N-Dimensional Lists (ndlist)

Version 0.1

Alex Baker

https://github.com/ambaker1/ndlist

September 9, 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. Additionally, this package provides the "ndlist" object variable type.

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. For example, a matrix is a 2D list, or a list of equal length row vectors (1D), which contain arbitrary scalar values, as shown below:

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

Example 1: Defining matrices in Tcl

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}}
```

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”.

Initialization

The command *nrepeat* can be used to initialize an ND list of any size.

```
nrepeat $shape $value
```

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

Example 2: Create nested ND list with one value

Code:

```
puts [nrepeat {1 2 3} 0]
```

Output:

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

Flattening and Reshaping

The command *nflatten* flattens an ND list, and the command *nreshape* flattens and reshapes an ND list to a compatible shape.

```
nflatten $nd $ndlist
```

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

```
nreshape $nd $ndlist $shape
```

$nd	Dimensionality of ND list (e.g. 2D for a matrix)
$ndlist	ND list to flatten and reshape.
$shape	New shape (list of dimensions).

Example 3: Flatten and reshape ND lists

Code:

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

Output:

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

Transpose

The command *ntranspose* swaps axes of an ndlist. By default, it just transposes the matrix representation of the data, swapping rows and columns.

```
ntranspose $nd $ndlist <$axis1 $axis2>
```

$nd	Dimensionality of ND list (e.g. 2D for a matrix)
$ndlist	ND list to manipulate.
$axis1	Axis to swap with axis 2 (default 0)
$axis2	Axis to swap with axis 1 (default 1)

Example 4: Transposing a matrix

Code:

```
puts [ntranspose 2D {{1 2} {3 4}}]
```

Output:

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

Example 5: Swapping axes of a tensor

Code:

```
puts [ntranspose 3D {{{1 2} {3 4}} {{5 6} {7 8}}} 0 2]
```

Output:

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

Insertion

The command *ninsert* allows you to insert a sublist into an ND list at a specified index and axis. Sublist must agree in dimension at all other axes.

```
ninsert $nd $ndlist $index $sublist <$axis>
```

`$nd`	Dimensionality of ND list (e.g. 2D for a matrix)
`$ndlist`	ND list to manipulate.
`$index`	Index to insert at.
`$axis`	Axis to insert at (default 0).

Example 6: Inserting rows and columns in a matrix

Code:

```
# Insert row
puts [ninsert 2D {{1 2 3} {4 5 6} {7 8 9}} 0 {{A B C}}]
# Insert column
puts [ninsert 2D {1 2 3} end {4 5 6} 1]
```

Output:

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

Example 7: Stack tensors

Code:

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

Output:

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

Access

Portions of an ND list can be accessed with the command *nget*. Aliases for matrices (2D) and vectors (1D) are provided with the commands *mget* and *vget*, and aliases for accessing matrix rows and columns (using $i* indexing), are provided with the commands *rget* and *cget*.

```
nget $ndlist $arg1 $arg2 ...
```

`$ndlist`	ND list to access
`$arg1 $arg2 ...`	Index arguments, using index notation (see pg. 7). The number of index arguments determines the interpreted dimensions.

Example 8: Accessing matrix values, rows, and columns

Code:

```
set A {{1 2 3} {4 5 6} {7 8 9}}
puts [nget $A 0 0]
puts [nget $A 0 :]
puts [nget $A : 0]
```

Output:

```
1
1 2 3
1 4 7
```

Index Notation

Index input for all ND list access and modification functions gets passed through the ND list index parser *::ndlist::ParseIndex*.

```
::ndlist::ParseIndex $input $n
```

`$input`	Index input. Options are shown below:
`:`	All indices
`$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 asterisk, "flattens" the ndlist (e.g. 0* or end-3*).
`$n`	Number of elements in list.

Additionally, index range arguments `$start` and `$stop`, all indices in `$iList`, and single indices `$i` 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.

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

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

Example 9: Index notation

Code:

```
set A {{1 2 3} {4 5 6} {7 8 9}}
puts [nget $A 0 :]
puts [nget $A 0* :]; # can "flatten" row
puts [nget $A 0:1 1]
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
2 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 same index argument syntax as *nget*. 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 agree in dimension to the to the index argument dimensions, or be unity. For example, you can replace a 4x3 portion of a matrix with 4x3, 4x1, 1x3, or 1x1 matrices.

```
nset $varName $arg1 $arg2 ...  $sublist
```

```
nreplace $ndlist $arg1 $arg2 ...  $sublist
```

`$varName`	Name of ND list to modify.
`$ndlist`	ND list to modify.
`$arg1 $arg2 ...`	Index arguments, using index notation (see pg. 7). The number of index arguments determines the interpreted dimensions.
`$sublist`	Replacement list, or blank to delete values.

Example 10: Swapping rows in a matrix

Code:

```
set a {{1 2} {3 4} {5 6}}
nset a {1 0} : [nget $a {0 1} :]
puts $a
```

Output:

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

Example 11: Deleting a column in a matrix

Code:

```
set a {{1 2} {3 4} {5 6}}
set b [nreplace $a : 1 ""]
puts $b
```

Output:

```
1 3 5
```

Note: if attempting to modify outside of the dimensions of the ND list, the ND list will be expanded and filled with the value in the variable `::ndlist::filler`. By default, the filler is 0, but this can easily be changed.

Element-Wise Operations

Basic math operators can be mapped over an ND list with the command *nop*.

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

$nd	Dimensionality of ND list (e.g. 2D for a matrix)
$ndlist	ND list to perform element-wise operation over.
$op	Math operator (using tcl::mathop namespace).
$arg...	Operator arguments (see tcl::mathop documentation).

Example 12: Element-wise operations

Code:

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

Output:

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

ND List Mapping

The command *nmap* is a general purpose mapping function for N-dimensional lists in Tcl. If multiple ND lists are provided for iteration, they must agree in dimension or be unity, like in *nset*. Returns an ND list in similar fashion to the Tcl *lmap* command. Additionally, elements can be skipped with *continue*, and the entire loop can be exited with *break*.

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

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

Example 13: ND list mapping

Code:

```
set testmat {{1 2 3} {4 5 6} {7 8 9}}
# Checkerboard sign pattern
puts [nmap 2D x $testmat {expr {
    $x*([i]%2 + [j]%2 == 1?-1:1)
}}]
# Simple formatting
puts [nmap 2D x $testmat {format %.2f $x}]
```

Output:

```
{1 -2 3} {-4 5 -6} {7 -8 9}
{1.00 2.00 3.00} {4.00 5.00 6.00} {7.00 8.00 9.00}
```

Index Access

The iteration indices of *nmap* are accessed with the commands *i*, *j*, & *k*.

```
i <$axis>
```

`$axis`	Dimension to access mapping index at. Default 0.

The commands *j* and *k* are simply shorthand for *i* with dimensions 1 and 2.

```
j
```

```
k
```

ND List Reference Mapping

Similar to the commands *leval* and *lexpr* in the "vutil" package, the commands *neval* and *nexpr* perform element-wise operations over ND list objects. Additionally, these are built into the "`:=`" and "`::=`" ND list operators.

```
neval $body <$nd $ndlist> <"-->" $refName>
```

```
nexpr $expr <$nd $ndlist> <"-->" $refName>
```

`$body`	Tcl script with list object references.
`$expr`	Tcl expression with list object references.
`$nd`	Dimensionality of ND list (e.g. 2D for a matrix)
`$ndlist`	ND list to iterate over, with `$@.` reference.
`$refName`	Optional reference variable to tie resulting ND list to. Blank to return value.

Example 14: Element-wise expressions

Code:

```
matrix x {{1 2} {3 4} {5 6}}
matrix y 5.0
nexpr {$@x + $@y}
```

Output:

```
{6.0 7.0} {8.0 9.0} {10.0 11.0}
```