

Tensor Node Notation

Nested boxtimes

Determining the order of contractions

Once we have any mode name appearing more than twice in a $[\times]$ -product, we may not simply resolve brackets. For example for N_1, \dots, N_4 as follows,

$$(N_1 [\times] N_2)^T [\times] (N_3 [\times] N_4) \neq [\times] (N_2^T, N_1^T, N_3, N_4).$$

The result may be rather unintended.

```
n = assign_mode_size({'alpha', 'beta', 'gamma'}, [4, 3, 2]);
N1 = init_node({'alpha', 'beta'}, n);
N2 = init_node({'beta', 'gamma'}, n);
N3 = N1;
N4 = N2;
N1 = randomize_node(N1)
```

```
N1 = struct with fields:
    mode_names: {'alpha' 'beta'}
    pos: [1×1 struct]
    data: [4×3 double]
```

```
N2 = randomize_node(N2)
```

```
N2 = struct with fields:
    mode_names: {'beta' 'gamma'}
    pos: [1×1 struct]
    data: [3×2 double]
```

```
N3 = randomize_node(N3)
```

```
N3 = struct with fields:
    mode_names: {'alpha' 'beta'}
    pos: [1×1 struct]
    data: [4×3 double]
```

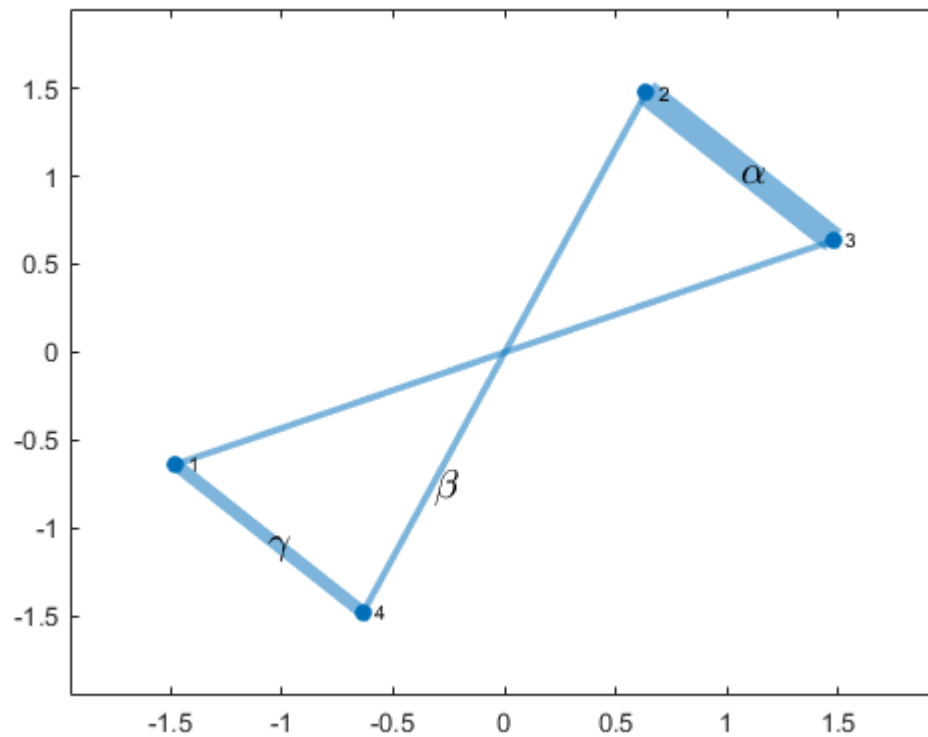
```
N4 = randomize_node(N4)
```

```
N4 = struct with fields:
    mode_names: {'beta' 'gamma'}
    pos: [1×1 struct]
    data: [3×2 double]
```

```
w = boxtimes(node_transpose(N2), node_transpose(N1), N3, N4);
w.data
```

```
ans = 0.1703
```

```
net_view(node_transpose(N2),node_transpose(N1),N3,N4)
```



But instead of performing three single $[\times]$ -products, we can transfer the bracket structure as before to one single `boxtimes` in form of nested cells. `boxtimes` will never call itself!

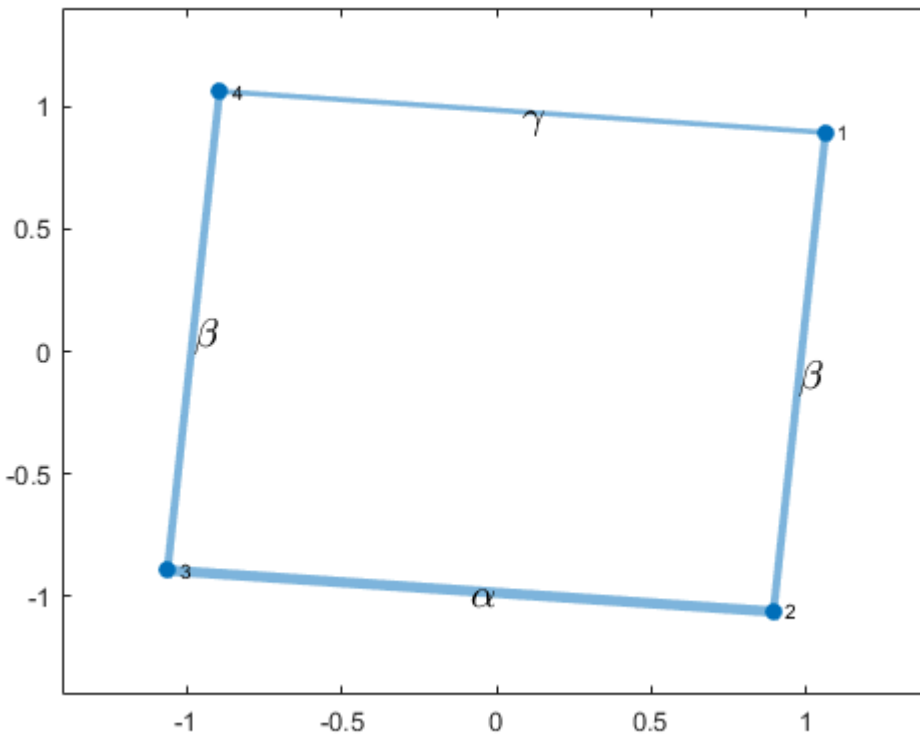
```
r1 = boxtimes(node_transpose(boxtimes(N1,N2)),boxtimes(N3,N4));
r1.data
```

```
ans = 0.0183
```

```
r2 = boxtimes(net_transpose({N1,N2}},{N3,N4});
r2.data
```

```
ans = 0.0183
```

```
net_view(net_transpose({N1,N2}},{N3,N4})
```

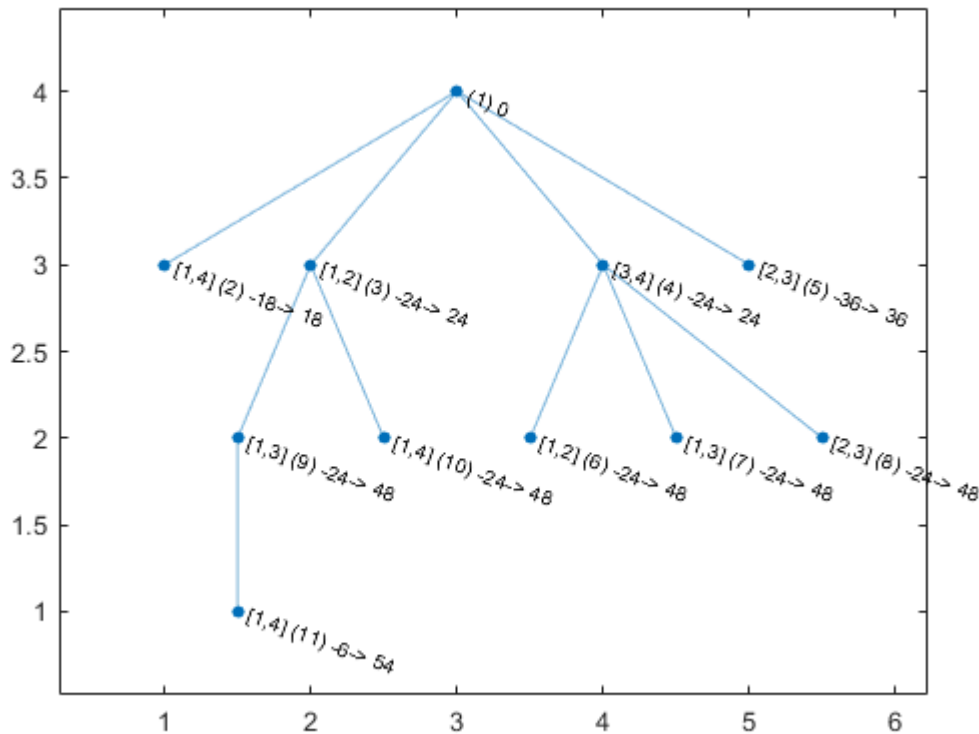


In the following call we can see that, interestingly, the optimal order of contractions (in terms of complexity) is not the one prescribed by the original brackets $(N_1 \times N_2)^T \times (N_3 \times N_4)$. So `boxtimes` indeed does not call itself, but uses the nested input solely to determine contraction rules.

```
a = n.alpha; b = n.beta; c = n.gamma;
cost_as_by_brackets = (a*b*c) + (a*b*c) + (a*c)
```

```
cost_as_by_brackets = 56
```

```
boxtimes(net_transpose({N1,N2}},{N3,N4},'mode','optimal_show')
```



```

Accum cost: 54, Iterations: 11
Contractions:
[ 1 <- 2] -      24 ->      24
[ 1 <- 3] -      24 ->      48
[ 1 <- 4] -        6 ->      54
Indices of nodes remaining: 1
ans = struct with fields:
    mode_names: {1x0 cell}
        pos: [1x1 struct]
        data: 0.0183

```

The difference is tiny for now, but points at a central idea behind tensor networks.

Scalar products between different tensor formats

This mechanic allows us to be more flexible when for example taking the scalar product between two different tensor formats in which we used β as mode names in both cases.

```

load('TT-format')
boxtimes(G)

```

```

ans = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4'}
        pos: [1x1 struct]
        data: [10x10x10x10 double]

```

```

load('Tucker-format')

```

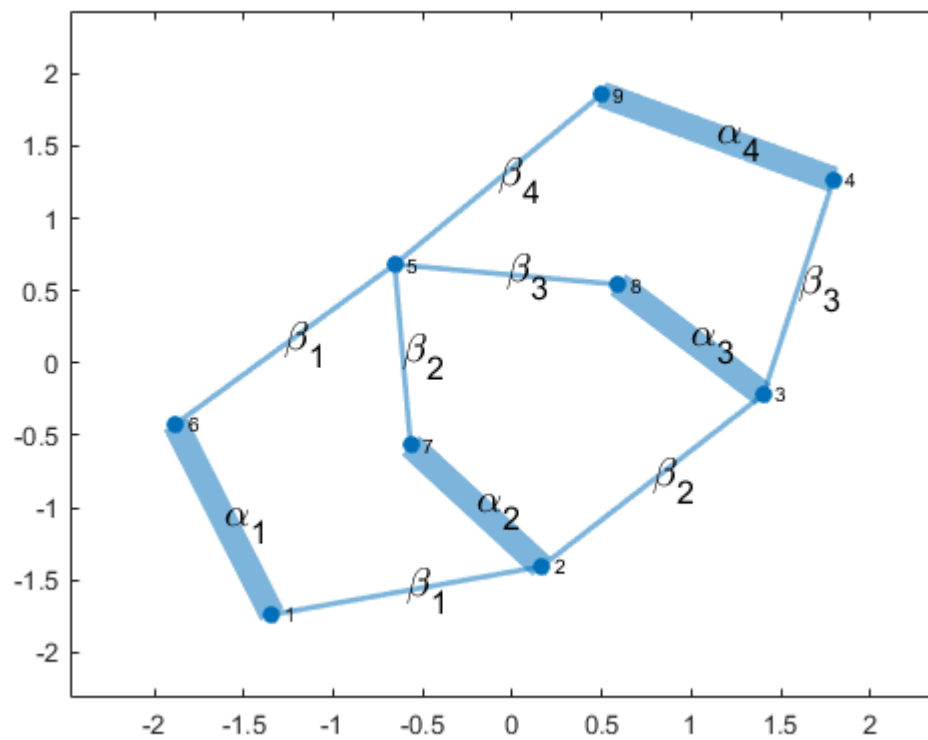
```
boxtimes(G,{C,U})
```

```
ans = struct with fields:
    mode_names: {1x0 cell}
    pos: [1x1 struct]
    data: 1.3561e-04
```

```
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]
```

```
net_view(G,{C,U}) % node numbers for G: 1-4, C: 5, U: 6-9
```



Here, the order of contractions is crucial. If we would first evaluate $[\times](G)$, then let alone the resulting tensor has size 1000. However, the optimal order of multiplication just requires an order of complexity of 308 (calculated classically) and might not be the order of contractions one expects at a first glance.

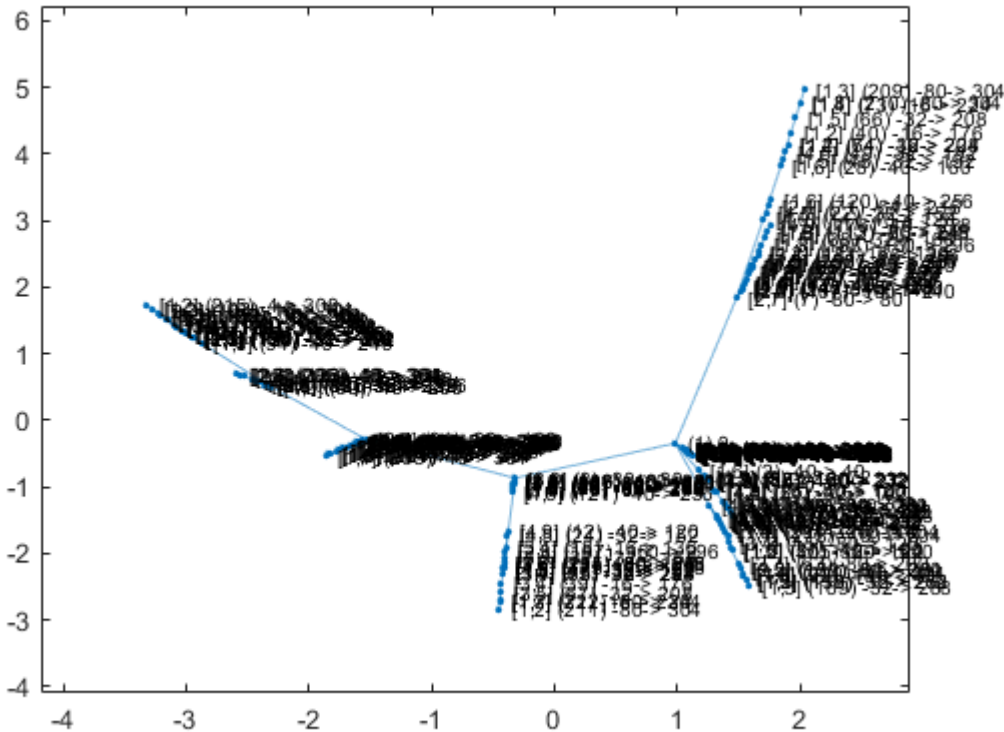
```
node_size(G)
```

```
ans = struct with fields:
    alpha_1: 10
    alpha_2: 10
    alpha_3: 10
    alpha_4: 10
    beta_1: 2
    beta_2: 2
    beta_3: 2
```

```
node_size(C)
```

```
ans = struct with fields:
    beta_1: 2
    beta_2: 2
    beta_3: 2
    beta_4: 2
```

```
boxtimes(G, {C,U}, 'mode', 'optimal_show')
```



Accum cost: 308, Iterations: 215
 Contractions:
 [3 <- 8] - 80 -> 80
 [2 <- 7] - 80 -> 160

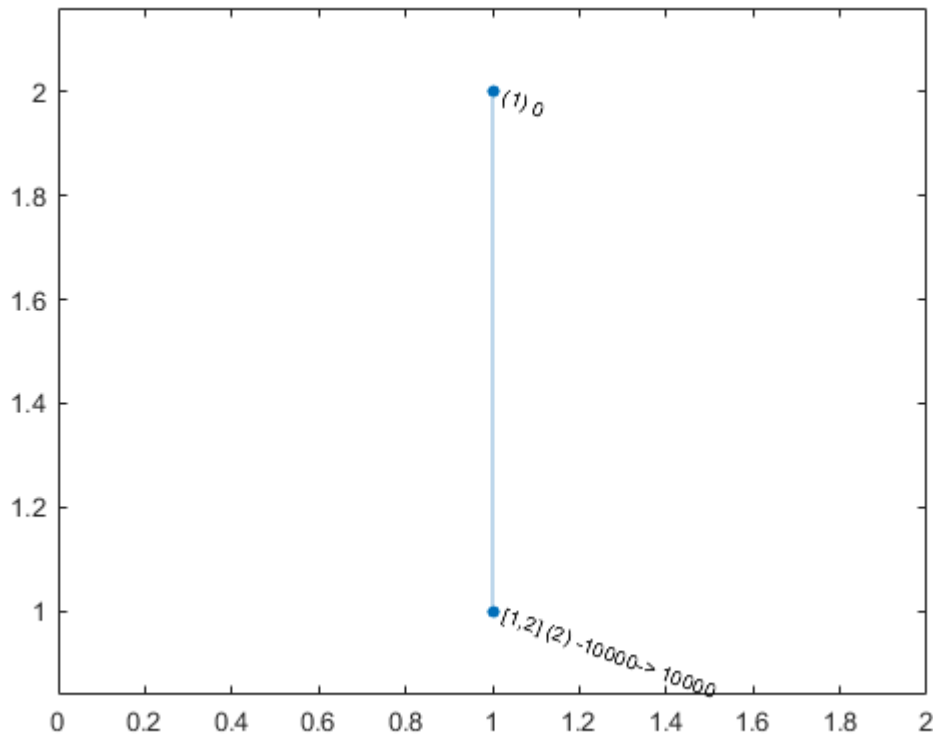
```
[ 4 <- 9] -      40 ->      200
[ 1 <- 6] -      40 ->      240
[ 3 <- 4] -      16 ->      256
[ 3 <- 5] -      32 ->      288
[ 2 <- 3] -      16 ->      304
[ 1 <- 2] -       4 ->      308
```

Indices of nodes remaining: 1

```
ans = struct with fields:
```

```
mode_names: {1×0 cell}
pos: [1×1 struct]
data: 1.3561e-04
```

```
boxtimes(boxtimes(G),boxtimes(C,U),'mode','optimal_show')
```



Accum cost: 10000, Iterations: 2

Contractions:

```
[ 1 <- 2] -      10000 ->      10000
```

Indices of nodes remaining: 1

```
ans = struct with fields:
```

```
mode_names: {1×0 cell}
pos: [1×1 struct]
data: 1.3561e-04
```

Boxtimes memory

Once this order of computations is found, we may of course reuse this insight for the next analogous multiplication.

```
activate_boxtimes_mem()
clear_boxtimes_mem()
```

```
tic
boxtimes(G,{C,U},'mode','optimal');
toc
```

Elapsed time is 0.308307 seconds.

Once the boxtimes memory is activated, the required cpu time may reduce considerably:

```
tic
boxtimes(G,{C,U},'mode','optimal');
toc
```

Elapsed time is 0.033543 seconds.

The memory of boxtimes is smart, but does not overthink, since its main purpose is to avoid repeated calculations in loops:

So while it realizes that changing the order of mode names in the single node *C* does not change it (since it has no duplicate mode names)...

```
CT = node_transpose(C)
```

```
CT = struct with fields:
    mode_names: {'beta_4' 'beta_3' 'beta_2' 'beta_1'}
        pos: [1x1 struct]
        data: [2x2x2x2 double]
```

```
tic
boxtimes(G,{CT,U},'mode','optimal');
toc
```

Elapsed time is 0.008410 seconds.

...it does not realize that transposing $\{C,U\}$ makes no difference either.

```
{C,U}
```

```
ans = 1x2 cell array
    {1x1 struct}    {1x4 cell}
```

```
CUT = net_transpose({C,U})
```

```
CUT = 1x2 cell array
    {1x4 cell}    {1x1 struct}
```

```
tic
boxtimes(CUT,G,'mode','optimal');
toc
```


Elapsed time is 0.216938 seconds.

```
deactivate_boxtimes_mem()
```