

Tensor Node Notation

Decompositions

Leaves-to-root truncation

Given a tensor node $T = T(\alpha)$ and a set of mode names m such that the corresponding graph G is plain, we can decompose this tensor into a representation N with according mode names. `LTRT` also returns the standard representation of the output N .

```
alpha = mna('alpha',1:6);

S = binary_tree(alpha);
[G,m,~,~,beta] = RTLGRAPH(alpha,S,'beta');

n = assign_mode_size(alpha,5);
```

We can use just `m` to call `LTRT`:

```
type LTRT.m
```

```
function [N,fN,sigma] = LTRT(T,m,r,G,tol)
% LTRT Leaves-to-root truncation of a tensor.
%
% Input:
%   T: tensor node
%   m: a cell for which m{k} are the mode names of node number k
%       (the corresponding network must be plain)
%   r: a root node
%
% -optional:
%   G: [] or the graph equal to derive_G(m)
%   tol: tolerance for singular value decompositions
%
% Output:
%   N: a tensor network representing the leaves-to-root truncation of T
%   fN/sigma: the standard representation of N,
%       where boxtimes(N{:}) = boxtimes(fN{:},sigma{:})
%
% Example:
%   alpha = mna('alpha',1:6);
%   n = assign_mode_size(alpha,1+(1:6));
%   T = init_node(alpha,n);
%
%   T = randomize_node(T);
%   node_size(T)
%
%   S = binary_tree(alpha);
%   [G,m,M,r] = RTLGRAPH(alpha,S);
%
%   [N,fN,sigma] = LTRT(T,m,r); % = LTRT(T,m,r,G,1e-14);
%   data_volume(N)
%   data_volume(T)
%
```

```

%      T_trunc = boxtimes(N{:});
%      norm(unfold(T_trunc,alpha) - unfold(T,alpha))
%      norm(unfold(T_trunc,alpha) - unfold(boxtimes(fN{:},sigma{:}),alpha))
%
%      check_if_standard(fN,sigma,G)
%
%      See also: TENSOR_NODE_NOTATION9_DECOMPOSITIONS, RTLT, NODE_SVD,
%      NODE_QR, BOXTIMES, UNFOLD, DATA_VOLUME

if nargin == 3 || isempty(G)
    G = derive_G(m);
end
if nargin <= 4
    tol = 1e-14;
end

k = numnodes(G);
N = cell(1,k);
fN = cell(1,k);
sigma = cell(1,k-1);
DECOMPREC(r, []);

function s = DECOMPREC(b,P)
    for h = int_setdiff(neighbors(G,b)',P)
        DECOMPREC(h,b);
    end

    if ~isempty(P)
        gamma = str_intersect(m{P},m{b});
        [U,s,Q] = node_svd(T,listdiff(m{b},gamma),gamma,tol);
        T = boxtimes_part(s,Q,{},gamma); % = boxtimes(node_diag(s),Q);
        N{b} = U;
        fN{b} = U;
        sigma{findedge(G,P,b)} = s;
    else
        N{b} = T;
        fN{b} = T;
    end

    for h = int_setdiff(neighbors(G,b)',P)
        s = sigma{findedge(G,h,b)};
        s.data = (s.data).^(-1);
        fN{b} = boxtimes_part(s,fN{b},{},s.mode_names);
    end
end

end

```

```

T = init_node(alpha,n);
x = 1:numel(T.data);
T.data(:) = 1/3*x.^5 + 1/3*x.^4 - 1/2*x.^3 - 1/2*x.^2 + 1/2*x - 1; % is low rank
[N,fN,sigma] = LTRT(T,m,1)

```

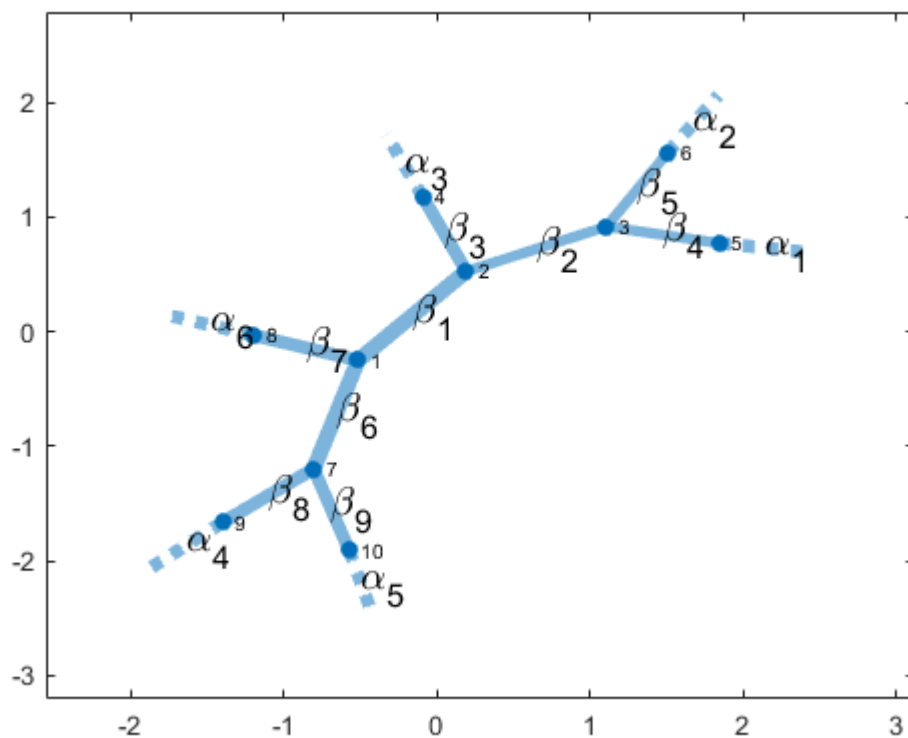
```

N = 1x10 cell array
    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1
fN = 1x10 cell array
    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1
sigma = 1x9 cell array
    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1

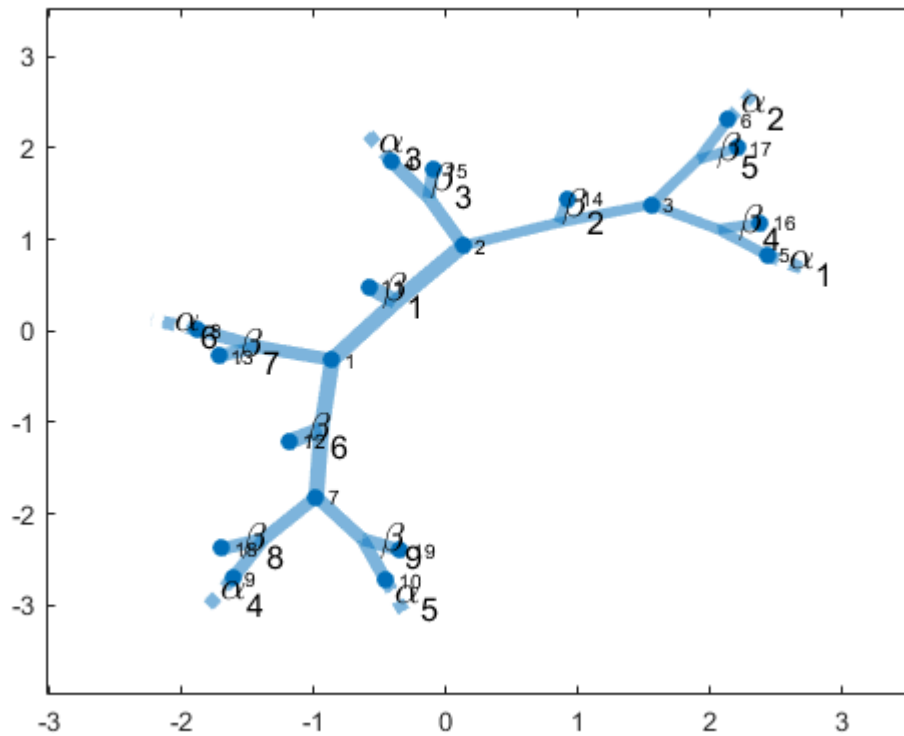
```

The thickness of the lines in the network plot represents its mode sizes and ranks, that is, the sizes of the domains of the internal mode names.

```
net_view(N)
```



```
net_view(fN{:},sigma{:}))
```



```
n = node_size(N)
```

```
n = struct with fields:
```

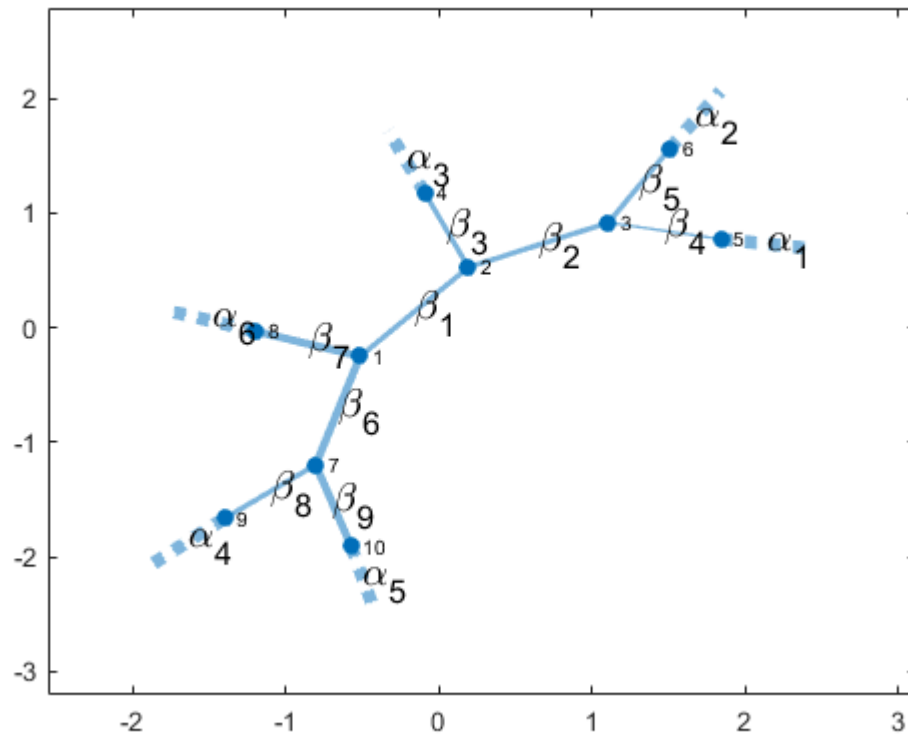
```
alpha_1: 5
alpha_2: 5
alpha_3: 5
alpha_4: 5
alpha_5: 5
alpha_6: 5
beta_1: 6
beta_2: 4
beta_3: 5
beta_4: 4
beta_5: 4
beta_6: 6
beta_7: 5
beta_8: 5
beta_9: 5
```

We have constructed T such that it is close to a rank 1 representation:

```
tol = 1/10000;
N_trunc = LTRT(T,m,1,[],tol)
```

```
N_trunc = 1x10 cell array
    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}
```

```
net_view(N_trunc)
```



```
n_trunc = node_size(N_trunc)
```

```
n_trunc = struct with fields:
```

```
alpha_1: 5
alpha_2: 5
alpha_3: 5
alpha_4: 5
alpha_5: 5
alpha_6: 5
beta_1: 2
beta_2: 2
beta_3: 2
beta_4: 1
beta_5: 2
beta_6: 3
beta_7: 3
beta_8: 2
beta_9: 3
```

```
data_volume(T)
```

```
ans = 15626
```

```
data_volume(N)
```

```
ans = 655
```

```
data_volume(N_trunc)
```

```
ans = 114
```

```
net_dist(N,N_trunc)/net_norm(N)
```

```
ans = 6.8193e-05
```

Root-to-leaves truncation

We can apply the root-to-leaves truncation as well. It usually yields slightly better approximation bounds, but is also slightly more time consuming. It also returns the truncated tensor represented by the output.

```
type RTLTL.m
```

```
function [N,Tprime] = RTLTL(T,m,r,G,M,tol)
% RTLTL Root-to-leaves truncation of a tensor.
%
% Input:
%   T: tensor node
%   m: a cell for which m{k} are the mode names of node number k
%       (the corresponding network must be plain)
%   r: a root node
%
% -optional:
%   G: [] or the graph equal to derive_G(m)
%   M: [] or the cell equal to derive_M(G,m,r)
%   tol: tolerance for singular value decompositions
%
% Output:
%   N: a tensor network representing the root-to-leaves truncation of T
%   Tprime: equals boxtimes(N{:})
%
% Example:
%   alpha = mna('alpha',1:6);
%   n = assign_mode_size(alpha,1+(1:6));
%   T = init_node(alpha,n);
%
%   T = randomize_node(T);
%   node_size(T)
%
%   S = binary_tree(alpha);
%   [G,m,M,r] = RTLGRAPH(alpha,S);
%
%   [N,Tprime] = RTLTL(T,m,r); % = RTLTL(T,m,r,G,M,1e-14);
%   data_volume(N)
%   data_volume(T)
%   view(G,m,N);
%
%   norm(unfold(Tprime,alpha) - unfold(T,alpha))
%   norm(unfold(Tprime,alpha) - unfold(boxtimes(N{:}),alpha))
%
%   boxtimes(N,N);
%   norm(T.data(:))
%
% See also: TENSOR_NODE_NOTATION9_DECOMPOSITIONS.mlx, RTLGRAPH, NODE_SVD,
%           NODE_QR, BOXTIMES, UNFOLD, DATA_VOLUME

if nargin == 3 || isempty(G)
    G = derive_G(m);
end
```

```

if nargin <= 4 || isempty(M)
    M = derive_M(G,m,r);
end
if nargin <= 5
    tol = 1e-14;
end

k = numnodes(G); % expects V = {1,...,k}
N = cell(1,k);

output_Tprime = false;
if nargin == 2
    output_Tprime = true;
    Tprime = T;
end

RTLTREC(r,[]);

function RTLTREC(b,P)
    if ~isempty(P)
        gamma = str_intersect(m{b},m{P});
        delta = M{b};
        [U,~,~] = node_svd(T,delta,gamma,tol);
        N{b} = U;
        N{P} = boxtimes(N{P},U);

        if output_Tprime
            Tprime = boxtimes(boxtimes(Tprime,U),U);
        end
    else
        N{b} = T;
    end

    for h = int_setdiff(neighbors(G,b)',P)
        RTLTREC(h,b);
    end
end
end

```

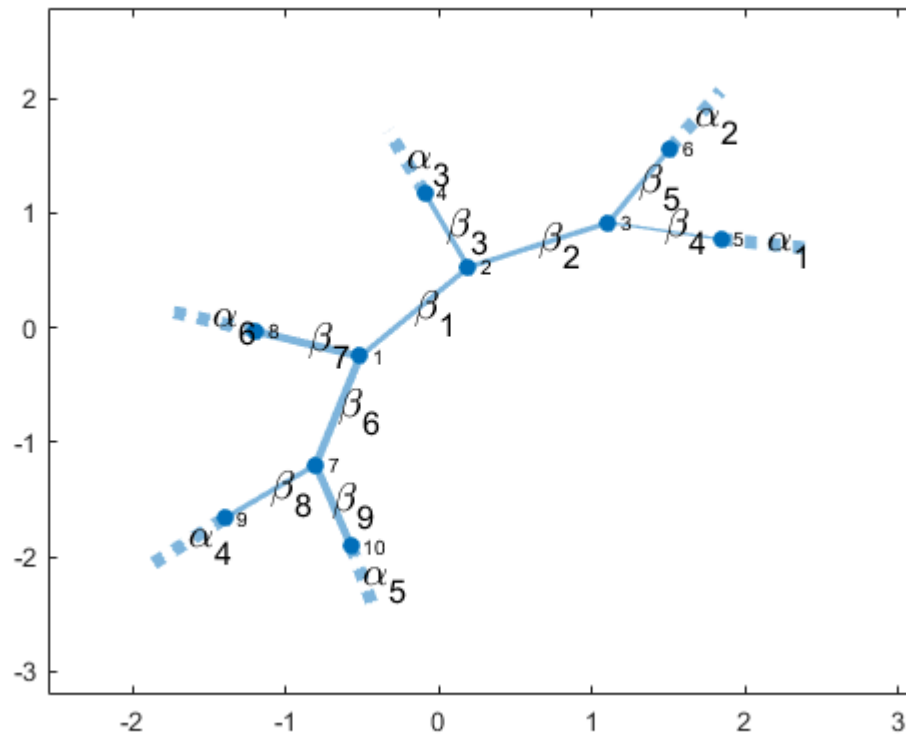
```
[N_rtl_trunc,T_rtl_trunc] = RTLT(T,m,1,[],[],tol)
```

```

N_rtl_trunc = 1x10 cell array
    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1 struct}    {1x1
T_rtl_trunc = struct with fields:
    mode_names: {'alpha_1' 'alpha_2' 'alpha_3' 'alpha_4' 'alpha_5' 'alpha_6'}
        pos: [1x1 struct]
        data: [6-D double]

```

```
net_view(N_rtl_trunc)
```



```
net_dist(N_rtl_trunc,N)/net_norm(N)
```

```
ans = 6.8193e-05
```

```
net_dist(N_rtl_trunc,T_rtl_trunc)/net_norm(T_rtl_trunc)
```

```
ans = 1.7007e-08
```

Truncation based on the standard representation

We may also truncate a tensor based on its representation using the standard representation (or tree SVD):

```
[~,fN,sigma] = STAND(N,1);
net_dist(N,[fN,sigma])/net_norm(N)
```

```
ans = 2.6890e-08
```

```
restr = cell(1,length(beta));
[restr{:}] = deal(1);
fNs_trunc = node_part([fN,sigma],beta,restr);
net_dist(fNs_trunc,N_rtl_trunc)/net_norm(N_rtl_trunc)
```

```
ans = 0.0241
```

```
net_dist(fNs_trunc,N)/net_norm(N)
```



```
ans = 0.0241
```

Without additional computation, the representation can be truncated to any other rank as well.

```
restr = cell(1,length(beta));  
[restr{:}] = deal(1:2);  
fNs_trunc = node_part([fN,sigma],beta,restr);  
net_dist(fNs_trunc,N_rtl_trunc)/net_norm(N_rtl_trunc)
```

```
ans = 7.4453e-04
```

```
restr = cell(1,length(beta));  
[restr{:}] = deal(1:3);  
fNs_trunc = node_part([fN,sigma],beta,restr);  
net_dist(fNs_trunc,N_rtl_trunc)/net_norm(N_rtl_trunc)
```

```
ans = 6.2065e-05
```

Instead of a common rank, we may also truncate to the earlier computed ranks in `n_trunc`:

```
for i = 1:length(beta)  
    restr{i} = 1:n_trunc.(beta{i});  
end  
fNs_trunc = node_part([fN,sigma],beta,restr);  
net_dist(fNs_trunc,N_rtl_trunc)/net_norm(N_rtl_trunc)
```

```
ans = 1.2026e-08
```