

Tensor Node Notation

Multiplication of Nodes

Product of two nodes

The $[\times]$ -product is used to multiply two or several nodes and is the central tool of this toolbox. There are several functionalities build into the command `boxtimes`, of which the simplest one is the multiplication of two nodes. In the subsequent parts, we will demonstrate the rules which `boxtimes` follows depending on the mode names of nodes.

We first fix mode sizes and mode names:

```
alpha = mna('alpha',1:3)
```

```
alpha = 1×3 cell array
    {'alpha_1'}    {'alpha_2'}    {'alpha_3'}
```

```
beta = mna('beta',1:3)
```

```
beta = 1×3 cell array
    {'beta_1'}    {'beta_2'}    {'beta_3'}
```

```
n_alpha = assign_mode_size(alpha,[10,20,30])
```

```
n_alpha = struct with fields:
    alpha_1: 10
    alpha_2: 20
    alpha_3: 30
```

```
n_beta = assign_mode_size(beta,[2,3,4])
```

```
n_beta = struct with fields:
    beta_1: 2
    beta_2: 3
    beta_3: 4
```

```
n = merge_fields(n_alpha,n_beta)
```

```
n = struct with fields:
    alpha_1: 10
    alpha_2: 20
    alpha_3: 30
    beta_1: 2
    beta_2: 3
    beta_3: 4
```

Next, we initialize two nodes:

```
N1 = init_node([alpha(1),beta(1)],n)
```

```
N1 = struct with fields:
    mode_names: {'alpha_1' 'beta_1'}
    pos: [1×1 struct]
    data: [10×2 double]
```

```
N1 = randomize_node(N1);
N2 = init_node([beta(1),alpha(2)],n)
```

```
N2 = struct with fields:
    mode_names: {'beta_1' 'alpha_2'}
    pos: [1×1 struct]
    data: [2×20 double]
```

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

The multiplication of these two nodes will lead to a contraction of the common mode β_1 .

```
N1xN2 = boxtimes(N1,N2)
```

```
N1xN2 = struct with fields:
    mode_names: {'alpha_1' 'alpha_2'}
    pos: [1×1 struct]
    data: [10×20 double]
```

```
N2xN1 = boxtimes(N2,N1)
```

```
N2xN1 = struct with fields:
    mode_names: {'alpha_2' 'alpha_1'}
    pos: [1×1 struct]
    data: [20×10 double]
```

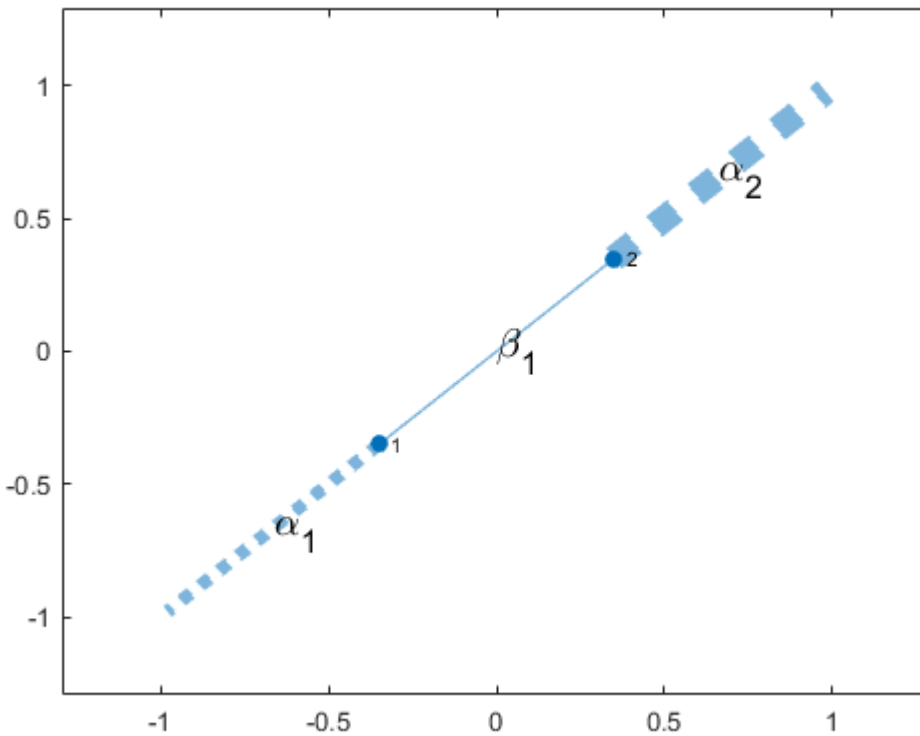
The field data in $N1 \times N2$ and $N2 \times N1$ is permuted, but as tensor nodes, they are identical. That means, if we unfold them with the same arguments, we get the same matrix:

```
norm(unfold(N1xN2,alpha(1:2))-unfold(N2xN1,alpha(1:2)),'fro')
```

```
ans = 0
```

The upper product $N_1 \times N_2$ can be also visualized:

```
figure; net_view(N1,N2)
```



We can as well do the same calculation manually using `unfold` and `fold`.

```
A1 = unfold(N1,alpha(1),beta(1));
size(A1)
```

```
ans = 1x2
      10      2
```

```
A2 = unfold(N2,beta(1),alpha(2));
size(A2)
```

```
ans = 1x2
      2      20
```

```
A1A2 = A1*A2;
N1xN2_man = fold(A1A2,alpha(1:2),n)
```

```
N1xN2_man = struct with fields:
    mode_names: {'alpha_1' 'alpha_2'}
        pos: [1x1 struct]
        data: [10x20 double]
```

```
norm(unfold(N1xN2_man,alpha(1:2))-unfold(N1xN2,alpha(1:2)),'fro')
```

```
ans = 0
```

Mode names shared by more than one node

If more than two nodes are involved, multiplication of these will again lead to a contraction of all common mode names.

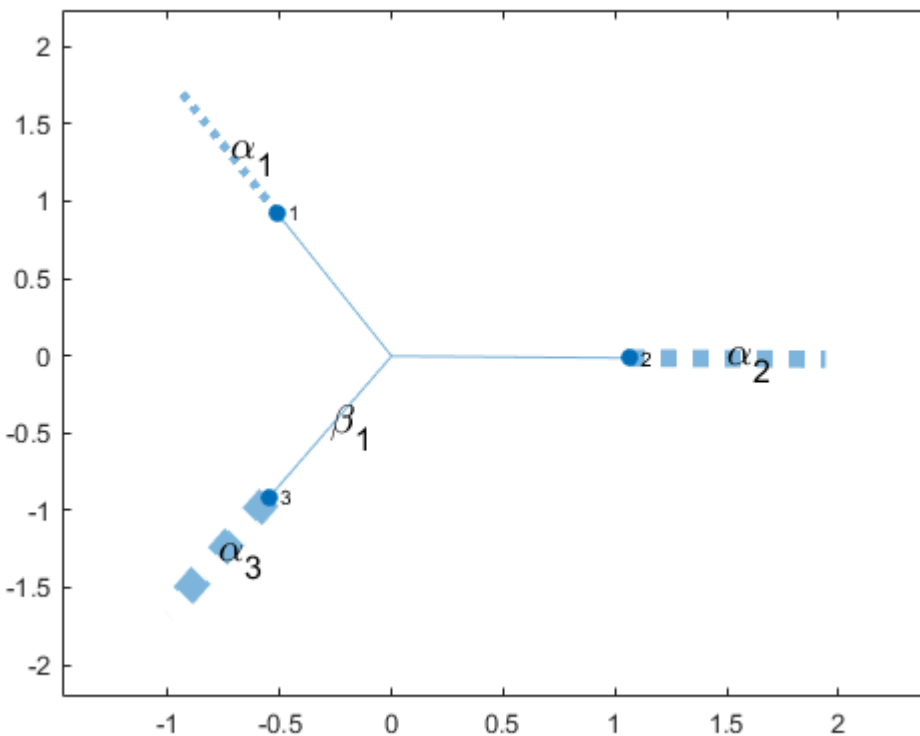
```
N3 = init_node([beta(1),alpha(3)],n)
```

```
N3 = struct with fields:
  mode_names: {'beta_1' 'alpha_3'}
  pos: [1x1 struct]
  data: [2x30 double]
```

```
N1N2N3 = boxtimes(N1,N2,N3)
```

```
N1N2N3 = struct with fields:
  mode_names: {'alpha_1' 'alpha_2' 'alpha_3'}
  pos: [1x1 struct]
  data: [10x20x30 double]
```

```
figure; net_view(N1,N2,N3)
```



In this case, the product can not that simply be split into two multiplications.

```
boxtimes(boxtimes(N1,N2),N3)
```

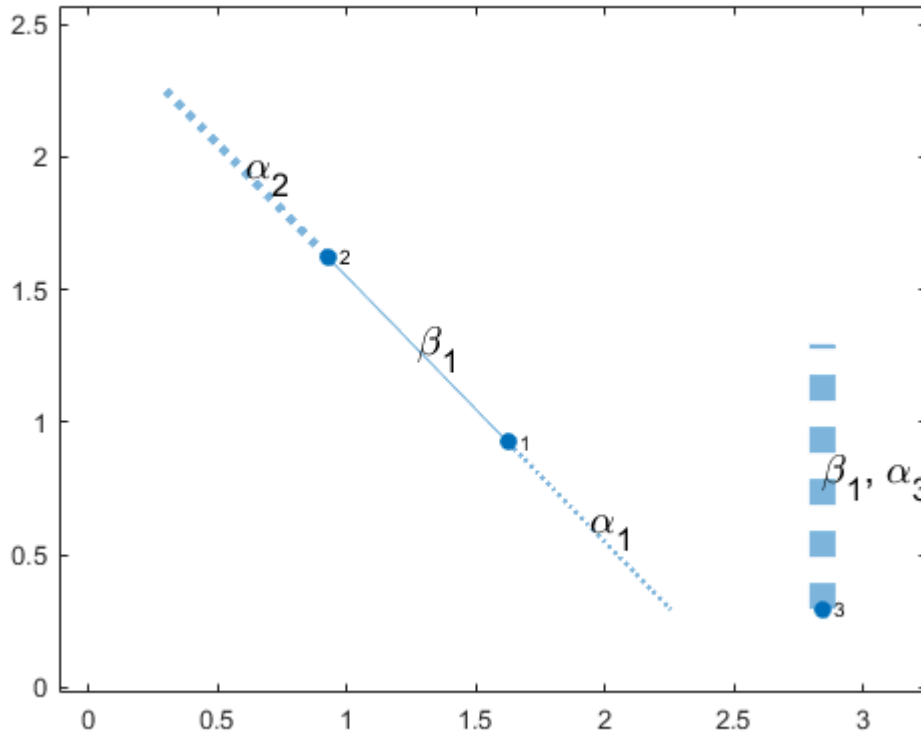
```
ans = struct with fields:
```

```

mode_names: {'alpha_1' 'alpha_2' 'beta_1' 'alpha_3'}
pos: [1x1 struct]
data: [10x20x2x30 double]

```

```
figure; net_view({N1,N2},N3)
```



So in the first multiplication, we have to tell boxtimes to keep β_1 , such that we can split

$$[\times](N_1, N_2, N_3) = (N_1 [\times]^{\beta_1} N_2) [\times] N_3$$

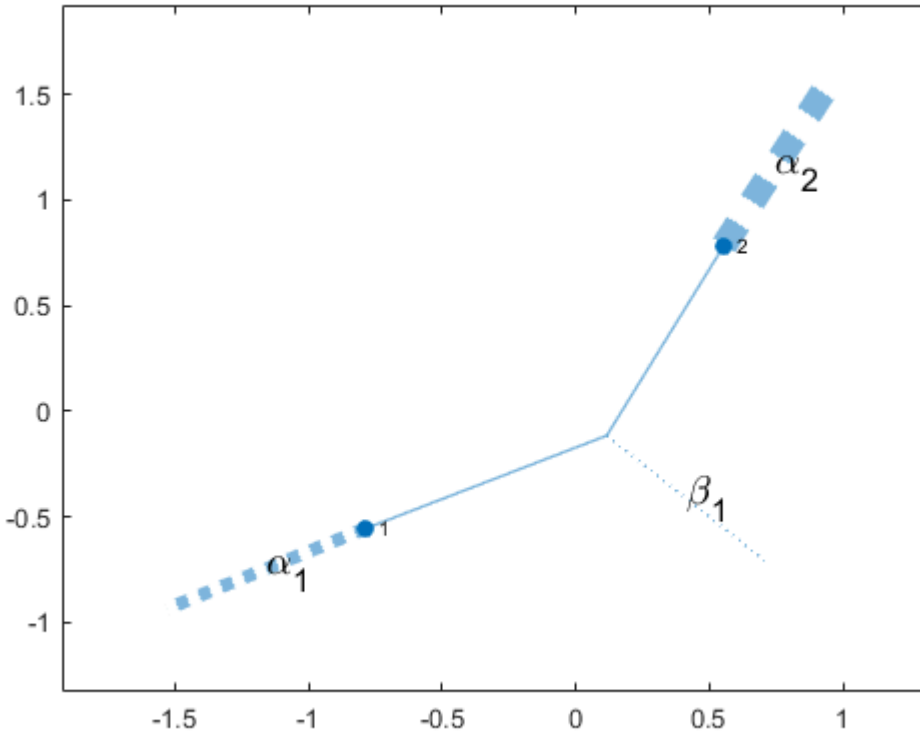
```
N1x_keepbeta1_N2 = boxtimes(N1,N2,'^',beta{1})
```

```

N1x_keepbeta1_N2 = struct with fields:
    mode_names: {'alpha_1' 'beta_1' 'alpha_2'}
    pos: [1x1 struct]
    data: [10x2x20 double]

```

```
figure; net_view(N1,N2,'^',beta{1})
```



This can manually be done with a for loop (the implementation uses `multiprod`):

```
A1 = unfold(N1,alpha(1),beta(1));
A2 = unfold(N2,alpha(2),beta(1));
B = zeros(n.(alpha{1}),n.(alpha{2}),n.(beta{1}));
for i = 1:n.(beta{1})
    B(:, :, i) = A1(:, i)*A2(:, i)';
end
N1x_keepbeta1_N2_ = fold(B,[alpha(1:2),beta(1)],n)
```

```
N1x_keepbeta1_N2_ = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'beta_1'}
    pos: [1x1 struct]
    data: [10x20x2 double]
```

```
norm(unfold(N1x_keepbeta1_N2_,alpha(1:2),beta(1))-unfold(N1x_keepbeta1_N2,alpha(1:2),beta(1)))
```

```
ans = 0
```

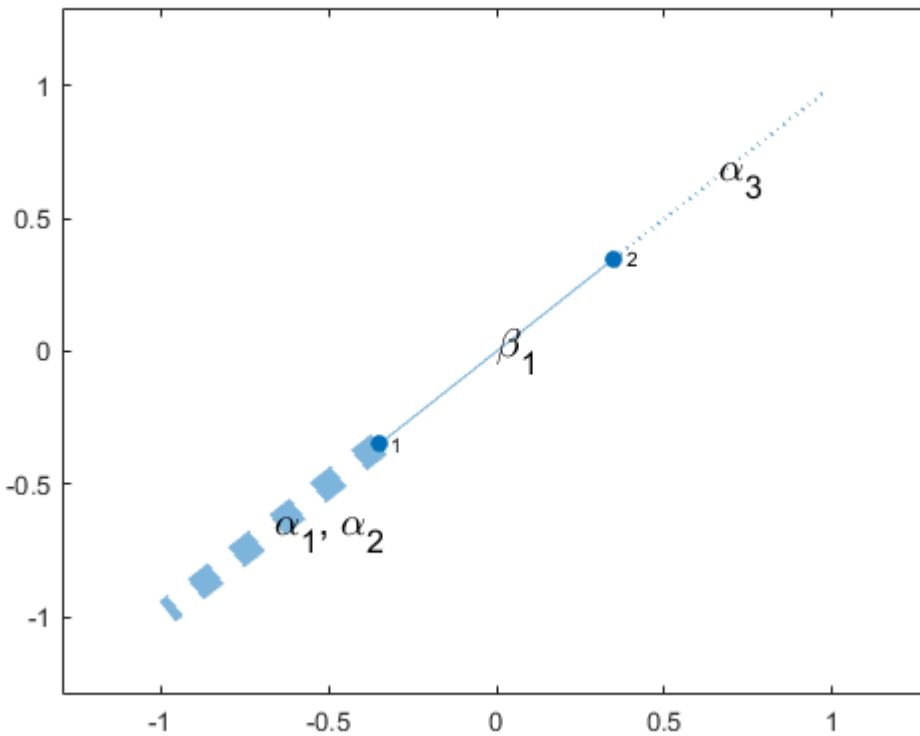
The second multiplication then contracts over β_1 :

```
N1N2N3 = boxtimes(N1x_keepbeta1_N2,N3)
```

```
N1N2N3 = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3'}
    pos: [1x1 struct]
```

```
data: [10×20×30 double]
```

```
figure; net_view(N1x_keepbeta1_N2,N3)
```



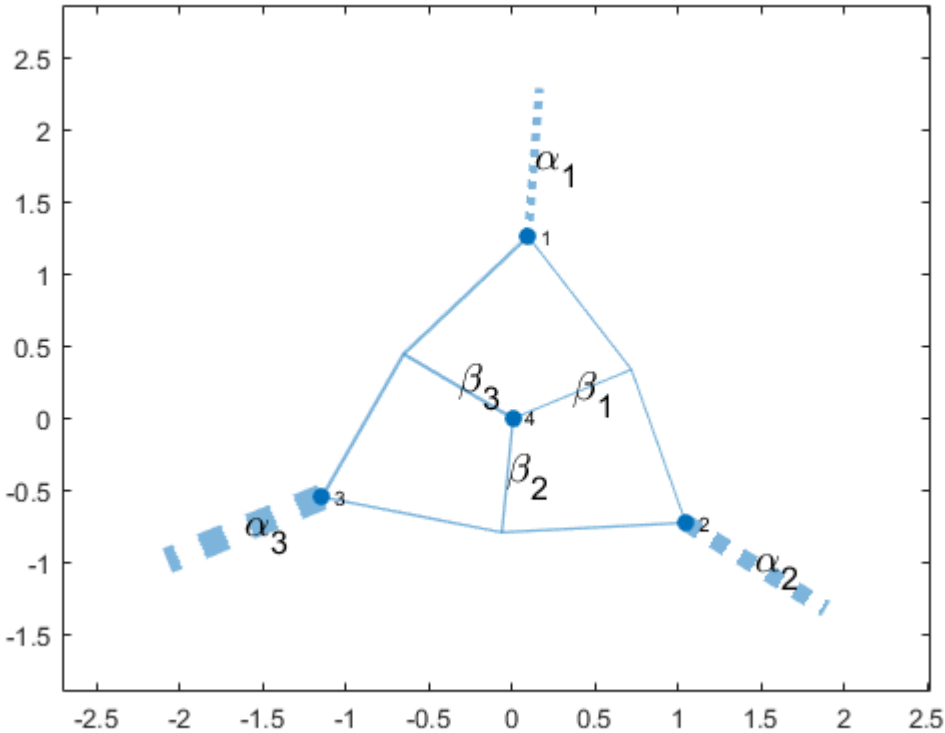
Multiplication of several nodes

These rules of contraction generalize to multiplications of any finite number of nodes. A contraction of two nodes can then both involve a contraction over one mode name and keeping another mode name.

```
n
```

```
n = struct with fields:
    alpha_1: 10
    alpha_2: 20
    alpha_3: 30
    beta_1: 2
    beta_2: 3
    beta_3: 4
```

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



```
boxtimes(N)
```

```
ans = struct with fields:
  mode_names: {'alpha_1' 'alpha_2' 'alpha_3'}
  pos: [1x1 struct]
  data: [10x20x30 double]
```

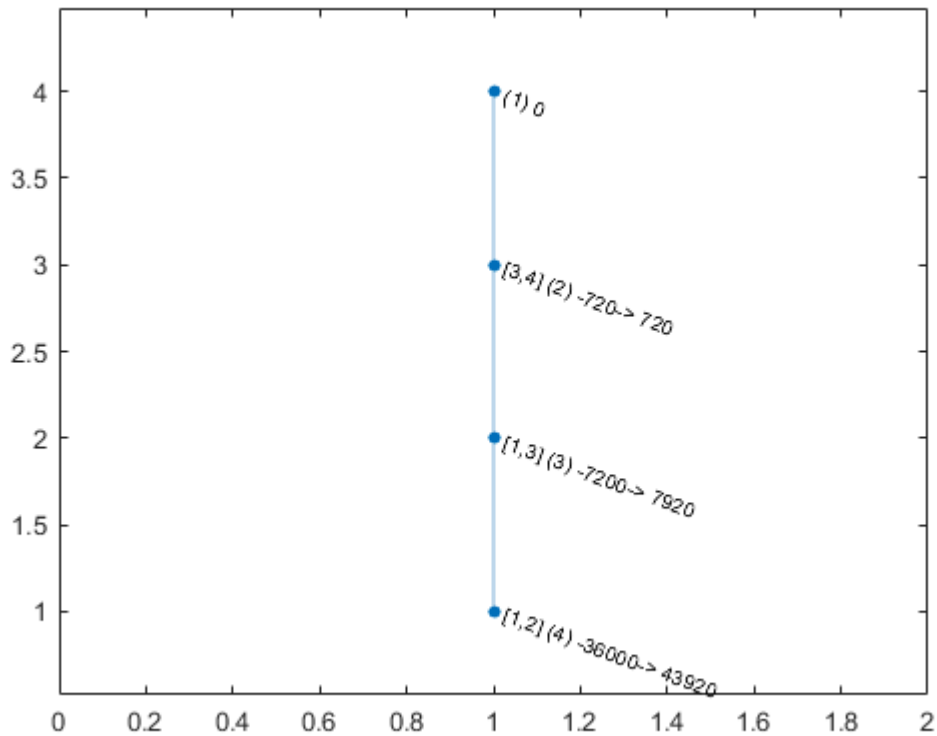
*(Order of contractions)

At this point, the question amounts in which order these nodes should be contracted. `boxtimes` will by default follow a certain greedy strategy to try to keep the total computational complexity of all required multiplication as low as possible. However, this strategy is not optimal in general. We can force `boxtimes` to find the optimal order of multiplications, but this may require a large amount of computation itself if many nodes are multiplied. In neither case will `boxtimes` account for specific patterns in the to be contracted network, as we could for example exploit in a matrix multiplication such as $(AB) \cdot (AB)$. If desired, this has to be done manually.

Once `boxtimes` has found an order of contractions for a certain network structure, it will save this strategy for later if this functionality is activated (more on that topic will follow).

First, the default strategy:

```
prodN = boxtimes(N, 'mode', 'show') % same as boxtimes(N) but with plot
```

```

Accum cost: 43920, Iterations: 4
Contractions:
[ 3 <- 4] -      720 ->      720
[ 1 <- 3] -      7200 ->     7920
[ 1 <- 2] -     36000 ->    43920
Indices of nodes remaining: 1
prodN = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3'}
        pos: [1x1 struct]
        data: [10x20x30 double]

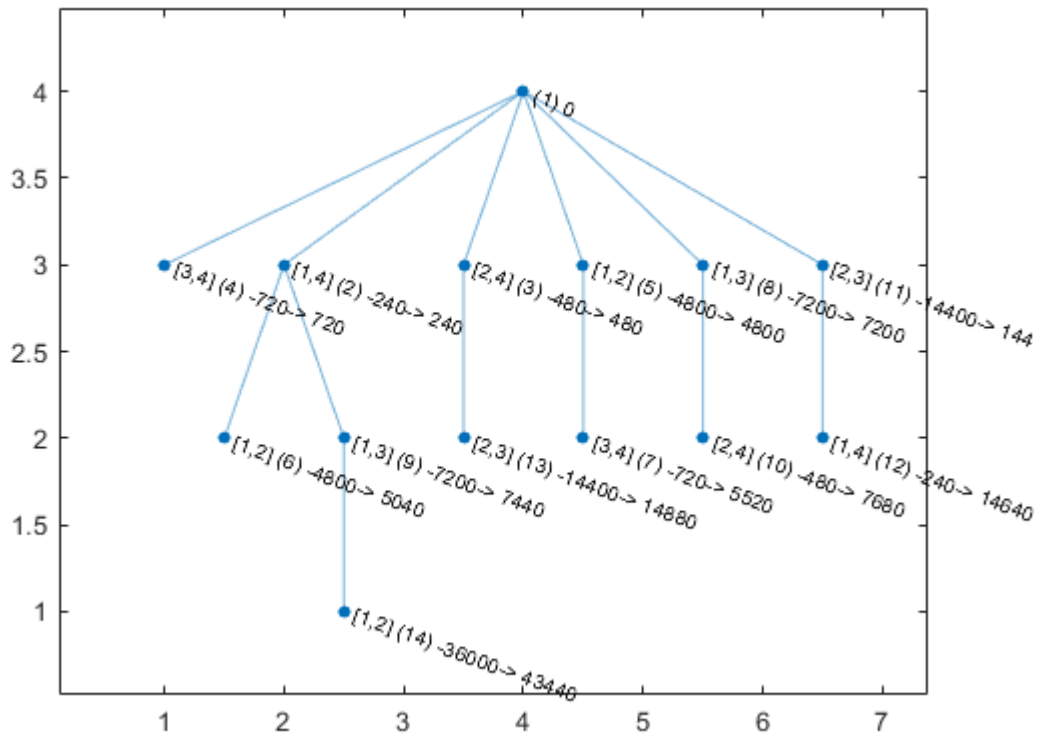
```

Then the optimal strategy:

```

prodN = boxtimes(N, 'mode', 'optimal_show') % same as boxtimes(N, 'mode', 'optimal') but w

```



```

Accum cost: 43440, Iterations: 14
Contractions:
[ 1 <- 4] -      240 ->      240
[ 1 <- 3] -     7200 ->     7440
[ 1 <- 2] -    36000 ->    43440
Indices of nodes remaining: 1
prodN = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3'}
        pos: [1x1 struct]
        data: [10x20x30 double]

```

We can see that the accumulated cost is only slightly lower when following the optimal order, while the results are of course the same (although the order of modes may vary). Yet, there are cases where the difference in cost can reach multiple orders of magnitude. If we follow the optimal order of contractions, we receive the following steps (`boxtimes` always contracts two nodes into the node with lower index).

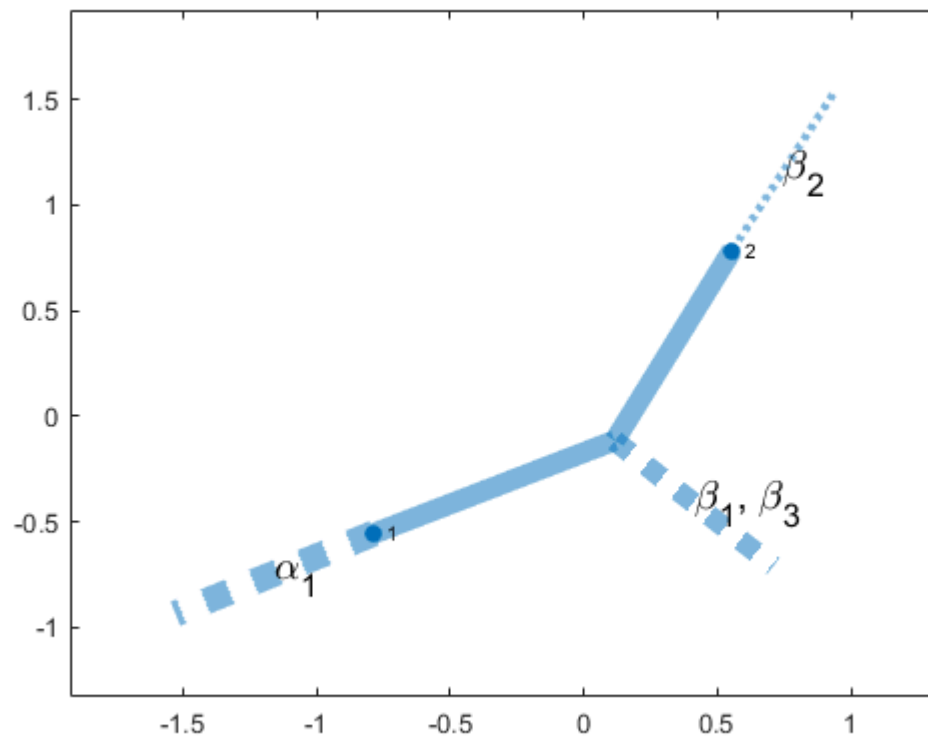
```
N14 = boxtimes(N{1},N{4}, '^',beta([1,3]))
```

```

N14 = struct with fields:
    mode_names: {'alpha_1' 'beta_1' 'beta_3' 'beta_2'}
        pos: [1x1 struct]
        data: [10x2x4x3 double]

```

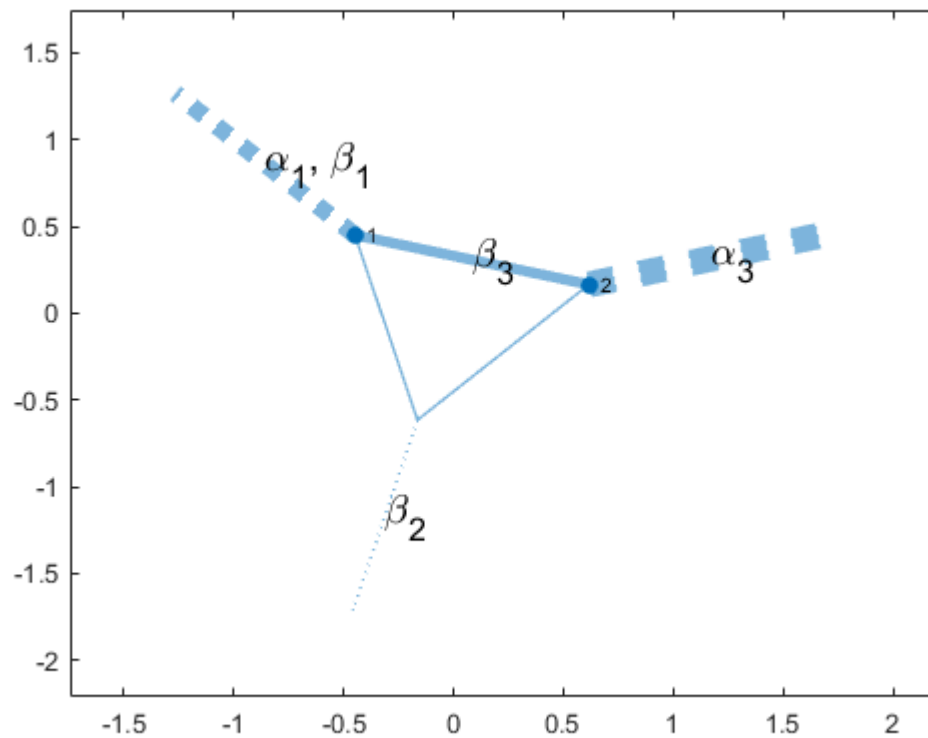
```
net_view(N{1},N{4}, '^',beta([1,3]))
```



```
N143 = boxtimes(N14,N{3},'^',beta(2))
```

```
N143 = struct with fields:
    mode_names: {'alpha_1' 'beta_1' 'beta_2' 'alpha_3'}
    pos: [1x1 struct]
    data: [10x2x3x30 double]
```

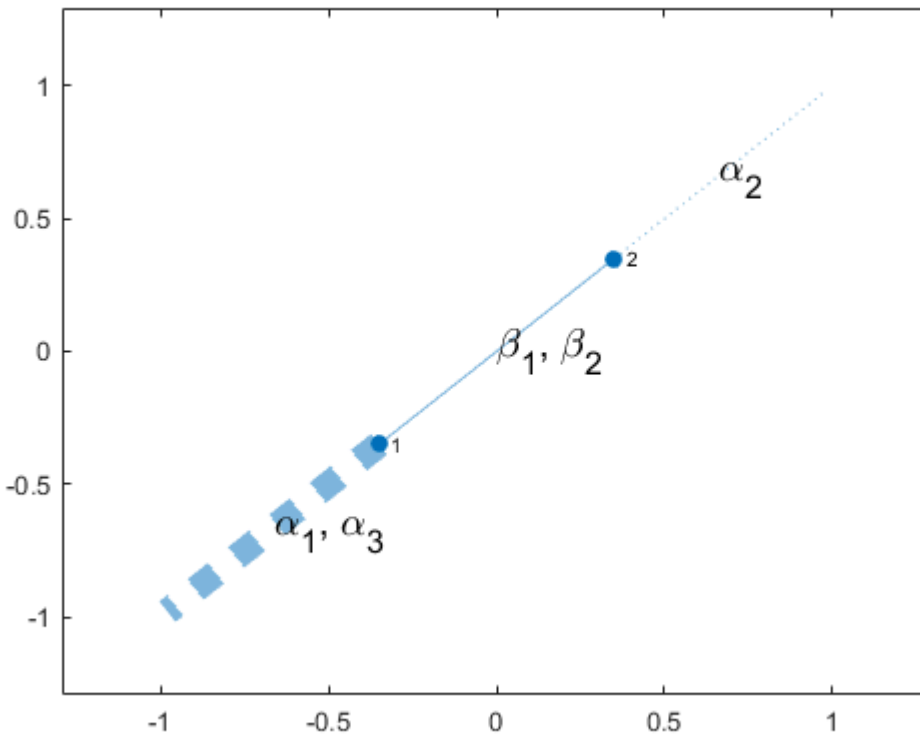
```
net_view(N14,N{3},'^',beta(2))
```



```
N1432 = boxtimes(N143,N{2})
```

```
N1432 = struct with fields:
  mode_names: {'alpha_1' 'alpha_3' 'alpha_2'}
  pos: [1x1 struct]
  data: [10x30x20 double]
```

```
net_view(N143,N{2})
```



We can check the result:

```
norm(unfold(prodN,alpha)-unfold(N1432,alpha),'fro')
```

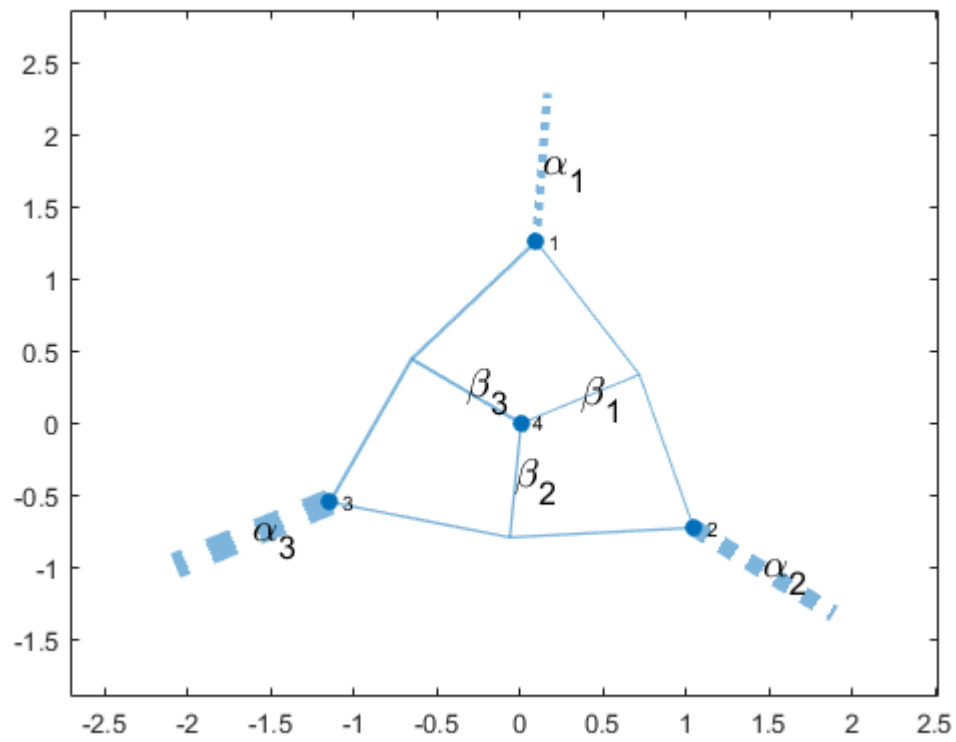
```
ans = 0
```

*(Ghost nodes)

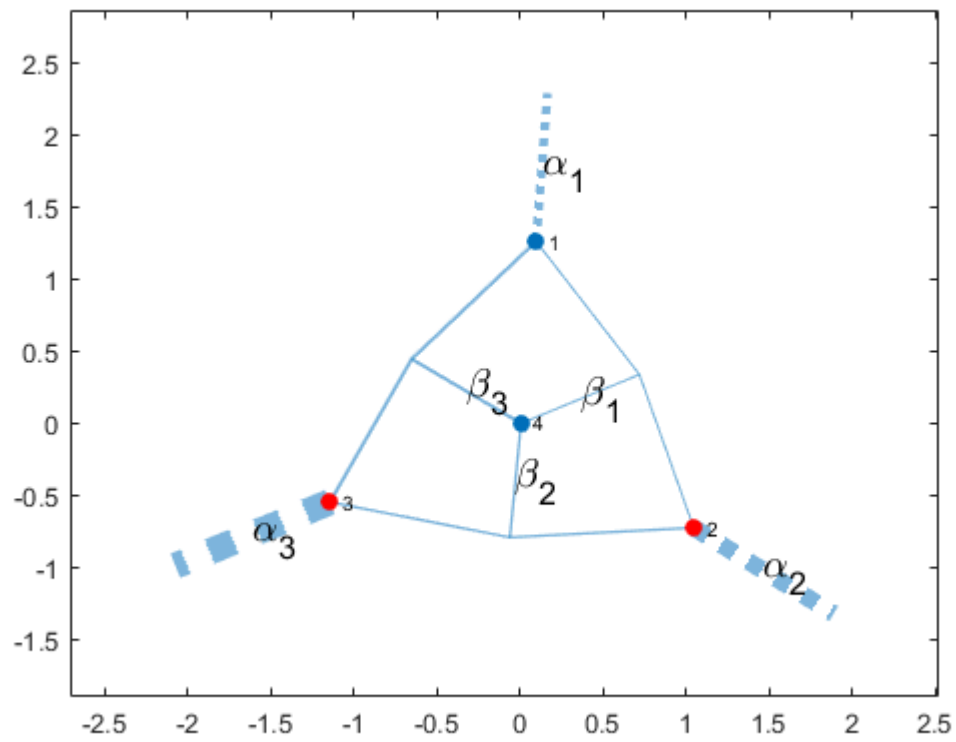
Ghost nodes are a bit of an unusual concept and might at this point be an answer to a question not yet asked, but they may ensure that the right mode names are contracted and kept. Any multiplication in which a ghost node appears, follows the same rules of contractions as if all ghost nodes were ordinary ones, but will never contract the ghost nodes itself (just if as they were not there).

Another point of view: we can use ghost nodes to tell `boxtimes` to only contract parts of a network, but still respect the network structure. This way, we may avoid telling `boxtimes` to keep certain mode names as above (using `^`).

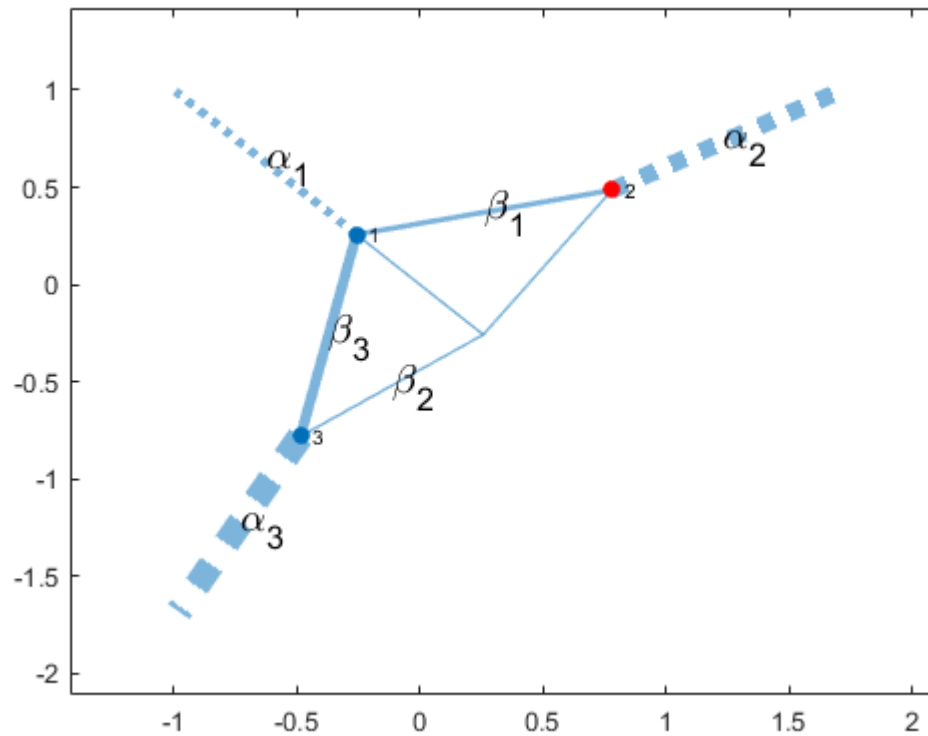
```
net_view(N)
```



```
N14 = boxtimes(N{1},ghost(N{2}),ghost(N{3}),N{4});
net_view(N{1},ghost(N{2}),ghost(N{3}),N{4})
```



```
N143 = boxtimes(N14,ghost(N{2}),N{3});
net_view(N14,ghost(N{2}),N{3})
```



```
N1432 = boxtimes(N143,N{2})
```

```
N1432 = struct with fields:
  mode_names: {'alpha_1' 'alpha_3' 'alpha_2'}
  pos: [1x1 struct]
  data: [10x30x20 double]
```

```
net_view(N143,N{2})
```