

N-Dimensional Lists (ndlist)

Version 0.3

Alex Baker

<https://github.com/ambaker1/ndlist>

October 12, 2023

Abstract

The “ndlist” module provides tools for vector, matrix, and tensor manipulation and processing, where vectors are represented by Tcl lists, and matrices are represented by nested Tcl lists, and higher dimension lists represented by additional levels of nesting.

List Utilities

Lists are foundational to Tcl, so in addition to providing n-dimensional list utilities, this package also provides utilities for working with 1-dimensional lists.

Range Generator

The command *range* simply generates a range of integer values. 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 1: 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
```

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 2: 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 $xq $xp $yp
```

\$xq	Value to query in \$xp
\$xp	Vector of x points, strictly increasing
\$yp	Vector of y points, same length as \$xp

Example 3: 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
```

Generate Linearly Spaced Vector

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 $x1 $x2 $n
```

\$x1	Starting value
\$x2	End value
\$n	Number of points

Example 4: Linearly spaced vector generation

Code:

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

Output:

```
0.0 0.25 0.5 0.75 1.0
```

Generate Fixed-Spacing Vector

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

```
linspace $step $x1 $x2 ...
```

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

Example 5: Intermediate value vector generation

Code:

```
puts [linspace 0.25 0 1 0]
```

Output:

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

Mapping Over a List

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*. These are effectively equivalent to the 1D versions of *napply* and *nop*.

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

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

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

Example 6: 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...
```

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

Example 7: Mapping over two lists

Code:

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

Output:

```
hello world
goodbye moon
```

Example 8: Adding two lists together

Code:

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

Output:

```
3 5 5
```

Extreme Values

The commands *max* and *min* compute the maximum and minimum values of a list.

```
max $list
```

```
min $list
```

\$list List (at least length 1) to compute statistic of.

Example 9: Extreme values

Code:

```
puts [max {-5 3 4 0}]  
puts [min {-5 3 4 0}]
```

Output:

```
4  
-5
```

Sum and Product

The commands *sum* & *product* compute the sum and product of a list.

```
sum $list
```

```
product $list
```

\$list List (at least length 1) to compute statistic of.

Example 10: Sum and product of a list

Code:

```
puts [sum {-5 3 4 0}]  
puts [product {-5 3 4 0}]
```

Output:

```
2  
0
```

Average Values

The commands *mean* & *median* calculate the mean and median of of a list.

```
mean $list
```

```
median $list
```

\$vector List (at least length 1) to compute statistic of.

Example 11: Mean and median

Code:

```
puts [mean {-5 3 4 0}]  
puts [median {-5 3 4 0}]
```

Output:

```
0.5  
1.5
```

Variance

The command *variance* calculates variance, and the command *stdev* calculates standard deviation. By default, they compute sample statistics.

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

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

\$vector List (at least length 2) to compute statistic of.

\$pop Compute population variance instead of sample variance. Default false.

Example 12: Variance and standard deviation

Code:

```
puts [variance {-5 3 4 0}]  
puts [stdev {-5 3 4 0}]
```

Output:

```
16.333333333333332  
4.041451884327381
```

Vectors and Matrices

This package provides some basic linear algebra routines for manipulating vectors and matrices, where a matrix is 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 and vectors 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 13: 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}}
```

Dot Product

The dot product of two vectors can be computed with *dot*.

```
dot $a $b
```

`$a`

First vector.

`$b`

Second vector. Must be same length as `$a`.

Example 14: Dot product of two vectors

Code:

```
set x {1 2 3}
set y {-2 -4 6}
puts [dot $x $y]
puts [sum [lop2 $x * $y]]; # alternative way to compute dot product
```

Output:

```
8
8
```

Vector Norm

The norm 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
```

Cross Product

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

```
cross $a $b
```

\$a First vector. Must be length 3.

\$b Second vector. Must be length 3.

Example 16: Compute area of plane

Code:

```
set a {3 -1 5}
set b {2 8 2}
set area [norm [cross $a $b]]
puts $area
```

Output:

```
49.558046773455466
```

Matrix Generation

The commands *zeros*, *ones* and *rand* generate matrices (or other ND lists) of a specified shape, filled with zeros, ones, and random numbers between 0 and 1, respectively.

```
zeros $n ...
```

```
ones $n ...
```

```
rand $n ...
```

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

Identity Matrix

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

```
eye $n
```

`$n` Size of identity matrix

Example 17: Generating matrices

Code:

```
puts [ones 2 3]  
puts [eye 3]
```

Output:

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

Transposing

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:

```
24.0 12.0 72.0 75.0
```

N-Dimensional Lists

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 ndlist. 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.

```
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 *nsiz*e 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>
```

```
nsiz
```

\$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 20: Getting shape and size of an ND list

Code:

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

Output:

```
2 3
6
```

Repeating and Expanding

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

```
nrepeat $value $n ...
```

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

Example 21: Create nested ND list with one value

Code:

```
puts [nrepeat foo 1 2 3]
```

Output:

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

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 22: Expand an ND list

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 23: 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 24: 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 $input $n
```

\$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.).
\$n	Number of elements in list.

Additionally, indices get passed through the `::ndlist::Index2Integer` command, which converts `end±integer`, `integer±integer` and negative wrap-around indexing (where -1 is equivalent to “end”) into normal integer indices. This command will return error if the index is out of range.

```
::ndlist::Index2Integer $index $n
```

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

Example 25: Index Notation

Code:

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

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 arguments, parsed with *::ndlist::ParseIndex*. The number of index arguments determines the interpreted dimensions.

Example 26: 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 the blank string is used as a replacement value, it will remove values from the ND lists, as long as it is only removing along one dimension. Otherwise, the replacement ND list must be expandable to the target index dimensions.

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

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

\$varName	Name of ND list to modify.
\$ndlist	ND list to modify.
\$i ...	Index arguments, parsed with <i>::ndlist::ParseIndex</i> . The number of index arguments determines the interpreted dimensions.
\$sublist	Replacement list, or blank to delete values.

Example 27: Swapping rows in a matrix

Code:

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

Output:

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

Combination

The command *ninsert* allows you to insert a sublist into an ND list at a specified index and axis. The command *nstack* is a special case of *ninsert* for the index “end”. Sublist must agree in dimension at all other axes.

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

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

<code>\$nd</code>	Dimensionality of ND list (e.g. 2D, 2d, or 2 for a matrix).
<code>\$ndlist1</code>	ND list to modify.
<code>\$index</code>	Index to insert at.
<code>\$ndlist2</code>	ND list to insert.
<code>\$axis</code>	Axis to insert at (default 0).

Example 28: Inserting rows and columns in a matrix

Code:

```
puts [ninsert 2D {{1 2 3} {4 5 6} {7 8 9}} 0 {{A B C}}]  
puts [ninsert 2D {1 2 3} end {4 5 6} 1]; # same as [nstack 2D {1 2 3} {4 5 6} 1]
```

Output:

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

Example 29: Stack 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}}
```

Swapping 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.

Example 30: Transposing a matrix

Code:

```
set x {{1 2} {3 4}}
puts [nswapaxes $x 0 1]; # same as [transpose $x]
```

Output:

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

Example 31: Swapping axes of a tensor

Code:

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

Output:

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

Mapping Over an ND List

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...
```

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

Example 32: 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 33: 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...
```

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

Example 34: 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 35: 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 *nswapaxes* 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 36: 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 37: 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 38: 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`, 16
`::ndlist::ParseIndex`, 16

`cross`, 10

`dot`, 9

`eye`, 11

`find`, 3

`lapply`, 5
`lapply2`, 6
`linspace`, 4
`linspace`, 4
`linterp`, 3
`lop`, 5
`lop2`, 6

`matmul`, 12
`max`, 7
`mean`, 8
`median`, 8
`min`, 7

`napply`, 21
`napply2`, 22
`ndlist`, 13
`nexpand`, 14
`nexpr`, 24
`nflatten`, 15
`nget`, 17
`ninsert`, 19
`nmap`, 24
`nop`, 21

`nop2`, 22
`norm`, 10
`nreduce`, 23
`nrepeat`, 14
`nreplace`, 18
`nreshape`, 15
`nset`, 18
`nshape`, 13
`nsize`, 13
`nstack`, 19
`nswapaxes`, 20

`ones`, 11

`product`, 7

`rand`, 11
`range`, 2

`stdev`, 8
`sum`, 7

`transpose`, 12

`variance`, 8

`zeros`, 11