Statistical Functions for Arrays

By David Adams

Technical Note 06-45

Overview

4th Dimension includes seven statistical functions for use on fields, including **Sum**, **Average**, **Min**, **Max**, **Std deviation**, **Variance** and **Sum squares**. However, these functions only work on fields and may be restricted to use within specific phases of a **PRINT SELECTION** report. In other situations, the built-in functions can't be used. Fortunately, it's straightforward to rewrite these basic statistical functions using native 4th Dimension code. The sample database accompanying this database implements standard statistical for use with numeric arrays. The routines go beyond replacements for the standard statistical functions with several enhancements:

- Weighted averages, range, mode, and frequency count are implemented.
- Variance and standard deviation can be calculated from samples or full populations.
- Sum squares can be calculated using either of two popular formulas.

This technical note documents the statistical routines and offers some background information on each function.

Statistical Functions

Documentation and Compilation

Within the sample database, each method is commented individually. Additionally, the ArrayStats Read Me includes summarized documentation for each method. All variables and parameters are declared where used and in the *Compiler_ArrayStats* method. The code may be compiled using any of the compilation typing preferences, including "All variables are typed".

Overview of Calling Requirements

Each of the routines in the sample database are designed to be used on numeric arrays and to return a real result. If there are any errors encountered, an error code is recorded in a variable named ArrayStats_ErrorCode_l. The error code value can always be read by calling *ArrayStatsError_Get*. A description of an error code can be read through *ArrayStatsError_GetDefinition*. In general, following the rules below avoids all errors:

- Pass a pointer to a numeric array.
- Make sure the array has at least one element.

Method Structure

Within the sample database, each method carefully tests incoming parameters, as illustrated by the code of the *ArrayStats_GetMax* method, listed below (whitespace adjusted for legibility):

```
C_REAL($0;$max_real)
C_POINTER($1;$array_pointer)

$max_real:=0` Initialize result.

` ------------------------
` Test Parameters
` ------------------------
ArrayStatsError_Set (0)

If (Count parameters>=1)
   $array_pointer:=$1
   Case of
     : (Nil($array_pointer))
       ArrayStatsError_Set (2)` A nil pointer was passed to a statistical array function.
     : ((Type($array_pointer->)#Real array )
       & (Type($array_pointer->)#Integer array )
       & (Type($array_pointer->)#LongInt array ))
       ArrayStatsError_Set (3)` A pointer to a non-numeric array was passed.
     : (Size of array($array_pointer->)=0)`  This may or may not be considered an error.
       ArrayStatsError_Set (4)` A pointer to an empty array was passed.
   End case
Else
   ArrayStatsError_Set (1)` Not enough parameters were passed to a statistical array function.
End if

If (ArrayStatsError_Get =0)` Array is of acceptable type and has at least 1 element.
   ` ------------------------
   ` Run Code of Routine
   ` ------------------------
   C_LONGINT($array_size)
   $array_size:=Size of array($array_pointer->)

   $max_real:=$array_pointer->{1}

   C_LONGINT($element)
   For ($element;2;$array_size)
     If ($array_pointer->{$element}>$max_real)
       $max_real:=$array_pointer->{$element}
     End if
   End for
End if

` ------------------------
` Return Result
` ------------------------
$0:=$max_real
```

The parameter testing code at the top of each routine prevents the working code of each routine from operating on inappropriate pointers or, where relevant, dividing by zero. However, the error checking code makes the routines longer. If updating or studying the code, keep the following points in mind:

- The error testing and functional portions of the method are kept separate, whenever practical.
- The functional portion of the routine is relatively simple because it doesn't need to test for dangerous conditions.

Code listings in this note leave out the error testing code implemented in the sample database.

ArrayStats_GetMean

(->Non-empty numeric array) : Mean average
(Pointer) : Real

This routine returns the mean average of the values in an array. The working code is listed below:

```
C_REAL($sum_real)
C_LONGINT($values_count)
$sum_real:=ArrayStats_GetSum ($array_pointer)
$values_count:=Size of array($array_pointer->)
$mean_real:=$sum_real/$values_count

$0:=$mean_real
```

ArrayStats_GetMedian

(->Non-empty numeric array;{Sort original?}) : Median value
(Pointer;{Boolean}) : Real

This routine returns the median value found in an array. By default, this routine sorts the original array. To leave the original intact, pass **False** in the optional parameter $2. However, when the original is not sorted a duplicate array is produced, which takes extra memory. The working code is listed below:

```
$sortOriginal_b:=True
If (Count parameters>=2)
   $sortOriginal_b:=$2
End if

If ($sortOriginal_b)
   SORT ARRAY($array_pointer->)` The easiest way to find the midpoint is by sorting the array.
Else ` Leave the original array intact.
      ` The easiest way to figure out the median is with a sorted array.
   ` Since we're not sorting the original, create a temporary copy
   ` for sorting.
   ARRAY REAL(ArrayStats_MedianValuesCopy_ar;0)
   ARRAY INTEGER(ArrayStats_MedianValuesCopy_ai;0)
   ARRAY LONGINT(ArrayStats_MedianValuesCopy_al;0)

   C_LONGINT($array_type)
   $array_type:=Type($array_pointer->)

   ` Copy the values and Update the array pointer for the working code.
   Case of
     : ($array_type=Real array )
       COPY ARRAY($array_pointer->;ArrayStats_MedianValuesCopy_ar)
       $array_pointer:=->ArrayStats_MedianValuesCopy_ar

     : ($array_type=Integer array )
       COPY ARRAY($array_pointer->;ArrayStats_MedianValuesCopy_ai)
       $array_pointer:=->ArrayStats_MedianValuesCopy_ai

     : ($array_type=LongInt array )
       COPY ARRAY($array_pointer->;ArrayStats_MedianValuesCopy_al)
       $array_pointer:=->ArrayStats_MedianValuesCopy_al

     Else ` Bug in this routine.
       ArrayStatsError_Set (13)` Did not detect non-numeric array type.
   End case
End if
If (ArrayStatsError_Get =0)
   C_LONGINT($elements_count)
   $elements_count:=Size of array($array_pointer->)

   C_LONGINT($firstElement_index)
   $firstElement_index:=$elements_count\2

   If (Mod($elements_count;2)=1)` Odd sized.
     C_LONGINT($element)
     $element:=1+$firstElement_index
     $median_real:=$array_pointer->{$element}
   Else
     $median_real:=$array_pointer->{$firstElement_index}
     $median_real:=$median_real+$array_pointer->{$firstElement_index+1}
     $median_real:=$median_real/2
   End if

   If (Not($sortOriginal_b))` The code worked on a temporary copy, clear it out.
   ` Remember: $array_pointer now points to the temporary array.
     DELETE ELEMENT($array_pointer->;1;Size of array($array_pointer->))
   End if
End if
```

ArrayStats_GetMin

(->Non-empty numeric array) : Minimum value found
(Pointer) : Real

This routine returns the minimum value found in an array. The working code of the routine is listed below:

```
C_LONGINT($array_size)
$array_size:=Size of array($array_pointer->)

$min_real:=$array_pointer->{1}

C_LONGINT($element)
For ($element;2;$array_size)
   If ($array_pointer->{$element}<$min_real)
      $min_real:=$array_pointer->{$element}
   End if
End for

$0:=$min_real
```

ArrayStats_GetMax

(->Non-empty numeric array) : Maximum value found
(Pointer) : Real

This routine returns the maximum value found in an array. The working code is listed below:

```
C_LONGINT($array_size)
$array_size:=Size of array($array_pointer->)

$max_real:=$array_pointer->{1}

C_LONGINT($element)
For ($element;2;$array_size)
   If ($array_pointer->{$element}>$max_real)
      $max_real:=$array_pointer->{$element}
   End if
End for

$0:=$max_real
```

ArrayStats_GetMode

(->Non-empty numeric array;->Mode values array;{->Frequency array}) :
(Pointer;Pointer;{Pointer}) :

This routine scans an array of values calculating the mode. Since a series of values may have zero or more modes, the output of this routine is placed in an array, not $0. Optionally, the output may include a second array with the frequency count for each modal value. As an example, consider an example with one modal value:

Input Values
1
2
5
2
4

Output Values	Output Counts
2	2

Alternatively, consider an example with two modal values:

Input Values
1
2
5
2
5

Output Values	Output Counts
2	2
5	2

Note that the input values and output values array must be of the same type. The output counts array, if used, may be of any numeric type. The main working code is listed below (whitespace adjusted for legibility):

```
` Get the unique values from the value arrays along with a count of each time they appear.
ArrayStats_GetFrequencyCounts ($valueArray_pointer;$uniqueValuesArray_pointer;
    ->ArrayStats_ModeUniqueCounts_al)

` Find any modal values.
C_LONGINT($count_min)
C_LONGINT($count_max)
$count_min:=ArrayStats_GetMin (->ArrayStats_ModeUniqueCounts_al)
$count_max:=ArrayStats_GetMax (->ArrayStats_ModeUniqueCounts_al)

If ($count_min=$count_max)` There's no mode.
Else

   C_LONGINT($startFrom_index)
   $startFrom_index:=1
   Repeat
     C_LONGINT($modalValue_index)
     $modalValue_index:=Find in array
       (ArrayStats_ModeUniqueCounts_al;$count_max;$startFrom_index)

     If ($modalValue_index>=1)
       $startFrom_index:=$modalValue_index+1` Make next find in array step past current match.

       C_LONGINT($newElementInOutputArrays_index)
       $newElementInOutputArrays_index:=Size of array($modeValuesArray_pointer->)+1
       INSERT ELEMENT($modeValuesArray_pointer->;$newElementInOutputArrays_index;1)
       $modeValuesArray_pointer->{$newElementInOutputArrays_index}:=
         $uniqueValuesArray_pointer->{$modalValue_index}

       If ($returnFrequencyCounts_b)
         INSERT ELEMENT($frequencyCountArray_pointer-
>;$newElementInOutputArrays_index;1)
         $frequencyCountArray_pointer->{$newElementInOutputArrays_index}:=
         ArrayStats_ModeUniqueCounts_al{$modalValue_index}
       End if

     End if

   Until ($modalValue_index<=0)

End if
```

ArrayStats_GetFrequenceyCounts

(->Non-empty numeric array;->Unique values array;->Frequency array) : (Pointer;Pointer;Pointer) :

This routine builds an array of unique value and a parallel array of frequency counts based on the values in a source array. The source values and output unique values arrays must be of the same type. The final results are sorted by value. The example below demonstrates the routine's behavior:

Input Values
1
2
5
2
4

Output Values	Output Counts
1	1
2	2
4	1
5	1

The main working code is listed below (whitespace adjusted for legibility):

```
C_LONGINT($sourceArray_size)
C_LONGINT($sourceArray_index)
$sourceArray_size:=Size of array($valueArray_pointer->)
For ($sourceArray_index;1;$sourceArray_size)
   C_LONGINT($positionInUniqueValues_index)
   $positionInUniqueValues_index:=Find in array
     ($uniqueValuesArray_pointer->;($valueArray_pointer->{$sourceArray_index}))
   If ($positionInUniqueValues_index<1)
     ` Not listed in the unique values+count arrays yet.
     ` Append items to unique values+count arrays.
     $positionInUniqueValues_index:=Size of array($uniqueValuesArray_pointer->)+1
     INSERT ELEMENT($uniqueValuesArray_pointer->;$positionInUniqueValues_index)
     INSERT ELEMENT($frequencyCountArray_pointer->;$positionInUniqueValues_index)
     $uniqueValuesArray_pointer->{$positionInUniqueValues_index}:=
       ($valueArray_pointer->{$sourceArray_index})
   End if

   $frequencyCountArray_pointer->{$positionInUniqueValues_index}:=
      ($frequencyCountArray_pointer->{$positionInUniqueValues_index})+1
End for

SORT ARRAY($uniqueValuesArray_pointer->;$frequencyCountArray_pointer->)
```

Note ArrayStats_GetFrequencyCounts *is used by* ArrayStats_GetMode *and is included in this technical note as a convenience*

ArrayStats_GetRange

(->Non-empty numeric array) : Maximum value found
(Pointer) : Real

This routine returns the range of value found in an array. Range is the distance between the smallest and largest values. The working code of the routine is listed below:

```
C_REAL($min_real)
C_REAL($max_real)
$min_real:=0
$max_real:=0

C_LONGINT($array_size)
$array_size:=Size of array($array_pointer->)

$min_real:=$array_pointer->{1}
$max_real:=$array_pointer->{1}


C_LONGINT($element)
For ($element;2;$array_size)

   If ($array_pointer->{$element}<$min_real)
      $min_real:=$array_pointer->{$element}
   End if

   If ($array_pointer->{$element}>$max_real)
      $max_real:=$array_pointer->{$element}
   End if

End for

$range_real:=$max_real-$min_real

$0:=$range_real
```

ArrayStats_GetStandardDeviation

(->Non-empty numeric array;{Type}) : Standard deviation
(Pointer;{Text}) : Real

This routine returns the standard deviation of the values in an array. The working code of the routine is listed below:

```
C_REAL($variance_real)
$variance_real:=ArrayStats_GetVariance ($array_pointer;$type_s)

If (ArrayStatsError_Get =0)` ArrayStats_GetVariance may set an error not detected above.
   $standardDeviation_real:=$variance_real^0.5` Square root of variance
End if

$0:=$standardDeviation_real
```

If used, the optional type parameter should contain one of two selectors:

population
sample

If the type parameter is not passed, the routine defaults to "sample".

Background:
Standard deviation is a common measurement of the average variance within a series. The simplest and most often used formula to calculate standard deviation is to take the square root of variance. See the notes on sum squares below for more information.

ArrayStats_GetSum

(->Non-empty numeric array) : Sum
(Pointer) : Real

This routine returns the sum of the values found in an array. The working code of the routine is listed below:

```
C_LONGINT($values_count)
$values_count:=Size of array($array_pointer->)
C_LONGINT($value_index)
For ($value_index;1;$values_count)
    $sum_real:=$sum_real+$array_pointer->{$value_index}
End for
$0:=$sum_real
```

ArrayStats_GetSumSquares

(->Non-empty numeric array) : Sum squares
(Pointer) : Real

This routine returns the sum squares of the values in an array. The working code of the routine is listed below:

```
C_REAL($mean_real)
$mean_real:=ArrayStats_GetMean ($array_pointer)
C_LONGINT($array_size)
$array_size:=Size of array($array_pointer->)
C_LONGINT($element)
C_REAL($distanceFromMean_real)
For ($element;1;$array_size)
    $distanceFromMean_real:=$mean_real-$array_pointer->{$element}
    $sumSquares_real:=$sumSquares_real+($distanceFromMean_real*$distanceFromMean_real)
End for
$0:=$sumSquares_real
```

Background:
Sum squares is a fundamental calculation in basic statistics that contributes to calculating both variance and standard deviation. The idea behind variance and standard deviation is to measure the degrees of variation within a series. To estimate the variability of a set of numbers we can sum the distance of each value from the mean of values. Unfortunately, this method always produces a sum of 0. (The mean is the average of the values, therefore the sum of distances from the average *has* to be zero.) To work around this problem, the distance of each value from the mean is squared. This method tends to produce a more useful number. Variance is approximately the average of the sum squares while standard deviation is approximately the square root of the variances.

ArrayStats_GetSumSquaresAlt

(->Non-empty numeric array) : Sum squares
(Pointer) : Real

This routine returns the sum squares of the values in an array calculated using the "computational method". This technique is easier when calculating sum squares by hand and is not as widely used as the method implemented in *ArrayStats_GetSumSquares*. This alternate method is included for anyone who needs to produce results matching a system based on the computational method. The working code of the routine is listed below:

```
C_REAL($mean_real)
$mean_real:=ArrayStats_GetMean ($array_pointer)

C_REAL($sum_real)
C_REAL($sumSquares_real)
$sum_real:=0
$sumSquares_real:=0

C_LONGINT($array_size)
$array_size:=Size of array($array_pointer->)

C_LONGINT($element)
For ($element;1;$array_size)
   C_REAL($value)
   $value:=$array_pointer->{$element}
   $sum_real:=$sum_real+$value
   $sumSquares_real:=$sumSquares_real+($value*$value)
End for

$sumSquaresAlt_real:=$sumSquares_real-(($sum_real*$sum_real)/$array_size)

$0:=$sumSquaresAlt_real
```

Note *Sum squares contributes to both variance and standard deviation. If matching results of a system based on the computational formula, both* ArrayStats_GetVariance *and* ArrayStats_GetStandardDeviation *should be duplicated or modified to call* ArrayStats_GetSumSquaresAlt*.*

ArrayStats_GetVariance

(->Non-empty numeric array;{Type}) : Variance
(Pointer;{Text}) : Real

This routine returns the variance of the values in an array. The working code of the routine is listed below:

```
C_REAL($sumSquares_real)
$sumSquares_real:=ArrayStats_GetSumSquares ($array_pointer)

C_LONGINT($values_count)
$values_count:=Size of array($array_pointer->)

If ($type_s="sample")
   $variance_real:=$sumSquares_real/($values_count-1)
Else ` population
   $variance_real:=$sumSquares_real/$values_count
End if

$0:=$variance_real
```

If used, the optional type parameter should contain one of two selectors:

population
sample

If the type parameter is not passed, the routine defaults to "sample".

ArrayStats_GetWeightedMean

(->Non-empty numeric value array;->Non-empty weighting array) : Weighted mean average
(Pointer;Pointer) : Real

This routine returns the weighted mean of an array. The working code of the routine is listed below (whitespace adjusted for legibility):

```
C_LONGINT($elements_count)
$elements_count:=Size of array($valueArray_pointer->)

C_REAL($weightedValues_sum)
C_REAL($weights_sum)
C_LONGINT($element_index)
$weightedValues_sum:=0
$weights_sum:=0

For ($element_index;1;$elements_count)
   C_REAL($weight)
   $weight:=$weightingArray_pointer->{$element_index}

   $weightedValues_sum:=
      $weightedValues_sum+($valueArray_pointer->{$element_index}*$weight)
   $weights_sum:=$weights_sum+$weight

End for

If ($weights_sum#0)
   $weightedMean_real:=$weightedValues_sum/$weights_sum
Else ` Avoid division by zero. This result may or may not be considered an error.
   ArrayStatsError_Set (8)` ArrayStats_GetWeightedMean detected a weight sum of zero.
End if

$0:=$weightedMean_real
```

Background:
Weighted means are useful when the values in a series don't all have the same importance, for example, when some data is known or suspected to be of lower quality or has subjectively been rated as less significant. A classic application for weighted means is calculating a single grade based on several inputs, for example, a term paper might contribute 40% of a final grade while two tests scores each contribute 30% of the total.

Error Management Routines

Overview

The statistical functions detect and block possible error conditions, such as nil pointers and division by zero. Whenever an error is detected, an error code is set and the function returns 0. The error management routines are documented below.

ArrayStatsError_Get

(): Error code
() : Longint

This routine returns the last set ArrayStats error value.

ArrayStatsError_GetDefinition

(Error code): Error description
(Longint) : Text

This routine returns descriptive text associated with an ArrayStats error code. The table below shows the defined error codes and their descriptions:

Code	Description
1	Not enough parameters were passed to a statistical array function.
2	A nil pointer was passed to a statistical array function.
3	A pointer to a non-numeric array was passed to statistical array function.
4	A pointer to an empty array was passed to statistical array function.
5	An unrecognized comparison type was passed to a statistical array function. The type must equal either 'population' or 'sample'.
6	Can't calculate the variance on a sample of less than two elements.
7	Passing unevenly sized arrays to ArrayStats_GetWeightedMean.
8	ArrayStats_GetWeightedMean detected a weight sum of zero.
9	No error code passed to ArrayStatsError_Set.
10	The source value and mode value arrays passed to ArrayStats_GetMode must be of the same numeric type.
11	ArrayStats_GetMode internal bug: did not detect non-numeric array type.
12	The source value and unique value arrays passed to ArrayStats_GetFrequencyCounts must be of the same numeric type.

13 ArrayStats_GetMedian internal bug: did not detect non-numeric array type.

14 No error code passed to ArrayStatsError_GetDefinition.

ArrayStatsError_Set

(Error code):
(Longint) :

This routine sets the current ArrayStats error code.
Primarily, this routine is for use by the ArrayStats functions.

Summary

4th Dimension's native statistical functions are only useful with records and often only during **PRINT SELECTION**. Fortunately, it's not difficult to write enhanced statistical functions for arrays using native 4th Dimension code. These tools are implemented in the sample database accompanying this note and described above.