Table of Contents

[TIP8419 - Algebra Linear e Multilinear]

Author: Lucas Abdalah

ND is a package developped for the Multilinear Algebra Course It is a shortcut for N-d array in reference to the homonym library in python

CONTENT

```
MATRIX OPERATIONS
    ND.VEC              - Vectorize a matrix.

TENSOR OPERATIONS
    ND.RANDN_COMPLEX    - Complex-valued
 array from normal distribution.
    ND.NMSE             - Normalized
 mean square error (NMSE) of a tensor.
    ND.SLICEORT         - Verify the orthogonality between the
  slices of a tensor

MATRIX PRODUCTS
    ND.HADAMARD_    - Hadamard product with two matrices.
    ND.KRON_        - Kronnecker product with two matrices.
    ND.KR_          - Khatri-Rao product with two matrices.

TENSOR FACTORS ESTIMATION
    ND.LSKRF        - Least-Squares Khatri-Rao Factorization (LSKRF)
    ND.LSKRONF      - Least-Squares
 Kronecker Product Factorization (LSKRONF)
    ND.KPSVD        - Kronecker
 Product Singular Value Decomposition (KPSVD)

TENSOR RESHAPE AND N-PRODUCT
    ND.UNFOLD       - Unfold a tensor into N-mode tensor (matrix)
    ND.FOLD         - Fold a N-mode tensor (matrix) into a tensor
    ND.N_MODE       - Compute the N-
mode product bewteen a tensor and factor matrices

TENSOR DECOMPOSTIONS
```

```
    ND.HOSVD        - Perfom
 the High Order Singular Value Decomposition (HOSVD) of a tensor,
 truncated or full version
    ND.HOOI         - Perfom
 the High Order Orthogonal Iteration (HOOI) of a tensor,
 truncated or full version
    ND.MLSKRF       - Perform the Multidimensional Least-
Squares Khatri-Rao Factorization (MLSKRF) of a tensor
    ND.MLSKRONF     -  Perform the Multidimensional Least-
Squares Kronecker Factorization (MLS-KronF) of a tensor

SAVE DATA TO TXT FILE
    ND.MAT2TXT      - Write a matrix X into a txt file
    ND.TENSOR2TXT   - Write a 3D tensor X into a txt file

PARAFAC/CP

classdef nd

methods(Static)
```

MATRIX OPERATIONS

```
    function y = vec(x)
        % ND.VEC - Vectorize a matrix.
        %   y = vec(x) draws a vector from a given matrix.
        %
        %   See also.
            y = x(:);
    end
```

TENSOR OPERATIONS

```
    function C = randn_complex(M, varargin)
    % ND.RANDN_COMPLEX - Complex-valued array from normal
 distribution.
    %   C = nd.randn_complex(M,N) draws a complex-valued array from
 normal
    %       distribution.
    %
    %   See also.
            C = complex(randn(M,varargin{:}), randn(M, varargin{:}));
    end


    function [X_nmse, X_nmse_dB] = nmse(X, X_hat)
    % ND.NMSE - Normalized mean square error (NMSE) of a tensor.
    %   [X_nmse, X_nmse_dB] = nd.nmse(X, X_hat) compute the NMSE of
 two arrays
    %
    %   See also.
        X_nmse = frob(X - X_hat)^2/(frob(X)^2);
        X_nmse_dB = 20*log10(X_nmse);
```

```
   end


   function f_ord = sliceort(Xten)
   % ND.SLICEORT - Verify the orthogonality between slices of a
tensor, by summing the
   %  the scalar between all the slices.
   %
   %   f_ord = sliceort(Xten) compute scalar product between tensor
slices
   %
   %   See also.
       size_Xten = size(Xten);
       f_ord = [];

       for kk_xT = 1:size_Xten(3)
           for kk_x = 1:size_Xten(3)
               if kk_xT ~= kk_x
                   f_ord(end+1) =
nd.vec(Xten(:,:,kk_xT))'*nd.vec(Xten(:,:,kk_x)) ;
               end
           end
       end
       f_ord = sum(f_ord);
   end
```

MATRIX PRODUCTS

```
   function [C, elaspedTime] = hadamard_(A, B)
   % ND.HADAMARD_  Hadamard product with two matrices.
   %   C = nd.hadamard_(A, B) compute the hadamard procuct.
   %
   %   [C, elaspedTime] = nd.hadamard_(A, B) compute the hadamard
   %   procuct elapsed time.
   %
   %   See also.
       tic;

       C = A.*B;

       elaspedTime = toc;
   end


   function [C, elaspedTime] = kron_(A, B)
   % ND.KRON_  Kronnecker product with two matrices.
   %   C = nd.kron_(A, B) compute the Kronnecker procuct.
   %
   %   [C, elaspedTime] = nd.kron_(A, B) compute the Kronnecker
   %   procuct elapsed time.
   %
   %   See also.
       tic;
```

```
    % [M_rows, N_columns] = size(B);
    % C = repelem(A, M_rows, N_columns).*repmat(B,[size(A)]);

    C = kron(A,B);

    elaspedTime = toc;
end


function [C, elaspedTime] = kr_(A, B)
% ND.KR_  Khatri-Rao product with two matrices.
%   C = nd.kr_(A, B) compute the Khatri-Rao procuct.
%
%   [C, elaspedTime] = nd.kr_(A, B) compute the Khatri-Rao
%   procuct elapsed time.
%
%   See also.
    tic;

    N = size(A,2);
    if N == size(B,2)
        P = reshape(A,1,[],N);
        Q = reshape(B,[],1,N);
        C = P.*Q;
        C = reshape(C,[],N);
    else
        error('number of columns should be equal')
    end

    elaspedTime = toc;
end
```

TENSOR FACTORS ESTIMATION

```
function [Ahat,Bhat] = lskrf(X, M, N)
% ND.LSKRF  Least-Squares Khatri-Rao Factorization (LSKRF)
%   [Ahat,Bhat] = nd.lskrf(X, M, N) compute the LSKRF.
%
%   See also.
    [iX, jX] = size(X);

    if iX == M*N % Verify the input dimensions
        Ahat = complex(zeros(M,jX),0);
        Bhat = complex(zeros(N,jX),0);

        for jj = 1:jX
            [U,S,V] = svd(reshape(X(:,jj), [N M]));
            Ahat(:,jj) = sqrt(S(1,1)).*conj(V(:,1));
            Bhat(:,jj) = sqrt(S(1,1)).*U(:,1);
        end
    else
        error('number of rows of X should be equal to size M*N');
```

```
       end

   end


   function [Ahat,Bhat] = lskronf(X, Ma, Na, Mb, Nb)
   % ND.LSKRONF  Least-Squares Kronecker Product Factorization
(LSKRONF)
   %   [Ahat,Bhat] = nd.lskronf(X, Ma, Na, Mb, Nb) compute the
LSKRONF.
   %
   %   See also.
       [Mx,Nx] = size(X);

       if Ma*Mb == Mx && Na*Nb == Nx % Verify the input dimensions
           Xhat = complex(zeros(Mb*Nb,Ma*Na),0);
           X_b = mat2cell(X, repelem(Mx/Ma,Ma), repelem(Nx/Na,Na));

           itCol = 1;
           for jj = 1:Na
               for ii = 1:Ma
                   Xhat(:,itCol) = nd.vec(cell2mat(X_b(ii,jj)));
                   itCol = itCol + 1;
               end
           end

           [U,S,V] = svd(Xhat);
           Ahat = reshape(sqrt(S(1,1)).*conj(V(:,1)),[Ma Na]);
           Bhat = reshape(sqrt(S(1,1)).*U(:,1), [Mb Nb]);

       else
           error('size of X(Mx, Nx) should match with Mc=Ma*Mb and
Nc=Na*Nb, A(Ma, Na) and B(Mb, Nb)');
       end
   end


   function [U,S,V,rkp] = kpsvd(X, Xstruct)
   % ND.KPSVD  Kronecker Product Singular Value Decomposition (KPSVD)
   %   [U,S,V,rkp] = nd.kpsvd(X, Xstruct) compute the KPSVD.
   %
   %   See also.
       [Mx,Nx] = size(X);

       if Xstruct(1)*Xstruct(3) == Mx && Xstruct(2)*Xstruct(4) ==
Nx % Verify the input dimensions
           Xhat =
complex(zeros(Xstruct(3)*Xstruct(4),Xstruct(1)*Xstruct(2)),0);
           X_b = mat2cell(X, repelem(Mx/Xstruct(1),Xstruct(1)),
repelem(Nx/Xstruct(2),Xstruct(2)));

           itCol = 1;
           for jj = 1:Xstruct(2)
               for ii = 1:Xstruct(1)
```

```
                Xhat(:,itCol) = nd.vec(cell2mat(X_b(ii,jj)));
                itCol = itCol + 1;
            end
        end
        [U,S,V] = svd(Xhat');
        rkp = rank(S);
    else
        error('size of X(Mx, Nx) should match with
Mc=Xstruct(1)*Xstruct(3) and Nc=Xstruct(2)*Xstruct(4), for
A(Xstruct(1), Xstruct(2)) and B(Xstruct(3), Xstruct(4))');
    end

end
```

TENSOR RESHAPE AND N-PRODUCT

```
function Xn = unfold(Xten,N_mode)
% ND.UNFOLD  Unfold a tensor into N-mode tensor (matrix)
%   Xn = unfold(Xten,N_mode) compute into N-mode tensor
%
%   See also.
    Xten_Size = size(Xten);
    reSort = 1:1:numel(Xten_Size); % prod(size(Xten_Size))
    reSort(N_mode) = [];
    Xn = reshape(permute(Xten,[N_mode reSort]), ...
                [], ...
                prod(Xten_Size)/Xten_Size(N_mode));
end


function Xten = fold(Xn,Xten_Size,N_mode)
% ND.FOLD  Fold a N-mode tensor (matrix) into a tensor
%   Xn = fold(Xn,Xten_Size,N_mode) fold a Xn into X tensor
%
%   See also.
    reSort = 1:1:numel(Xten_Size);
    reSort(N_mode) = [];
    reSort = [N_mode reSort];
    Xten = reshape(Xn,Xten_Size(reSort));

    switch N_mode
        case 1
            Xten = permute(Xten,reSort);
        otherwise
            reSort = 1:numel(Xten_Size);
            for ii = 2:N_mode
                reSort([ii-1, ii]) = reSort([ii, ii-1]);
            end
            Xten = permute(Xten,reSort);
    end
end
```

```
   function Yten = N_mode(Xten,factors,N_mode)
   % ND.N_MODE  Compute the N-mode product bewteen a tensor and
factor matrices
   %   Yten = N_mode(Xten,factors,N_mode) N-mode product bewteen a
tensor and matrices
   %
   %   See also.
       if nargin < 3
           N_mode = 1:numel(factors);
       end
       Xten_Size = size(Xten);
       for nIt = N_mode
           [Xten_Size(nIt), ~] = size(cell2mat(factors(nIt)));
           Yten =
nd.fold(cell2mat(factors(nIt))*nd.unfold(Xten,nIt), ...
                           Xten_Size, ...
                           nIt);
       end
   end
```

TENSOR DECOMPOSTIONS

```
   function [S,U] = hosvd(ten, Atype, ranksInput)
   % ND.HOSVD  Perfom the High Order Singular Value Decomposition
(HOSVD)
   %  of a tensor, truncated or full version.
   %   [S,U] = hosvd(ten, 'trunc') compute the truncated-HOSVD
   %   [S,U] = hosvd(ten, 'full') compute the full-HOSVD
   %
   %   See also.
       N = numel(size(ten));
       U = cell(N, 1);

       switch Atype
       case 'trunc'
               for i = 1:N
                   [Ur, Sr, ~] = svd(nd.unfold(ten,i));
                   if nargin < 3
                       Ur = Ur(:,1:rank(Sr));
                   else
                       Ur = Ur(:,1:ranksInput(i));
                   end
                   U{i} = Ur;
               end
       case 'full'
           for i = 1:N
               [Ur,~,~] = svd(nd.unfold(ten,i));
               U{i} = Ur;
           end
       end
       S = nd.N_mode(ten, (cellfun(@(x) x',
U,'UniformOutput',false)));
       U = cellfun(@(x) x, U, 'UniformOutput',false);
   end

   function [S, U, it] = hooi(ten, Atype, maxIt, ranksInput)
   % ND.HOOI  Perfom the High Order Orthogonal Iteration (HOOI)
   %   of a tensor, truncated or full version.
   %
   %   [S,U] = hooi(ten, 'trunc') compute the truncated-HOOI
   %   [S,U] = hooi(ten, 'full') compute the full-HOOI
   %
   %   See also.
       N = numel(size(ten));
       [~, U_ten] = nd.hosvd(ten, 'full');

       if nargin < 3
           maxIt = 20;
       end

       switch Atype
       case 'trunc'
           for it = 1:maxIt
               for ii = 1:N
                   N_mode = 1:N;
                   N_mode(ii) = [];
                   Un = nd.N_mode(ten, U_ten, N_mode);
                   [Ur, Sr, ~] = svd(nd.unfold(Un,ii));
                   if nargin < 3
                       U{ii} = Ur(:,1:rank(Sr));
                   else
                       U{ii} = Ur(:,1:ranksInput(ii));
                   end
               end
           end
       case 'full'
           for it = 1:maxIt
               for ii = 1:N
                   N_mode = 1:N;
                   N_mode(ii) = [];
                   Un = nd.N_mode(ten, U_ten, N_mode);
                   [Usvd,~,~] = svd(nd.unfold(Un,ii));
                   U{ii} = Usvd;
               end
           end
       end
       S = nd.N_mode(ten, cellfun(@(x) x', U, 'UniformOutput',
false)) ;
   end


   function factors = mlskrf(X, N_mode, order)
   % ND.MLSKRF  Perform the Multidimensional Least-Squares Khatri-Rao
   %   Factorization (MLSKRF) of a tensor.
   %
   %   factors = mlskrf(X, N_mode, order) compute the MLSKRF of a
tensor
```

```
   %
   %   See also.
       [~,R] = size(X);
       factors = cell(N_mode, 1);
       factors_r = cell(R, N_mode);
       for rr = 1:R
           [Sr,Ur] = nd.hosvd(reshape(X(:,rr), flip(order)), 'full');
           for nn = 1:N_mode
               sr = (Sr(1)^(1/N_mode));
               ur = Ur{N_mode-nn+1}(:,1);
               factors_r{rr,nn} = sr*ur;
           end
       end
       for n = 1:N_mode
          factors{n} = reshape(cell2mat(factors_r(:,n)) ,[order(n)
R]);
       end
   end

   function Ahat = mlskronf(X, rowsInput, colsInput, Atype)
   % ND.MLSKRONF  Perform the Multidimensional Least-Squares
Kronecker
   %   Factorization (MLS-KronF) of a tensor.
   %
   %   factors = mlskronf(X, rowsInput, colsInput, Atype) compute the
MLSKRF of a tensor
   %
   %   See also.
       dim = {repelem(rowsInput(2)*rowsInput(3), 1, rowsInput(1));
repelem(colsInput(2)*colsInput(3), 1, colsInput(1))};
       Xb = mat2cell(X,dim{1},dim{2});
       Inv = {flip(rowsInput), flip(colsInput)};
       K = 1;

       for jA = 1:colsInput(1)
           for iA = 1:rowsInput(1)
               dim = {repelem(rowsInput(3), 1, rowsInput(2)),
repelem(colsInput(3), 1, colsInput(2))};
               X_bc = mat2cell(cell2mat(Xb(iA,jA)), dim{1}, dim{2});
               for jB = 1:colsInput(2)
                   for iB = 1:rowsInput(2)
                       vb(:,iB,jB) = nd.vec(cell2mat(X_bc(iB,jB)));
                   end
               end
               Xhat(:,K) = reshape(vb,[],1);
               K = K + 1;
           end
       end

       switch Atype
       case 'hosvd'
           [S,U] = nd.hosvd(reshape(Xhat, flip(rowsInput.*
colsInput)), 'full');
       case 'hooi'
```

```
           [S,U] = nd.hooi(reshape(Xhat, flip(rowsInput.*
colsInput)), 'full');
       end

       UN = length(U);

       for u = 1:UN
           Ahat{UN - u + 1} = reshape((S(1)^(1/length(U)))*U{u}(:,1),
[Inv{1}(u) Inv{2}(u)]);
       end
   end
```

SAVE DATA TO TXT FILE

```
   function mat2txt(file, X, permission, header)
   % ND.MAT2TXT  Write a matrix X into a txt file
   %   mat2txt(file, X, 'w', header) - Overwite the file
   %   mat2txt(file, X, 'a', header) - Append to the file end
   %
   %   See also.
       [I, J] = size(X);
       fileID = fopen(file, permission);
       fprintf(fileID, [repelem('-', strlength(header)+3), '\n',
header, ...
               '\n', repelem('-', strlength(header)+3), '\n']);
       fprintf(fileID, 'X(%d, %d)\n', I, J);
           for ii = 1:I
               for jj = 1:J
                   fprintf(fileID, ' %2.0f', X(ii,jj));
               end
               fprintf(fileID, ';\n');
           end
       fprintf(fileID, '\n');
       fclose(fileID);
   end


   function tensor2txt(file, X, permission, header)
   % ND.TENSOR2TXT  Write a 3D tensor X into a txt file
   %   tensor2txt(file, X, 'w', header) - Overwite the file
   %   tensor2txt(file, X, 'a', header) - Append to the file end
   %
   %   See also.
       [I, J, K] = size(X);

       fileID = fopen(file, permission);

       fprintf(fileID, [repelem('-', strlength(header)+3), '\n',
header, ...
       '\n', repelem('-', strlength(header)+3), '\n']);

       for kk = 1:K
           fprintf(fileID, 'X(:, :, %d)\n', kk);
```

```
            for ii = 1:I
                for jj = 1:J
                    fprintf(fileID, ' %2.0f', X(ii,jj,kk));
                end
                fprintf(fileID, ';\n');
            end
            fprintf(fileID, '\n');
        end
        fclose(fileID);
    end
```

PLACE HOLDER SECTION

```
end

end
```

```
ans =

  nd with no properties.
```

Published with MATLAB® R2021a