

Tensor Node Notation

Node-QR and Node-SVD

Node-QR

The node-qr is the simple analog of a matrix-qr for tensor nodes.

In order to proceed a node-qr, we only need to specify a mode name in which we want to perform the orthogonalization:

```
load('TT-format')
G{:}
```

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

```
n = node_size(G)
```

```
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
```

```
[Q,R] = node_qr(G{1},'beta_1')
```

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

Q is now β_1 -orthogonal and $Q \times R = G_1$.

```
QT_alpha_Q = boxtimes(node_transpose(Q), Q, '_', 'alpha_1')
```

```
QT_alpha_Q = struct with fields:
    mode_names: {'beta_1' 'beta_1'}
    pos: [1x1 struct]
    data: [2x2 double]
```

```
unfold(QT_alpha_Q, 'beta_1', 'beta_1')
```

```
ans = 2x2
    1.0000    0
    0    1.0000
```

```
unfold(R, 'beta_1', 'beta_1')
```

```
ans = 2x2
    0.5084    0.3404
    0    0.7910
```

If we set $G_1 = Q$ and $G_2 = R \times G_2$ then we have not changed the tensor represented by G , yet G_1 is now β_1 -orthogonal - or *left-orthogonal*.

```
G{1} = Q;
G{2} = boxtimes(R, G{2});
```

We can also right-orthogonalize G_3 and G_4 . At this point it may feel unusual not to transpose R or similar when right-orthogonalizing. Due to the mode name notation, we however do not need to do this. We just have to keep in mind which side of R belongs to Q , which is the left side. Hence, in the following example, the right side belongs to G_3 .

```
[Q, R] = node_qr(G{4}, 'beta_3');
G{4} = Q;
G{3} = boxtimes(R, G{3});
[Q, R] = node_qr(G{3}, 'beta_2');
G{3} = Q;
G{2} = boxtimes(R, G{2});
```

The network G is now orthogonal with respect to G_2 . This simplifies the calculation of the norm of the represented tensor:

```
norm(unfold(G{2}))
```

```
ans = 0.3062
```

```
sqrt(get_data(boxtimes(G{2}, G{2})))
```

```
ans = 0.3062
```

```
T = boxtimes(G);
norm(T.data(:))
```

```
ans = 0.3062
```

Node-SVD

Similarly, we need to specify which mode names we want to split in the SVD, and also how we want to name the new mode names, which will appear in σ .

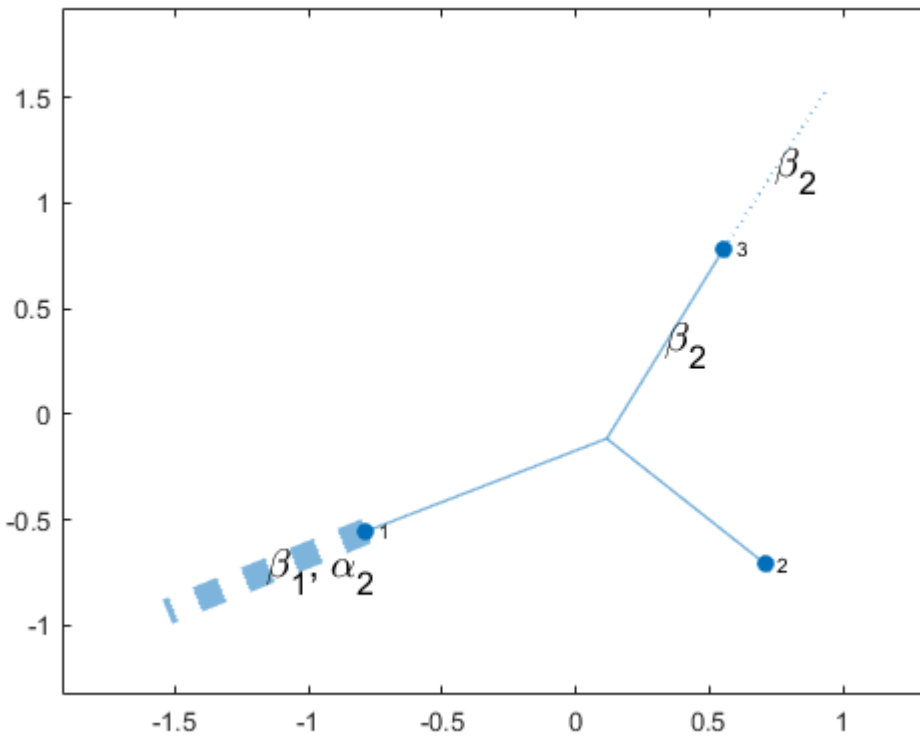
```
G{:}
```

```
ans = struct with fields:
    mode_names: {'alpha_1' 'beta_1'}
    pos: [1x1 struct]
    data: [10x2 double]
ans = struct with fields:
    mode_names: {'beta_2' 'beta_1' 'alpha_2'}
    pos: [1x1 struct]
    data: [2x2x10 double]
ans = struct with fields:
    mode_names: {'beta_3' 'alpha_3' 'beta_2'}
    pos: [1x1 struct]
    data: [2x10x2 double]
ans = struct with fields:
    mode_names: {'alpha_4' 'beta_3'}
    pos: [1x1 struct]
    data: [10x2 double]
```

```
[U,sigma,V] = node_svd(G{2},{ 'beta_1','alpha_2'}, 'beta_2')
```

```
U = struct with fields:
    mode_names: {'beta_1' 'alpha_2' 'beta_2'}
    pos: [1x1 struct]
    data: [2x10x2 double]
sigma = struct with fields:
    mode_names: {'beta_2'}
    pos: [1x1 struct]
    data: [2x1 double]
V = struct with fields:
    mode_names: {'beta_2' 'beta_2'}
    pos: [1x1 struct]
    data: [2x2 double]
```

```
net_view(U,sigma,V)
```



We will see easier ways to check the distance between the two tensor nodes in subsequent worksheets, but for now:

```
norm(unfold(boxtimes(U,sigma,V),G{2}.mode_names)-unfold(G{2}))
```

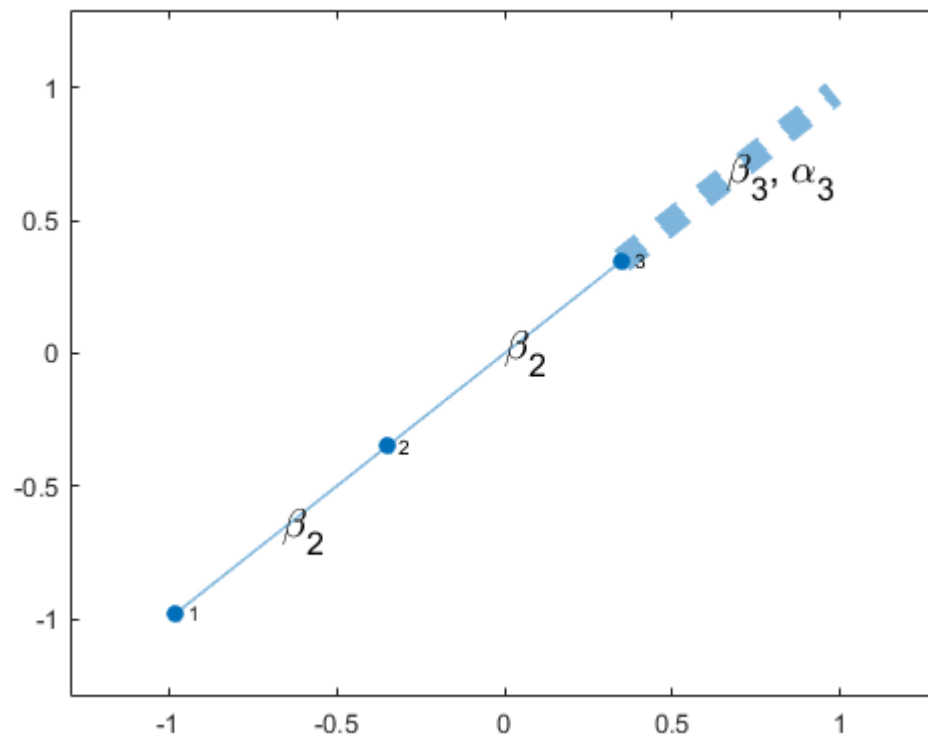
```
ans = 6.6692e-17
```

We have to be careful not to contract β_1 here:

```
G{2} = U;
warning('wrong contraction')
```

```
Warning: wrong contraction
```

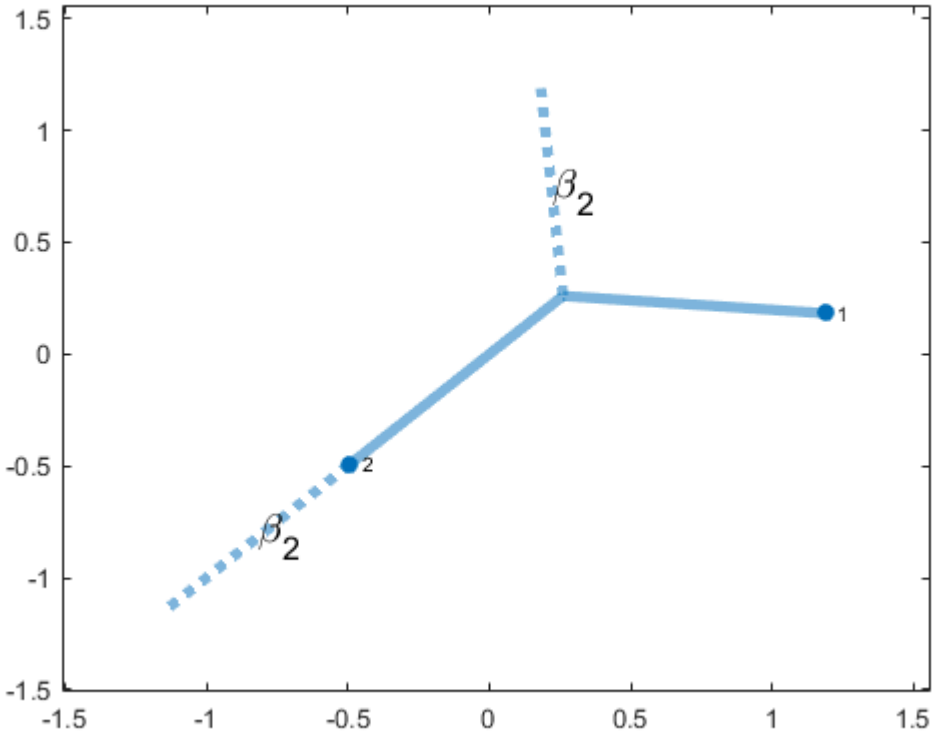
```
net_view(sigma,V,G{3})
```



```
sV = boxtimes(sigma,V,'^','beta_2')
```

```
sV = struct with fields:
  mode_names: {'beta_2' 'beta_2'}
  pos: [1x1 struct]
  data: [2x2 double]
```

```
net_view(sigma,V,'^','beta_2')
```



For vectors nodes there is a simple functionality:

```
Sigma = matrix_node_diag(sigma)
```

```
Sigma = struct with fields:
    mode_names: {'beta_2' 'beta_2'}
    pos: [1x1 struct]
    data: [2x2 double]
```

```
sV = boxtimes(Sigma,V)
```

```
sV = struct with fields:
    mode_names: {'beta_2' 'beta_2'}
    pos: [1x1 struct]
    data: [2x2 double]
```

```
G{3} = boxtimes(V,G{3})
```

```
G = 1x4 cell array
    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}
```

At this point, $(G_1 [\times] G_2, \sigma, G_3 [\times] G_4)$ is an SVD of $T = [\times] (G_1, G_2, \sigma, G_3, G_4)$ with respect to $\{\alpha_1, \alpha_2\}$ (because we orthogonalized G with respect to G_2 in the previous section). Note that we only form T to confirm the result.

```
[U_T,sigma_T,V_T] = node_svd(T,{'alpha_1','alpha_2'},'beta_2')
```

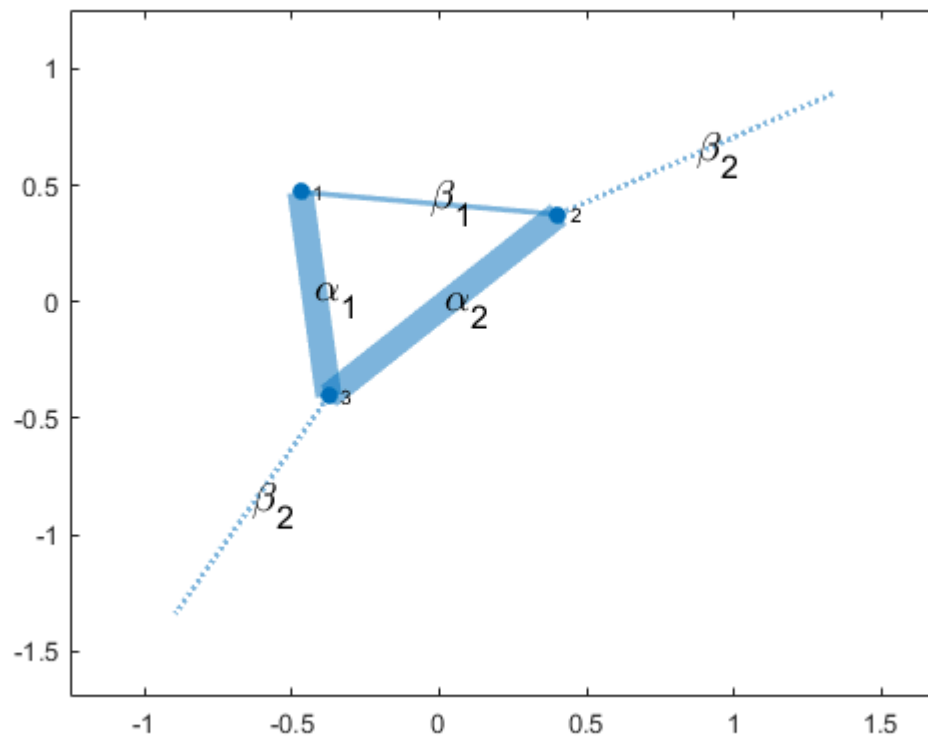
```
U_T = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'beta_2'}
    pos: [1x1 struct]
    data: [10x10x2 double]
sigma_T = struct with fields:
    mode_names: {'beta_2'}
    pos: [1x1 struct]
    data: [2x1 double]
V_T = struct with fields:
    mode_names: {'beta_2' 'alpha_3' 'alpha_4'}
    pos: [1x1 struct]
    data: [2x10x10 double]
```

We compare the singular vectors of both SVDs:

```
U_U_T = boxtimes(boxtimes(G(1:2)),U_T,'_',{'alpha_1','alpha_2'})
```

```
U_U_T = struct with fields:
    mode_names: {'beta_2' 'beta_2'}
    pos: [1x1 struct]
    data: [2x2 double]
```

```
net_view({G(1:2)},U_T,'_',{'alpha_1','alpha_2'}) % net_view is smart here
```



```
unfold(U_U_T,'beta_2','beta_2') % same singular vectors
```

```
ans = 2x2
    -1.0000    -0.0000
     0.0000    -1.0000
```

```
unfold(boxtimes(boxtimes(G(3:4)),V_T,'_{',{'alpha_3','alpha_4'}}),'beta_2','beta_2') % s
```

```
ans = 2x2
    -1.0000    -0.0000
     0.0000    -1.0000
```

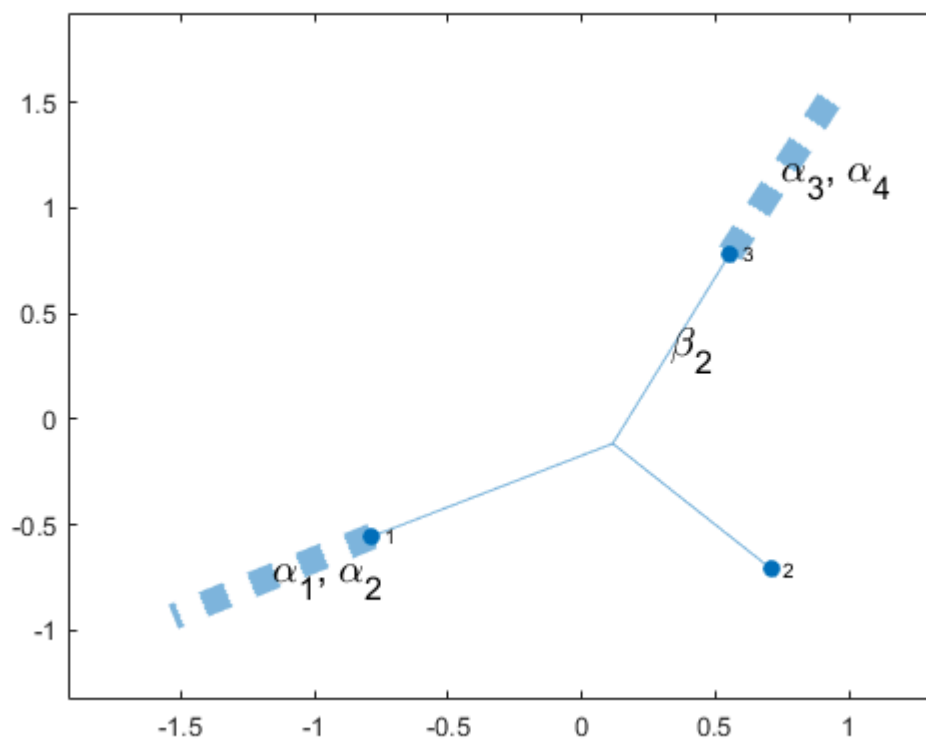
```
unfold(sigma_T,'beta_2')
```

```
ans = 2x1
    0.2427
    0.1868
```

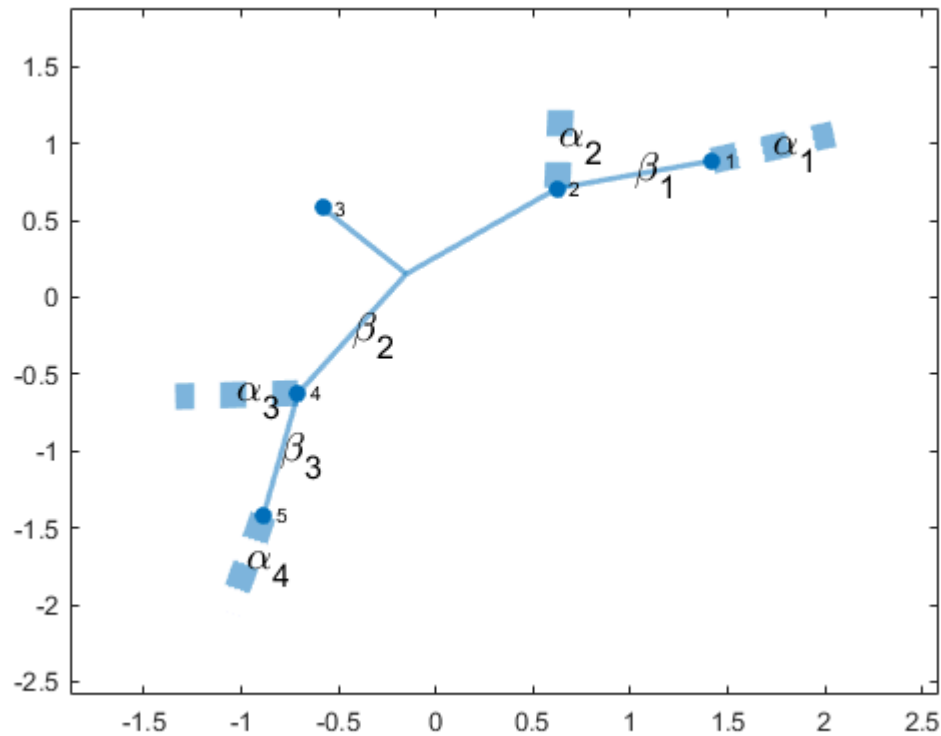
```
unfold(sigma,'beta_2')
```

```
ans = 2x1
    0.2427
    0.1868
```

```
net_view(U_T,sigma_T,V_T)
```



```
net_view(G(1:2),sigma,G(3:4))
```

The algorithms ORTHO.m and PATHQR.m

Instead of manual node-QRs we can use to orthogonalize any tree network to one node r , just as we did before with the tensor train format. `ORTHO` expects `N` to be a cell containing the nodes of the network corresponding to a tree graph (not a hypergraph!). Each node must not have duplicate mode names. Every mode name must further appear in at most two nodes. We call such a plain network.

```
help ORTHO
```

ORTHO Orthogonalization with respect to node r of a tree network

`N = ORTHO(N,r)` orthogonalizes the network `N` with respect to the root `r`.

`N` must be a cell containing the tensor nodes of the network. Each node must not have duplicate mode names. Every mode name must further appear in at most two nodes (we call this a plain network).

`N = ORTHO(N,r,G)` expects `G` to be the same as `net_derive_G(N)` and does hence not need to calculate the graph `G` corresponding to the tree network.

See also: `tensor_node_notation5_node_qr_and_node_svd.mlx`, `PATHQR`, `net_derive_G`

```
dbtype ORTHO.m 18:36 % it is a rather short code
```

```
18 if nargin <= 2
```

```

19     G = net_derive_G(N);
20 end
21
22 ORTHOREC(r, []);
23
24 function ORTHOREC(b,P)
25     for h = int_setdiff(neighbors(G,b)',P)
26         ORTHOREC(h,b);
27     end
28
29     if ~isempty(P)
30         gamma = str_intersect(N{P}.mode_names,N{b}.mode_names);
31         [Q,R] = node_qr(N{b},gamma);
32         N{b} = Q;
33         N{P} = boxtimes(R,N{P});
34     end
35 end
36 end

```

```

load('Tucker-format');
Net = [U,C];
Net{:}

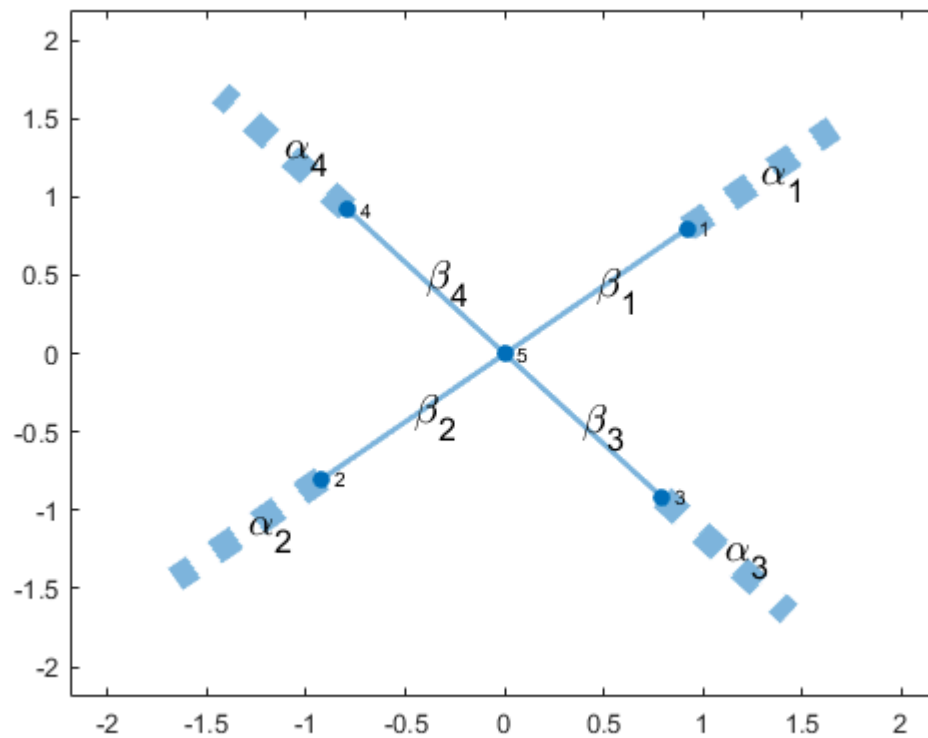
```

```

ans = struct with fields:
    mode_names: {'alpha_1' 'beta_1'}
    pos: [1x1 struct]
    data: [10x2 double]
ans = struct with fields:
    mode_names: {'alpha_2' 'beta_2'}
    pos: [1x1 struct]
    data: [10x2 double]
ans = struct with fields:
    mode_names: {'alpha_3' 'beta_3'}
    pos: [1x1 struct]
    data: [10x2 double]
ans = struct with fields:
    mode_names: {'alpha_4' 'beta_4'}
    pos: [1x1 struct]
    data: [10x2 double]
ans = struct with fields:
    mode_names: {'beta_1' 'beta_2' 'beta_3' 'beta_4'}
    pos: [1x1 struct]
    data: [2x2x2x2 double]

```

```
net_view(Net)
```



```
T = boxtimes(Net);
Net = ORTHO(Net,2);
get_data(boxtimes(T,T))
```

```
ans = 0.0674
```

```
get_data(boxtimes(Net{2},Net{2}))
```

```
ans = 0.0674
```

Once a network is orthogonal to a node r , it suffices to call `PATHQR` to change the orthogonalization:

```
type PATHQR.m
```

```
function [N,t,P] = PATHQR(N,s,t,G)
% PATHQR changes the orthogonalization of a plain network
%
% PATHQR(N,s,t) takes a network orthogonalized to s and changes the
% orthogonalization to t. The network must be plain, for example the
% output of ORTHO(N,s).
%
% See also: TENSOR_NODE_NOTATION5_NODE_QR_AND_NODE_SVD.mlx, ORTHO

if nargin <= 3
    G = net_derive_G(N);
end

P = shortestpath(G,s,t);
```

```

for i = 1:length(P)-1
    b = P(i); h = P(i+1);
    gamma = str_intersect(N{b}.mode_names,N{h}.mode_names);
    [Q,R] = node_qr(N{b},gamma);
    N{b} = Q;
    N{h} = boxtimes(R,N{h});
end

```

```

Net = PATHQR(Net,2,5);
get_data(boxtimes(Net{5},Net{5}))

```

```

ans = 0.0674

```