N-Dimensional Lists (ndlist)

Version 0.3

Alex Baker

https://github.com/ambaker1/ndlist

October 14, 2023

Abstract

The "ndlist" package is a pure-Tcl package for tensor manipulation and processing.

This package is also a Tin package, and can be loaded in as shown below:

Example 1: Installing and loading "ndlist"

Code:

```
package require tin
tin add -auto ndlist https://github.com/ambaker1/ndlist install.tcl
tin import ndlist
```

1-Dimensional Lists (Vectors)

Lists are foundational to Tcl, so in addition to providing utilities for ND-lists, this package also provides utilities for working with 1D-lists, or vectors.

Range Generator

The command *range* simply generates a range of integer values. This can be used in conjunction with the Tcl *foreach* loop to simplify writing "for" loops. There are two ways of calling this command, as shown below.

```
range $n
range $start $stop <$step>
```

$n	Number of indices, starting at 0 (e.g. 3 returns 0 1 2).
$start	Starting value.
$stop	Stop value.
$step	Step size. Default 1 or -1, depending on direction of start to stop.

Example 2: Integer range generation

Code:

```
puts [range 3]
puts [range 0 2]
puts [range 10 3 -2]
```

Output:

```
0 1 2
0 1 2
10 8 6 4
```

Example 3: Simpler for-loop

Code:

```
foreach i [range 3] {
    puts $i
}
```

Output:

```
0
1
2
```

Logical Indexing

The command *find* returns the indices of non-zero elements of a boolean list, or indices of elements that satisfy a given criterion. Can be used in conjunction with *nget* to perform logical indexing.

```
find $list <$op $scalar>
```

`$list`	List of values to compare.
`$op`	Comparison operator. Default "!=".
`$scalar`	Comparison value. Default 0.

Example 4: Filtering a list

Code:

```
set x {0.5 2.3 4.0 2.5 1.6 2.0 1.4 5.6}
puts [nget $x [find $x > 2]]
```

Output:

```
2.3 4.0 2.5 5.6
```

Linear Interpolation

The command *linterp* performs linear 1D interpolation. Converts input to "double".

```
linterp $x $xList $yList
```

`$x`	Value to query in `$xList`
`$xList`	List of x points, strictly increasing
`$yList`	List of y points, same length as `$xList`

Example 5: Linear interpolation

Code:

```
puts [linterp 2 {1 2 3} {4 5 6}]
puts [linterp 8.2 {0 10 20} {2 -4 5}]
```

Output:

```
5.0
-2.92
```

Vector Generation

The command *linspace* can be used to generate a vector of specified length and equal spacing between two specified values. Converts input to "double"

```
linspace $n $start $stop
```

$n	Number of points
$start	Starting value
$stop	End value

Example 6: Linearly spaced vector generation

Code:

```
puts [linspace 5 0 1]
```

Output:

```
0.0 0.25 0.5 0.75 1.0
```

The command *linsteps* generates intermediate values given an increment size and a sequence of targets. Converts input to "double".

```
linsteps $step $x1 $x2 ...
```

$step	Maximum step size
$x1 $x2 ...	Targets to hit.

Example 7: Intermediate value vector generation

Code:

```
puts [linsteps 0.25 0 1 0]
```

Output:

```
0.0 0.25 0.5 0.75 1.0 0.75 0.5 0.25 0.0
```

Functional Mapping

The command *lapply* simply applies a command over each element of a list, and returns the result. Basic math operators can be mapped over a list with the command *lop*.

```
lapply $command $list $arg ...
```

```
lop $list $op $arg...
```

$list	List to map over.
$command	Command prefix to map with.
$op	Math operator (see ::tcl::mathop documentation).
$arg ...	Additional arguments to append to command after each list element.

Example 8: Applying a math function to a list

Code:

```
# Add Tcl math functions to the current namespace path
namespace path [concat [namespace path] ::tcl::mathfunc]
puts [lapply abs {-5 1 2 -2}]
```

Output:

```
5 1 2 2
```

Mapping Over Two Lists

The commands *lapply* and *lop* only map over one list. The commands *lapply2* and *lop2* allow you to map, element-wise, over two lists. List lengths must be equal.

```
lapply2 $command $list1 $list2 $arg ...
```

```
lop2 $list1 $op $list2 $arg...
```

`$list1 $list2`	Lists to map over, element-wise.
`$command`	Command prefix to map with.
`$op`	Math operator (see ::tcl::mathop documentation).
`$arg ...`	Additional arguments to append to command after list elements.

Example 9: Mapping over two lists

Code:

```
lapply puts [lapply2 {format "%s %s"} {hello goodbye} {world moon}]
```

Output:

```
hello world
goodbye moon
```

Example 10: Adding two lists together

Code:

```
puts [lop2 {1 2 3} + {2 3 2}]
```

Output:

```
3 5 5
```

List Math

The Tcl command *lmap* allows you to loop over an arbitrary number of lists in parallel, evaluating a script at each iteration, and collecting the results of each loop iteration into a new list. The command *lexpr* is an extension of this concept, just calling *lmap* and passing the input through the Tcl *expr* command.

```
lexpr $varList $list <$varList $list ...> $expr
```

`$varList ...`	List(s) of variables to iterate with.
`$list ...`	List(s) to iterate over.
`$expr`	Tcl expression to evaluate at every loop iteration.

Example 11: Filtering a list

Code:

```
set numbers [range 10]
set odds [lexpr x $numbers {$x % 2 ? $x : [continue]}]; # only odd numbers
puts $odds
```

Output:

```
1 3 5 7 9
```

Example 12: Adding three lists together

Code:

```
set x {1 2 3}
set y {2 9 2}
set z {5 -2 0}
puts [lexpr xi $x yi $y zi $z {$xi + $yi + $zi}]
```

Output:

```
8 9 5
```

List Statistics

The commands *max*, *min*, *sum*, *product*, *mean*, *median*, *variance*, and *stdev* compute the maximum, minimum, sum, product, mean, median, variance, and standard deviation of values in a list. For more advanced statistics, check out the Tcllib math::statistics package.

```
max $list
```

```
min $list
```

```
sum $list
```

```
product $list
```

```
mean $list
```

```
median $list
```

```
variance $list <$pop>
```

```
stdev $list <$pop>
```

$list	List (at least length 1) to compute statistic of.
$pop	Compute population statistic instead of sample statistic. Default false.

Example 13: List Statistics

Code:

```
set list {-5 3 4 0}
foreach stat {max min sum product mean median variance stdev} {
    puts [list $stat [$stat $list]]
}
```

Output:

```
max 4
min -5
sum 2
product 0
mean 0.5
median 1.5
variance 16.333333333333332
stdev 4.041451884327381
```

Vector Algebra

The dot product of two equal length vectors can be computed with *dot*. The cross product of two vectors of length 3 can be computed with *cross*.

```
dot $a $b
```

```
cross $a $b
```

$a	First vector.
$b	Second vector.

Example 14: Dot and cross product

Code:

```
set x {1 2 3}
set y {-2 -4 6}
puts [dot $x $y]
puts [cross $x $y]
```

Output:

```
8
24 -12 0
```

The norm, or magnitude, of a vector can be computed with *norm*.

```
norm $a <$p>
```

$a	Vector to compute norm of.
$p	Norm type. 1 is sum of absolute values, 2 is euclidean distance, and Inf is absolute maximum value. Default 2.

Example 15: Normalizing a vector

Code:

```
set x {3 4}
set x [lop $x / [norm $x]]
puts $x
```

Output:

```
0.6 0.8
```

For more advanced vector algebra routines, check out the Tcllib math::linearalgebra package.

2-Dimensional Lists (Matrices)

A matrix is a two-dimensional list, or a list of row vectors. This is consistent with the format used in the Tcllib math::linearalgebra package. See the example below for how matrices are interpreted.

$$A = \begin{bmatrix} 2 & 5 & 1 & 3 \\ 4 & 1 & 7 & 9 \\ 6 & 8 & 3 & 2 \\ 7 & 8 & 1 & 4 \end{bmatrix}, \quad B = \begin{bmatrix} 9 \\ 3 \\ 0 \\ -3 \end{bmatrix}, \quad C = \begin{bmatrix} 3 & 7 & -5 & -2 \end{bmatrix}$$

Example 16: Matrices and vectors

Code:

```
set A {{2 5 1 3} {4 1 7 9} {6 8 3 2} {7 8 1 4}}
set B {9 3 0 -3}
set C {{3 7 -5 -2}}
```

Identity Matrix

The command *eye* generates an identity matrix of a specified size.

```
eye $n
```

$n	Size of identity matrix

Example 17: Generating an identity matrix

Code:

```
puts [eye 3]
```

Output:

```
{1 0 0} {0 1 0} {0 0 1}
```

Matrix Transpose

The command *transpose* simply swaps the rows and columns of a matrix.

```
transpose $A
```

$A	Matrix to transpose, nxm.

Returns an mxn matrix.

Example 18: Transposing a matrix

Code:

```
puts [transpose {{1 2} {3 4}}]
```

Output:

```
{1 3} {2 4}
```

Matrix Multiplication

The command *matmul* performs matrix multiplication for two matrices. Inner dimensions must match.

```
matmul $A $B
```

$A	Left matrix, nxq.
$B	Right matrix, qxm.

Returns an nxm matrix (or the corresponding dimensions from additional matrices)

Example 19: Multiplying a matrix

Code:

```
puts [matmul {{2 5 1 3} {4 1 7 9} {6 8 3 2} {7 8 1 4}} {9 3 0 -3}]
```

Output:

```
24 12 72 75
```

Iteration Tools

The commands *zip* zips two lists into a list of tuples, and *zip3* zip three lists into a list of triples. Lists must be the same length.

```
zip $a $b
```

```
zip3 $a $b $c
```

`$a $b $c`	Lists to zip together.

Example 20: Zipping lists

Code:

```
puts [zip {A B C} {1 2 3}]
puts [zip3 {Do Re Mi} {A B C} {1 2 3}]
```

Output:

```
{A 1} {B 2} {C 3}
{Do A 1} {Re B 2} {Mi C 3}
```

The command *cartprod* computes the Cartesian product of an arbitrary number of vectors, returning a matrix where the columns correspond to the input vectors and the rows correspond to all the combinations of the vector elements.

```
cartprod $list1 $list2 ...
```

`$list1 $list2 ...`	Lists, or vectors, to take Cartesian product of.

Example 21: Cartesian product

Code:

```
puts [cartprod {A B C} {1 2 3}]
```

Output:

```
{A 1} {A 2} {A 3} {B 1} {B 2} {B 3} {C 1} {C 2} {C 3}
```

N-Dimensional Lists (Tensors)

A ND-list is defined as a list of equal length (N-1)D-lists, which are defined as equal length (N-2)D-lists, and so on until (N-N)D-lists, which are scalars of arbitrary size. This definition is flexible, and allows for different interpretations of the same data. For example, the list "1 2 3" can be interpreted as a scalar with value "1 2 3", a vector with values "1", "2", and "3", or a matrix with row vectors "1", "2", and "3".

The command *ndlist* validates that the input is a valid ND-list. If the input value is "ragged", as in it has inconsistent dimensions, it will throw an error. In general, if a value is a valid for N dimensions, it will also be valid for dimensions 0 to N-1. All other ND-list commands assume a valid ND-list.

```
ndlist $nd $value
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$value	List to interpret as an ndlist

Shape and Size

The commands *nshape* and *nsize* return the shape and size of an ND-list, respectively. The shape is a list of the dimensions, and the size is the product of the shape.

```
nshape $nd $ndlist <$axis>
```

```
nsize $nd $ndlist
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$ndlist	ND-list to get dimensions of.
$axis	Axis to get dimension along. Blank for all.

Example 22: Getting shape and size of an ND-list

Code:

```
set A [ndlist 2D {{1 2 3} {4 5 6}}]
puts [nshape 2D $A]
puts [nsize 2D $A]
```

Output:

```
2 3
6
```

Initialization

The command *nfull* initializes a valid ND-list of any size filled with a single value.

```
nfull $value $n ...
```

`$value`	Value to repeat
`$n ...`	Shape (list of dimensions) of ND-list.

Example 23: Generate ND-list filled with one value

Code:

```
puts [nfull foo 3 2]; # 3x2 matrix filled with "foo"
puts [nfull 0 2 2 2]; # 2x2x2 tensor filled with zeros
```

Output:

```
{foo foo} {foo foo} {foo foo}
{{0 0} {0 0}} {{0 0} {0 0}}
```

The command *nrand* initializes a valid ND-list of any size filled with random values between 0 and 1.

```
nrand $n ...
```

`$n ...`	Shape (list of dimensions) of ND-list.

Example 24: Generate random matrix

Code:

```
expr {srand(0)}; # resets the random number seed (for the example)
puts [nrand 1 2]; # 1x2 matrix filled with random numbers
```

Output:

```
{0.013469574513598146 0.3831388500440581}
```

Repeating and Expanding

The command *nrepeat* repeats portions of an ND-list a specified number of times.

```
nrepeat $ndlist $n ...
```

$value	Value to repeat
$n ...	Repetitions at each level.

Example 25: Repeat elements of a matrix

Code:

```
puts [nrepeat {{1 2} {3 4}} 1 2]
```

Output:

```
{1 2 1 2} {3 4 3 4}
```

The command *nexpand* repeats portions of an ND-list to expand to new dimensions. New dimensions must be divisible by old dimensions. For example, 1x1, 2x1, 4x1, 1x3, 2x3 and 4x3 are compatible with 4x3.

```
nexpand $ndlist $n ...
```

$ndlist	ND-list to expand.
$n ...	New dimensions of ND-list.

Example 26: Expand an ND-list to new dimensions

Code:

```
puts [nexpand {1 2 3} 3 2]
puts [nexpand {{1 2}} 2 4]
```

Output:

```
{1 1} {2 2} {3 3}
{1 2 1 2} {1 2 1 2}
```

Flattening and Reshaping

The command *nreshape* reshapes a vector into a compatible shape. Vector length must equal ND-list size.

```
nreshape $vector $n ...
```

$vector	Vector (1D-list) to reshape.
$n ...	Shape (list of dimensions) of ND-list.

Example 27: Reshape a vector to a matrix

Code:

```
puts [nreshape {1 2 3 4 5 6} 2 3]
```

Output:

```
{1 2 3} {4 5 6}
```

The inverse is *nflatten*, which flattens an ND-list to a vector, which can be then used with *nreshape*.

```
nflatten $nd $ndlist
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$ndlist	ND-list to flatten.

Example 28: Reshape a matrix to a 3D tensor

Code:

```
set x [nflatten 2D {{1 2 3 4} {5 6 7 8}}]
puts [nreshape $x 2 2 2]
```

Output:

```
{{1 2} {3 4}} {{5 6} {7 8}}
```

Index Notation

This package provides generalized n-dimensional list access/modification commands, using an index notation parsed by the command *::ndlist::ParseIndex*, which returns the index type and an index list for the type.

::ndlist::ParseIndex $n $input

$n	Number of elements in list.
$input	Index input. Options are shown below:
*** or :**	All indices
$start:$stop	Range of indices (e.g. 0:4 or 1:end-2).
$start:$step:$stop	Stepped range of indices (e.g. 0:2:-2 or 2:3:end).
$iList	List of indices (e.g. {0 end-1 5} or 3).
$i.	Single index with a dot, "flattens" the ndlist (e.g. 0. or end-3.).

Additionally, indices get passed through the *::ndlist::Index2Integer* command, which converts the inputs "end", "end±integer", "integer±integer" and negative wrap-around indexing (where -1 is equivalent to "end") into normal integer indices. Note that this command will return an error if the index is out of range.

::ndlist::Index2Integer $n $index

$n	Number of elements in list.
$index	Single index.

Example 29: Index Notation

Code:

```
set n 10
puts [::ndlist::ParseIndex $n *]
puts [::ndlist::ParseIndex $n 1:8]
puts [::ndlist::ParseIndex $n 0:2:6]
puts [::ndlist::ParseIndex $n {0 5 end-1}]
puts [::ndlist::ParseIndex $n end.]
```

Output:

```
A {}
R {1 8}
L {0 2 4 6}
L {0 5 8}
S 9
```

Access

Portions of an ND-list can be accessed with the command *nget*, using the index parser *::ndlist::ParseIndex* for each dimension being indexed. Note that unlike the Tcl *lindex* and *lrange* commands, *nget* will return an error if the indices are out of range.

```
nget $ndlist $i ...
```

$ndlist	ND-list value.
$i ...	Index inputs, parsed with *::ndlist::ParseIndex*. The number of index arguments determines the interpreted dimensions.

Example 30: ND-list access

Code:

```
set A {{1 2 3} {4 5 6} {7 8 9}}
puts [nget $A 0 *]; # get row matrix
puts [nget $A 0. *]; # flatten row matrix to a vector
puts [nget $A 0:1 0:1]; # get matrix subset
puts [nget $A end:0 end:0]; # can have reverse ranges
puts [nget $A {0 0 0} 1.]; # can repeat indices
```

Output:

```
{1 2 3}
1 2 3
{1 2} {4 5}
{9 8 7} {6 5 4} {3 2 1}
2 2 2
```

Modification

A ND-list can be modified by reference with *nset*, and by value with *nreplace*, using the index parser *::ndlist::ParseIndex* for each dimension being indexed. Note that unlike the Tcl *lset* and *lreplace* commands, the commands *nset* and *nreplace* will return an error if the indices are out of range. If all the index inputs are "*" except for one, and the replacement list is blank, it will delete values along that axis by calling *nremove*. Otherwise, the replacement ND-list must be expandable to the target index dimensions.

```
nset $varName $i ...  $sublist
```

```
nreplace $ndlist $i ...  $sublist
```

$varName	Variable that contains an ND-list (must exist).
$ndlist	ND-list to modify.
$i ...	Index inputs, parsed with *::ndlist::ParseIndex*. The number of index inputs determines the interpreted dimensions.
$sublist	Replacement list, or blank to delete values.

Example 31: Swapping rows in a matrix

Code:

```
# ND-list Value Modification
set a {{1 2} {3 4} {5 6}}
nset a {1 0} * [nget $a {0 1} *]; # Swap rows and columns (modify by reference)
puts $a
```

Output:

```
{3 4} {1 2} {5 6}
```

Removal

The command *nremove* removes portions of an ND-list at a specified axis.

```
nremove $nd $ndlist $i <$axis>
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$ndlist	ND-list to modify.
$i	Index input, parsed with *::ndlist::ParseIndex*.
$axis	Axis to remove at. Default 0.

Example 32: Deleting a column from a matrix

Code:

```
set a {{1 2 3} {4 5 6} {7 8 9}}
puts [nremove $a 2 1]; # Delete column 2
```

Output:

```
{1 2} {4 5} {7 8}
```

Example 33: Removing list elements that satisfy criteria

Code:

```
set x [range 10]
puts [nremove $x [find $x > 4]]
```

Output:

```
0 1 2 3 4
```

Insertion and Concatenation

The command *ninsert* allows you to insert an ND-list into another ND-list at a specified index and axis, as long as the ND-lists agree in dimension at all other axes. If "end" or "end-integer" is used for the index, it will insert after the index. Otherwise, it will insert before the index.

```
ninsert $nd $ndlist1 $index $ndlist2 <$axis>
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$ndlist1 $ndlist2	ND-lists to combine.
$index	Index to insert at.
$axis	Axis to insert at (default 0).

The command *nstack* is shorthand for inserting at "end", and concatenates two ND-lists.

```
nstack $nd $ndlist1 $ndlist2 <$axis>
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$ndlist1 $ndlist2	ND-lists to concatenate.
$axis	Axis to concatenate at (default 0).

Example 34: Inserting a column into a matrix

Code:

```
set matrix {{1 2} {3 4} {5 6}}
set column {A B C}
puts [ninsert 2D $matrix 1 $column 1]
```

Output:

```
{1 A 2} {3 B 4} {5 C 6}
```

Example 35: Concatenate tensors

Code:

```
set x [nreshape {1 2 3 4 5 6 7 8 9} 3 3 1]
set y [nreshape {A B C D E F G H I} 3 3 1]
puts [nstack 3D $x $y 2]
```

Output:

```
{{1 A} {2 B} {3 C}} {{4 D} {5 E} {6 F}} {{7 G} {8 H} {9 I}}
```

Changing Order of Axes

The command *nswapaxes* is a general purpose transposing function that swaps the axes of an ND-list. For simple matrix transposing, the command *transpose* can be used instead.

```
nswapaxes $ndlist $axis1 $axis2
```

$ndlist	ND-list to manipulate.
$axis1 $axis2	Axes to swap.

The command *nmoveaxis* moves a specified source axis to a target position. For example, moving axis 0 to position 2 would change "i,j,k" to "j,k,i".

```
nmoveaxis $ndlist $source $target
```

$ndlist	ND-list to manipulate.
$source	Source axis.
$target	Target position.

The command *npermute* is more general purpose, and defines a new order for the axes of an ND-list. For example, the axis list "1 0 2" would change "i,j,k" to "j,i,k".

```
npermute $ndlist $axis ...
```

$ndlist	ND-list to manipulate.
$axis ...	List of axes defining new order.

Example 36: Changing tensor axes

Code:

```
set x {{{1 2} {3 4}} {{5 6} {7 8}}}
set y [nswapaxes $x 0 2]
set z [nmoveaxis $x 0 2]
puts [lindex $x 0 0 1]
puts [lindex $y 1 0 0]
puts [lindex $z 0 1 0]
```

Output:

```
2
2
2
```

ND Functional Mapping

The command *napply* simply applies a command over each element of an ND-list, and returns the result. Basic math operators can be mapped over an ND-list with the command *nop*, which is a special case of *napply*, using the ::tcl::mathop namespace.

```
napply $nd $command $ndlist $arg ...
```

```
nop $nd $ndlist $op $arg...
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$ndlist	ND-list to map over.
$command	Command prefix to map with.
$op	Math operator (see ::tcl::mathop documentation).
$arg ...	Additional arguments to append to command after ND-list element.

Example 37: Chained functional mapping over a matrix

Code:

```
napply 2D puts [napply 2D {format %.2f} [napply 2D expr {{1 2} {3 4}} + 1]]
```

Output:

```
2.00
3.00
4.00
5.00
```

Example 38: Element-wise operations

Code:

```
puts [nop 1D {1 2 3} + 1]
puts [nop 2D {{1 2 3} {4 5 6}} > 2]
```

Output:

```
2 3 4
{0 0 1} {1 1 1}
```

Mapping Over Two ND-lists

The commands *napply* and *nop* only map over one ND-list. The commands *napply2* and *nop2* allow you to map, element-wise, over two ND-lists. If the input lists have different shapes, they will be expanded to their maximum dimensions with *nexpand* (if compatible).

```
napply2 $nd $command $ndlist1 $ndlist2 $arg ...
```

```
nop2 $nd $ndlist1 $op $ndlist2 $arg...
```

$nd	Dimensionality of ND-list (e.g. 2D for a matrix).
$ndlist1 $ndlist2	ND-lists to map over, element-wise.
$command	Command prefix to map with.
$op	Math operator (see ::tcl::mathop documentation).
$arg ...	Additional arguments to append to command after ND-list elements.

Example 39: Format columns of a matrix

Code:

```
set data {{1 2 3} {4 5 6} {7 8 9}}
set formats {{%.1f %.2f %.3f}}
puts [napply2 2D format $formats $data]
```

Output:

```
{1.0 2.00 3.000} {4.0 5.00 6.000} {7.0 8.00 9.000}
```

Example 40: Adding matrices together

Code:

```
set A {{1 2} {3 4}}
set B {{4 9} {3 1}}
puts [nop2 2D $A + $B]
```

Output:

```
{5 11} {6 5}
```

Reducing an ND-list

The command *nreduce* combines *nmoveaxis* and *napply* to reduce an axis of an ND-list with a function that reduces a vector to a scalar, like *max* or *sum*.

```
nreduce $nd $command $ndlist <$axis>
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$command	Command prefix to map with.
$ndlist	ND-list to map over.
$axis	Axis to reduce. Default 0.

Example 41: Matrix row and column statistics

Code:

```
set x {{1 2} {3 4} {5 6} {7 8}}
puts [nreduce 2D max $x]; # max of each column
puts [nreduce 2D max $x 1]; # max of each row
puts [nreduce 2D sum $x]; # sum of each column
puts [nreduce 2D sum $x 1]; # sum of each row
```

Output:

```
7 8
2 4 6 8
16 20
3 7 11 15
```

Generalized N-Dimensional Mapping

The command *nmap* is a general purpose mapping function for N-dimensional lists in Tcl, and the command *nexpr* a special case for math expressions. If multiple ND-lists are provided for iteration, they must be expandable to their maximum dimensions. The actual implementation flattens all the ND-lists and calls the Tcl *lmap* command, and then reshapes the result to the target dimensions. So, if "continue" or "break" are used, it will return an error; it cannot be used for filtering.

```
nmap $nd $varName $ndlist <$varName $ndlist ...> $body
```

```
nexpr $nd $varName $ndlist <$varName $ndlist ...> $expr
```

$nd	Dimensionality of ND-list (e.g. 2D, 2d, or 2 for a matrix).
$varName	Variable name to iterate with.
$ndlist	ND-list to iterate over.
$body	Tcl script to evaluate at every loop iteration.
$expr	Tcl expression to evaluate at every loop iteration.

Example 42: Expand and map over matrices

Code:

```
set phrases [nmap 2D greeting {{hello goodbye}} subject {world moon} {
    list $greeting $subject
}]
napply 2D puts $phrases
```

Output:

```
hello world
goodbye world
hello moon
goodbye moon
```

Example 43: Adding two matrices together, element-wise

Code:

```
set x {{1 2} {3 4}}
set y {{4 1} {3 9}}
set z [nexpr 2D xi $x yi $y {$xi + $yi}]
puts $z
```

Output:

```
{5 3} {6 13}
```

Command Index

::ndlist::Index2Integer, 17
::ndlist::ParseIndex, 17

cartprod, 12
cross, 9

dot, 9

eye, 10

find, 3

lapply, 5
lapply2, 6
lexpr, 7
linspace, 4
linsteps, 4
linterp, 3
lop, 5
lop2, 6

matmul, 11
max, 8
mean, 8
median, 8
min, 8

napply, 23
napply2, 24
ndlist, 13
nexpand, 15
nexpr, 26
nflatten, 16
nfull, 14
nget, 18
ninsert, 21
nmap, 26
nmoveaxis, 22
nop, 23
nop2, 24
norm, 9
npermute, 22
nrand, 14
nreduce, 25
nremove, 20
nrepeat, 15
nreplace, 19
nreshape, 16
nset, 19
nshape, 13
nsize, 13
nstack, 21
nswapaxes, 22

product, 8

range, 2

stdev, 8
sum, 8

transpose, 11

variance, 8

zip, 12
zip3, 12