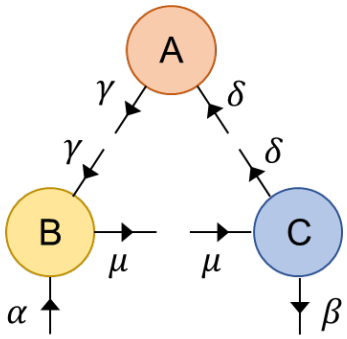
Tensor contraction

Author: Seung-Sup Lee

Here we will explain how to contract tensors in MATLAB. Consider three tensors A, B, and C:



The legs with the same indices will be contracted. In this tutorial, we will treat the tensors as mere numerical arrays in which the directions of tensor legs are not important. Thus we will omit the arrows for the rest of this tutorial. Of course, when the tensors are in physical context (e.g., bras, kets) or non-Abelian symmetries are exploited (to be covered later in the lecture course), the direction of legs does matter!

Initialization

Clear workspace (clear pre-existing variables to avoid any collision), and set the bond dimension `D` for tensors as 10. Then generate rank-2 tensor `A` (of size `D*D`) and rank-3 tensors `B` and `C` (of size `D*D*D`) with random elements.

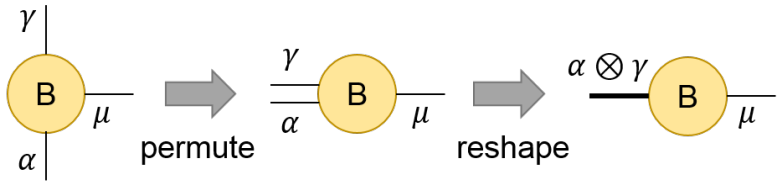
```
clear

% bond dimensions
Da = 10; % alpha
Db = 12; % beta
Dc = 14; % gamma
Dd = 17; % delta
Dm = 20; % mu

A = rand(Dc,Dd);    % tensor A(gamma,delta)
B = rand(Da,Dm,Dc); % tensor B(alpha,mu,gamma)
C = rand(Db,Dm,Dd); % tensor C(beta,mu,delta)
```

Initiate timers for real/CPU tiume.

```
tobj = tic2;
```

Contract B and C

Let's contract B and C first. Reshape rank-3 tensor B as matrix, by permuting leg indices and by reshaping, as a diagram below:



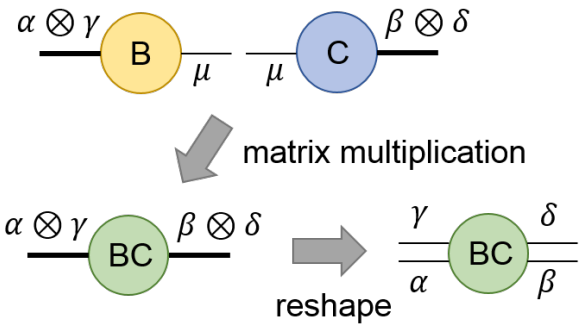
Here thick leg means that the associated bond dimension is big, since two legs are combined by `reshape`.

```
B1 = permute(B,[1,3,2]); % B(alpha,mu,gamma) -> B(alpha,gamma,mu)
B1 = reshape(B1,[Da*Dc,Dm]); % B(alpha,gamma,mu) -> B(alpha*gamma,mu)
```

Treat tensor C similarly.

```
C1 = permute(C,[2,1,3]); % C(beta,mu,delta) -> C(mu,beta,delta)
C1 = reshape(C1,[Dm,Db*Dd]); % C(mu,beta,delta) -> C(mu;beta*delta)
```

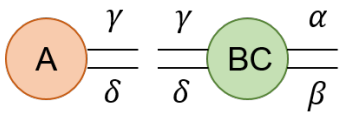
The treated `B` and `C` (or equivalently `B1` and `C1`) are matrices (i.e. rank-2 tensors), so the legs can be contracted via matrix multiplication. As mentioned in the previous tutorial, MATLAB is very efficient when it performs (standard) linear algebra operations. Let's contract `B1` and `C1` via the legs μ, and seprate the combined legs $\alpha \otimes \gamma$ and $\delta \otimes \beta$ into four legs $\alpha, \beta, \gamma, \delta$, as a diagram below.



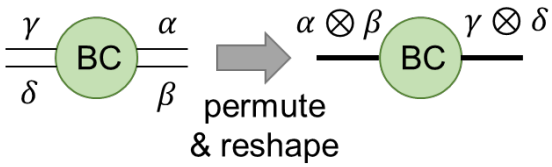

```
BC = B1*C1;% \sum_{mu} B(alpha*gamma,mu) * C(mu,beta*delta)
           % = BC(alpha*gamma,beta,delta)
% BC(alpha*gamma,beta*delta) -> BC(alpha,gamma,beta,delta)
BC = reshape(BC,[Da,Dc,Db,Dd]);
```

Contract BC and A

Current tensor networks looks like:

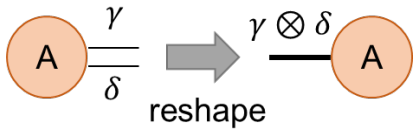


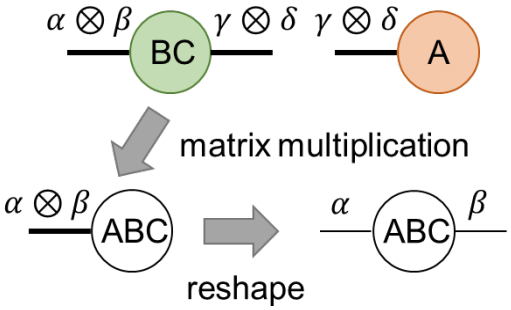
Reshape `BC` as a matrix, as the diagram below:




```
% BC(alpha,gamma,beta,delta) -> BC(alpha,beta,gamma,delta)
BC = permute(BC,[1,3,2,4]);
% BC(alpha,beta,gamma,delta) -> BC(alpha*beta;gamma*delta)
BC = reshape(BC,[Da*Db,Dc*Dd]);
```

Then reshape tensor `A` as a **vector**, and multiply with `BC`. By reshaping `ABC` as rank-2 tensor, we have rank-2 tensor `ABC`.






```
A1 = A(:);    % A(gamma,delta) -> A(gamma*delta)
% \sum_{gamma,delta} BC(alpha*beta,gamma*delta) * A(gamma*delta)
%       = ABC(alpha,beta)
ABC1 = BC*A1;
ABC1 = reshape(ABC1,[Da,Db]);% ABC(alpha*beta) -> ABC(alpha,beta)
```

How much time has been taken?

```
toc2(tobj,'-v');
```

```
Elapsed time: 0.0314s, CPU time: 0.08s, Avg # of cores: 2.547
```

Usually CPU time lapse is several times larger than real time lapse. It shows that MATLAB automatically parallelized computation.

Short remark: Why do we use matrix multiplication, instead of for-loops?

One may ask why we bother with reshaping and permuting tensors. So let's compare the computational efficiency between two approaches. First, below is the part of the above code contracting `B` and `C`.

```
% % Scheme 1: Tensor contraction using matrix multiplication
```

```
tobj = tic2;

B1 = permute(B,[1,3,2]); % B(alpha,mu,gamma) -> B(alpha,gamma,mu)
B1 = reshape(B1,[Da*Dc,Dm]);% B(alpha,gamma,mu) -> B(alpha*gamma,mu)
C1 = permute(C,[2,1,3]); % C(beta,mu,delta) -> C(mu,beta,delta)
C1 = reshape(C1,[Dm,Db*Dd]);% C(mu,beta,delta) -> C(mu,beta*delta)
% \sum_{mu} B(alpha*gamma,mu) * C(mu,beta*delta)
%       = BC(alpha*gamma,beta*delta)
BC = B1*C1;
% BC(alpha*gamma,beta*delta) -> BC(alpha,gamma,beta,delta)
BC = reshape(BC,[Da,Dc,Db,Dd]);

toc2(tobj,'-v');
```

```
Elapsed time: 0.01535s, CPU time: 0.02s, Avg # of cores: 1.303
```

Second, this is a contraction using for-loops.

```
% % Scheme 2: Tensor contraction using for-loops
tobj = tic2;

% create an 4D-array initialized with zeros
BC = zeros(Da,Dc,Db,Dd);
for it1 = (1:size(BC,1)) % alpha
    for it2 = (1:size(BC,2)) % gamma
        for it3 = (1:size(BC,3)) % beta
            for it4 = (1:size(BC,4)) % delta
                for it5 = (1:size(B,2)) % mu
                    BC(it1,it2,it3,it4) = ...
                        BC(it1,it2,it3,it4) + ...
                        B(it1,it5,it2)*C(it3,it5,it4);
                end
            end
        end
    end
end

toc2(tobj,'-v');
```

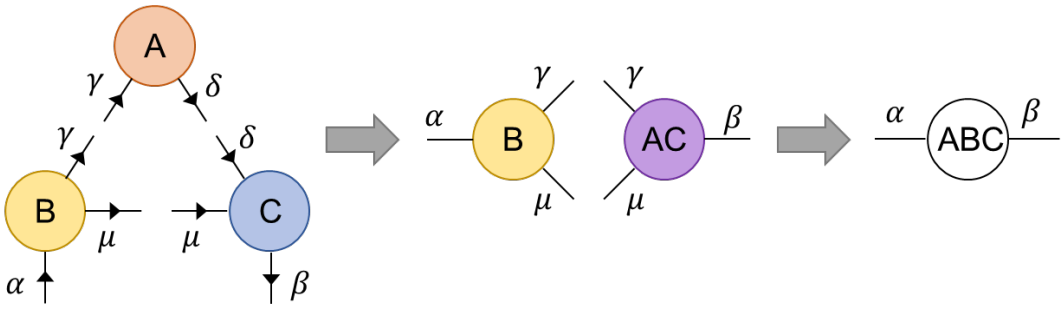
```
Elapsed time: 2.162s, CPU time: 3.39s, Avg # of cores: 1.568
```

We see that the latter scheme takes much longer time (> 100 times)!

In MATLAB, matrix operation is much faster than for-loops, since MATLAB implements a state-of-the-art linear algebra algorithm (which is, e.g., better parallelized). Try to avoid use for-loops for matrix or tensor operations as possible!

Exercise (a): First contract A and C, and then contract AC and B

Try a different order of the tensor contraction. Contract A and C first, then contract B, as a diagram below.



Write a script which implements this way of tensor contraction, and compare the computational time (both real time and CPU time). Which one is faster, by which factor?