	128
[1 <- 4] -	16 ->	144
[3 <- 9] -	40 ->	184
[1 <- 3] -	8 ->	192

Indices of nodes remaining: 1

ans = 0.0013

In this case, a search for the optimal contraction order will require too many iterations if we increase d .

```
try
  get_data(boxtimes(NTT,NHT,'mode','optimal_show'))
catch ME
  ME.message
end
```



Accum cost: 188, Iterations: 694

Contractions:

[3 <- 9]	-	40	->	40
[2 <- 8]	-	40	->	80
[4 <- 10]	-	20	->	100
[1 <- 7]	-	20	->	120
[4 <- 5]	-	16	->	136
[3 <- 4]	-	16	->	152
[3 <- 6]	-	16	->	168
[2 <- 3]	-	16	->	184
[1 <- 2]	-	4	->	188

Indices of nodes remaining: 1

```
ans = 0.0013
```

We can restrict us to solutions which in the worst case will be 5 times as complex. We might however be lucky:

```
try
  get_data(boxtimes(NTT,NHT, 'mode', 'optimal_show_optfac_0.2'))
catch ME
  ME.message
end
```


Accum cost: 194, Iterations: 38

Contractions:

[2 <- 8]	-	40	->	40
[1 <- 7]	-	20	->	60
[1 <- 2]	-	16	->	76
[1 <- 6]	-	16	->	92
[1 <- 5]	-	16	->	108
[4 <- 10]	-	20	->	128
[1 <- 4]	-	16	->	144
[1 <- 3]	-	40	->	184
[1 <- 9]	-	10	->	194

Indices of nodes remaining: 1

```
ans = 0.0013
```

Distances

The distance of two networks is defined as the distance of the tensors which they represent.

Since the addition of networks which are not of the same type is troublesome, but scalar products are usually not, `net dist` evaluates it using

$$\|A - B\|^2 = \langle A, A \rangle - 2\langle A, B \rangle + \langle B, B \rangle.$$

```
net_dist(NTT,NTT)
```

```
ans = 0
```

```
net dist (NHT, NTT)
```

```
ans = 0.3442
```

```
net_dist(THT,TTT)
```

```
ans = 0.3442
```

```
norm(unfold(TTT,alpha)-unfold(THT,alpha),'fro')
```

```
ans = 0.3442
```