Datamanip Toolbox Manual

Chapter 2

Operator Descriptions

Chapter 2 - Operator Descriptions

A. Introduction

This chapter contains the man1 pages for all programs in the Datamanip toolbox, organized alphabetically by object name. Any man1 page can be accessed by typing the following command.

```
% kman <program_name>
```

This information can also be accessed by selecting the *Help* button on the program's graphical user interface, which is displayed when running the program from within cantata, or when running it from the command line using the *-gui* option.

The keywords (-k) option in kman allows the user to provide a keyword string for searching. When invoked with the *-k* option, kman will perform the search and provide a list of the names and short descriptions of programs found by the search.

```
% kman -k "keyword string"
```

If you wish to display only the command line usage information for a program, enter the program name, followed by -U.

```
% <program_name> -U
```

B. General Operator Behavior

In the following printed man1 pages, descriptions of general operator behavior with respect to the polymorphic data model are referenced back to the *General Operator Behavior* section in Chapter 1 of the Datamanip toolbox manual. Exceptions to the general behavior are explicitly called out in the man1 pages. When using online help and kman, these sections will automatically be included in the text. The part of the system that helps generate this type of referencing is still under development, so specific chapter sections are not called out in the manual pages - rather, the path to the general information file is given as a tag for the section in Chapter 1. This will be fixed by the next release of the system.

C. Polymorphic Data Processing Operators

C.1. kabs — *Output = Absolute Value of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Absolute Value*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

Output = abs(Input)

The *Absolute Value* operator returns the absolute value of each data point in the input data object, **Input**. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Absolute Value* runs the program *karith1* with the -abs flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kabs: Output = Absolute Value of Input
% kabs
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.2. kabsdiff — *Output = Absolute Value of (Input 1 - (Input 2 or Constant))*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Absolute Diff*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

Output = Absolute Value(Input 1 - (Input 2 or a Constant))

The *Absolute Difference* operator computes the absolute value of the result of subtracting either **Input 2** or the **Constant** value, which ever is specified by the user, from **Input 1**.

Executing *Absolute Difference* runs the program *karith2* with the -absdiff flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real.

Command Line Arguments

```
Usage for kabsdiff: Output = Absolute Value of (Input 1 - (Input 2 or Constant))
% kabsdiff
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real]  (double) Real constant value
             (unbounded, default = 0)
```

C.3. kadd — *Output = Input 1 + (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Add*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

Output = Input 1 + (Input 2 or a Constant)

The *Add* operator adds each point in **Input 1** with either **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Add* runs the program *karith2* with the -add flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Command Line Arguments

```
Usage for kadd: Output = Input 1 + (Input 2 or Constant)
% kadd
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) Second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 0)
      [-imag] (double) Imaginary constant value
              (unbounded, default = 0)
```

C.4. kappend — *Append (or Prepend) Input Objects to Base Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*Append*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Append* operator (kappend) combines all input data objects (i1 to i10) into a single output object (o). The objects are combined in order from Input 1 to Input 10. If the user specifies placement as "Prepend Objects to Base Object" (-append 0), then the first input will be the last appended to the output object. The dimension flags Width (w_flag), Height (h_flag), Depth (d_flag), Time (t_flag), and Elements (e_flag), specify which dimension the objects will be stacked along. For example, stacking two images along width would produce an output with the images side by side. One and only one of the dimension flags can be true.

Value Data

All input objects must have value data. The value size of each input object will be zero-padded to the maximum of all sizes, except in the stacking dimension. The output size in the stacking dimension will be the sum of that dimension across all input objects. The output object data type will be the highest data type of the input data objects.

Map Data

When some or all of the input objects have map data, the treatment of the maps, and how the data is represented in the output object, depends on the mapping option (mapping) specified by the user. Possible mapping options are: (0) Map Data Thru Maps, (1) Use First Map Only, and (2) Create a Map for Each Input. If no input objects have maps, the mapping option is ignored. If there are doubts about which mapping option to use, the safest bet is to map the data thru the maps.

Map Data Thru Maps:

All data will be mapped before the data objects are combined. The output object will not have a map.

Use First Map Only:

In this case, the map data and color attributes of the first input object that has a map are directly transferred to the output object. The depth, time, and elements dimensions of that map must be 1, otherwise this mapping option will fail. Note that by selecting this mapping option, you are assuming that the value segments of all objects being appended have valid indices into that map.

Create a Map for Each Input:

(All input objects may have maps.) For each input object that does not have a map, one will be created for it. The the output object color attributes will be copied from the first input object that has a map. New map values will be assigned using the autocolor attribute.

The map dimension corresponding to the stacking dimension (Depth, Time, or Elements) will reflect the new size of the value data in that dimension, and the maps from each object will be appended accordingly. The maximum width and height of all maps will be propagated to the output, with zero-padding being applied where necessary. Padding in the other map dimensions may occur as well. For example, if objects with the following map and value Depth sizes are being stacked along the Depth dimension,

obj1 DEP=3 MAP_DEP = 3
obj2 DEP=4 MAP_DEP = 1
obj2 DEP=2 MAP_DEP = 2

the resulting value data will have a depth of 9, so the map depth will be 9 too. In this example, the second object's map will be zero-padded to have a map depth of four.

If the objects are being combined along the width or height dimensions, this mapping option will fail.

The output object map data type will be the highest of the input object map data types.

Mask Data

If any input object has mask data, then the output object will also have mask data. The mask data will be resized in the same manner as value data is resized. The mask will be padded as valid (1). The highest data type of input masks will be propagated to the output mask.

Location and Time Data

In all cases, the time and location data and/or attributes of the first input object will transferred to the output object. If location or time data is propagated, data values along the stacking dimension will not be changed, and therefore, the location or time data may become invalid.

See Also

DATAMANIP::kinset

Command Line Arguments

```
Usage for kappend: Append (or Prepend) Input Objects to Base Object
% kappend
  -i1        (infile)  Base Input data object (input 1)
  -o         (outfile) Resulting output data object

  [-i2]      (infile)  input file 2
             (default = {none})
  [-i3]      (infile)  input file 3
             (default = {none})
  [-i4]      (infile)  input file 4
             (default = {none})
  [-i5]      (infile)  input file 5
             (default = {none})
  [-i6]      (infile)  input file 6
             (default = {none})
  [-i7]      (infile)  input file 7
             (default = {none})
  [-i8]      (infile)  input file 8
             (default = {none})
  [-i9]      (infile)  input file 9
             (default = {none})
  [-i10]     (infile)  input file 10
             (default = {none})
  [-append]  (boolean) If True, append objects to base, otherwise prepend
             (default = true)
  [-mapping] (list)    Defines map treatment when some, but not all, objects have maps
          0  "Map Data Thru Maps ",
          1  "Use First Map Only",
          2  "Create Map for Each Input"
          [default: 0 "Map Data Thru Maps "]

  [Optional Mutually Exclusive Group - Specify one or none of the 5 options:]
    [-w] (flag) Stack objects along width dimension
    [-h] (flag) Stack objects along height dimension
    [-d] (flag) Stack objects along depth dimension
    [-t] (flag) Stack objects along time dimension
    [-e] (flag) Stack objects along elements dimension
```

C.5. karith1 — *Single Operator Pointwise Arithmetic*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*1-Operand Arith*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

karith1 performs the operation specified by the operator flag (-abs, -sqrt, etc.) on each point in the input data object, **Input**. The *karith1* operators are: Absolute Value, Square Root, Cube Root, Negative, Reciprocal, Ceiling, Floor, Truncate, Fraction, Lgamma, Error Function, Complementary Error Function, Bitwise NOT, Bitwise CLEAR, Bitwise SET, Sine, Arc Sine, Cosine, Arc Cosine, Tangent, Arc Tangent, Hyperbolic Sin and Arc Sine, Hyperbolic Cosine and Arc Cosine, Hyperbolic Tangent and Arc Tangent, and the sinc (sin(x)/x) function.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Restrictions

Operations on complex data are not supported at this time.

For the bitwise operations (not, set, clear), if the source data is floating point, it is cast to long integer before processing. This may result in loss of precision.

Command Line Arguments

```
Usage for karith1: Single Operator Pointwise Arithmetic
% karith1
  -i         (infile)  input data object
  -o         (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 28 options:
    [-abs]    (flag) Output = Absolute Value of input
```

```
[-sin]    (flag) Output = Sine of Input
[-sqrt]   (flag) Output = Square Root of Input
[-asin]   (flag) Output = Arc Sine of Input
[-cbrt]   (flag) Output = Cube Root of Input
[-cos]    (flag) Output = Cosine of Input
[-neg]    (flag) Output = Negative of Input
[-acos]   (flag) Output = Arc Cosine of Input
[-recip]  (flag) Output = Reciprocal of Input
[-tan]    (flag) Output = Tangent of Input
[-ceil]   (flag) Output = integer Ceiling of Input
[-atan]   (flag) Output = Arc Tangent of Input
[-floor]  (flag) Output = Integer Floor of Input
[-sinh]   (flag) Output = Hyperbolic Sine of Input
[-trunc]  (flag) Output = integer Truncate of Input
[-asinh]  (flag) Output = Hyperbolic Arc Sine of Input
[-fract]  (flag) Output = Fractional Part of Input
[-cosh]   (flag) Output = Hyperbolic Cosine of Input
[-lgamma] (flag) Output = Natural log  of Gamma Function of Input
[-acosh]  (flag) Output = Hyperbolic Arc Cosine of Input
[-erf]    (flag) Output = Error Function of Input
[-tanh]   (flag) Output = Hyperbolic Tangent of Input
[-erfc]   (flag) Output = Complementary Error Function of Input
[-atanh]  (flag) Output = Hyperbolic Arc Tangent of Input
[-sinc]   (flag) Output = Sinc Function:  sin(x)/x
[-not]    (flag) Bitwise NOT operation
[-clear]  (flag) Bitwise CLEAR: set all valid bits to 0
[-set]    (flag) Bitwise SET: set all valid bits to 1
```

C.6. karith2 — *Dual Operator Pointwise Arithmetic*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*2-Operand Arith*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

karith2 performs the operation specified by the operator flag (-add, -sub, etc.) between each point in **Input 1** and either the corresponding point in **Input 2** or the real (and imaginary) **Constant** values, which ever the user specifies. The *karith2* operators are: Add, Subtract, Subtract From, Multiply, Divide, Divide Into, Power, Absolute Difference, Minimum, Maximum, Modulo, Hypotenuse, atan2, and ldexp.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are determined by the operation. The default pad value of (0.0, 0.0) will be used for the Add (-add), Subtract (-sub), Subtract From (-subfrom), Absolute Difference (-absdiff), Minimum (-min), Maximum (-max), Hypotenuse (-hypot), and ldexp operations. The default pad value of (1.0, 0.0) will be used for the Multiply (-mul), Divide (-div), Divide Into (-divinto), Power (-pow), Modulo (-mod) and atan2 operations.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::kabsdiff, DATAMANIP::kadd, DATAMANIP::katan2, DATAMANIP::kcube, DATAMANIP::kdeg2rad, DATAMANIP::kdiv, DATAMANIP::kdivinto, DATAMANIP::khypot, DATAMANIP::kldexp, DATAMANIP::kmaximum, DATAMANIP::kminimum, DATAMANIP::kmod, DATAMANIP::kmul, DATAMANIP::kpow, DATAMANIP::krad2deg, DATAMANIP::ksqr, DATAMANIP::ksub, DATAMANIP::ksubfrom

Restrictions

Operations on complex data are supported only for add, subtract, subtract from, multiply, divide, divide into, and power at this time. For other operations, if the input object is complex, only the real component of the data is processed, and the output object is real.

For additional complex operators, see DATAMANIP::kcmplx, DATAMANIP::kcmplx2real, DATAMANIP::kreal2cmplx, and the associated cantata pane objects (phase, magnitude, etc).

Command Line Arguments

```
Usage for karith2: Dual Operator Pointwise Arithmetic
% karith2
  -i1        (infile)  first input data object
  -o         (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 1)
      [-imag] (double) Imaginary constant value
              (unbounded, default = 1)

  Required Mutually Exclusive Group - Specify exactly one of the 14 options:
    [-add]     (flag) Add: o = i1 + (i2 or val)
    [-hypot]   (flag) Hypotenuse:  o = sqrt(i1*i1 + (i2*i2* or val*val))
    [-sub]     (flag) Subtract: o = i1 - (i2 or val)
    [-atan2]   (flag) atan2: o = arc tangent (i1 / (i2 or val)
    [-subfrom] (flag) Subtract From: o = (i2 or val) - i1
    [-mul]     (flag) Multiply: o = i1 * (i2 or val)
    [-div]     (flag) Divide: o = i1 / (i2 or val)
    [-ldexp]   (flag) ldexp: i1 * 2 ** (i2 or val)
    [-divinto] (flag) Divide Into:  o = (i2 or val) / i1
    [-pow]     (flag) Power: o = i1 raised to power of (i2 or val)
    [-absdiff] (flag) Absolute Difference:  o = | i1 - (i2 or val) |
    [-min]     (flag) Minimum: Returns lower value between i1 and (i2 or val)
    [-max]     (flag) Maximum: Returns higher value between i1 and (i2 or val)
    [-mod]     (flag) Modulus: o = remainder (i1/i2)
```

C.7. kasc2loc — *Import ASCII Data File to Location Segment*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*ASCII to Location*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine will import an ASCII data file and insert it into a data object if it is specified. If there is no source object to insert the new segment into, the data object specified as the output file will be created. The new segment will be inserted into this data object in the location segment. If a source object was specified, then all of its data segments and the global object attributes will be copied to the destination.

The index order is specified by setting the -windex, -hindex, -dindex and -nindex flags. They can not be set to the same value.

The start offset is the point to start importing in the ASCII data. All points before this one are ignored.

The skip factor is the number of points to skip in the ASCII data between reads. If the size of the new segment's lowest index is 40 and the skip factor is 5, then this routine will read in 40 points skip 5 and read in 40 more, etc. until the entire ASCII file read in.

Complex values in ASCII are represented by number pairs, surrounded by parenthesis and separated by a comma. For example, (8, 3) represents the complex number 8+3i.

See Also

kasc2val, kasc2map, kasc2time, kasc2mask, kimportasc

Command Line Arguments

```
Usage for kasc2loc: Import ASCII Data File to Location Segment
% kasc2loc
  -i1        (infile)  ASCII input data file containing new data segment
  -o         (outfile) output data object

  [-i2]      (infile)  data object input to insert the new segment into
             (default = {none})
  [-so]      (integer) starting point in ASCII file to start importing at
             (so >= 0, default = 0)
  [-sf]      (integer) number of points in ASCII file to skip between reads
             (sf >= 0, default = 0)
  [-wsize]   (integer) width size must be > 0
             (wsize > 0, default = 1)
  [-windex]  (list)    position in the index order for the width dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth"
          [default: 1 "First"]
  [-hsize]   (integer) height size must be > 0
             (hsize > 0, default = 1)
  [-hindex]  (list)    position in the index order for the height dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth"
          [default: 2 "Second"]
  [-dsize]   (integer) depth size must be > 0
             (dsize > 0, default = 1)
  [-dindex]  (list)    position in the index order for the depth dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth"
          [default: 3 "Third"]
  [-nsize]   (integer) dimension size must be > 0
             (nsize > 0, default = 1)
  [-nindex]  (list)    position in the index order for the dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth"
          [default: 4 "Fourth"]
  [-type]    (list)    data type
```

```
1  "Bit",
2  "Byte",
3  "Unsigned Byte",
4  "Short",
5  "Unsigned Short",
6  "Integer",
7  "Unsigned Integer",
8  "Long",
9  "Unsigned Long",
10  "Float",
11  "Double",
12  "Complex",
13  "Double Complex"
[default: 11 "Double"]
```

C.8. kasc2map — *Import ASCII Data File to Map Segment*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*ASCII to Map*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine will import an ASCII data file and insert it into a data object if it is specified. If there is no source object to insert the new segment into, the data object specified as the output file will be created. The new segment will be inserted into this data object in the map segment. If a source object was specified, then all of its data segments and the global object attributes will be copied to the destination.

The index order is specified by setting the -windex, -hindex, -eindex, -tindex and -dindex flags. They can not be set to the same value.

The start offset is the point to start importing in the ASCII data. All points before this one are ignored.

The skip factor is the number of points to skip in the ASCII data between reads. If the size of the new segment's lowest index is 40 and the skip factor is 5, then this routine will read in 40 points skip 5 and read in 40 more, etc. until the entire ASCII file read in.

Complex values in ASCII are represented by number pairs, surrounded by parenthesis and separated by a comma. For example, (8, 3) represents the complex number 8+3i.

See Also

kasc2val, kasc2loc, kasc2time, kasc2mask, kimportasc

Command Line Arguments

```
Usage for kasc2map: Import ASCII Data File to Map Segment
% kasc2map
  -i1        (infile)  ASCII input data file containing new data segment
```

```
-o         (outfile) output data object

[-i2]      (infile)  data object input to insert the new segment into
           (default = {none})
[-so]      (integer) starting point in ASCII file to start importing at
           (so >= 0, default = 0)
[-sf]      (integer) number of points in ASCII file to skip between reads
           (sf >= 0, default = 0)
[-wsize]   (integer) map width size must be > 0
           (wsize > 0, default = 1)
[-windex]  (list)    position in the index order for the map width dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 1 "First"]
[-hsize]   (integer) map height size must be > 0
           (hsize > 0, default = 1)
[-hindex]  (list)    position in the index order for the map height dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 2 "Second"]
[-dsize]   (integer) map depth size must be > 0
           (dsize > 0, default = 1)
[-dindex]  (list)    position in the index order for the map depth dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 3 "Third"]
[-tsize]   (integer) map time size must be > 0
           (tsize > 0, default = 1)
[-tindex]  (list)    position in the index order for the map time dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 4 "Fourth"]
[-esize]   (integer) map elements size must be > 0
           (esize > 0, default = 1)
[-eindex]  (list)    position in the index order for the map elements dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 5 "Fifth"]
[-type]    (list)    data type
        1  "Bit",
        2  "Byte",
        3  "Unsigned Byte",
        4  "Short",
        5  "Unsigned Short",
```

```
6  "Integer",
7  "Unsigned Integer",
8  "Long",
9  "Unsigned Long",
10  "Float",
11  "Double",
12  "Complex",
13  "Double Complex"
[default: 11 "Double"]
```

C.9. kasc2mask — *Import ASCII Data File to Mask Segment*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*ASCII to Mask*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine will import an ASCII data file and insert it into a data object if it is specified. If there is no source object to insert the new segment into, the data object specified as the output file will be created. The new segment will be inserted into this data object in the mask segment. If a source object was specified, then all of its data segments and the global object attributes will be copied to the destination.

The index order is specified by setting the -windex, -hindex, -dindex, -tindex and -eindex flags. They can not be set to the same value.

The start offset is the point to start importing in the ASCII data. All points before this one are ignored.

The skip factor is the number of points to skip in the ASCII data between reads. If the size of the new segment's lowest index is 40 and the skip factor is 5, then this routine will read in 40 points skip 5 and read in 40 more, etc. until the entire ASCII file read in.

Complex values in ASCII are represented by number pairs, surrounded by parenthesis and separated by a comma. For example, (8, 3) represents the complex number 8+3i.

See Also

kasc2val, kasc2loc, kasc2time, kasc2map, kimportasc

Command Line Arguments

```
Usage for kasc2mask: Import ASCII Data File to Mask Segment
% kasc2mask
  -i1        (infile)  ASCII input data file containing new data segment
  -o         (outfile) output data object

  [-i2]      (infile)  data object input to insert the new segment into
             (default = {none})
  [-so]      (integer) starting point in ASCII file to start importing at
```

```
             (so >= 0, default = 0)
[-sf]        (integer) number of points in ASCII file to skip between reads
             (sf >= 0, default = 0)
[-wsize]     (integer) width size must be > 0
             (wsize > 0, default = 1)
[-windex]    (list)    position in the index order for the width dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth",
          5  "Fifth"
          [default: 1 "First"]
[-hsize]     (integer) height size must be > 0
             (hsize > 0, default = 1)
[-hindex]    (list)    position in the index order for the height dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth",
          5  "Fifth"
          [default: 2 "Second"]
[-dsize]     (integer) depth size must be > 0
             (dsize > 0, default = 1)
[-dindex]    (list)    position in the index order for the depth dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth",
          5  "Fifth"
          [default: 3 "Third"]
[-tsize]     (integer) time size must be > 0
             (tsize > 0, default = 1)
[-tindex]    (list)    position in the index order for the time dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth",
          5  "Fifth"
          [default: 4 "Fourth"]
[-esize]     (integer) elements size must be > 0
             (esize > 0, default = 1)
[-eindex]    (list)    position in the index order for the elements dimension
          1  "First",
          2  "Second",
          3  "Third",
          4  "Fourth",
          5  "Fifth"
          [default: 5 "Fifth"]
[-type]      (list)    data type
          1  "Bit",
          2  "Byte",
          3  "Unsigned Byte",
          4  "Short",
          5  "Unsigned Short",
          6  "Integer",
          7  "Unsigned Integer",
          8  "Long",
          9  "Unsigned Long",
          10  "Float",
```

```
              11  "Double",
              12  "Complex",
              13  "Double Complex"
              [default: 11 "Double"]
```

C.10. kasc2time — *Import ASCII Data File to Time Segment*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*ASCII to Time*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine will import an ASCII data file and insert it into a data object if it is specified. If there is no source object to insert the new segment into, the data object specified as the output file will be created. The new segment will be inserted into this data object in the time segment. If a source object was specified, then all of its data segments and the global object attributes will be copied to the destination.

The start offset is the point to start importing in the ASCII data. All points before this one are ignored.

Complex values in ASCII are represented by number pairs, surrounded by parenthesis and separated by a comma. For example, (8, 3) represents the complex number 8+3i.

See Also

kasc2val, kasc2mask, kasc2loc, kasc2map, kimportasc

Command Line Arguments

```
Usage for kasc2time: Import ASCII Data File to Time Segment
% kasc2time
  -i1         (infile)  ASCII input data file containing new data segment
  -o          (outfile) output data object

  [-i2]       (infile)  data object input to insert the new segment into
              (default = {none})
  [-so]       (integer) starting point in ASCII file to start importing at
              (so >= 0, default = 0)
  [-tsize]    (integer) time
              (tsize > 0, default = 1)
  [-type]     (list)    data type
          1  "Bit",
          2  "Byte",
          3  "Unsigned Byte",
          4  "Short",
          5  "Unsigned Short",
          6  "Integer",
          7  "Unsigned Integer",
          8  "Long",
          9  "Unsigned Long",
```

```
10  "Float",
11  "Double",
12  "Complex",
13  "Double Complex"
[default: 11 "Double"]
```

C.11. kasc2val — *Import ASCII Data File to Value Segment*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*ASCII to Value*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine will import an ASCII data file and insert it into a data object if it is specified. If there is no source object to insert the new segment into, the data object specified as the output file will be created. The new segment will be inserted into this data object in the value segment. If a source object was specified, then all of its data segments and the global object attributes will be copied to the destination.

The index order is specified by setting the -windex, -hindex, -dindex, -tindex and -eindex flags. They can not be set to the same value.

The start offset is the point to start importing in the ASCII data. All points before this one are ignored.

The skip factor is the number of points to skip in the ASCII data between reads. If the size of the new segment's lowest index is 40 and the skip factor is 5, then this routine will read in 40 points skip 5 and read in 40 more, etc. until the entire ASCII file read in.

Complex values in ASCII are represented by number pairs, surrounded by parenthesis and separated by a comma. For example, (8, 3) represents the complex number 8+3i.

See Also

kasc2mask, kasc2loc, kasc2time, kasc2map, kimportasc

Command Line Arguments

```
Usage for kasc2val: Import ASCII Data File to Value Segment
% kasc2val
  -i1        (infile)  ASCII input data file containing new data segment
  -o         (outfile) output data object

  [-i2]      (infile)  data object input to insert the new segment into
             (default = {none})
  [-so]      (integer) starting point in ASCII file to start importing at
             (so >= 0, default = 0)
  [-sf]      (integer) number of points in ASCII file to skip between reads
             (sf >= 0, default = 0)
  [-wsize]   (integer) width size must be > 0
```

```
            (wsize > 0, default = 1)
[-windex]  (list)    position in the index order for the width dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 1 "First"]
[-hsize]   (integer) height size must be > 0
            (hsize > 0, default = 1)
[-hindex]  (list)    position in the index order for the height dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 2 "Second"]
[-dsize]   (integer) depth size must be > 0
            (dsize > 0, default = 1)
[-dindex]  (list)    position in the index order for the depth dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 3 "Third"]
[-tsize]   (integer) time size must be > 0
            (tsize > 0, default = 1)
[-tindex]  (list)    position in the index order for the time dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 4 "Fourth"]
[-esize]   (integer) elements size must be > 0
            (esize > 0, default = 1)
[-eindex]  (list)    position in the index order for the elements dimension
        1  "First",
        2  "Second",
        3  "Third",
        4  "Fourth",
        5  "Fifth"
        [default: 5 "Fifth"]
[-type]    (list)    data type
        1  "Bit",
        2  "Byte",
        3  "Unsigned Byte",
        4  "Short",
        5  "Unsigned Short",
        6  "Integer",
        7  "Unsigned Integer",
        8  "Long",
        9  "Unsigned Long",
        10  "Float",
        11  "Double",
        12  "Complex",
        13  "Double Complex"
        [default: 11 "Double"]
```

C.12. katan2 — *Output = Arc Tangent[Input 1 / (Input 2 or Constant)]*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Atan2*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Atan2* operator returns the Arc Tangent of the result produced by dividing each point in **Input 1** by the corresponding point in **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Atan2* runs the program *karith2* with the -atan2 flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (1.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real. See the *Phase* (kphase) or *Complex to Real* (kcmplx2real) operators for complex atan2 calculation.

Command Line Arguments

```
Usage for katan2: Output = Arc Tangent[Input 1 / (Input 2 or Constant)]
% katan2
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) Second input data object
            (default = {none})
    [-real] (double) Real constant value
            (unbounded, default = 1)
```

C.13. kaxis — *Reorient Data on the Width, Height, and Depth Axes*

Object Information

Category:	*Data Manip*	Subcategory:	*Reorganize Data*
Operator:	*Switch Axes*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kaxis reorients the input data to the specified output axes. For example, if the input data has the dimensions width = 100, height = 200 and depth = 20, and the user specifies that the input height data be assigned to the output width axis and the input width data be assigned to the output height axis, then the data would be reassigned so that the output dimensions would be width = 200, height = 100, depth = 20. Note that only one input axis may be assigned to each output axis.

The axis orientation is performed on value, mask, and location data, whenever they exist in the input object. The output data type is the same as the input data type. If map or time data exist, they are directly transferred to the output object, since they are not affected by width, height, or depth axis reassignments.

See Also

korient

Command Line Arguments

```
Usage for kaxis: Reorient Data on the Width, Height, and Depth Axes
```

```
% kaxis
  -i         (infile)  input data object
  -o         (outfile) resulting data object


  [Optional Mutually Exclusive Group - Specify one or none of the 3 options:]
    [-ww] (flag) assign input width data to output width axis
    [-wh] (flag) Assign input width data to output height axis
    [-wd] (flag) Assign input width data to output depth axis

  [Optional Mutually Exclusive Group - Specify one or none of the 3 options:]
    [-hw] (flag) Assign input height data to output width axis
    [-hh] (flag) Assign input height data to output height axis
    [-hd] (flag) Assign input height data to output depth axis

  [Optional Mutually Exclusive Group - Specify one or none of the 3 options:]
    [-dw] (flag) Assign input depth data to output width axis
    [-dh] (flag) Assign input depth data to output height axis
    [-dd] (flag) Assign input depth data to output depth axis
```

C.14. kbessel — *Compute Bessel Functions*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*Bessel*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Bessel* function returns the Bessel function of the specified kind and order for each data point in the input data object, **Input**. The data type of the output data object, **Output**, will always be the same as the data type of **Input**.

All processing is done in double precision float. If the input data is complex, only the real component of the data is processed.

Map Data

The Bessel function operator has not been written to be fully polymorphic yet. It does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing ' into the map. It is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the Bessel operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model.

All processing is done in double precision float. If the input data is complex, only the real component of the data is processed.

Command Line Arguments

```
Usage for kbessel: Compute Bessel Functions
% kbessel
  -i          (infile)  input data object
  -o          (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-j] (flag) Bessel function of the first kind
    [-y] (flag) Bessel function of the second kind

  Required Mutually Exclusive Group - Specify exactly one of the 3 options:
    [-ord0]  (flag)    Compute Bessel function of order 0
    [-ord1]  (flag)    Compute Bessel function of order 1
    [-order] (integer) Compute Bessel function of specified order
             (unbounded, default = 2)
```

C.15. kbitand — *Output = Input 1 AND (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*AND*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise AND operator, kbitand, performs a bitwise AND operation between each point in **Input 1** (i1) and each corresponding point in **Input 2** (i2), if the second input is specified. If Input 2 is not specified, the operation will be performed using the **Constant** value (real).

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects
Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **AND** runs the program *kbitwise* with the -and flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

In the future, the padding value used for this operation when the input images are of different sizes, will be modified from binary 0 to binary 1.

Command Line Arguments

```
Usage for kbitand: Output = Input 1  AND  (Input 2 or Constant)
% kbitand
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile)  Second input data object
             (default = {none})
    [-real]  (integer) Constant value
             (unbounded, default = 1)
```

C.16. kbitandinv — *Output = NOT(Input 1) AND (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*AND Inverted*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise AND Inverted operator performs a bitwise NOT on **Input 1** and then a bitwise AND between the result of the NOT operation and either the corresponding value in **Input 2** or the **Constant** value, which ever is specified by the user.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **ANDinv** runs the program *kbitwise* with the -andinv flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

In the future, the padding value used for ANDinv when the input images are of different sizes, will be modified from binary 0 to binary 1.

Command Line Arguments

```
Usage for kbitandinv: Output = NOT(Input 1)  AND  (Input 2 or Constant)
% kbitandinv
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile)  Second input data object
             (default = {none})
    [-real] (integer) Constant value
             (unbounded, default = 1)
```

C.17. kbitandrev — *Output = Input 1 AND NOT(Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*AND Reverse*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise AND Reverse operator performs a bitwise NOT on either **Input 2** or the **Constant** value, which ever is specified by the user, and then performs a bitwise AND between the result of the NOT operation and **Input 1**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **AND Reverse** runs the program *kbitwise* with the -andrev flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

In the future, the padding value used for this operation when the input images are of different sizes, will be modified from binary 0 to binary 1.

Command Line Arguments

```
Usage for kbitandrev: Output = Input 1  AND  NOT(Input 2 or Constant)
% kbitandrev
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile)  Second input data object
            (default = {none})
    [-real] (integer) Constant value
            (unbounded, default = 0)
```

C.18. kbitclear — *Set All Bits to Zero*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*CLEAR*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *CLEAR* operator sets all bits to zero on each data point in the input data object, **Input**.

Executing *CLEAR* runs the program *karith1* with the -clear flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

For the bitwise operations (not, set, clear), if the source data is floating point, it is cast to long integer before processing. This may result in loss of precision.

Command Line Arguments

```
Usage for kbitclear: Set All Bits to Zero
% kbitclear
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.19. kbitlshift — *Bitwise LEFT SHIFT of Input 1 by (Input 2 or Constant) Bits*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*Left Shift*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The **Left Shift** operator performs a bitwise left shift on each point in **Input 1** by the number of bits defined by either the corresponding point in **Input 2** or the **Constant** value, which ever is specified by the user. Zero's are shifted into the least significant bits. In effect each data value is multiplied by two.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **Left Shift** runs the program *kbitwise* with the -ls flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitlshift: Bitwise LEFT SHIFT of Input 1 by (Input 2 or Constant) Bits
% kbitlshift
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile)  Second input data object
            (default = {none})
    [-real] (integer) Constant value
            (unbounded, default = 0)
```

C.20. kbitnand — *Output = NOT(Input 1) OR NOT(Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*NAND*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise NAND operator performs a bitwise NAND between each point in **Input 1** and either **Input 2** or the **Constant** value, which ever is specified by the user.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **NAND** runs the program *kbitwise* with the -nand flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

In the future, the padding value used for this operation when the input images are of different sizes, will be modified from binary 0 to binary 1.

Command Line Arguments

```
Usage for kbitnand: Output = NOT(Input 1)  OR  NOT(Input 2 or Constant)
% kbitnand
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile)  Second input data object
             (default = {none})
    [-real] (integer) Constant value
             (unbounded, default = 1)
```

C.21. kbitnor — *Output = NOT(Input 1) AND NOT(Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*NOR*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise NOR operator performs a bitwise NOR operation between each point in **Input 1** and either **Input 2** or the **Constant** value, which ever is specified by the user.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **NOR** runs the program *kbitwise* with the -nor flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

In the future, the padding value used for this operation when the input images are of different sizes, will be modified from binary 0 to binary 1.

Command Line Arguments

```
Usage for kbitnor: Output = NOT(Input 1)  AND  NOT(Input 2 or Constant)
% kbitnor
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile)  Second input data object
            (default = {none})
    [-real] (integer) Constant value
            (unbounded, default = 0)
```

C.22. kbitnot — *Perform Bitwise NOT (Invert) Operation on Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*NOT*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

Output = NOT(Input) = ˜(Input)

The *NOT* operator performs a bitwise NOT, or invert, on each data point in the input data object, **Input**.

Executing *NOT* runs the program *karith1* with the -not flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

For the bitwise operations (not, set, clear), if the source data is floating point, it is cast to long integer before processing. This may result in loss of precision.

Command Line Arguments

```
Usage for kbitnot: Perform Bitwise NOT (Invert) Operation on Input
% kbitnot
  -i         (infile)  Input data object
```

```
-o        (outfile) Resulting output data object
```

C.23. kbitor — *Output = Input 1 OR (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*OR*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise OR operator performs a bitwise OR between each point in **Input 1** and either **Input 2** or the **Constant** value, which ever is specified by the user.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **OR** runs the program *kbitwise* with the -or flag.

See Also
DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitor: Output = Input 1  OR  (Input 2 or Constant)
% kbitor
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile)  Second input data object
            (default = {none})
    [-real] (integer) Constant value
            (unbounded, default = 0)
```

C.24. kbitorinv — *Output = NOT(Input 1) OR (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*OR Inverted*	Object Type:	*pane*
In Cantata:	*Yes*		

Description
The bitwise OR Inverted operator performs a bitwise NOT on **Input 1** and then a bitwise OR between the result of the NOT operation and either **Input 2** or the **Constant** value, which ever is specified by the

user.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **OR Inverted** runs the program *kbitwise* with the -orinv flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitorinv: Output = NOT(Input 1)  OR  (Input 2 or Constant)
% kbitorinv
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile)  Second input data object
            (default = {none})
    [-real] (integer) Constant value
            (unbounded, default = 0)
```

C.25. kbitorrev — *Output = Input 1 OR NOT(Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*OR Reverse*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise OR Reverse operator performs a bitwise NOT on either **Input 2** or the **Constant** value, which ever is specified by the user, and then performs a bitwise OR between the result of the NOT operation and **Input 1**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects
Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **OR Reverse** runs the program *kbitwise* with the -orrev flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitorrev: Output = Input 1  OR  NOT(Input 2 or Constant)
% kbitorrev
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object
```

```
Required Mutually Exclusive Group - Specify exactly one of the 2 options:
  [-i2]   (infile)  Second input data object
          (default = {none})
  [-real] (integer) Constant value
          (unbounded, default = 1)
```

C.26. kbitrshift — *Bitwise RIGHT SHIFT of Input 1 by (Input 2 or Constant) Bits*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*Right Shift*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The **Right Shift** operator performs a bitwise right shift on each point in **Input 1** by the number of bits defined by either the corresponding point in **Input 2** or the **Constant** value, which ever is specified by the user. Zero's are shifted into the most significant bits, and, in effect, each data value is divided by two and the remainder is discarded.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **Right Shift** runs the program *kbitwise* with the -rs flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitrshift: Bitwise RIGHT SHIFT of Input 1 by (Input 2 or Constant) Bits
% kbitrshift
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile)  Second input data object
             (default = {none})
    [-real]  (integer) Constant value
             (unbounded, default = 0)
```

C.27. kbitset — *Set All Bits to One*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*SET*	Object Type:	*pane*

In Cantata: *Yes*

Description

The *SET* operator sets all bits to one for each data point in the input data object, **Input**.

Executing *SET* runs the program *karith1* with the -set flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

For the bitwise operations (not, set, clear), if the source data is floating point, it is cast to long integer before processing. This may result in loss of precision.

Command Line Arguments

```
Usage for kbitset: Set All Bits to One
% kbitset
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.28. kbitwise — *Dual Operand Bitwise Operations*

Object Information

Category: *Arithmetic* Subcategory: *Bitwise Operators*

Operator: *Bitwise Ops* Object Type: *kroutine*
In Cantata: *No*

Description

The bitwise operator, kbitwise, performs a the specified bitwise operation between each point in **Input 1** (i1) and each corresponding point in **Input 2** (i2), if the second input is specified. If Input 2 is not specified, the operation will be performed using the **Constant** value (real).

Therefore, if kbitwise were called with two input objects:
kbitwise -i1 image:ball -i2 image:moon -o out -and
then the following operation would be performed
out = i1 & i2
Whereas if kbitwise were called with a single data object and a constant:
kbitwise -i1 image:ball -real 10 -o out -and
then the following would be performed:
o = i1 & 10

The supported functions are:
AND (and) o = i1 & (i2 or val)
OR (or) o = i1 | (i2 or val)
Exclusive Or (xor) o = i1 ^ (i2 or val)
NAND (nand) o = (~i1) | (~(i2 or val))
NOR (nor) o = (~i1) & (~(i2 or val))
AND Reverse (andrev) o = i1 & (~(i2 or val))
AND Inverted (andinv) o = (~i1) & (i2 or val)
OR Reverse (orrev) o = i1 | (~(i2 or val))
OR Inverted (orinv) o = (~i1) | i2
Unsigned Right Shift (rs) o = i1 >> by (i2 or val)
Unsigned Left Shift (ls) o = i1 << by (i2 or val)

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data
The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run kbitwise on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects
Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitwise: Dual Operand Bitwise Operations
% kbitwise
  -i1        (infile)  first input data object
  -o         (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 14 options:
    [-and]    (flag) Bitwise AND:  o = i1 & (i2 or val)
    [-ls]     (flag) Bitwise Left Shift:  o = i1 << by (i2 or val)
    [-or]     (flag) Bitwise OR:   o = i1 | (i2 or val)
    [-rs]     (flag) Bitwise Right Shift:  o = i1 >> by (i2 or val)
    [-xor]    (flag) Bitwise XOR:  o = i1 ^ (i2 or val)
    [-srs]    (flag) Bitwise Signed Right Shift by (i2 or val)
    [-nand]   (flag) Bitwise NAND: o = (~i1) | (~(i2 or val))
    [-rr]     (flag) Bitwise Right Rotate by (i2 or val)
    [-nor]    (flag) Bitwise NOR:  o = (~i1) & (~(i2 or val))
    [-lr]     (flag) Bitwise Left Rotate by (i2 or val)
    [-andrev] (flag) Bitwise AND Reverse:   o = i1 & (~(i2 or val))
    [-andinv] (flag) Bitwise AND Inverted:  o = (~i1) & (i2 or val)
```

```
    [-orrev]  (flag) Bitwise OR Reverse:    o = i1 | (~(i2 or val))
    [-orinv]  (flag) Bitwise OR Inverted:   o = (~i1) | i2


  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-i2]   (infile)  second input data object
            (default = {none})
    [-real] (integer) constant value
            (unbounded, default = 1)
```

C.29. kbitxor — *Output = Input 1 XOR (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Bitwise Operators*
Operator:	*XOR*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The bitwise XOR operator performs a bitwise Exclusive OR between each point in **Input 1** and either **Input 2** or the **Constant** value, which ever is specified by the user.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

The bitwise operators will not operate on float or complex data types.

Map Data

The bitwise operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the bitwise operation on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the bitwise operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the bitwise operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The bitwise operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing **XOR** runs the program *kbitwise* with the -xor flag.

See Also

DATAMANIP::kbitwise

Restrictions

THIS ROUTINE is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

All processing is currently performed as UNSIGNED long, so the operations may not work properly on negative values.

The bitwise operators will not operate on float or complex data types.

Command Line Arguments

```
Usage for kbitxor: Output = Input 1  XOR  (Input 2 or Constant)
% kbitxor
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile)  Second input data object
            (default = {none})
    [-real] (integer) Constant value
            (unbounded, default = 0)
```

C.30. kblend — *Output = (Input 1 * Constant) + (Input 2 (1 - Constant))*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Blend Data*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Blend* operator (kblend) forms a blend of two objects by adding X*(object1)+(1-X)*(object2).

Instead of specifying a constant for the blending ratio, an object can be supplied for the blending values. The value segment of the ratio object is used to determine the proportion in which the two objects are blended. It must take on values between 0 and 1, otherwise an error is generated.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask

If there is a validity mask associated with any of the input objects, the resulting output object will have a mask. The output object mask is the logical AND of the input object masks. If only one input object has a mask, then the other input object's mask is assumed to be "all valid" (mask value = 1). If a mask is resized for the operation, the mask padding value will be 1.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

Restrictions

The blend operator has not been updated to be fully polymorphic.

Command Line Arguments

```
Usage for kblend: Output = (Input 1 * Constant) + (Input 2 (1 - Constant))
% kblend
  -i1        (infile)  first input data object
  -i2        (infile)  second input file object
  -o         (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-ratio]  (double) Ratio of blending the two objects
              (0 <= ratio <= 1, default = 0.5)
    [-iratio] (infile) Input object for providing blending ratios
```

```
(default = {none})
```

C.31. kcbrt — *Output = Cube Root of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Cube Root*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Cube Root* operator returns the cubed root (value raised to the power of 1/3) of each data point in the input data object, **Input**. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Cube Root* runs the program *karith1* with the -cbrt flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kcbrt: Output = Cube Root of Input
% kcbrt
  -i         (infile)  Input data object
```

```
  -o         (outfile) Resulting output data object
```

C.32. kceiling — *Output = Integer Ceiling of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Ceiling*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Ceiling* operator returns the integer ceiling of each data point in the input data object, **Input**.

Executing *Ceiling* runs the program *karith1* with the -ceil flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kceiling: Output = Integer Ceiling of Input
% kceiling
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.33. kclip — *Clip or Threshold the Range of Values in Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Clip & Thresh*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

kclip performs clipping and thresholding operations on the input data object (i). Clipping limits the range of values in the data object, and thresholding generates a two-level output by thresholding the values in the object. For both operations, upper and lower cutoff values can be specified as either single constant values (lc, uc), or sets of values passed in as data objects (ilc, iuc). Output levels can be set using the true and false value options (tval, fval). The data type of the output object will be the same as the input object data type.

Clipping

When clipping is enabled (-thresh 0), the values in the object are clipped within the range specified by the upper and/or lower cutoff limits. In the default mode (where neither fval nor tval are selected) data values above the upper cutoff are set equal to the upper cutoff, and values below the lower cutoff are set equal to the lower cutoff value. If the fval parameter is provided, then data values below the lower cutoff are set to fval. If the tval parameter is provided, data values greater than the upper cutoff are set to that value. Exception: any time that both the upper and lower cutoff values are provided, and the upper cutoff is less than the lower cutoff, then data values between the two cutoffs are always set equal to fval.

Thresholding

If thresholding is enabled (-thresh 1), thresholding is performed using the specified lower and upper cutoff values, creating a two-level output data object. It is not necessary to specify both of the upper and lower cutoff values. If only the lower cutoff is specified, all values above it are in the threshold range. If only the upper cutoff is specified, all values below it are in the threshold range. The true value (tval) is assigned to values that are within the threshold range, and all values outside of the threshold range are set to fval.

The following four diagrams illustrate how the given operation is evaluated for different combinations of upper and lower cutoffs. The plots are example data histograms. Clipping and threshold output assignment rules are given following the diagrams.

Lower Cutoff Only:

```
lower cutoff result:       |<-F->|<----T---->|
                           |     | :     .   |
                           |    .:.:.  .:::  |
                           |:. .::::::.::::: |
                           |::::::::::::::::.|
                           ------|------------
                                 lower cutoff (lc)

   if value <  lc  cutoff result = False
   if value >= lc  cutoff result = True
```

Upper Cutoff Only:

```
   upper cutoff result:    |<----T---->|<-F->|
                           |       :   | .   |
                           |    .:.:.  .:::  |
                           |:. .::::::.::::: |
                           |::::::::::::::::.|
                           ------------|------
                                       upper cutoff (uc)

   if value >  uc  cutoff result = False
   if value <= uc  cutoff result = True
```

Upper Cutoff is Greater than Lower Cutoff:

When both upper and lower cutoff values are specified, and the upper cutoff is greater than the lower cutoff, the combined cutoff result is the logical AND of the individual results.

```
lower cutoff result:    |<-F->|<----T---->|
upper cutoff result:    |<----T---->|<-F->|

combined cutoff result: |<-F->|<-T->|<-F->|
                        |     | :   | .   |
                        |    .:.:.  .:::  |
                        |:. .:::::::.:::: |
                        |::::::::::::::::.|
                        ------|-----|------
                             lc     uc

if value <  lc OR  value >  uc  cutoff result = False
if value >= lc AND value <= uc  cutoff result = True
```

Upper Cutoff is Less than Lower Cutoff:

When both upper and lower cutoff values are specified, but the upper cutoff is less than the lower cutoff, the combined cutoff result is the logical OR of the individual results. (Specifying the lower cutoff to be greater than the upper cutoff inverts the operation.) In this case, the false value assigned to the output will always be fval for both clipping and thresholding.

```
upper cutoff result:    |<-T->|<----F---->|
lower cutoff result:    |<----F---->|<-T->|

combined cutoff result: |<-T->|<-F->|<-T->|
                        |     | :   | .   |
                        |    .:.:.  .:::  |
                        |:. .:::::::.:::: |
                        |::::::::::::::::.|
                        ------|-----|------
                             uc     lc

if value <  lc AND value >  uc  cutoff result = False
if value >= lc OR  value <= uc  cutoff result = True
```

Clipping results are assigned according to the following rules. ("Lower result" and "upper result" are either the clipping values or fval and tval - see discussion above.)

1. When lower cutoff < upper cutoff
   If the data value is less than the lower cutoff, set the output value to "lower result".

   If the data value is greater than the upper cutoff, set the output value to "upper result".

   Otherwise, the data value remains the same.

2. When upper cutoff < lower cutoff
   If the data value is less than the upper cutoff and it is greater than the lower cutoff, set output value to the "lower result".

   Otherwise, the data value remains the same.

Thresholding results are assigned according to the following rules.

1. When lower cutoff < upper cutoff
   If the data value is less than the lower cutoff, or it is greater than the upper cutoff, set the output value to fval.

   Otherwise, set the output value to tval.

2. When upper cutoff < lower cutoff
   If the data value is less than the lower cutoff and it is greater than the upper cutoff, set output value to fval.

   Otherwise, set the output value to tval.

When upper and lower cutoff values are equal, the only data values that are evaluated as TRUE are those that are equal to the cutoff values.

Data Type
    include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Objects of Different Sizes
    If, in addition to the input data object (i), upper or lower cutoff objects (ilc, iuc) are supplied, the source objects can be of different sizes. If the sizes are different, the size of the destination object's data will be the the same as that of the input data object. When resizing the cutoff objects, the data is zero padded, or truncated to the size of the input object. Alignment between the input files will always be the upper-left hand corner, position (0,0,0,0,0).

Map Data - Single Input
    include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Multiple Input Objects
    include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask - Single Input
    include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Multiple Input Objects
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

Examples

kclip -thresh 1 -i image:ball -lc 128 -uc 202 -tval 12 -fval 0 -o thresh.out

causes all pixels between the values on 128 and 202 (inclusively) to have a value of 12 in the output image. All other output image pixels have the value of zero.

kclip -i full_range.viff -uc 123 -lc 17 -o restricted_range.viff

Limits the range of values in the input file full_range.viff to 17...123 and writes the result to restricted_range.viff.

See Also

kbitwise

Restrictions

This program has not yet been updated to completely support the polymorphic data model.

Command Line Arguments

```
Usage for kclip: Clip or Threshold the Range of Values in Object
% kclip
  -i         (infile)  Input file
  -o         (outfile) output file

  Required Loose Group - Specify at least one of the 2 options:

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-lc]  (double) Constant lower cutoff value
             (unbounded, default = 128)
      [-ilc] (infile) File containing lower cutoff values
             (default = {none})

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-uc]  (double) Constant upper cutoff value
```

```
              (unbounded, default = 255)
       [-iuc] (infile) File containing upper cutoff values
              (default = {none})

    [-thresh]  (boolean) Clip (0) or Threshold (1) Object Data
               (default = false)
    [-fval]    (double)  Value assigned to output if evaluation is FALSE
               (unbounded, default = 0)
    [-tval]    (double)  Value assigned to output if evaluation is TRUE (thresholding only)
               (unbounded, default = 255)
```

C.34. kclipabove — *Clip Data Values that are Above Specified Cutoff*

Object Information

Category: *Data Manip* Subcategory: *Threshold & Clip Operators*
Operator: *Clip Above* Object Type: *pane*
In Cantata: *Yes*

Description

The *Clip Above* operator performs clipping on the Input data object (i), limiting the range of values in the object. Data values less than or equal to the Cutoff Value (uc) remain the same, and values above the Cutoff Value are set to the Cutoff Value. If the Alternate Output value parameter (fval) is specified, clipped data are set to that value instead of the Cutoff Value.

The following histogram defines the results of the clipping operation.

```
                        |<----T---->|<-F->|
                        |      | :      . |
                        |     .:.:.  .::: |
                        |:. .::::::.::::: |
                        |::::::::::::::::.|
                        ------------|------
                                   Cutoff

        T:  if value <= Cutoff,  Output = Input
        F:  if value >  Cutoff,  Output = Cutoff (or Alternate Value)
```

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Clip Above* runs the program *kclip* with the -thresh option set to FALSE (-thresh 0).

See Also

DATAMANIP::kclipbelow, DATAMANIP::kclipin, DATAMANIP::kclipout, DATAMANIP::kmegaclip, DATAMANIP::kclip

Command Line Arguments

```
Usage for kclipabove: Clip Data Values that are Above Specified Cutoff
% kclipabove
  -i         (infile)  Input file
  -o         (outfile) output file

  [-uc]      (double)  Upper cutoff - values above uc are clipped
             (unbounded, default = 128)
  [-tval]    (double)  Alternate value assigned to clipped output data
             (unbounded, default = 0)
```

C.35. kclipbelow — *Clip Data Values that are Below Specified Cutoff*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Clip Below*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Clip Below* operator performs clipping on the Input data object (i), limiting the range of values in the object. Data values greater than or equal to the Cutoff Value (lc) remain the same, and values below the Cutoff Value are set to the Cutoff Value. If the Alternate Output value parameter (fval) is specified, clipped data are set to that value instead of the Cutoff Value.

The following histogram defines the results of the clipping operation.

```
|<-F->|<----T---->|
|     | :      .  |
|    .:.:.  .:::  |
|:. .:::::::.:::: |
|::::::::::::::::.|
------|-----------
```

```
                              Cutoff

       T:  if value >= Cutoff,  Output = Input
       F:  if value <  Cutoff,  Output = Cutoff (or Alternate Value)
```

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Clip Below* runs the program *kclip* with the -thresh option set to FALSE (-thresh 0).

See Also

DATAMANIP::kclipabove, DATAMANIP::kclipin, DATAMANIP::kclipout, DATAMANIP::kmegaclip, DATAMANIP::kclip

Command Line Arguments

```
Usage for kclipbelow: Clip Data Values that are Below Specified Cutoff
% kclipbelow
  -i         (infile)  Input file
  -o         (outfile) output file

  [-lc]      (double)  Lower cutoff - values below lc are clipped
             (unbounded, default = 128)
  [-fval]    (double)  Alternate value assigned to clipped output data
             (unbounded, default = 0)
```

C.36. kclipin — *Clip Data Values Inside the Specified Range*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Clip Inside*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Clip Inside* operator (kclipin) performs clipping on the Input data object (i) within the range defined by the specified Lower and Upper Cutoff Values (uc, lc). Data values that are GREATER THAN the *Lower Cutoff Value* (uc), and LESS THAN the *Upper Cutoff Value* (lc) are are set to the Output Value (fval). All other data remain the same.

The following histogram defines the clipping operation.

```
          |<-T->|<-F->|<-T->|
          |     | :   | .   |
          |    .:.:.  .:::  |
          |:. .:::::::.:::: |
          |::::::::::::::::.|
          ------|-----|------
     Lower Cutoff     Upper Cutoff
              (uc)     (lc)

if value <  Upper Cutoff AND value >  Lower Cutoff   output = fval
if value >= Upper Cutoff OR  value <= Lower Cutoff   output = input
```

P "Data Type" 5 *include file* `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Clip Inside* runs the program *kclip* with the -thresh option set to FALSE (-thresh 0).

NOTE: When running kclipin from the command line, the cutoff parameters, uc and lc, are opposite to what would seem logical (uc = lower cutoff and lc = upper cutoff). This is due to the logic in the parent program, kclip.

See Also

DATAMANIP::kclipabove, DATAMANIP::kclipbelow, DATAMANIP::kclipout, DATAMANIP::kmegaclip, DATAMANIP::kclip

Command Line Arguments

```
Usage for kclipin: Clip Data Values Inside the Specified Range
% kclipin
  -i         (infile)  Input file
```

```
-o          (outfile) output file

[-uc]       (double)  Lower cutoff value (this is a reverse logic operation)
            (unbounded, default = 64)
[-lc]       (double)  Upper cutoff value (this is a reverse logic operation)
            (unbounded, default = 128)
[-fval]     (double)  Value assigned to output when data is GREATER THAN uc, AND LESS THAN lc
            (unbounded, default = 0)
```

C.37. kclipout — *Clip Data Values Outside of the Specified Range*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Clip Outside*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Clip Outside* operator (kclipout) performs clipping on the Input data object (i) outside of the range defined by the specified Lower and Upper Cutoff Values (lc, uc). All data values in the Input object that are LESS THAN the Lower Cutoff are set to the Lower Cutoff in the resulting Output object, unless the alternate "Lower Cutoff Output Value" (fval) is explicitly provided. If this value is provided, clipped data are set to that value instead of the Lower Cutoff. Likewise, data values GREATER THAN the Upper Cutoff are set to the Upper Cutoff, unless the alternate "Upper Cutoff Output Value" is explicitly provided. All other data remain the same.

The following histogram defines the clipping operation.

```
 |<-A->|<-B->|<-C->|
 |     | :   | .   |
 |    .:.:.  .:::  |
 |:. .:::::::.:::: |
 |::::::::::::::::.|
 ------|-----|------
      lc     uc

    A:  if value <  lc                  output = lc or fval
    B:  if value >= lc AND value <= uc  output = input
C:  if value >  uc                  output = uc or tval
```

P "Data Type" 5 *include file* `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Clip Outside* runs the program *kclip* with the -thresh option set to FALSE (-thresh 0).

See Also

DATAMANIP::kclipabove, DATAMANIP::kclipbelow, DATAMANIP::kclipin, DATAMANIP::kmegaclip, DATAMANIP::kclip

Command Line Arguments

```
Usage for kclipout: Clip Data Values Outside of the Specified Range
% kclipout
  -i         (infile)  Input file
  -o         (outfile) output file

  [-lc]      (double)  Lower cutoff value
             (unbounded, default = 64)
  [-uc]      (double)  Upper cutoff value
             (unbounded, default = 128)
  [-fval]    (double)  Output value when data is LESS THAN lower cutoff
             (unbounded, default = 0)
  [-tval]    (double)  Output value when data is GREATER THAN upper cutoff
             (unbounded, default = 1)
```

C.38. kcmplx — *Single Operand Complex Arithmetic*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Complex Arith*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

kcmplx performs complex to complex conversions on each complex point in the input data object, **Input**. The operation is specified by one of three flags, -conj for conjugation, -r2p for rectangular to polar conversion and -p2r for polar to rectangular conversion. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Segments that are not operated on
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Restrictions

The input object data type must be COMPLEX or DOUBLE COMPLEX.

Command Line Arguments

```
Usage for kcmplx: Single Operand Complex Arithmetic
% kcmplx
  -i         (infile)  input data object
  -o         (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 3 options:
    [-conj] (flag) Complex Conjugate
    [-p2r]  (flag) Convert Polar Coordinates to Rectangular Coordinates
    [-r2p]  (flag) Convert Rectangular Coordinates to Polar Coordinates
```

C.39. kcmplx2real — *Output = Real, Imaginary, Phase, or Magnitude Component of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Complex to Real*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

Complex Conversion (kcmplx2real) performs simple mappings of **complex** data to **real** form. The **complex** input object is transformed into **real** output objects via four methods: (If the input data is not complex, the imaginary component is assumed to be zero.)

Real Output

Assign the real part of the input to the output data object (or).

Imaginary Output

Assign the imaginary part of the input to the output data object (oi).

Phase Output

Assign the phase of the input to the output data object (op).

Magnitude Output

Assign the magnitude of the input to the output data object (om). The user can specify one of six methods for calculating the magnitude output, where *Log* is the logarithm, base 10.

(mag)

Magnitude = sqrt(real * real + imaginary * imaginary)

(logmag1)
Log (Magnitude + 1)

(logmag)
Log (Magnitude)

(magsq)
Magnitude Squared

(logmagsq1)
Log (Magnitude Squared + 1)

(logmagsq)
Log (Magnitude Squared)

The output object data type will be the same as the input (regarding single or double floats).

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

kcmplx, kreal2cmplx

Command Line Arguments

```
Usage for kcmplx2real: Output = Real, Imaginary, Phase, or Magnitude Component of Input
% kcmplx2real
  -i         (infile) Input data object

  Required Loose Group - Specify at least one of the 5 options:
    [-or] (outfile) Output data object containing REAL component of input
          (default = {none})
    [-oi] (outfile) Output data object containing imaginary component of input
          (default = {none})
    [-op] (outfile) Output data object containing phase of input
          (default = {none})
    [-om] (outfile) Output data object containing magnitude of input
          (default = {none})

    [Optional Mutually Exclusive Group - Specify one or none of the 6 options:]
      [-mag]       (flag) Magnitude = sqrt(real*real + imag*imag)
```

```
[-magsq]      (flag) Magnitude Squared = real*real + imag*imag
[-logmag1]    (flag) Log(Magnitude+1) = log(sqrt(real*real + imag*imag)+1)
[-logmagsq1] (flag) Log(Magnitude Squared+1) = log(real*real+imag*imag+1)
[-logmag]     (flag) Log(Magnitude) = log(sqrt(real*real + imag*imag))
[-logmagsq]   (flag) Log(Magnitude Squared) = log(real*real + imag*imag)
```

C.40. kcomment — *Change the Comment on a Data Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Object Attributes*
Operator:	*Comment Data*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Comment Data* operator, kcomment, is used to change the comment field in a Khoros data object. The comment is replaced by the given string.

If an output data object is specified, the input data object is copied to the output data object and the comment on the output data object is changed. If there was no output data object specified, then the comment is changed on the input data object. Note that the underlying data format for the image actually determines whether the comment can be stored with the data object or not. For instance, Khoros VIFF and XVIMAGE formats support the storage of a comment while the XPM format does not. Thus changing the comment on an XPM image will have essentially no effect.

A comment string can be provided (-s). Depending on the selected option, this string will replace the comment (-re), be appended to the existing comment (-ap), or be prepended to the existing comment (-pr). Yet another option allows the comment can be edited directly (-ed). If editing is desired, this routine will bring up an editor session as determined by your KEDITOR environment variable. Within this session will be the previous comment. Once you have completed editing the comment, save and quit the editor, and the edited comment will be assigned.

Examples

kcomment -i image.viff -c "This is a new comment"
will change the comment on image.viff to "This is a new comment"

See Also

kprdata

Restrictions

This routine can not be used to set the comment on a data object if the underlying data storage format does not support storage for the comment. Restrictions on read-write objects require this operator to copy the data on the input-only case.

Command Line Arguments

```
Usage for kcomment: Change the Comment on a Data Object
% kcomment
  -i         (infile)  Input Data Object

  [-s]       (string)  Comment String
             (default = {none})
  [-o]       (outfile) Resulting Output Data Object
             (default = {none})

  [Optional Mutually Exclusive Group - Specify one or none of the 4 options:]
    [-ed] (boolean) Edit Existing Comment
          (default = true)
    [-re] (flag)    Replace Comment with String
    [-pr] (flag)    Prepend String to Comment
    [-ap] (flag)    Append Comment to String
```

C.41. kcompare — *Comparison Operators*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	*Comparison Ops*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

The kcompare operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value (real), which ever is specified by the user. If the two values satisfy the condition flag passed in,

Equal (eq)
Greater Than or Equal (ge)
Greater Than (gt)
Less Than of Equal (le)
Less Than (lt)
Not Equal (ne)

the corresponding point in **Output** will be assigned the value specified by **TRUE Value** (tval). It the values being compared do not satisfy the condition, the corresponding point in the output object will be the **FALSE Value** (fval).

If a non-zero **Tolerance** (tol) is specified, the two values being compared can be within this range of one another and still satisfy the comparison. The tolerance feature is not implemented at this time.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Tolerance is not implemented

Command Line Arguments

```
Usage for kcompare: Comparison Operators
% kcompare
  -i1        (infile)  first input data object
  -o         (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) second input data object
            (default = {none})
    [-real] (double) constant value
            (real >= 0.0, default = 1)
```

```
Required Mutually Exclusive Group - Specify exactly one of the 6 options:
  [-eq] (flag) Equal: if i1 == (i2 or val), i1 = TRUE
  [-ne] (flag) Not Equal:  if i1 != (i2 or val), i1 = TRUE
  [-gt] (flag) Greater Than:  if i1 >  (i2 or val), i1 = TRUE
  [-ge] (flag) Greater Than or Equal:  if i1 >= (i2 or val), i1 = TRUE
  [-lt] (flag) Less Than:  if i1 <  (i2 or val), i1 = TRUE
  [-le] (flag) Less Than or Equal:  if i1 <= (i2 or val), i1 = TRUE

[-tol]     (double)  Tolerance
           (unbounded, default = 0)
[-tval]    (double)  Value assigned to output if evaluation is TRUE
           (unbounded, default = 1)
[-fval]    (double)  Value assigned to output if evaluation is FALSE
           (unbounded, default = 0)
```

C.42. kconj — *Output(real, imaginary) = Input(real, -imaginary)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Conjugate*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Conjugate* operator returns the complex conjugate of each data point in the input data object, **Input**. The conjugate is obtained by negating the imaginary part of the complex pair. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Conjugate* runs the program *kcmplx* with the -conj flag.

Segments that are not operated on

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

See Also

DATAMANIP::kcmplx

Command Line Arguments

```
Usage for kconj: Output(real, imaginary) = Input(real, -imaginary)
% kconj
  -i         (infile)  input data object
  -o         (outfile) resulting output data object
```

C.43. kconvert — *Convert Data Storage Type (Optional Scale & Offset)*

Object Information

Category:	*Data Manip*	Subcategory:	*Data Conversion*
Operator:	*Convert Type*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This program will convert, offset and/or scale the specified segment from the input object and create a new object containing the modified segment and all the other segments, which will be unmodified. Any of the polymorphic segments can be specified with a flag. To specify another segment you use the -segment option, and specify the segment name: "segment1".

Command Line Arguments

```
Usage for kconvert: Convert Data Storage Type  (Optional Scale & Offset)
% kconvert
  -i         (infile)  input data object
  -o         (outfile) resulting data object

  Required Mutually Exclusive Group - Specify exactly one of the 6 options:
    [-val]     (flag)   If -val flag is selected, operate on Value segment
    [-map]     (flag)   If -map flag is selected, operate on Map segment
    [-mask]    (flag)   If -mask flag is selected, operate on Mask segment
    [-loc]     (flag)   If -loc flag is selected, operate on Location segment
    [-time]    (flag)   If -time flag is selected, operate on Time segment
    [-segment] (string) Perform operation on user specified segment
               (default = {none})

  [-type]    (list)    Convert to specified data type
          0  "Propagate Input Type",
          1  "Bit",
          2  "Byte",
          3  "Unsigned Byte",
          4  "Short",
          5  "Unsigned Short",
          6  "Integer",
          7  "Unsigned Integer",
          8  "Long",
          9  "Unsigned Long",
          10  "Float",
          11  "Double",
          12  "Complex",
          13  "Double Complex"
          [default: 0 "Propagate Input Type"]
  [-scale]   (double)  Real scale factor
             (unbounded, default = 1)
  [-roff]    (double)  Real offset value
             (unbounded, default = 0)
  [-ioff]    (double)  Imaginary offset value
             (unbounded, default = 0)
```

C.44. kcos — *Compute Cosine or Arc Cosine Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Cos & ArcCos*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Cos & ArcCos* operator returns the cosine, or arc cosine, value of each data point in the input data object, **Input**. The forward or inverse trigonometric operation is specified by selecting the **Cosine** or **Arc Cosine** option on the user interface.

Executing *Cos & ArcCos* runs the program *karith1* with the -cos or -acos flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kcos: Compute Cosine or Arc Cosine Function
% kcos
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-cos]  (flag) Calculate cosine of input
```

```
[-acos] (flag) Calculate arc cosine of input
```

C.45. kcosh — *Compute Hyperbolic Cosine or Arc Cosine Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Cosh & ArcCosh*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Cosh & ArcCosh* operator returns the hyperbolic cosine, or hyperbolic arc cosine, value of each data point in the input data object, **Input**. The forward or inverse trigonometric operation is specified by selecting the **Hyperbolic Cosine** or **Hyperbolic Arc Cosine** option on the user interface.

Executing *Cosh & ArcCosh* runs the program *karith1* with the -cosh or -acosh flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kcosh: Compute Hyperbolic Cosine or Arc Cosine Function
% kcosh
  -i         (infile)  Input data object
```

```
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-cosh]  (flag) Calculate hyperbolic cosine of input
    [-acosh] (flag) Calculate hyperbolic arc cosine of input
```

C.46. kcpfromval — *Copy Value Segment Into One Segment of Output*

Object Information

Category:	*Data Manip*	Subcategory:	*Segment Operators*
Operator:	*Copy from Value*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine copies the five dimensional value segment from the source object to the specified segment specified in the destination_object. The data will be copied into the lowest order dimension of the destination segment, then the next order will be used, and the next, and so forth, until all the value data is copied. If the dimensionality of the destination is less than five, then the last dimension of the destination segment will have all the higher order dimensions rolled into it.

If segment is specified to be a polymorphic segment by setting one of the toggles (value, location, time, mask, map) then the segment dimensionalities and index orders specified by the polymorphic model will be used. If the segment is not one of the polymorphic segments, then the dimensionality of the value segment will be used, and the segment will be given an index order of width, height, depth, time, and elements.

Only the size and data type attribute will be changed, all others will not be transferred to the destination segment. If the segment does not exist in the destination and is not a polymorphic segment the dimension and index order will be set. If it is a polymorphic segment the default polymorphic dimension and index order will be used.

Restrictions

Results guaranteed only if dimension of destination segment is 5. Other dimensions are not tested and may produce poor results.

Command Line Arguments

```
Usage for kcpfromval: Copy Value Segment Into One Segment of Output
% kcpfromval
  -i         (infile)  Source input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 6 options:
    [-val]     (flag)   If -val flag is selected, operate on Value segment
    [-map]     (flag)   If -map flag is selected, operate on map segment
    [-mask]    (flag)   If -mask flag is selected, operate on Mask segment
    [-loc]     (flag)   If -loc flag is selected, operate on Location segment
```

```
    [-time]    (flag)   If -time flag is selected, operate on Time segment
    [-segment] (string) Perform operation on user specified segment
               (default = {none})
```

C.47. kcpseg — *Copy Selected Data Segments from Input*

Object Information

Category: *Data Manip* Subcategory: *Segment Operators*
Operator: *Copy Segments* Object Type: *kroutine*
In Cantata: *Yes*

Description

The program will copy the segments from one object and create a new object containing those segments. If the destination segment is specified to be "Copy All" all the segments and their associated attributes will be copied. If all segments are copied, then the global attributes will also be copied to the new object. It will not copy the object level attributes if just individual segments are copied. The segment specification can also specify multiple segments by typing a string containing the segment names separated by space(s):

"segment1 segment2 segment3"

Command Line Arguments

```
Usage for kcpseg: Copy Selected Data Segments from Input
% kcpseg
  -i         (infile)  input file
  -o         (outfile) output file

  Required Loose Group - Specify at least one of the 6 options:
    [-val]     (flag)   If -val flag is selected, operate on Value segment
    [-map]     (flag)   If -map flag is selected, operate on map segment
    [-mask]    (flag)   If -mask flag is selected, operate on Mask segment
    [-loc]     (flag)   If -loc flag is selected, operate on Location segment
    [-time]    (flag)   If -time flag is selected, operate on Time segment
    [-segment] (string) Perform operation on user specified segment
               (default = Copy All)
```

C.48. kcptoval — *Copy One Segment Into Value Segment of Output*

Object Information

Category: *Data Manip* Subcategory: *Segment Operators*
Operator: *Copy to Value* Object Type: *kroutine*
In Cantata: *Yes*

Description

This routine copies a segment from the source object and put it into the destination object's value segment. The higher dimensions will be rolled into the elements dimension of the value segment. If the dimension is less than 5, the data will be copied into the lowest order dimension of the value segment first, then the next, etc. and the higher dimensions ignored. If the value segment exists in the source object it will be replaced.

Only the size and data type attribute will be changed in the value segment, all others will not be transferred. If the value does not yet exist, then the dimension and index order of the value segment will be set to the defaults: 5 and width, height, depth, time, and elements. If it does exist the index order will not be changed.

Restrictions

Results guaranteed only if dimension of destination segment is 5. Other dimensions are not tested and may produce poor results.

Command Line Arguments

```
Usage for kcptoval: Copy One Segment Into Value Segment of Output
% kcptoval
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 6 options:
    [-val]     (flag)   If -val flag is selected, operate on Value segment
    [-map]     (flag)   If -map flag is selected, operate on map segment
    [-mask]    (flag)   If -mask flag is selected, operate on Mask segment
    [-loc]     (flag)   If -loc flag is selected, operate on Location segment
    [-time]    (flag)   If -time flag is selected, operate on Time segment
    [-segment] (string) Perform operation on user specified segment
               (default = {none})
```

C.49. kcube — *Output = Input**3*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Cube*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

Output = Input*Input*Input

The *Cube* operator returns the cube (value raised to the power of 3) of each data point in the input data object, **Input**.

Executing *Cube* runs the program *karith2* with the -pow flag and -real 3.0 -imag 0.0.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith2

Command Line Arguments

```
Usage for kcube: Output = Input**3
% kcube
  -i1        (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.50. kdeg2rad — *Output = Input * (pi/180)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Degree to Radian*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Degree to Radian* operator returns the result of multiplying each data point in the input data object, **Input** by pi/180.

Executing *Degree to Radian* runs the program *karith2*.
karith2 -mul -real 0.0174533 -imag 0.0

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith2

Restrictions

Since this pane object calls karith2 -mul, and multiplication is a legal complex operation, this operation will be performed on complex data. The complex scale factor used is (0.0174533 + i0.0).

Command Line Arguments

```
Usage for kdeg2rad: Output = Input * (pi/180)
% kdeg2rad
  -i1        (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.51. kdiv — *Output = Input 1 / (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Divide*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Divide* operator divides each point in **Input 1** by either **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Divide* runs the program *karith2* with the -div flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (1.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2, DATAMANIP::kdivinto

Command Line Arguments

```
Usage for kdiv: Output = Input 1 / (Input 2 or Constant)
% kdiv
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) Second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 1)
```

```
[-imag] (double) Imaginary constant value
        (unbounded, default = 0)
```

C.52. kdivinto — *Output = (Input 2 or Constant) / Input 1*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Divide Into*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Divide Into* operator divides either the **Constant** value(s), or each point in **Input 2**, which ever is specified by the user, by **Input 1**.

Executing *Divide* runs the program *karith2* with the -divinto flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (1.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2, DATAMANIP::kdiv

Command Line Arguments

```
Usage for kdivinto: Output = (Input 2 or Constant) / Input 1
% kdivinto
  -i1        (infile)  First Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) Second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 0)
      [-imag] (double) Imaginary constant value
              (unbounded, default = 0)
```

C.53. keq — *IF Input 1 is Equal to (Input 2 or Constant), Output = TRUE*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	==	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The EQUAL comparison operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value, which ever is specified by the user. If the two values are equal, the corresponding point in **Output** will be assigned the value specified by **TRUE Value**. It the values being compared are not equal, the corresponding point in **Output** will be the value specified by **FALSE Value**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects
Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing the equal operator runs the program *kcompare* with the -eq flag.

See Also

DATAMANIP::kcompare

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Command Line Arguments

```
Usage for keq: IF Input 1 is Equal to (Input 2 or Constant), Output = TRUE
% keq
  -i1         (infile)  First input data object
  -o          (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real] (double) Constant value
```

```
            (real >= 0.0, default = 0)

  [-tval]    (double)  Value assigned to output if evaluation is TRUE
             (unbounded, default = 1)
  [-fval]    (double)  Value assigned to output if evaluation is FALSE
             (unbounded, default = 0)
```

C.54. kerf — *Compute Error & Complement Error Functions*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*Error Function*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Error Function* operator returns the error or complement error function of each data point in the input data object, **Input**. The error or complement error function is specified by selecting the **Error Function** or **Complement Error Function** option on the user interface.

Error Function: erf(x)=2/sqrt(pi)*integral_from_0

Complement Error Function: 1.0-erf(x)

Executing *Error* or *Complement Error* runs the program *karith1* with the -erf or -erfc flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kerf: Compute Error & Complement Error Functions
% kerf
  -i          (infile)  Input data object
  -o          (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-erf]  (flag) Output is error function of input
    [-erfc] (flag) Output is the complement error function of input
```

C.55. kexp — *Compute Exponential (antilog)*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*Exponential*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Exponential* operator, kexp, computes the exponential of an object element by element. If the object has an associated mask, the operation will be gated, so that the exponential is computed only on valid elements, indicated by non-zero mask values. A zero mask value will cause the value of the corresponding element in the input object to be transferred the output object.

The exponential of the base value is the base raised to the value of the object element. For example, b10(x) = 10**x. Three standard-base exponentials are provided, and alternate bases can be specified using the -base option. *be* returns the natural exponential of x (e**x), and *b2* and *b10* return 2**x and 10**x respectively.

If the -offset option is set to TRUE, the value returned will be the result of the exponential operation minus 1 (exp(result) - 1). SPECIAL NOTE: The exponential() has machine-specific characteristics! To fix this, an error message is returned if the user specified base is less than or equal to zero.

See Also

DATAMANIP::klogexp

Command Line Arguments

```
Usage for kexp: Compute Exponential (antilog)
% kexp
  -i          (infile)  input data object
  -o          (outfile) resulting output data object
```

```
Required Mutually Exclusive Group - Specify exactly one of the 4 options:
  [-be]   (flag)   Compute exponential using natural base 2.71828...
  [-b10]  (flag)   Compute exponential using base 10
  [-b2]   (flag)   Compute exponential using base 2
  [-base] (double) User Specified base
          (unbounded, default = 2)

[-offset]  (boolean) Return either result [False] or result-1 [True]
           (default = true)
```

C.56. kexpand — *Expand Object Via Pixel Replication*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*Expand*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Expand Object* operator (kexpand) increases the size of the Input data object (i) using pixel replication. Magnification factors for each dimension, Width (wmag), Height (hmag) Depth (dmag), Time (tmag), and Elements (emag) can be specified. Magnification factors greater than one will increase the object size. (Factor values less than 1 but greater than zero are also valid, and will reduce the size of the object.) The resultant output object size will be the original size multiplied by the magnification factor for that dimension, and rounded.

Pixel replication is done on a floating point indexing basis, where the data point index is obtained by truncating the cumulative floating point index. Therefore, when the calculated replication factor is non-integer, the resulting integer step size varies during the process.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input` If there is map and value data in the source object, the map is transferred to the destination object.

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

If explicit location or time data are available, the sampling algorithm can not resample these data yet, and will fail. This will be implemented later ....

Executing *Expand Object* runs the program *kresample* with -resize set to true and -inter set to 0.

See Also

DATAMANIP::kresample, DATAMANIP::kshrink

Command Line Arguments

```
Usage for kexpand: Expand Object Via Pixel Replication
% kexpand
  -i         (infile)  Input object
  -o         (outfile) Resulting subsampled object

  [-wmag]    (double)  Magnification factor in width dimension
             (wmag > 0.0, default = 2)
  [-hmag]    (double)  Magnification factor in height dimension
             (hmag > 0.0, default = 2)
  [-dmag]    (double)  Magnification factor in depth dimension
             (dmag > 0.0, default = 1)
  [-tmag]    (double)  Magnification factor in time dimension
             (tmag > 0.0, default = 1)
  [-emag]    (double)  Magnification factor in elements dimension
             (emag > 0.0, default = 1)
```

C.57. kexpon — *Introduce Exponential Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Exponential Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kexpon introduces exponential noise of the specified parameters in the input data object (-i). The noise can either replace the existing value data or it can be added to the existing value data by specifying the appropriate flag (-replace or -add). The variance (-evar) of the exponential distribution must be specified.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::knoise

Command Line Arguments

```
Usage for kexpon: Introduce Exponential Noise in Input Object
% kexpon
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-replace] (flag) Replace existing values with noise
    [-add]     (flag) Add noise to the specified segment

  [-evar]    (double)  Variance for exponential noise
             (evar > 0.0, default = 1)
```

C.58. kextract — *Extract Rectangular Region from Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*Extract*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Extract Region* operator (kextract) extracts a region of data from the source object *Input* (i) and places it in the destination object *Output* (o). The region size is specified by the Width (wsize), Height (hsize), Depth (dsize), Time (tsize), and Elements (esize) Region Size parameters. The region will be extracted starting at the point defined by the Region Origin parameters, Width (woff), Height (hoff), Depth (doff), Time (toff), and Elements (eoff). The datum at this starting point will be placed at the origin (0,0,0,0,0) in the destination object.

The output data object size for each dimension is determined first by the Region Size parameters *if* they are specified, and then by the Region Origin parameters in combination with the original source data size. In other words, if the size for a dimension is not selected on the graphical user interface (or explicitly specified when running kextract from the command line), all data on that dimension will be extracted, starting at the specified origin. This will ensure the following behavior. Given an input data file with dimensions (width=100, height=100, depth=10, time=1, elements = 1), and the following *specified* parameters:

Region Size: Width = 50, Height = 50

Region Origin: Width = 2, Height = 2, Depth = 2

the output object size will be (width=50, height=50, depth=8, time=1, elements = 1).

The extract operation is based on implicit indexing. This means that if location or time data exist, the extract operation is not done in terms an interpretation of the location/time data values, but in terms of the implicit indexing of these data (which is specified by the width, height, depth, time, elements indices of the polymorphic data model). This program can operate on any combination of value, mask, location and/or time data in the input object. Map data is unaffected by the extract operation.

If the *Region Origin* and/or the *Region Size* fall outside of the size of the input object, destination object value, location, and time data will be padded with zero, and mask data will be padded with one.

The subobject position attribute of a data object defines the coordinates at which that subobject is located in its parent data object. The "Record Region Origin" option (subpos) allows the extract operator to set the sub-object position attribute in the output object to the Region origin. If this option is set to "Yes", the subobject position of the output object will be set to (woff, hoff, doff, toff, eoff). If "Record Region Origin" is set to "No", the output subobject position is set to (0,0,0,0,0). Programs like the insert operator (kinsert) can use this attribute for automatically inserting the subregion back into the original after processing.

The subobject position attribute can also be explicitly set by using the Set Data Attributes operator (ksetdattr), and can be printed using the Print File Information operator (kfileinfo).

The destination object data types are the same as the source object data types.

Restrictions

This program has not yet been updated to completely support the polymorphic data model.

Command Line Arguments

```
Usage for kextract: Extract Rectangular Region from Object
% kextract
  -i         (infile)  Input object
  -o         (outfile) Resulting output region

  [-wsize]   (integer) Specifies the region width
             (wsize > 0, default = 128)
  [-hsize]   (integer) Specifies the region height
             (hsize > 0, default = 128)
  [-dsize]   (integer) Specifies the region depth
             (dsize > 0, default = 1)
  [-tsize]   (integer) Specifies the region size on time dimension
             (tsize > 0, default = 1)
  [-esize]   (integer) Specifies the region size on elements dimension
             (esize > 0, default = 1)
  [-woff]    (integer) Specifies origin of region on width dimension
             (unbounded, default = 0)
  [-hoff]    (integer) Specifies origin of region on height dimension
             (unbounded, default = 0)
  [-doff]    (integer) Specifies origin of region on depth dimension
             (unbounded, default = 0)
  [-toff]    (integer) Specifies origin of region on time dimension
             (unbounded, default = 0)
  [-eoff]    (integer) Specifies origin of region on elements dimension
             (unbounded, default = 0)
  [-subpos]  (boolean) Record region origin (Subobject Position) in output
             (default = false)
```

C.59. kfft — *Fast Fourier Transform (Forward and Inverse)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Linear Transforms*
Operator:	*FFT*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kfft performs a multidimensional FFT on the input object specified by **Input File**. The dimensions along which the FFT is taken are specified using the -w,-h,-d,-t and -e flags. A 1D FFT is evaluated along each of the specified dimensions. For example, if the -w and -h flags are set for an image then the FFT is first taken along each width of the image and it is taken again for each height of the image resulting from the first FFT. This is equivalent to taking the 2D FFT of the image. Thus *kfft* can perform up to a 5D FFT on the input object depending on the number of flags set by the user. The size of each dimension specified must be a power of 2 and greater than 1. The dimensions for which the FFT is not calculated can have any size. The FFT is evaluated for all the primitives of the specified dimension. For example, if the input object has size (w,h,d,t,e) and the -w flag is specified then the FFT is calculated for all h*d*t*e width primitives in the object.

This program will compute either a forward or inverse FFT, depending on the direction requested with the -dir option. A -dir value of 0 or FALSE will result in a forward FFT, and a -dir value of 1 or TRUE will result in an inverse FFT. The default is a -dir value of 0 or FALSE (forward FFT).

Input File

The input object is assumed of type complex. However if the data type of the object is not complex, then it is assumed that this object represents only the real part of the data. The imaginary part of the complex pair is assumed to be zero.

Centering

The program also has the option (-center) of shifting the FFT result so that the DC component is at (W/2,H/2,D/2,T/2,E/2) instead of (0,0,0,0,0) for the 5-D case. This option is supplied because the center-shifted frequency domain representation is much easier to visualize and filter, particularly for image processing applications. For a forward FFT, the center-shifted option multiplies the input data by (-1)**(w+h+d+t+e) where (w,h,d,t,e) is the coordinate. For the inverse FFT case, the data is multiplied by (-1)**(w+h+d+t+e) AFTER the FFT processing, accounting for the fact that the input frequency domain data was center-shifted by the forward FFT.

Note that you must keep track of how the transform was centered in the forward FFT in order to know how to de-center it during the inverse FFT. For example, if the forward FFT was centered, then you must also ask that the inverse FFT be centered or you'll end up with your output data being multiplied by an alternating +1 / -1 sequence. Likewise, a non-centered forward FFT must be followed by a non-centered inverse FFT or you will end up with data multiplied by an alternating +1 / -1 sequence. Look out for the possibility of hiding the +1 / -1 mutliplication due to the use of the magnitude operator to inspect the results!

Output Scaling

The scaling factor can also be specified. The user can choose from no (unity) scaling (-scale 1), scaling by a factor of 1/N (-scale 2) per transform axis, or scaling by a factor of 1/sqrt(N) (-scale 3) per transform axis, where N is the dimension of the axis along which the FFT is being taken.

It is the responsibility of the user to make sure that the data going through multiple FFT's remains properly scaled. Since scaling can be done on both the forward and inverse FFTs, the aggregate scale factor is the product of the scaling done during the forward and inverse processing steps. Fox example, if the forward FFT is along a single axis of length N and scaling of 1/sqrt(N) is selected, and the reverse FFT is along the same axis again with scaling of 1/sqrt(N), then the aggregate scale factor would be (1/sqrt(n))*(1/sqrt(n)) or 1/N, which is the correct scaling for a combined set of forward and inverse operations.

In general, the aggregate scale factor after forward and inverse processing should be arranged to be 1/Ni per axis where Ni is the size of the i'th axis along which the transforn is being taken. This means that for an NxM 2D data set, the combination of forward and inverse FFT scale factors should be arranged to be 1/(M*N). This can be done three ways; Unity forward and 1/N inverse, or 1/N forward and unity inverse, or 1/sqrt(n) forward and 1/sqrt(n) inverse. s

Output File

The output of the FFT is a set of complex numbers. The data type representation can be selected as single precision complex (COMPLEX) or double precision complex (DCOMPLEX), both of which are floating point formats.

Restrictions

kfft cannot operate on objects that have either of the mask, location or time segments in addition to the value segment.

The IEEE FFT used internally is limited to a maximum vector size of 32768 data points. This means that no object dimension can be larger than this number. This restriction will be removed in a later release when the IEEE FFT is replaced. This FFT is currently used because it is trusted and fast, although that fact may be hidden by data services overhead.

Command Line Arguments

```
Usage for kfft: Fast Fourier Transform (Forward and Inverse)
% kfft
  -i         (infile)  complex input object
  -o         (outfile) complex output object

  Required Loose Group - Specify at least one of the 5 options:
    [-w] (flag) if set, FFT is taken along the width dimension
    [-h] (flag) if set, FFT is taken along the height dimension
    [-d] (flag) if set, FFT is taken along the depth dimension
    [-t] (flag) if set, FFT is taken along the time dimension
    [-e] (flag) if set, FFT is taken along the element dimension

  [-dir]     (boolean) FFT direction (0 is forward, 1 is inverse)
             (default = false)
  [-center]  (boolean) select to center (1) or not to center (0) the transform
             (default = true)
  [-scale]   (flag)    scale FFT by 1, 1/N or 1/sqrt(N)
          1  (Scale the output data by unity),
```

```
            2  (Scale output data by 1/N per transform axis),
            or 3  (Scale output data by 1/sqrt(N) per transform axis)
     [default: 3]
       [-type]    (cycle)   Output object data type
            12  "Complex",
            13  "Double Complex"
            [default: 12 "Complex"]
```

C.60. kfileinfo — *Print File Information*

Object Information

Category:	*Input/Output*	Subcategory:	*Information*
Operator:	*Data Object Info*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kfileinfo displays information about a file. The information about the file is printed on the screen. An output file can optionally be specified for printing the information.

Executing *File Info* runs the program *kprdata* with the -attr flag.

See Also

DATAMANIP::kprdata

Command Line Arguments

```
Usage for kfileinfo: Print File Information
% kfileinfo
  -i         (infile)  input data object

  [-f]       (outfile) resulting data object
             (default = {none})
```

C.61. kflip — *Reflect Data Along Specified Directions*

Object Information

Category:	*Data Manip*	Subcategory:	*Reorganize Data*
Operator:	*Flip*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kflip performs the reflection of data values and the validity mask. The directions supported by kflip are: 1) width, 2) height, 3) depth, 4) time, and 5) elements. The user is able to select one, all directions, or any combination for reflecting the data in an object.

kflip will fail if the source object has explicit time or location data because we have not defined the algorithm for flipping.

If the source object contains MAP data it will be directly transferred to the destination object.

Examples

% kflip -i source_obj -o dest_obj -w
performs reflection of internal parameters in the width direction.

% kflip -i source_.obj -o dest_obj -w -h -d -t -e
performs reflection of parameters in all directions.

Restrictions

kflip performs the reflection of parameters in the VALUE and MASK data segments. If LOCATION and TIME segments exist, the reflection is performed in the VALUE and MASK segments at this time. LOCATION and TIME are transferred directly.

References

1) FUNDAMENTALS OF IMAGE PROCESSING by A. K. Jain, Prentice Hall, 1989.
2) DIGITAL IMAGE PROCESSING by Rafael C. Gonzalez and Richard E. Woods, Addison Wesley, 1992.
3) DIGITAL IMAGE PROCESSING, 2nd. Ed., by William K. Pratt, Wiley, 1991.
4) MULTIDIMENSIONAL DIGITAL SIGNAL PROCESSING, by Dan E. Dudgeon and Russell M. Mersereau, Prentice Hall, 1984.
5) DISCRETE-TIME SIGNAL PROCESSING, by Alan V. Oppenheim and Ronald W. Schafer, Prentice Hall, 1989.
6) TWO-DIMENSIONAL SIGNAL AND IMAGE PROCESSING, by Jae S. Lim, Prentice Hall, 1990.

Command Line Arguments

```
Usage for kflip: Reflect Data Along Specified Directions
% kflip
  -i         (infile)  Source (input) object
  -o         (outfile) Destination (output) object


  [Optional Loose Group - Specify at least one of the 5 options:]
    [-w] (flag) reflection in the WIDTH direction
    [-h] (flag) reflection in the HEIGHT direction
    [-d] (flag) reflection in the DEPTH direction
    [-t] (flag) reflection in the TIME direction
    [-e] (flag) reflection in the ELEMENTS direction
```

C.62. kfloor — *Output = Integer Floor of Input*

Object Information

Category: *Arithmetic* Subcategory: *Single Operand Arithmetic*

Operator:	*Floor*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Floor* operator returns the integer floor of each data point in the input data object, **Input**.

Executing *Floor* runs the program *karith1* with the -floor flag.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kfloor: Output = Integer Floor of Input
% kfloor
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.63. kformats — *Conversion Between the Formats Supported in Khoros 2*

Object Information

Category:	*Input/Output*	Subcategory:	*Export Data*
Operator:	*Supported Formats*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kformats converts between formats internally supported by Khoros 2.0. Currently, these formats include

ASCII Format

AVS - Application Visualization System (AVS) Format

PNM - Portable Any Map

The PBM image file format describes the format used by the pbm conversion routines. The format was developed by Jef Poskanzer. The PBM format is really three different formats; PBM, PGM, & PPM. The formats correspond to bitmap, greymap, and pixmap formats.

Raw (headerless) Format

Sun Raster - Sun Rasterfile File Format

The Sun Raster file format describes the format used by the Sun Rasterfile File Format conversion routines. The format was developed by Sun Microsystems.

VIFF - Khoros 2.0 Internal Visualization and Image File Format

XBM - X Bitmap Format

Xvimage - Khoros Visualization/Image File Format (Khoros 1.0)

A Khoros Xvimage data file is organized as a 1Kbyte header followed by additional information. The first two bytes of the header tell what kind of stuff the additional information is. For Khoros image files, the additional information consists of the maps, the location data, and then the image or vector data.

XWD - X Window Dump Format

XPM - X Pixmap Format

User Specified Supported Format

This string entry allows the user to enter a supported format that is not available in the list of formats described above. The string is passed directly into the data services call that sets the format attribute on the data object, so it must be accurate.

Importing Other Formats

We have not forgotten about other formats, such as TGA, FITS, and TIFF; they are on the list of bizillions of things to add to Khoros. Khoros 2 is designed to be portable, therefore we made the decision that each format that we bring on line should be implemented as a portable solution. (Otherwise we end up with a software management nightmare.) As a workaround until these other formats come on line, we recommend using an intermediate converter. For example, the TIFF workaround would be to get the PBMPLUS package, which has a TIFF reader in it. Use tiff2pbm to convert your TIFF to a PBM image, which can then be read directly without further conversion by any Khoros 2 program.

Postscript Output

For postscript output, use the ipostscr operator in the Image toolbox. The object is located under "Input/Output - Hardcopy Output - Postscript" in cantata.

See Also

IMAGE::ipostscr(1)

Restrictions

If the input data cannot be represented on the output format, kformats will- Conversion Between Khoros 2.0 Supported Formats fail.

Command Line Arguments

```
Usage for kformats: Conversion Between the Formats Supported in Khoros 2
% kformats
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 12 options:
    [-ascii]   (flag)   Output in ASCII format
    [-viff]    (flag)   Output as VIFF (Khoros 2.0 Format)
    [-avs]     (flag)   Output in Application Visualization System (AVS) format
    [-xbm]     (flag)   Output in X Bitmap (XBM) format
    [-arf]     (flag)   Output in ARF Format
    [-xvimage] (flag)   Output as Xvimage (Khoros 1.0 Format)
    [-pnm]     (flag)   Output in Portable Any Map (PNM) format
    [-xwd]     (flag)   Output in X Window Dump (XWD) format
    [-raw]     (flag)   Output in Raw (headerless) format
    [-xpm]     (flag)   Output in X Pixmap (XPM) format
    [-rast]    (flag)   Output in Sun Raster format
    [-format]  (string) Supported format not available via the format flags
               (default = {none})
```

C.64. kfraction — *Output = Fractional Part of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Fractional Part*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Fractional Part* operator returns the fractional part of each data point in the input data object, **Input**.

Executing *Fractional Part* runs the program *karith1* with the -fract flag.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also
DATAMANIP::karith1

Restrictions
Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kfraction: Output = Fractional Part of Input
% kfraction
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.65. kgamma — *Compute Log Gamma Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*Log Gamma*	Object Type:	*pane*
In Cantata:	*Yes*		

Description
The *Gamma Function* operator returns the natural log of the gamma function (lgamma) for each data point in the input data object, **Input**.

Executing *Gamma Function* runs the program *karith1* with the -lgamma flag.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kgamma: Compute Log Gamma Function
% kgamma
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.66. kgate — *Output Assigned Value in Input 1 or Input 2 Based on Gating Element*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Gate Data*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

For each point in Input 1 (i1) and Input 2 (i2), *kgate* assigns to the Output (o) either the value from Input 1 or Input 2, depending on the corresponding value in the Gating Input (igate), the specified Gate Value (gval), and the gating logic (logic).

kgate can be used to gate operations. For example, to gate the addition of a constant to an input object, first add the constant to the entire data object. Use the output of the addition operation as Input 1 to *kgate*, and the original object as Input 2. Using the default logic (TRUE), the output will be the result of the addition except where the gating condition is met. In the same manner, operation gating can be achieved for multiple input operators.

Gating Logic TRUE

If the Gating Logic is set to TRUE, Input 1 is predominant, and is only gated when the gating condition is met. Therefore the Output will be assigned values from Input 1 *unless* the value in the Gating Input equals the specified Gate Value, in which case Input 1 is gated and the Output is assigned the value from Input 2.

Gating Logic FALSE

If the Gating Logic is set to FALSE, Input 2 is predominant, and is only gated when the gating condition is met. Therefore the Output will be assigned values from Input 2 *unless* the value in the Gating Input equals the specified Gate Value, in which case Input 2 is gated and the Output is assigned the value from Input 1.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask

If there is a validity mask associated with either source object, the resulting destination object mask value depends on the gating object. The mask of the first source object is used where the gate is equal to zero and the mask of the second source object is used where the gate is non-zero. If only one source object has a mask, then the other source object's mask is set to the default mask value = 1.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

If the gating source object's value data is a different size than the calculated destination object's value data size, then it is truncated if larger or padded with zeros if smaller.

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

Command Line Arguments

```
Usage for kgate: Output Assigned Value in Input 1 or Input 2 Based on Gating Element
% kgate
  -i1        (infile)  First input data object
  -i2        (infile)  Second input file object
  -igate     (infile)  Gating input data object
  -o         (outfile) Resulting output data object

  [-logic]   (boolean) Gating logic: if gate data = gval, (0) use i1, (1) use i2
             (default = true)
  [-gval]    (double)  Value on which to gate
```

```
(unbounded, default = 0)
```

C.67. kgauss — *Introduce Gaussian Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Gaussian Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kgauss introduces Gaussian noise of the specified parameters in the input data object (-i). The noise can either replace the existing value data or it can be added to the existing value data by specifying the appropriate flag (-replace or -add). The mean (-gmean) and the variance (-gvar) of the Gaussian distribution must be specified.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::knoise, DATAMANIP::kpoisson, DATAMANIP::krayleigh, DATAMANIP::kexpon, DATAMANIP::kuniform

Command Line Arguments

```
Usage for kgauss: Introduce Gaussian Noise in Input Object
% kgauss
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-replace] (flag) Replace existing values with noise
    [-add]     (flag) Add noise to the specified segment
```

```
  [-gmean]    (double)  Mean for Gaussian noise
              (unbounded, default = 0)
  [-gvar]     (double)  Variance for Gaussian noise
              (gvar > 0.0, default = 1)
```

C.68. kgconst — *Generate Object Containing Constant Value Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Data*
Operator:	*Constant*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kgconst creates a data object with a constant valued **value segment** of dimension Width * Height * Depth * Time * Elements. The data type of the value segment can be **bit, byte, unsigned byte, short, unsigned short, integer, unsigned integer, long, unsigned long, float, double, complex** and **double complex**. The real, and imaginary constant levels are specified using the level parameters (real, imag).

Examples

kgconst -o const -wsize 256 -hsize 256 -esize 3 -type integer -real 10

Creates a data object called const of size width = 256, height = 256 where each vector contains 3 elements. The data storage type is integer, and the constant is 10.

Command Line Arguments

```
Usage for kgconst: Generate Object Containing Constant Value Data
% kgconst
  -o          (outfile) Output file containing constant value data

  [-wsize]    (integer) Width size of constant value data
              (wsize > 0, default = 1)
  [-hsize]    (integer) Height size of constant value data
              (hsize > 0, default = 1)
  [-dsize]    (integer) Depth size of constant value data
              (dsize > 0, default = 1)
  [-tsize]    (integer) Time size of constant value data
              (tsize > 0, default = 1)
  [-esize]    (integer) Element dimension size of constant value data
              (esize > 0, default = 1)
  [-real]     (double)  Real constant level
              (unbounded, default = 0)
  [-imag]     (double)  Imaginary constant level
              (unbounded, default = 0)
  [-type]     (list)    Data type of constant value data
            1  "bit",
            2  "byte",
            3  "unsigned byte",
            4  "short",
```

```
5  "unsigned short",
6  "integer",
7  "unsigned integer",
8  "long",
9  "unsigned long",
10  "float",
11  "double",
12  "complex",
13  "double complex"
[default: 10 "float"]
```

C.69. kge — *IF Input1 is Greater or Equal to (Input2 or Constant), Output = TRUE*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	>=	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The GREATER or EQUAL comparison operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value, which ever is specified by the user. If the value from **Input 1** is greater than or equal to the value from **Input 2** or the **Constant**, the corresponding point in **Output** will be assigned the value specified by **TRUE Value**. If the value from **Input 1** is not greater than or equal to the value against which it is being compared, the corresponding point in **Output** will be the value specified by **FALSE Value**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing the Greater Than or Equal operator runs the program *kcompare* with the -ge flag.

See Also

DATAMANIP::kcompare

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Command Line Arguments

```
Usage for kge: IF Input1 is Greater or Equal to (Input2 or Constant), Output = TRUE
% kge
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real]  (double) Constant value
             (real >= 0.0, default = 0)

  [-tval]     (double)  Value assigned to output if evaluation is TRUE
              (unbounded, default = 1)
  [-fval]     (double)  Value assigned to output if evaluation is FALSE
              (unbounded, default = 0)
```

C.70. kgeneric — *Generic Routine Interface (glue)*

Object Information

Category: *Program Utilities* Subcategory: *Command Icons*

Operator: *Generic Interface* Object Type: *kroutine*
In Cantata: *Yes*

Description

kgeneric is a programmable interface that allows a generalized Un*x command line to be executed by cantata.

Entries on the pane are spit out to a string that is then system()'d off. The string is built up by appending the contents of the SELECTED entries together in scan-line order.

Note: All entries are strings. Unfortunately, the forms don't seem to like multiple-worded strings, so you have to build up your command piece at a time. You CAN use a multiple-worded string in the prefix and trailer.

Note: Anything entered into a FILENAME box will be treated as a file using the normal Khoros file access checking. This means the UI will try to open the filename for access. If it is a Un*x command or other object then things won't work.

Note: There appears to be no way to protect special characters from the shell.

Note: The oval button to the left of argument name selection allows you to put a minus sign ("-") immediately in front of the name itself in the command string that is executed. The parser won't let you put a minus sign in the string field itself - it will complain about unknown arguments. So if you want an argument name that looks like

-blah

then click the button to get the - to show and enter blah as the name. The actual command string executed will look like that above.

Note: There is no job control from this routine. Hitting the kill button in cantata is not guaranteed to kill the command that was system()'d off. Additionally, status reporting is unsupported.

Examples

To execute:

ls -al > snarf

using this routine, do:

% kgeneric -prefix ls -iname1 al -iname2 > -o1 snarf -di1 1 -di2 0 -di3 0 -do1 0 -do2 0 -do3 0 -dn1 0 -dn2 0 -dn3 0 -dn4 0 -dn5 0 -dn6 0 -dn7 0 -dn8 0

and you should get the desired result. Note that this routine is not much good for use on the command line. It is for cantata.

See Also

None. Yet.

Restrictions

Kgeneric executes the command via a ksystem() call, and thus there is no reporting of errors from the command. If nothing happens when you run a command, try executing the whole command from command line. If the command line fails, fix the problem before trying to use kgeneric.

No data type or range checking is available when using this interface.

References

None.

Command Line Arguments

```
Usage for kgeneric: Generic Routine Interface (glue)
% kgeneric
  -di1       (boolean) selects minus sign (TRUE) or blank (FALSE) for input name #1
  -di2       (boolean) selects minus sign (TRUE) or blank (FALSE) for input name #2
  -di3       (boolean) selects minus sign (TRUE) or blank (FALSE) for input name #3
  -do1       (boolean) selects minus sign (TRUE) or blank (FALSE) for output name #1
  -do2       (boolean) selects minus sign (TRUE) or blank (FALSE) for output name #2
  -do3       (boolean) selects minus sign (TRUE) or blank (FALSE) for output name #3
  -dn1       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #1
  -dn2       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #2
  -dn3       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #3
  -dn4       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #4
  -dn5       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #5
  -dn6       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #6
  -dn7       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #7
  -dn8       (boolean) selects minus sign (TRUE) or blank (FALSE) for arg name #8

  [-prefix]  (string)  prefix string
             (default = {none})
  [-i]       (infile)  input image
             (default = {none})
  [-o]       (outfile) output image or file
             (default = {none})
  [-iname1]  (string)  name of input argument #1
             (default = {none})
  [-i1]      (infile)  input filename #1
             (default = {none})
  [-iname2]  (string)  name of input argument #2
             (default = {none})
  [-i2]      (infile)  input filename #2
             (default = {none})
  [-iname3]  (string)  name of input argument #3
             (default = {none})
  [-i3]      (infile)  input filename #3
             (default = {none})
  [-oname1]  (string)  name of output argument #1
             (default = {none})
  [-o1]      (outfile) output filename #1
             (default = {none})
  [-oname2]  (string)  name of output argument #2
             (default = {none})
```

```
[-o2]      (outfile) output filename #2
           (default = {none})
[-oname3]  (string)  name of output argument #3
           (default = {none})
[-o3]      (outfile) output filename #3
           (default = {none})
[-name1]   (string)  Name of argument #1
           (default = {none})
[-val1]    (string)  argument #1 value
           (default = {none})
[-name2]   (string)  Name of argument #2
           (default = {none})
[-val2]    (string)  argument #2 value
           (default = {none})
[-name3]   (string)  Name of argument #3
           (default = {none})
[-val3]    (string)  argument #3 value
           (default = {none})
[-name4]   (string)  Name of argument #4
           (default = {none})
[-val4]    (string)  argument #4 value
           (default = {none})
[-name5]   (string)  Name of argument #5
           (default = {none})
[-val5]    (string)  argument #5 value
           (default = {none})
[-name6]   (string)  Name of argument #6
           (default = {none})
[-val6]    (string)  argument #6 value
           (default = {none})
[-name7]   (string)  Name of argument #7
           (default = {none})
[-val7]    (string)  argument #7 value
           (default = {none})
[-name8]   (string)  Name of argument #8
           (default = {none})
[-val8]    (string)  argument #8 value
           (default = {none})
[-trailer] (string)  trailer string
           (default = {none})
```

C.71. kgenloc — *Generate Location Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Data*
Operator:	*Generate Location*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kgenloc creates explicit location data in the specified destination object (-o). If an input source object is provided (-i) then any data segments in that source object will be copied to the destination object.

The explicit location data is specified by three sets of begin and end values (-xbegin, -xend), (-ybegin, -yend), (-zbegin, -zend) which together compose explicit corner markers for volume in (x,y,z) three space. The x-axis corresponds uniformly to the implicit width dimension on the polymorphic data model, with the y-axis and z-axis corresponding to implicit height, and depth dimensions, respectively.

Data is generated uniformly from those specified corner markers, however the explicitness of the generated data can be specified by the grid (-grid) option. A KUNIFORM grid will dictate that the generated data consist simply of setting the uniform corner markers on the output data set. A KRECTILINEAR grid will dictate that the generated data consist of three arrays of locations, an x-array, a y-array, and a z-array, with the data within those arrays changing uniformly over the specified (x,y,z) begin and end values. A KCURVILINEAR grid will dictate that the generated data consist of a four dimensional array of location data. Essentially, this data set is a three dimensional array of (x,y,z) points.

The size and datatype of the resulting location data segment can be specified. The size of the location segment is controlled with three parameters (-wsize, -hsize, -dsize) which correspond to the implicit width, height, and depth of the location segment. The dimension size of the destination location segment will always be three corresponding to the size of the location points (x,y,z). The data type of the location segment is controlled with a single parameter (-type).

Any size specifications will be overwritten by existing size information in the input source object. If the input source object contains any explicit location data, the size and data type of the existing location segment will be used to determine the size and data type of the destination location segment. Any explicit location data in the source object will be destroyed in favor of the newly generated data. Additionally, if the input source object contains no explicit location segment but does contain a value segment or a mask segment, the width, height, and depth size of either of those segments will be used to set the width, height, and depth size of the destination location segment.

See Also

ksetdattr

Restrictions

The data is generated and stored as one block, so generating large location data sets are subject to the memory limitations of your machine.

Command Line Arguments

```
Usage for kgenloc: Generate Location Data
% kgenloc
  -o         (outfile) Resulting output data object

  [-i]       (infile)  First Input data object
             (default = {none})
  [-xbegin]  (double)  Explicit Location Begin for X dimension
             (unbounded, default = 0)
  [-xend]    (double)  Explicit Location End for X dimension
             (unbounded, default = 1)
  [-ybegin]  (double)  Explicit Location Begin for Y dimension
             (unbounded, default = 0)
  [-yend]    (double)  Explicit Location End for Y dimension
             (unbounded, default = 1)
  [-zbegin]  (double)  Explicit Location Begin for Z dimension
```

```
                (unbounded, default = 0)
     [-zend]    (double)  Explicit Location End for Z dimension
                (unbounded, default = 1)
     [-grid]    (flag)    Grid Type to use when creating Location Data
             1  (create uniform location data),
             2  (create rectilinear location data),
             or 3  (create curvilinear location data)
   [default: 1]
     [-wsize]   (integer) Width of location data
                (wsize > 0, default = 512)
     [-hsize]   (integer) Height of location data
                (hsize > 0, default = 512)
     [-dsize]   (integer) Depth of location data
                (dsize > 0, default = 1)
     [-type]    (list)    Data type of location data
             1  "bit",
             2  "byte",
             3  "unsigned byte",
             4  "short",
             5  "unsigned short",
             6  "integer",
             7  "unsigned integer",
             8  "long",
             9  "unsigned long",
             10  "float",
             11  "double",
             12  "complex",
             13  "double complex"
             [default: 10 "float"]
```

C.72. kgexpon — *Generate Exponential Noise Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Noise*
Operator:	*Exponential Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kgexpon generates exponential noise of the specified parameters in the output data object (-o). The variance (-evar) must be specified for the exponential distribution to be generated.

The size of all five dimensions of the object can be specified. The data type of the output object can also be specified. *kgexpon* can also generate complex valued exponential noise data.

See Also

DATAMANIP::kgnoise, DATAMANIP::kggauss, DATAMANIP::kgpoisson, DATAMANIP::kgrayleigh, DATAMANIP::kguniform

Command Line Arguments

```
Usage for kgexpon: Generate Exponential Noise Data
% kgexpon
  -o         (outfile) Output file containing noise data

  [-wsize]   (integer) Width size of noise value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of noise value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of noise value data
             (dsize > 0, default = 1)
  [-tsize]   (integer) Time size of noise value data
             (tsize > 0, default = 1)
  [-esize]   (integer) Element dimension size of noise data
             (esize > 0, default = 1)
  [-evar]    (double)  Variance for exponential noise
             (evar > 0.0, default = 1)
  [-type]    (list)    Data type of noise value data
          1  "bit",
          2  "byte",
          3  "unsigned byte",
          4  "short",
          5  "unsigned short",
          6  "integer",
          7  "unsigned integer",
          8  "long",
          9  "unsigned long",
          10  "float",
          11  "double",
          12  "complex",
          13  "double complex"
          [default: 10 "float"]
```

C.73. kggauss — *Generate Gaussian Noise Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Noise*
Operator:	*Gaussian Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kggauss generates Gaussian noise of the specified parameters in the output data object (-o). The mean (-gmean) and variance (-gvar) must be specified for the Gaussian distribution to be generated.

The size of all five dimensions of the object can be specified. The data type of the output object can also be specified. *kggauss* can also generate complex valued Gaussian noise data.

See Also

DATAMANIP::kgnoise, DATAMANIP::kgexpon, DATAMANIP::kgpoisson, DATAMANIP::kgrayleigh, DATAMANIP::kguniform

Command Line Arguments

```
Usage for kggauss: Generate Gaussian Noise Data
% kggauss
  -o         (outfile) Output file containing noise data

  [-wsize]   (integer) Width size of noise value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of noise value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of noise value data
             (dsize > 0, default = 1)
  [-tsize]   (integer) Time size of noise value data
             (tsize > 0, default = 1)
  [-esize]   (integer) Element dimension size of noise data
             (esize > 0, default = 1)
  [-gmean]   (double)  Mean for Gaussian noise
             (unbounded, default = 0)
  [-gvar]    (double)  Variance for Gaussian noise
             (gvar > 0.0, default = 1)
  [-type]    (list)    Data type of noise value data
          1  "bit",
          2  "byte",
          3  "unsigned byte",
          4  "short",
          5  "unsigned short",
          6  "integer",
          7  "unsigned integer",
          8  "long",
          9  "unsigned long",
          10  "float",
          11  "double",
          12  "complex",
          13  "double complex"
          [default: 10 "float"]
```

C.74. kgnoise — *Generate Noise Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Noise*
Operator:	*Generate Noise*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

kgnoise generates noise of the specified type in the output data object (-o). *kgnoise* can generate the following kinds of noise: Gaussian, Rayleigh, Uniform, Poisson and Exponential. The appropriate parameters must be specified for the kind of noise being generated e.g. Mean & Variance for the Gaussian distribution and just the Variance for the Exponential distribution.

The size of all five dimensions of the object can be specified. The data type of the output object can also be specified. *kgnoise* can also generate complex valued noise data.

Examples

kgnoise -o random -wsize 512 -hsize 256 -esize 3 -type float -exp -evar 100

Creates a data object called random having width 512, height 256, depth 1, time 1 and 3 elements. The data storage type is float. The data object contains random data which is exponentially distributed and has a variance of 100.

Command Line Arguments

```
Usage for kgnoise: Generate Noise Data
% kgnoise
  -o         (outfile) Output file containing noise data

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:

    [Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
      [-gauss] (flag)   Generate Gaussian noise
      [-gmean] (double) Mean for Gaussian noise
               (unbounded, default = 0)
      [-gvar]  (double) Variance for Gaussian noise
               (gvar > 0.0, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-ray]  (flag)   Generate Rayleigh Noise
      [-rvar] (double) Variance for Rayleigh noise
              (rvar > 0.0, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
      [-unif] (flag)   Generate uniformly distributed noise
      [-umax] (double) Maximum value for uniform distribution
              (unbounded, default = 1)
      [-umin] (double) Minimum value for uniform noise
              (unbounded, default = 0)

    [Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
      [-poiss] (flag)   Generate Poisson Noise
      [-ptime] (double) Amount of time
               (ptime > 0.0, default = 1)
      [-pvar]  (double) Variance for Poisson distribution
               (pvar > 0.0, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-exp]  (flag)   Generate exponential noise
      [-evar] (double) Variance for exponential distribution
              (evar > 0.0, default = 1)

  [-wsize]   (integer) Width size of noise value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of noise value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of noise value data
             (dsize > 0, default = 1)
  [-tsize]   (integer) Time size of noise value data
             (tsize > 0, default = 1)
  [-esize]   (integer) Element dimension size of noise data
             (esize > 0, default = 1)
  [-type]    (list)    Data type of noise value data
          1  "bit",
```

```
2  "byte",
3  "unsigned byte",
4  "short",
5  "unsigned short",
6  "integer",
7  "unsigned integer",
8  "long",
9  "unsigned long",
10  "float",
11  "double",
12  "complex",
13  "double complex"
[default: 10 "float"]
```

C.75. kgpoisson — *Generate Poisson Noise Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Noise*
Operator:	*Poisson Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kgpoisson generates Poisson noise of the specified parameters in the output data object (-o). The amount of time (-ptime) and variance (-pvar) must be specified for the Poisson distribution to be generated.

The size of all five dimensions of the object can be specified. The data type of the output object can also be specified. *kgpoisson* can also generate complex valued Poisson noise data.

See Also

DATAMANIP::kgnoise, DATAMANIP::kgexpon, DATAMANIP::kggauss, DATAMANIP::kgrayleigh, DATAMANIP::kguniform

Command Line Arguments

```
Usage for kgpoisson: Generate Poisson Noise Data
% kgpoisson
  -o         (outfile) Output file containing noise data

  [-wsize]   (integer) Width size of noise value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of noise value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of noise value data
             (dsize > 0, default = 1)
  [-tsize]   (integer) Time size of noise value data
             (tsize > 0, default = 1)
  [-esize]   (integer) Element dimension size of noise data
             (esize > 0, default = 1)
```

```
  [-ptime]   (double)  Amount of time for Poisson noise
             (ptime > 0.0, default = 1)
  [-pvar]    (double)  Variance for Poisson noise
             (pvar > 0.0, default = 1)
  [-type]    (list)    Data type of noise value data
          1  "bit",
          2  "byte",
          3  "unsigned byte",
          4  "short",
          5  "unsigned short",
          6  "integer",
          7  "unsigned integer",
          8  "long",
          9  "unsigned long",
          10  "float",
          11  "double",
          12  "complex",
          13  "double complex"
          [default: 10 "float"]
```

C.76. kgrayleigh — *Generate Rayleigh Noise Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Noise*
Operator:	*Rayleigh Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kgrayleigh generates Rayleigh noise of the specified parameters in the output data object (-o). The variance (-rvar) must be specified for the Rayleigh distribution to be generated.

The size of all five dimensions of the object can be specified. The data type of the output object can also be specified. *kgrayleigh* can also generate complex valued Rayleigh noise data.

Restrictions

DATAMANIP::kgnoise, DATAMANIP::kgexpon, DATAMANIP::kggauss, DATAMANIP::kpoisson, DATAMANIP::kguniform

Command Line Arguments

```
Usage for kgrayleigh: Generate Rayleigh Noise Data
% kgrayleigh
  -o         (outfile) Output file containing noise data

  [-wsize]   (integer) Width size of noise value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of noise value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of noise value data
             (dsize > 0, default = 1)
```

```
[-tsize]    (integer) Time size of noise value data
            (tsize > 0, default = 1)
[-esize]    (integer) Element dimension size of noise data
            (esize > 0, default = 1)
[-rvar]     (double)  Variance for rayleigh noise
            (rvar > 0.0, default = 1)
[-type]     (list)    Data type of noise value data
        1  "bit",
        2  "byte",
        3  "unsigned byte",
        4  "short",
        5  "unsigned short",
        6  "integer",
        7  "unsigned integer",
        8  "long",
        9  "unsigned long",
        10  "float",
        11  "double",
        12  "complex",
        13  "double complex"
        [default: 10 "float"]
```

C.77. kgsin — *Generate Object Containing Sinusoidal Value Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Data*
Operator:	*Sinusoid*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kgsin creates a data object with sinusoidal values of dimension Width * Height * Depth * Time * Elements. The data type of the value segment can be **bit, byte, unsigned byte, short, unsigned short, integer, unsigned integer, long, unsigned long, float, double, complex** and **double complex**. When the desired data type is complex, the generated function is cos(w+h+d..) + j*sin(angle).

The sine wave generated can be either a sum of sine functions along all five dimensions(-sin5) or the sine of the sum of the coordinates at a point (-sin1). When the sum of sine functions is specified the output is generated using the expression:

sin(w) + sin(h) + sin(d) + sin(t) + sin(e)

For the sine of sum option the function used for generating the output looks like:

sin(w + h + d + t +e)

The number of sine waves and phase offset for each of the five dimensions can be specified. The number of cycles along each dimension represents the number of sine waves that will appear along each dimension. The phase offset is specified in degrees. wnum sine cycles and a phase offset of wp changes the argument to the sine generating function as follows:

sin(2*PI*wnum*w/width + wp) where width denotes the size of the width dimension.

The amplitude of the sinusoidal function can also be specified. It represents the maximum value that each sine wave can take. In the case when a sum of sine waves is being generated the amplitude along each dimension has to be specified. The generated expression looks like:

ampl*sin(angle) or wampl*sin(wnum*w/width+ wp) + hampl*sin(angle) + ...

Command Line Arguments

```
Usage for kgsin: Generate Object Containing Sinusoidal Value Data
% kgsin
  -o         (outfile) Output file containing sinusoidal value data

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-sin1] (flag)   Apply sine function to the sum of coordinates at a point
      [-ampl] (double) Maximum level of output
              (unbounded, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 6 options:]
      [-sin5]  (flag)   Apply sine to each coordinate and sum up the result
      [-wampl] (double) Amplitude along the width direction
               (unbounded, default = 1)
      [-hampl] (double) Amplitude along the height direction
               (unbounded, default = 1)
      [-dampl] (double) Amplitude along the depth direction
               (unbounded, default = 1)
      [-tampl] (double) Amplitude along the time direction
               (unbounded, default = 1)
      [-eampl] (double) Amplitude along the element direction
               (unbounded, default = 1)

  [-wsize]   (integer) Width size of sinusoidal value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of sinusoidal value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of sinusoidal value data
             (dsize > 0, default = 1)
  [-tsize]   (integer) Time size of sinusoidal value data
             (tsize > 0, default = 1)
  [-esize]   (integer) Elements dimension size of sinusoidal value data
             (esize > 0, default = 1)
  [-wnum]    (double)  Number of sine waves along width dimension
             (wnum >= 0.0, default = 1)
  [-wp]      (double)  Phase offset along width dimension
             (-100 <= wp <= 100, default = 0)
  [-hnum]    (double)  Number of sine waves along height dimension
             (hnum >= 0.0, default = 1)
  [-hp]      (double)  Phase offset along the height dimension
             (-100 <= hp <= 100, default = 0)
  [-dnum]    (double)  Number of sine waves along depth dimension
             (dnum >= 0.0, default = 1)
  [-dp]      (double)  Phase offset along depth dimension
             (-100 <= dp <= 100, default = 0)
```

```
[-tnum]     (double)  Number of sine waves along time dimension
            (tnum >= 0.0, default = 1)
[-tp]       (double)  Phase Offset for time dimension
            (-100 <= tp <= 100, default = 0)
[-enum]     (double)  Number of sine waves along Elements Dimension
            (enum >= 0.0, default = 1)
[-ep]       (double)  Phase Offset in the elements dimension
            (-100 <= ep <= 100, default = 0)
[-type]     (list)    Data type of sinusoidal value data
        1  "bit",
        2  "byte",
        3  "unsigned byte",
        4  "short",
        5  "unsigned short",
        6  "integer",
        7  "unsigned integer",
        8  "long",
        9  "unsigned long",
        10  "float",
        11  "double",
        12  "complex",
        13  "double complex"
        [default: 10 "float"]
```

C.78. kgt — *IF Input 1 is Greater Than (Input 2 or Constant), Output = TRUE*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	*>*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The GREATER THAN comparison operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value, which ever is specified by the user. If the value from **Input 1** is greater than the value from **Input 2** or the **Constant**, the corresponding point in **Output** will be assigned the value specified by **TRUE Value**. If the value from **Input 1** is not greater than the value against which it is being compared, the corresponding point in **Output** will be the value specified by **FALSE Value**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects
Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing the Greater Than operator runs the program *kcompare* with the -gt flag.

See Also

DATAMANIP::kcompare

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Command Line Arguments

```
Usage for kgt: IF Input 1 is Greater Than (Input 2 or Constant), Output = TRUE
% kgt
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) Second input data object
            (default = {none})
    [-real] (double) Constant value
            (real >= 0.0, default = 0)

  [-tval]    (double)  Value assigned to output if evaluation is TRUE
```

```
             (unbounded, default = 1)
  [-fval]    (double)  Value assigned to output if evaluation is FALSE
             (unbounded, default = 0)
```

C.79. kguniform — *Generate Uniform Noise Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Noise*
Operator:	*Uniform Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kguniform generates uniform noise of the specified parameters in the output data object (-o). The minimum (-umin) and maximum (-umax) must be specified for the uniform distribution to be generated.

The size of all five dimensions of the object can be specified. The data type of the output object can also be specified. *kguniform* can also generate complex valued uniform noise data.

See Also

DATAMANIP::kgnoise, DATAMANIP::kgexpon, DATAMANIP::kggauss, DATAMANIP::kgpoisson, DATAMANIP::kgrayleigh

Command Line Arguments

```
Usage for kguniform: Generate Uniform Noise Data
% kguniform
  -o         (outfile) Output file containing noise data

  [-wsize]   (integer) Width size of noise value data
             (wsize > 0, default = 512)
  [-hsize]   (integer) Height size of noise value data
             (hsize > 0, default = 512)
  [-dsize]   (integer) Depth size of noise value data
             (dsize > 0, default = 1)
  [-tsize]   (integer) Time size of noise value data
             (tsize > 0, default = 1)
  [-esize]   (integer) Element dimension size of noise data
             (esize > 0, default = 1)
  [-umin]    (double)  Minimum for uniform noise
             (unbounded, default = 0)
  [-umax]    (double)  Maximum for uniform noise
             (umax > 0.0, default = 1)
  [-type]    (list)    Data type of noise value data
          1  "bit",
          2  "byte",
          3  "unsigned byte",
          4  "short",
          5  "unsigned short",
          6  "integer",
          7  "unsigned integer",
```

```
8  "long",
9  "unsigned long",
10  "float",
11  "double",
12  "complex",
13  "double complex"
[default: 10 "float"]
```

C.80. kheq — *Perform Histogram Equalization*

Object Information

Category:	*Data Manip*	Subcategory:	*Histogram Operators*
Operator:	*Equalize*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Equalize Histogram* operator (kheq) is used to perform histogram equalization on the Input object (i). The idea is to spread out the histogram values to fully occupy the range of the data type to enhance detail that would be otherwise lost. The grey levels in the data can be optionally inverted after histogram processing (this is often helpful for astronomical data).

Histogram equalization employs a nonlinear mapping of input values to output values. The nonlinear mapping is a scaled integral of the histogram of the input data. Equalization can make the data look visually harsh, but can bring out amazing detail in poor contrast data. Histogram equalization is described in detail in texts on image processing.

If a mask is present with the input object, data points that are masked off are ignored in the histogramming. The mask is propagated to the output object.

If a map is present with the input object, all data will be pulled through the mask before histogramming. The output object will have no map.

For signed data types, the output will span the whole range of the data type, just as for the unsigned data types. Note that for signed types, this will include the negative part of the range. If this behavior is not desired, the data should be converted to unsigned prior to the application of the histogram equalization.

The data types that can be handled are currently limited to signed and unsigned byte, and signed and unsigned short integer. The number of bits in a data point is further limited by the internal algorithms to be at most 16, signed or unsigned. If an otherwise acceptable data type is found that exceeds 16 bits (say a 32 bit short) then an error message will be printed. Resolution of this problem is architecture dependent.

The histogram equalization can be performed in units of data that can span any of the polymorphic data model dimensions. By default, the unit is composed of 2D planes across width and height (i.e. a normal "image"). It is possible to select processing in volumes, sequences, slices, rows, columns or the whole 5D data set if that is desired. Each unit is histogrammed independently of the others.

This operator supports processing of large data sets via data services. Data access speed is a function of the processing unit dimensions, so significant processing time variance can be observed for varying unit geometries even though the total number of points processed is the same.

Executing *Equalize Histogram* runs the program *khistops* with the -equalize flag.

See Also

DATAMANIP::khistops, DATAMANIP::khstr

Restrictions

Only KBYTE, KUBYTE, KSHORT, and KUSHORT data is supported, and only then if they are 16 bits or less in length.

References

R.C. Gonzalez and P.A. Wintz, "Digital Image Processing", Addison-Wesley, 1987, ISBN 0-201-11026-1.

Command Line Arguments

```
Usage for kheq: Perform Histogram Equalization
% kheq
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  [-invert]  (flag)    invert grey levels after processing

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-whole] (flag) process whole data set at one time

    [Optional Loose Group - Specify at least one of the 5 options:]
      [-w] (flag) include width in processing unit
      [-h] (flag) include height in processing unit
      [-d] (flag) include depth in processing unit
      [-t] (flag) include time in processing unit
      [-e] (flag) include elements in processing unit
```

C.81. khisto — *Compute Histogram for Data Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Histogram Operators*
Operator:	*Histogram*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

khisto computes histograms from data objects and stores the histogram data in another data object. One histogram is computed and stored for each "unit" of data in the input data object, where a unit is defined by the settings of the -w,-h,-d,-t,-e and -whole options. If none of these flags are supplied, then the default will be WxH unit processing.

Histograms are stored in the output data object as column vectors in the WxH plane, one column per unit. If the input data object has many units spanning several dimensions, the histograms are still stored only in the WxH plane. This promotes rapid visual inspection of the histograms using the image visualization and plotting tools. Histograms are stored as the units are read using the standard whdte storage order.

The histogram structure is specified by three parameters:

minimum side of minimum bin
: This number specifies the most negative number that can be trapped in the histogram array. Another way to define it is as the left edge of first histogram bin.

binwidth
: This is the width of each histogram bin.

bins
: This is the total number of bins to be used in the histogram.

Values are assigned to bin index [0...bins-1] using the following procedure:

bin_num = (input_val - mina) / binwidth

if (input_val == min + bins*binwidth) bin_num = bins - 1

With this arrangement, a bin collects input values ranging from the value of the left edge inclusively up to the value of the right edge exclusively. The exception is the last bin which collects values between the left and right edges inclusively (for both edges). If the computed bin_num falls outside of the array, then that data point is ignored in the histogram.

If the input data object has a mask, then data points marked as invalid by the mask are ignored in the histogram calculations.

If a gate object is supplied, then only those data points with a value corresponding to non-zero in the gate image will be histogrammed. If the input has a mask, then data points are ignored if masked or marked for "ignore" by the gate object. The gate object must have the same dimensions as the input object.

If the input data object has a map, then the data is pulled through the map before histogramming. Note that this can significantly multiply the number of "units" in the data. For example, a 3-band image might have only 1 unit of dimensions WxH when not mapped, but when the data is pulled through the map there will be 3 units of dimension WxH.

If the out-of-bounds flag (-oob) is supplied, then an extra two bins will be placed at the end of the histogram. The first one will contain the number of data values that fell outside of the histogram array on the low (negative) side, and the second will contain the number of data values that fell outside the histogram array on the high (positive) side. This option is useful when little is known about the range of the data and one is not sure if the specified range of the histogram array is capturing all of the data.

khisto utilizes data services internally and can process very large data sets. All internal calculations are done in double precision. All input data is converted to KDOUBLE by data services after being read from the input data object. The output histogram data object is of type KULONG, unsigned long integer. Complex input data is converted by taking the real part.

Examples

% khisto -i sequence:clouds -o h.clouds -w -h -min 0 -binwidth 1 -bins 256- Compute Histogram for Data Object
will construct in interesting histogram data set from the cloud sequence. This sequence is comprise of a set of 512x480 frames, taken 15 seconds apart, viewing a section of sky as a storm develops, for a total of 582 frames. The output is best visualized using editimage with the SA pseudocolor option turned on.

See Also

khistops(1)

Restrictions

If the input data is complex, then you may want to convert it yourself using whatever means you require; the default for complex input is to take only the real part.

Command Line Arguments

```
Usage for khisto: Compute Histogram for Data Object
% khisto
  -i          (infile)  Input data object
  -o          (outfile) Output data object

  [-igate]    (infile)  Gating input data object
              (default = {none})
  [-bins]     (integer) number of histogram bins
              (bins > 0, default = 256)
  [-binwidth] (double)  width of histogram bin
              (unbounded, default = 1)
  [-min]      (double)  Min side of min histogram bin
              (unbounded, default = 0)
  [-oob]      (flag)    append a pair of bins for out-of-bounds data

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-whole] (flag) histogram whole data set at one time

    [Optional Loose Group - Specify at least one of the 5 options:]
      [-w] (flag) include width in histogram unit
      [-h] (flag) include height in histogram unit
      [-d] (flag) include depth in histogram unit
      [-t] (flag) include time in histogram unit
      [-e] (flag) include elements in histogram unit
```

C.82. khistops — *Perform Histogram Equalization and Stretching*

Object Information

Category:	*Data Manip*	Subcategory:	*Histogram Operators*
Operator:	*Histogram Ops*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

khistops is used to perform histogram equalization and stretching operations on object. In both cases, the idea is to spread out the histogram values to fully occupy the range of the data type to enhance detail that would be otherwise lost. The grey levels in the data can be optionally inverted after histogram processing (this is often helpful for astronomical data).

Histogram equalization employs a nonlinear mapping of input values to output values. The nonlinear mapping is a scaled integral of the histogram of the input data. Stretching employs a simple linear mapping. Equalization can make the data look visually harsh, but can bring out amazing detail in poor contrast data. Stretching can bring out some detail but preserves the relative brightness of the data. Both of these techniques are described in detail in texts on image processing.

If a mask is present with the input object, data points that are masked off are ignored in the histogramming. The mask is propagated to the output object.

If a map is present with the input object, all data will be pulled through the mask before histogramming. The output object will have no map.

For signed data types, the output will span the whole range of the data type, just as for the unsigned data types. Note that for signed types, this will include the negative part of the range. If this behavior is not desired, the data should be converted to unsigned prior to the application of *khistops.*

The data types that can be handled are currently limited to signed and unsigned byte, and signed and unsigned short integer. The number of bits in a data point is further limited by the internal algorithms to be at most 16, signed or unsigned. If an otherwise acceptable data type is found that exceeds 16 bits (say a 32 bit short) then an error message will be printed. Resolution of this problem is architecture dependent.

The histogram operations can be performed in units of data that can span any of the polymorphic data model dimensions. By default, the unit is composed of 2D planes across width and height (i.e. a normal "image"). It is possible to select processing in volumes, sequences, slices, rows, columns or the whole 5D data set if that is desired. Each unit is histogrammed independently of the others.

khistops supports processing of large data sets via data services. Data access speed is a function of the processing unit dimensions, so significant processing time variance can be observed for varying unit geometries even though the total number of points processed is the same.

Examples

% khistops -i image:moon -o moon.heq -equalize -whole- Perform Histogram Equalization and Stretching
will produce a histogram equalized rendition of the moon image.

Restrictions

Only KBYTE, KUBYTE, KSHORT, and KUSHORT data is supported, and only then if they are 16 bits or less in length.

References

R.C. Gonzalez and P.A. Wintz, "Digital Image Processing", Addison-Wesley, 1987, ISBN 0-201-11026-1.

Command Line Arguments

```
Usage for khistops: Perform Histogram Equalization and Stretching
% khistops
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-equalize] (flag) perform histogram equalization
    [-stretch]  (flag) perform linear histogram stretch

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-whole] (flag) process whole data set at one time

    [Optional Loose Group - Specify at least one of the 5 options:]
      [-w] (flag) include width in processing unit
      [-h] (flag) include height in processing unit
      [-d] (flag) include depth in processing unit
      [-t] (flag) include time in processing unit
      [-e] (flag) include elements in processing unit

  [-invert]  (flag)    invert grey levels after processing
```

C.83. khstr — *Perform Histogram Stretching*

Object Information

Category:	*Data Manip*	Subcategory:	*Histogram Operators*
Operator:	*Stretch*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Stretch Histogram* operator (kheq) is used to perform histogram stretching on the Input object (i). The idea is to spread out the histogram values to fully occupy the range of the data type to enhance detail that would be otherwise lost. The grey levels in the data can be optionally inverted after histogram processing (this is often helpful for astronomical data).

Histogram stretching employs a simple linear mapping. Stretching can bring out some detail but preserves the relative brightness of the data. Histogram stretching is described in detail in texts on image processing.

If a mask is present with the input object, data points that are masked off are ignored in the histogramming. The mask is propagated to the output object.

If a map is present with the input object, all data will be pulled through the mask before histogramming. The output object will have no map.

For signed data types, the output will span the whole range of the data type, just as for the unsigned data types. Note that for signed types, this will include the negative part of the range. If this behavior is not desired, the data should be converted to unsigned prior to the application of the histogram stretching.

The data types that can be handled are currently limited to signed and unsigned byte, and signed and unsigned short integer. The number of bits in a data point is further limited by the internal algorithms to be at most 16, signed or unsigned. If an otherwise acceptable data type is found that exceeds 16 bits (say a 32 bit short) then an error message will be printed. Resolution of this problem is architecture dependent.

The histogram stretching can be performed in units of data that can span any of the polymorphic data model dimensions. By default, the unit is composed of 2D planes across width and height (i.e. a normal "image"). It is possible to select processing in volumes, sequences, slices, rows, columns or the whole 5D data set if that is desired. Each unit is histogrammed independently of the others.

This operator supports processing of large data sets via data services. Data access speed is a function of the processing unit dimensions, so significant processing time variance can be observed for varying unit geometries even though the total number of points processed is the same.

Executing *Stretch Histogram* runs the program *khistops* with the -stretch flag.

See Also

DATAMANIP::khistops, DATAMANIP::kheq

Restrictions

Only KBYTE, KUBYTE, KSHORT, and KUSHORT data is supported, and only then if they are 16 bits or less in length.

References

R.C. Gonzalez and P.A. Wintz, "Digital Image Processing", Addison-Wesley, 1987, ISBN 0-201-11026-1.

Command Line Arguments

```
Usage for khstr: Perform Histogram Stretching
% khstr
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  [-invert]  (flag)    invert grey levels after processing

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-whole] (flag) process whole data set at one time

    [Optional Loose Group - Specify at least one of the 5 options:]
      [-w] (flag) include width in processing unit
      [-h] (flag) include height in processing unit
      [-d] (flag) include depth in processing unit
      [-t] (flag) include time in processing unit
      [-e] (flag) include elements in processing unit
```

C.84. khypot — *Output = sqrt[(Input 1)**2 + (Input 2 or Constant)**2]*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Hypotenuse*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Hypotenuse* operator returns the square root of the result produced by adding the square of each point in **Input 1** and the square of either the corresponding point in **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Hypotenuse* runs the program *karith2* with the -hypot flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2, DATAMANIP::kcmplx2real, DATAMANIP::kmag

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real. To calculate the hypotenuse, or magnitude of a complex number, use DATAMANIP::kcmplx2real.

Command Line Arguments

```
Usage for khypot: Output = sqrt[(Input 1)**2 + (Input 2 or Constant)**2]
% khypot
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) Second input data object
            (default = {none})
    [-real] (double) Real constant value
            (unbounded, default = 0)
```

C.85. kimagpart — *Output = Imaginary Component of (Complex) Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Imaginary Part*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Imaginary Part* operator returns the imaginary component of each complex data point in the input data object, **Input**. If the input data is not complex, the imaginary component is assumed to be zero.

The output object data type will be the same as the input (regarding single or double floats).

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Imaginary Part* runs the program *kcmplx2real* with the -oi output option.

See Also

DATAMANIP::kcmplx2real, DATAMANIP::kcmplx, DATAMANIP::kreal2cmplx

Command Line Arguments

```
Usage for kimagpart: Output = Imaginary Component of (Complex) Input
% kimagpart
  -i         (infile)  input data object
  -oi        (outfile) resulting output data object
```

C.86. kimportasc — *Import ASCII Data File to Data Segment*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*Import ASCII*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine will import an ASCII data file and insert it into a specified data object. The new segment will be inserted into this data object. If a source object was specified, then all of its data segments and the global object attributes will be copied to the destination. If no source object was specified, then the resulting output object will contain only the imported segment.

There are four attributes of the new segment that must be passed in : dimension, index order, size and data type. The number of indices and sizes specified must be equal to the dimension.

If the segment_name is one of the Polymorphic segments, the correct dimensionality must be provided :

Segment Name	Dimension
KDA_SEGMENT_VALUE	5
KDA_SEGMENT_MAP	5
KDA_SEGMENT_MASK	5
KDA_SEGMENT_LOCATION	4
KDA_SEGMENT_TIME	1

The parameter dimension must be between 0 and 5.

The string index order contains the indices of the segment separated by spaces, "elements width height depth time" or "e w h d t". The only allowed values are:

- width (w)
- height (h)
- depth (d)
- time (t)
- elements (e)
- dimension (n)
- map width (mw)
- map height (mh)
- map depth (md)
- map time (mt)
- map elements (me)

where the characters in the ()'s are accepted abbreviations.

The segment size is a string that contains the size of the segment to create. All the values must be > 1. It has the size value of each index separated by a space "40 32 32 1 1".

The start offset is the point to start importing in the ASCII data. All points before this one are ignored.

The skip factor is the number of points to skip in the ASCII data between reads. If the size of the new segment's lowest index is 40 and the skip factor is 5, then this routine will read in 40 points skip 5 and read in 40 more, etc. until the entire ASCII file read in.

Complex values in ASCII are represented by number pairs, surrounded by parenthesis and separated by a comma. For example, (8, 3) represents the complex number 8+3i.

See Also

kasc2val, kasc2mask, kasc2loc, kasc2time, kasc2map

Command Line Arguments

```
Usage for kimportasc: Import ASCII Data File to Data Segment
% kimportasc
  -i1        (infile)  ASCII input data file containing new data segment
  -size      (string)  This is a string containing the size of the segment in the index order specified , 40 32 32 1 1 will be a 5D segment that is 40x32x32x1x1
  -o         (outfile) output data object
```

```
  [-i2]      (infile)  data object input to insert the new segment into
             (default = {none})
  [-so]      (integer) starting point in ASCII input file to start importing at
             (so >= 0, default = 0)
  [-sf]      (integer) number of points in ASCII file to skip between reads
             (sf >= 0, default = 0)
  [-segment] (string)  the name of the segment to insert the ASCII data into
             (default = value)
  [-dim]     (integer) dimension of the segment
             (dim > 0, default = 5)
  [-type]    (list)    data type
          1  "Bit",
          2  "Byte",
          3  "Unsigned Byte",
          4  "Short",
          5  "Unsigned Short",
          6  "Integer",
          7  "Unsigned Integer",
          8  "Long",
          9  "Unsigned Long",
          10  "Float",
          11  "Double",
          12  "Complex",
          13  "Double Complex"
          [default: 11 "Double"]
  [-index]   (string)  index order of the segment
             (default = elements width height depth time)
```

C.87. kimportraw — *Low Level Import of Raw Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Import Data*
Operator:	*Import Raw*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kimport_raw can be used to read raw binary data and import it to a data segment. If the second input file is supplied, *Add-to Object 2*, the new data will be appended onto that object. Otherwise, an new object is created. The data is stored in the *Destination segment* specified.

The source architecture and data type information must be accurate to allow the data to be imported correctly.

Command Line Arguments

```
Usage for kimportraw: Low Level Import of Raw Data
% kimportraw
  -i1        (infile)  raw (binary) input
  -o         (outfile) output object
```

```
[-i2]       (infile)  Object to add this data to
            (default = {none})
[-arch]     (list)    data source architecture
         1  "Local",
         2  "Intel 80x86",
         3  "SPARC (Sun4,Solbourne)",
         4  "MIPS-EL (DecStation)",
         5  "IBM RT",
         6  "RS6000 (SP-1,SP-2)",
         7  "POWER PC",
         8  "Motorola 680x0 (Mac, Next, Sun3)",
         9  "Motorola 88x00 (DG/Avion, LUNA)",
         10  "MIPS-EB (SGI)",
         11  "PA-RISC (HP9000)",
         12  "Convex",
         13  "Apollo",
         14  "DEC Alpha",
         15  "CDC Cyber",
         16  "Cray",
         17  "DEC VAX",
         18  "Intel Paragon"
         [default: 1 "Local"]
[-type]     (list)    data type of raw data
         1  "Bit",
         2  "Byte",
         3  "Unsigned Byte",
         4  "Short",
         5  "Unsigned Short",
         6  "Integer",
         7  "Unsigned Integer",
         8  "Long",
         9  "Unsigned Long",
         10  "Float",
         11  "Double",
         12  "Complex",
         13  "Double Complex"
         [default: 3 "Unsigned Byte"]
[-wsize]    (integer) Size of width dimension
            (wsize > 0, default = 1)
[-hsize]    (integer) Size of height dimension
            (hsize > 0, default = 1)
[-dsize]    (integer) Size of depth dimension
            (dsize > 0, default = 1)
[-tsize]    (integer) Size of time dimension
            (tsize > 0, default = 1)
[-esize]    (integer) Size of elements dimension
            (esize > 0, default = 1)
[-segment] (list)    destination segment for incoming data
         1  "Value",
         2  "Mask",
         3  "Map",
         4  "Location ",
         5  "Time "
         [default: 1 "Value"]
```

C.88. kimpulse — *Generate Object Containing Impulse Value Data*

Object Information

Category:	*Input/Output*	Subcategory:	*Generate Data*
Operator:	*Impulse*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kimpulse creates a data object with impulses in the value segment. The dimension of the object is Width * Height * Depth * Time * Elements. The data type of the value segment can be **bit, byte, unsigned byte, short, unsigned short, integer, unsigned integer, long, unsigned long, float, double, complex** and **double complex**.

The value assumed by each impulse can be specified (-fgreal and -fgimag). Both the real and imaginary component of the value assumed by the impulse, are accepted but the imaginary value is used only when the output type is complex or double complex. The value assumed by the background is also specifiable by the -bgreal and -bgimag variables. Like the impulse value, the imaginary background value is used only when the output type is complex or double complex.

The spacing between two impulses can be specified for each of the 5 dimensions of the output object (-wsp -hsp -dsp -tsp & -esp). The number of impulses along each dimension can also be specified (-wnum -hnum -dnum -tnum & -enum). This parameter is useful if impulses are not desired beyond a certain point in each dimension. If the number of impulses specified is greater than the number obtained by dividing the size of the object by the spacing between impulses then the latter denotes the number of impulses along a dimension.

It is also possible to specify an offset for each of the 5 dimensions. The first impulse is generated relative to the offset value specified by the variables -woff -hoff -doff -toff -eoff. An error message is generated if the offset specified is greater than the size of that dimension.

Examples

kimpulse -wsize 512 -hsize 512 -w_off 255 -h_off 255 -w_num 1 -h_num 1 -o example

creates an image example of size 512x512 with an impulse of value 255 in the center. The background has the value 0.

Command Line Arguments

```
Usage for kimpulse: Generate Object Containing Impulse Value Data
% kimpulse
  -o         (outfile) Output file containing impulse value data

  [-wsize]   (integer) Width of impulse value data
             (wsize > 0, default = 512)
  [-wnum]    (integer) Number of impulses generated along the width dimension
             (wnum > 0, default = 1)
  [-hsize]   (integer) Height of impulse value data
```

```
             (hsize > 0, default = 512)
[-hnum]      (integer) Number of impulses generated along the height dimension
             (hnum > 0, default = 1)
[-dsize]     (integer) Depth of impulse value data
             (dsize > 0, default = 1)
[-dnum]      (integer) Number of impulses generated along the depth dimension
             (dnum > 0, default = 1)
[-tsize]     (integer) Time of impulse value data
             (tsize > 0, default = 1)
[-tnum]      (integer) Number of impulses generated along the time dimension
             (tnum > 0, default = 1)
[-esize]     (integer) Element dimension of impulse value data
             (esize > 0, default = 1)
[-enum]      (integer) Number of impulses generated along the element dimension
             (enum > 0, default = 1)
[-wsp]       (integer) Spacing between two impulses along the width dimension
             (wsp > 0, default = 256)
[-woff]      (integer) Offset of first impulse along width dimension
             (unbounded, default = 1)
[-hsp]       (integer) Spacing between two impulses along the height dimension
             (hsp > 0, default = 256)
[-hoff]      (integer) Offset of first impulse along height dimension
             (unbounded, default = 1)
[-dsp]       (integer) Spacing between two impulses along the depth dimension
             (dsp > 0, default = 1)
[-doff]      (integer) Offset of first impulse along depth dimension
             (unbounded, default = 0)
[-tsp]       (integer) Spacing between two impulses along the time dimension
             (tsp > 0, default = 1)
[-toff]      (integer) Offset of first impulse along time dimension
             (unbounded, default = 0)
[-esp]       (integer) Spacing between two impulses along the element dimension
             (esp > 0, default = 1)
[-eoff]      (integer) Offset of the first impulse along the element dimension
             (unbounded, default = 0)
[-fgreal]    (double)  Real value assigned to the generated impulse
             (unbounded, default = 255)
[-bgreal]    (double)  Real value assigned to background value data
             (unbounded, default = 0)
[-fgimag]    (double)  Imaginary value of generated impulse
             (unbounded, default = 0)
[-bgimag]    (double)  Imaginary value assigned to background
             (unbounded, default = 0)
[-type]      (list)    Data type of impulse value data
          1  "bit",
          2  "byte",
          3  "unsigned byte",
          4  "short",
          5  "unsigned short",
          6  "integer",
          7  "unsigned integer",
          8  "long",
          9  "unsigned long",
          10  "float",
          11  "double",
          12  "complex",
          13  "double complex"
          [default: 3 "unsigned byte"]
```

C.89. kinsertseg — *Insert Selected Data Segments from Input 1 into Input 2*

Object Information

Category:	*Data Manip*	Subcategory:	*Segment Operators*
Operator:	*Insert Segments*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This program will insert the specified segment(s) for the first input object into the second input object producing a new or output object. If -segment is "Insert All" all the segments and their associated attributes will be inserted. The -segment parameter can also specify multiple segments by typing a string containing the segment names separated by space(s):

"segment1 segment2 segment3"

Command Line Arguments

```
Usage for kinsertseg: Insert Selected Data Segments from Input 1 into Input 2
% kinsertseg
  -i1        (infile)  Source Input File
  -i2        (infile)  Destination Input File
  -o         (outfile) output file

  Required Loose Group - Specify at least one of the 6 options:
    [-val]     (flag)   If -val flag is selected, operate on Value segment
    [-map]     (flag)   If -map flag is selected, operate on map segment
    [-mask]    (flag)   If -mask flag is selected, operate on Mask segment
    [-loc]     (flag)   If -loc flag is selected, operate on Location segment
    [-time]    (flag)   If -time flag is selected, operate on Time segment
    [-segment] (string) Perform operation on user specified segment
               (default = Insert All)
```

C.90. kinset — *Inset Object 2 into Object 1*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*Inset*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The Inset operator, kinset, insets data from *Input 2* (i2) into *Input 1* (i1) at the specified position, replacing the data in *Input 1*. The resulting *Output* object (o) size will be at least the size of Input 1, and may be larger, depending on the specified position values and the size of Input 2.

If Input 2 has a mask, an "Only Inset VALID Data" (insetvalid) parameter can be set to define whether only data from Input 2 that is marked as valid by the validity mask is inset into Input 1. If this parameter is set to FALSE (-insetvalid 0), all value data and the mask from Input 2 are inset into Input 1. If the parameter is TRUE (-insetvalid 1), for each point in Input 2, the mask is checked, and only valid points are inset into Input 1. In this case, the mask from Input 2 is not propagated to the output.

The inset operation is based on implicit indexing. This means that if location or time data exist, the inset operation is not done in terms an interpretation of the location/time data values, but in terms of the implicit indexing of these data (which is specified by the Width, Height, Depth, Time, Elements indices of the polymorphic data model).

The position in Input 1 where the origin of Input 2 will be placed can be determined from the Sub-Object Position attribute that is stored in Input 2, or it can be specified explicitly by providing Width, Height, Depth, Time and Element Coordinates (w,h,d,t,e). The subobject position attribute is automatically set by programs such as the Extract operator (kextract). It is legal to specify a subset of coordinates, and let the subobject position attribute define the rest of the coordinates.

In some instances, padding is necessary to maintain the integrity of the polymorphic data model - for example when the final size of the Output object is larger than that of Input 1. The Real and Imaginary Pad Values (real, imag) define the values that will be used when padding.

If padding occurs, and the option to "Identify padded data added by this program as Valid" is selected (valid TRUE), all padded data is considered valid and, if either input object contains a validity mask, the output object will have a mask, and the mask value for the padded data will be 1. The output object will not contain a mask if neither input object has a mask. If padded data is to be identified as Invalid (valid FALSE), the padded data will be masked as invalid (0). In this case, even if neither input object has a validity mask, the output object will have one.

Map Data

When either of the input objects has map data, the treatment of the maps, and how the data is represented in the output object, depends on the mapping option (mapping) specified by the user. Possible mapping options are: (0) Map Data Thru Maps and (1) Use First Map Only. If neither input object has a map, the mapping option is ignored. If there are doubts about which mapping option to use, the safest bet is to map the data thru the maps.

Map Data Thru Maps:

All data will be mapped before the data objects are combined. The output object will not have a map.

Use First Map Only:

In this case, the map data and color attributes of the first input object that has a map are directly transferred to the output object. Note that by selecting this mapping option, you are assuming that the value segments of both objects have valid indices into that map.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Location & Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes

This program will fail if either input object lacks value data.

See Also

DATAMANIP::kappend
DATAMANIP::kextract

Command Line Arguments

```
Usage for kinset: Inset Object 2 into Object 1
% kinset
  -i1           (infile)  Base input image
  -i2           (infile)  Second (inset) input image
  -o            (outfile) Resulting output object

  [-insetvalid] (boolean) If Object has Mask, Only Inset VALID DataIf
                (default = true)
  [-real]       (double)  Real constant pad level
                (unbounded, default = 0)
  [-imag]       (double)  Imaginary constant pad level
                (unbounded, default = 0)
  [-padvalid]   (boolean) Identify padded data added by this program as valid or invalid in mask
                (default = true)
  [-mapping]    (list)    Defines map treatment when some, but not all, objects have maps
          0  "Map Data Thru Maps ",
          1  "Use First Map Only"
          [default: 0 "Map Data Thru Maps "]

  [Optional Loose Group - Specify at least one of the 6 options:]
    [-attr] (flag)    Insert at i2 subobject position (attribute) - w,h,d,t,e will override if specified
    [-woff] (integer) Insert region beginning at this width coordinate
            (unbounded, default = 0)
    [-hoff] (integer) Insert region beginning at this height coordinate
            (unbounded, default = 0)
    [-doff] (integer) Insert region beginning at this depth coordinate
            (unbounded, default = 0)
    [-toff] (integer) Insert region beginning at this time coordinate
            (unbounded, default = 0)
    [-eoff] (integer) Insert region beginning at this element coordinate
            (unbounded, default = 0)
```

C.91. kinverse — *Inverse Filtering (Restoration) in Fourier Frequency Domain*

Object Information

Category:	*Data Manip*	Subcategory:	*Frequency Filters*
Operator:	*InvFilter*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kinverse performs restoration of data in the spatial domain by inverse filtering in the frequency domain.

The idea is to deconvolve the blurring function from the blurred data. A few problems exist in performing this operation, as described in great detail in any good image processing book.

The major difficulty lies in how small the denominator is allowed to be before you call it garbage. The value chosen is supplied as the -t thresh value. Ideally, this will be just above the noise floor in the image, but experimentation is the best guide here. *kinverse* places zeros in the output data for those data points where the PSF data object magnitude is below the threshold. In this respect, kinverse is actually an implementation of a *pseudoinverse* filter.

Note that the input data objects will be treated simply as a field of complex data, presumably obtained by application of an FFT, although the data could be generated by many other means. The output data will be either KCOMPLEX if the input data is any type other than KDCOMPLEX. Otherwise, the output will be of type KDCOMPLEX. Data objects with maps attached are not supported.

References

R.C. Gonzalez, P. Wintz, "Digital Image Processing", Addison-Wesley 1987, ISBN 0-201-11026-1, pp. 221-224.

Command Line Arguments

```
Usage for kinverse: Inverse Filtering (Restoration) in Fourier Frequency Domain
% kinverse
  -i1        (infile)  source data object (FFT)
  -i2        (infile)  PSF data object (FFT)
  -o         (outfile) filtered data object (FFT)

  [-t]       (double)  filter threshold (magnitude)
             (t >= 0.0, default = 1)
```

C.92. kkmeans — *Perform K-Means Clustering*

Object Information

Category:	*Data Manip*	Subcategory:	*Clustering Operators*
Operator:	*K-Means*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kkmeans accepts an input data object containing vectors of equal size and performs the K-means clustering algorithm on the vectors. The length of each vector is determined by the elements (E) dimension of the input data object.

The K-means algorithm is based on minimization of the sum of the squared distances from all points in a cluster to a cluster center. The user chooses K initial cluster centers and the input vectors are iteratively distributed among the K cluster domains. New cluster centers are computed from these results, such that the sum of the squared distances from all points in a cluster to the new cluster center is minimized.

Although the K-means algorithm does not really converge (in a continuous space), it may converge in a discrete space or a practical upper limit can be chosen for convergence. The user has the option of specifying the maximum number of iterations using the -n option. The default is 50000 iterations.

There are two ways to specify the initial cluster centers. If the -i2 argument is supplied, then the cluster centers are read from the specified object. The vectors are assumed to be stored along the E direction. Only the first K centers (as specified by the -k argument) will be read. If the -i2 argument is *not* present, then the first K vectors in the -i1 object will be used as the initial cluster centers.

It should be noted that it is possible to specify an initial cluster that lies at a sufficient distance from all input vectors that it will have no vectors assigned to it during a pass of the K-means algorithm. If this happens, *kkmeans* will reinitialize the value of that cluster to the mean value of a moving pair of the existing cluster centers, thus avoiding degeneracy.

If no options are selected, the output object specified by the -o1 argument will contain a value segment specifying the cluster number to which each input vector was assigned. If the -map flag is also specified, then a map segment will also be generated. The final cluster centers will be stored row by row in the map. The values in the value segment can be interpreted as "pointing" to a particular row in the map where the associated cluster for that input vector can be found.

If the -spectrum flag is specified, then the -o1 output object will contain a special map segment (regardless of the -map flag) with additional information required for use with the *spectrum* program in the most general sense. Here, not only the cluster centers are stored, but so are the number of vectors associated with each cluster and the packed upper triangle of the covariance matrix for each cluster. See the *spectrum* manual for additional information on how this data is used and the additional capabilities that become available when the extra data supplied by the -spectrum flag is present.

The -o2 optional argument will generate an output data object containing the cluster centers (mean vectors), stored row by row in the value segment. The dimensions of the value segment will be WxHx1x1x1 where W is the number of elements in each mean vector and W is the number of clusters.

The -o3 optional argument will generate an output data object containing the cluster variances, stored row by row in the value segment. The dimensions of the value segment will be WxHx1x1x1 where W is the number of elements in each vector of variances and W is the number of clusters.

The -o4 optional argument will generate an output data object containing the cluster membership counts, stored row by row in the value segment. The dimensions of the value segment will be 1xHx1x1x1 where H is the number of clusters. The membership counts simply state the number of vectors that were present in the input object that were assigned to each of the final cluster centers.

The statistics file (-o5) contains statistics obtained during the execution of *kkmeans*. This file includes the following information:

Total Number of K-means Iterations
Total Number of Clusters
Number of Vectors Per Cluster
Cluster Center Values
Cluster Center Variance Values
Trace of Covariance Matrix

Results obtained by the K-means algorithm can be influenced by the number and choice of initial cluster centers and the geometrical properties of the data.

For the -o2, and -o3, output objects, the data will be stored as type KDOUBLE. For the -o4 output object, the data will be stored as type KINT. For the -o1 output object, the value data will be stored as type KSHORT and all map data as type KDOUBLE.

kkmeans was converted from the K1.5 vkmeans program, which was written by Tom Sauer and Charlie Gage, with assistance and ideas from Dr. Don Hush, University of New Mexico, Dept. of EECE. Significant modifications were made to the algorithm by Scott Wilson during conversion to K2.

Examples

kkmeans -i1 image1 -n 10000 -k 6 -o1 image2 -o2 image3

this will apply the K-means clustering algorithm to image1 using the first K vectors as cluster centers. The number of iterations selected is 10000, and the number of clusters selected is 6. Image2 will contain a map linking each input vector to it's respective cluster center, while image3 will contain the actual cluster centers.

kkmeans -i1 image1 -i2 file1 -k 8 -o1 image2 -o2 image3 -o5 file2 -map

this will apply the K-means clustering algorithm to image1 using the cluster centers specified in file1. The -k option specifies 8 cluster centers. An ASCII file containing the K-means statistics (file2) is created. The other output objects are as specified above, except that image2 will also have a map attached containing the cluster centers.

kkmeans -i1 object1 -i2 object2 -k 8 -o1 object3 -spectrum

this will apply the K-means clustering algorithm to object1 using as input object2 to specify the initial cluster centers. The -k option specifies 8 cluster centers. Object3 will contain not only the mapping from vectors to clusters (in the value segment), but an extended map segment containing the cluster centers, counts, and covariance matrices. This object can be automatically classified using the Auto-Classify utilities in *spectrum*.

See Also

spectrum(1)

Restrictions

kkmeans will not operate on any form of COMPLEX data. Mask data is currently ignored. If map data is present, then the value data is pulled through the map prior to application of the K-means algorithm.

A maximum of 32767 clusters can be requested due to the use of the KSHORT output data type. If more clusters than this are desired, then the code can be easily modified to change the output data type to KINT.

References

The K-means algorithm is also called out as the Basic Isodata algorithm in R. Duda and P. Hart, **Pattern Classification and Scene Analysis**, Wiley, N.Y., 1973, p. 201. ISBN 0-471-22361-1. This is a dated, by very useful reference.

Command Line Arguments

```
Usage for kkmeans: Perform K-Means Clustering
% kkmeans
  -i1         (infile)  input data object
  -o1         (outfile) cluster number output object

  [-i2]       (infile)  cluster center input object
              (default = {none})
  [-map]      (flag)    generate output map
  [-spectrum] (flag)    SPECTRUM compatable map segment
  [-n]        (integer) max number of iterations
              (0 <= n <= 100000, default = 50000)
  [-k]        (integer) number of clusters
              (k > 0, default = 2)
  [-o2]       (outfile) cluster center output object
              (default = {none})
  [-o3]       (outfile) cluster variance output object
              (default = {none})
  [-o4]       (outfile) Cluster membership count output
              (default = {none})
  [-o5]       (outfile) K-means statistics output (ASCII)
              (default = {none})
```

C.93. kldexp — *Output = Input 1 * 2**(Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*ldexp*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *ldexp* operator multiplies each point in **Input 1** with the result of 2 taken to the power of either **Input 2** or the **Constant** value, whichever is specified by the user.

Executing *ldexp* runs the program *karith2* with the -ldexp flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real.

Command Line Arguments

```
Usage for kldexp: Output = Input 1 * 2**(Input 2 or Constant)
% kldexp
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real]  (double) Real constant value
             (unbounded, default = 0)
```

C.94. kle — *IF Input 1 is Less or Equal to (Input 2 or Constant), Output = TRUE*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	<=	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The LESS or EQUAL comparison operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value, which ever is specified by the user. If the value from **Input 1** is less than or equal to the value from **Input 2** or the **Constant**, the corresponding point in **Output** will be assigned the value specified by **TRUE Value**. If the value from **Input 1** is not less than or equal to the value against which it is being compared, the corresponding point in **Output** will be the value specified by **FALSE Value**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing the Less Than or Equal operator runs the program *kcompare* with the -le flag.

See Also

DATAMANIP::kcompare

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Command Line Arguments

```
Usage for kle: IF Input 1 is Less or Equal to (Input 2 or Constant), Output = TRUE
% kle
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) Second input data object
            (default = {none})
    [-real] (double) Constant value
            (real >= 0.0, default = 0)

  [-tval]    (double)  Value assigned to output if evaluation is TRUE
             (unbounded, default = 1)
  [-fval]    (double)  Value assigned to output if evaluation is FALSE
             (unbounded, default = 0)
```

C.95. klinearop — *Performs linear operations (convolution/correlation)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Linear Transforms*
Operator:	*Linear Operator*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

klinearop performs both convolution and correlation operations as defined by the operator flag -c between an object/sub-object as **Input #1** and a kernel as **Input #2**. Default values are: convolution (flag -c is false); the entire object in **Input #1**; and a kernel with "sweet spot" or origin determined by the KPDS_KERNEL_ORIGIN attribute from the kernel. *klinearop* allows the user to define the origin or "sweep spot" for **Input #2** or kernel. The origin of the kernel can be modified by an offset that points to any cell inside **Input #2** or kernel. When modifying the origin of the kernel, the user must select (enable the offset fields) and enter the new offset. When the offset fields are enabled, the starting origin corresponds to (0,0,0,0,0). Thus, only positive offset values are allowed. In addition, the dimensions in which *klinearop* will perform an operation and the available dimensions in the kernel are independent, what means that the user can perform a convolution or correlation in any dimension using any kernel. For instance, the user can perform a filtering operation in the width dimension of an object using a kernel defined in the height dimension. However, in the latter case the kernel origin defined by the user must be consistent with the kernel attribute size.

Data Type

The *klinearop* accepts any kind of data type for **Input #1** and **Input #2**. If the option *upcast* is set to "No", internally *klinearop* chooses the best data type before processing, which is the highest order of the two source objects, and changes the destination object data type (output) accordingly. The data is processed using one of the following data types: unsigned byte, long, double or double complex. If the option *upcast* is set to "Yes", internally *klinearop* changes the data type before processing to double (if the object data type of **Input #1** and **Input #2** is neither complex nor double complex) or double complex (if the object data type of either object **Input #1** or **Input #2** is complex) and casts the destination object data type (output) accordingly. Data is not cast to a lower type for processing.

Map Data

If either **Input #1** and/or **Input #2** contains map data as well as value data, then *klinearop* first maps the data before performing the operation. The destination object is always unmapped.

Mask, Time and Location Segments

If either source object contains a mask, time or location segment, then an error message is printed out. Mask, time and location segments have to be removed before processing.

Data Size

The object for **Input #2** must be of the same size or less than object in **Input #1** otherwise *klinearop* generates and error message indicating the first dimension where the error occurred. When processing sub-objects of the input object **Input #1** the routine *klinearop* do not touches the region extern to the sub-object. In addition, the sub-object, after processed, will be surrounded by a transition region. In other words, for sub-object processing the output object is equal to the input object with the processed sub-object surrounded by a transition region inserted. The values of the points in the transition region are defined by the user through the pad value entry for linear operation type and are equal to zero for the circular operation type.

Examples

Convolution between a 2D source object, input_image, and a 2D kernel object, kernel_image, with offset (0,0), pad value 0 for the real part and 0 for the imaginary part, truncated output (output_image) is:

```
klinearop -i1 input_image -i2 kernel_image -w -h
-o output_image -flip 0 -size 0 -l -w0 0 -h0 0
-real 0 -imag 0
```

Restrictions

The routine **klinearop** will generate an error if **Input #1** and **Input #2** are interchanged.

References

1) FUNDAMENTALS OF IMAGE PROCESSING by A. K. Jain, Prentice Hall, 1989.
2) DIGITAL IMAGE PROCESSING by Rafael C. Gonzalez and Richard E. Woods, Addison Wesley, 1992.
3) DIGITAL IMAGE PROCESSING, 2nd. Ed., by William K. Pratt, Wiley, 1991.
4) MULTIDIMENSIONAL DIGITAL SIGNAL PROCESSING, by Dan E. Dudgeon and Russell M. Mersereau, Prentice Hall, 1984.
5) DISCRETE-TIME SIGNAL PROCESSING, by Alan V. Oppenheim and Ronald W. Schafer, Prentice Hall, 1989.
6) TWO-DIMENSIONAL SIGNAL AND IMAGE PROCESSING, by Jae S. Lim, Prentice Hall, 1990.

Command Line Arguments

```
Usage for klinearop: Performs linear operations (convolution/correlation)
% klinearop
  -i1        (infile)  Input object filename
  -i2        (infile)  Input kernel filename
  -o         (outfile) output object filename

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:

    [Optional Mutually Inclusive Group - Specify all or none of the 4 options:]
      [-l]    (flag)    If flag is set, pad object
      [-real] (double)  Real part of pad constant
              (unbounded, default = 0)
      [-imag] (double)  Imaginary part of pad constant
              (unbounded, default = 0)
      [-size] (boolean) Determine size of output (def: truncate)
              (default = false)
    [-c] (flag) If flag is set, do not pad object

  Required Loose Group - Specify at least one of the 5 options:
    [-w] (flag) Perform operation in the width direction.
    [-h] (flag) Perform operation in the height direction.
    [-d] (flag) Perform operation in the depth direction
    [-t] (flag) Perform operation in the time direction.
    [-e] (flag) Perform operation in the elements direction.

  [-flip]    (boolean) Kernel reflection (def: convolution)
             (default = false)
  [-upcast]  (flag)    Upcast data type for processing.
  [-w0]      (integer) Width location of kernel origin
             (w0 >= 0, default = 0)
  [-h0]      (integer) Height location of kernel origin
             (h0 >= 0, default = 0)
  [-d0]      (integer) Depth location of kernel origin
             (d0 >= 0, default = 0)
  [-t0]      (integer) Time location of kernel origin
             (t0 >= 0, default = 0)
  [-e0]      (integer) Element location of kernel origin
             (e0 >= 0, default = 0)
  [-wp]      (integer) Sub-object width position.
             (unbounded, default = 0)
  [-ws]      (integer) Sub-object width size.
             (ws >= 0, default = 0)
  [-hp]      (integer) Sub-object height position.
             (unbounded, default = 0)
```

```
[-hs]       (integer) Sub-object height size.
            (hs >= 0, default = 0)
[-dp]       (integer) Sub-object depth position.
            (unbounded, default = 0)
[-ds]       (integer) Sub-object depth size.
            (ds >= 0, default = 0)
[-tp]       (integer) Sub-object time position.
            (unbounded, default = 0)
[-ts]       (integer) Sub-object time size.
            (ts >= 0, default = 0)
[-ep]       (integer) Sub-object elements position.
            (unbounded, default = 0)
[-es]       (integer) Sub-object elements size.
            (es >= 0, default = 0)
```

C.96. klocxform — *Rotate, Translate, and Scale Location Data*

Object Information

Category:	*Data Manip*	Subcategory:	*Location Operators*
Operator:	*Transform Location*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

This routine performs a homogeneous coordinate transformation on the location data in the source object, and saves the transformed location data into the destination object.

The source object must have location data, and the destination object must not have location data for this routine to function properly. This routine will create location data of the proper grid type in the destination.

The coordinate transformation can be specfied by nine parameters which specify an x-y-z rotation, an x-y-z translation, and an x-y-z scaling. The rotations are all performed with respect to the (0,0,0) origin. A scaling matrix, a translation matrix, and three rotation matrices are constructed from these specified parameters and composited into a single transformation matrix in the following order :

[1] Scaling
[2] X Rotation
[3] Y Rotation
[4] Z Rotation
[5] Translation

A value of 1.0 should be used for any scaling parameters where no scaling is desired. A value of 0.0 should be used for any rotation parameters where no rotation is desired or for any translation parameters where no translation is desired.

The location points are interpreted to be in three-space for the transformations. Explicit location data located in dimensions higher than three will be lost during the transformation. The transformation will be performed using the datatype KFLOAT.

In general, the grid type of the source object will be preserved in the destination object. For example, uniform location data, when transformed and stored in the destination object, will still be uniform location data. The exception to this rule is rectilinear data, which must be converted to curvilinear data in the event of a rotation. In this case the destination location data will be stored as explicit curvilinear data points.

Reference : Foley & VanDam "Computer Graphics : Principles and Practice", Second Edition, 1990, Chapter 5.

Command Line Arguments

```
Usage for klocxform: Rotate, Translate, and Scale Location Data
% klocxform
  -i         (infile)  the input object to be processed
  -o         (outfile) the output object

  [-xrot]    (float)   rotate around x axis
             (-360 <= xrot <= 360, default = 0)
  [-yrot]    (float)   rotate around y axis
             (-360 <= yrot <= 360, default = 0)
  [-zrot]    (float)   rotate around z axis
             (-360 <= zrot <= 360, default = 0)
  [-fx]      (flag)    flip x coordinates
  [-xtrans]  (float)   translate in x direction
             (unbounded, default = 0)
  [-fy]      (flag)    flip y coordinates
  [-ytrans]  (float)   translate in y direction
             (unbounded, default = 0)
  [-fz]      (flag)    flip z coordinates
  [-ztrans]  (float)   translate in z direction
             (unbounded, default = 0)

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-scale] (float) orthogonal scale factor
             (scale > 0.0, default = 1)

    [Optional Loose Group - Specify at least one of the 3 options:]
      [-xscale] (float) scale x coordinates
                (xscale > 0.0, default = 1)
      [-yscale] (float) scale y coordinates
                (yscale > 0.0, default = 1)
      [-zscale] (float) scale z coordinates
                (zscale > 0.0, default = 1)
```

C.97. klog — *Compute Logarithm*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*Logarithm*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

klog computes the logarithm of an object element by element. If the object has an associated mask, the operation will be gated, so that the logarithm is computed only on valid elements, indicated by non-zero mask values. A zero mask value will cause the value of the corresponding element in the input object to be transferred the output object.

Three standard-base logarithms are provided, and alternate bases can be specified using the -base option. *be* returns the natural logarithm of x, and *b2* and *b10* return the logarithm to base 2 and 10 respectively.

The -offset option can be set to TRUE if the logarithm is to operate on the input value plus 1 (log(x+1)). This improves accuracy for very small input values.

SPECIAL NOTE: The log() function has machine-specific characteristics! To fix this, an error message is returned if the argument is less than or equal to zero.

See Also

DATAMANIP::klogexp

Command Line Arguments

```
Usage for klog: Compute Logarithm
% klog
  -i          (infile)  input data object
  -o          (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 4 options:
    [-be]   (flag)   Compute logarithm using natural base 2.71828...
    [-b10]  (flag)   Compute logarithm using base 10
    [-b2]   (flag)   Compute logarithm using base 2
    [-base] (double) User Specified base
            (unbounded, default = 2)

  [-offset]  (boolean) Operate either on value [False] or value+1 [True]
             (default = true)
```

C.98. klogexp — *Compute Logarithm and Exponential (Antilogarithm)*

Object Information

Category:	*Arithmetic*	Subcategory:	*NonLinear Functions*
Operator:	*Log & Exponential*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

klogexp computes the log/antilog(exponential) of an object element by element. If the object has an associated mask, the operation will be gated, so that the operation is done only on valid elements, indicated by non-zero mask values. A zero mask value will cause the value of the corresponding element in the input object to be transferred the output object.

Three standard-bases are provided, and alternate bases can be specified using the -base option. *be* returns the natural logarithm of x, and *b2* and *b10* return the log/antilog(exponential) to base 2 and 10 respectively.

The -offset option can be set to TRUE if the logarithm is to operate on the input value plus 1 (log(x+1) or if the antilog(exponential) should return the output with a 1 subtracted from it (base**(x)-1). This improves accuracy for very small input values.

SPECIAL NOTE: The log/antilog(exponential) function is not defined for base values less than or equal to zero. An error is returned if the base value is less than or equal to zero.

Restrictions

This program has not yet been updated to completely support the polymorphic data model.

Command Line Arguments

```
Usage for klogexp: Compute Logarithm and Exponential (Antilogarithm)
% klogexp
  -i          (infile)  input data object
  -o          (outfile) resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 4 options:
    [-be]    (flag)   Compute using natural base 2.71828...
    [-b10]   (flag)   Compute using base 10
    [-b2]    (flag)   Compute using base 2
    [-base]  (double) User Specified base
             (unbounded, default = 2)

  [-inv]      (flag)    Compute inverse logarithm
  [-offset]   (boolean) Operate either on value [False] or value+1 for log and value -1 for antilog [True]
              (default = true)
```

C.99. klt — *IF Input 1 is Less Than (Input 2 or Constant), Output = TRUE*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	<	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The LESS THAN comparison operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value, which ever is specified by the user. If the value from **Input 1** is less than the value from **Input 2** or the **Constant**, the corresponding point in **Output** will be assigned the value specified by **TRUE Value**. If the value from **Input 1** is not less than the value against which it is being compared, the corresponding point in **Output** will be the value specified by

FALSE Value.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing the Less Than operator runs the program *kcompare* with the -lt flag.

See Also

DATAMANIP::kcompare

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Command Line Arguments

```
Usage for klt: IF Input 1 is Less Than (Input 2 or Constant), Output = TRUE
% klt
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real]  (double) Constant value
             (real >= 0.0, default = 0)

  [-tval]     (double)  Value assigned to output if evaluation is TRUE
              (unbounded, default = 1)
  [-fval]     (double)  Value assigned to output if evaluation is FALSE
              (unbounded, default = 0)
```

C.100. kmag — *Output is a Function of the Magnitude of the Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Magnitudes*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Magnitudes* operator returns the selected calculation of the magnitude of each data point in the input data object, **Input**. The user can specify one of six methods for calculating the magnitude output, where *Log* is the logarithm, base 10. (If the input data is not complex, the imaginary component is assumed to be zero.)

(mag)
Magnitude = sqrt(real * real + imaginary * imaginary)

(logmag1)
Log (Magnitude + 1)

(logmag)
Log (Magnitude)

(magsq)
Magnitude Squared

(logmagsq1)
Log (Magnitude Squared + 1)

(logmagsq)
Log (Magnitude Squared)

The output object data type will be the same as the input (regarding single or double floats).

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Magnitudes* runs the program *kcmplx2real* with the -om output option.

See Also

DATAMANIP::kcmplx2real, DATAMANIP::kcmplx, DATAMANIP::kreal2cmplx

Command Line Arguments

```
Usage for kmag: Output is a Function of the Magnitude of the Input
% kmag
  -i          (infile)  Input data object

  [-om]       (outfile) Output data object containing magnitude of input
              (default = {none})

  [Optional Mutually Exclusive Group - Specify one or none of the 6 options:]
    [-mag]       (flag) Magnitude = sqrt(real*real + imag*imag)
    [-magsq]     (flag) Magnitude Squared = real*real + imag*imag
    [-logmag1]   (flag) Log (Magnitude + 1) = log(sqrt(real*real + imag*imag)+1)
    [-logmagsq1] (flag) Log (Magnitude Squared + 1) = log(real*real+imag*imag+1)
    [-logmag]    (flag) Log (Magnitude) = log(sqrt(real*real + imag*imag))
    [-logmagsq]  (flag) Log (Magnitude):  log(real*real + imag*imag)
```

C.101. kmapdata — *Map Value Data Through the Map*

Object Information

Category:	*Data Manip*	Subcategory:	*Map Operators*
Operator:	*Map Data*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

If the source object contains map data as well as value data, then the value data will be mapped through the map (the value data type becomes that of the map) before the operation is performed. The destination object contains value data with no associated map.

If the data has a validity mask as well as a map, the mapped data is also masked. When mapping a value datum that has a mask value of zero (invalid), the mapping process will first look for an entry in the map that corresponds to the "invalid" value, if one does not exist (for example, the value is NaN), the mapped data is set to the pad value. In either case, the data remains masked. The mask is resized to correspond to the size of the mapped value data to properly mask it. For example, the value data may have the dimensions of
width = 100,
height = 50,
elements = depth = time = 1.

If the map dimensions are
map width = 3,
map height = whatever,
map element = map time = map spatial = 1,

the resulting value (and mask) size will be
width = 100,
height = 50,
elements = 3,
depth = time = 1.

All other data, such as location, are assumed to be unaffected, and are transferred to the output object.

Examples

kmapdata -i image:lizard -o lizard.mapped

This will map the value data from "image:mandril" through its map, resulting in an output object with of dimensionality Width = 512, Height = 480, Depth = 1, Time = 1, Elements = 3, and no map.

See Also

igamut(1)

Command Line Arguments

```
Usage for kmapdata: Map Value Data Through the Map
% kmapdata
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.102. kmaximum — *Returns Higher Value between Input 1 and (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Maximum*	Object Type:	*pane*

In Cantata: *Yes*

Description

This operator compares each data point in **Input 1** with either the value of the corresponding point in **Input 2** or the **Constant** value (which ever is specified by the user), and returns the maximum of the two values.

Executing *Clip Maximum* runs the program *karith2* with the -max flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real.

Command Line Arguments

```
Usage for kmaximum: Returns Higher Value between Input 1 and (Input 2 or Constant)
% kmaximum
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) Second input data object
            (default = {none})
    [-real] (double) Real constant value
            (unbounded, default = 255)
```

C.103. kmegaclip — *Clip the Range of Values in Data Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*MegaClip*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *MegaClip* operator (kmegaclip) performs clipping on the Input data object (i), limiting the range of values in the object. Upper and lower cutoff values can be specified as either single constant values (lc, uc), or sets of values passed in as data objects (ilc, iuc). It is not necessary to specify both the Upper Cutoff and the Lower Cutoff.

All data values in the Input object that are LESS THAN the Lower Cutoff are set to the Lower Cutoff in the resulting Output object, unless the alternate "Lower Cutoff Output Value" (fval) is explicitly provided. If this value is provided, clipped data are set to that value instead of the Lower Cutoff. Likewise, data values GREATER THAN the Upper Cutoff are set to the Upper Cutoff, unless the alternate "Upper Cutoff Output Value" is explicitly provided. All other data remain the same.
Exception: Any time that both the upper and lower cutoff values are provided, and the upper cutoff is LESS THAN the lower cutoff, then data values between the two cutoffs are always set equal to the "Lower Cutoff Output Value" (fval).

The following four diagrams illustrate how the data is clipped for different combinations of upper and lower cutoffs. The plots are example data histograms. Clipping output assignment rules are given following the diagrams.

Lower Cutoff Only:

```
     lower cutoff result:       |<-F->|<----T---->|
                                |     | :     .   |
                                |    .:.:.  .:::  |
                                |:. .:::::::.:::: |
                                |:::::::::::::::::.|
                                ------|-----------
                                      lower cutoff (lc)

        if value <  lc  cutoff result = False
        if value >= lc  cutoff result = True
```

Upper Cutoff Only:

```
        upper cutoff result:    |<----T---->|<-F->|
                                |       :   | .   |
                                |    .:.:.  .:::  |
                                |:. .:::::::.:::: |
                                |:::::::::::::::::.|
                                ------------|------
                                            upper cutoff (uc)

        if value >  uc  cutoff result = False
        if value <= uc  cutoff result = True
```

Upper Cutoff is Greater than Lower Cutoff:

When both upper and lower cutoff values are specified, and the upper cutoff is GREATER THAN the lower cutoff, the combined cutoff result is the logical AND of the individual results.

```
lower cutoff result:    |<-F->|<----T---->|
upper cutoff result:    |<----T---->|<-F->|

combined cutoff result: |<-F->|<-T->|<-F->|
                        |     | :   | .   |
                        |    .:.:.  .:::  |
                        |:. .::::::.::::: |
                        |::::::::::::::::.|
                        ------|-----|------
                             lc    uc

if value <  lc OR  value >  uc  cutoff result = False
if value >= lc AND value <= uc  cutoff result = True
```

Upper Cutoff is Less than Lower Cutoff:

When both upper and lower cutoff values are specified, but the upper cutoff is LESS THAN the lower cutoff, the combined cutoff result is the logical OR of the individual results. (Specifying the lower cutoff to be greater than the upper cutoff inverts the operation.) In this case, the false value assigned to the output will always be fval.

```
upper cutoff result:    |<-T->|<----F---->|
lower cutoff result:    |<----F---->|<-T->|

combined cutoff result: |<-T->|<-F->|<-T->|
                        |     | :   | .   |
                        |    .:.:.  .:::  |
                        |:. .::::::.::::: |
                        |::::::::::::::::.|
                        ------|-----|------
                             uc    lc

if value <  lc AND value >  uc  cutoff result = False
if value >= lc OR  value <= uc  cutoff result = True
```

Clipping results are assigned according to the following rules. ("Lower result" and "upper result" are either the clipping values or fval and tval - see discussion above.)

1. When lower cutoff < upper cutoff

   If the data value is less than the lower cutoff, set the output value to "lower result".

   If the data value is greater than the upper cutoff, set the output value to "upper result".

   Otherwise, the data value remains the same.

2. When upper cutoff < lower cutoff

   If the data value is less than the upper cutoff and it is greater than the lower cutoff, set output value to the "lower result".

   Otherwise, the data value remains the same.

When upper and lower cutoff values are equal, the only data values that are evaluated as TRUE are those that are equal to the cutoff values.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Objects of Different Sizes

If, in addition to the input data object (i), upper or lower cutoff objects (ilc, iuc) are supplied, the source objects can be of different sizes. If the sizes are different, the size of the destination object's data will be the the same as that of the input data object. When resizing the cutoff objects, the data is zero padded, or truncated to the size of the input object. Alignment between the input files will always be the upper-left hand corner, position (0,0,0,0,0).

Map Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Multiple Input Objects

include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Multiple Input Objects

include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

Executing *MegaClip* runs the program *kclip* with the -thresh option set to FALSE (-thresh 0).

See Also

DATAMANIP::kclipabove, DATAMANIP::kclipbelow, DATAMANIP::kclipin, DATAMANIP::kclipout, DATAMANIP::kclip

Command Line Arguments

```
Usage for kmegaclip: Clip the Range of Values in Data Object
% kmegaclip
  -i         (infile)  Input file
  -o         (outfile) output file

  Required Loose Group - Specify at least one of the 2 options:

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-lc]  (double) Constant lower cutoff value
             (unbounded, default = 64)
      [-ilc] (infile) File containing lower cutoff values
             (default = {none})

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-uc]  (double) Constant upper cutoff value
             (unbounded, default = 128)
      [-iuc] (infile) File containing upper cutoff values
             (default = {none})

  [-fval]    (double)  Output value if data is LESS THAN lower cutoff
             (unbounded, default = 0)
  [-tval]    (double)  Output value if data is GREATER THAN upper cutoff
             (unbounded, default = 1)
```

C.104. kmegathresh — *Threshold the Range of Data Values in Data Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*MegaThresh*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *MegaThresh* operator (kemgathresh) generates a two-level Output data object (o) by thresholding the values in the Input data object (i). Upper and lower cutoff values can be specified as either single constant values (lc, uc), or sets of values passed in as data objects (ilc, iuc). It is not necessary to specify both of the upper and lower cutoff values. If only the Lower Cutoff is specified, all values at and above it are in the threshold range. If only the Upper Cutoff is specified, all values at and below it are in the threshold range. Output levels can be set using the "True" and "False" Value options (tval, fval). The True Value (tval) is assigned to values that are within the threshold range, and all values outside of the threshold range are set to the False Value. The data type of the Output object will be the same as the Input object data type.

The following four diagrams illustrate how the given operation is evaluated for different combinations of upper and lower cutoffs. The plots are example data histograms. Threshold output assignment rules are given following the diagrams.

Lower Cutoff Only:

```
lower cutoff result:       |<-F->|<----T---->|
                           |     | :     .   |
                           |    .:.:.  .:::  |
                           |:. .:::::::.:::: |
                           |::::::::::::::::.|
                           ------|------------
                                 lower cutoff (lc)
   if value <  lc  cutoff result = False
   if value >= lc  cutoff result = True
```

Upper Cutoff Only:

```
   upper cutoff result:    |<----T---->|<-F->|
                           |       :   | .   |
                           |    .:.:.  .:::  |
                           |:. .:::::::.:::: |
                           |::::::::::::::::.|
                           ------------|------
                                       upper cutoff (uc)

   if value >  uc  cutoff result = False
   if value <= uc  cutoff result = True
```

Upper Cutoff is Greater than Lower Cutoff:

When both upper and lower cutoff values are specified, and the upper cutoff is GREATER THAN the lower cutoff, the combined cutoff result is the logical AND of the individual results.

```
lower cutoff result:    |<-F->|<----T---->|
upper cutoff result:    |<----T---->|<-F->|

combined cutoff result: |<-F->|<-T->|<-F->|
                        |     | :   | .   |
                        |    .:.:.  .:::  |
                        |:. .::::::.::::: |
                        |::::::::::::::::.|
                        ------|-----|------
                              lc    uc

if value <  lc OR  value >  uc  cutoff result = False
if value >= lc AND value <= uc  cutoff result = True
```

Upper Cutoff is Less than Lower Cutoff:

When both upper and lower cutoff values are specified, but the upper cutoff is LESS THAN the lower cutoff, the combined cutoff result is the logical OR of the individual results. (Specifying the lower cutoff to be greater than the upper cutoff inverts the operation.)

```
upper cutoff result:    |<-T->|<----F---->|
lower cutoff result:    |<----F---->|<-T->|

combined cutoff result: |<-T->|<-F->|<-T->|
                        |     | :   | .   |
                        |    .:.:.  .:::  |
                        |:. .::::::.::::: |
                        |::::::::::::::::.|
                        ------|-----|------
                              uc    lc

if value <  lc AND value >  uc  cutoff result = False
if value >= lc OR  value <= uc  cutoff result = True
```

Thresholding results are assigned according to the following rules.

1. When lower cutoff < upper cutoff

   If the data value is less than the lower cutoff, or it is greater than the upper cutoff, set the output value to fval.

Otherwise, set the output value to tval.

2. When upper cutoff < lower cutoff

   If the data value is less than the lower cutoff and it is greater than the upper cutoff, set output value to fval.

   Otherwise, set the output value to tval.

When upper and lower cutoff values are equal, the only data values that are evaluated as TRUE are those that are equal to the cutoff values.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Objects of Different Sizes

If, in addition to the input data object (i), upper or lower cutoff objects (ilc, iuc) are supplied, the source objects can be of different sizes. If the sizes are different, the size of the destination object's data will be the the same as that of the input data object. When resizing the cutoff objects, the data is zero padded, or truncated to the size of the input object. Alignment between the input files will always be the upper-left hand corner, position (0,0,0,0,0).

Map Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Multiple Input Objects

include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Multiple Input Objects

include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Explicit Location and Time Data - Single Input

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

Executing *MegaThresh (kmegathresh)* runs the program *kclip* with the -thresh option set to true (-thresh 1).

See Also

DATAMANIP::kthreshabove, DATAMANIP::kthreshbelow, DATAMANIP::kthreshin, DATAMANIP::kthreshout, DATAMANIP::kclip

Command Line Arguments

```
Usage for kmegathresh: Threshold the Range of Data Values in Data Object
% kmegathresh
  -i         (infile)  Input file
  -o         (outfile) output file

  Required Loose Group - Specify at least one of the 2 options:

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-lc]  (double) Constant lower cutoff value
             (unbounded, default = 64)
      [-ilc] (infile) File containing lower cutoff values
             (default = {none})

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-uc]  (double) Constant upper cutoff value
             (unbounded, default = 128)
      [-iuc] (infile) File containing upper cutoff values
             (default = {none})

  [-fval]    (double)  Output value if evaluation is FALSE
             (unbounded, default = 0)
  [-tval]    (double)  Output value if evaluation is TRUE
             (unbounded, default = 1)
```

C.105. kminimum — *Returns Lower Value between Input 1 and (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Minimum*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

This operator compares each data point in **Input 1** with either the value of the corresponding point in **Input 2** or the **Constant** value (which ever is specified by the user), and returns the minimum of the two values.

Executing *Clip Minimum* runs the program *karith2* with the -min flag.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real.

Command Line Arguments

```
Usage for kminimum: Returns Lower Value between Input 1 and (Input 2 or Constant)
% kminimum
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real] (double) Real constant value
             (unbounded, default = 0)
```

C.106. kmod — *Output = Remainder of [Input 1 / (Input 2 or Constant)]*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Modulus*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Modulus* operator returns the remainder of the result produced by dividing each point in **Input 1** by either **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Modulus* runs the program *karith2* with the -mod flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (1.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Restrictions

Operations on complex data are not supported at this time. If the input object is complex, only the real component of the data is processed, and the output object is real.

Command Line Arguments

```
Usage for kmod: Output = Remainder of [Input 1 / (Input 2 or Constant)]
% kmod
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]    (infile) Second input data object
             (default = {none})
    [-real]  (double) Real constant value
             (unbounded, default = 1)
```

C.107. kmsquish — *Compress Map to One Column by Means of Average, RMS, or MAX*

Object Information

Category:	*Data Manip*	Subcategory:	*Map Operators*
Operator:	*Squish Map*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kmsquish compresses a map down to one column, effectively converting the map to a "greyscale" map. An example of this operation is encountered when trying to print a pseudocolor RGB image on a B/W laser printer: the RGB color information must be somehow converted to greyscale in order for the printer to know how to dither the image during printing.

kmsquish is very happy to operate on simple 3-column maps, such as would be encountered when processing a color image in any of the standard coordinate systems (ntscRGB, etc). However, kmsquish is not limited to 3 column maps - it will operate on maps with any number of cloumns.

The conversion from multiple columns to a single column is done by computing a grey level for each row of the map, one at a time. For t=1, the grey level for a row is the average of the columns in the row. For t=2, the grey level is the RMS value of the columns in the row. For t=3, the grey level is the maximum value found in the colums of the row. Clearly, these schemes are not optimal in any sense, but one of them will usually give an acceptable result.

Ordinarily (map=0), the map is the only part of the input object that is modified. The value data is not touched; it still points to various rows in the map just as before the compression. If map=1, the value data is pulled thru the map and the map segment itself is removed. If mask data is present, then this operation proceeds as described in the man page for kmapdata(1).

Examples

Squishing and mapping an image that is to be printed.

kmsquish -i ball.xv.color -o ball.ready.to.print

here, -m pand -t do not need to be specified since the defaults are set correctly.

See Also

igamut(1), kmapdata(1)

Restrictions

Complex data is not supported.

Command Line Arguments

```
Usage for kmsquish: Compress Map to One Column by Means of Average, RMS, or MAX
% kmsquish
  -i         (infile)  First Input data object
  -o         (outfile) Resulting output data object

  [-map]     (boolean) map data after squishing map (1) or leave map with only one column (0)
             (default = true)
  [-t]       (integer) squishing scheme (1)average, (2)RMS, (3)Maximum
          1  (Average of map columns),
          2  (RMS value of map columns),
          or 3  (Maximum value of mape columns)
[default: 1]
```

C.108. kmul — *Output = Input 1 * (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Multiply*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Multiply* operator multiplies each point in **Input 1** with either **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Multiply* runs the program *karith2* with the -mul flag.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (1.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Command Line Arguments

```
Usage for kmul: Output = Input 1 * (Input 2 or Constant)
% kmul
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) Second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 1)
      [-imag] (double) Imaginary constant value
              (unbounded, default = 0)
```

C.109. kne — *IF Input 1 is Not Equal to (Input 2 or Constant), Output = TRUE*

Object Information

Category:	*Arithmetic*	Subcategory:	*Comparison Operators*
Operator:	*!=*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The NOT EQUAL comparison operator compares each point in **Input 1** either with the corresponding point in **Input 2** or with the **Constant** value, which ever is specified by the user. If the two values are not equal, the corresponding point in **Output** will be assigned the value specified by **TRUE Value**. It the values being compared are equal, the corresponding point in **Output** will be the value specified by **FALSE Value**.

Data Type - Single Input

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data

The comparison operations have not been written to be fully polymorphic yet. They does not check for map data, and will therefore always operate on the value data, even if a map exists. This will most likely corrupt indexing into the map.

In the case of a single input, it is recommended to use the *Copy to Value* (kcptoval) segment operator to temporarily move the map data into the value segment, run the operator on the data, then move it back to the map with the *Copy from Value* (kcpfromval) operator.

When operating with two input objects, where at least one of the objects contains map data, data should be mapped prior to running the comparison operator. The *Map Data* operator (kmapdata) can be used to perform the mapping operation.

Validity Mask - Single Input

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Validity Mask - Two Input Objects

Masking has not been implemented yet for the comparison operators. Therefore, only the mask from the first input object will be transferred to the output.

Input Objects of Different Sizes

include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The value used to pad the data when the input files are not the same size is zero.

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
The comparison operations have not been extended to understand location and time data. Therefore, only location and time data present in the first input object will be transferred to the output.

Executing the Not Equal operator runs the program *kcompare* with the -ne flag.

See Also

DATAMANIP::kcompare

Restrictions

This routine is still under development, and has not been modified to fully support the polymorphic data model. See paragraphs above for discussions concerning the polymorphic data segments.

Command Line Arguments

```
Usage for kne: IF Input 1 is Not Equal to (Input 2 or Constant), Output = TRUE
% kne
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2]   (infile) Second input data object
            (default = {none})
    [-real] (double) Constant value
            (real >= 0.0, default = 0)

  [-tval]    (double)  Value assigned to output if evaluation is TRUE
             (unbounded, default = 1)
  [-fval]    (double)  Value assigned to output if evaluation is FALSE
             (unbounded, default = 0)
```

C.110. kneg — *Output = 0.0 - Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Negative*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

Output = 0 - Input The *Negative* operator returns the negative (0 - value) of each data point in the input data object, **Input**. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Negative* runs the program *karith1* with the -neg flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for kneg: Output = 0.0 - Input
% kneg
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.111. knoise — *Introduce Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Introduce Noise*	Object Type:	*kroutine*
In Cantata:	*No*		

Description

knoise introduces noise of the specified type in the input data object (-i). The noise can either replace the existing value data or it can be added to the existing value data by specifying the appropriate flag (-replace or -add). *knoise* can introduce the following kinds of noise to the input data object: Gaussian, Rayleigh, Uniform, Poisson or Exponential. The appropriate parameters must be specified for the kind of noise being introduced in the system e.g. Mean & Variance for the Gaussian distribution and just the Variance for the Exponential distribution.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Command Line Arguments

```
Usage for knoise: Introduce Noise in Input Object
% knoise
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-replace] (flag) Replace existing values with noise
    [-add]     (flag) Add noise to the specified segment

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:

    [Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
      [-gauss] (flag)   Generate Gaussian noise
      [-gmean] (double) Mean for Gaussian noise
               (unbounded, default = 0)
      [-gvar]  (double) Variance for Gaussian noise
               (gvar > 0.0, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-ray]  (flag)   Generate Rayleigh Noise
      [-rvar] (double) Variance for Rayleigh noise
              (rvar > 0.0, default = 1)

    [Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
      [-unif] (flag)   Generate uniformly distributed noise
      [-umax] (double) Maximum value for uniform distribution
              (unbounded, default = 1)
      [-umin] (double) Minimum value for uniform noise
              (unbounded, default = 0)

    [Optional Mutually Inclusive Group - Specify all or none of the 3 options:]
      [-poiss] (flag)   Generate Poisson Noise
      [-ptime] (double) Amount of time
               (ptime > 0.0, default = 1)
      [-pvar]  (double) Variance for Poisson distribution
               (pvar > 0.0, default = 1)
```

```
[Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
  [-exp]  (flag)   Generate exponential noise
  [-evar] (double) Variance for exponential distribution
          (evar > 0.0, default = 1)
```

C.112. knormal — *Normalize Data Regions Using Minimum & Maximum of Each Region*

Object Information

Category:	*Data Manip*	Subcategory:	*Data Conversion*
Operator:	*Normalize*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Normalize Data* operator (knormal) normalizes all data within each specified region so that the minimum data value in that region will be equal to the Lower Value (lval) provided by the user, and the maximum data value in the region will be equal to the Upper Value (uval). If Predetermined Data Minimum (dmin) and Maximum (dmax) values are specified, these values will be used instead of scanning the data for minimum and maximum values. This eliminates an otherwise required pass through the data before the normalization operation.

Regions are defined by the settings of the Processing Unit options, which can be either the Whole Data Set (whole), or any combination of Width (w), Height (h), Depth (d), Time (t), and Elements (e). If none of these flags are supplied, then the default will be the whole data set.

If the source object contains value data and no map, normalization is performed on the value data. Likewise, if the source object contains map data and no value data, normalization is performed on the map. If the source object has both map and value data, certain region conditions must be met for the operation to succeed.

1. The first condition is that the normalization region must be defined by at least width and height.

2. If the map depth, time, and elements dimensions are all one, the normalization region must be the entire data set.

3. If the map depth, time, or elements dimensions are greater than 1, the corresponding dimension does not have to be part of the normalization region.

Output Data Type

The Output Object Data Type (type) can also be specified. If "Propagate Input Type" is selected, the output data type will be the same as the input. If a different data type is selected, data is cast to that type after the transformation is calculated. Warning - casting to an unsigned type might result in wrap-around.

All internal processing is performed in double.

Validity Mask

Currently, the validity mask is only transferred from the source to the destination object. The Masked Value Presentation attribute is used, so valid values can be substituted for invalid data (to explicitly set this attribute, use the *Set Data Attribute* (ksetdattr) operator in the Datamanip toolbox

As a future enhancement, invalid data will not be figured into the normalization calculations.

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

kconvert

Restrictions

Does not support complex input data types yet.

The validity mask is simply transferred from the input object to the output object. Invalid data will be included in the normalization calculations.

Command Line Arguments

```
Usage for knormal: Normalize Data Regions Using Minimum & Maximum of Each Region
% knormal
  -i          (infile)  Input data object
  -o          (outfile) resulting data object

  [-lval]     (double)  Lower normalization value
              (unbounded, default = 0)
  [-uval]     (double)  Upper normalization value
              (unbounded, default = 255)
  [-type]     (list)    Data type of Output
          0  "Propagate Input Type",
          1  "Bit",
          2  "Byte",
          3  "Unsigned Byte",
          4  "Short",
          5  "Unsigned Short",
          6  "Integer",
          7  "Unsigned Integer",
          8  "Long",
          9  "Unsigned Long",
          10  "Float",
          11  "Double",
          12  "Complex",
          13  "Double Complex"
          [default: 0 "Propagate Input Type"]

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
```

```
    [-whole] (flag) normalize whole data set at one time

    [Optional Loose Group - Specify at least one of the 5 options:]
      [-w] (flag) include width in normalization unit
      [-h] (flag) include height in normalization unit
      [-d] (flag) include depth in normalization unit
      [-t] (flag) include time in normalization unit
      [-e] (flag) include elements in normalization unit

  [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
    [-dmin] (double) If specified, this value used as data minimum
            (unbounded, default = 0)
    [-dmax] (double) If specified, this value used as data maximum
            (unbounded, default = 0)
```

C.113. korient — *Change Orientation of Data on Dimensions*

Object Information

Category:	*Data Manip*	Subcategory:	*Reorganize Data*
Operator:	*Reorient*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

korient allows data to be reorganized from its original orientation on the dimensions of WIDTH (W), HEIGHT (H), ELEMENTS (E), DEPTH (D), and TIME (T), to another orientation on these dimensions.

```
                  +--------------+                                     +--------------+
             E   /--------------/|                                E   /--------------/|
                /--------------/||                                   /--------------/||
        +--------------+------+|||                           +--------------+------+|||
    E  /--------------/|      ||||                       E  /--------------/|      ||||
      /--------------/||      ||||                         /--------------/||      ||||
     +--------------+|||      ||||                        +--------------+|||      ||||
     |              ||||      ||/  /                      |              ||||      ||/  /
H    |              ||||------+/  /                  H    |              ||||------+/  /
     |              ||||         D     .   .   .          |              ||||         D
     |              ||/         /                         |              ||/         /
     +--------------+/        /                           +--------------+/        /
           W                                                    W

     <------------------------------------ T -------------------------------------->
```

The new orientation is defined by setting the *OLD DIMENSIONS* to *NEW DIMENSIONS* flags. On the graphical user interface accessible via cantata, these flags are represented as a matrix of buttons, where each row of the matrix represents the set of dimensions that will be oriented to the new dimension. On the command line, these flags are specified by two letter combinations of w,h,d,t,e -- the first letter indicates the source, or old, dimension, the second indicates the destination, or new, dimension. For example,

korient -i infile -o outfile -hw

will cause the height dimension (along with the width dimension) to be reoriented along the width dimension. If the input file has width = 4, height = 2, as shown below, the output dimension will have width = 8, height = 1.

```
Input Data:     1 2 3 4
        5 6 7 8

Output Data: 1 2 3 4 5 6 7 8
```

To transpose the same data set, one would execute

korient -i infile -o outfile -wh -hw

```
Output Data:    1 5
        2 6
        3 7
        4 8
```

As is illustrated above, data existing on a set of dimensions can be reoriented to a single dimension. When this occurs, data will be arranged according to the following dimensional order: Width, Height, Elements, Depth, Time.

If defaults (korient -i infile -o outfile -ww, -hh, -ee, -dd, -tt) are used, no change in orientation will occur between the input and output data files.

Implementation issue: If data on a dimension is not being transferred to another dimension, first set that dimension for no transfer. For example, if no flags have been set to orient width data along another dimension, the -ww flag is set. Then build the set of dimensions that will be oriented along the new dimension by going vertical on the user interface matrix. For example, the two letter flags that end in w that have been specified on the command line indicate the dimensions that will be reoriented.

Map Data

If the source object has map data, the map data cannot have depth, time, or elements dimensions of greater than one. The map width and height dimensions can be any size.

Location & Time Data

If the source object has rectilinear or curvilinear location or time data, the orientation operator will fail. If the source object has uniform location data, orientation will continue, but the location data will not be modified to reflect the reorientation.

Map Data

If the object has mask data, it will be reoriented along with the value data.

Failure Modes

The orientation operator will fail if the source object does not contain value data, or if it contains rectilinear or curvilinear location or time data.

See Also
kaxis

Restrictions
This program has not yet been updated to completely support the polymorphic data model.

Command Line Arguments

```
Usage for korient: Change Orientation of Data on Dimensions
% korient
  -i         (infile)  input data object
  -o         (outfile) resulting data object

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:
    [-ww] (flag) orient WIDTH data to be on WIDTH dimension
    [-wh] (flag) orient WIDTH data to be on HEIGHT dimension
    [-we] (flag) orient WIDTH data to be on ELEMENTS dimension
    [-wd] (flag) orient WIDTH data to be on DEPTH dimension
    [-wt] (flag) orient WIDTH data to be on TIME dimension

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:
    [-hw] (flag) orient HEIGHT data to be on WIDTH dimension
    [-hh] (flag) orient HEIGHT data to be on HEIGHT dimension
    [-he] (flag) orient HEIGHT data to be on ELEMENTS dimension
    [-hd] (flag) orient HEIGHT data to be on DEPTH dimension
    [-ht] (flag) orient HEIGHT data to be on TIME dimension

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:
    [-ew] (flag) orient ELEMENT data to be on WIDTH dimension
    [-eh] (flag) orient ELEMENT data to be on HEIGHT dimension
    [-ee] (flag) orient ELEMENT data to be on ELEMENTS dimension
    [-ed] (flag) orient ELEMENT data to be on DEPTH dimension
    [-et] (flag) orient ELEMENT data to be on TIME dimension

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:
    [-dw] (flag) orient DEPTH data to be on WIDTH dimension
    [-dh] (flag) orient DEPTH data to be on HEIGHT dimension
    [-de] (flag) orient DEPTH data to be on ELEMENTS dimension
    [-dd] (flag) orient DEPTH data to be on DEPTH dimension
    [-dt] (flag) orient DEPTH data to be on TIME dimension

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:
    [-tw] (flag) orient TIME data to be on WIDTH dimension
    [-th] (flag) orient TIME data to be on HEIGHT dimension
    [-te] (flag) orient TIME data to be on ELEMENTS dimension
    [-td] (flag) orient TIME data to be on DEPTH dimension
    [-tt] (flag) orient TIME data to be on TIME dimension
```

C.114. kpad — *Pad Data with a Constant Value*

Object Information
Category: *Data Manip* Subcategory: *Size & Region Operators*

Operator:	*Pad*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

Pad Object (kpad) performs the padding of a data object using the pad values defined by the Real and Imaginary padding value options.

Size The output data object will be the size specified by the final padded size parameters, Padded Width, Padded Height, etc (wsize, hsize, dsize, tsize, esize).

Offset Values

The offset parameters (woff, hoff, doff, toff, eoff) define where the origin of the source object will lie with respect to the output object. Specifying an offset greater than zero can be used to generate a border around the original data. Negative offset values will result in truncation of data lying in the region (0-woff, 0-hoff, 0-doff, 0-toff, 0-eoff). For example, if an offset of (-1,-1,0,0,0) is specified, the object will effectively be shifted by -1 in the width and height dimensions, and that data truncated in the destination object. If the offset plus the original data size is greater than the specified destination size, truncation of the source data will occur. If the input object is larger than the final padded size specified, the output data object will be truncated to the specified size.

Pad Values

The Real Pad Value (real) specifies the pad value that will be assigned to the real component of a complex pair, and the Imaginary Pad Value (imag) specifies the imaginary component of the pair. If the input data object type is not complex, the imaginary pad value will be ignored. For input data objects that are not of type double, the specified pad values are cast to the same type as the input data object.

Mask Data & Validity Option

The validity mask corresponding to padded data will be zero if the user specifies that padded data added by this program be marked as invalid (valid FALSE). Likewise, if the user specifies that padded data is valid (valid TRUE), the mask corresponding to padded data will be non-zero. In the first case (invalid), if no mask exists in the source object, on will be created. In the valid case, no mask will be created. Validity mask data corresponding to the original source object data is transferred to the destination unaltered.

Location & Time Data

If curvilinear or rectilinear time or location data exist, it is unclear how to alter the dimensionality, so the pad operation will fail. If uniform location data exist padding occurs, but the location information is not altered in any way.

Map Data

The pad operator does not modify map data.

Failure Modes

The input object must contain value or mask data.

Examples

kpad -i input_file -o output_file- Pad Data with a Constant Value
Will create a zero padded output object, using the default size and zero offset.

kpad -i input_file -o output_file -wsize 512 -hsize 256 -real 200 -imag 100- Pad Data with a Constant Value
This padding operation will produce an output object with a width of 512 and height of 256. If the input data is complex, padding for the real component will be 200, and 100 for the imaginary component. If the data is not complex, the imaginary value is ignored.

Restrictions

kpad will fail if the source object contains curvilinear or rectilinear location or time data. If uniform time or location data exist, it is transferred, but not modified.

Command Line Arguments

```
Usage for kpad: Pad Data with a Constant Value
% kpad
  -i         (infile)  Input data file
  -o         (outfile) Resulting output file

  [-wsize]   (integer) Size of width dimension after padding
             (wsize > 0, default = 512)
  [-hsize]   (integer) Size of height dimension after padding
             (hsize > 0, default = 512)
  [-dsize]   (integer) Size of depth dimension after padding
             (dsize > 0, default = 1)
  [-tsize]   (integer) Size of time dimension after padding
             (tsize > 0, default = 1)
  [-esize]   (integer) Size of element dimension after padding
             (esize > 0, default = 1)
  [-woff]    (integer) Offset of original object in width dimension
             (unbounded, default = 0)
  [-hoff]    (integer) Offset of original object in height dimension
             (unbounded, default = 0)
  [-doff]    (integer) Offset of original object in depth dimension
             (unbounded, default = 0)
  [-toff]    (integer) Offset of original object in time dimension
             (unbounded, default = 0)
  [-eoff]    (integer) Offset of original object in elements dimension
             (unbounded, default = 0)
  [-real]    (double)  Real part of pad constant
             (unbounded, default = 0)
  [-imag]    (double)  Imaginary part of pad constant
             (unbounded, default = 0)
  [-valid]   (boolean) Identify data added by kpad as valid/invalid in mask
             (default = true)
```

C.115. kphase — *Output = Phase Component of Input [atan2(imaginary, real)]*

Object Information

Category: *Arithmetic* Subcategory: *Complex Operators*

Operator:	*Phase*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Phase* operator returns the phase of each complex data point in the input data object, **Input**. If the input data is not complex, the imaginary component is assumed to be zero.

The output object data type will be the same as the input (regarding single or double floats).

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Phase* runs the program *kcmplx2real* with the -op output option.

See Also

DATAMANIP::kcmplx2real, DATAMANIP::kcmplx, DATAMANIP::kreal2cmplx

Restrictions

The input object data type must be COMPLEX or DOUBLE COMPLEX.

This program has not yet been updated to completely support the polymorphic data model.

Command Line Arguments

```
Usage for kphase: Output = Phase Component of Input  [atan2(imaginary, real)]
% kphase
  -i         (infile)  input data object
  -op        (outfile) resulting output data object
```

C.116. kpoisson — *Introduce Poisson Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Poisson Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kpoisson introduces Poisson noise of the specified parameters in the input data object (-i). The noise can either replace the existing value data or it can be added to the existing value data by specifying the appropriate flag (-replace or -add). The time (-ptime) and the variance (-pvar) of the Poisson distribution must be specified.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::knoise, DATAMANIP::kexpon, DATAMANIP::kgauss, DATAMANIP::krayleigh, DATAMANIP::kuniform

Command Line Arguments

```
Usage for kpoisson: Introduce Poisson Noise in Input Object
% kpoisson
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-replace] (flag) Replace existing values with noise
    [-add]     (flag) Add noise to the specified segment

  [-ptime]   (double)  Mean for Poisson noise
             (ptime > 0.0, default = 1)
  [-pvar]    (double)  Variance for Poisson noise
             (pvar > 0.0, default = 1)
```

C.117. kpol2rect — *Convert (r, theta) to (real, imaginary) Coordinates*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Polar to Rect*	Object Type:	*pane*

In Cantata: *Yes*

Description

The *Polar to Rect.* operator converts data from the polar to the rectangular coordinate system for each complex data point in the input data object, **Input**. The angles component of the data is assumed to be measured in radians. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Polar to Rect.* runs the program *kcmplx* with the -p2r flag.

Segments that are not operated on
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

See Also

DATAMANIP::kcmplx

Command Line Arguments

```
Usage for kpol2rect: Convert (r, theta) to (real, imaginary) Coordinates
% kpol2rect
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.118. kpow — *Output = (Input 1) ** (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Power*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Power* operator raises each point in **Input 1** to the power specified either by the corresponding point in **Input 2** or the **Constant** value, which ever is specified by the user.

Executing *Power* runs the program *karith2* with the -pow flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (1.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2

Command Line Arguments

```
Usage for kpow: Output = (Input 1) ** (Input 2 or Constant)
% kpow
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) Second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 0)
```

```
[-imag] (double) Imaginary constant value
        (unbounded, default = 0)
```

C.119. kprdata — *Print Data in Formatted ASCII*

Object Information

Category:	*Input/Output*	Subcategory:	*Information*
Operator:	*Print Data*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kprdata prints the selected segments of data from the input object in ASCII format to a file or to the screen.

If the Print Attributes option (-att) is selected, all header and data information is printed before any data is printed. (These non-data lines begin with a #, which indicates that the line is a "comment".)

After the attribute lines, data fields selected by the user are formatted in the following order.

Implicit Indexing: Location Data: Time Data: Value Data: Mask Data.

This data can be followed by map data (if selected):

Map Indexing: Map Data.

This data can be followed by connectivity data.

When map data exists, selecting the "Map Value Data Through Map" option will cause the value data to be mapped through the map data before being presented.

The implicit indexing options, Print Implicit Indexing (-ind), and Print Map Indexing (-mind), if selected, will print the indices of the data. The index values index both the value data and explicit location and time data. For instance, in the following example, the index [0][0][0][1] points to the 3D location data [4][27][38], and the value of 104.5.

```
t d h w   Location  Value
0 0 0 1   4 27 38   104.5
```

The **Replacement for Masked Values** option (-replace) allows value data which has mask data associated with it, to be printed with the masked values replaced by the given string.

Complex data is printed as real/imaginary pairs (real, imaginary).

If data segments that do not exist in the input data object are selected, kprdata will return a warning message indicating that the segments do not exist. Information about other selected segments will be printed.

If the location grid is uniform or rectilinear and only the location data is being printed, then the location data will be printed in a manner consistent with the grid type. That is, a uniform grid implies that the location begin and end will be printed, and a rectilinear grid implies that the data for the width, height, and depth axes will be printed. When location data is printed with any other parts of the data model, it is presented as if it were curvilinear location data.

Examples

kprdata -i in_object -attr -val -ind
This command sequence will cause the attributes, followed by the value data, with indices, to be printed to stdout.

kprdata -i in_obj -map -mind -val -f out_file
This command sequence will print all of the value data, followed by the map data, with indices, to the file out_file.

Command Line Arguments

```
Usage for kprdata: Print Data in Formatted ASCII
% kprdata
  -i         (infile)  Input Data Object

  [-attr]    (flag)    Print Object Information & Attributes
  [-loc]     (flag)    Print Location Data
  [-time]    (flag)    Print Time Data
  [-val]     (flag)    Print Value Data
  [-mask]    (flag)    Print Mask Data
  [-ind]     (flag)    Print Implicit Indexing
  [-replace] (string)  Replacement String for Masked Values
             (default = *)
  [-f]       (outfile) Resulting Output File
             (default = {none})

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-mind] (flag) Print Implicit Map Indexing
      [-map]  (flag) Print Map Data
    [-mval] (flag) Map Value Data Through Map
```

C.120. kprval — *Print Data Value to Parser and/or Concatenate to File*

Object Information

Category:	*Data Manip*	Subcategory:	*Analysis & Information*
Operator:	*Print Value*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The **Print Value** operator (kprval) extracts a single data value from the Input data object (i). The location of the value is based upon the user indicated position within the specified data segment (Value, Map, Mask, Location, or Time). The extracted data value can be assigned to a Variable Name (var) via the parser. This is useful for variable assignment within *cantata*, and can be used in any valid expression. The extracted value can also be appended to an output file (o), which may be useful for purposes such as tracking data during processing. If neither the variable name nor the the output file is specified, the value will be printed to standard out (kstdout).

The user can select which component of the polymorphic data model the value will be extracted from. The components are Value Data (val), Map Data (map), Mask Data (mask), Time Data (time), and Location Data (loc). If Value Data is selected, and a map exists in the data object, the user can specify whether the data should be mapped before the value is extracted.

The position of the data point can be specified by the offset values Width (woff), Height (hoff), Depth (doff), Time (toff), and Elements (eoff). If a Time Data point is being extracted, only the Time offset needs to be specified. If Location Data is selected, the Width, Height, and Depth offsets can be specified, as well as a Dimension offset (dimoff). The default position is the first point in the data set, located at position 0.

If the input object contains complex data, the real component of the complex pair is returned.

Command Line Arguments

```
Usage for kprval: Print Data Value to Parser and/or Concatenate to File
% kprval
  -i         (infile)  Input data object

  Required Mutually Exclusive Group - Specify exactly one of the 5 options:
    [-val]  (flag) operate on value data
    [-map]  (flag) operate on map data
    [-mask] (flag) operate on mask data
    [-time] (flag) operate on time data

    [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
      [-loc]    (flag)    operate on location data
      [-dimoff] (integer) offset of point along location dimension
                (dimoff >= 0, default = 0)

  [-mapdata] (boolean) map value data through the map before processing
             (default = false)
  [-woff]    (integer) offset of point along width dimension
             (woff >= 0, default = 0)
  [-hoff]    (integer) offset of point along height dimension
             (hoff >= 0, default = 0)
  [-doff]    (integer) offset of point along depth dimension
             (doff >= 0, default = 0)
  [-toff]    (integer) offset of point along time dimension
             (toff >= 0, default = 0)
  [-eoff]    (integer) offset of point along elements dimension
             (eoff >= 0, default = 0)
  [-var]     (string)  Name of variable to which output value is assigned
             (default = {none})
```

```
[-o]        (outfile) Output ASCII in which to append data value
            (default = {none})
```

C.121. krad2deg — *Output = Input * (180/pi)*

Object Information

Category: *Arithmetic* Subcategory: *Trigonometry*
Operator: *Radian to Degree* Object Type: *pane*
In Cantata: *Yes*

Description

The *Radian to Degree* operator returns the result of multiplying each data point in the input data object, **Input** by 180/pi.

Executing *Radian to Degree* runs the program *karith2*.
karith2 -mul -real 57.29578 -imag 0.0

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith2

Restrictions

Since this pane object calls karith2 -mul, and multiplication is a legal complex operation, this operation will be performed on complex data. The complex scale factor used is (57.29578 + i0.0).

Command Line Arguments

```
Usage for krad2deg: Output = Input * (180/pi)
% krad2deg
```

```
  -i1        (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.122. krayleigh — *Introduce Rayleigh Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Rayleigh Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

krayleigh introduces Rayleigh noise of the specified parameters in the input data object (-i). The noise can either replace the existing value data or it can be added to the existing value data by specifying the appropriate flag (-replace or -add). The variance (-rvar) of the Rayleigh distribution must be specified.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::knoise, DATAMANIP::kexpon, DATAMANIP::kgauss, DATAMANIP::kpoisson, DATAMANIP::kuniform

Command Line Arguments

```
Usage for krayleigh: Introduce Rayleigh Noise in Input Object
% krayleigh
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-replace] (flag) Replace existing values with noise
    [-add]     (flag) Add noise to the specified segment
```

```
[-rvar]     (double)  Variance for Rayleigh noise
            (rvar > 0.0, default = 1)
```

C.123. kreal2cmplx — *Output = Input 1 + j(Input 2)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Real to Complex*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The *Real to Complex* operator (kreal2cmplx) generates a complex Output object (o) containing complex data compiled from the supplied source object(s), Real Input (i1) and Imaginary Input (i2). The data type of the output, which will be complex or double complex, is determined from the highest of the source data types. Only one source object must be specified.

Single Source Object:

If only one source object is specified, the other component of the complex pair will be the Real Constant (real) or Imaginary Constant (imag). If the source object has map data, the operation will be performed on the map only. For the single source case, location, time, and mask data are not needed or modified.

Two Source Objects

If both real and imaginary source objects are specified, and both contain map data and no value data the operation is performed directly on the map, and the output object will contain a map. For this case of two sources that have no value data, location, time, and mask data are not modified, but are transferred to the output.

If either source object contains both map and value data, then both objects must have value data, and the data will be mapped before processing. The output object will have value data and no map data.

If the source objects are the same size, then location and time data can be ignored.

If the source objects are different sizes, then location and time data are checked, and if they exist, the real to complex conversion will only continue if the grid type is uniform. The destination size will be the maximum of both source sizes. If padding is required by the Real Input object, it will be padded with the Real Constant. If padding is required by the Imaginary Input, it will be padded with the Imaginary Constant.

Validity Mask

If either source object has a validity mask, the destination object will get a mask. If both source objects have masks, the masks will be combined (logical AND) after processing.

Failure Modes

Real to complex conversion will fail if neither source object is supplied; if the source objects do not contain at least value or map data; or if either source object is already complex.

Command Line Arguments

```
Usage for kreal2cmplx: Output = Input 1 + j(Input 2)
% kreal2cmplx
  -o         (outfile) Resulting output data object

  Required Loose Group - Specify at least one of the 2 options:
    [-i1] (infile) Input to be used as real component
          (default = {none})
    [-i2] (infile) Input to be used as imaginary component
          (default = {none})

  [-real]    (double)  Value to be used as real component if no i1, or if padding occurs
             (unbounded, default = 0)
  [-imag]    (double)  Value to be used as imaginary component if no i2, or if padding occurs
             (unbounded, default = 0)
```

C.124. krealpart — *Output = Real Component of (Complex) Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Real Part*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Real Part* operator returns the real component of each complex data point in the input data object, **Input**. If the input data is not complex, the imaginary component is assumed to be zero.

The output object data type will be the same as the input (regarding single or double floats).

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Real Part* runs the program *kcmplx2real* with the -or output option.

See Also

DATAMANIP::kcmplx2real, DATAMANIP::kcmplx, DATAMANIP::kreal2cmplx

Command Line Arguments

```
Usage for krealpart: Output = Real Component of (Complex) Input
% krealpart
  -i         (infile)  input data object
  -or        (outfile) resulting output data object
```

C.125. krecip — *Output = 1/Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Reciprocal*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Reciprocal* operator returns the reciprocal of each data point in the input data object, **Input**. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Reciprocal* runs the program *karith1* with the -recip flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for krecip: Output = 1/Input
% krecip
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.126. krect2pol — *Convert (real, imaginary) to (r, theta) Coordinates*

Object Information

Category:	*Arithmetic*	Subcategory:	*Complex Operators*
Operator:	*Rect to Polar*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Rect. to Polar* operator converts data from the rectangular to the polar coordinate system for each complex data point in the input data object, **Input**. The conversion from rectangular to polar will result in angles measured in radians. The data type of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Rect. to Polar* runs the program *kcmplx* with the -r2p flag.

Segments that are not operated on
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

See Also

DATAMANIP::kcmplx

Command Line Arguments

```
Usage for krect2pol: Convert (real, imaginary) to (r, theta) Coordinates
% krect2pol
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.127. kreplace — *Replace All Occurrences of X with Y*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Replace Value*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The **Replace Value** operator, kreplace, replaces each occurrence of the specified old value (X) with a new value (Y). The operations occur on the map segment if one is present in the source object otherwise the operations occur on the value segment.

The optional tolerance values allow the value that is being replaced to fall within a given range (X - Lower tolerance to X + Upper tolerance).

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Command Line Arguments

```
Usage for kreplace: Replace All Occurrences of X with Y
% kreplace
  -i         (infile)  Input file
  -oval      (double)  Value to be replaced (X)
              unbounded
  -nval      (double)  New value to be substituted (Y)
              unbounded
  -o         (outfile) Output file

  [-utol]    (double)  Upper tolerance allowed in searched value
             (unbounded, default = 0)
  [-ltol]    (double)  Lower tolerance allowed in searched value
```

```
(unbounded, default = 0)
```

C.128. kresample — *Resample Object Data*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*Resample*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kresample is used to magnify or reduce the Input data object (i) by using the selected interpolation method (inter). (At this time, only pixel replication is supported.) Magnification factors greater than one magnify the object, and magnification factors between zero and one reduce the object.

Pixel replication and subsampling are done on a floating point indexing basis, where the data point index is obtained by truncating the cumulative floating point index. Therefore, when the calculated replication factor or the subsampling step size is non-integer, the resulting integer step size varies during the process.

Magnification Factors

Magnification factors for each dimension, Width (wmag), Height (hmag) Depth (dmag), Time (tmag), and Elements (emag) can be specified. Magnification factors can be any value greater than zero.

Center of Action

The magnification or reduction is done about a center point that can be specified to be anywhere in the object (even outside of the object). The correct way to think about this is by analogy to a magnifying glass where you place the magnifying glass over the point of interest and then choose the lens to get the magnification you want. The center of action coordinates can be specified individually for each dimension (wc, hc, dc, tc, ec) or, for convenience, the center of action can be automatically computed to be in the middle of all dimensions. This is accomplished by selecting the "center" flag. Automatic centering can be overridden on selected dimensions by explicitly specifying the centers of action for those dimensions.

Output Object Size

The resultant output object size can either be the same as the original input object size if *Resize dimensions* is set to "No" (-resize 0), or it can be resized to fit the magnified or reduced data set if *Resize dimensions* is set to "Yes" (-resize 1).

If the user chooses not to resize the dimensionality of the output object, and the input object is reduced (magnification factor < 1.0) then unknown pixels are set to zero. Likewise, if the input object is magnified (magnification factor > 1.0) then input data that would be mapped outside of the output object are lost.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

If there is map and value data in the source object, and zero order interpolation (pixel replication) is used, the map is transferred to the destination object. If any interpolation algorithm is used, the value data is mapped before the operation can begin and the destination will not have map data.

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

This program fails if there is mask data and the operation uses anything other than zero order interpolation. In other words, if value data is being replicated or subsample on grid, then the mask can be correspondingly replicated or subsampled. (Bilinear interpolation can not be performed on mask data.)

If explicit location or time data are available, the sampling algorithm can not resample these data yet, and will fail. This will be implemented later ....

See Also

kshrink, kexpand

Command Line Arguments

```
Usage for kresample: Resample Object Data
% kresample
  -i         (infile)  Input Object
  -o         (outfile) Resulting output object

  [-wmag]    (double)  Magnification in the width dimension
             (wmag > 0.0, default = 1)
  [-hmag]    (double)  Magnification in the height dimension
             (hmag > 0.0, default = 1)
  [-dmag]    (double)  Magnification in the depth dimension
             (dmag > 0.0, default = 1)
  [-tmag]    (double)  Magnification in the time dimension
             (tmag > 0.0, default = 1)
  [-emag]    (double)  Magnification in the elements dimension
             (emag > 0.0, default = 1)
  [-inter]   (list)    Interpolation Type
          0  "Pixel Replication",
          1  "Bilinear"
          [default: 0 "Pixel Replication"]
  [-resize]  (boolean) resize dimensions of object to the expanded/shrunk size
             (default = true)
  [-center]  (flag)    center
  [-wc]      (integer) Width coordinate for the center of action
             (wc >= 0, default = 0)
  [-hc]      (integer) Height coordinate for the center of action
             (hc >= 0, default = 0)
  [-dc]      (integer) Depth coordinate for the center of action
             (dc >= 0, default = 0)
  [-tc]      (integer) Time coordinate for the center of action
             (tc >= 0, default = 0)
  [-ec]      (integer) Elements coordinate for the center of action
             (ec >= 0, default = 0)
```

C.129. krmseg — *Remove Selected Data Segments from Input*

Object Information

Category:	*Data Manip*	Subcategory:	*Segment Operators*
Operator:	*Remove Segments*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

The program will remove, or delete, the segment(s) from one object and create a new object without those segment(s). If -segment is "Delete All" all the segments and their associated attributes will be deleted. The object level attributes will not be deleted in this, "Delete All", case so an out object will still be created. The -segment parameter can also specify multiple segments by typing a string containing the segment names separated by space(s):

"segment1 segment2 segment3"

Command Line Arguments

```
Usage for krmseg: Remove Selected Data Segments from Input
% krmseg
  -i         (infile)  input file
  -o         (outfile) output file

  Required Loose Group - Specify at least one of the 6 options:
    [-val]     (flag)   If -val flag is selected, operate on Value segment
    [-map]     (flag)   If -map flag is selected, operate on map segment
    [-mask]    (flag)   If -mask flag is selected, operate on Mask segment
    [-loc]     (flag)   If -loc flag is selected, operate on Location segment
    [-time]    (flag)   If -time flag is selected, operate on Time segment
    [-segment] (string) Perform operation on user specified segment
               (default = Delete All)
```

C.130. ksampline — *Sample a data object along an arbitrary line*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*SampleLine*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

ksampline is used to sample the data in an object along an arbitrary line and at arbitrary intervals. The value at each sample point is obtained by an inverse distance weighting scheme.

The first sample point is precisely at the starting coordinate and the final sample is precisely at the ending coordinate.

The line may traverse any portion of the 5D data space. If a particular sample on the line lies outside the data space then the values will be interpolated using a padding value of zero.

The output object will be of the same data type as the input object. If a map is present in the input object, then the value data is sent through the map before processing. The output data will thus have no map.

Mask data, if present, is used to help control the interpolation. Value points with a corresponding mask value of zero will not be included in the interpolated value. However, a minimum number of neighboring values with a valid mask are required in order to produce a valid output value; non-valid output values are marked by a zero mask. If the mask segment is not present, then all value points in the source object are assumed to be valid.

The actual minimum required neighbor count is determined by the setting of the "nc" argument. If nc is zero, then the minimum required neighbor count will be computed from the dimensionality of the source data set. For nc=0, the minimum required neighbor count is 0.5*(2^dim)+1 where dim is the dimensionality of the source object (1 for a line, 2 for a plane, 3 for a volume, etc). If nc is non-zero, then the minimum required neighbor count is simply nc. Beware of non-intuitive results when interpolating across mask wise non-convex parts of the data object. The default minimum required neighbor count gives "reasonable" answers in these cases. Reducing the minimum required neighbor count can provide answers that are less and less meaningful.

Note that ksampline samples the data as if it were using a delta function. This implies that you may get significantly different results depending on the position of each sample location. You may wish to lowpass the source data before sampling with ksampline to avoid aliasing. This can be particularly important when dealing with "holey data" or when the sampling involves degenerate geometry (such as sampling a plane object with a sampling line that does not lie in the plane).

ksampline is implemented using *point* data access. A point data access is done for each sample in the line. This implies that a large number of samples may require quite some time to compute, particularly if operating on a higher dimension data set.

Command Line Arguments

```
Usage for ksampline: Sample a data object along an arbitrary line
% ksampline
  -i         (infile)  input data object
  -o         (outfile) output data object

  [-wc1]     (double)  coordinate of start of line along W axis
             (unbounded, default = 0)
  [-wc2]     (double)  coordinate of end of line along W axis
             (unbounded, default = 511)
  [-hc1]     (double)  coordinate of start of line along H axis
             (unbounded, default = 0)
  [-hc2]     (double)  coordinate of end of line along H axis
             (unbounded, default = 511)
  [-dc1]     (double)  coordinate of start of line along D axis
```

```
              (unbounded, default = 0)
    [-dc2]    (double)  coordinate of end of line along D axis
              (unbounded, default = 0)
    [-tc1]    (double)  coordinate of start of line along T axis
              (unbounded, default = 0)
    [-tc2]    (double)  coordinate of end of line along T axis
              (unbounded, default = 0)
    [-ec1]    (double)  coordinate of start of line along E axis
              (unbounded, default = 0)
    [-ec2]    (double)  coordinate of end of line along E axis
              (unbounded, default = 0)
    [-n]      (integer) number of samples (including endpoints)
              (unbounded, default = 1000)
    [-nc]     (integer) minimum required neighbor count
              (0 <= nc <= 32, default = 0)
```

C.131. ksegcmp — *Compare Attributes & Data of Input 1 and Input 2*

Object Information

Category:	*Data Manip*	Subcategory:	*Analysis & Information*
Operator:	*Compare Objects*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

ksegcmp compares the data and attributes between two input data objects, (i1) and (i2), and prints their differences. The data within the specified data segments is compared point by point, allowing for precise comparisons of data objects independent of file format or machine architecture.

The user can specify either that all of the data segments contained in both objects be compared, or that only specific data segments be compared. The specific polymorphic segments value (val), mask (mask), location (location), time (time), and map (map) can be called out for comparison, along with an user-specified segment name (segment).

The comparison of two segments can only be performed if the segments are of the same size and dimensionality. If the data types of the two segments differ, the user can specify that the data be compared in the higher of the two data types (cast). Additionally, a +/- comparison tolerance can be specified (t). For complex comparisons, this tolerance is applied to both the real and imaginary parts of the values.

If desired, the attributes of each segment can be compared (sat). The segment attributes such as size, dimension, and data type, are not compared by here since they would have already been compared during the data comparison. Also if desired, the object-level attributes, such as the comment, can be compared (oat). The date attribute is ignored in this comparison since it is almost always different.

The program will print the results of the comparison in ASCII either to an user-specified file (o) or, if no file is specified, to kstdout.

The program can optionally print a summary of the differences it finds between each data segment (psum). The program can also optionally print a detailed report on the specific differences found in each segment. At the very minimum, a simple one line report on whether the data objects were determined to be different or identical will be printed. If no output at all is desired, the program can told to run silently (s).

The summary printed for each compared segment reports such things as differences in size or data type, as well as the number of points that were determined to be different.

The precise report of the specific points that were found to be different between two data segments will be printed in the following format :

[POSITION]

< Input #1 Value (Input #1 Data Type)

> Input #2 Value (Input #2 Data Type)

The user may specify which components of this precise report are printed with the use of different flags. The position (pos), the value at that position in the segment from input 1 (i1val), the value at that position in the segment from input 2 (i2val), along with the data types of the values (i1type) and (i2type) can all be specified to be in the detailed report. Note that this program does not allow for the data types of either segment to be printed without their values, because that provides no information.

If desired, the exit return status of the program can be set to indicate whether the data objects were determined to be different or identical (rt). A KEXIT_SUCCESS (0) exit status is used to indicate that the objects were identical, while a KEXIT_FAILURE (1) exit status is used to indicate that they are different.

Also if desired, this routine can return the exit status to a Cantata variable for use in conditional visual programming operations (var).

The comparison is performed "intelligently" using data services, so that that the entire data set from either data object is never read into memory at one time. This allows very large data objects to be compared on machines with limited virtual memory.

Examples

ksegcmp -i1 image:ball -i2 image:moon -val -t 220 -pos -i1val -i2val

will compare the value segment of the ball image with the value segment of the moon image with a comparison tolerance of +/- 220.0. The position and values of the eight points that are different will be printed.

See Also

kcpseg, kcptoval, kcpfromval

Restrictions

ksegcmp is not yet able to return a variable to Cantata. The segment list when all segments are compared is constructed from the first input object. If the second object contains segments which are not contained in the first object, the comparison will miss them.

References

none

Command Line Arguments

```
Usage for ksegcmp: Compare Attributes & Data of Input 1 and Input 2
% ksegcmp
  -i1        (infile)  first input data object
  -i2        (infile)  second input data object

  [-cast]    (flag)    cast data to common type before comparing
  [-tol]     (double)  tolerance for data comparison
             (unbounded, default = 0)
  [-sat]     (boolean) compare segment level attributes
             (default = false)
  [-oat]     (boolean) compare object level attributes
             (default = false)
  [-s]       (flag)    silent running - do not print to stdout
  [-psum]    (boolean) print difference summary
             (default = true)
  [-pos]     (flag)    print the position of differing points
  [-i1val]   (flag)    print value from first input for differing points
  [-i1type]  (flag)    print data type of first input for differing points
  [-i2val]   (flag)    print value from second input for differing points
  [-i2type]  (flag)    print data type of second input for differing points
  [-o]       (outfile) output ASCII file for summary
             (default = {none})
  [-rt]      (boolean) return results as exit status
             (default = true)
  [-var]     (string)  return results to Cantata
             (default = variable)

  [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
    [-all] (flag) compare all data segments

    [Optional Loose Group - Specify at least one of the 6 options:]
      [-val]     (flag)   compare value segments
      [-mask]    (flag)   compare mask segments
      [-loc]     (flag)   compare location segments
      [-time]    (flag)   compare time segments
      [-map]     (flag)   compare map segments
      [-segment] (string) explicit segment name to compare
                 (default = value)
```

C.132. ksetdattr — *Modify Data Object Attributes*

Object Information

Category:	*Data Manip*	Subcategory:	*Object Attributes*
Operator:	*Set Attribute*	Object Type:	*kroutine*

In Cantata: *Yes*

Description

The *Set Data Attribute* operator allows the user to modify a subset of the Input (i) data object attributes, and store the result to the Output (o). The following data attributes can be modified by this program.

Mask Attributes:

Present Masked Data Values As...

This attribute has two possible settings: *Actual Value* and *Substitute Value*. When datamanip processing routines operate on value data that has a corresponding validity mask, processing speed is increased by not checking the mask value for each point to gate the operation based on that value. Rather, all value data, valid or invalid, is processed, and the mask, after any necessary modifications, is transferred to the output. For certain operations, data marked as invalid may cause mathematical or range errors. Setting the *Present Masked Data Values* attribute to *Substitute Value* allows alternate values (real, imaginary pair) to be substituted in place of the invalid data for the operation. Setting the attribute to *Actual Value* will allow the processing routines to operate on the invalid data values.

Substitution Value for Masked Data

If Present Masked Data Values is set to *Substitute Value*, the values used in the substitution, *Real Substitution Value* and *Imaginary Substitution Value* can be explicitly set. If they are not set, the default value is (1,0).

Data Object Uniform Location Data Attributes:

Begin

The Location *Begin* attribute specifies the explicit coordinate of the data set that represents the implicit origin of the data model. When set, explicit uniform location data will be created on the data object. If explicit rectilinear or curvilinear location data already exists on the input, it will be destroyed. Warning - if the begin for any dimension is set using this program, the begin for all dimensions must be set (or the default value of 0 will be applied to unspecified dimensions).

End

The Location *End* attribute specifies the explicit coordinate of the data set that represents the implicit end of the data model. When set, explicit uniform location data will be created on the data object. If explicit rectilinear or curvilinear location data already exists on the input, it will be destroyed. Warning - if the end for any dimension is set using this program, the end for all dimensions must be set (or the default value of 0 will be applied to unspecified dimensions).

Data Object Sub-Object Position

The *Sub-Object Position* attribute is the offset of the current data object in a parent object. Typically, the value of this attribute will be (0,0,0,0,0), but if this object was extracted from a "parent object", via kextract or other means, then this attribute will indicate the position in the parent object from which this region was extracted. Programs like kinsert can use this attribute to reinsert the subobject in the parent object. Warning - if the sub-object position for any dimension is set using this program, the sub-object position for all dimensions must be set (or the default value of 0 will be applied to unspecified dimensions).

Data Object Point Size

The *Point Size* attribute indicates the physical dimension in world coordinates of points in the data object. Unless explicitly set, the data object Point Size is (1,1,1,1,1). Warning - if the point size for any dimension is set using this program, the size for all dimensions must be set (or the default value of 1 will be applied to unspecified dimensions).

Color Attributes:

The *Colorspace* and *Has Alpha* attributes can be modified for color interpretation and visualization. For more information on these attributes and their settings, see the help page for *Change Colorspace* operator (kman icolorspace), and refer to Programming Services, Volume II, Chapter 4, *Color Services*.

Other attributes that require modification of the data as well as the attribute value, such as data type and size, are better set using other programs such as the conversion operators (kconvert, kscale, etc.), and the resize operators (kresample, kextract, etc.).

Command Line Arguments

```
Usage for ksetdattr: Modify Data Object Attributes
% ksetdattr
  -i          (infile)  First Input data object
  -o          (outfile) Resulting output data object

  [-maskpres] (list)    Set the KPDS_MASKED_VALUE_PRESENTATION attribute
          0  "Actual Value",
          1  "Substitute Value"
          [default: 0 "Actual Value"]
  [-wbegin]   (double)  Explicit Location Begin for width dimension
              (unbounded, default = 0)
  [-wend]     (double)  Explicit Location End for width dimension
              (unbounded, default = 1)
  [-hbegin]   (double)  Explicit Location Begin for height dimension
              (unbounded, default = 0)
  [-hend]     (double)  Explicit Location End for height dimension
              (unbounded, default = 1)
  [-dbegin]   (double)  Explicit Location Begin for depth dimension
              (unbounded, default = 0)
  [-dend]     (double)  Explicit Location End for depth dimension
              (unbounded, default = 1)
  [-wpos]     (integer) Sub-object width position with respect to parent object
              (unbounded, default = 0)
  [-wpsize]   (double)  World coordinate point size in width dimension
              (wpsize > 0.0, default = 1)
  [-hpos]     (integer) Sub-object height position with respect to parent object
              (unbounded, default = 0)
  [-hpsize]   (double)  World coordinate point size in height dimension
              (hpsize > 0.0, default = 1)
  [-dpos]     (integer) Sub-object depth position with respect to parent object
              (unbounded, default = 0)
  [-dpsize]   (double)  World coordinate point size in depth dimension
              (dpsize > 0.0, default = 1)
  [-tpos]     (integer) Sub-object time position with respect to parent object
              (unbounded, default = 0)
  [-tpsize]   (double)  World coordinate point size in time dimension
              (tpsize > 0.0, default = 1)
```

```
[-epos]     (integer) Sub-object elements position with respect to parent object
            (unbounded, default = 0)
[-epsize]   (double)  World coordinate point size in elements dimension
            (epsize > 0.0, default = 1)
[-cspace]   (list)    Set the KCOLOR_COLORSPACE attribute
        1  "GREY",
        2  "RGB",
        3  "CMY",
        4  "YIQ ",
        5  "HSV ",
        6  "HLS ",
        7  "IHS ",
        8  "XYZ ",
        9  "UVW ",
        10  "UCSUVW ",
        11  "UCSSOW ",
        12  "UCSLab ",
        13  "UCSLuv ",
        14  "USERDEFINED "
        [default: 3 "CMY"]
[-alpha]    (list)    Set the KCOLOR_HAS_ALPHA attribute
        0  "FALSE",
        1  "TRUE"
        [default: 0 "FALSE"]

[Optional Loose Group - Specify at least one of the 2 options:]
  [-rmasksub] (double) Set the real component of the KPDS_MASK_SUBSTITUTE_VALUE attribute
              (unbounded, default = 0)
  [-imasksub] (double) Set the imaginary component of the KPDS_MASK_SUBSTITUTE_VALUE attribute
              (unbounded, default = 0)
```

C.133. kshot — *Introduce Shot Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Shot Noise*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kshot corrupts a data object uniformly with spikes. The percentage of spikes in the image must be specified by the user. The spike value to be inserted can also be specified. For complex objects both the real and imaginary components need to be specified. If the input object is not complex then the imaginary value specified is ignored. Spike locations are determined using a uniformly distributed random number generator.

Validity Mask

If there is a validity mask associated with the source object, the mask is transferred to the destination object. If the random number generator comes up with a spike location that has corresponding mask value set to zero, the spike is not inserted. However the point is included in the count of the number of points spiked.

Map Segment

If the source object has a map, the destination object will have the mapped source data with the spikes inserted at random locations.

Examples

kshot -i obj1 -o obj2 -spikes 20 -real 255 -imag 255

Inserts a spike in 20% of the points of input complex object obj1. The value of the spike inserted is 255+j255.

Command Line Arguments

```
Usage for kshot: Introduce Shot Noise in Input Object
% kshot
  -i          (infile)  Input File
  -spikes     (double)  Percentage of spikes in the output
               0 <= spikes <= 100
  -o          (outfile) Output File

  [-real]     (double)  Real value of spike
              (unbounded, default = 255)
  [-imag]     (double)  Imaginary value of spike
              (unbounded, default = 0)
```

C.134. kshrink — *Shrink Object Via Pixel Subsampling*

Object Information

Category:	*Data Manip*	Subcategory:	*Size & Region Operators*
Operator:	*Shrink*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Shrink Object* operator (kshrink) reduces the size of the Input data object (i) using pixel subsampling. Reduction factors for each dimension, Width (wmag), Height (hmag) Depth (dmag), Time (tmag), and Elements (emag) can be specified. Reduction factors between zero and one will reduce the object size. (Factor values greater than 1 are also valid, and will expand the object.) The resultant output object size will be the original size multiplied by the reduction factor for that dimension, and rounded.

Pixel subsampling is done on a floating point indexing basis, where the data point index is obtained by truncating the cumulative floating point index. Therefore, when the calculated subsampling step size is non-integer, the resulting integer step size varies during the process.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input` If there is map and value data in the source object, the map is transferred to the destination object.

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

If explicit location or time data are available, the sampling algorithm can not resample these data yet, and will fail. This will be implemented later ....

Executing *Shrink Object* runs the program *kresample* with -resize set to true and -inter set to 0.

See Also

DATAMANIP::kresample, DATAMANIP::kexpand

Command Line Arguments

```
Usage for kshrink: Shrink Object Via Pixel Subsampling
% kshrink
  -i         (infile)  Input object
  -o         (outfile) Resulting subsampled object

  [-wmag]    (double)  Reduction factor in width dimension
             (wmag > 0.0, default = 0.5)
  [-hmag]    (double)  Reduction factor in height dimension
             (hmag > 0.0, default = 0.5)
  [-dmag]    (double)  Reduction factor in depth dimension
             (dmag > 0.0, default = 1)
  [-tmag]    (double)  Reduction factor in time dimension
             (tmag > 0.0, default = 1)
  [-emag]    (double)  Reduction factor in elements dimension
             (emag > 0.0, default = 1)
```

C.135. ksin — *Compute Sine or Arc Sine Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Sin & ArcSin*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Sin & ArcSin* operator returns the sine, or arc sine, value of each data point in the input data object, **Input**. The forward or inverse trigonometric operation is specified by selecting the **Sine** or **Arc Sine** option on the user interface.

Executing *Sin & ArcSin* runs the program *karith1* with the -sin or -asin flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also
DATAMANIP::karith1

Restrictions
Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ksin: Compute Sine or Arc Sine Function
% ksin
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-sin]  (flag) Calculate sine of input
    [-asin] (flag) Calculate arc sine of input
```

C.136. ksinc — *Output = sin(Input)/Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Sinc*	Object Type:	*pane*
In Cantata:	*Yes*		

Description
The *Sinc* operator calculates the sinc function (sin(x)/x) of each data point in the input data object, **Input**.

Executing *Sinc* runs the program *karith1* with the -sinc flag.

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ksinc: Output = sin(Input)/Input
% ksinc
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.137. ksinh — *Compute Hyperbolic Sine or Arc Sine Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Sinh & ArcSinh*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Sinh & ArcSinh* operator returns the hyperbolic sine, or hyperbolic arc sine, value of each data point in the input data object, **Input**. The forward or inverse trigonometric operation is specified by selecting the **Hyperbolic Sine** or **Hyperbolic Arc Sine** option on the user interface.

Executing *Sinh & ArcSinh* runs the program *karith1* with the -sinh or -asinh flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ksinh: Compute Hyperbolic Sine or Arc Sine Function
% ksinh
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-sinh]  (flag) Calculate hyperbolic sine of input
    [-asinh] (flag) Calculate hyperbolic arc sine of input
```

C.138. ksqr — *Output = Input**2*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Square*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Square* operator returns the square (value raised to the power of 2) of each data point in the input data object, **Input**. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Square* runs the program *karith2* with the -pow flag, -real 2.0 and -imag 0.0.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith2

Command Line Arguments

```
Usage for ksqr: Output = Input**2
% ksqr
  -i1        (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.139. ksqrt — *Output = Square Root of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Square Root*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Square Root* operator returns the square root (value raised to the power of 1/2) of each data point in the input data object, **Input**. The data type of of the output data object, **Output**, will be the same as the data type of **Input**.

Executing *Square Root* runs the program *karith1* with the -sqrt flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ksqrt: Output = Square Root of Input
% ksqrt
  -i          (infile)  Input data object
  -o          (outfile) Resulting output data object
```

C.140. kstats — *Compute Statistics of Data Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Analysis & Information*
Operator:	*Statistics*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kstats computes the mean, variance, standard deviation, RMS level, skew, kurtosis, minimum value, maximum value, and integral (positive, negative, and total sums) of the Input data object (i). Information such as the number of unmasked or ungated points (positive, negative, zero-valued, and total counts), the coordinates of the minimum and maximum values, and the dimensions of the data object are also provided.

If there is a validity mask associated with the source object, the calculation is performed using only data that have corresponding mask values not equal to zero. The mask data is not transferred to the output object.

If the source object contains map data as well as value data, then the value data will be mapped through the map (the value data type becomes that of the map) before the operation is performed. The destination object contains value data with no associated map.

If the data has a validity mask as well as a map, the mapped data is also masked. When mapping a value datum that has a mask value of zero (invalid), the mapping process will first look for an entry in the map that corresponds to the "invalid" value, if one does not exist (for example, the value is NaN), the mapped data is set to the pad value. In either case, the data remains masked. The mask is resized to correspond to the size of the mapped value data to properly mask it. For example, the value data may have the dimensions of

width = 100,
height = 50,
elements = depth = time = 1.

If the map dimensions are

map width = 3,
map height = whatever,
map element = map time = map spatial = 1,

the resulting value (and mask) size will be

width = 100,
height = 50,
elements = 3,
depth = time = 1.

All statistics and information will reflect the mapping. For example, if the dimensions of the data are requested (wsize, hsize, etc), kstats returns the dimensionality after mapping.
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`
All statistics are calculated as double, and the Binary Output data type is double.

Location and time data do not affect the statistics calculations, and are not transferred to the output object.

A flag can be set for each statistic and information option so that a subset of all available options can be calculated and stored or printed. The flags are mutually exclusive with the *Calculate All Statistics* flag (all). (The command line flag names are: mean, var, sd, rms, skew, kur, minval, maxval, wmin, hmin, dmin, tmin, emin, wmax, hmax, dmax, tmax, emax, sum, psum, nsum, pts, ppts, npts, zpts, wsize, hsize, dsize, tsize, and esize.) If no flag is specified when running kstats from the command line, all statistics and information will be stored.

Statistics are computed according to the equations given below. In the equations, N is the number of samples, SUM is the sum from i=0 to i=N-1, and x(i) is the sample value of x at i.

Mean (-mean)

mean = (1/N) * SUM(x(i)) i=0..N-1

Variance (-var)

variance = (1/(N-1)) * (SUM (x(i) - mean)**2) i=0..N-1

Standard Deviation (-sd)

standard deviation = sqrt(variance)

RMS (-rms)

RMS = sqrt(1/N * SUM(x(i)**2)) i=0..N-1

Skewness (-skew)

Skewness is a measure of the tendency of the deviations from the mean to be larger in one direction than in the other. It is a measure of the asymmetry of a distribution about its mean. A positive skew value signifies a distribution whose tail extends out towards more positive x, and a negative tail signifies a distribution whose tail extends out towards more negative x. Population skewness is unitless and is defined as:

E[((x-mean)/(stddev))**3],
where stddev is the standard deviation of the data. *kstats* computes the sample skewness as:

skew = 1/N * SUM(((x(i) - mean)/stddev) **3) i=0..N-1

If the variance is equal to zero, skewness will be set to 0.0.

Kurtosis

Kurtosis is a unitless measure of the tail heaviness of a distribution and is defined as:

E[((x-mean)/(stddev))^4] - 3,
where stddev is the standard deviation of the data. *kstats* computes the sample kurtosis as:

kurtosis= (1/N * SUM(((x(i) - mean)/stddev) **4)) - 3 i=0..N-1

If the variance is equal to zero, kurtosis will be set to 0.0.

Sums or Integrals (-sum, -psum, -nsum)

total integral = SUM x(i)
positive part of integral = SUM x(i), for all x(i) > 0
negative part of integral = SUM x(i), for all x(i) < 0

Define Processing Unit

To support analysis of subregions within the data object, such as lines, planes, volumes, and vectors, an option for defining processing units, or regions, is provided. These regions are defined by the settings of the Processing Unit options, which can be either the Whole Data Set (whole), or any combination of Width (w), Height (h), Depth (d), Time (t), and Elements (e). Statistics for each region will be computed and printed separately. If none of these flags are supplied, then a single set of statistics will be computed for the entire data object.

Gating Input

If a gating source object is supplied, the operation is only performed where the gating value is non-zero.

ASCII Output

The *ASCII Output* (f) option allows the user to specify a device or file for printing the specified information in formatted ASCII. A filename of # will send the output to stderr. If neither an *ASCII Output* or a *Binary Output* is supplied, the formatted ASCII will automatically go to stdout.

Binary Output

If the *Binary Output* (-o) is selected, the selected statistics and information are stored in as double float data in the given file.

Each statistic is stored as an element of a N-D vector defined along the "elements" dimension of the output object. N is the number of statistics and information specified by the user. The order in which the information is stored in the statistics vector is given in the output object's comment attribute.

When M sets of statistics are calculated for M multiple regions of the input data object (see discussion on independent region analysis above), the statistics vectors are incremented along the width dimension of the output statistics object - the output object would have a resulting dimensionality of width=M, elements=N (height = depth = time = 1). For example, if the input data object had the dimensionality width = 256, height = 256, elements = 7 (depth = time = 1), and the user defined the analysis regions to be width-height, the output statistics object would have dimensionality width = 7, elements = N (height = depth = time = 1). Where the vector at w=0 would contain information pertaining to the first region (starting at w=0, h=0, d=0, t=0, e=0), the vector at w=1 would contain information about the second region (starting at w=0, h=0, d=0, t=0, e=1), and so forth.

Examples

kstats -i object.xv -f ascii -all

Will create an ASCII statitics file. Since the -region option is not specified, kstats defaults to calculating statistical information for the object as a whole.

kstats -i object.xv -f ascii_file -region -e

Will create an ASCII statistics file. The statistical information is now calculated by slicing the object along the element dimension. For example, on a 3-band RGB color image, this would compute the statistics for each RGB vector.

Restrictions

kstats cannot compute complex statistics at this time. If you wish to process the real and imaginary components of a complex data set independently, first separate components using kcmplx2real.

In the current implementation, the statistics are accumulated. This can result in overflow. kstats will be rewritten in the future to better handle large values and large data sets.

Command Line Arguments

```
Usage for kstats: Compute Statistics of Data Object
% kstats
  -i         (infile)  Input object

  [-igate]   (infile)  Gating input data object
             (default = {none})
  [-f]       (outfile) Formatted ASCII output file
             (default = {none})
  [-o]       (outfile) Binary output statistics file
             (default = {none})
```

```
[Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
  [-whole] (flag) compute single set of statistics for entire data set

  [Optional Loose Group - Specify at least one of the 5 options:]
    [-w] (flag) include width in processing unit
    [-h] (flag) include height in processing unit
    [-d] (flag) include depth in processing unit
    [-t] (flag) include time in processing unit
    [-e] (flag) include elements in processing unit

[Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
  [-all] (flag) compute and store all statistics information

  [Optional Loose Group - Specify at least one of the 30 options:]
    [-mean]   (flag) compute mean
    [-sum]    (flag) compute total integral
    [-var]    (flag) compute variance
    [-wmin]   (flag) store width coordinate of the minimum value
    [-psum]   (flag) compute positive part of integral
    [-sd]     (flag) compute standard deviation
    [-hmin]   (flag) store height coordinate of the minimum value
    [-nsum]   (flag) compute negative part of integral
    [-rms]    (flag) compute root mean square
    [-dmin]   (flag) store depth coordinate of the minimum value
    [-pts]    (flag) compute total number of contributing points
    [-skew]   (flag) compute skewness
    [-tmin]   (flag) store time coordinate of the minimum value
    [-ppts]   (flag) compute number of positive contributing points
    [-kur]    (flag) compute kurtosis
    [-emin]   (flag) store elements coordinate of the minimum value
    [-npts]   (flag) compute number of negative contributing points
    [-minval] (flag) compute minimum value
    [-zpts]   (flag) compute number of zero-valued contributing pts
    [-maxval] (flag) compute maximum value
    [-wmax]   (flag) store width coordinate of the maximum value
    [-wsize]  (flag) store size of objects width dimension
    [-hmax]   (flag) store height coordinate of the maximum value
    [-hsize]  (flag) store size of objects height dimension
    [-dmax]   (flag) store depth coordinate of the maximum value
    [-dsize]  (flag) store size of objects depth dimension
    [-tmax]   (flag) store time coordinate of the maximum value
    [-tsize]  (flag) store size of objects time dimension
    [-emax]   (flag) store elements coordinate of the maximum value
    [-esize]  (flag) store size of objects elements dimension
```

C.141. ksub — *Output = Input 1 - (Input 2 or Constant)*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Subtract*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Subtract* operator subtracts each point in either **Input 2** or the **Constant** value, which ever is specified by the user, from **Input 1**.

Executing *Subtract* runs the program *karith2* with the -sub flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2, DATAMANIP::ksubfrom

Command Line Arguments

```
Usage for ksub: Output = Input 1 - (Input 2 or Constant)
% ksub
  -i1        (infile)  First input data object
  -o         (outfile) Resulting output data object
```

```
Required Mutually Exclusive Group - Specify exactly one of the 2 options:
  [-i2] (infile) Second input data object
        (default = {none})

  [Optional Loose Group - Specify at least one of the 2 options:]
    [-real] (double) Real constant value
            (unbounded, default = 0)
    [-imag] (double) Imaginary constant value
            (unbounded, default = 0)
```

C.142. ksubfrom — *Output = (Input 2 or Constant) - Input 1*

Object Information

Category:	*Arithmetic*	Subcategory:	*Two Operand Arithmetic*
Operator:	*Subtract From*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Subtract From* operator subtracts each point in **Input 1** from either **Input 2** or the **Constant** value(s), which ever is specified by the user.

Executing *Subtract* runs the program *karith2* with the -sub flag.

Data Type - Single Input
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Data Type - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/value_type_2input`

Map Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Map Data - Two Input Objects
include file `$DATAMANIP/repos/shared/man/sections/map_2input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_2input`

Input Objects of Different Sizes
include file `$DATAMANIP/repos/shared/man/sections/resize_2input`

The values used to pad the data when input files are not the same size are (0.0, 0.0).

Explicit Location and Time Data - Single Input
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Explicit Location and Time Data - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_2input`

Failure Modes - Single Input

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Failure Modes - Two Input Objects

include file `$DATAMANIP/repos/shared/man/sections/fail_2input`

See Also

DATAMANIP::karith2, DATAMANIP::ksub

Command Line Arguments

```
Usage for ksubfrom: Output = (Input 2 or Constant) - Input 1
% ksubfrom
  -i1        (infile)  First Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-i2] (infile) Second input data object
          (default = {none})

    [Optional Loose Group - Specify at least one of the 2 options:]
      [-real] (double) Real constant value
              (unbounded, default = 0)
      [-imag] (double) Imaginary constant value
              (unbounded, default = 0)
```

C.143. ktan — *Compute Tangent or Arc Tangent Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Tan & ArcTan*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Tan & ArcTan* operator returns the tangent, or arc tangent, value of each data point in the input data object, **Input**. The forward or inverse trigonometric operation is specified by selecting the **Tangent** or **Arc Tangent** option on the user interface.

Executing *Tan & ArcTan* runs the program *karith1* with the -tan or -atan flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ktan: Compute Tangent or Arc Tangent Function
% ktan
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-tan]  (flag) Calculate tangent of input
    [-atan] (flag) Calculate arc tangent of input
```

C.144. ktanh — *Compute Hyperbolic Tangent or Arc Tangent Function*

Object Information

Category:	*Arithmetic*	Subcategory:	*Trigonometry*
Operator:	*Tanh & ArcTanh*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Tanh & ArcTanh* operator returns the hyperbolic tangent, or hyperbolic arc tangent, value of each data point in the input data object, **Input**. The forward or inverse trigonometric operation is specified by selecting the **Hyperbolic Tangent** or **Hyperbolic Arc Tangent** option on the user interface.

Executing *Tanh & ArcTanh* runs the program *karith1* with the -tanh or -atanh flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ktanh: Compute Hyperbolic Tangent or Arc Tangent Function
% ktanh
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-tanh]  (flag) Calculate Hyperbolic Tangent of input
    [-atanh] (flag) Calculate Hyperbolic Arc Tangent of input
```

C.145. kthreshabove — *Threshold Data Values that are Above Specified Cutoff*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Thresh Above*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Threshold Above* operator generates a two-level output by thresholding the *Input* data object (i). Data values greater than the *Cutoff Value* (uc) are set to the "GREATER THAN" output value (fval), and values less than or equal to the *Cutoff Value* are set to the "LESS THAN or EQUAL" Output value (tval).

The following histogram defines the results of the thresholding operation.

```
|<----T---->|<-F->|
|      | :      .   |
|    .:.:.  .:::  |
|:. .::::::.:::::  |
|::::::::::::::::.|
------------|------
             Cutoff

F:  if value >  Cutoff,  Output = fval
T:  if value <= Cutoff,  Output = tval
```

Data Type
include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Threshold Above* runs the program *kclip* with the -thresh option set to TRUE (-thresh 1).

See Also

DATAMANIP::kthreshbelow, DATAMANIP::kthreshin, DATAMANIP::kthreshout, DATAMANIP::kmegathresh, DATAMANIP::kclip

Command Line Arguments

```
Usage for kthreshabove: Threshold Data Values that are Above Specified Cutoff
% kthreshabove
  -i         (infile)  Input file
  -o         (outfile) output file

  [-uc]      (double)  For values > uc, output = fval, otherwise output = tval
             (unbounded, default = 128)
  [-fval]    (double)  Output value if data is GREATER THAN cutoff (uc)
             (unbounded, default = 1)
  [-tval]    (double)  Output value if data is LESS THAN or EQUAL to cutoff (uc)
             (unbounded, default = 0)
```

C.146. kthreshbelow — *Threshold Data Values that are Below Specified Cutoff*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Thresh Below*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Threshold Below* operator generates a two-level output by thresholding the *Input* data object (i). Data values less than the *Cutoff Value* (lc) are set to the "LESS THAN" output value (fval), and values greater than or equal to the *Cutoff Value* are set to the "GREATER THAN or EQUAL" Output value (tval).

The following histogram defines the results of the thresholding operation.

```
                    |<-F->|<----T---->|
                    |     | :     .   |
                    |    .:.:.  .:::  |
                    |:. .::::::.:::::  |
                    |::::::::::::::::.|
                    ------|------------
                        Cutoff

F:  if value <= Cutoff,  Output = fval
T:  if value >  Cutoff,  Output = tval
```

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Threshold Above* runs the program *kclip* with the -thresh option set to TRUE (-thresh 1).

See Also

DATAMANIP::kthreshabove, DATAMANIP::kthreshin, DATAMANIP::kthreshout, DATAMANIP::kmegathresh, DATAMANIP::kclip

Command Line Arguments

```
Usage for kthreshbelow: Threshold Data Values that are Below Specified Cutoff
% kthreshbelow
  -i         (infile)  Input file
  -o         (outfile) output file

  [-lc]      (double)  For values < lc, output = fval, otherwise output = tval
             (unbounded, default = 128)
  [-fval]    (double)  Output value if data is LESS THAN cutoff (lc)
             (unbounded, default = 1)
  [-tval]    (double)  Output value if data is GREATER THAN or EQUAL to cutoff (lc)
             (unbounded, default = 0)
```

C.147. kthreshin — *Threshold Data Values Inside the Specified Range*

Object Information

Category: *Data Manip* Subcategory: *Threshold & Clip Operators*
Operator: *Thresh Inside* Object Type: *pane*
In Cantata: *Yes*

Description

The *Threshold Inside* operator generates a two-level output by thresholding the *Input* data object (i) inside the range defined by the specified Lower and Upper Cutoff values (uc, lc). Data values that are greater than the *Lower Cutoff Value* (uc), and less than the *Upper Cutoff Value* (lc) are are set to the first output value (fval). Data values that are less than or equal to The Lower Cutoff, or are greater than or equal to the upper cutoff are set to the second output value (tval).

The following histogram defines the results of the thresholding operation.

```
      |<-T->|<-F->|<-T->|
      |     | :   | .   |
      |    .:.:.  .:::  |
      |:. .:::::::.:::: |
      |:::::::::::::::::.|
      ------|-----|------
Lower Cutoff     Upper Cutoff
         (uc)      (lc)

if value <  Upper Cutoff AND value >  Lower Cutoff   output = fval
if value >= Upper Cutoff OR  value <= Lower Cutoff   output = tval
```

P "Data Type" 5 *include file* `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Threshold Inside* runs the program *kclip* with the -thresh option set to TRUE (-thresh 1).

NOTE: When running kthreshin from the command line, the cutoff parameters, uc and lc, are opposite to what would seem logical (uc = lower cutoff and lc = upper cutoff). This is due to the logic in the parent program, kclip.

See Also

DATAMANIP::kthreshabove, DATAMANIP::kthreshbelow, DATAMANIP::kthreshout, DATAMANIP::kmegathresh, DATAMANIP::kclip

Command Line Arguments

```
Usage for kthreshin: Threshold Data Values Inside the Specified Range
% kthreshin
  -i         (infile)  Input file
  -o         (outfile) output file

  [-uc]      (double)  Lower cutoff value (this is a reverse logic operation)
             (unbounded, default = 64)
  [-lc]      (double)  Upper cutoff value (this is a reverse logic operation)
             (unbounded, default = 128)
  [-fval]    (double)  Output value if data is GREATER THAN uc, AND LESS THAN lc
             (unbounded, default = 0)
  [-tval]    (double)  Output value if data is LESS THAN/EQUAL to uc, or GREATER THAN/EQUAL to lc
             (unbounded, default = 1)
```

C.148. kthreshout — *Threshold Data Values Outside of the Specified Range*

Object Information

Category:	*Data Manip*	Subcategory:	*Threshold & Clip Operators*
Operator:	*Thresh Outside*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Threshold Outside* operator generates a two-level output by thresholding the *Input* data object (i) outside the specified Lower and Upper Cutoff values (lc, uc). Data values that are either less than the *Lower Cutoff Value* (lc), or greater than the *Upper Cutoff Value* (uc) are considered outside the threshold range, and are set to the first output value (fval). Data values that are between, or equal to the cutoff values are within the threshold range, and are set to the second output value (tval).

The following histogram defines the results of the thresholding operation.

```
|<-F->|<-T->|<-F->|
|     | :   | .   |
|    .:.:.  .:::  |
|:. .:::::::.:::: |
|::::::::::::::::.|
------|-----|------
     lc     uc
```

```
F:  if value <  lc OR  value >  uc  output = fval
T:  if value >= lc AND value <= uc  output = tval
```

P "Data Type" 5 *include file* `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

Executing *Threshold Outside* runs the program *kclip* with the -thresh option set to TRUE (-thresh 1).

See Also

DATAMANIP::kthreshabove, DATAMANIP::kthreshbelow, DATAMANIP::kthreshin, DATAMANIP::kmegathresh, DATAMANIP::kclip

Command Line Arguments

```
Usage for kthreshout: Threshold Data Values Outside of the Specified Range
% kthreshout
  -i         (infile)  Input file
  -o         (outfile) output file

  [-lc]      (double)  Lower cutoff value
             (unbounded, default = 64)
  [-uc]      (double)  Upper cutoff value
             (unbounded, default = 128)
  [-fval]    (double)  Output value if data is LESS THAN lc, or GREATER THAN uc
```

```
               (unbounded, default = 0)
      [-tval]    (double)  Output value if data is BETWEEN, or EQUAL TO the Cutoff Values
               (unbounded, default = 1)
```

C.149. ktranslate — *Translate Data in Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Reorganize Data*
Operator:	*Translate*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Translate* operator, ktranslate, performs the translation of a data object using the pad values defined by the Real and Imaginary padding value options. The (0,0,0,0,0) point of the input data will be translated to the position specified by the translation offset values. Values that are translated outside of the original data space are truncated. Wrapping is currently not supported, but will be provided in the future.

Size

The output data object will be the same as the input data size.

Pad Values

The Real Pad Value (real) specifies the pad value that will be assigned to the real component of a complex pair, and the Imaginary Pad Value (imag) specifies the imaginary component of the pair. If the input data object type is not complex, the imaginary pad value will be ignored. For input data objects that are not of type double, the specified pad values are cast to the same type as the input data object.

Mask Data & Validity Option

The validity mask corresponding to padded data will be zero if the user specifies that padded data added by this program be marked as invalid (valid FALSE). Likewise, if the user specifies that padded data is valid (valid TRUE), the mask corresponding to padded data will be non-zero. In the first case (invalid), if no mask exists in the source object, on will be created. In the valid case, no mask will be created. Validity mask data corresponding to the original source object data is transferred to the destination unaltered.

Location & Time Data

If curvilinear or rectilinear time or location data exist, it is unclear how to alter the dimensionality, so the operation will fail. If uniform location data exist translation occurs, but the location information is not altered in any way.

Map Data

The translate operator does not modify map data.

Failure Modes

The input object must contain value or mask data.

Executing *Translate* runs the program *kpad* with offsets specified.

See Also

DATAMANIP::kpad

Restrictions

The translate operator will fail if the source object contains curvilinear or rectilinear location or time data. If uniform time or location data exist, it is transferred, but not modified.

Wrapping is not supported at this time.

Command Line Arguments

```
Usage for ktranslate: Translate Data in Object
% ktranslate
  -i         (infile)  Input data file
  -o         (outfile) Resulting output file

  [-woff]    (integer) Translate data in width dimension by this amount
             (unbounded, default = 0)
  [-hoff]    (integer) Translate data in height dimension by this amount
             (unbounded, default = 0)
  [-doff]    (integer) Translate data in depth dimension by this amount
             (unbounded, default = 0)
  [-toff]    (integer) Translate data in time dimension by this amount
             (unbounded, default = 0)
  [-eoff]    (integer) Translate data in elements dimension by this amount
             (unbounded, default = 0)
  [-real]    (double)  Real part of pad constant
             (unbounded, default = 0)
  [-imag]    (double)  Imaginary part of pad constant
             (unbounded, default = 0)
  [-valid]   (boolean) Identify data added by program as valid/invalid in mask
             (default = true)
```

C.150. ktruncate — *Output = Integer Truncate of Input*

Object Information

Category:	*Arithmetic*	Subcategory:	*Single Operand Arithmetic*
Operator:	*Truncate*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

The *Truncate* operator returns the integer portion of each data point input data object, **Input**.

Executing *Truncate* runs the program *karith1* with the -trunc flag.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data

include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask

include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data

include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes

include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::karith1

Restrictions

Operations on complex data are not supported at this time.

Command Line Arguments

```
Usage for ktruncate: Output = Integer Truncate of Input
% ktruncate
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object
```

C.151. kuniform — *Introduce Uniform Noise in Input Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Introduce Noise*
Operator:	*Uniform Noise*	Object Type:	*pane*
In Cantata:	*Yes*		

Description

kuniform introduces uniform noise of the specified parameters in the input data object (-i). The noise can either replace the existing value data or it can be added to the existing value data by specifying the appropriate flag (-replace or -add). The minimum (-umin) and the maximum (-umax) of the uniform distribution must be specified.

Data Type

include file `$DATAMANIP/repos/shared/man/sections/value_type_1input`

Map Data
include file `$DATAMANIP/repos/shared/man/sections/map_1input`

Validity Mask
include file `$DATAMANIP/repos/shared/man/sections/mask_1input`

Explicit Location and Time Data
include file `$DATAMANIP/repos/shared/man/sections/loc_and_time_1input`

Failure Modes
include file `$DATAMANIP/repos/shared/man/sections/fail_1input`

See Also

DATAMANIP::knoise, DATAMANIP::kexpon, DATAMANIP::kgauss, DATAMANIP::kpoisson, DATAMANIP::krayleigh

Command Line Arguments

```
Usage for kuniform: Introduce Uniform Noise in Input Object
% kuniform
  -i         (infile)  input file
  -o         (outfile) output file

  Required Mutually Exclusive Group - Specify exactly one of the 2 options:
    [-replace] (flag) Replace existing values with noise
    [-add]     (flag) Add noise to the specified segment

  [-umin]     (double)  Minimum for uniform noise
              (unbounded, default = 0)
  [-umax]     (double)  Maximum for uniform noise
              (umax > 0.0, default = 1)
```

C.152. kunmask — *Remove Mask from Data Object*

Object Information

Category:	*Data Manip*	Subcategory:	*Mask Operators*
Operator:	*Unmask Data*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

Unmask Data (kunmask) will remove the validity mask from the Input data object (i) in the manner specified by the user, and store the unmasked results in the Output data object (o). The object's mask may also be stored to an output data object, Output Mask (omask), as either Value (val) or Mask (mask) data (if neither option is selected, the data will be stored in the mask). This option may be useful for restoring an object's mask if it must be removed prior to an operation that is unable to interpret or operate on objects containing validity masks, or it may be used for operation gating purposes. Masks can be re-inserted into the object using the *Copy Segments* (kcpseg) or *Copy from Value* (kcpfromval) operators. There are currently 3 unmasking options:

Propagate

The *Propagate Original Data* option (-prop) simply removes the mask from the data object. Masked, or invalid, data is propagated directly from the input to the output.

Replace

The validity mask is used to mark points of data which are not valid for processing -- sometimes the data at these points is genuinely *not* processable, such as NaN or infinity. The *Replace Masked Data with Values* option (-replace) allows the invalid data to be replaced with the Real and Imaginary values (real, imag) specified by the user.

Use Attributes

KPDS_MASKING_MODE and KPDS_MASK_SUBSTITUTE_VALUE are mask attributes that can be explicitly set and stored with the data object. These attributes allow the user to provide some information about the data under the validity mask, and processing routines can act accordingly. The *Process as Defined by Object Mask Attributes* option (-attr) queries these attributes to determine the unmasking process and the replacement values. If these attributes are not explicitly set in the input data object, the default unmasking operation is to propagate the input data to the output.

If none of the processing flags are selected, processing defaults to *Use Attributes* (attr).

If there is map, location, or time data associated with the source object, they are transferred to the destination object.

Command Line Arguments

```
Usage for kunmask: Remove Mask from Data Object
% kunmask
  -i         (infile)  Input data object
  -o         (outfile) Resulting output data object

  [-real]    (double)  If replace selected, replace real data with this value
             (unbounded, default = 0)
  [-imag]    (double)  If replace selected, replace imaginary data with this value
             (unbounded, default = 0)

  [Optional Mutually Exclusive Group - Specify one or none of the 3 options:]
    [-prop]    (flag) Propagate masked (invalid) data to output
    [-replace] (flag) Replace masked (invalid) data with real/imag values
    [-attr]    (flag) Process as defined by the object mask attributes

  [Optional Mutually Inclusive Group - Specify all or none of the 2 options:]
    [-omask] (outfile) Output file containing mask that was removed
             (default = {none})

    [Optional Mutually Exclusive Group - Specify one or none of the 2 options:]
      [-mask] (flag) If selected, save input mask as mask data in omask object
      [-val]  (flag) If selected, save input mask as value data in omask object
```

C.153. kwiener — *Wiener Filtering (Restoration) in Fourier Frequency Domain*

Object Information

Category:	*Data Manip*	Subcategory:	*Frequency Filters*
Operator:	*Wiener*	Object Type:	*kroutine*
In Cantata:	*Yes*		

Description

kwiener performs restoration of data in the spatial domain by Wiener filtering in the frequency domain. The idea is to deconvolve the blurring function from the blurred data. A few problems exist in performing this operation, as described in great detail in any good image processing book.

A full implementation of a Wiener filter requires detailed knowledge of the signal and noise power spectra. Many times this information is simply unavailable or is very poorly known. In such situations, it is convenient to assign some constant value to the SNR and use that in the Wiener filter implentation. This is a somewhat extreme restriction of the Wiener filter, but it makes the filter very easy to use and can produce better results than the pseudoinverse filter in *kinverse*. (Better in the sense of much less ringing).

Note that the input data objects will be treated simply as a field of complex data, presumably obtained by application of an FFT, although the data could be generated by many other means. The output data will be either KCOMPLEX if the input data is any type other than KDCOMPLEX. Otherwise, the output will be of type KDCOMPLEX. Data objects with maps attached are not supported. Mask data is ignored, if present.

See Also

kinverse(1), kfft(1), kcmplx2real(1)

References

K.R. Castleman, "Digital Image Processing", Prentice-Hall, 1979, ISBN 0-13-212365-7, p281 Eq. 2.
Rosenfeld & Kak, p287.

Command Line Arguments

```
Usage for kwiener: Wiener Filtering (Restoration) in Fourier Frequency Domain
% kwiener
  -i1        (infile)  source data object (FFT)
  -i2        (infile)  PSF data object (FFT)
  -o         (outfile) filtered data object (FFT)

  [-w]       (double)  Wiener parameter
             (w >= 0.0, default = 1)
```

Table of Contents

This page left intentionally blank