

# Tensor Node Notation

## Common Representations

We now have enough tools to use tensor nodes as representations in the most common tensor formats, such as TT, Tucker, binary HT or CP-decompositions (and in fact any other network).

The rules of `boxtimes` together with the mode names of the nodes define these formats and any parts of such themselves.

```
rng(1)
alpha = mna('alpha',1:4);
beta = [mna('beta',1:4), 'beta_1_2', 'beta_3_4'];
n_alpha = assign_mode_size(alpha,10);
n_beta = assign_mode_size(beta,2);
n_gamma = assign_mode_size({'gamma'},2);
n = merge_fields(n_alpha,n_beta,n_gamma)
```

```
n = struct with fields:
    alpha_1: 10
    alpha_2: 10
    alpha_3: 10
    alpha_4: 10
    beta_1: 2
    beta_2: 2
    beta_3: 2
    beta_4: 2
    beta_1_2: 2
    beta_3_4: 2
    gamma: 2
```

## Tensor Train (or Matrix Products States)

As first example, we initialize a TT/MPS representations of a 4-dimensional tensor randomly.

```
G = cell(1,4);
G{1} = init_node([alpha(1),beta(1)],n);
G{2} = init_node([beta(1),alpha(2),beta(2)],n);
G{3} = init_node([beta(2),alpha(3),beta(3)],n);
G{4} = init_node([beta(3),alpha(4)],n);
G = randomize_net(G);
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:
```

```

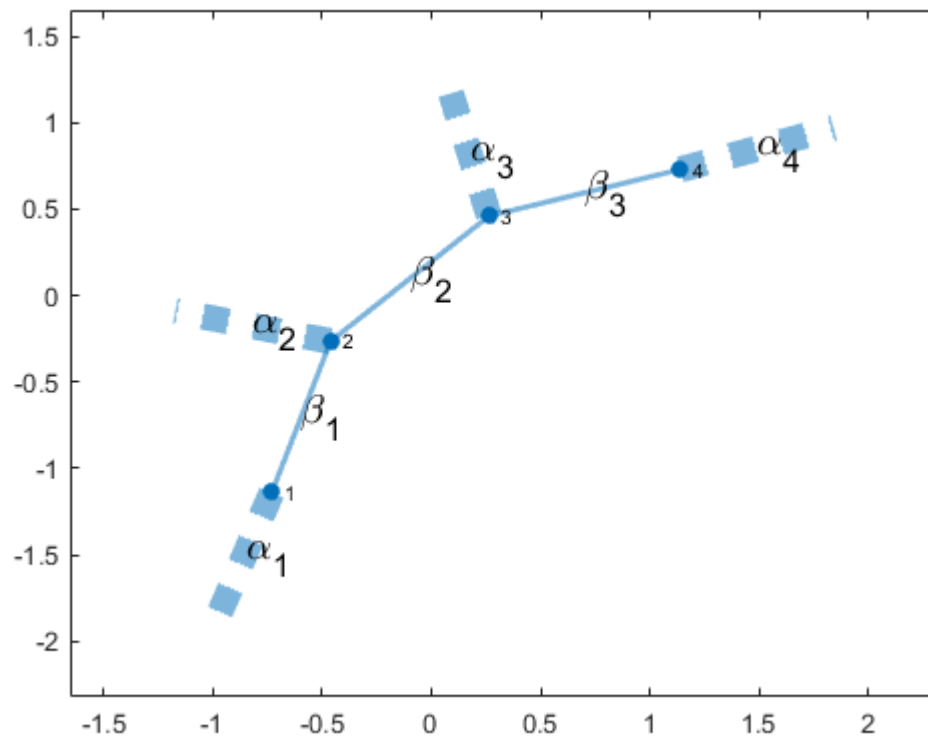
mode_names: {'beta_2' 'alpha_3' 'beta_3'}
pos: [1x1 struct]
data: [2x10x2 double]

ans = struct with fields:
mode_names: {'beta_3' 'alpha_4'}
pos: [1x1 struct]
data: [2x10 double]

```

The structure of the network behind this TT-representation is, as mentioned above, automatically determined by the mode names in the single nodes. Matlab might plot it in an unfamiliar way though.

```
net_view(G)
```



```
save('TT-format','G');
```

We can also calculate the full tensor. We should of course only enter this if dimension and mode sizes are not too large.

```
T = boxtimes(G)
```

```

T = struct with fields:
mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4'}
pos: [1x1 struct]
data: [10x10x10x10 double]

```

At this point we can compare the number of entries contained in both T and G.

```
data_volume(T)
```

```
ans = 10001
```

```
data_volume(G)
```

```
ans = 121
```

To access a single entry, we call

```
I = {6,4,2,3};  
T_I = node_part(T,alpha,I)
```

```
T_I = struct with fields:  
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4'}  
        pos: [1x1 struct]  
        data: -9.7210e-04
```

```
unfold(T_I,alpha)
```

```
ans = -9.7210e-04
```

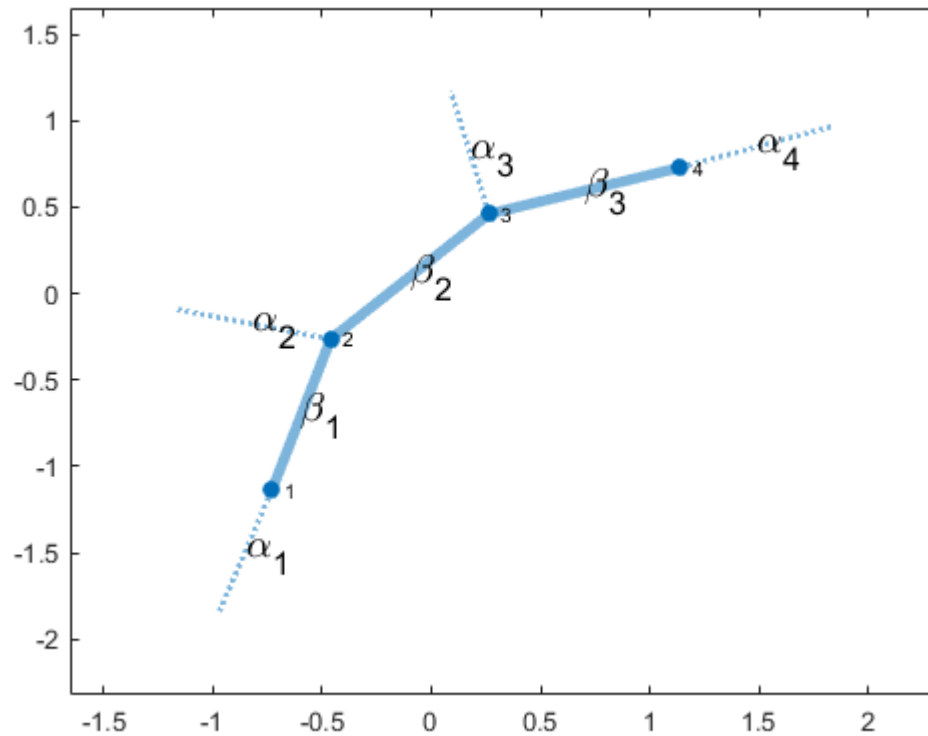
If we want to avoid the full tensor, we may as well call

```
G1_I1 = node_part(G{1},alpha{1},I{1});  
G2_I2 = node_part(G{2},alpha{2},I{2});  
G3_I3 = node_part(G{3},alpha{3},I{3});  
G4_I4 = node_part(G{4},alpha{4},I{4});  
T_I = boxtimes(G1_I1,G2_I2,G3_I3,G4_I4);  
unfold(T_I,alpha)
```

```
ans = -9.7210e-04
```

Or we restrict the entire representation and then call `boxtimes` (this does not construct the full tensor)

```
G_I = node_part(G,alpha,I);  
net_view(G_I) % the legs are smaller now
```



```
boxtimes(G_I) % is still a tensor node
```

```
ans = struct with fields:
  mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4'}
  pos: [1x1 struct]
  data: -9.7210e-04
```

## Tucker

In the case of the Tucker format, the different  $[X]$  in the product

$$T = C [X] U_1 [X] U_2 [X] U_3 [X] U_4$$

are, conventionally, often denoted  $x_1, \dots, x_4$ .

```
C = init_node(beta(1:4),n);
U = cell(1,4);
for i = 1:4
    U{i} = init_node([alpha(i),beta(i)],n);
end
C = randomize_node(C)
```

```
C = struct with fields:
  mode_names: {'beta_1' 'beta_2' 'beta_3' 'beta_4'}
  pos: [1x1 struct]
```

```
data: [2×2×2×2 double]
```

```
U = randomize_net(U);  
U{:}
```

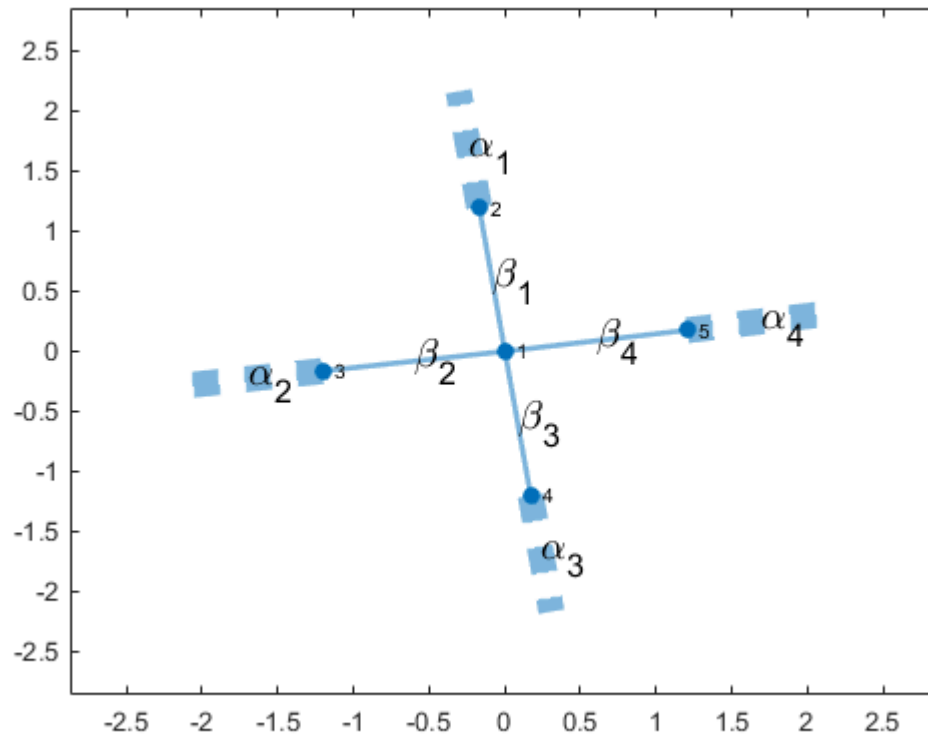
```
ans = struct with fields:  
    mode_names: {'alpha_1' 'beta_1'}  
        pos: [1×1 struct]  
        data: [10×2 double]  
  
ans = struct with fields:  
    mode_names: {'alpha_2' 'beta_2'}  
        pos: [1×1 struct]  
        data: [10×2 double]  
  
ans = struct with fields:  
    mode_names: {'alpha_3' 'beta_3'}  
        pos: [1×1 struct]  
        data: [10×2 double]  
  
ans = struct with fields:  
    mode_names: {'alpha_4' 'beta_4'}  
        pos: [1×1 struct]  
        data: [10×2 double]
```

The resulting network is again only depends on the mode names.

```
T = boxtimes(C,U)
```

```
T = struct with fields:  
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4'}  
        pos: [1×1 struct]  
        data: [10×10×10×10 double]
```

```
net_view(C,U{:})
```



```
save('Tucker-format', 'C', 'U')
```

## Binary HT-format

In case of the binary HT-format, we have to decide whether to actually use a root transfer tensor, or not. For now, we keep it, but in later steps we will omit for reasons which will then become clear..

```
B12 = init_node(beta([1,2,5]),n);
B34 = init_node(beta([3,4,6]),n);
B12 = randomize_node(B12)
```

```
B12 = struct with fields:
    mode_names: {'beta_1' 'beta_2' 'beta_1_2'}
    pos: [1x1 struct]
    data: [2x2x2 double]
```

```
B34 = randomize_node(B34)
```

```
B34 = struct with fields:
    mode_names: {'beta_3' 'beta_4' 'beta_3_4'}
    pos: [1x1 struct]
    data: [2x2x2 double]
```

```
root_transfer_tensor = init_node(beta(5:6),n);
root_transfer_tensor = randomize_node(root_transfer_tensor)
```

```

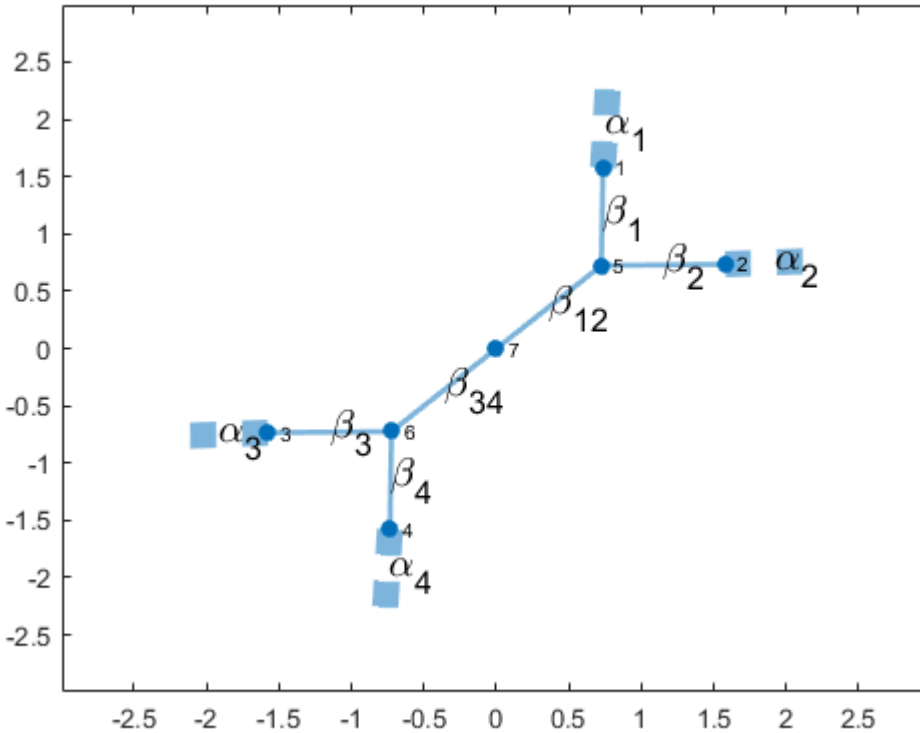
root_transfer_tensor = struct with fields:
    mode_names: {'beta_1_2' 'beta_3_4'}
    pos: [1×1 struct]
    data: [2×2 double]

```

```

save('binHT','U','B12','B34');
net_view(U,B12,B34,root_transfer_tensor)

```



## CP

This is the only case in which the multiplication of the single tensor nodes may not be performed in smaller steps, since all nodes share one common mode name.

```

Phi = cell(1,4);
for i = 1:4
    Phi{i} = init_node([alpha(i), 'gamma'],n);
end
Phi = randomize_net(Phi);
Phi{:}

```

```

ans = struct with fields:
    mode_names: {'alpha_1' 'gamma'}
    pos: [1×1 struct]
    data: [10×2 double]
ans = struct with fields:
    mode_names: {'alpha_2' 'gamma'}

```

```

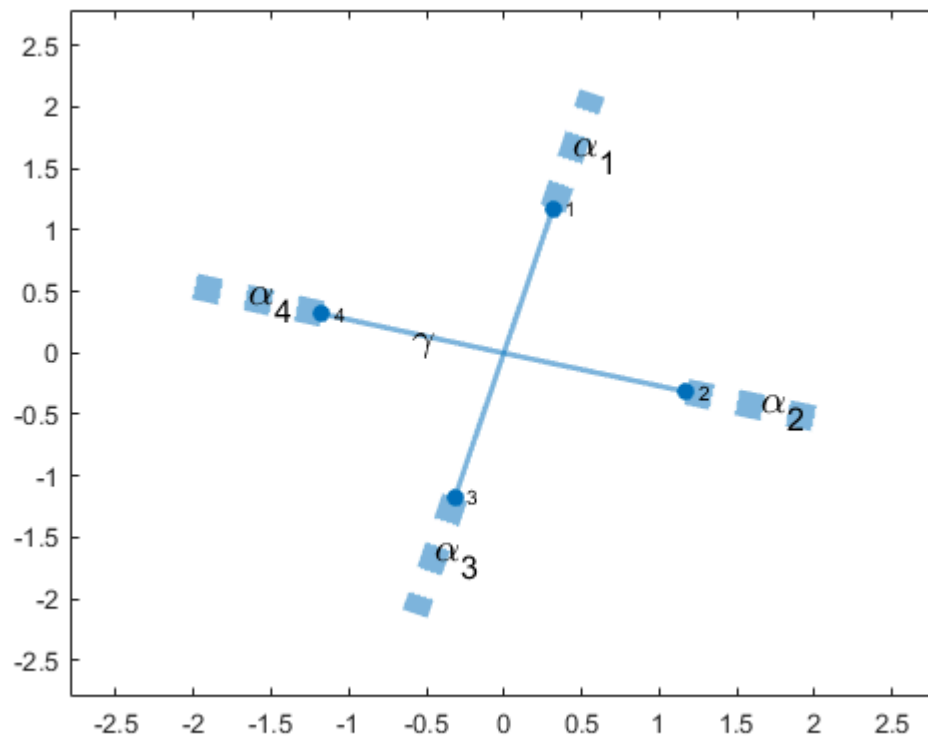
pos: [1×1 struct]
data: [10×2 double]
ans = struct with fields:
    mode_names: {'alpha_3' 'gamma'}
    pos: [1×1 struct]
    data: [10×2 double]
ans = struct with fields:
    mode_names: {'alpha_4' 'gamma'}
    pos: [1×1 struct]
    data: [10×2 double]

```

```

save('CP','Phi')
net_view(Phi)

```



## PEPS

Formats with loops are possible as well

```

N = 4;
alpha = reshape(mna('alpha',1:N^2),[N,N]);
beta = reshape(mna('beta',1:N^2),[N,N]);
gamma = reshape(mna('gamma',1:N^2),[N,N]);
n_alpha = assign_mode_size(alpha,2);
n_beta = assign_mode_size(beta,2);
n_gamma = assign_mode_size(gamma,2);
n = merge_fields(n_alpha,n_beta,n_gamma);

```

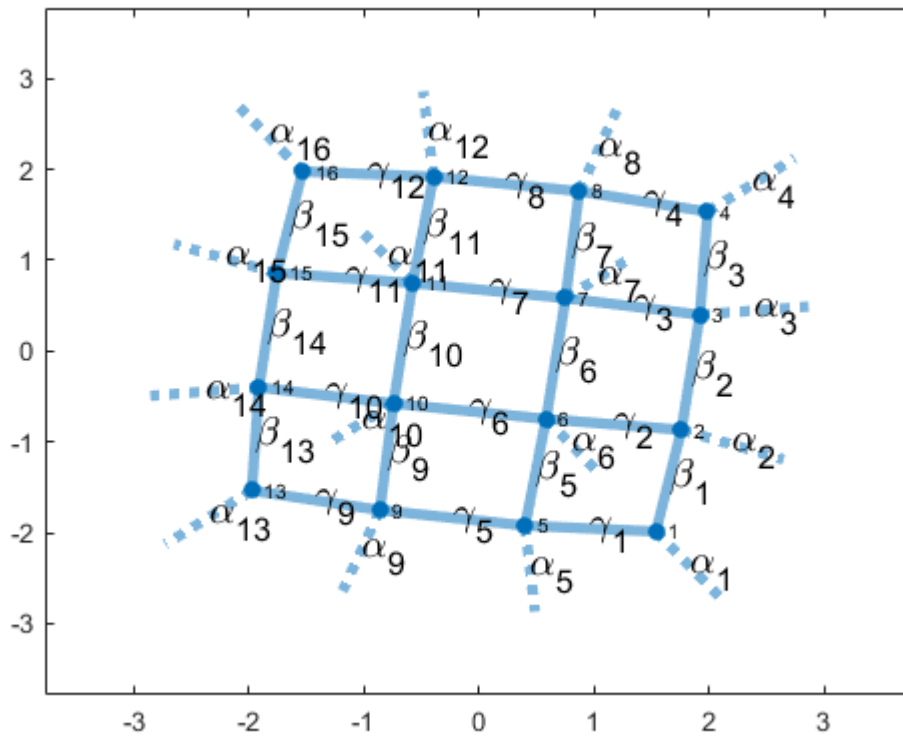


```

P = cell(N,N);
for i = 1:N
    for j = 1:N
        mn = {alpha{i,j}};
        if i < N
            mn = [mn,beta{i,j}];
        end
        if i > 1
            mn = [mn,beta{i-1,j}];
        end
        if j < N
            mn = [mn,gamma{i,j}];
        end
        if j > 1
            mn = [mn,gamma{i,j-1}];
        end

        P{i,j} = init_node(mn,n);
    end
end
% P = P(:);
net_view(P); %,'Layout','force3') % might not look as nice as used to

```



```
save('PEPS','P')
```

Contracting this network for anything but mode size 2 would likely exceed memory capacities:

```
TP = boxtimes(P)
```

```
TP = struct with fields:
```

```
  mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4' 'alpha_5' 'alpha_6' 'alpha_7' 'alpha_8'}  
      pos: [1×1 struct]  
      data: [16-D double]
```

```
numel(TP.data) % = 2^16
```

```
ans = 65536
```