# Array Manipulation Utilities

Jeff Anding
jganding@outlook.com

## Introduction

I use the data structure of Maple arrays extensively in performing various data analysis activities using Maple worksheets. I use Maple 2018. Yes, an upgrade is due but I am a hobbyist and I use the personal version. For cost reasons, I need to make this version last as long as practical. Unfortunately, Maple 2018 doesn't have extensive, easy-to-use utilities for manipulating arrays in all the ways I need. Therefore I have developed a number of procedures that greatly improve the efficiency of my work. I am sharing these in hopes they will assist others in their efforts.

Included below is the code as well as demos for the utilities which provide easy-to-use means for:

- inserting a new range of rows into an existing array
- inserting a new range of columns into an existing array
- deleting a range of rows from an array
- deleting a range of columns from an array
- copying a range of rows from an array into a new array
- copying a range of columns from an array into a new array
- deleting the selection of rows from an array that match a user specified value in a user specified column in the array
- retaining the selection of rows in an array that match a user specified value in a user specified column in the array and deleting the rest of the rows
- copying the selection of rows from an array that match a user specified value in a user specified column in the array into a new array
- copying rows from one array and appending them to the bottom of another array
- shorthand querying of the dimensions of an array
- creating "data window" ranges for calculating things like moving averages and moving standard deviations
- cleansing of one dimensional arrays of non-numeric data to enable error free data analysis and operation

It should be noted that these utilities are designed to work with 2 dimensional arrays that are indexed starting from 1 in each dimension, i.e., 1 to number of rows by 1 to number of columns.

# Initialization and Procedure Code

*restart*;
*interface*(*rtablesize* = 25) :

**Insert a new range of rows into an existing array**

**insrowArray**(array name, insertions are made after this row number, number of rows to insert, value to populate new rows with)
**insrowArray**(array name, -1, number of rows to insert before row 1, value to populate new rows with)

```
insrowArray := proc (a::evaln(Array),row1::integer, num2insert
```

**Insert a new range of columns into an existing array**

**inscolArray**(array name, insertions are made after this column number, number of columns to insert, value to populate new columns with)
**inscolArray**(array name, -1, number of columns to insert before column 1, value to populate new columns with)

```
inscolArray := proc (a::evaln(Array), column1::integer, num2ir
```

**Delete a range of rows from an array**

**delrowArray**(array name, first row to delete, number of rows to delete)

```
delrowArray := proc (a::evaln(Array), row1::integer, num2del::
```

**Delete a range of columns from an array**

**delcolArray**(array name, first column to delete, number of columns to delete)

```
delcolArray := proc (a::evaln(Array), column1::integer, num2de
```

**Copy a range of rows from an array into an new array**

**copyrowArray**(array of source data, row to start at, number of rows to copy, destination array which is created);

Note: this procedure will create the destination array (caution, it will replace an existing array of the same name)

```
copyrowArray := proc (a::evaln(Array), row1::integer, num2keep
```

**Copy a range of columns from an array into an new array**

**copycolArray**(array of source data, column to start at, number of columns to copy, destination array which is created);

Note: this procedure will create the destination array (caution, it will replace an existing array of the same name)

```
copycolArray := proc (a::evaln(Array), column1::integer, num2k
```

**Delete the selection of rows from an array that match a user specified value in a user specified column in the array**

**delselrowArray**(array name, column to test, value in the column that triggers a "delete" for that row)

Note: this procedure has a dependency and requires procedure **delrowArray** be included in your worksheet/workbook

```
delselrowArray:=proc(a::evaln(Array),col::integer,val)
```

**Retain the selection of rows in an array that match a user specified value in a user specified column in the array and delete the rest of the rows**

**keepselrowArray**(array name, column to test, value in the column that triggers a "keep" for that row);

Note: this procedure has a dependency and requires procedure **delrowArray** be included in your worksheet/workbook

```
keepselrowArray:=proc(a::evaln(Array),col::integer,val)
```

**Copy the selection of rows from an array that match a user specified value in a user specified column in the array into a new array**

**copyselrowArray**(source array name, column to test, value in the column that triggers a "copy" for that row, destination array which is created);

Note: this procedure will create the destination array (caution, it will replace an existing array of the same name)

```
copyselrowArray:=proc (a::evaln(Array), col::integer, val, b::
```

**Copy a row from one array, based on row number, and append it to the bottom of another array**

**appendrowArray**(source array name, row from source array to append to destination array, destination array name)

Note: this procedure will create the destination array based on the name in the 3rd argument if it doesn't already exist. If if does exist, it will append data to that array. The destination array must be of the same datatype as the source array (typically, "datatype=anything") and have the same number of columns as source array for this routine to work.

Note: this procedure has dependencies and requires procedures **insrowArray** and **delrowArray** be included in your worksheet/workbook

```
appendrowArray := proc (a::(evaln(Array)), idx::integer, b::ur
```

**Shorthand querying of the dimensions of an array**

I was tired of typing "*upperbound(array name)[1]* or *[2]*" and made something shorter

**ubr**(array name) this returns the number of rows in the array "array name"

**ubc**(array name) this returns the number of columns in the array "array name"

**dims**(array name) this returns the rows and columns in the array "array name".
Shorthand for "*ArrayTools:-Dimensions(array name)*"

## Array Dimension Query Utilites (ubr, ubc, and dims)

**Create "data window" ranges for calculating things like moving averages and moving standard deviations**

These procedures are data window selectors for calculating the likes of moving averages and moving standard deviations. They select and return k elements to the left and k elements to the right of the "current" data point for use in the calculations.

**k0elems**(array name,column to extract data from, number of values to left and right of "center" value, center value index) this does not return the current "center" data element in the results

**k1elems**(array name,column to extract data from, number of values to left and right of "center" value, center value index) this does return the current "center" data elements in the results

Known Limitation: the span of 2k (as well as 2k+1) elements cannot overlap the two ends of the source data "list" at the same time. This should not be an issue since a large k value (i.e., k approaching n/2 where n is the number of elements in the array) is not likely to be a very common use case.

## Data Window Selectors (k0elems and k1elems)

**Cleanse one dimensional arrays of non-numeric data to enable error free data analysis and operation**

Two procedures are included here. They use different methods for cleaning (removing non-numeric data) one dimensional arrays and they have shown to perform differently for different scenarios. Each is faster under certain conditions. You may need to experiment with your data to see which one gives you the best performance. Regardless of the one you use, the result will be the same.

Here is a table with my findings when used on arrays of different sizes and different "relative cleanliness":

| | **100,001 rows in array 1 row removed** | **13,427 rows in array 8,196 rows removed** |
|---|---|---|
| **clean1d_A** | 0.438 sec | 18 sec |
| **clean1d_B** | 178 sec | 0.897 sec |

It seems that the larger the number of clean rows an array has, the more likely **clean1d_A** is the suitable choice. Try both and see which works best for your use case. Keep in mind that both are performant on "small" arrays (i.e., several hundred to a couple of thousand records) and in such cases, it probably doesn't matter much which procedure you use.

**clean1d_A**(array name);

Note: this procedure has a dependencies and requires procedures **delseldrowArray**, **inscolArray**, **delcolArray,** and **Dimension Query Utilities** be included in your worksheet/workbook

```
clean1d_A:=proc(a::evaln(Array))
```

**clean1d_B**(array name);

Note: this procedure has a dependencies and requires procedures **appendrowArray**, **inscolArray**, **delcolArray,** and **Dimension Query Utilities** be included in your worksheet/workbook. Also, **clean1d_B** includes a progress counter since it may run for a couple of minutes on extremely large arrays.

```
clean1d_B:=proc(a::evaln(Array))
```

>

# Demonstration of Application

Here each procedure above is demonstrated

First, to give us something to work with for the demos, we load data into a couple of arrays.

```
> A := Array(1..12, 1..7, [seq([seq(i + 10·j, i = 1..8)], j = 1..12)]) :

  for k from 1 to 12 do
    A[k, 7] := modp(A[k, 5], 7)
  end do:
  AA := copy(A) :
  data := Array(1..12, 1..7, [seq([seq(i + 10·j, i = 1..8)], j = 1..12)]) :
> A;
  data;
```

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 1 \\ 21 & 22 & 23 & 24 & 25 & 26 & 4 \\ 31 & 32 & 33 & 34 & 35 & 36 & 0 \\ 41 & 42 & 43 & 44 & 45 & 46 & 3 \\ 51 & 52 & 53 & 54 & 55 & 56 & 6 \\ 61 & 62 & 63 & 64 & 65 & 66 & 2 \\ 71 & 72 & 73 & 74 & 75 & 76 & 5 \\ 81 & 82 & 83 & 84 & 85 & 86 & 1 \\ 91 & 92 & 93 & 94 & 95 & 96 & 4 \\ 101 & 102 & 103 & 104 & 105 & 106 & 0 \\ 111 & 112 & 113 & 114 & 115 & 116 & 3 \\ 121 & 122 & 123 & 124 & 125 & 126 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 51 & 52 & 53 & 54 & 55 & 56 & 57 \\ 61 & 62 & 63 & 64 & 65 & 66 & 67 \\ 71 & 72 & 73 & 74 & 75 & 76 & 77 \\ 81 & 82 & 83 & 84 & 85 & 86 & 87 \\ 91 & 92 & 93 & 94 & 95 & 96 & 97 \\ 101 & 102 & 103 & 104 & 105 & 106 & 107 \\ 111 & 112 & 113 & 114 & 115 & 116 & 117 \\ 121 & 122 & 123 & 124 & 125 & 126 & 127 \end{bmatrix} \quad \textbf{(3.1)}$$

Now to demo the array manipulation functions...

## Insert new range of rows into an existing array

```
> # insert after row 4, 2 rows of value 99
  insrowArray(A, 4, 2, 99);
> A;
```

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 1 \\ 21 & 22 & 23 & 24 & 25 & 26 & 4 \\ 31 & 32 & 33 & 34 & 35 & 36 & 0 \\ 41 & 42 & 43 & 44 & 45 & 46 & 3 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 51 & 52 & 53 & 54 & 55 & 56 & 6 \\ 61 & 62 & 63 & 64 & 65 & 66 & 2 \\ 71 & 72 & 73 & 74 & 75 & 76 & 5 \\ 81 & 82 & 83 & 84 & 85 & 86 & 1 \\ 91 & 92 & 93 & 94 & 95 & 96 & 4 \\ 101 & 102 & 103 & 104 & 105 & 106 & 0 \\ 111 & 112 & 113 & 114 & 115 & 116 & 3 \\ 121 & 122 & 123 & 124 & 125 & 126 & 6 \end{bmatrix} \quad \textbf{(3.2)}$$

```
> # insert before row 1, 3 rows of value 0
  insrowArray(A,-1, 3, 0);
> A;
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 11 & 12 & 13 & 14 & 15 & 16 & 1 \\ 21 & 22 & 23 & 24 & 25 & 26 & 4 \\ 31 & 32 & 33 & 34 & 35 & 36 & 0 \\ 41 & 42 & 43 & 44 & 45 & 46 & 3 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 51 & 52 & 53 & 54 & 55 & 56 & 6 \\ 61 & 62 & 63 & 64 & 65 & 66 & 2 \\ 71 & 72 & 73 & 74 & 75 & 76 & 5 \\ 81 & 82 & 83 & 84 & 85 & 86 & 1 \\ 91 & 92 & 93 & 94 & 95 & 96 & 4 \\ 101 & 102 & 103 & 104 & 105 & 106 & 0 \\ 111 & 112 & 113 & 114 & 115 & 116 & 3 \\ 121 & 122 & 123 & 124 & 125 & 126 & 6 \end{bmatrix} \quad \textbf{(3.3)}$$

```
>
```

## Insert new range of columns into an existing array

```
> # insert after column 5, 3 columns of value 16.8
  inscolArray(A, 5, 3, 16.8);
> A;
```

**(3.4)**

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\ 11 & 12 & 13 & 14 & 15 & 16.8 & 16.8 & 16.8 & 16 & 1 \\ 21 & 22 & 23 & 24 & 25 & 16.8 & 16.8 & 16.8 & 26 & 4 \\ 31 & 32 & 33 & 34 & 35 & 16.8 & 16.8 & 16.8 & 36 & 0 \\ 41 & 42 & 43 & 44 & 45 & 16.8 & 16.8 & 16.8 & 46 & 3 \\ 99 & 99 & 99 & 99 & 99 & 16.8 & 16.8 & 16.8 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 16.8 & 16.8 & 16.8 & 99 & 99 \\ 51 & 52 & 53 & 54 & 55 & 16.8 & 16.8 & 16.8 & 56 & 6 \\ 61 & 62 & 63 & 64 & 65 & 16.8 & 16.8 & 16.8 & 66 & 2 \\ 71 & 72 & 73 & 74 & 75 & 16.8 & 16.8 & 16.8 & 76 & 5 \\ 81 & 82 & 83 & 84 & 85 & 16.8 & 16.8 & 16.8 & 86 & 1 \\ 91 & 92 & 93 & 94 & 95 & 16.8 & 16.8 & 16.8 & 96 & 4 \\ 101 & 102 & 103 & 104 & 105 & 16.8 & 16.8 & 16.8 & 106 & 0 \\ 111 & 112 & 113 & 114 & 115 & 16.8 & 16.8 & 16.8 & 116 & 3 \\ 121 & 122 & 123 & 124 & 125 & 16.8 & 16.8 & 16.8 & 126 & 6 \end{bmatrix} \quad \textbf{(3.4)}$$

> *# insert before column 1, 2 columns of value 3.14*
> *inscolArray*$(A, -1, 2, 3.14)$;

> $A$;

**(3.5)**

$$\begin{bmatrix}
3.14 & 3.14 & 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\
3.14 & 3.14 & 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\
3.14 & 3.14 & 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\
3.14 & 3.14 & 11 & 12 & 13 & 14 & 15 & 16.8 & 16.8 & 16.8 & 16 & 1 \\
3.14 & 3.14 & 21 & 22 & 23 & 24 & 25 & 16.8 & 16.8 & 16.8 & 26 & 4 \\
3.14 & 3.14 & 31 & 32 & 33 & 34 & 35 & 16.8 & 16.8 & 16.8 & 36 & 0 \\
3.14 & 3.14 & 41 & 42 & 43 & 44 & 45 & 16.8 & 16.8 & 16.8 & 46 & 3 \\
3.14 & 3.14 & 99 & 99 & 99 & 99 & 99 & 16.8 & 16.8 & 16.8 & 99 & 99 \\
3.14 & 3.14 & 99 & 99 & 99 & 99 & 99 & 16.8 & 16.8 & 16.8 & 99 & 99 \\
3.14 & 3.14 & 51 & 52 & 53 & 54 & 55 & 16.8 & 16.8 & 16.8 & 56 & 6 \\
3.14 & 3.14 & 61 & 62 & 63 & 64 & 65 & 16.8 & 16.8 & 16.8 & 66 & 2 \\
3.14 & 3.14 & 71 & 72 & 73 & 74 & 75 & 16.8 & 16.8 & 16.8 & 76 & 5 \\
3.14 & 3.14 & 81 & 82 & 83 & 84 & 85 & 16.8 & 16.8 & 16.8 & 86 & 1 \\
3.14 & 3.14 & 91 & 92 & 93 & 94 & 95 & 16.8 & 16.8 & 16.8 & 96 & 4 \\
3.14 & 3.14 & 101 & 102 & 103 & 104 & 105 & 16.8 & 16.8 & 16.8 & 106 & 0 \\
3.14 & 3.14 & 111 & 112 & 113 & 114 & 115 & 16.8 & 16.8 & 16.8 & 116 & 3 \\
3.14 & 3.14 & 121 & 122 & 123 & 124 & 125 & 16.8 & 16.8 & 16.8 & 126 & 6
\end{bmatrix} \quad \textbf{(3.5)}$$

>

## Delete a range of rows from an array

```
> # starting with row 6, delete 4 rows
  delrowArray(A, 6, 4);
> A;
```

**(3.6)**

$$\begin{bmatrix} 3.14 & 3.14 & 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 0 & 0 & 16.8 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 11 & 12 & 13 & 14 & 15 & 16.8 & 16.8 & 16.8 & 16 & 1 \\ 3.14 & 3.14 & 21 & 22 & 23 & 24 & 25 & 16.8 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 54 & 55 & 16.8 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 64 & 65 & 16.8 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 74 & 75 & 16.8 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 84 & 85 & 16.8 & 16.8 & 16.8 & 86 & 1 \\ 3.14 & 3.14 & 91 & 92 & 93 & 94 & 95 & 16.8 & 16.8 & 16.8 & 96 & 4 \\ 3.14 & 3.14 & 101 & 102 & 103 & 104 & 105 & 16.8 & 16.8 & 16.8 & 106 & 0 \\ 3.14 & 3.14 & 111 & 112 & 113 & 114 & 115 & 16.8 & 16.8 & 16.8 & 116 & 3 \\ 3.14 & 3.14 & 121 & 122 & 123 & 124 & 125 & 16.8 & 16.8 & 16.8 & 126 & 6 \end{bmatrix} \quad \textbf{(3.6)}$$

>

## Delete a range of columns from an array

> *# starting with column 6, delete 3 columns*
*delcolArray*(*A*, 6, 3);

> *A*;

$$\begin{bmatrix} 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 11 & 12 & 13 & 16.8 & 16.8 & 16 & 1 \\ 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \\ 3.14 & 3.14 & 91 & 92 & 93 & 16.8 & 16.8 & 96 & 4 \\ 3.14 & 3.14 & 101 & 102 & 103 & 16.8 & 16.8 & 106 & 0 \\ 3.14 & 3.14 & 111 & 112 & 113 & 16.8 & 16.8 & 116 & 3 \\ 3.14 & 3.14 & 121 & 122 & 123 & 16.8 & 16.8 & 126 & 6 \end{bmatrix} \quad \textbf{(3.7)}$$

>

## Copy a range of rows from an array into an new array

```
> # in array A, starting with row 5, copy 5 rows and put them into array B.  Array B is created.
  copyrowArray(A, 5, 5, B);
> A; B;
```

$$\begin{bmatrix} 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 11 & 12 & 13 & 16.8 & 16.8 & 16 & 1 \\ 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \\ 3.14 & 3.14 & 91 & 92 & 93 & 16.8 & 16.8 & 96 & 4 \\ 3.14 & 3.14 & 101 & 102 & 103 & 16.8 & 16.8 & 106 & 0 \\ 3.14 & 3.14 & 111 & 112 & 113 & 16.8 & 16.8 & 116 & 3 \\ 3.14 & 3.14 & 121 & 122 & 123 & 16.8 & 16.8 & 126 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \end{bmatrix} \quad \textbf{(3.8)}$$

```
>
```

## Copy a range of columns from an array into a new array

```
> # in array B, starting with column 3, copy 4 columns and put them into array C.  Array C is
      created.
  copycolArray(B, 3, 4, C);
> B; C;
```

$$\begin{bmatrix} 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 21 & 22 & 23 & 16.8 \\ 51 & 52 & 53 & 16.8 \\ 61 & 62 & 63 & 16.8 \\ 71 & 72 & 73 & 16.8 \\ 81 & 82 & 83 & 16.8 \end{bmatrix} \quad \textbf{(3.9)}$$

>

## Copy the selection of rows from an array that match a user specified value in a user specified column in the array into a new array

```
> # copy all the rows from array A that in column 9, have a value of 4 and put them into array
      "sample".  Array "sample" is created.
  copyselrowArray(A, 9, 4, sample);
      rows copied to new array:     2

> A; sample;
```

$$\begin{bmatrix} 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 11 & 12 & 13 & 16.8 & 16.8 & 16 & 1 \\ 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \\ 3.14 & 3.14 & 91 & 92 & 93 & 16.8 & 16.8 & 96 & 4 \\ 3.14 & 3.14 & 101 & 102 & 103 & 16.8 & 16.8 & 106 & 0 \\ 3.14 & 3.14 & 111 & 112 & 113 & 16.8 & 16.8 & 116 & 3 \\ 3.14 & 3.14 & 121 & 122 & 123 & 16.8 & 16.8 & 126 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 91 & 92 & 93 & 16.8 & 16.8 & 96 & 4 \end{bmatrix} \quad \textbf{(3.10)}$$

>

## Delete the selection of rows from an array that match a user specified value in a user specified column in the array

```
> # delete all the rows from array A that in column 9, have a value of 6
  delselrowArray(A, 9, 6);
```

```
rows deleted:     2
   rows kept:    11
```

> $A$;

$$\begin{bmatrix} 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 0 & 0 & 0 & 16.8 & 16.8 & 0 & 0 \\ 3.14 & 3.14 & 11 & 12 & 13 & 16.8 & 16.8 & 16 & 1 \\ 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \\ 3.14 & 3.14 & 91 & 92 & 93 & 16.8 & 16.8 & 96 & 4 \\ 3.14 & 3.14 & 101 & 102 & 103 & 16.8 & 16.8 & 106 & 0 \\ 3.14 & 3.14 & 111 & 112 & 113 & 16.8 & 16.8 & 116 & 3 \end{bmatrix} \quad \textbf{(3.11)}$$

>

## Retain the selection of rows in an array that match a user specified value in a user specified column in the array and delete the rest of the rows

> *# keep all the rows in array A that in column 9, have a value of 1 and delete the other rows*
*keepselrowArray*($A$, 9, 1);

```
     rows kept:     2
 rows deleted:     9
```

> $A$;

$$\begin{bmatrix} 3.14 & 3.14 & 11 & 12 & 13 & 16.8 & 16.8 & 16 & 1 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \end{bmatrix} \quad \textbf{(3.12)}$$

>

***Here's a more involved example using row selection***

Note: this set of steps will be easier to perform using the next utility demoed. But until we get to that later, here is a set of techniques that can work well.

>

> $r := rand(0.0..1.0)$;

$$r := () \mapsto RandomTools:-Generate(float('range' = 0...1.0, 'method' = 'uniform')) \quad \textbf{(3.13)}$$

> *# for this exercise, let's create an array and populate it with random numbers between 0 and 1*

$R := Array(1..10, 1..1)$ :

```
r := rand(0.0..1.0):

for i from 1 to ubr(R) do
    R[i, 1] := r( ):
 end do:
 R;
```

$$\begin{bmatrix} 0.2342493224 \\ 0.1799302829 \\ 0.5137385362 \\ 0.2907448089 \\ 0.8953600369 \\ 0.2617341097 \\ 0.7780122500 \\ 0.06587642124 \\ 0.7235311453 \\ 0.3157837057 \end{bmatrix} \tag{3.14}$$

```
> # insert a column to store our "flags" for keeping or deleting rows
  inscolArray(R, -1, 1, 0);
> R;
```

$$\begin{bmatrix} 0 & 0.2342493224 \\ 0 & 0.1799302829 \\ 0 & 0.5137385362 \\ 0 & 0.2907448089 \\ 0 & 0.8953600369 \\ 0 & 0.2617341097 \\ 0 & 0.7780122500 \\ 0 & 0.06587642124 \\ 0 & 0.7235311453 \\ 0 & 0.3157837057 \end{bmatrix} \tag{3.15}$$

```
>
> # I only want to flag rows that fall within a certain range (0.1 <x <0.4)

  for i from 1 to ubr(R) do
     if R[i, 2] < 0.4 and R[i, 2] > 0.1 then
        R[i, 1] := 1;
     end if;
  end do;
  R;
```

$$\begin{bmatrix} 1 & 0.2342493224 \\ 1 & 0.1799302829 \\ 0 & 0.5137385362 \\ 1 & 0.2907448089 \\ 0 & 0.8953600369 \\ 1 & 0.2617341097 \\ 0 & 0.7780122500 \\ 0 & 0.06587642124 \\ 0 & 0.7235311453 \\ 1 & 0.3157837057 \end{bmatrix} \quad \textbf{(3.16)}$$

> *# copy the rows that meet our criteria to array T*
*copyselrowArray*(*R*, 1, 1, *T*);

```
    rows copied to new array:     5
```

> *T*;

$$\begin{bmatrix} 1 & 0.2342493224 \\ 1 & 0.1799302829 \\ 1 & 0.2907448089 \\ 1 & 0.2617341097 \\ 1 & 0.3157837057 \end{bmatrix} \quad \textbf{(3.17)}$$

> *# we can get rid of the flag column now*
*delcolArray*(*T*, 1, 1);

> *T*;

$$\begin{bmatrix} 0.2342493224 \\ 0.1799302829 \\ 0.2907448089 \\ 0.2617341097 \\ 0.3157837057 \end{bmatrix} \quad \textbf{(3.18)}$$

> *# let's delete the rows that met the criteria since we stored them in array T*
*# we could also have used the* ***keepselrow*** (***R*, 1, 0**) *function.*

*delselrowArray*(*R*, 1, 1);

```
    rows deleted:     5
       rows kept:     5
```

> *R*;

$$\begin{bmatrix} 0 & 0.5137385362 \\ 0 & 0.8953600369 \\ 0 & 0.7780122500 \\ 0 & 0.06587642124 \\ 0 & 0.7235311453 \end{bmatrix} \quad (3.19)$$

```
> # we can get rid of the flag column in array R now too
  delcolArray(R, 1, 1);
> R;
```

$$\begin{bmatrix} 0.5137385362 \\ 0.8953600369 \\ 0.7780122500 \\ 0.06587642124 \\ 0.7235311453 \end{bmatrix} \quad (3.20)$$

```
> # in this contrived example, we now have one array with the values that meet our criteria and
      one array with those that don't.
>
```

## Copy a row from one array, based on row number, and append it to the bottom of another array

```
> AA;
```

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 1 \\ 21 & 22 & 23 & 24 & 25 & 26 & 4 \\ 31 & 32 & 33 & 34 & 35 & 36 & 0 \\ 41 & 42 & 43 & 44 & 45 & 46 & 3 \\ 51 & 52 & 53 & 54 & 55 & 56 & 6 \\ 61 & 62 & 63 & 64 & 65 & 66 & 2 \\ 71 & 72 & 73 & 74 & 75 & 76 & 5 \\ 81 & 82 & 83 & 84 & 85 & 86 & 1 \\ 91 & 92 & 93 & 94 & 95 & 96 & 4 \\ 101 & 102 & 103 & 104 & 105 & 106 & 0 \\ 111 & 112 & 113 & 114 & 115 & 116 & 3 \\ 121 & 122 & 123 & 124 & 125 & 126 & 6 \end{bmatrix} \quad (3.21)$$

```
> # copy row 4 from AA and append it to "newArray".  "newArray" doesn't exist so it will be
      created.
  appendrowArray(AA, 4, newArray);
> newArray;
```

$$\begin{bmatrix} 41 & 42 & 43 & 44 & 45 & 46 & 3 \end{bmatrix} \tag{3.22}$$

```
> # more involved example of use

  for i from 1 to upperbound(AA)[1] do;      #  we could have used ubr(AA) here :-)
     if AA[i, 6] > 43.7 and AA[i, 6] < 100 then
       appendrowArray(AA, i, newArray2);
     end if;
  end do:

> newArray2;
```

$$\begin{bmatrix} 41 & 42 & 43 & 44 & 45 & 46 & 3 \\ 51 & 52 & 53 & 54 & 55 & 56 & 6 \\ 61 & 62 & 63 & 64 & 65 & 66 & 2 \\ 71 & 72 & 73 & 74 & 75 & 76 & 5 \\ 81 & 82 & 83 & 84 & 85 & 86 & 1 \\ 91 & 92 & 93 & 94 & 95 & 96 & 4 \end{bmatrix} \tag{3.23}$$

```
>
```

## Shorthand querying of the dimensions of an array

```
> B;
```

$$\begin{bmatrix} 3.14 & 3.14 & 21 & 22 & 23 & 16.8 & 16.8 & 26 & 4 \\ 3.14 & 3.14 & 51 & 52 & 53 & 16.8 & 16.8 & 56 & 6 \\ 3.14 & 3.14 & 61 & 62 & 63 & 16.8 & 16.8 & 66 & 2 \\ 3.14 & 3.14 & 71 & 72 & 73 & 16.8 & 16.8 & 76 & 5 \\ 3.14 & 3.14 & 81 & 82 & 83 & 16.8 & 16.8 & 86 & 1 \end{bmatrix} \tag{3.24}$$

```
> # query the dimensions of array B

> ubr(B);
```

$$5 \tag{3.25}$$

```
> ubc(B);
```

$$9 \tag{3.26}$$

```
> dims(B);
```

$$[1..5, 1..9] \tag{3.27}$$

```
>
```

## Create "data window" ranges for calculating things like moving averages and moving standard deviations

```
> data;
```

$$\begin{bmatrix} 11 & 12 & 13 & 14 & 15 & 16 & 17 \\ 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 31 & 32 & 33 & 34 & 35 & 36 & 37 \\ 41 & 42 & 43 & 44 & 45 & 46 & 47 \\ 51 & 52 & 53 & 54 & 55 & 56 & 57 \\ 61 & 62 & 63 & 64 & 65 & 66 & 67 \\ 71 & 72 & 73 & 74 & 75 & 76 & 77 \\ 81 & 82 & 83 & 84 & 85 & 86 & 87 \\ 91 & 92 & 93 & 94 & 95 & 96 & 97 \\ 101 & 102 & 103 & 104 & 105 & 106 & 107 \\ 111 & 112 & 113 & 114 & 115 & 116 & 117 \\ 121 & 122 & 123 & 124 & 125 & 126 & 127 \end{bmatrix} \quad \textbf{(3.28)}$$

> $k := 3$ ;
*# experiment with different values of k to see how the procedure works. k will vary based on your statistical analysis needs.*

$$k := 3 \quad \textbf{(3.29)}$$

> *# Examples using the first column, 6th element as the "center" element. k is 3; this was defined above.*

```
k0elems(data, 1, k, 6); # excludes "center" element
k1elems(data, 1, k, 6); # includes "center" element
```

31, 41, 51, 71, 81, 91

31, 41, 51, 61, 71, 81, 91 **(3.30)**

> *# Lets run this excluding the "center" element for the entirety of column 1*

```
for i from 1 to ubr(data) do
    k0elems(data, 1, k, i);
end do;
```

21, 31, 41

11, 31, 41, 51

11, 21, 41, 51, 61

11, 21, 31, 51, 61, 71

21, 31, 41, 61, 71, 81

31, 41, 51, 71, 81, 91

41, 51, 61, 81, 91, 101

51, 61, 71, 91, 101, 111

61, 71, 81, 101, 111, 121

71, 81, 91, 111, 121

81, 91, 101, 121

91, 101, 111 **(3.31)**

```
> # Now lets do a run including the "center" element for the entirety of column 1

  for i from 1 to ubr(data) do
      k1elems(data, 1, k, i);
   end do;
```

11, 21, 31, 41

11, 21, 31, 41, 51

11, 21, 31, 41, 51, 61

11, 21, 31, 41, 51, 61, 71

21, 31, 41, 51, 61, 71, 81

31, 41, 51, 61, 71, 81, 91

41, 51, 61, 71, 81, 91, 101

51, 61, 71, 81, 91, 101, 111

61, 71, 81, 91, 101, 111, 121

71, 81, 91, 101, 111, 121

81, 91, 101, 111, 121

91, 101, 111, 121 **(3.32)**

```
>
> # Now let's put this to use and calculate a moving average and moving standard deviation for
      column 1 of the array "data"

  moving_average := Array(1 ..ubr(data), 1 ..1) :
  moving_std_dev := Array(1 ..ubr(data), 1 ..1) :

   for i from 1 to ubr(data) do
     moving_average[i, 1] := Statistics:-Mean([k1elems(data, 1, k, i)]) :
     moving_std_dev[i, 1] := Statistics:-StandardDeviation([k1elems(data, 1, k, i)]) :
   end do:

> moving_average;
```

**(3.33)**

$$\begin{bmatrix} 26. \\ 31. \\ 36. \\ 41. \\ 51. \\ 61. \\ 71. \\ 81. \\ 91. \\ 96. \\ 101. \\ 106. \end{bmatrix} \quad (3.33)$$

> *moving_std_dev*;

$$\begin{bmatrix} 12.9099444873581 \\ 15.8113883008419 \\ 18.7082869338697 \\ 21.6024689946929 \\ 21.6024689946929 \\ 21.6024689946929 \\ 21.6024689946929 \\ 21.6024689946929 \\ 21.6024689946929 \\ 18.7082869338697 \\ 15.8113883008419 \\ 12.9099444873581 \end{bmatrix} \quad (3.34)$$

>

## Cleanse one dimensional arrays of non-numeric data to enable error free data analysis and operation

> *# Let's create some "dirty data" to cleanse*

```
dirty_data := Array(1..10, 1..1, [seq([i], i = 1..10)]) :
dirty_data[3, 1] := "hello" :
dirty_data[5, 1] := "world" :
dirty_data[6, 1] := "!!!" :
dirty_data;
```

$$\begin{bmatrix} 1 \\ 2 \\ \text{"hello"} \\ 4 \\ \text{"world"} \\ \text{"!!!"} \\ 7 \\ 8 \\ 9 \\ 10 \end{bmatrix} \quad (3.35)$$

```
> # attempt to use the data as is
  Statistics:-Mean(dirty_data);
Error, (in Statistics:-Mean) unable to evaluate hello to
floating-point
> #   Oops! That won't do!

> # now clean the data before trying again

  clean1d_A(dirty_data);    #   clean1d_B(dirty_data) yeilds the same result
     rows deleted:      3
        rows kept:      7

> dirty_data;
```

$$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 7 \\ 8 \\ 9 \\ 10 \end{bmatrix} \quad (3.36)$$

```
> # now use the cleansed data
  Statistics:-Mean(dirty_data);  #   Much better!!
```

$$\begin{bmatrix} 5.85714285714286 \end{bmatrix} \quad (3.37)$$

# Conclusions

While Maplesoft provides many utilities for working with arrays, some essential array manipulation procedures are still useful beyond those in Maple 2018.  Therefore I have developed some utilities that

make working with arrays easier. The procedures included here are capable and performant and have saved me many hours of work. I have successfully used these utilities to manipulate arrays of 10's of thousands of rows though I typically work only with arrays of a dozen or two columns.

If you find these tools useful, let me know. Also, feel free to improve upon them. jganding@outlook.com

# References

*Maple 2018 help files*