

Tensor Node Notation

Orthogonality and Duplicate Mode Names

Duplicate mode names

For many applications, we require one addition to the basic rules of tensor nodes and the contraction of these. We allow an identical mode name to appear twice in a tensor node.

Whenever a duplicate mode name appears

- *in an assignment, the first appearance and second appearance within both sides each correspond to each other*
- *in one single tensor node within a chain of tensor nodes, its first appearance corresponds to the last appearance of its left neighbour, and its second appearance to the first appearance of its right neighbour.*

The function `boxtimes` literally renames duplicate mode names temporarily until all nodes have been contracted (`boxtimes` can deal with nodes that have triplicate mode names or more, but there is not much use for this).

```
n = assign_mode_size({'alpha', 'beta', 'gamma', 'delta', 'epsilon'}, [7, 6, 4, 3, 2]);
N = init_node({'alpha', 'beta'}, n);
N = randomize_node(N)
```

```
N = struct with fields:
    mode_names: {'alpha' 'beta'}
    pos: [1×1 struct]
    data: [7×6 double]
```

```
A = init_node({'beta', 'beta'}, n);
A = randomize_node(A)
```

```
A = struct with fields:
    mode_names: {'beta' 'beta'}
    pos: [1×1 struct]
    data: [6×6 double]
```

We might even assign different mode sizes to the first and second identical mode name. If we do so, we can no longer speak of a global assignment of a mode size to a mode name.

```
ntilde = assign_mode_size({'alpha', 'beta', 'beta', 'gamma'}, [7, 6, 5, 4])
```

```
ntilde = struct with fields:
    alpha: 7
    beta: [6 5]
    gamma: 4
```

```
Atilde = init_node({'beta', 'beta'}, ntilde);
Atilde = randomize_node(Atilde)
```

```
Atilde = struct with fields:
    mode_names: {'beta' 'beta'}
        pos: [1×1 struct]
        data: [6×5 double]
```

Once we use duplicate mode names, the $[\times]$ -product is not commutative anymore, except if the nodes with duplicate mode names are symmetric regarding these.

```
AN = boxtimes(A,N)
```

```
AN = struct with fields:
    mode_names: {'beta' 'alpha'}
        pos: [1×1 struct]
        data: [6×7 double]
```

```
unfold(AN,{'alpha','beta'})
```

```
ans = 42×1
    0.0047
   -0.0096
    0.0411
   -0.0439
    0.0265
   -0.0187
    0.0292
   -0.0571
   -0.0754
    0.0499
     ⋮
     ⋮
```

```
NA = boxtimes(N,A)
```

```
NA = struct with fields:
    mode_names: {'alpha' 'beta'}
        pos: [1×1 struct]
        data: [7×6 double]
```

```
unfold(NA,{'alpha','beta'})
```

```
ans = 42×1
   -0.0540
    0.0439
   -0.1187
   -0.0053
   -0.1624
   -0.0001
    0.0986
    0.0717
   -0.0227
    0.0655
     ⋮
     ⋮
```

The tranpose applied to a tensor node with dublicate mode names is the generalization of a transpose of a matrix. For example, for

$$H = H(\alpha, \beta, \alpha, \beta)$$

```
H = init_node({'alpha', 'beta', 'alpha', 'beta'}, 4)
```

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

```
H = randomize_node(H); HT = node_transpose(H);
i = 1; j = 2; k = 3; ell = 4;
```

we have

$$H^T(\alpha = i, \beta = j, \alpha = k, \beta = \ell) = H(\alpha = k, \beta = \ell, \alpha = i, \beta = j)$$

```
node_part(HT, 'alpha', i, 'beta', j, 'alpha', k, 'beta', ell)
```

```
ans = struct with fields:
    mode_names: {'beta' 'alpha' 'beta' 'alpha'}
        pos: [1×1 struct]
        data: -6.6825e-04
```

```
node_part(H, 'alpha', k, 'beta', ell, 'alpha', i, 'beta', j)
```

```
ans = struct with fields:
    mode_names: {'alpha' 'beta' 'alpha' 'beta'}
        pos: [1×1 struct]
        data: -6.6825e-04
```

The implementation of `node_transpose` only has to invert the order of modes of the underlying data `H.data` (and then adapt `H.pos` and `H.mode_names`). For tensor nodes without duplicate mode names, applying transpose has no actual consequence.

We obtain the equality

$$A \begin{bmatrix} \times \end{bmatrix} N = (A \begin{bmatrix} \times \end{bmatrix} N)^T = N^T \begin{bmatrix} \times \end{bmatrix} A^T = N \begin{bmatrix} \times \end{bmatrix} A^T.$$

```
NAT = boxtimes(N, node_transpose(A))
```

```
NAT = struct with fields:
    mode_names: {'alpha' 'beta'}
        pos: [1×1 struct]
        data: [7×6 double]
```

```
unfold(NAT, {'alpha', 'beta'})
```

```
ans = 42×1
    0.0047
```

```

-0.0096
 0.0411
-0.0439
 0.0265
-0.0187
 0.0292
-0.0571
-0.0754
 0.0499
  ⋮

```

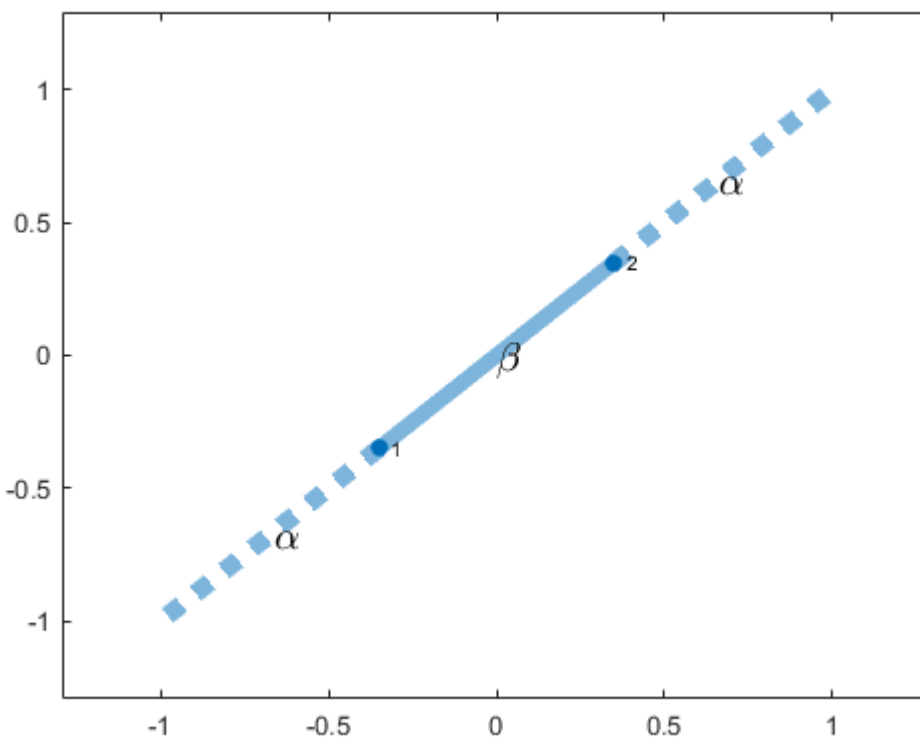
Partial contraction

For the concept of orthogonality, we require to use partial contractions such as $[\times]_\gamma$. This denotes that any (even duplicate) mode names other than γ are (for the duration of the multiplication) identified as different mode names. The results thereby may have duplicate mode names.

```
boxtimes(N,N,'_', 'beta')
```

```
ans = struct with fields:
    mode_names: {'alpha' 'alpha'}
        pos: [1×1 struct]
        data: [7×7 double]
```

```
net_view(N,N,'_', 'beta')
```



The general boxtimes product

The following demonstrates how a combination $[\times]_{\beta}^{\gamma}$ of kept mode names γ and partial contractions β works:

```
N1 = init_node({'alpha','beta','gamma','delta'},n)
```

```
N1 = struct with fields:
    mode_names: {'alpha' 'beta' 'gamma' 'delta'}
    pos: [1×1 struct]
    data: [7×6×4×3 double]
```

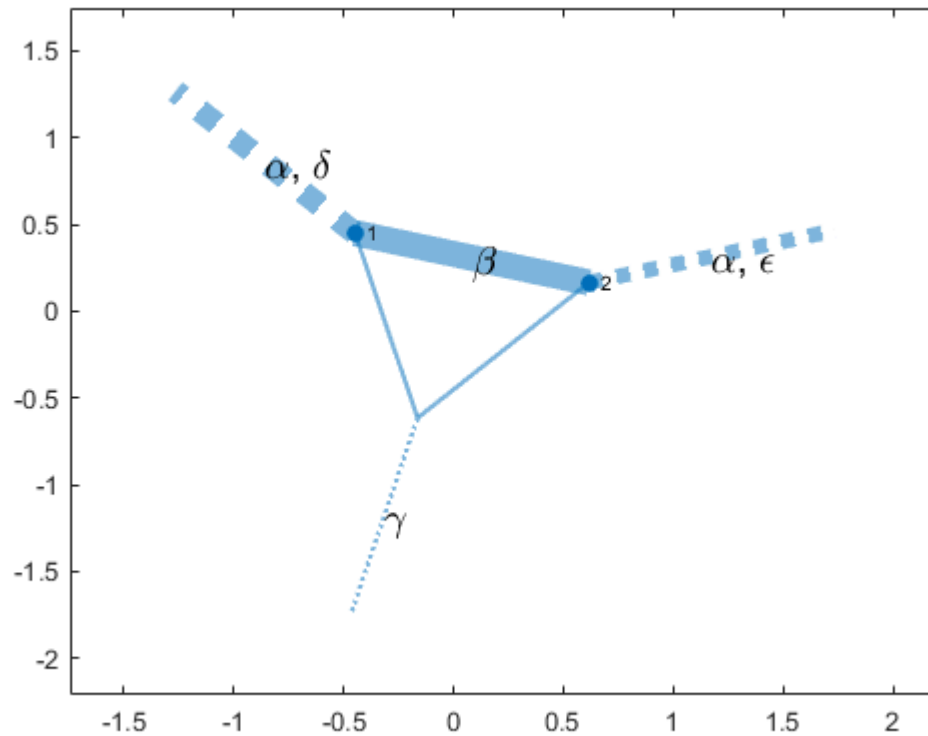
```
N2 = init_node({'alpha','beta','gamma','epsilon'},n)
```

```
N2 = struct with fields:
    mode_names: {'alpha' 'beta' 'gamma' 'epsilon'}
    pos: [1×1 struct]
    data: [7×6×4×2 double]
```

```
boxtimes(N1,N2,'_','beta','^','gamma')
```

```
ans = struct with fields:
    mode_names: {'alpha' 'gamma' 'delta' 'alpha' 'epsilon'}
    pos: [1×1 struct]
    data: [5-D double]
```

```
net_view(N1,N2,'_','beta','^','gamma')
```



Orthogonality

By definition, $N = N(\alpha, \beta)$ is β -orthogonal, if $N^T [\times]_{\alpha} N = I = I(\beta, \beta)$, where I is the identity matrix (i.e. the node with duplicate mode name β). Applying the transpose to N is not necessary if $\alpha \neq \beta$, but it makes things more familiar.

```
[Q,~] = qr(rand(n.alpha,n.beta),0);
N1 = fold(Q,{'alpha','beta'},n)
```

```
N1 = struct with fields:
    mode_names: {'alpha' 'beta'}
    pos: [1x1 struct]
    data: [7x6 double]
```

```
N1T_alpha_N1 = boxtimes(node_transpose(N1),N1,'_','alpha');
unfold(N1T_alpha_N1,'beta','beta')
```

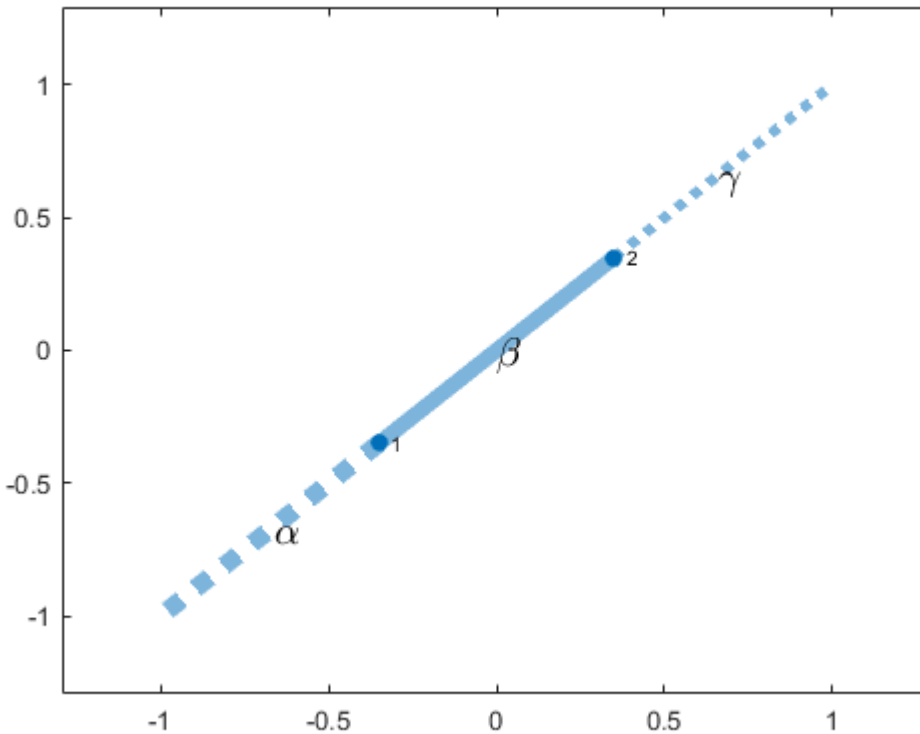
```
ans = 6x6
    1.0000    0.0000    0.0000         0    0.0000   -0.0000
    0.0000    1.0000   -0.0000   -0.0000   -0.0000   -0.0000
    0.0000   -0.0000    1.0000    0.0000    0.0000    0.0000
         0   -0.0000    0.0000    1.0000    0.0000   -0.0000
    0.0000   -0.0000    0.0000    0.0000    1.0000   -0.0000
   -0.0000   -0.0000   -0.0000   -0.0000   -0.0000    1.0000
```

We can verify a "chain rule" for orthogonality:

```
[Q2,~] = qr(rand(n.beta,n.gamma),0);
N2 = fold(Q2,{'beta','gamma'},n); % gamma-orthogonal
N12 = boxtimes(N1,N2)
```

```
N12 = struct with fields:
    mode_names: {'alpha' 'gamma'}
        pos: [1x1 struct]
        data: [7x4 double]
```

```
net_view(N1,N2)
```



```
N12T_alpha_N12 = boxtimes(node_transpose(N12),N12,'_','alpha')
```

```
N12T_alpha_N12 = struct with fields:
    mode_names: {'gamma' 'gamma'}
        pos: [1x1 struct]
        data: [4x4 double]
```

```
unfold(N12T_alpha_N12,'gamma','gamma')
```

```
ans = 4x4
    1.0000    -0.0000     0.0000     0.0000
   -0.0000     1.0000    -0.0000     0.0000
    0.0000    -0.0000     1.0000     0.0000
    0.0000     0.0000     0.0000     1.0000
```

We will from here on not anylonger apply the unnecessary transpose:

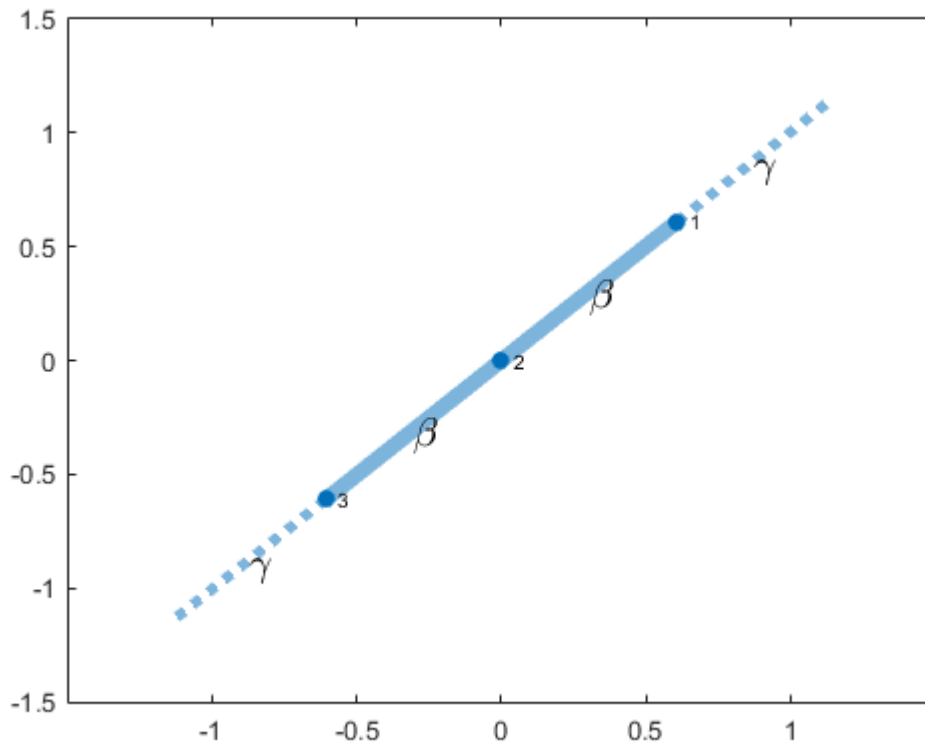
```
the_same = boxtimes(N2,boxtimes(N1,N1,'_','alpha'),N2,'_','beta')
```

```
the_same = struct with fields:
    mode_names: {'gamma' 'gamma'}
    pos: [1x1 struct]
    data: [4x4 double]
```

```
unfold(the_same,'gamma','gamma')
```

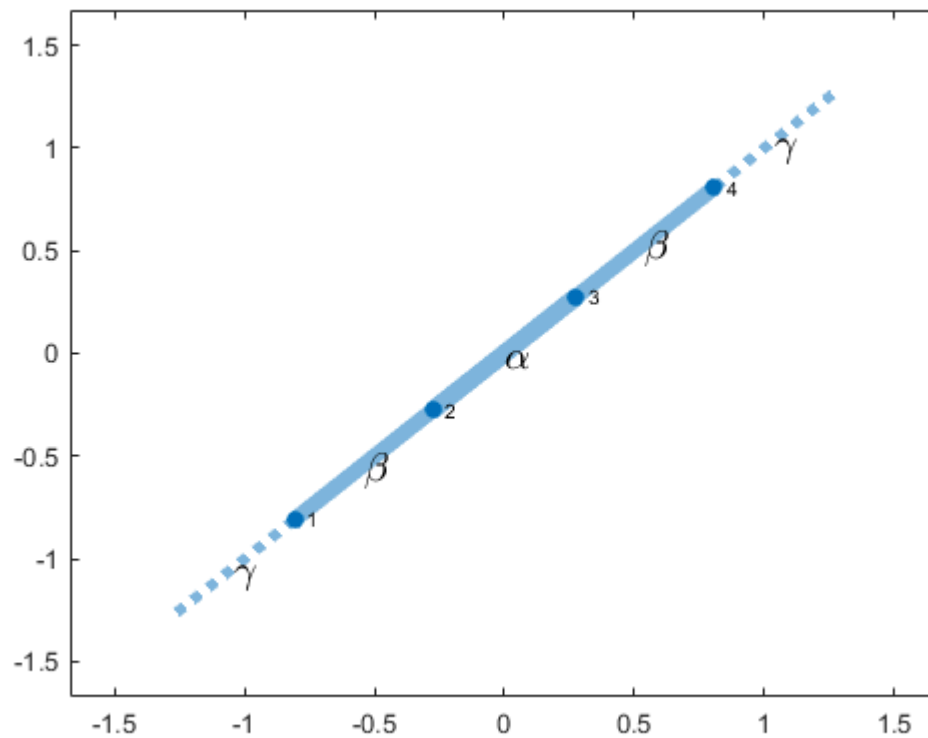
```
ans = 4x4
    1.0000    -0.0000     0.0000    -0.0000
   -0.0000     1.0000    -0.0000    -0.0000
    0.0000    -0.0000     1.0000    -0.0000
    0.0000    -0.0000    -0.0000     1.0000
```

```
net_view(N2,boxtimes(N1,N1,'_','alpha'),N2,'_','beta')
```



boxtimes and net_view can even do a bit more, but we will come to this later:

```
net_view(N2,{N1,N1,'_','alpha'},N2,'_','beta')
```

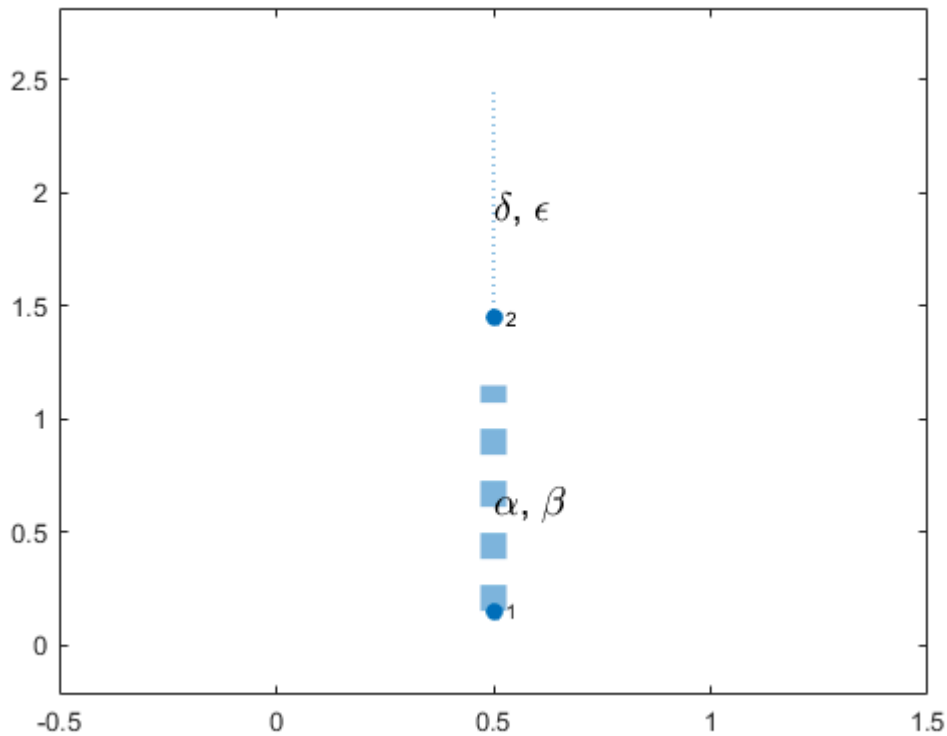



We can also verify that orthogonality is kept for nodes with distinct mode names:

```
[Q3,~] = qr(rand(n.delta,n.epsilon),0);
N3 = fold(Q3,{'delta','epsilon'},n); % epsilon-orthogonal
N13 = boxtimes(N1,N3)
```

```
N13 = struct with fields:
    mode_names: {'alpha' 'beta' 'delta' 'epsilon'}
    pos: [1x1 struct]
    data: [7x6x3x2 double]
```

```
net_view(N1,N3)
```



```
N13T_alpha_epsilon_N13 = boxtimes(node_transpose(N13),N13,'_',{'alpha','delta'})
```

```
N13T_alpha_epsilon_N13 = struct with fields:
    mode_names: {'epsilon' 'beta' 'beta' 'epsilon'}
    pos: [1x1 struct]
    data: [2x6x6x2 double]
```

```
unfold(N13T_alpha_epsilon_N13,{'beta','epsilon'},{'beta','epsilon'})
```

```
ans = 12x12
    1.0000    -0.0000         0         0    -0.0000    -0.0000     0.0000     0.0000 ...
   -0.0000     1.0000    -0.0000         0    -0.0000    -0.0000     0.0000     0.0000
         0    -0.0000     1.0000     0.0000     0.0000     0.0000     0.0000     -0.0000
         0         0     0.0000     1.0000     0.0000    -0.0000     0.0000         0
   -0.0000    -0.0000     0.0000     0.0000     1.0000    -0.0000    -0.0000     0.0000
   -0.0000    -0.0000     0.0000    -0.0000    -0.0000     1.0000     0.0000    -0.0000
     0.0000     0.0000         0     0.0000    -0.0000     0.0000     1.0000     0.0000
     0.0000     0.0000    -0.0000         0     0.0000    -0.0000     0.0000     1.0000
     0.0000    -0.0000     0.0000    -0.0000     0.0000     0.0000     0.0000     0.0000
         0     0.0000         0     0.0000     0.0000    -0.0000    -0.0000    -0.0000
        ⋮
```

Orthogonal nodes also conserve the Frobenius norm of other nodes in direction of their orthogonality. We will often use this when working with tensor formats.

N1

```
N1 = struct with fields:
    mode_names: {'alpha' 'beta'}
        pos: [1×1 struct]
        data: [7×6 double]
```

```
norm(N1.data(:)) % beta orthogonal
```

```
ans = 2.4495
```

N2

```
N2 = struct with fields:
    mode_names: {'beta' 'gamma'}
        pos: [1×1 struct]
        data: [6×4 double]
```

```
norm(N2.data(:)) % gamma orthogonal
```

```
ans = 2
```

N12

```
N12 = struct with fields:
    mode_names: {'alpha' 'gamma'}
        pos: [1×1 struct]
        data: [7×4 double]
```

```
norm(N12.data(:)) % still gamma orthogonal
```

```
ans = 2
```

```
net_view(N1,N2)
```

